

# **Intro to SQL Querying**



# 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 19c
- Discuss the theoretical and physical aspects of a relational database



# Lesson Objectives

- Describe Oracle server's implementation of relational database management system (RDBMS) and object relational database management system (ORDBMS)
- Identify the development environments that can be used for this course
- Describe the database and schema used in this course



# Lesson Agenda

- Course objectives, roadmap, and appendixes used in the course
- Overview of Oracle Database 19c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in the course
- Introduction to SQL and its development environments
- Oracle Database 19c SQL Documentation and Additional Resources

# 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

# Course Objectives

- 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 Roadmap

Lesson 1: Introduction

**Unit 1: Retrieving, Restricting,  
and Sorting Data**

Unit 2: Joins, Subqueries, and  
Set Operators

Unit 3: DML and DDL

► Lesson 2: Retrieving Data using SQL SELECT

► Lesson 3: Restricting and Sorting Data

► Lesson 4: Using Single-Row Functions to  
Customize Output

► Lesson 5: Using Conversion Functions and  
Conditional Expressions

# Course Roadmap

Lesson 1: Introduction

Unit 1: Retrieving, Restricting,  
and Sorting Data

**Unit 2: Joins, Subqueries, and  
Set Operators**

Unit 3: DML and DDL

▶ Lesson 6: Reporting Aggregated Data Using  
Group Functions

▶ Lesson 7: Displaying Data from Multiple  
Tables Using Joins

▶ Lesson 8: Using Subqueries to Solve Queries

▶ Lesson 9: Using Set Operators

# Course Roadmap

Lesson 1: Introduction

Unit 1: Retrieving, Restricting,  
and Sorting Data

Unit 2: Joins, Subqueries, and  
Set Operators

**Unit 3: DML and DDL**

▶ Lesson 10: Managing Tables Using DML  
Statements

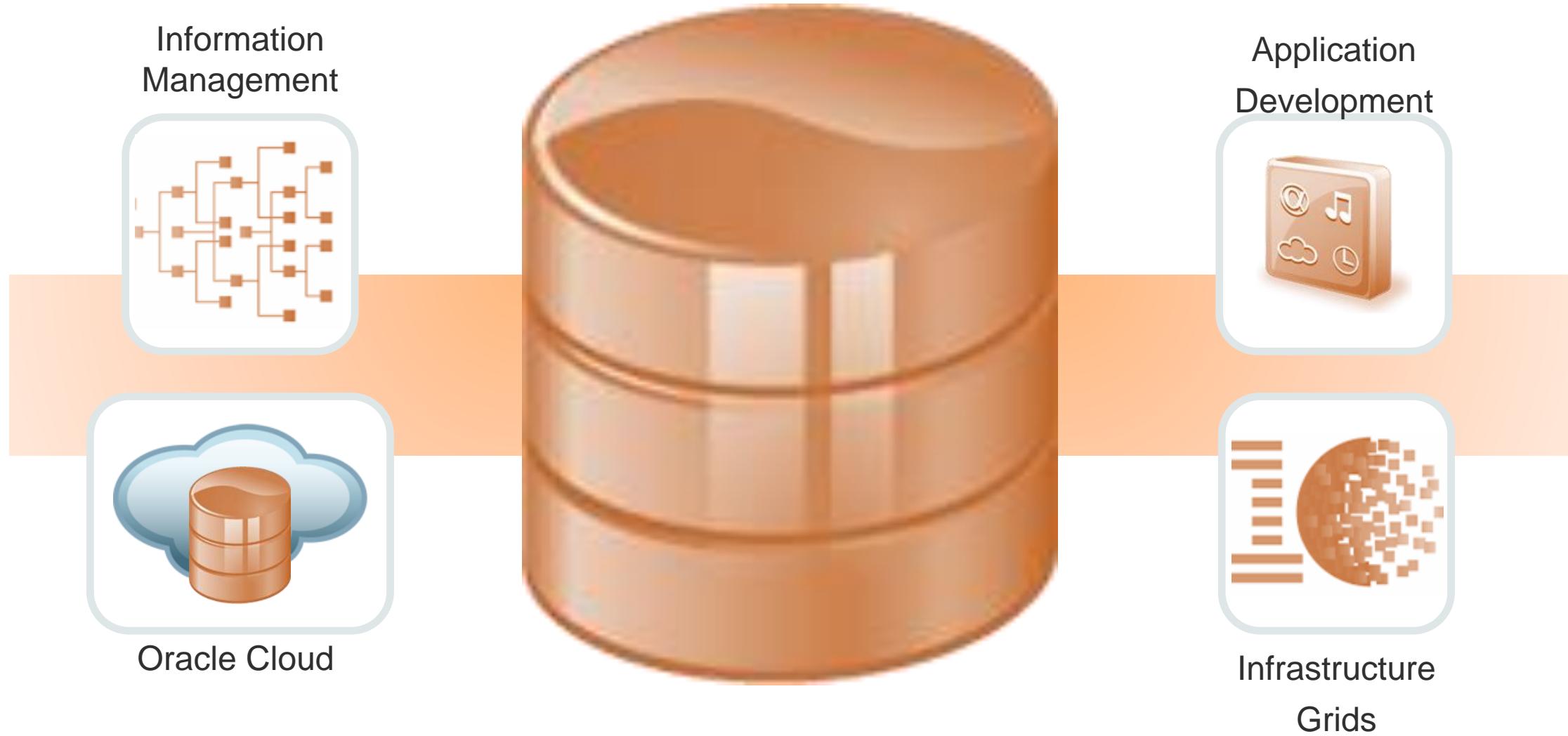
▶ Lesson 11: Introduction to Data Definition  
Language



# Lesson Agenda

- Course objectives, roadmap, and appendixes used in the course
- Overview of Oracle Database 19c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in the course
- Introduction to SQL and its development environments
- Oracle Database 19c SQL Documentation and Additional Resources

# Oracle Database 19c: Focus Areas



# Oracle Database 19c



High Availability



Manageability



Performance



Security



Information  
Integration



# Lesson Agenda

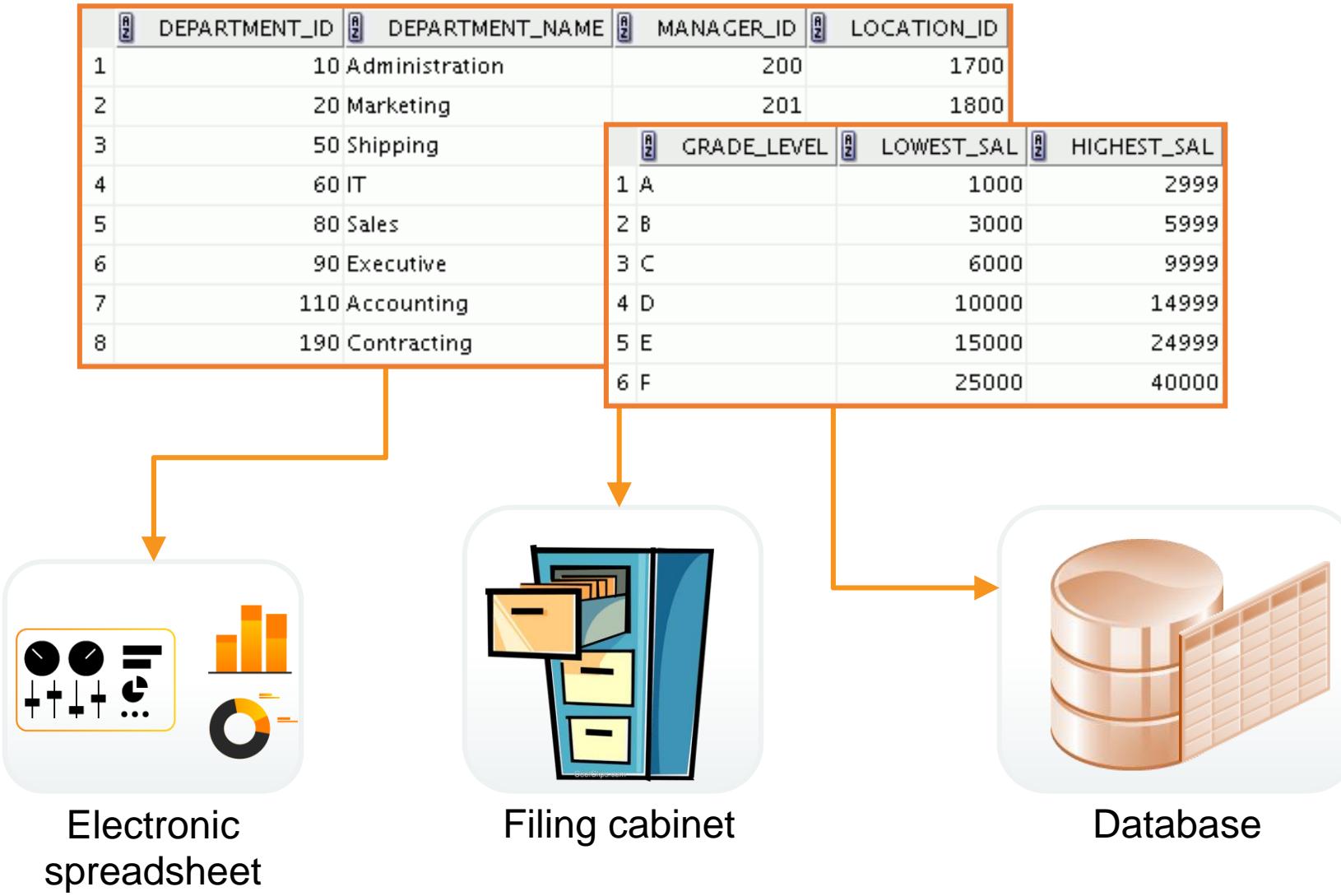
- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 19c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in this course
- Introduction to SQL and its development environments
- Oracle Database 19c SQL Documentation and Additional Resources

# 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



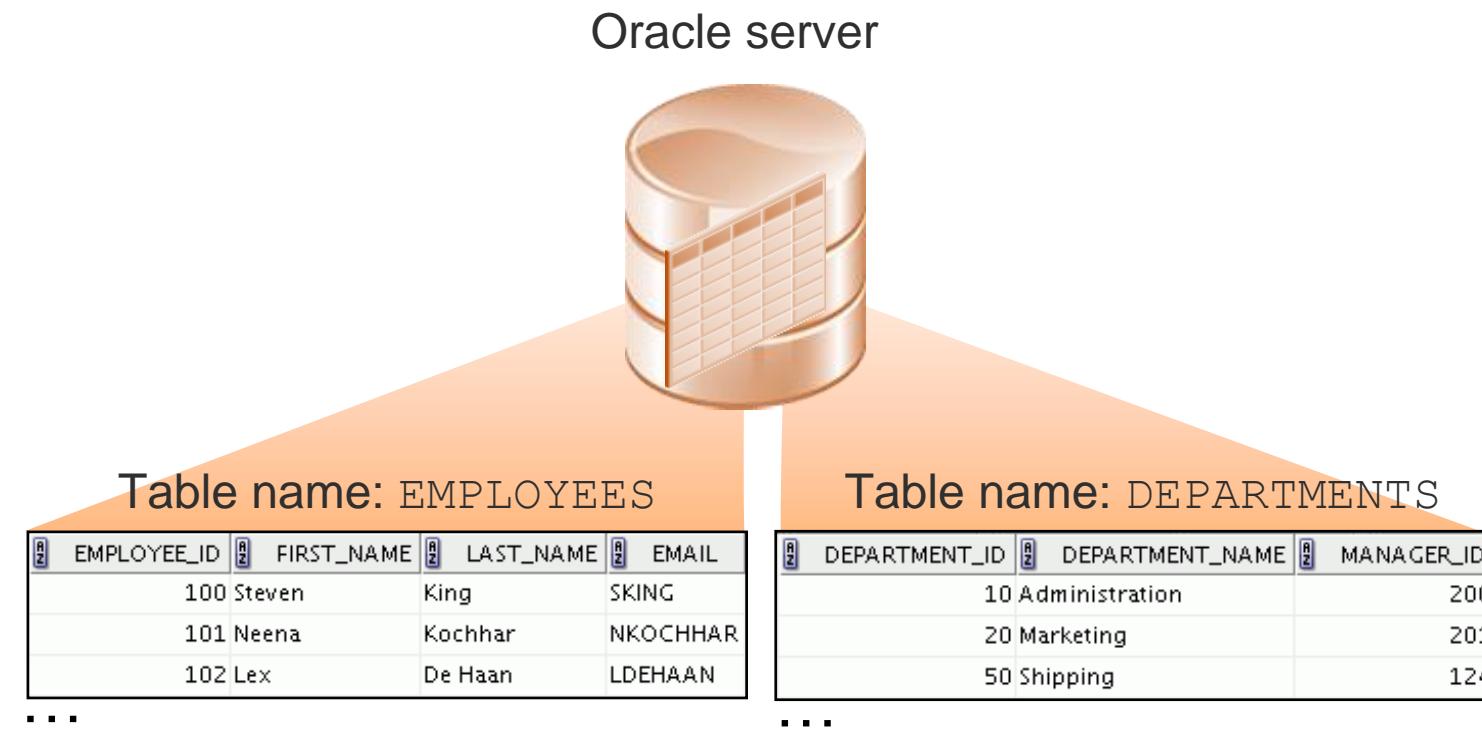
# Relational Database Concept

- Dr. E. F. Codd proposed the relational model for database systems in 1970.
- It is the basis for 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

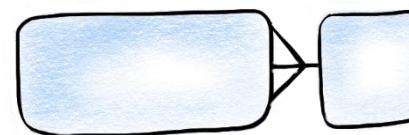


# Relational Database

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



# Data Models



Entity model of  
client's model

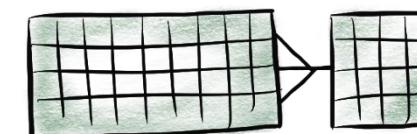


Table model  
of entity model

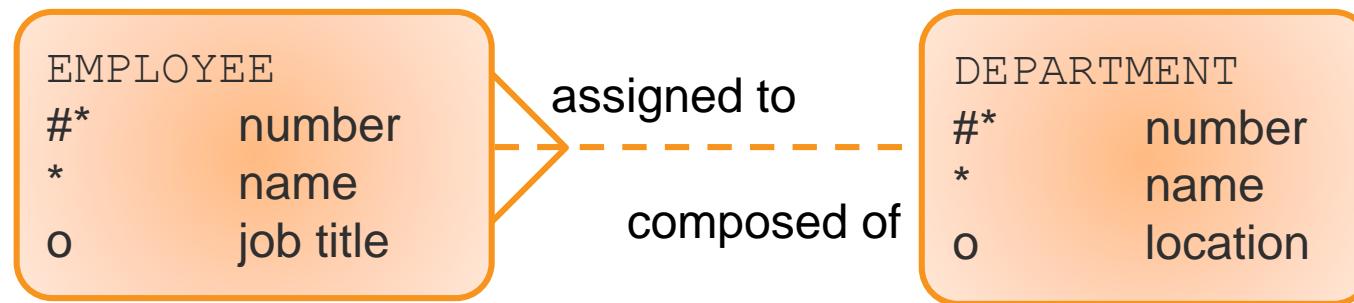


Oracle  
server

Tables  
on disk

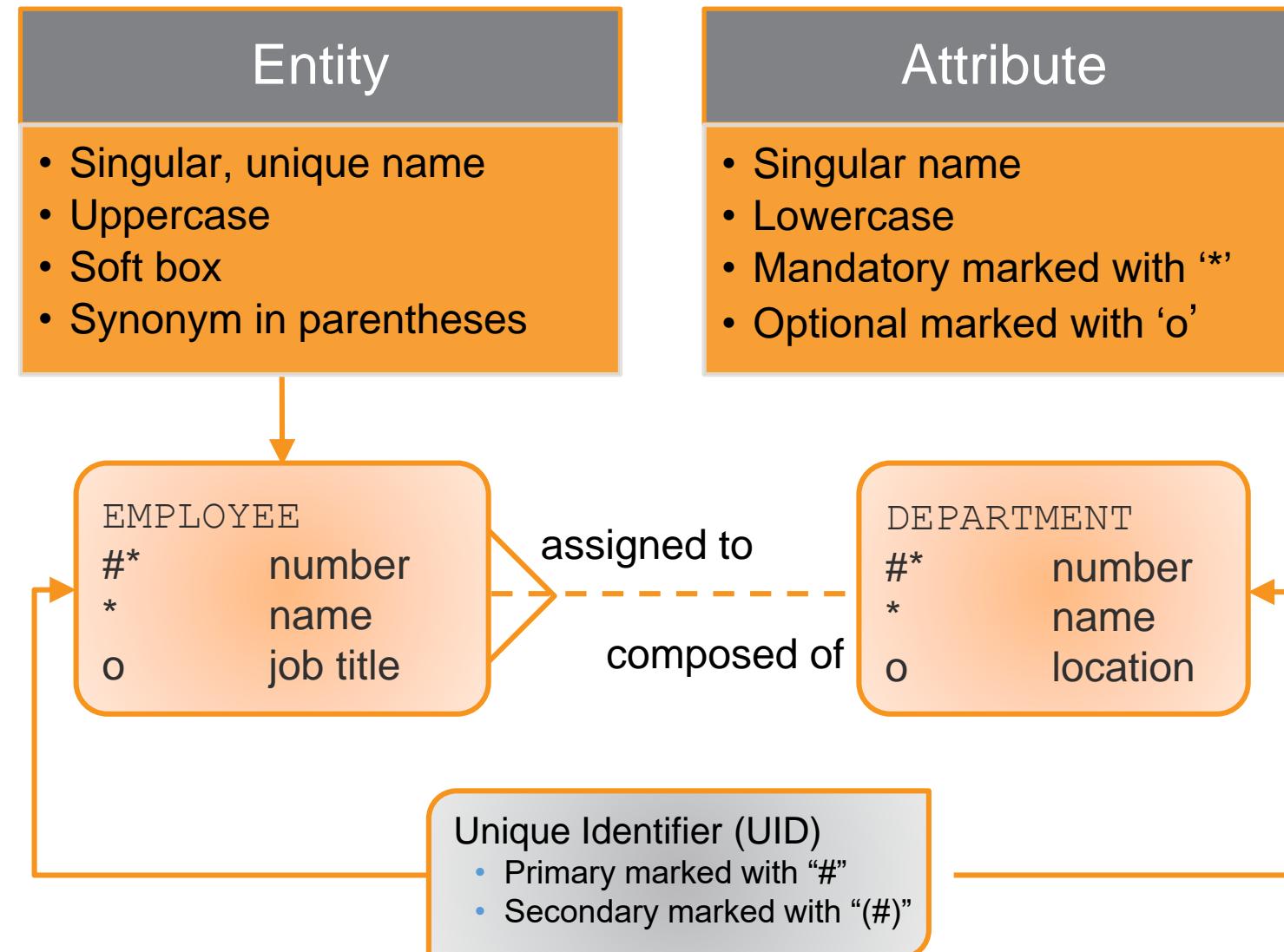
# Entity Relationship Model

- 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...”

# Relationship Modelling Conventions



# 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

The diagram illustrates a relational database table with various annotations:

- 1**: A large orange rectangle surrounds the entire table structure.
- 2**: An orange rectangle highlights the first four columns: EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, and SALARY.
- 3**: An orange circle highlights the fifth column: COMMISSION\_PCT.
- 4**: An orange rectangle highlights the last two columns: COMMISSION\_PCT and DEPARTMENT\_ID.
- 5**: An orange circle highlights the value "50" in the DEPARTMENT\_ID column for employee 143.
- 6**: An orange circle highlights the value "0.2" in the COMMISSION\_PCT column for employee 149.

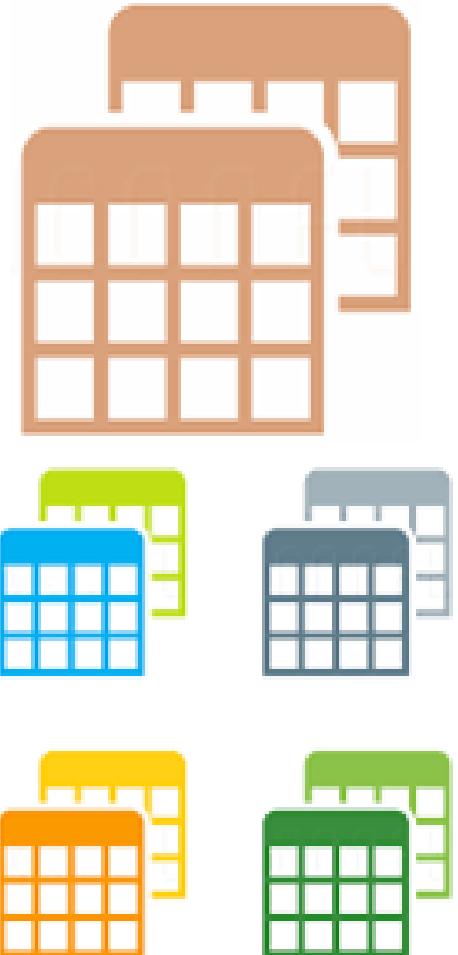
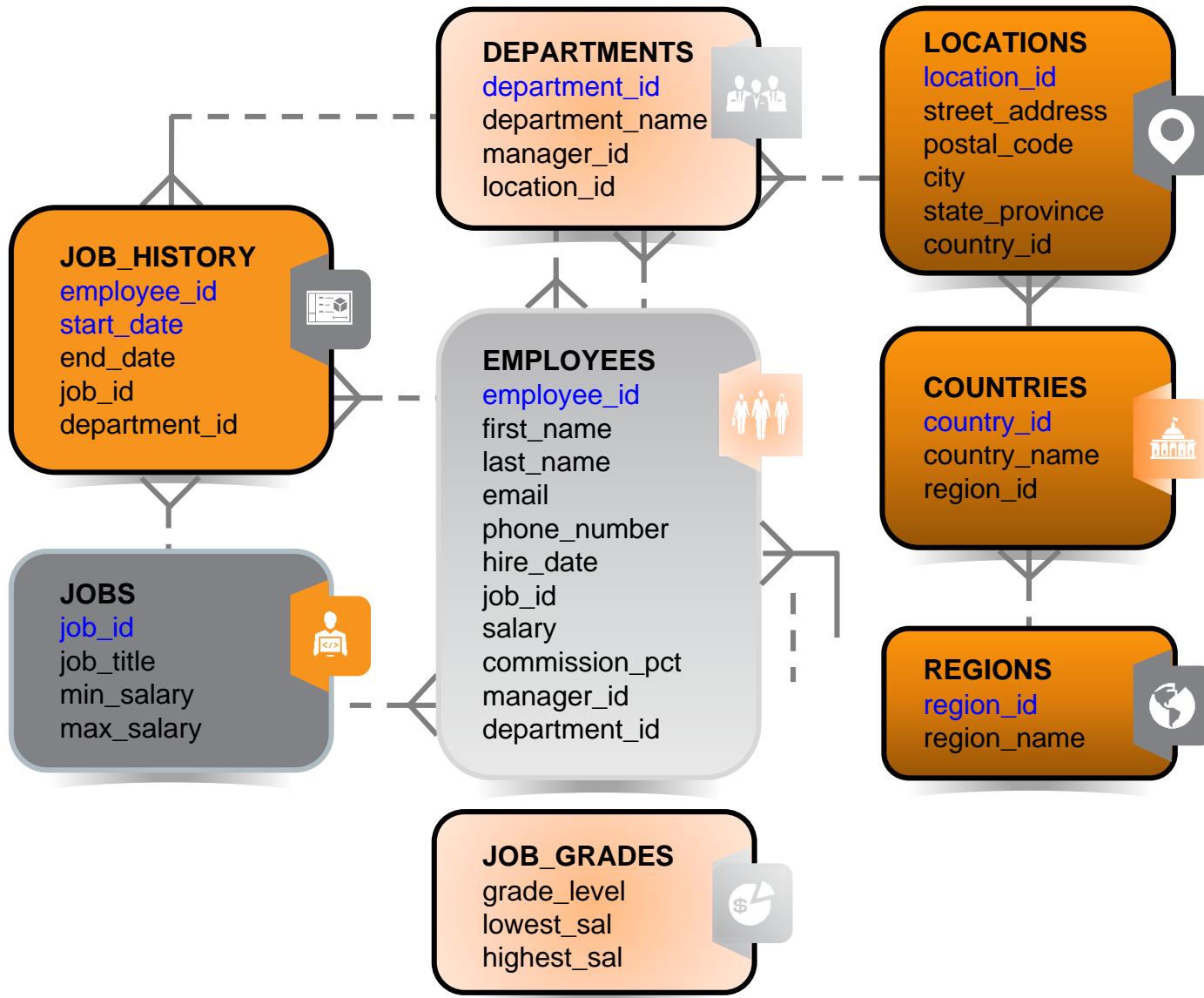
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	90
101	Neena	Kochhar	17000	(null)	90
102	Lex	De Haan	17000	(null)	90
103	Alexander	Hunold	9000	(null)	60
104	Bruce	Ernst	6000	(null)	60
107	Diana	Lorentz	4200	(null)	60
124	Kevin	Mourgos	5800	(null)	50
141	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	8800	(null)	110



# Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 19c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in this course
- Introduction to SQL and its development environments
- Oracle Database 19c SQL Documentation and Additional Resources

# Tables Used in This Course



# Tables Used in the Course

EMPLOYEES

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

JOBs

#	JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1	AD_PRES	President	20000	40000
2	AD_VP	Administration Vice President	15000	30000
3	AD_ASST	Administration Assistant	3000	6000
4	FI_MGR	Finance Manager	8200	16000
5	FI_ACCOUNT	Accountant	4200	9000
6	AC_MGR	Accounting Manager	8200	16000
7	AC_ACCOUNT	Public Accountant	4200	9000

DEPARTMENTS

#	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	114	1700
4	40	Human Resources	203	2400
5	50	Shipping	121	1500
6	60	IT	103	1400
7	70	Public Relations	204	2700
8	80	Sales	145	2500

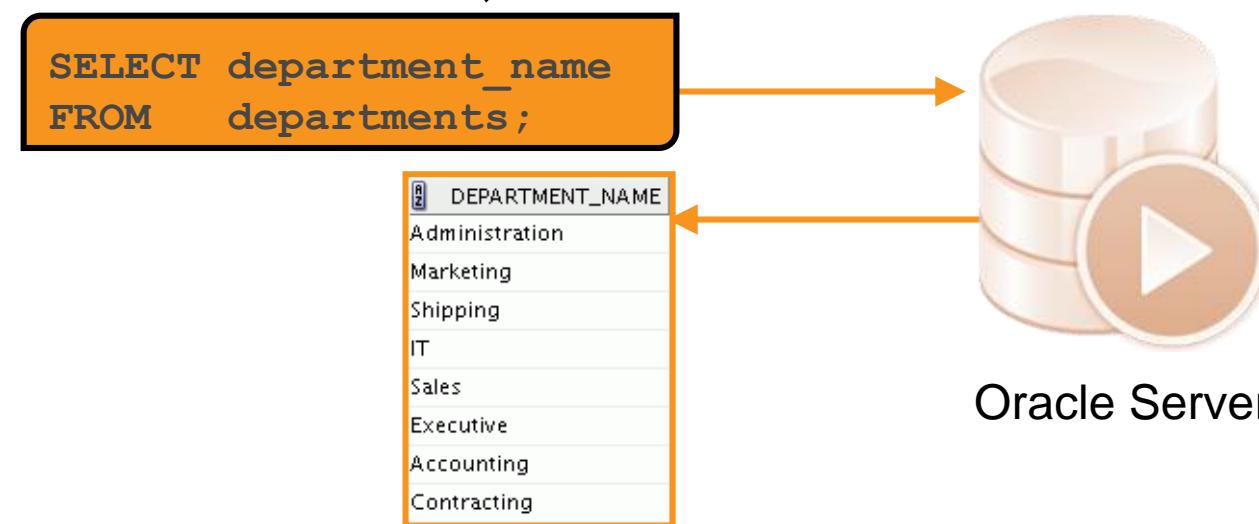
# Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 19c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in this course
- Introduction to SQL and its development environments
- Oracle Database 19c SQL Documentation and Additional Resources

# SQL to Query Your Database

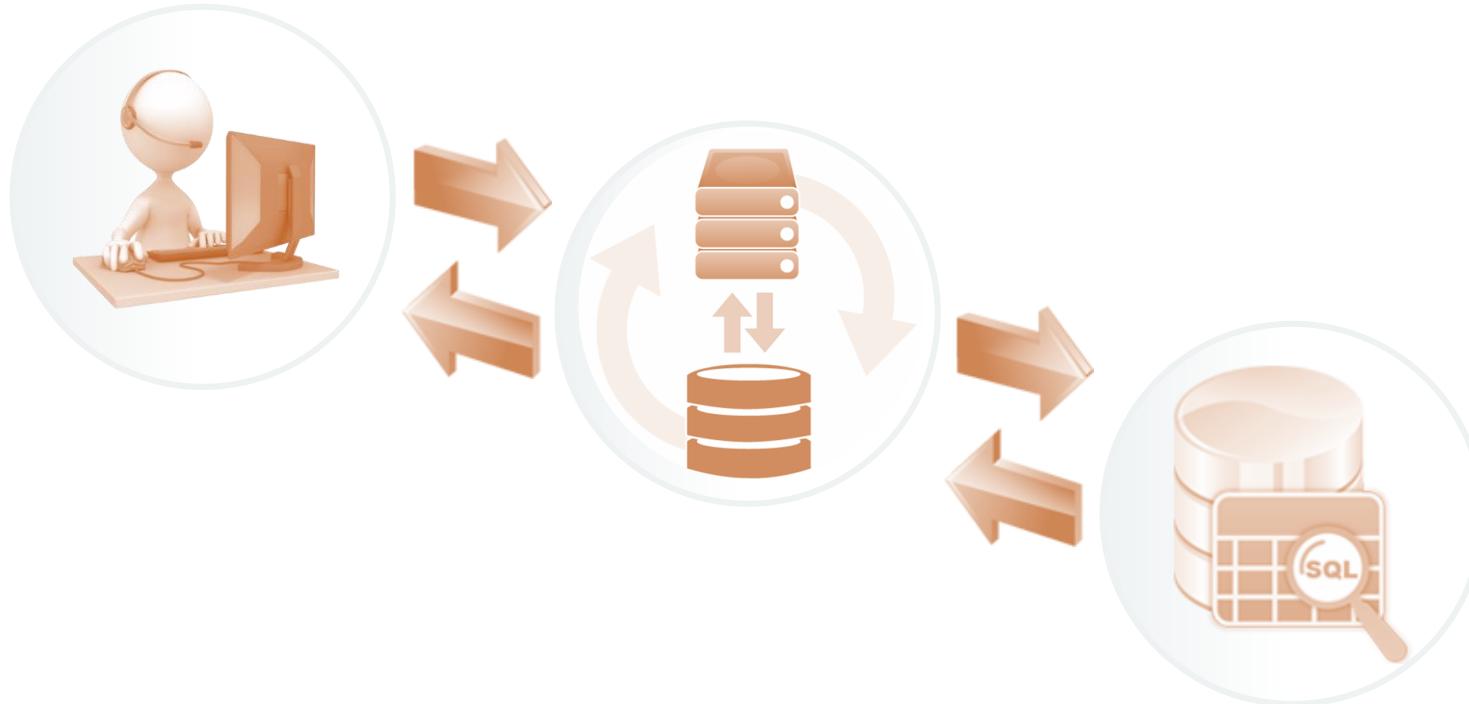
Structured query language (SQL) is:

- 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.)



# How SQL Works

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



# SQL Statements

Data manipulation language  
(DML)

**SELECT  
INSERT  
UPDATE  
DELETE  
MERGE**

Data definition language (DDL)

**CREATE  
ALTER  
DROP  
RENAME  
TRUNCATE  
COMMENT**

Data control language (DCL)

**GRANT  
REVOKE**

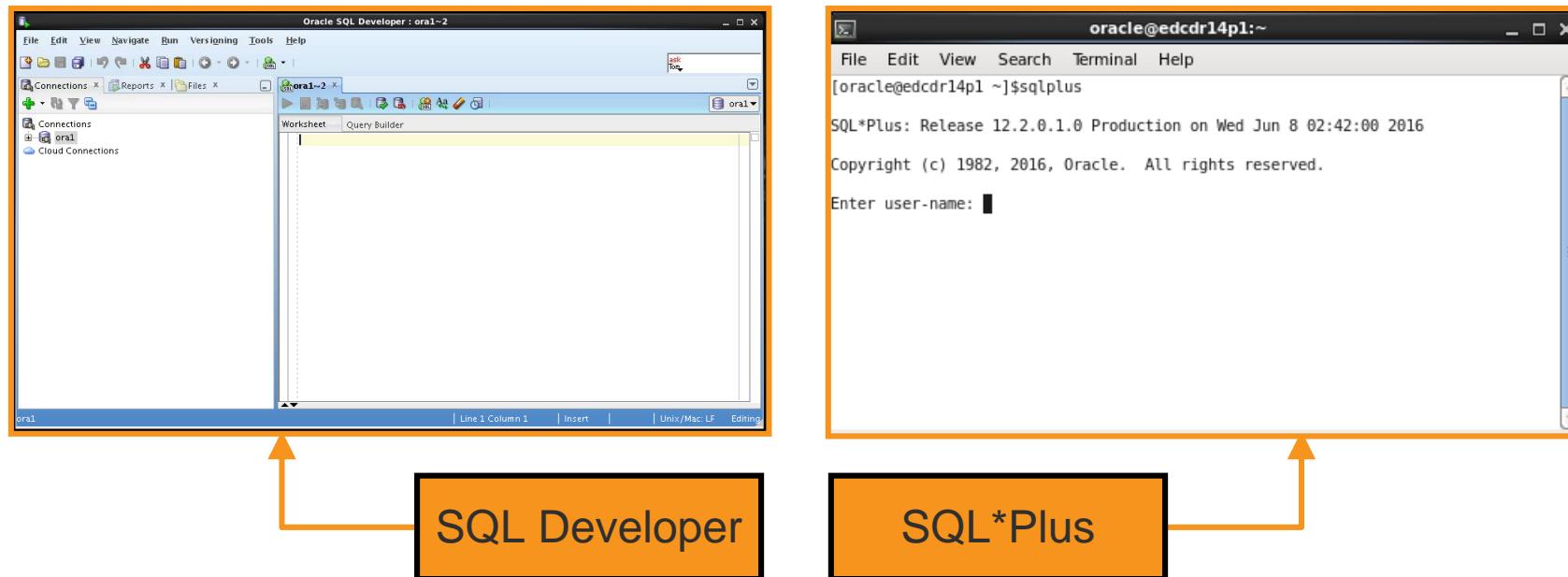
Transaction control

**COMMIT  
ROLLBACK  
SAVEPOINT**

# Development Environments

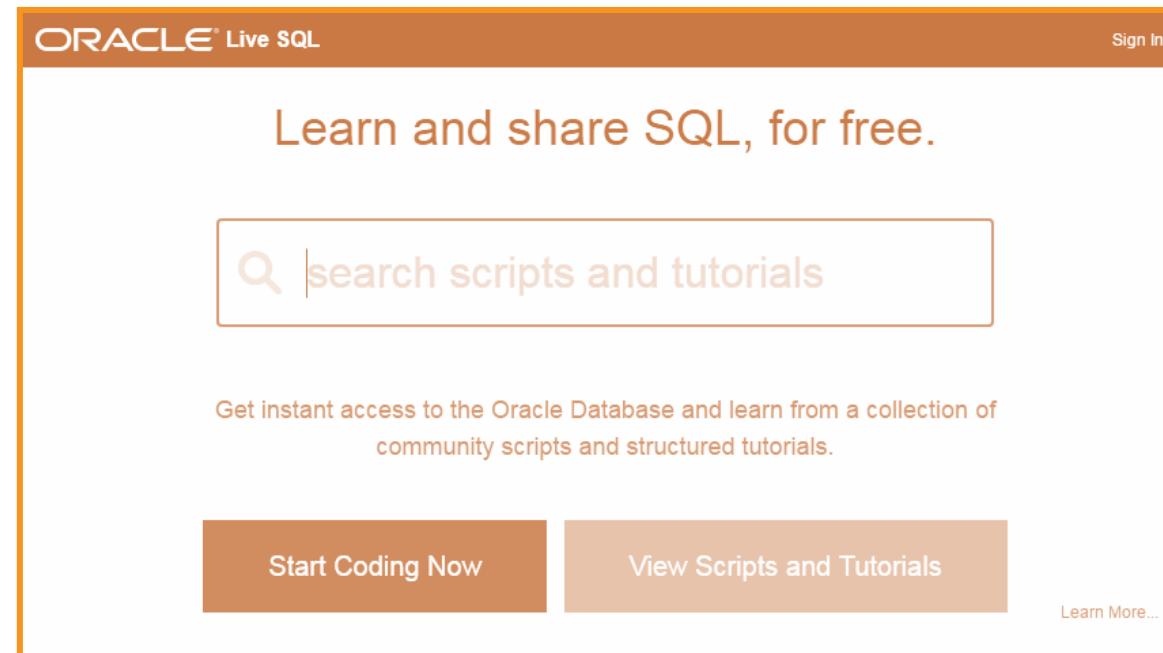
There are two development environments for this course:

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



# Introduction to Oracle Live SQL

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





# Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 19c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in this course
- Introduction to SQL and its development environments
- Oracle Database 19c SQL Documentation and Additional Resources

# Oracle Database Documentation

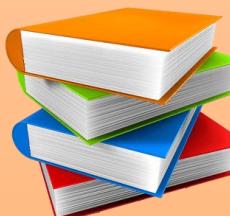
- *Oracle Database New Features Guide*
- *Oracle Database Reference*
- *Oracle Database SQL Language Reference*
- *Oracle Database Concepts*
- *Oracle Database SQL Developer User's Guide*



# Additional Resources

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

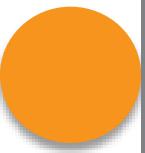
- *Oracle Database 19c: New Features eStudies*
- *Oracle Learning Library:*
  - <http://www.oracle.com/goto/oll>
- *Oracle Cloud:*
  - [cloud.oracle.com](http://cloud.oracle.com)
- The online SQL Developer Home Page, which is available at:
  - [http://www.oracle.com/technology/products/database/sql\\_developer/index.html](http://www.oracle.com/technology/products/database/sql_developer/index.html)
- The SQL Developer tutorial, which is available online at:
  - <http://download.oracle.com/oll/tutorials/SQLDeveloper/index.htm>



# Summary

In this lesson, you should have learned about:

- The goals of the course
- Features of Oracle Database 19c
- The theoretical and physical aspects of a relational database
- Oracle server's implementation of RDBMS and ORDBMS
- The development environments that can be used for this course
- The database and schema used in this course



# Practice 1: Overview

This practice covers the following topics:

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

# Select Statement

Retrieving Data Using  
the SQL SELECT Statement

# Course Roadmap

Lesson 1: Introduction

**Unit 1: Retrieving, Restricting  
and Sorting Data**

Unit 2: Joins, Subqueries, and  
Set Operators

Unit 3: DML and DDL

▶ Lesson 2: Retrieving Data using SQL SELECT

▶ Lesson 3: Restricting and Sorting Data

▶ Lesson 4: Using Single-Row Functions to  
Customize Output

▶ Lesson 5: Using Conversion Functions and  
Conditional Expressions

You are here!

# Objectives

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

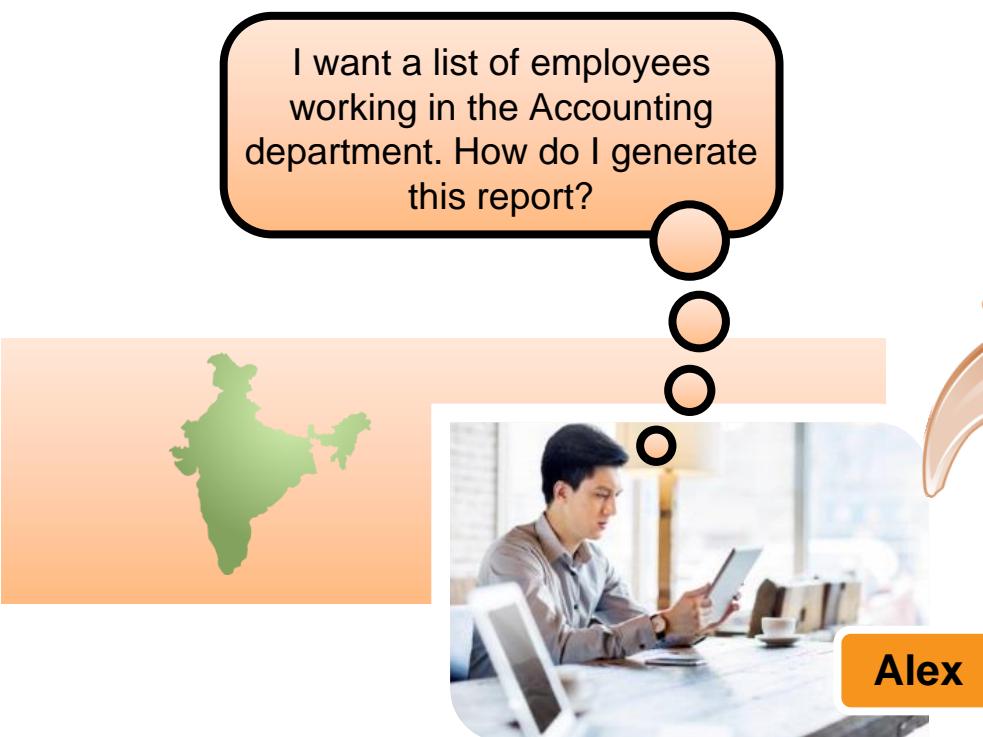
- List the capabilities of SQL SELECT statements
- Execute a basic SELECT statement



# Lesson Agenda

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

# HR Application Scenario



**HR Application**

Employee Search:

Advanced Search:

First Name  Location

Last Name  Department

**GO**

**Result Set**

**HR Application**

Emp_ID	First Name	Last Name	Department
205	Sheldon	Cooper	Accounting
109	Racheal	Higgins	Accounting
123	Parvathy	Patil	Accounting

...

# Basic SELECT Statement

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

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

Selecting from a table



# 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	30	Purchasing	114	1700
4	40	Human Resources	203	2400
5	50	Shipping	121	1500
6	60	IT	103	1400
7	70	Public Relations	204	2700
8	80	Sales	145	2500
9	90	Executive	100	1700

# Selecting Specific Columns

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

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	30	1700
4	40	2400
5	50	1500
6	60	1400
7	70	2700
8	80	2500
9	90	1700

# Selecting from DUAL

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

```
SELECT *
FROM    dual;
```

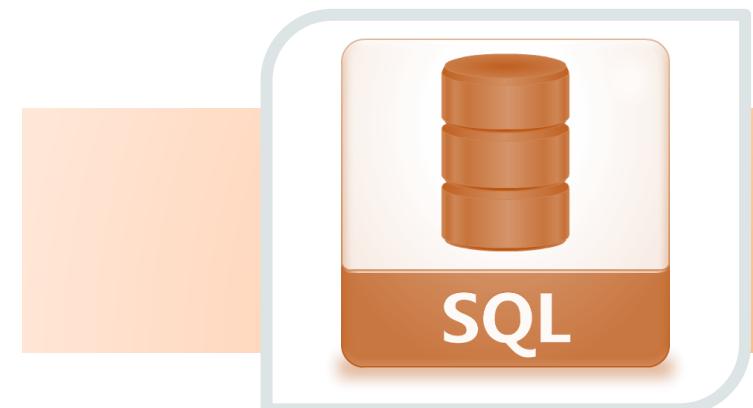
DUMMY
X

```
SELECT sysdate
FROM    dual;
```

SYSDATE
14-JUN-16

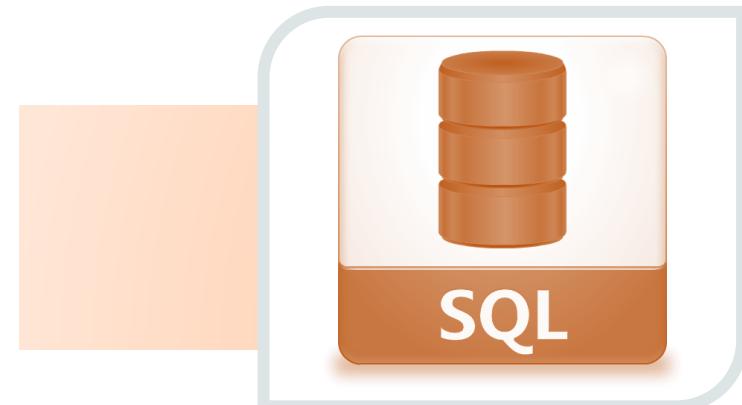
# Writing SQL Statements

- 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.



# Writing SQL Statements

- 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 the concatenation operator, literal character strings, the alternative quote operator, and the DISTINCT keyword
- DESCRIBE command

# Arithmetic Expressions

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

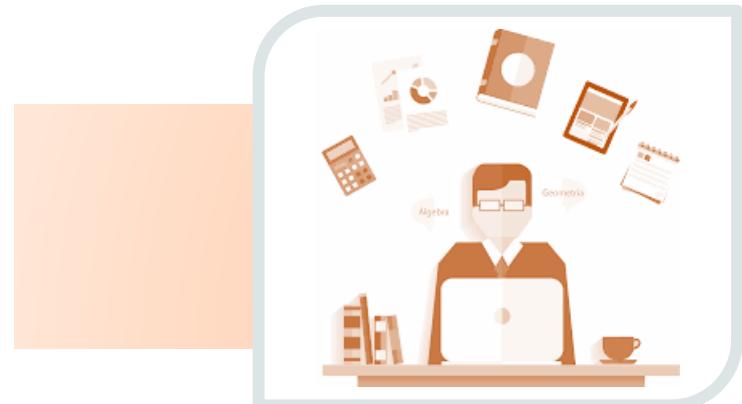
Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

# 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	Austin	4800	5100
7	Pataballa	4800	5100
8	Lorentz	4200	4500
9	Greenberg	12000	12300

...



# 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 inapplicable.
- Null is not the same as zero or a blank space.

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

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

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

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

# Null Values

Arithmetic expressions containing a null value evaluate to null.

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

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

46	Russell	67200
47	Partners	48600
48	Errazuriz	43200
49	Cambrault	39600

65	Marvins	8640
66	Lee	8160
67	Ande	7680
68	Banda	7440



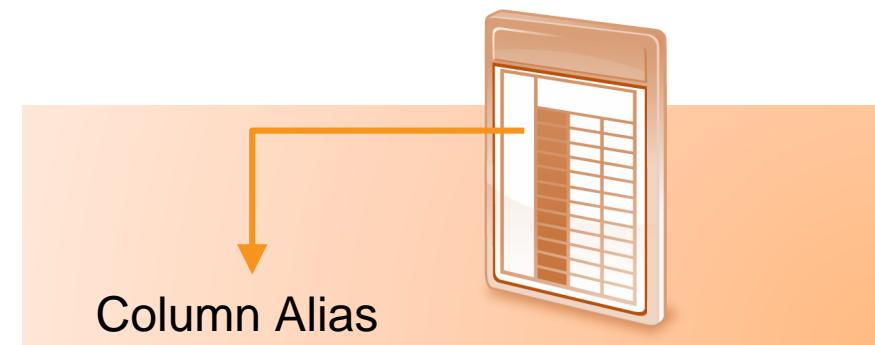
# Lesson Agenda

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

# 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 the SELECT statement
- Column aliases
- Use of the concatenation operator, literal character strings, the alternative quote operator, and the DISTINCT keyword
- DESCRIBE command

# Concatenation Operator

The concatenation operator:

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

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

Employees
1 AbelsA REP
2 AndeSA REP
3 AtkinsonST CLERK
4 AustinIT PROG
5 BaerPR REP
6 BaidaPU CLERK
7 BandaSA REP
8 BatesSA REP
9 BellSH CLERK

...

# 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	Ande is a SA_REP
3	Atkinson is a ST_CLERK
4	Austin is a IT_PROG
5	Baer is a PR_REP
6	Baida is a PU_CLERK
7	Banda is a SA_REP
8	Bates is a SA_REP
9	Bell is a SH_CLERK
...	

# 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 Purchasing Department's Manager Id: 114
4 Human Resources Department's Manager Id: 203
5 Shipping Department's Manager Id: 121
6 IT Department's Manager Id: 103
7 Public Relations Department's Manager Id: 204
8 Sales Department's Manager Id: 145
9 Executive Department's Manager Id: 100

# 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
9
100

2

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

DEPARTMENT_ID
1
2
3
4
5
6
7
8
9
(null)
10
20



# Lesson Agenda

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

# Displaying Table Structure

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

The screenshot shows the Oracle SQL Developer interface. At the top, there is a command line with the text "DESC [RIBE] tablename". Below it, the Connections tree shows a connection named "myconnection" expanded to show "Tables", which contains "COUNTRIES" and "DEPARTMENTS". The "DEPARTMENTS" node is selected. In the main area, a red box highlights the "Columns" tab of the toolbar. The "Actions..." button is also highlighted with a red box. The "Columns" tab is selected, and the table structure is displayed in a grid:

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 DEPARTMENT_ID	NUMBER(4,0)	No	(null)	1	Primary key column
2 DEPARTMENT_NAME	VARCHAR2(30 BYTE)	No	(null)	2	A not null column t
3 MANAGER_ID	NUMBER(6,0)	Yes	(null)	3	Manager_id of a dep
4 LOCATION_ID	NUMBER(4,0)	Yes	(null)	4	Location id where a

# Using the DESCRIBE Command

```
DESCRIBE employees
```

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

# 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;
```

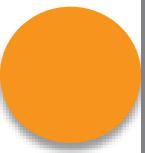
```
SELECT first_name+last_name AS name, job_Id,
      salary*12 yearly sal
   FROM employees;
```



# 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
- Describes the structure of a table



# 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

# Sorting Data

Restricting and Sorting Data

# Course Roadmap

Lesson 1: Introduction

**Unit 1: Retrieving, Restricting  
and Sorting Data**

Unit 2: Joins, Subqueries, and  
Set Operators

Unit 3: DML and DDL

▶ Lesson 2: Retrieving Data using SQL SELECT

▶ **Lesson 3: Restricting and Sorting Data**

▶ Lesson 4: Using Single-Row Functions to  
Customize Output

▶ Lesson 5: Using Conversion Functions and  
Conditional Expressions

← You are here!



# 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

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	105	Austin	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

# Filtering 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 display format for date is DD-MON-RR.

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

LAST_NAME	JOB_ID	DEPARTMENT_ID
Whalen	AD_ASST	10

```
SELECT last_name  
FROM employees  
WHERE hire_date = '17-OCT-95' ;
```

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
<b>BETWEEN ...AND...</b>	Between two values (inclusive)
<b>IN (set)</b>	Match any of a list of values
<b>LIKE</b>	Match a character pattern
<b>IS NULL</b>	Is a null value

# Using Comparison Operators

Let us look at some examples:

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

	LAST_NAME	SALARY
1	Baida	2900
2	Tobias	2800

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

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000	0.3	149	80

# BETWEEN Operator

You can use the BETWEEN operator to display rows based on a range of values:

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



Lower limit



Upper limit

	LAST_NAME	SALARY
1	Khoo	3100
2	Baida	2900
3	Tobias	2800
4	Himuro	2600

# 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	114	Raphaely	11000	100
4	120	Weiss	8000	100
5	121	Fripp	8200	100
6	122	Kaufling	7900	100
7	123	Vollman	6500	100
8	124	Mourgos	5800	100

# Pattern Matching Using the LIKE

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

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

FIRST_NAME
Sundar
Shelli

# Combining Wildcard Symbols

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

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

LAST_NAME
Colmenares
Doran
Fox

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

# Using NULL Conditions

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

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

	LAST_NAME	MANAGER_ID
1	King	(null)

# Defining Conditions Using Logical OPS

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

Operator	Meaning
<b>AND</b>	Returns TRUE if <i>both</i> component conditions are true
<b>OR</b>	Returns TRUE if <i>either</i> component condition is true
<b>NOT</b>	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	114	Raphaely	PU_MAN	11000
2	145	Russell	SA_MAN	14000
3	146	Partners	SA_MAN	13500
4	147	Errazuriz	SA_MAN	12000
5	148	Cambrault	SA_MAN	11000

# 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	108	Greenberg	FI_MGR	12000
5	114	Raphaely	PU_MAN	11000
6	120	Weiss	ST_MAN	8000
7	121	Fripp	ST_MAN	8200
8	122	Kaufling	ST_MAN	7900

# Using the NOT Operator

NOT is used to negate a condition:

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

	LAST_NAME	JOB_ID
1	Baer	PR_REP
2	Baida	PU_CLERK
3	Bell	SH_CLERK
4	Bull	SH_CLERK
5	Cabrio	SH_CLERK
6	Cambrault	SA_MAN
7	Chen	FI_ACCOUNT
8	Chung	SH_CLERK
9	Colmenares	PU_CLERK



# 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

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

You can use parentheses to override rules of precedence.

# 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	Austin	60	4800
4	Pataballa	60	4800
5	Lorentz	60	4200

```
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	Russell	80	14000
2	Partners	80	13500



# 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

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

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

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

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	King	AD_PRES		90 17-JUN-87
2	Whalen	AD_ASST		10 17-SEP-87
3	Kochhar	AD_VP		90 21-SEP-89
4	Hunold	IT_PROG		60 03-JAN-90
5	Ernst	IT_PROG		60 21-MAY-91
6	De Haan	AD_VP		90 13-JAN-93
7	Mavris	HR_REP		40 07-JUN-94

...

# 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

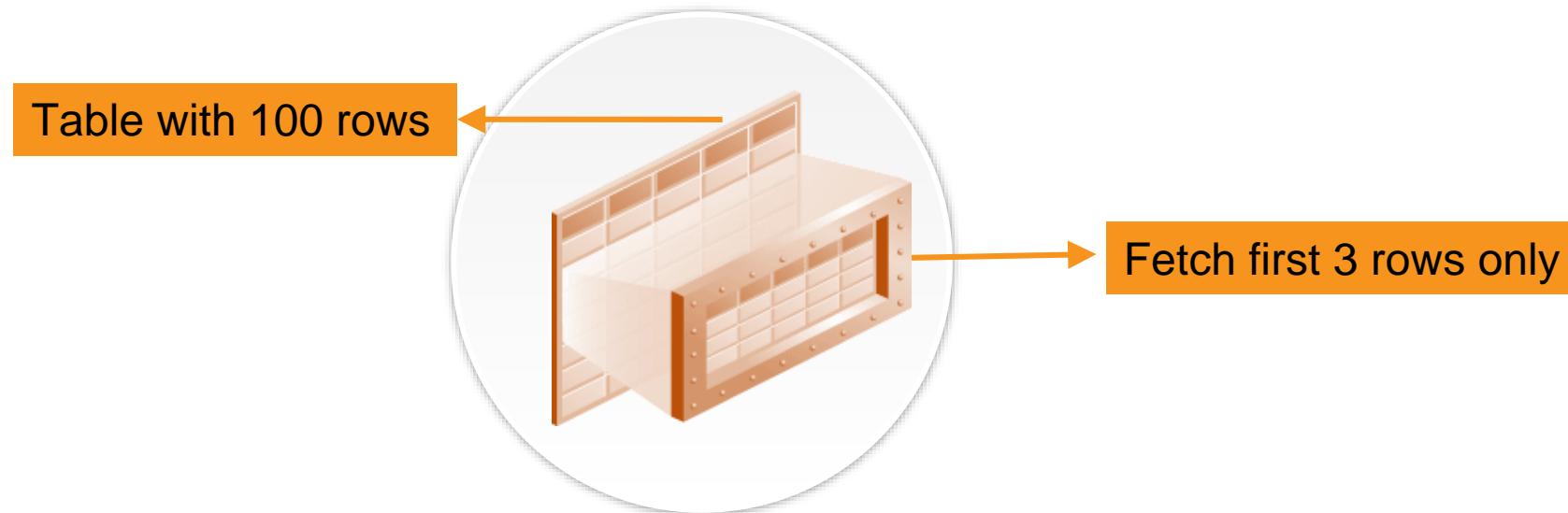


# Lesson Agenda

- 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.



# Using SQL Row Limiting

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:

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

EMPLOYEE_ID	FIRST_NAME
1	100 Steven
2	101 Neena
3	102 Lex
4	103 Alexander
5	104 Bruce

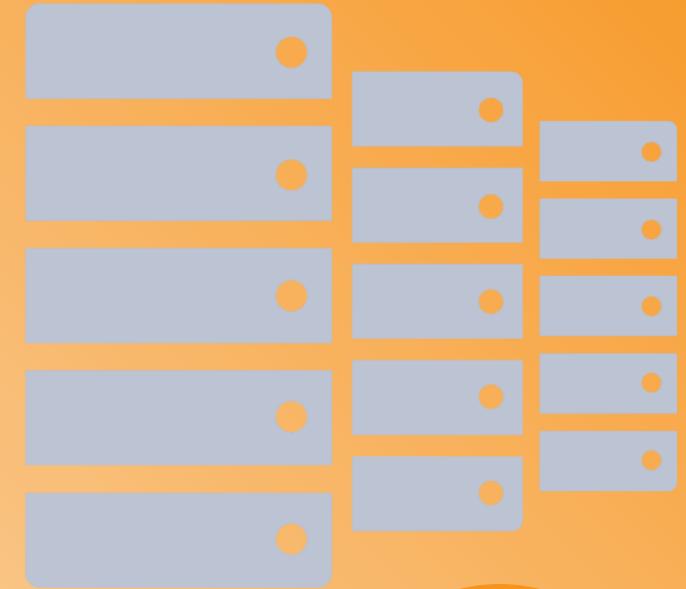
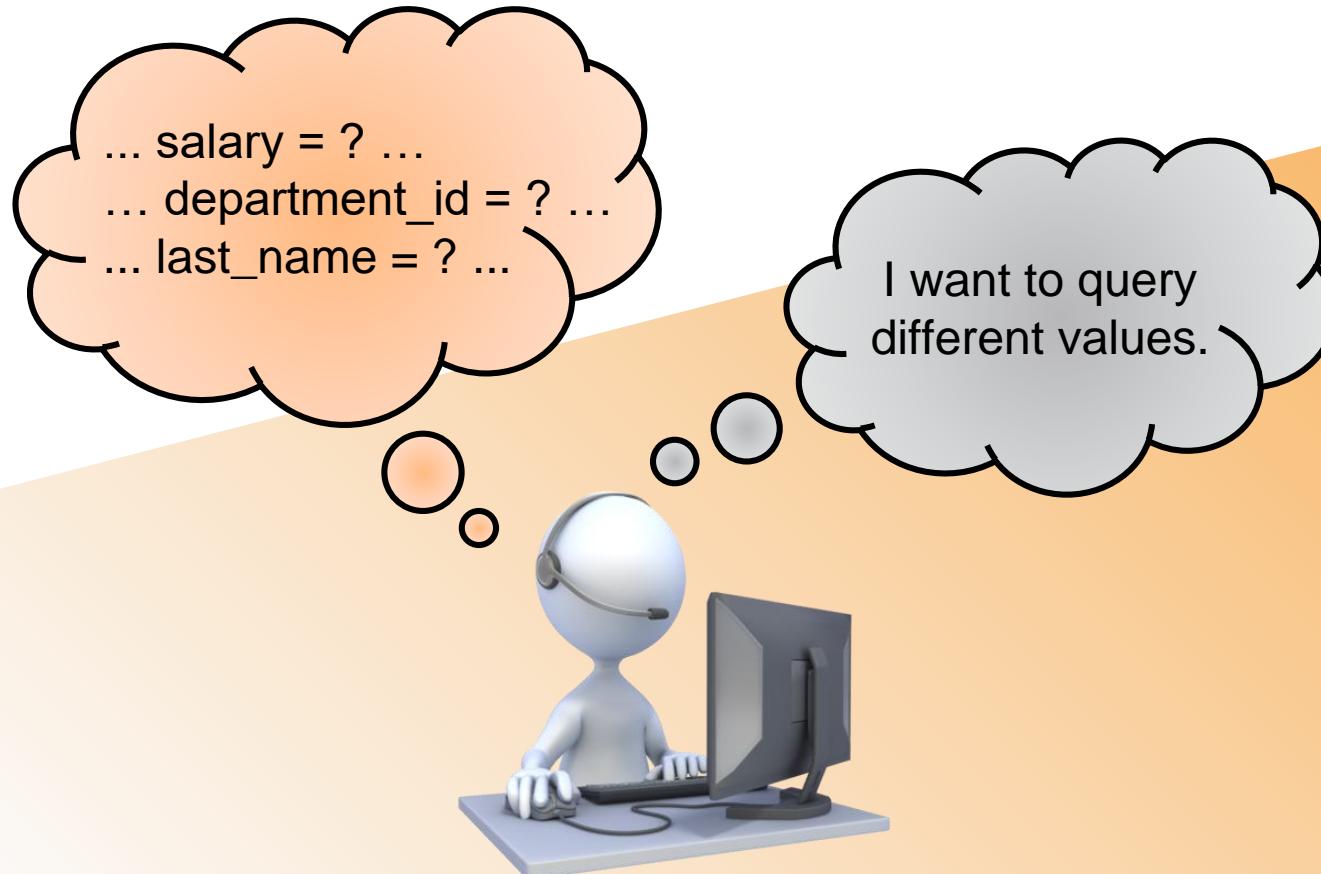
```
SELECT employee_id, first_name  
FROM employees  
ORDER BY employee_id  
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

EMPLOYEE_ID	FIRST_NAME
1	105 David
2	106 Valli
3	107 Diana
4	108 Nancy
5	109 Daniel

# 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

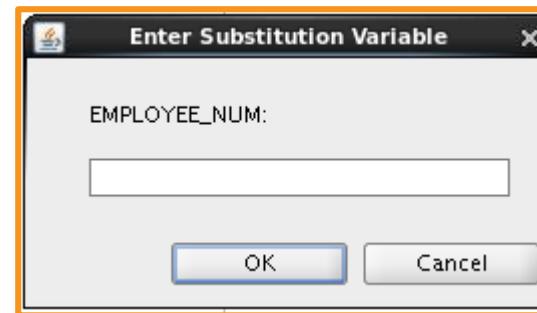
- Use substitution variables to:
  - Temporarily store values with single-ampersand (&) and double-ampersand (&&) substitution
- Use substitution variables to supplement the following:
  - WHERE conditions
  - ORDER BY clauses
  - Column expressions
  - Table names
  - Entire SELECT statements



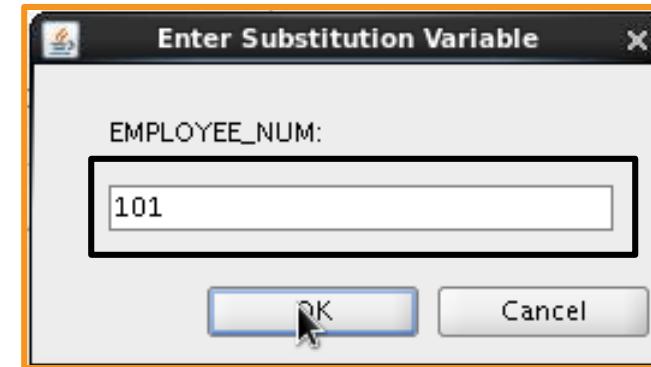
# Using the Single-Ampersand

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

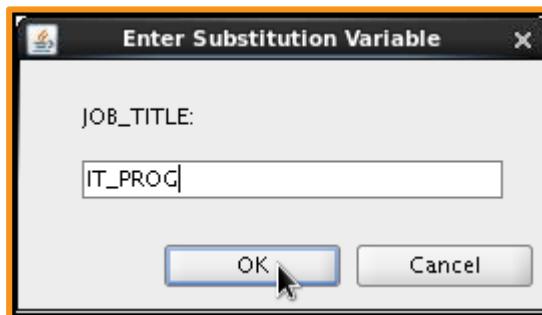


	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

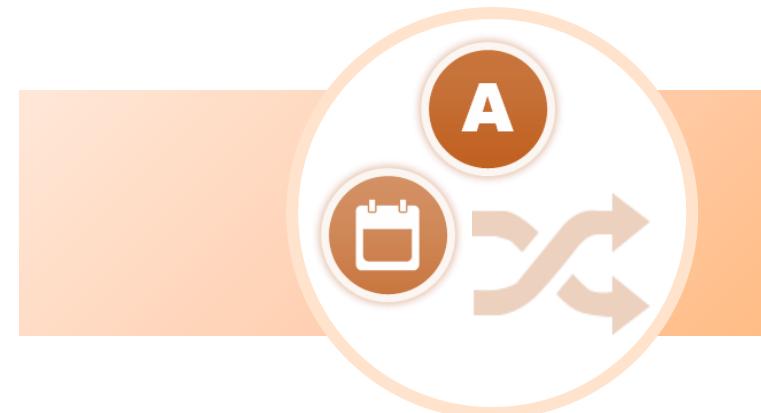
# Character and Date values with Substitution Variables

Use single quotation marks for date and character values:

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

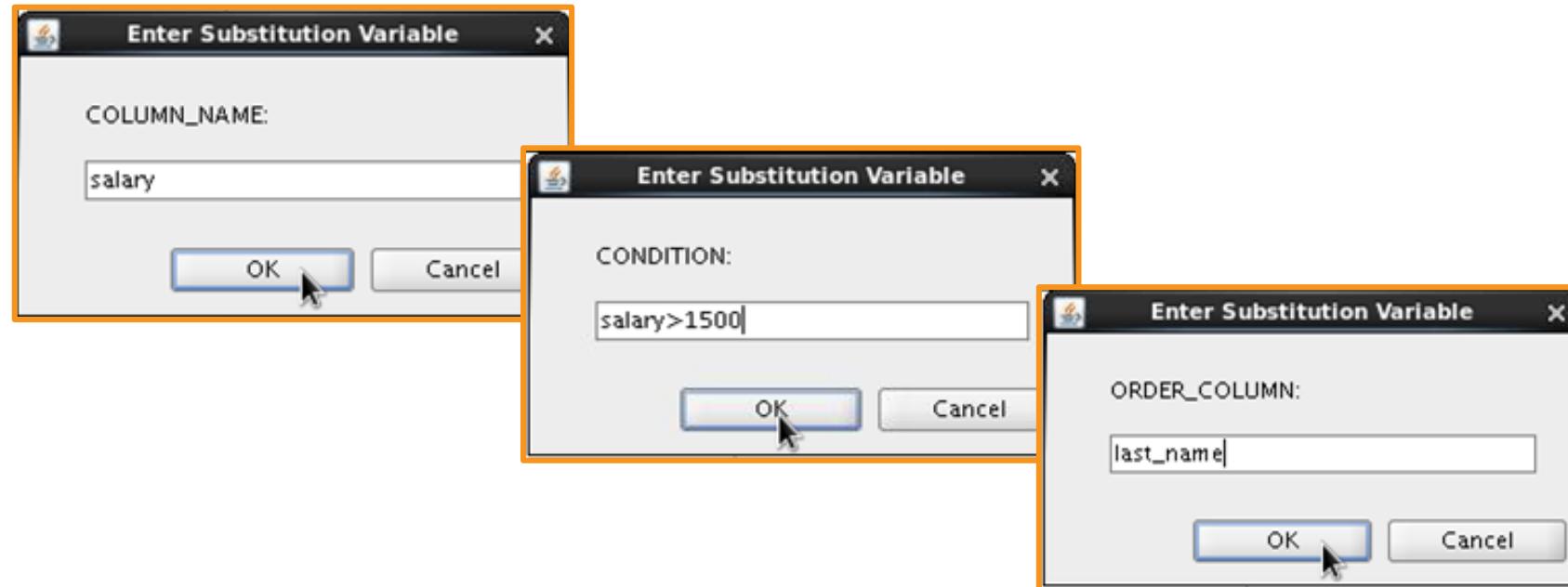


	LAST_NAME	DEPARTMENT_ID	SALARY*12
1	Hunold	60	108000
2	Ernst	60	72000
3	Austin	60	57600



# Specifying Column Names, Expressions

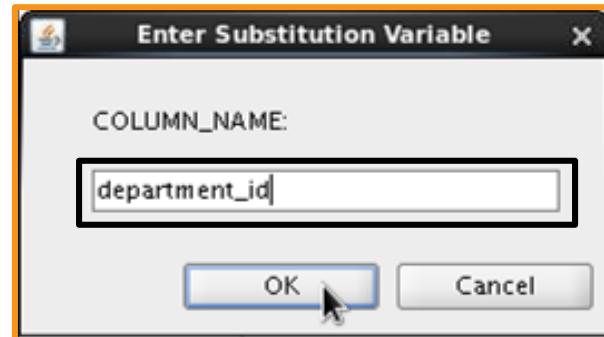
```
SELECT employee_id, last_name, job_id,&column name  
FROM employees  
WHERE &condition  
ORDER BY &order column ;
```



# Using the Double-Ampersand

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

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

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

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

SQL>
```

# 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 DEFINE Command

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

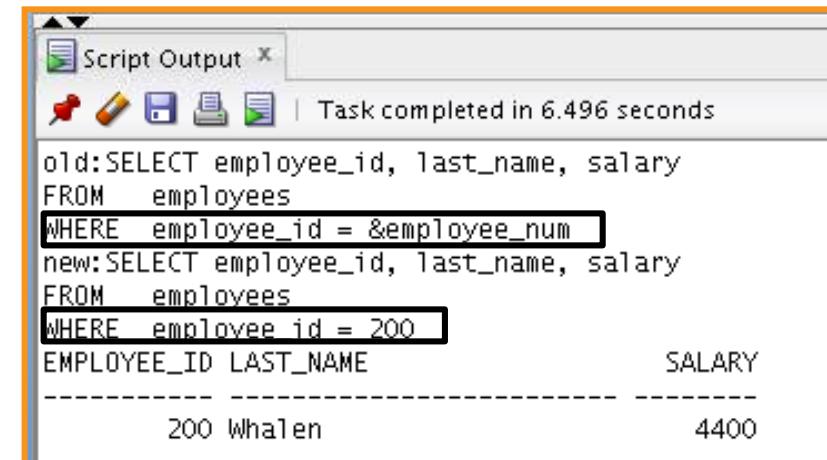
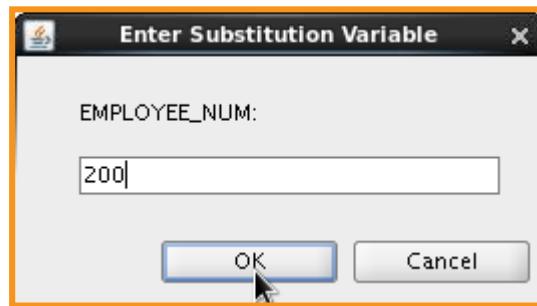
```
DEFINE employee_num = 200
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num;
UNDEFINE employee_num
```

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	200	Whalen	4400	10

# 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. It displays the original query with the substitution variable, followed by the modified query with the value substituted. The output then shows the resulting table with one row for employee ID 200, last name Whalen, and salary 4400.

```
old:SELECT employee_id, last_name, salary  
FROM employees  
WHERE employee_id = &employee_num  
new:SELECT employee_id, last_name, salary  
FROM employees  
WHERE employee_id = 200  
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

# Single Row Functions

Using Single-Row Functions  
to Customize Output

# Course Roadmap

Lesson 1: Introduction

**Unit 1: Retrieving, Restricting,  
and Sorting Data**

Unit 2: Joins, Subqueries, and  
Set Operators

Unit 3: DML and DDL

▶ Lesson 2: Retrieving Data using SQL SELECT

▶ Lesson 3: Restricting and Sorting Data

▶ **Lesson 4: Using Single-Row Functions to  
Customize Output**

▶ Lesson 5: Using Conversion Functions and  
Conditional Expressions

← You are here!

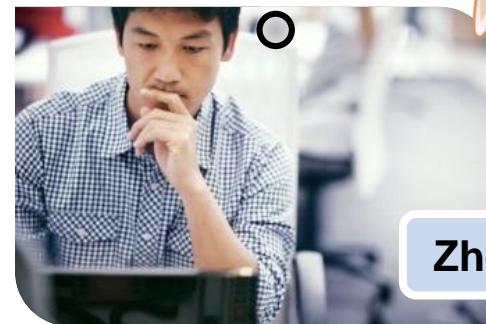
# Objectives

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

# HR Application Scenario

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



Zhen

**HR Application**

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

Operation:

Average (Salary)      GO

**HR Application**

The average salary is \$10500.



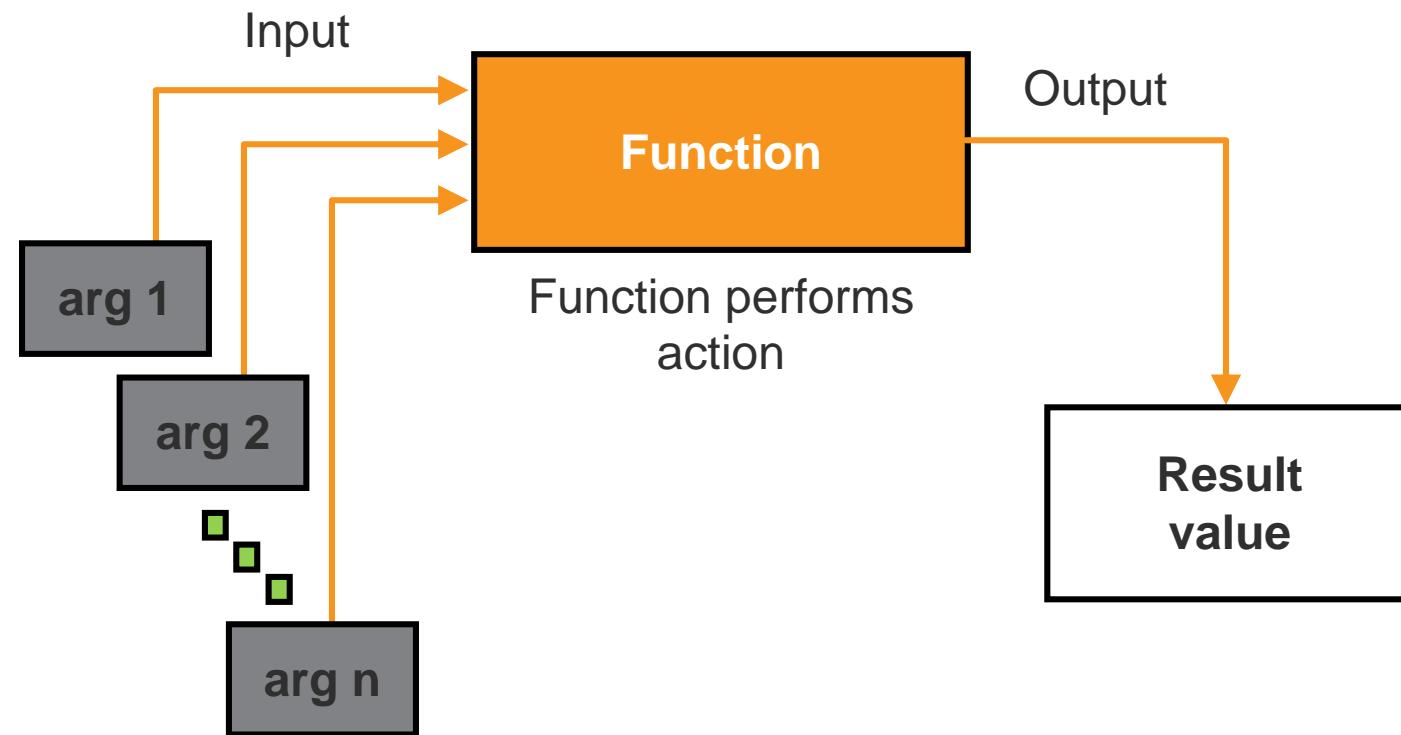
- Accounts
- IT
- Sales
- Marketing



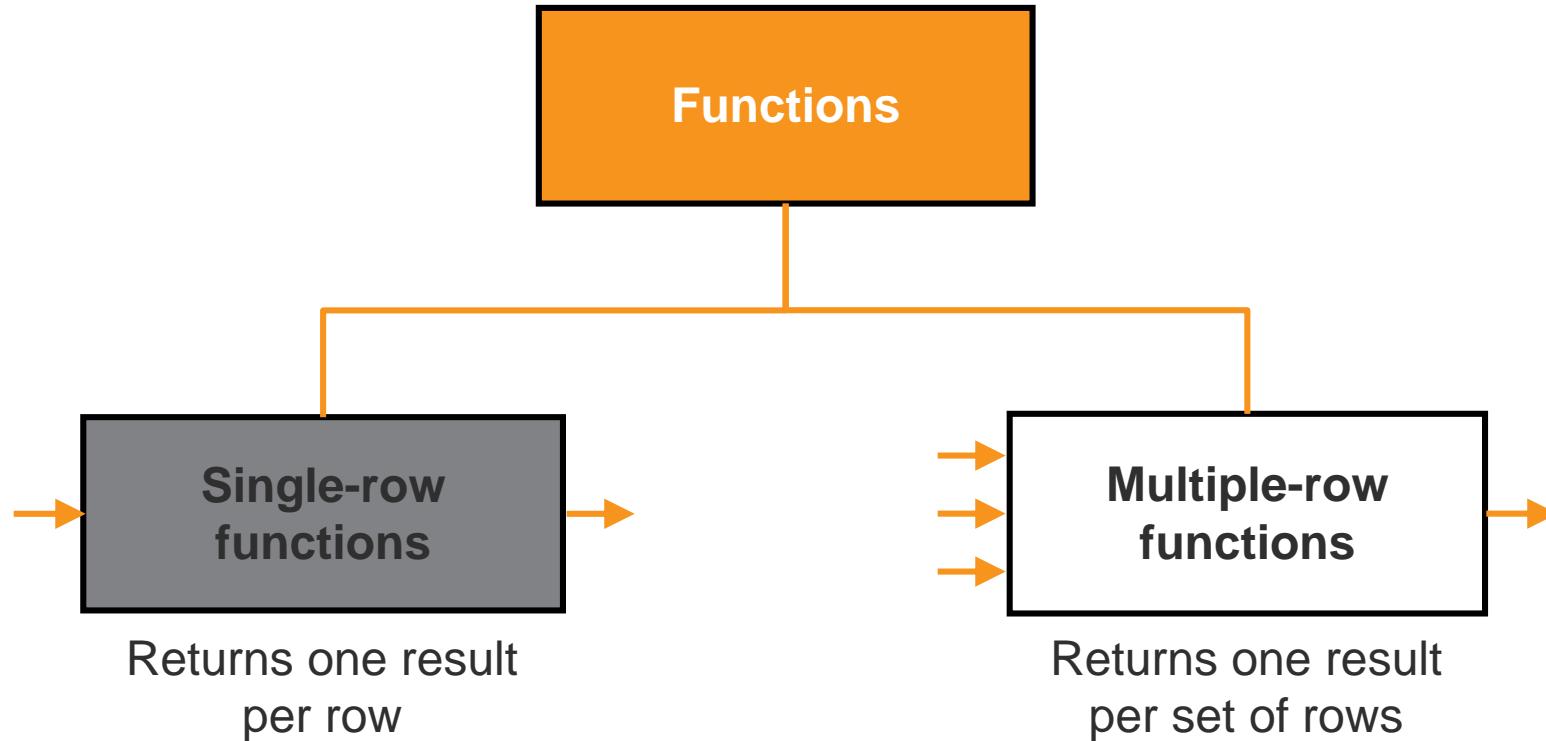
# Lesson Agenda

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

# SQL Functions



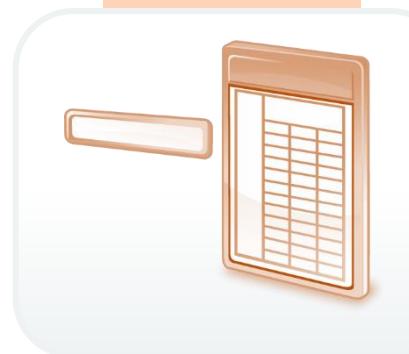
# Two Types of SQL Functions



# Single-Row Functions

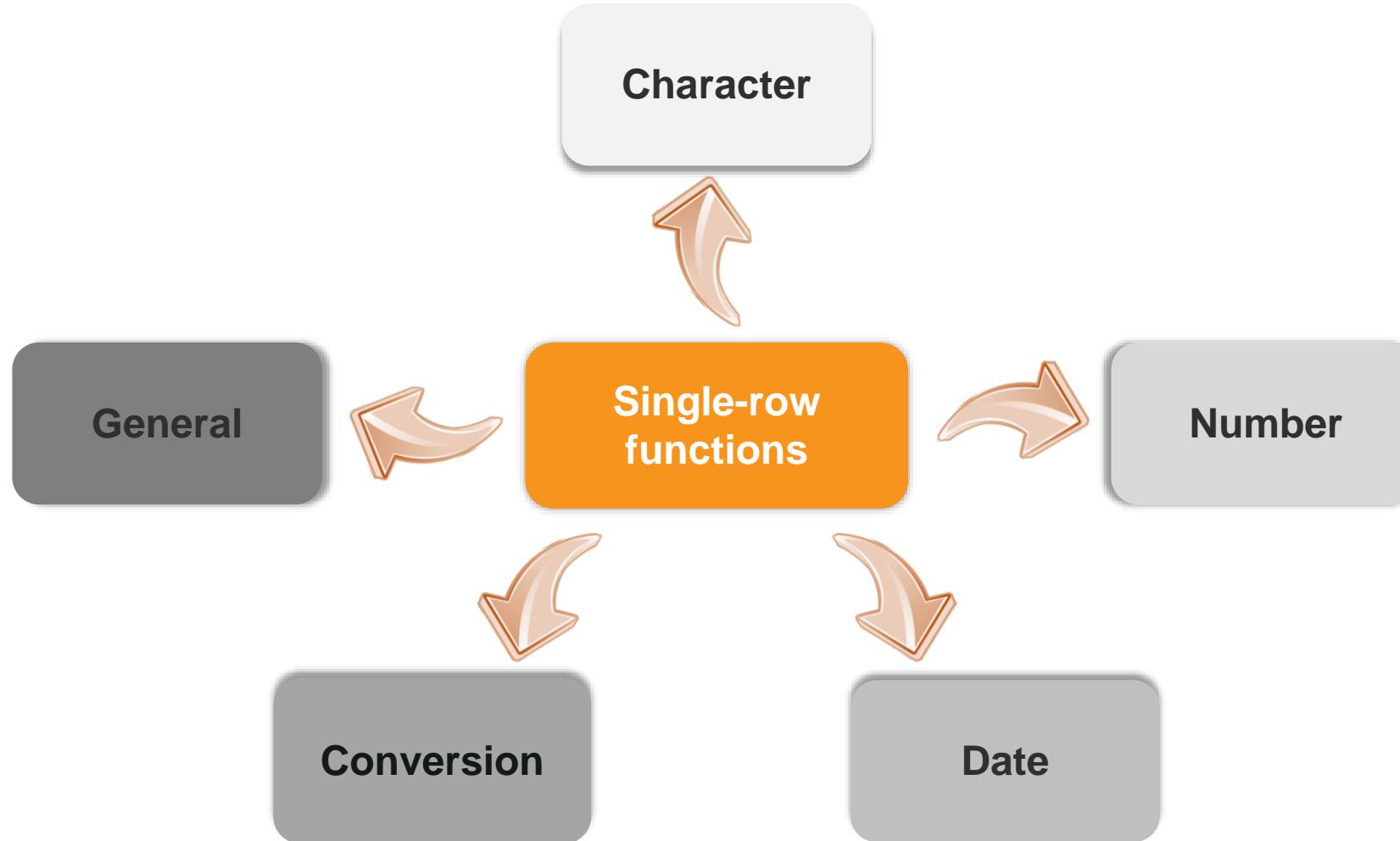
Single-row functions:

- 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

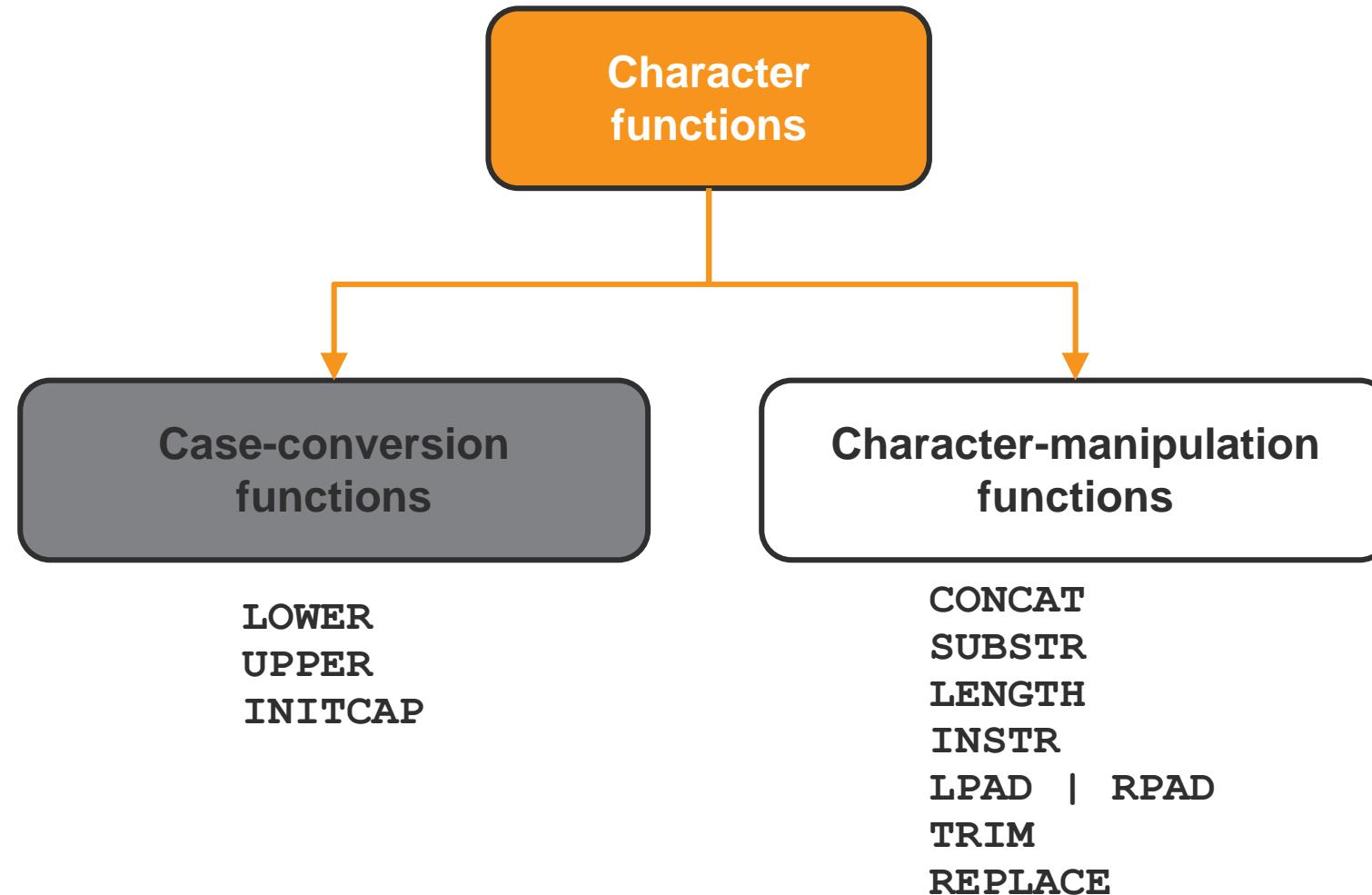




# Lesson Agenda

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

# Character Functions



# Case-Conversion Functions

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

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

# 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

You can use these functions to manipulate character strings:

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

# Using Character-Manipulation

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

1

	LAST_NAME	Job
1	Abel	Job category is SA_REP
2	Ande	Job category is SA_REP
3	Baer	Job category is PR_REP
4	Banda	Job category is SA_REP

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

2

	EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
1	102	LexDe Haan	7	5
2	105	DavidAustin	6	0
3	110	JohnChen	4	0
	110	ManeetKumar	-	-

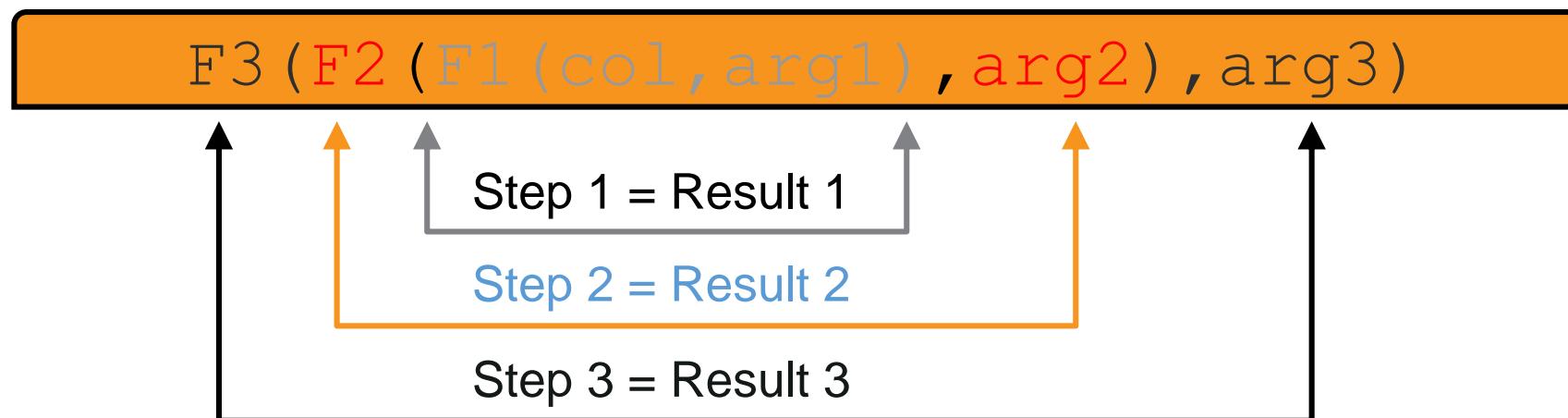


# Lesson Agenda

- 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	Austin	AUSTIN_US
4	Pataballa	PATABALL_US
5	Lorentz	LORENTZ_US



# Lesson Agenda

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

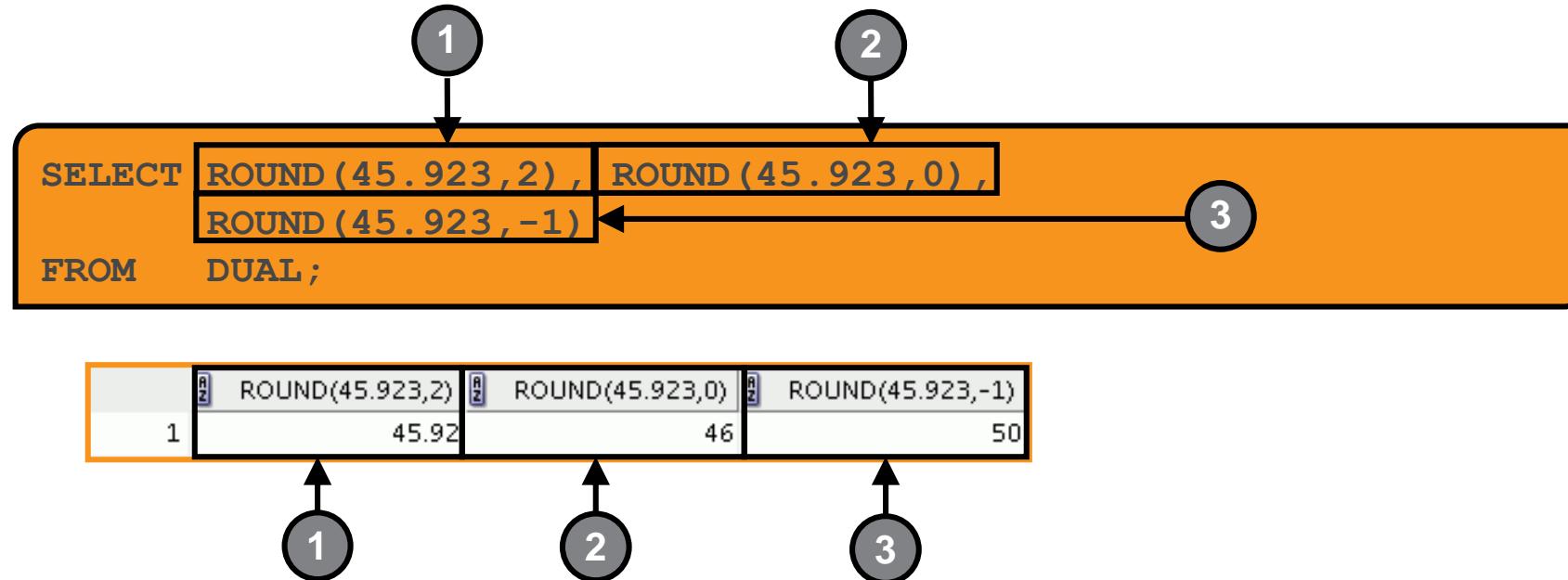
# Numeric Functions

- 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

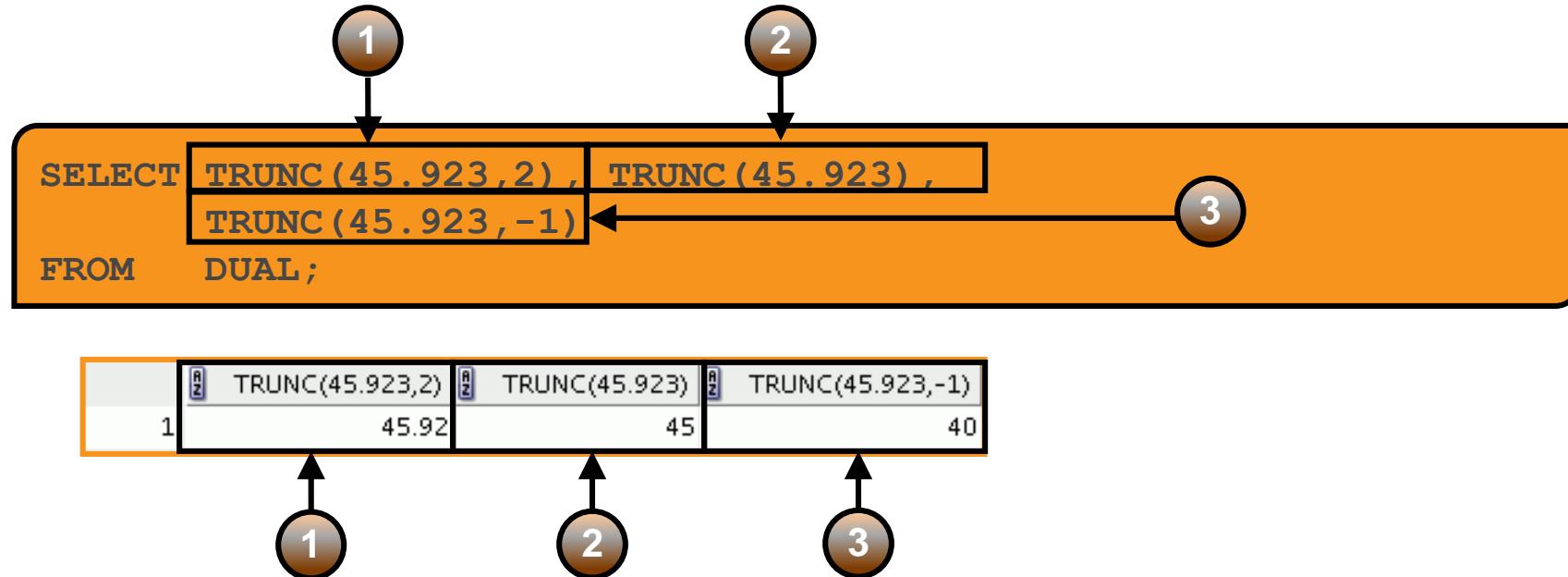
# Numeric Functions

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



# 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		166 Ande
3		130 Atkinson
4		204 Baer
5		116 Baida
6		172 Bates
7		192 Bell
8		148 Cambrault
9		154 Cambrault
10		110 Chen
11		188 Chung
12		142 Davies



# 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-2013';
```

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

# 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

Use the SYSDATE function to get:

- Date
- Time

```
SELECT sysdate  
FROM dual;
```

	SYSDATE
1	13-MAR-19

# 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
America/New_York	13-MAR-19

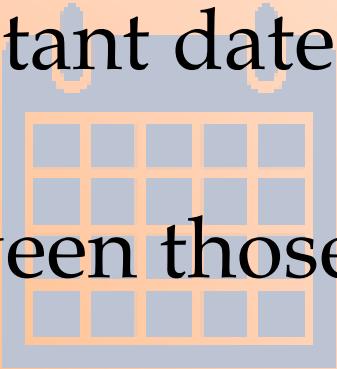
- CURRENT\_TIMESTAMP returns the current date and time from the user session.

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

SESSIONTIMEZONE	CURRENT_TIMESTAMP
America/New_York	13-MAR-19 08.25.50.192287000 PM AMERICA/NEW_YORK

# 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

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

LAST_NAME	WEEKS
King	1656.1220436507936507936507936507936508
Kochhar	1537.979186507936507936507936507936507943
De Haan	1365.122043650793650793650793650793650794



# 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	Date of the next occurrence of the specified day
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

# Using Date Functions

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

# Using ROUND and TRUNC Functions

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

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

# Quiz

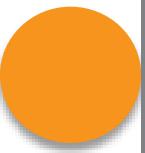
Which four of the following statements are true about single-row functions?

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

# Summary

In this lesson, you should have learned how to:

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



# 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

# Conversion Functions

Using Conversion Functions and Conditional Expressions

# Course Roadmap

Lesson 1: Introduction

**Unit 1: Retrieving, Restricting, and Sorting Data**

Unit 2: Joins, Subqueries, and Set Operators

Unit 3: DML and DDL

▶ Lesson 2: Retrieving Data using SQL SELECT

▶ Lesson 3: Restricting and Sorting Data

▶ Lesson 4: Using Single-Row Functions to Customize Output

▶ Lesson 5: Using Conversion Functions and Conditional Expressions

← You are here!

# Objectives

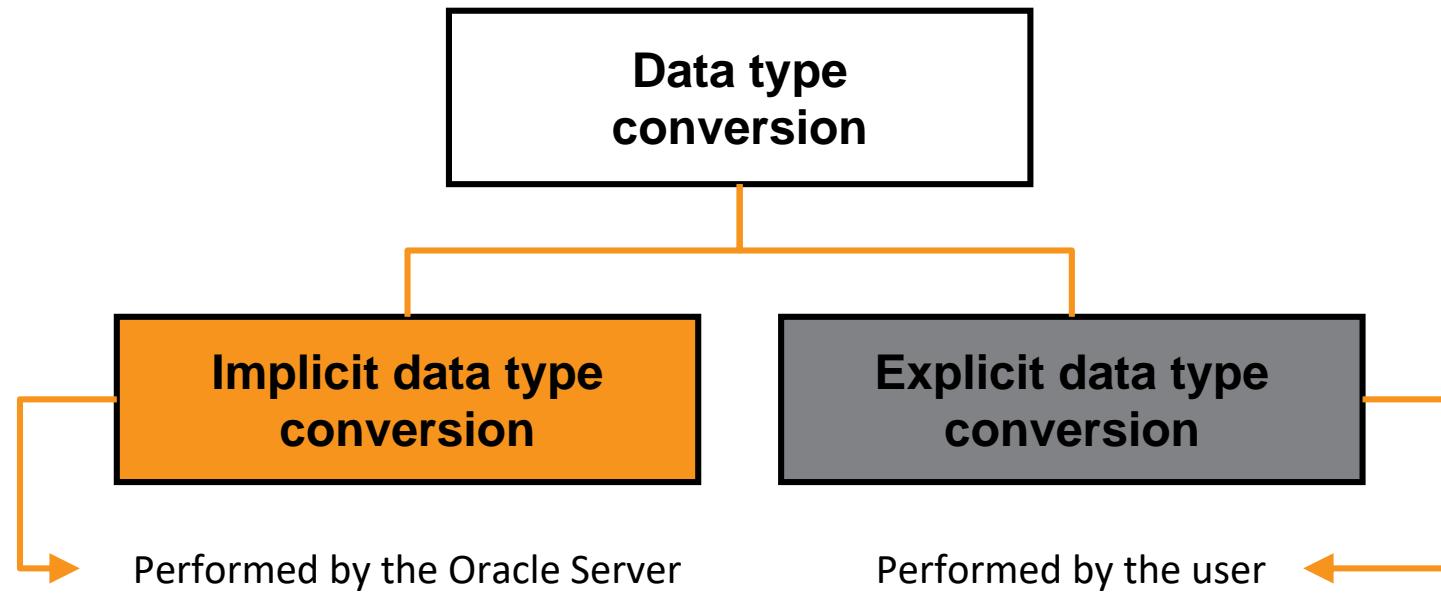
After completing this lesson, you should be able to do the following:

- 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

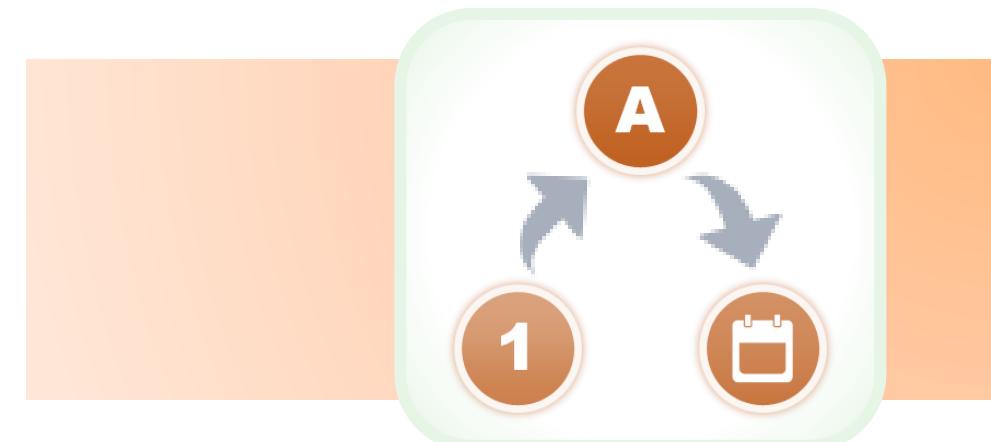
# Conversion Functions



# Implicit Data Type Conversion

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

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

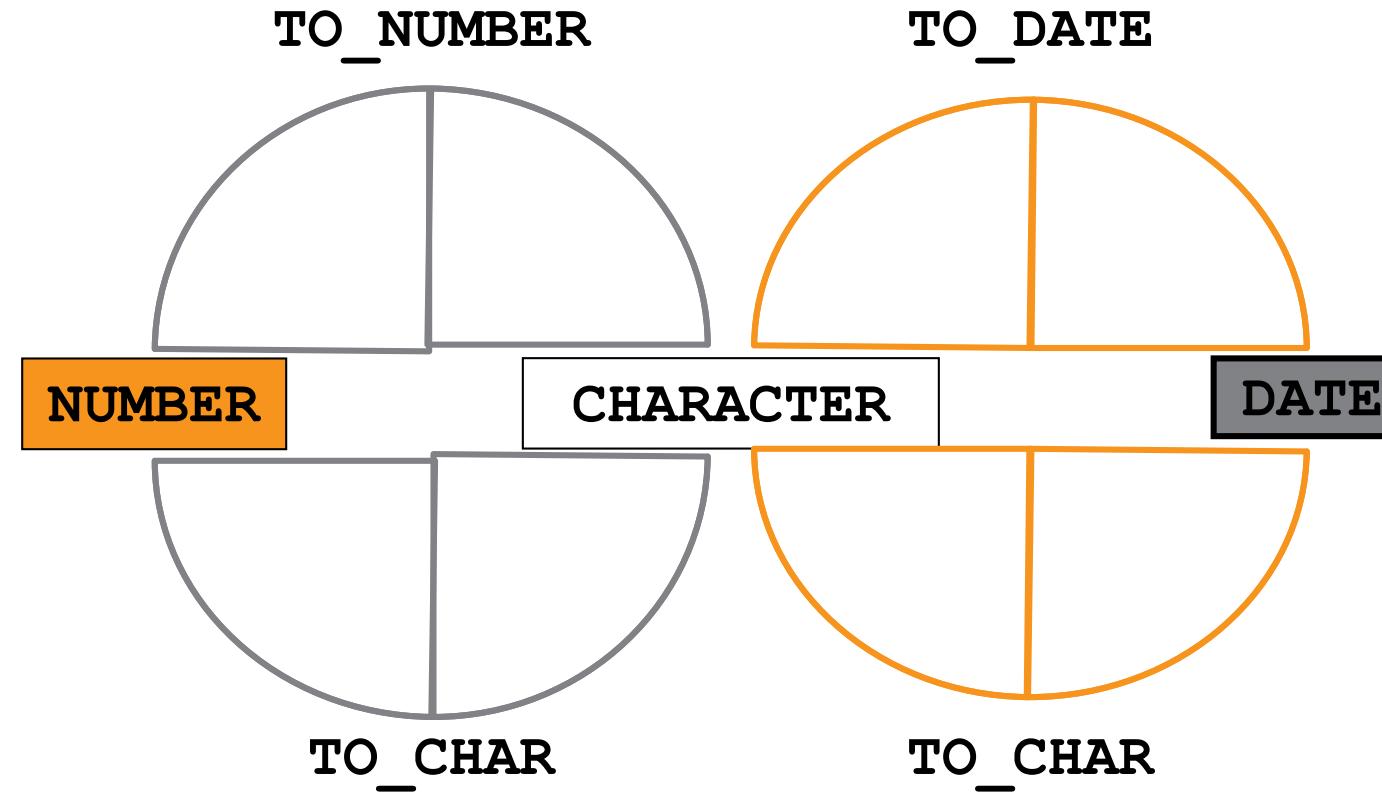


# Implicit Data Type Conversion

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

Example:

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

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

	EMPLOYEE_ID	MONTH_HIRED
1		205 06/10

# Elements of the Date Format

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

- Time elements help you format the time portion of the date:

HH24 :MI :SS AM

15:45:32 PM

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

DD "of" MONTH

12 of OCTOBER

- Number suffixes help in spelling out numbers:

ddspth

fourteenth

# Using the TO\_CHAR Function

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

	LAST_NAME	HIREDATE
1	King	17 June 2011
2	Kochhar	21 September 2009
3	De Haan	13 January 2009
4	Hunold	3 January 2014
5	Ernst	21 May 2015
6	Lorentz	7 February 2015
7	Mourgos	16 November 2015
8	Rajs	17 October 2011

...

# Using the TO\_CHAR Function

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

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

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

# Using the TO\_CHAR Function

Let us look at an example:

```
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

- 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'])
```

# Using TO\_CHAR and TO\_DATE Functions with the RR Date

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

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

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



# 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 pertain to using nulls and can be used with any data type:

`NVL (expr1, expr2)`

`NVL2 (expr1, expr2, expr3)`

`NULLIF (expr1, expr2)`

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

# NVL Function

Converts a null value to an actual value:

- Data types that can be used are date, character, and number.
- Data types must match.
- Examples:
  - NVL (commission\_pct, 0)
  - NVL (hire\_date, '01-JAN-97')
  - NVL (job\_id, 'No Job Yet')

`NVL (expr1, expr2)`

# 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

NVL2 (expr1, expr2, expr3)

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

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



# Using the NULLIF Function

NULLIF (expr1, expr2)

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

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

...

1

2

3

# Using the COALESCE Function

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

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

# 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

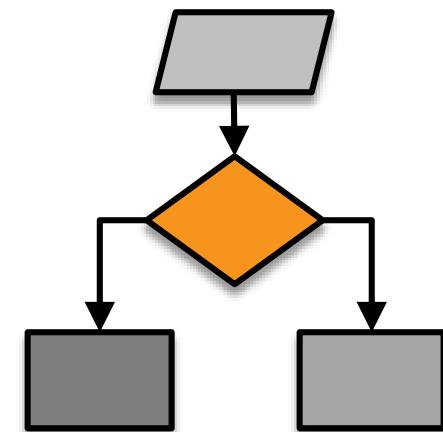


# Lesson Agenda

- 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

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



# 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.

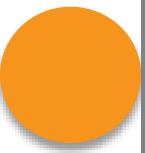
- a. True
- b. False



# 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

# Group Functions

Reporting Aggregated Data  
Using the Group Functions

# Course Roadmap

Lesson 1: Introduction

Unit 1: Retrieving, Restricting, and Sorting Data

**Unit 2: Joins, Subqueries, and Set operators**

Unit 3: DML and DDL

▶ Lesson 6: Reporting Aggregated Data Using Group Functions

▶ Lesson 7: Displaying Data from Multiple Tables Using Joins

▶ Lesson 8: Using Subqueries to Solve Queries

▶ Lesson 9: Using Set Operators

You are here!

# 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

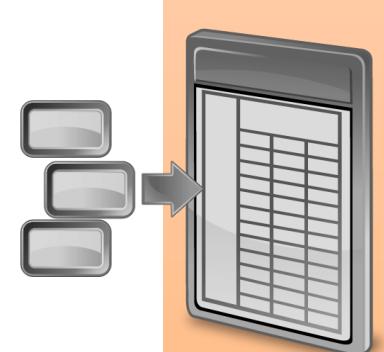
Group functions operate on sets of rows to give one result per group.

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

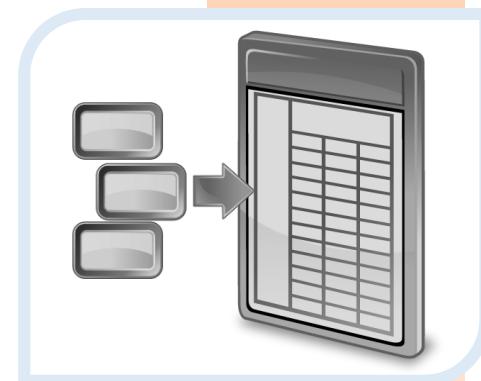
Maximum salary in  
EMPLOYEES table

MAX(SALARY)  
24000



# Types of Group Functions

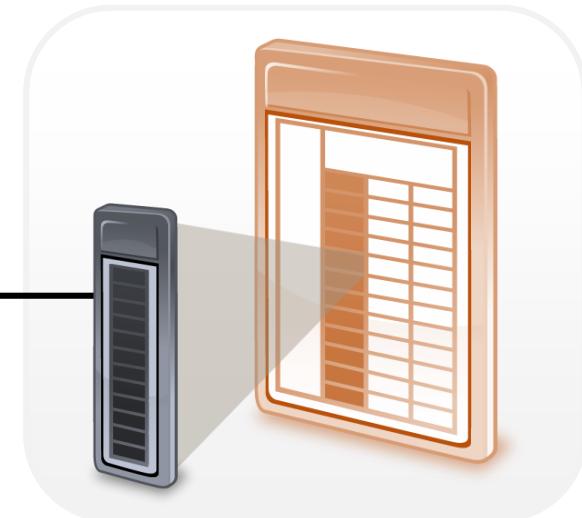
- AVG
- COUNT
- MAX
- MIN
- SUM
- LISTAGG
- STDDEV
- VARIANCE



# Group Functions: Syntax

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

Group all rows in a column



# Using the AVG and SUM Functions

You can use the AVG and SUM functions for numeric data.

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

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

# Using the MIN and MAX Functions

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

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

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



# Using the COUNT Function

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

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

1

	COUNT(*)
1	5

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

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

2

	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:

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

1

	Avg(COMMISSION_PCT)
1	0.2125

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

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

2

	Avg(NVL(COMMISSION_PCT,0))
1	0.0425



# 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



# 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

# 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

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

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

# 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 Multiple Columns

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

```
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

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

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

1

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

A GROUP BY clause must be added to count the last names for each department\_id.

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

2

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

Either add job\_id in the GROUP BY clause or remove the job\_id column from the SELECT list.

# Illegal Queries Using Group

- 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

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 the 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;
```

AZ MAX(AVG(SALARY))

# Quiz

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

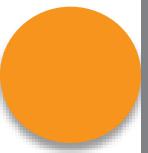
- a. You cannot use a column alias in the GROUP BY clause.
- b. The GROUP BY column must be in the SELECT clause.
- c. By using a WHERE clause, you can exclude rows before dividing them into groups.
- d. The GROUP BY clause groups rows and ensures the order of the result set.
- e. If you include a group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

# 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

# Joins

- Displaying Data from Multiple Tables Using Joins

# Course Roadmap

Lesson 1: Introduction

Unit 1: Retrieving, Restricting,  
and Sorting Data

**Unit 2: Joins, Subqueries, and  
Set Operators**

Unit 3: DML and DDL

▶ Lesson 6: Reporting Aggregated Data Using  
Group Functions

▶ Lesson 7: Displaying Data from Multiple  
Tables Using Joins

▶ **Lesson 8: Using Subqueries to Solve  
Queries**

▶ Lesson 9: Using Set Operators

You are here!

# Objectives

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

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

# 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

# Why Join?

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



Jody

## HR Application

Select the desired columns:

### EMPLOYEE

first_name	>	employee_id
last_name	>>	job_id
...		

### JOBs

min_salary	>	job_id
max_salary	>>	job_title
...		

GO

Combines columns  
from both the tables

## HR Application

Emp_ID	Job_id	Job_title
206	AC_ACCOUNTANT	Public Accountant
103	AC_MGR	Accounting Manager
100	AD_PRES	President

# Obtaining Data from Multiple Tables

EMPLOYEES

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

JOBS

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

...

...

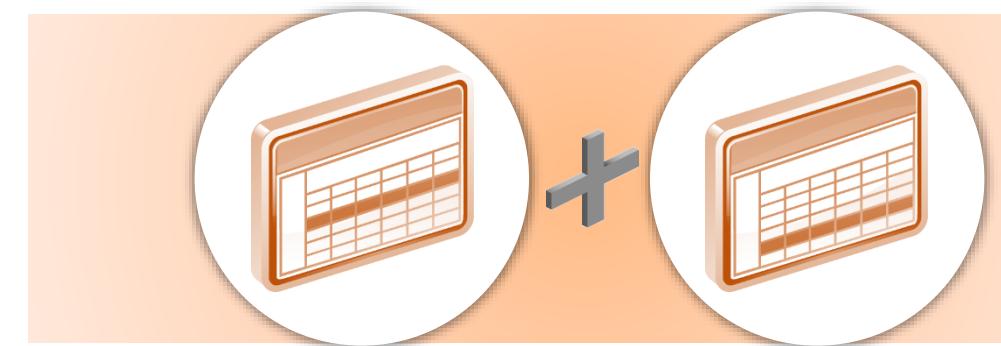
	EMPLOYEE_ID	JOB_ID	JOB_TITLE
1	206	AC_ACCOUNT	Public Accountant
2	205	AC_MGR	Accounting Manager
3	200	AD_ASST	Administration Assistant
4	100	AD_PRES	President
5	101	AD_VP	Administration Vice President
6	102	AD_VP	Administration Vice President
7	109	FI_ACCOUNT	Accountant

...

# 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
  - Nonequi joins
  - 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	206	William	AC_ACCOUNT	Public Accountant
2	205	Shelley	AC_MGR	Accounting Manager
3	200	Jennifer	AD_ASST	Administration Assistant
4	100	Steven	AD_PRES	President
5	102	Lex	AD_VP	Administration Vice President
6	101	Neena	AD_VP	Administration Vice President
7	103	Alexander	IT_PROG	Programmer
8	104	Bruce	IT_PROG	Programmer
9	107	Diana	IT_PROG	Programmer
10	201	Michael	MK_MAN	Marketing Manager
11	202	Pat	MK_REP	Marketing Representative
12	149	Eleni	SA_MAN	Sales Manager
13	174	Ellen	SA REP	Sales Representative
14	178	Kimberely	SA REP	Sales Representative
15	176	Jonathon	SA REP	Sales Representative
16	143	Randall	ST_CLERK	Stock Clerk
17	142	Curtis	ST_CLERK	Stock Clerk
18	141	Trenna	ST_CLERK	Stock Clerk
19	144	Peter	ST_CLERK	Stock Clerk
20	124	Kevin	ST_MAN	Stock Manager

# Creating Joins with the USING Clause

When should you use the USING clause?

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

# Joining Column Names

EMPLOYEES

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

DEPARTMENTS

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

Foreign key

Primary key

# 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

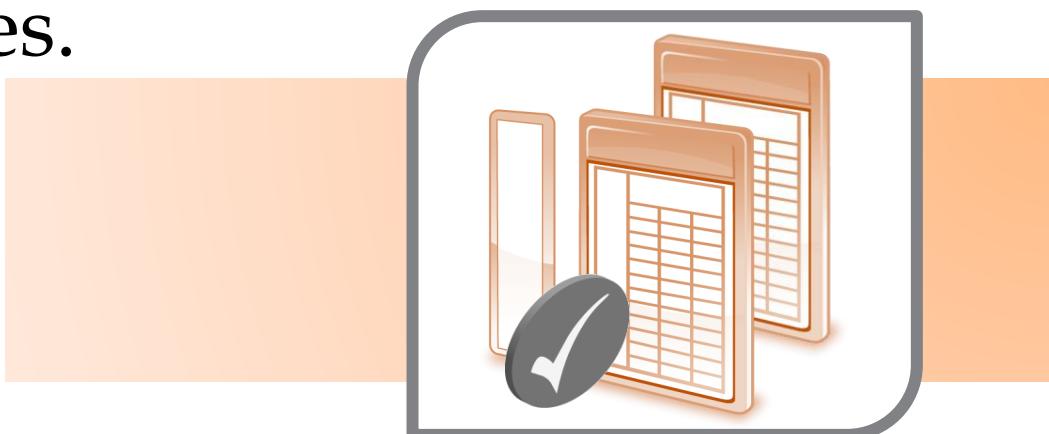
...

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



# 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 the columns to join.
- Use the ON clause to separate the join condition from other search conditions.
- The ON clause makes code easy to understand.



# 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

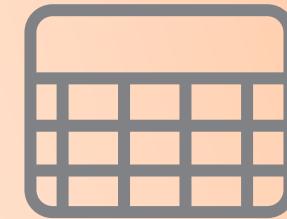
...



# 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  
  ON      (e.department_id = d.department_id)  
 AND     e.manager_id = 149 ;
```

OR

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

# 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

# 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

# Nonequi joins

EMPLOYEES

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

JOB\_GRADES

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

The JOB\_GRADES table defines the LOWEST\_SAL and HIGHEST\_SAL range of values for each GRADE\_LEVEL.

Therefore, the GRADE\_LEVEL column can be used to assign grades to each employee based on his salary.

# 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
- Nonequijoins
- 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

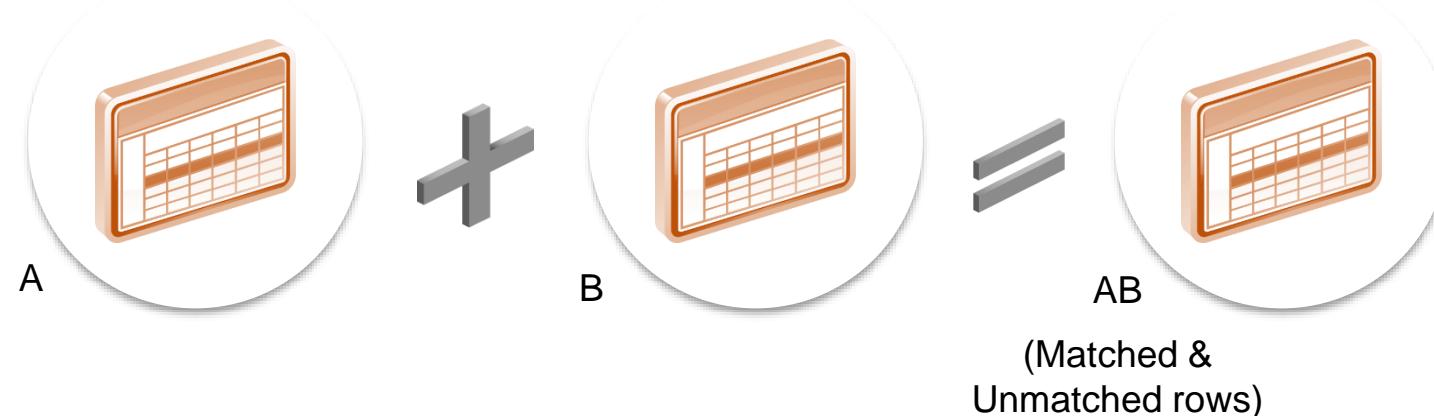
There are no employees in department 190.

Employee "Grant" has not been assigned a department ID.

Therefore, the above two records do not appear in the equijoin result.

# 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

A Cartesian product:

- Is a join of every row of one table to every row of another table
- Generates a large number of rows and the result is rarely useful



# 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 a Cartesian product of two tables.
- To create a Cartesian product, specify CROSS JOIN in your SELECT statement.

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

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

158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting

# Quiz

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

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

# Summary

In this lesson, you should have learned how to :

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

# Practice 7: Overview

This practice covers the following topics:

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

# Subqueries

- Using Subqueries to Solve Queries

# Course Roadmap

Lesson 1: Introduction

Unit 1: Retrieving, Restricting,  
and Sorting Data

**Unit 2: Joins, Subqueries, and  
Set Operators**

Unit 3: DML and DDL

▶ Lesson 6: Reporting Aggregated Data Using  
Group Functions

▶ Lesson 7: Displaying Data from Multiple  
Tables Using Joins

▶ **Lesson 8: Using Subqueries to Solve  
Queries**

▶ Lesson 9: Using Set Operators

← You are here!

# Objectives

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

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

# Lesson Agenda

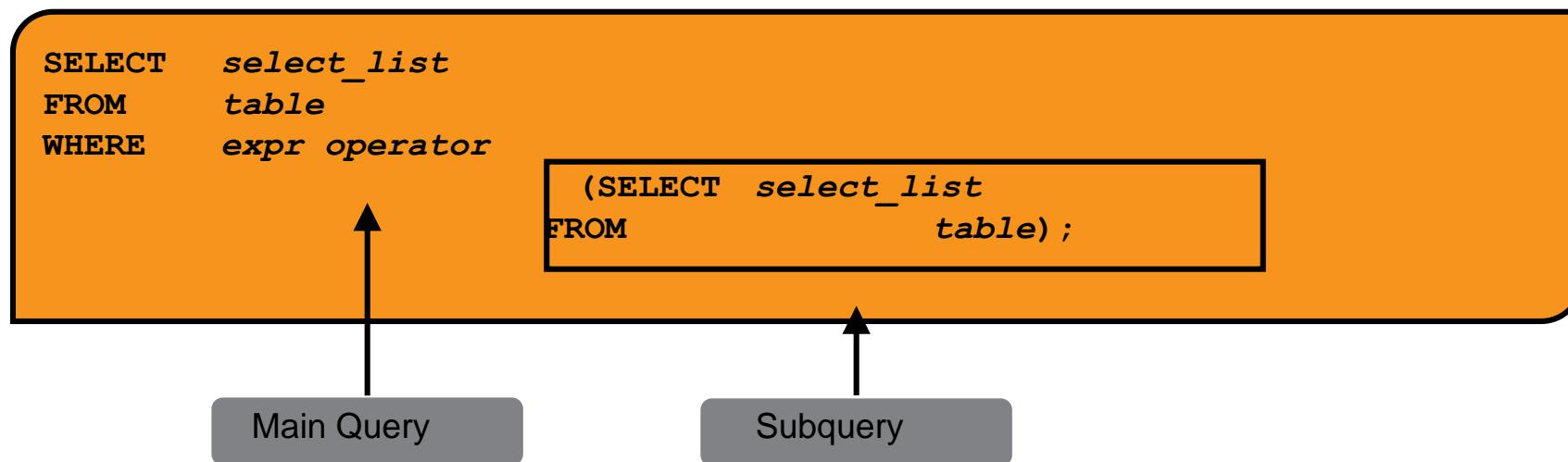
- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
  - Group functions in a subquery
  - HAVING clause with subqueries
- Multiple-row subqueries
  - Using ALL or ANY operator
- Multiple-column subqueries
- Null values in a subquery

# Using a Subquery to Solve a Problem



# Subquery Syntax

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



# Using a Subquery



## Main Query:

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



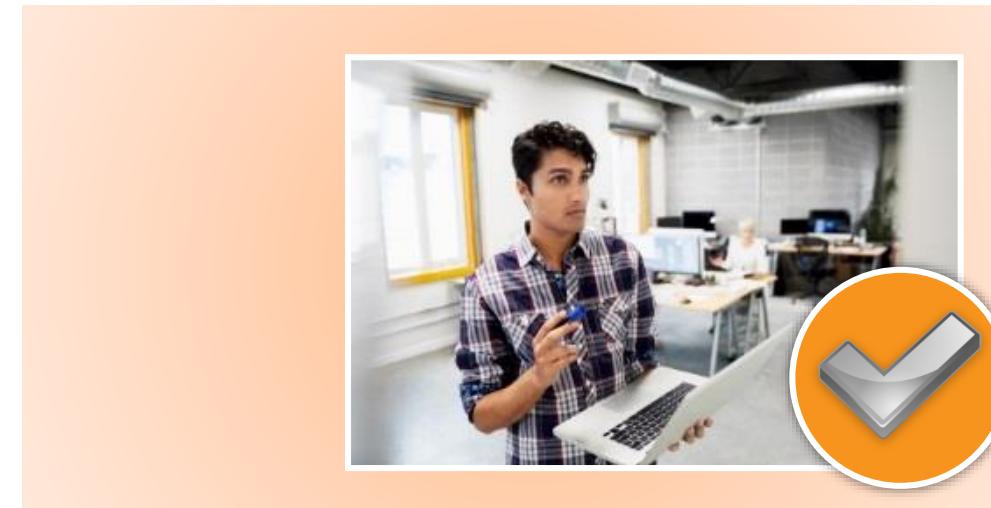
## Sub Query:

When was Davies hired?

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

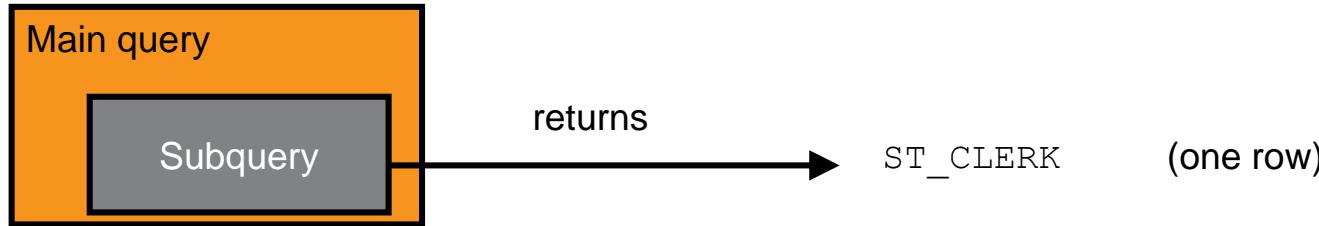
# Rules and Guidelines for Using Subqueries

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

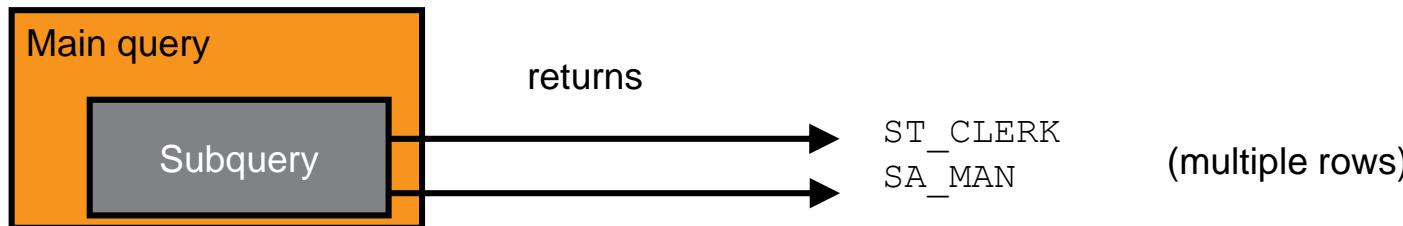


# Types of Subqueries

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

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

# Using Group Functions in a Subquery

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

2500

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

# HAVING Clause with Subqueries

The Oracle server:

- Executes the subqueries first
- Returns the result into the HAVING clause of the main query

```
SELECT      department_id, MIN(salary)
FROM        employees
GROUP BY    department_id
HAVING      MIN(salary) > 2500
           ←
           (SELECT MIN(salary)
            FROM   employees
            WHERE  department_id = 50);
```

	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	90	17000
3	20	6000
4	110	8300
5	80	8600
6	60	4200
7	10	4400

# What Is Wrong with This Statement?

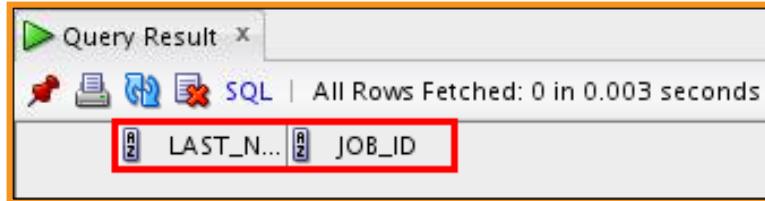
```
SELECT employee_id, last_name
FROM   employees
WHERE  salary = (SELECT MIN(salary)
                  FROM   employees
                  GROUP BY department_id);
```

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 jobs  
       WHERE job_title = 'Architect');
```



The subquery returns no rows because there is no job with the title "Architect."

# 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 =, !=, >, <, <=, >=. This returns TRUE if at least one element exists in the result set of the subquery for which the relation is TRUE.
ALL	Must be preceded by =, !=, >, <, <=, >=. This returns TRUE if the relation is TRUE for all elements in the result set of the subquery.

# Using the ANY Operator in Multiple-Row Subqueries

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

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
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
WHERE  salary < ALL
        9000, 6000, 4200
        (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.

Syntax:

```
SELECT column, column, ...
FROM table
WHERE (column1, column2, ...) IN
      (SELECT column1, column2, ...
       FROM table
       WHERE condition);
```

# Multiple-Column Subquery: Example

Display all the employees with the lowest salary in each department.

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

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

# 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);
```



The 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 subqueries can solve
- Identify the types of subqueries
- 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

# Set Operators

- Using Set Operators

# Course Roadmap

Lesson 1: Introduction

Unit 1: Retrieving, Restricting,  
and Sorting Data

**Unit 2: Joins, Subqueries, and  
Set Operators**

Unit 3: DML and DDL

▶ Lesson 6: Reporting Aggregated Data Using  
Group Functions

▶ Lesson 7: Displaying Data from Multiple  
Tables Using Joins

▶ Lesson 8: Using Subqueries to Solve Queries

▶ **Lesson 9: Using Set Operators**

You are here!

# Objectives

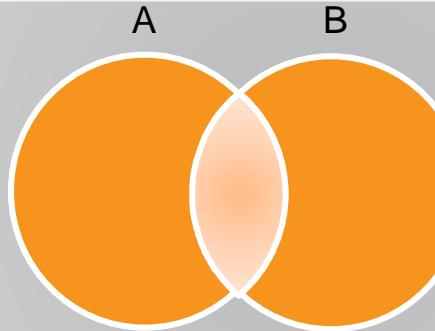
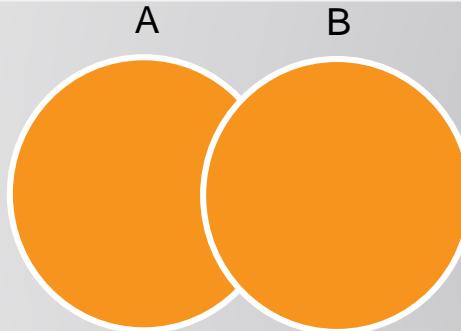
After completing this lesson, you should be able to do the following:

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

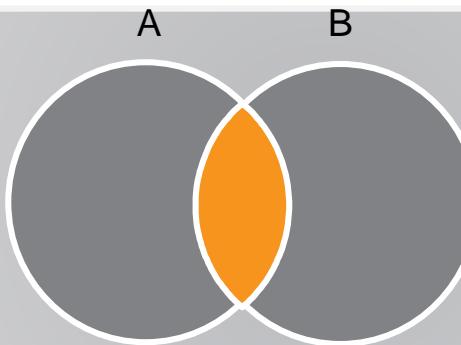
# Lesson Agenda

- 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

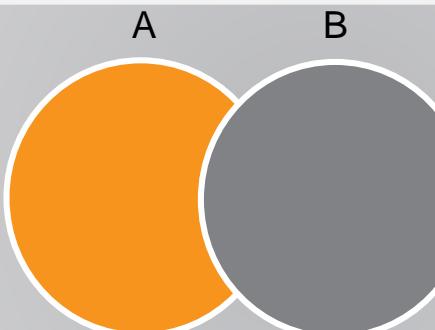
# Set Operators



UNION/UNION ALL



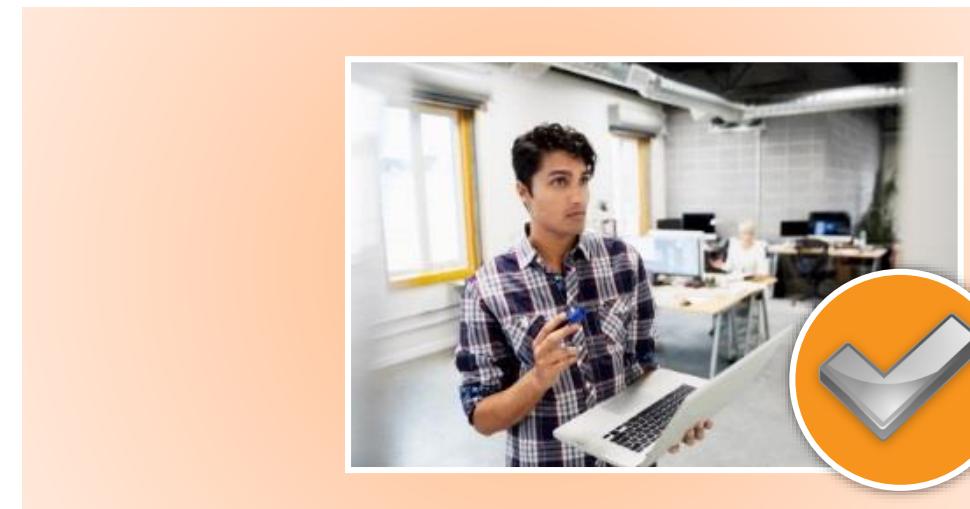
INTERSECT



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.
- The ORDER BY clause can appear only at the very end of the statement.



# Oracle Set 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:

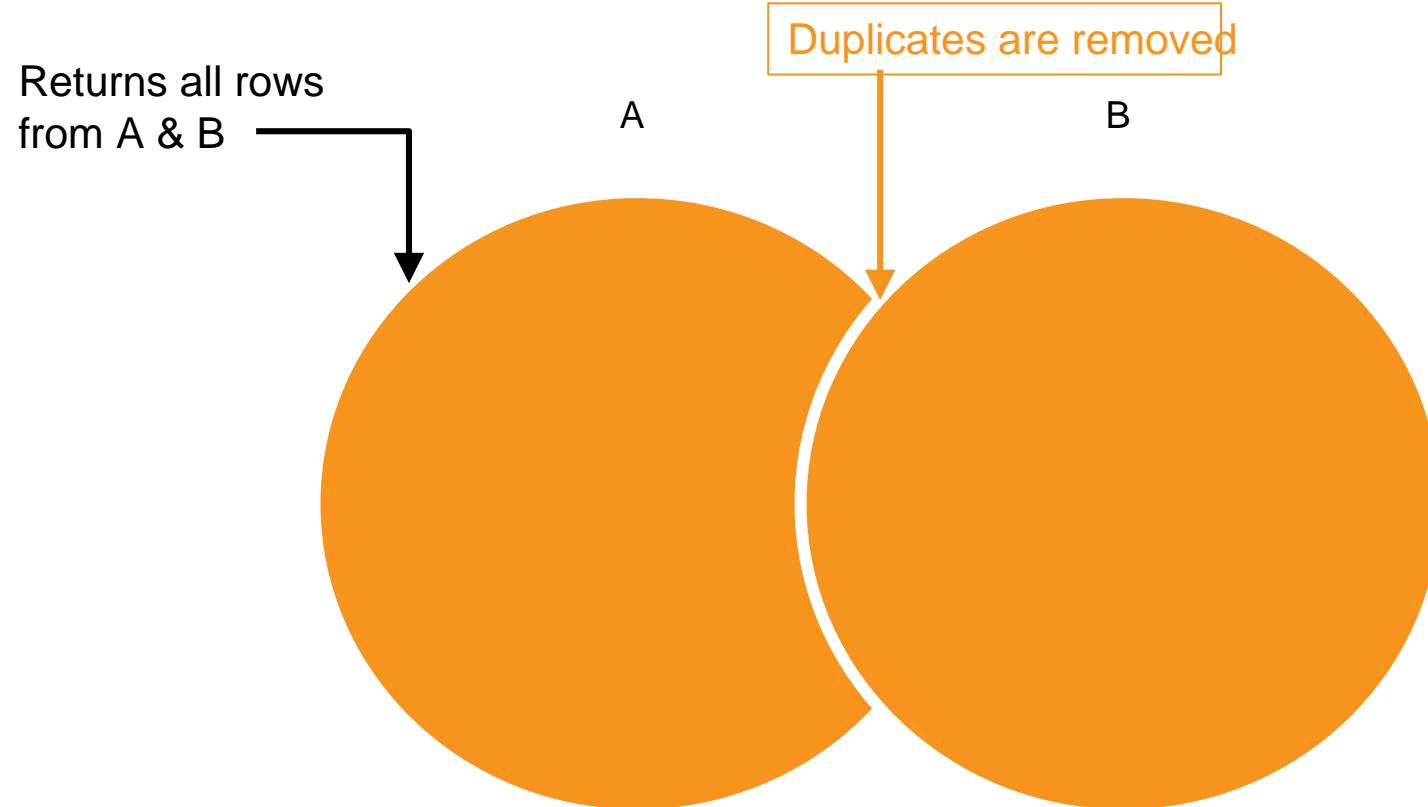
- EMPLOYEES: Provides details about all current employees
- RETIRED\_EMPLOYEES: Provides details about all past employees



# 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 duplicates.

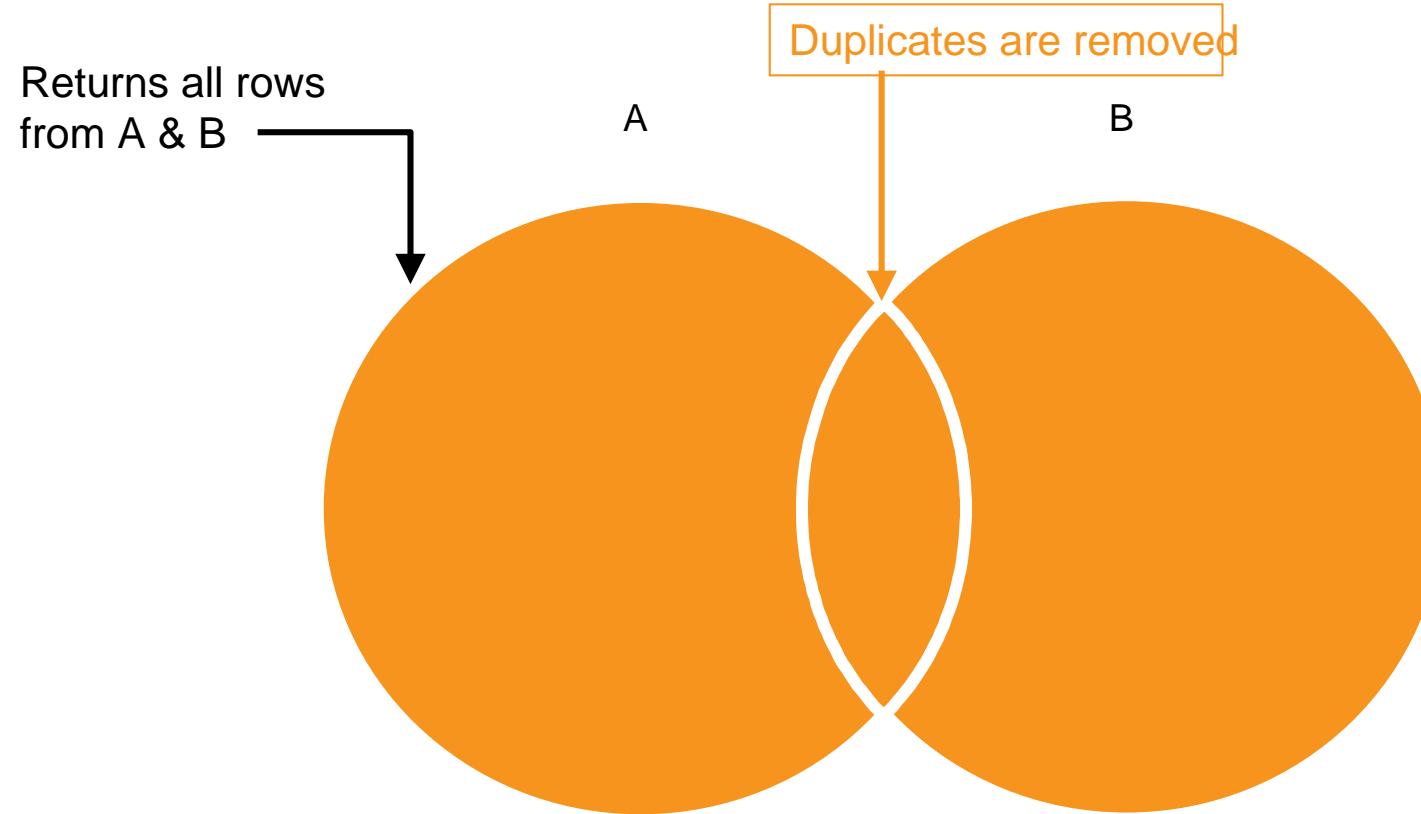
# Using the UNION Operator

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;
```

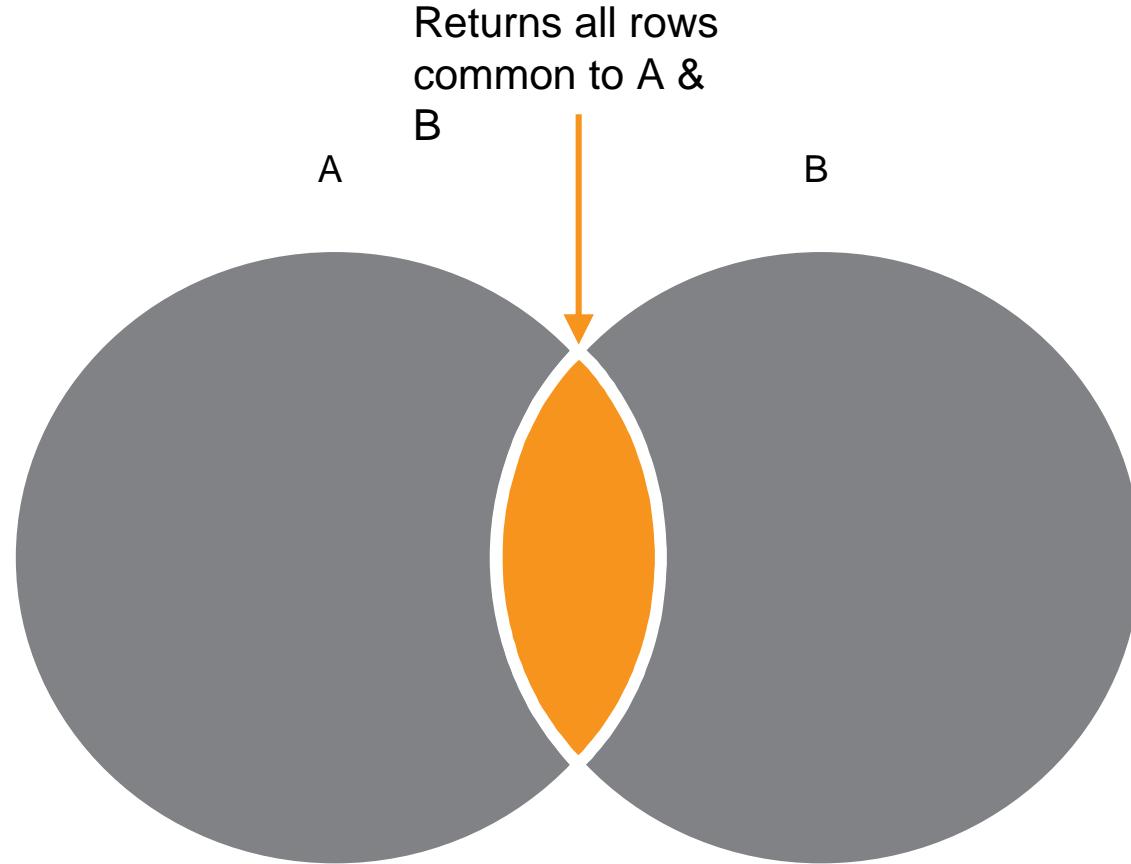
JOB_ID	DEPARTMENT_ID
1 AC_ACCOUNT	110
2 AC_MGR	110
3 AD_ASST	10
4 AD_PRES	90
5 AD_PRES	90
6 AD_VP	90
7 AD_VP	80
8 AD_VP	90
9 AD_VP	90
...	

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.

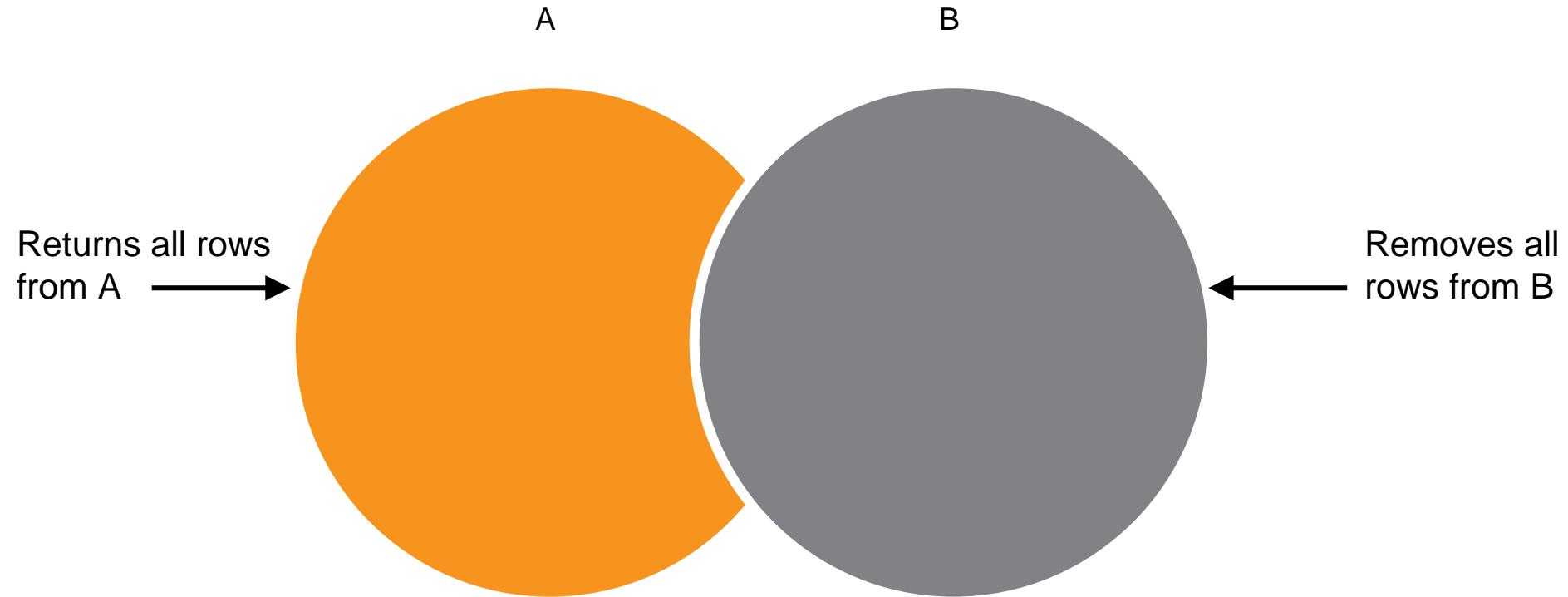
```
SELECT manager_id,department_id
FROM employees
INTERSECT
SELECT manager_id,department_id
FROM retired_employees
```

	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 manager IDs and Job IDs of employees whose managers have never managed retired employees in the Sales department.

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

MANAGER_ID	JOB_ID
1	100 SA_MAN
2	149 SA_REP

# 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, job ID, and hire date of all employees.

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

	FIRST_NAME	JOB_ID	HIRE_DATE
1	Alex	PU_CLERK	(null)
2	Alexander	IT_PROG	03-JAN-14
3	Alexandera	IT_PROG	(null)
4	Bruce	IT_PROG	21-MAY-15
5	Bruk	IT_PROG	(null)
6	Curtis	ST_CLERK	29-JAN-13
7	Dany	FI_ACCOUNT	(null)
8	Del	PU_MAN	(null)

...

# 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 ascending order.

# Quiz

Identify two set operator guidelines.

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

# 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

# DML

- Managing Tables Using DML Statements

# Course Roadmap

Lesson 1: Introduction

Unit 1: Retrieving, Restricting,  
and Sorting Data

Unit 2: Joins, Subqueries, and  
Set Operators

**Unit 3: DML and DDL**

▶ **Lesson 10: Managing Tables Using DML  
Statements**

▶ Lesson 11: Introduction to Data Definition  
Language

← You are here!

# Objectives

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

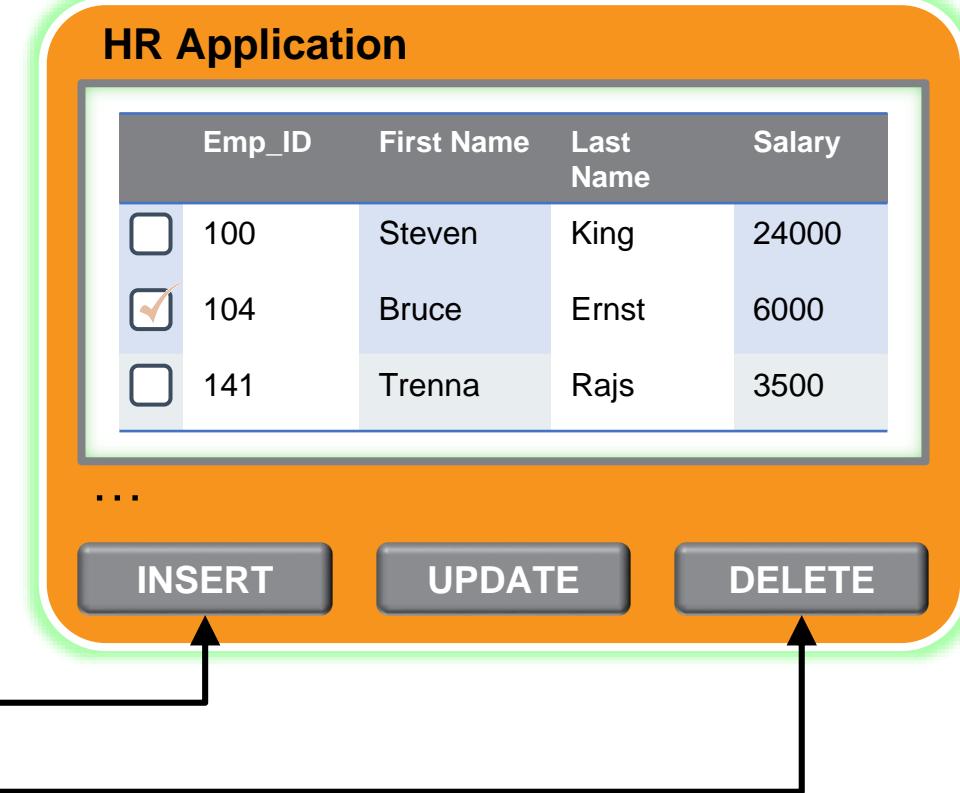
- Describe each data manipulation language (DML) statement
- Control transactions

# HR Application Scenario

It is time for me to update the employee directory! Let me first delete the employees who have quit and insert new hires.



Clicks INSERT and enters values for the new employee.



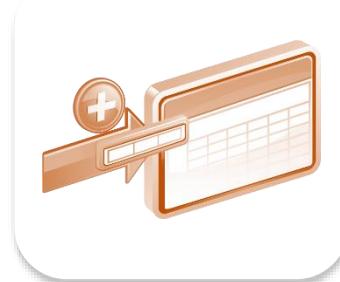
Selects a record and clicks DELETE to delete an employee.

# Lesson Agenda

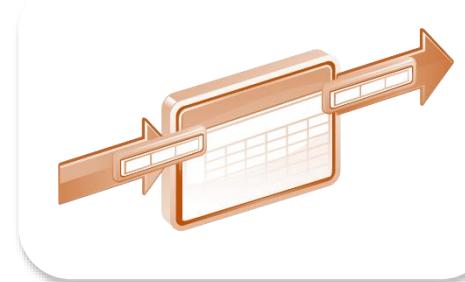
- Adding new rows in a table
  - INSERT statement
- Changing data in a table
  - UPDATE statement
- Removing rows from a table:
  - DELETE statement
  - TRUNCATE statement
- Database transaction control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- Manual Data Locking
  - FOR UPDATE clause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

# Data Manipulation Language

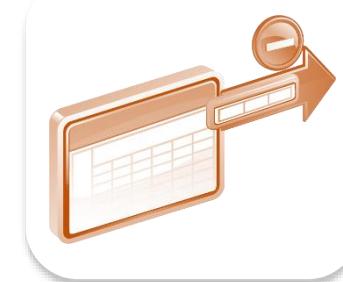
- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.



Insert



Update



Delete

# 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 a new row  
into the  
DEPARTMENTS table.

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

# 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 row 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 row inserted.

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

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

1 row 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',
       SYSDATE, 'AC_ACCOUNT', 6900,
       NULL, 205, 110);
```

1 row 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, 2016', 'MON DD, YYYY'),
              'SA REP', 11000, 0.2, 100, 60);
```

1 row inserted.

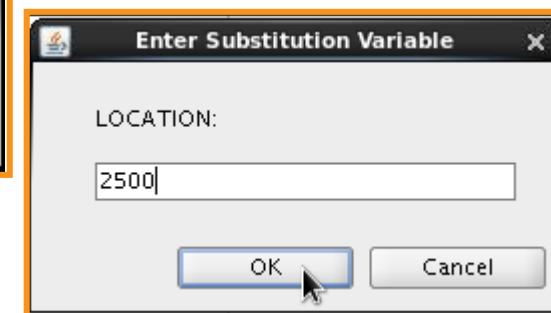
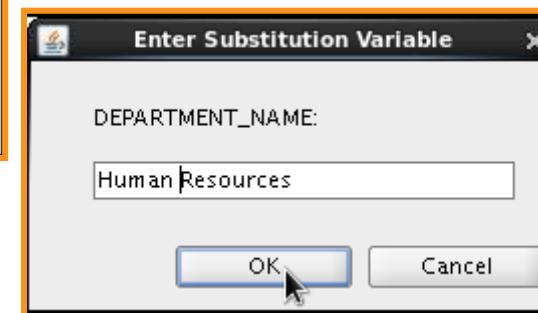
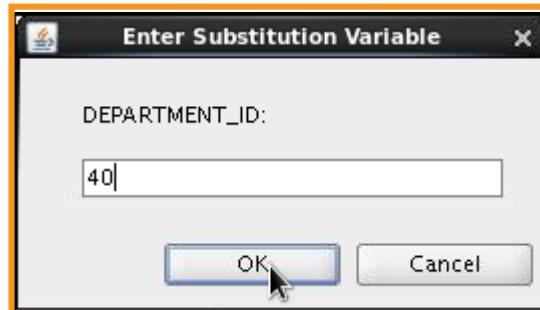
- Verify your addition.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-16	SA REP	11000	0.2	100	60

# 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

- 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
- Manual Data Locking
  - FOR UPDATE clause in a SELECT statement
  - LOCK TABLE 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 row updated.

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

```
UPDATE copy_emp  
SET department_id = 110;  
22 rows updated
```

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

# Updating Two Columns with a Subquery

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

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

1 row updated.

# 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)
      job_id          = (SELECT job_id
                        FROM employees
                        WHERE employee_id = 200);

1 row 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
- Manual Data Locking
  - FOR UPDATE clause in a SELECT statement
  - LOCK TABLE 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 row 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 row 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 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
- Manual Data Locking
  - FOR UPDATE clause in a SELECT statement
  - LOCK TABLE statement

# Database Transactions

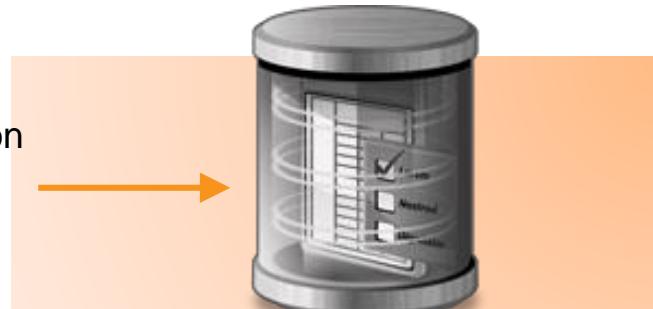
A database transaction consists of one of the following:

- 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.

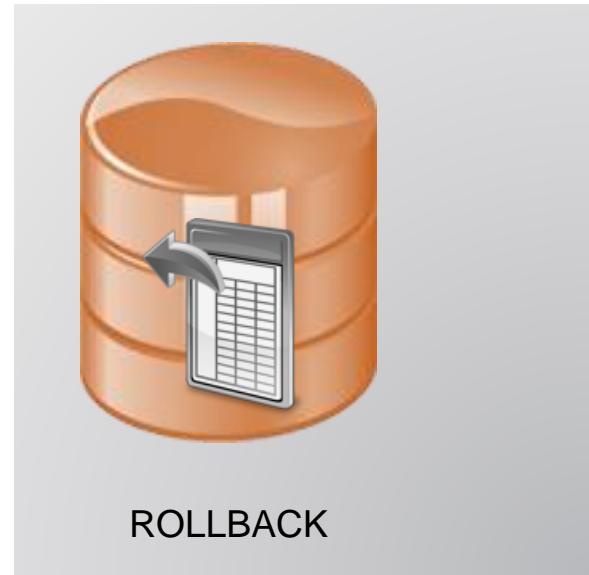
Database transaction on  
table/s



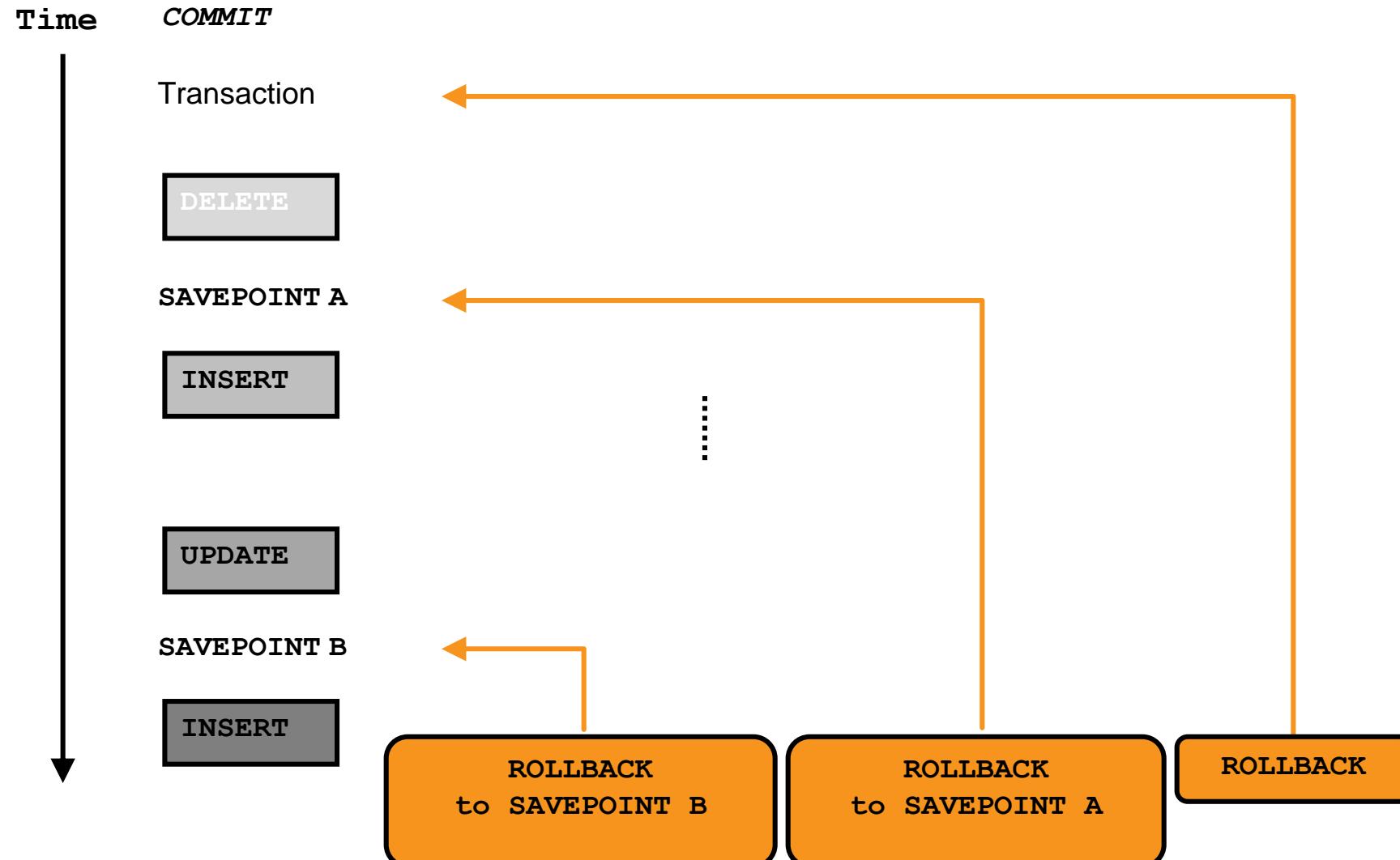
# Advantages of COMMIT and ROLLBACK Statements

Using COMMIT and ROLLBACK statements, you can:

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

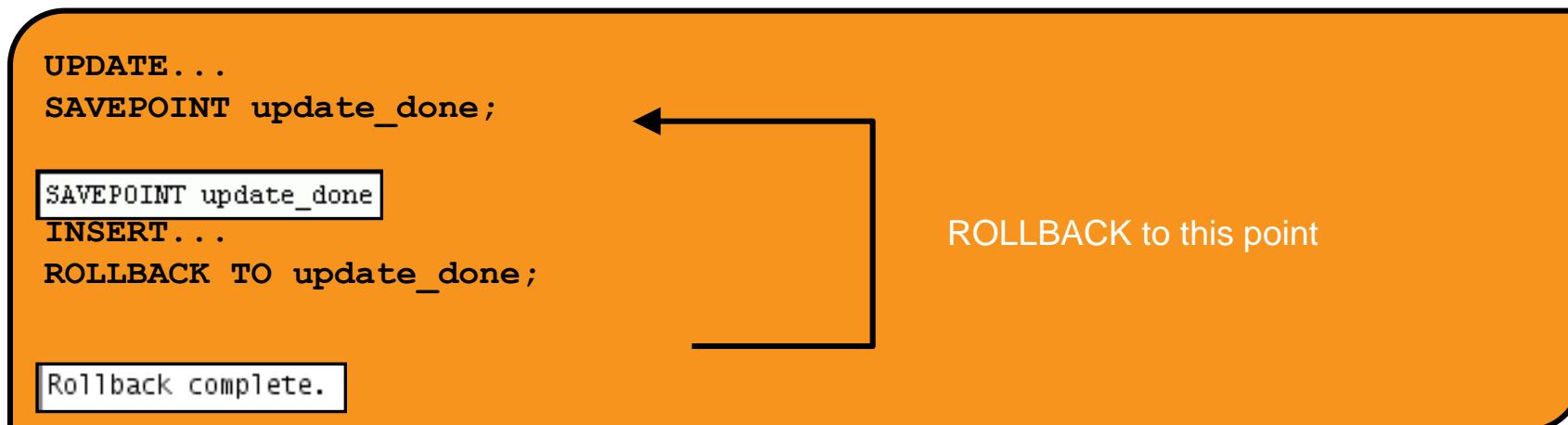


# Explicit Transaction Control Statements



# 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.

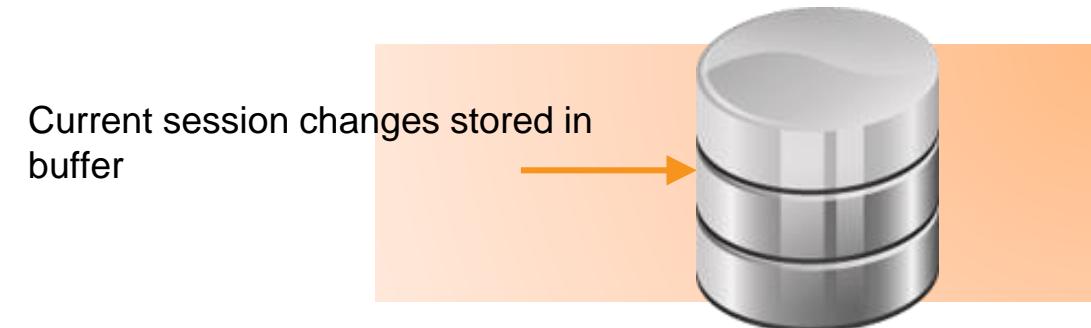


# Implicit Transaction Processing

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

# State of Data Before COMMIT or ROLLBACK

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



# 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 row deleted.  
  
INSERT INTO departments  
VALUES (290, 'Corporate Tax', NULL, 1700);
```

1 row inserted.

COMMIT;

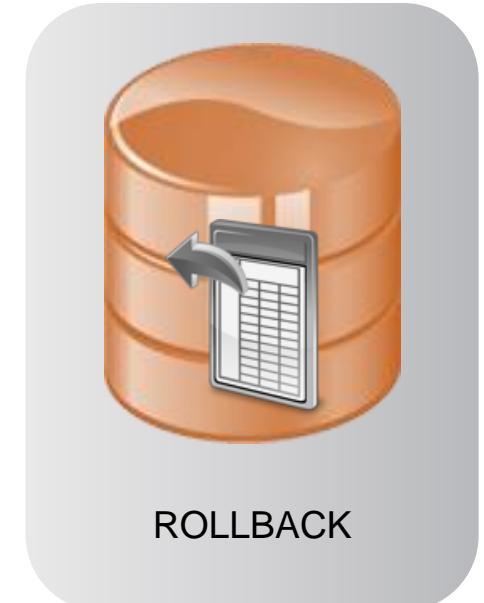
Commit complete.

# 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
- Manual Data Locking
  - FOR UPDATE clause in a SELECT statement
  - LOCK TABLE statement

# Read Consistency

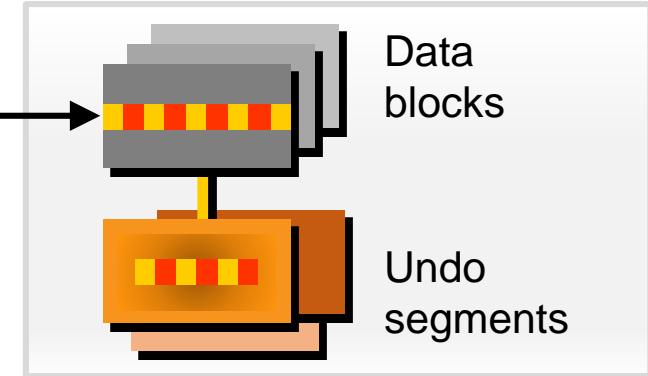
- Read consistency guarantees a consistent view of data at all times.
- Changes made by one user do not conflict with the changes made by another user.
- Read consistency ensures that, on the same data:
  - Readers do not wait for writers
  - Writers do not wait for readers
  - Writers wait for writers

# Implementing Read Consistency

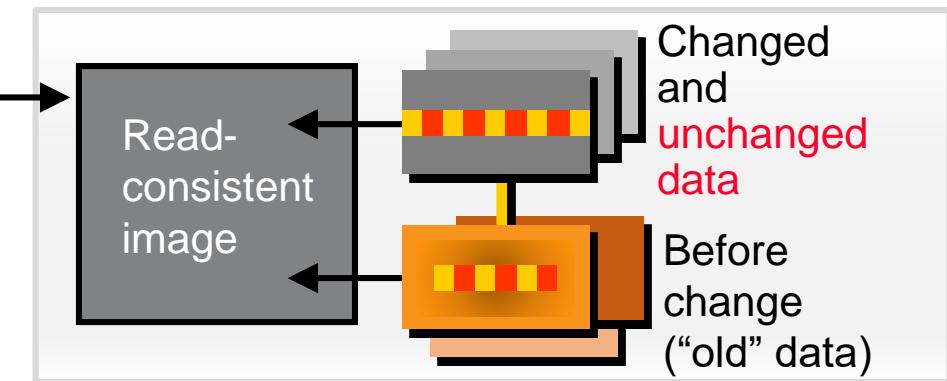
User A



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



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



User B

# Lesson Agenda

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

# 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 that you intend to change; then only the rows from that specific table are locked.

# LOCK TABLE Statement

- Use the LOCK TABLE statement to lock one or more tables in a specified mode.
- This manually overrides automatic locking.
- Tables are locked until you COMMIT or ROLLBACK.

```
LOCK TABLE table_name
IN [ROW SHARE/ROW EXCLUSIVE/SHARE UPDATE/SHARE/
    SHARE ROW EXCLUSIVE/ EXCLUSIVE] MODE
[NOWAIT];
```

# 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

# DDL

- Introduction to Data Definition Language

# Course Roadmap

Lesson 1: Introduction

Unit 1: Retrieving, Restricting,  
and Sorting Data

Unit 2: Joins, Subqueries, and  
Set Operators

**Unit 3: DML and DDL**

▶ Lesson 10: Managing Tables Using DML  
Statements

▶ Lesson 11: Introduction to Data Definition  
Language

← You are here!

# Objectives

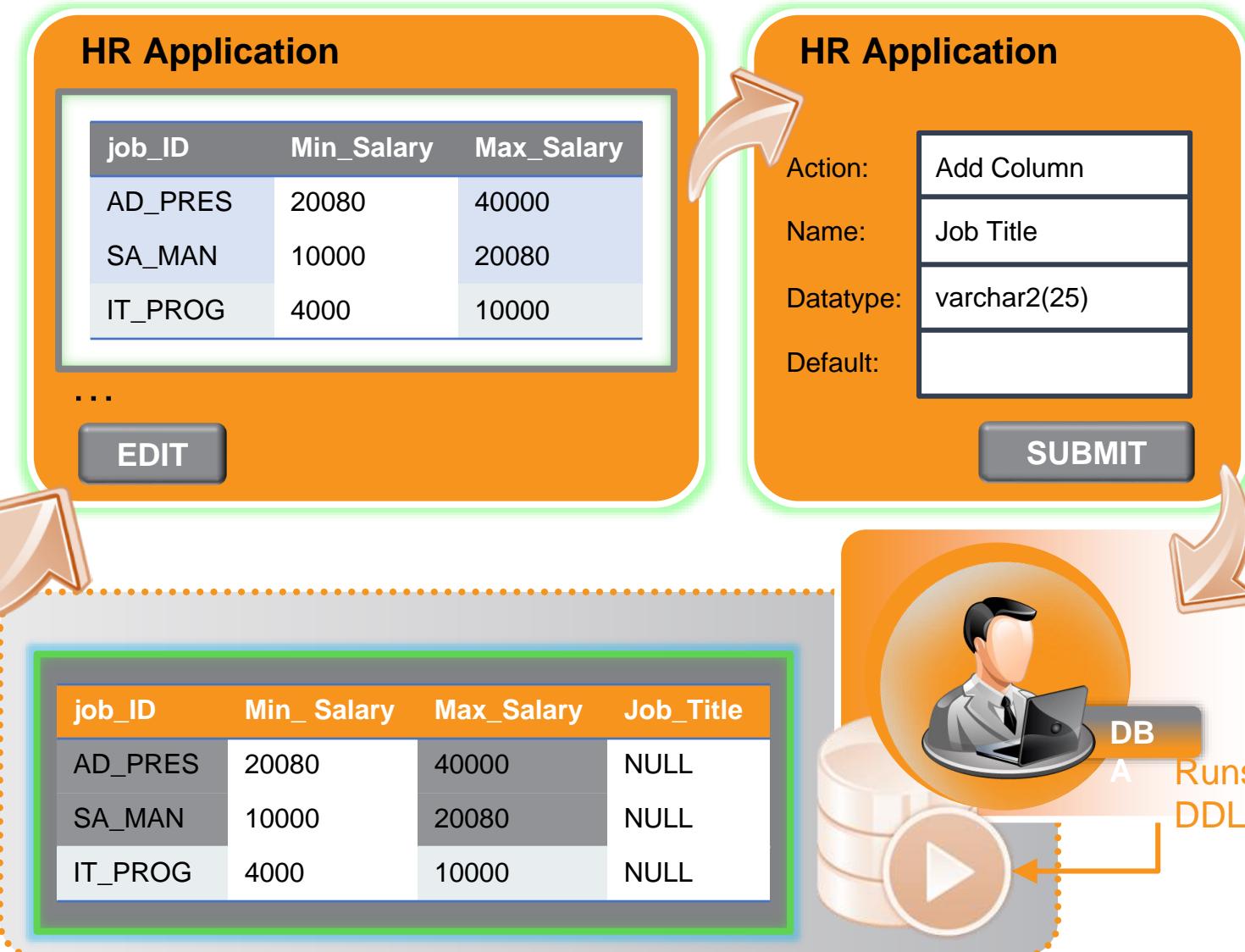
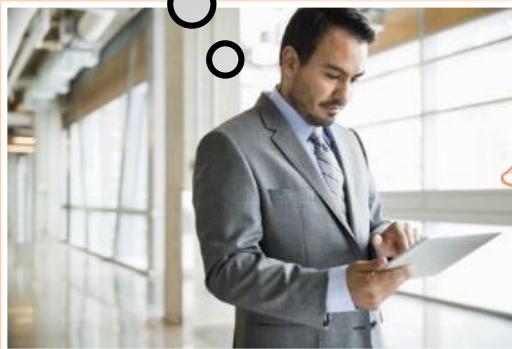
After completing this lesson, you should be able to do the following:

- 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

# HR Application Scenario

The JOBS table looks fine except it does not contain a column for JOB\_TITLE. What should I do now?

Bob



# 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

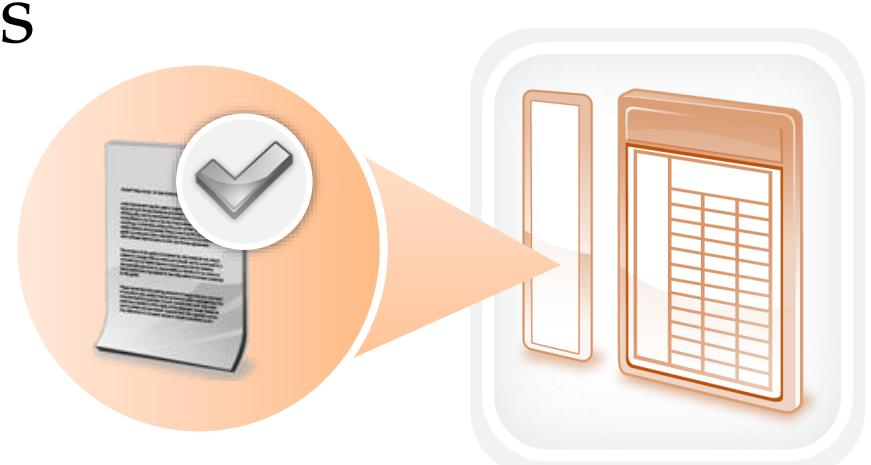
Object	Description
Table	Is the basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative name to an object



# Naming Rules for Tables and Columns

Ensure that the table names and column names:

- Begin with a letter
- Are 1–30 characters long
- Contain only A-Z, a-z, 0-9, \_, \$, and #
- Do *not* duplicate the name of another object owned by the same user
- Are *not* Oracle server-reserved words



# 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
```

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
<b>VARCHAR2 (size)</b>	Variable-length character data
<b>CHAR (size)</b>	Fixed-length character data
<b>NUMBER (p, s)</b>	Variable-length numeric data
<b>DATE</b>	Date and time values
<b>LONG</b>	Variable-length character data (up to 2 GB)
<b>CLOB</b>	Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE).
<b>RAW and LONG RAW</b>	Raw binary data
<b>BLOB</b>	Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)).
<b>BFILE</b>	Binary data stored in an external file (up to 4 GB)
<b>ROWID</b>	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
<b>TIMESTAMP</b>	Date with fractional seconds
<b>INTERVAL YEAR TO MONTH</b>	Stored as an interval of years and months
<b>INTERVAL DAY TO SECOND</b>	Stored as an interval of days, hours, minutes, and seconds

# DEFAULT Option

- Specify a default value for a column in the CREATE TABLE statement.

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

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

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

```
Table HIRE_DATES created.
```

# 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 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 time of table creation
  - After the creation of the table
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.

# 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

- 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:

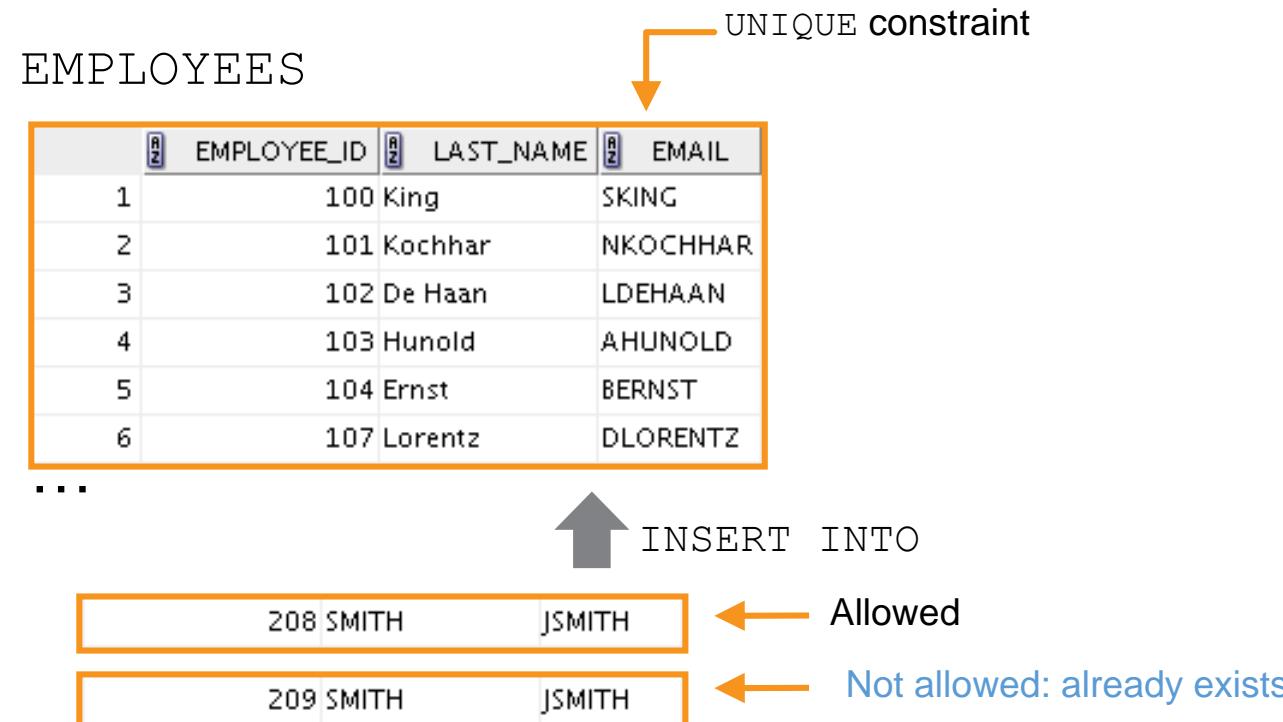
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	24000	(null)	90	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	17000	(null)	90	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	17000	(null)	90	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	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

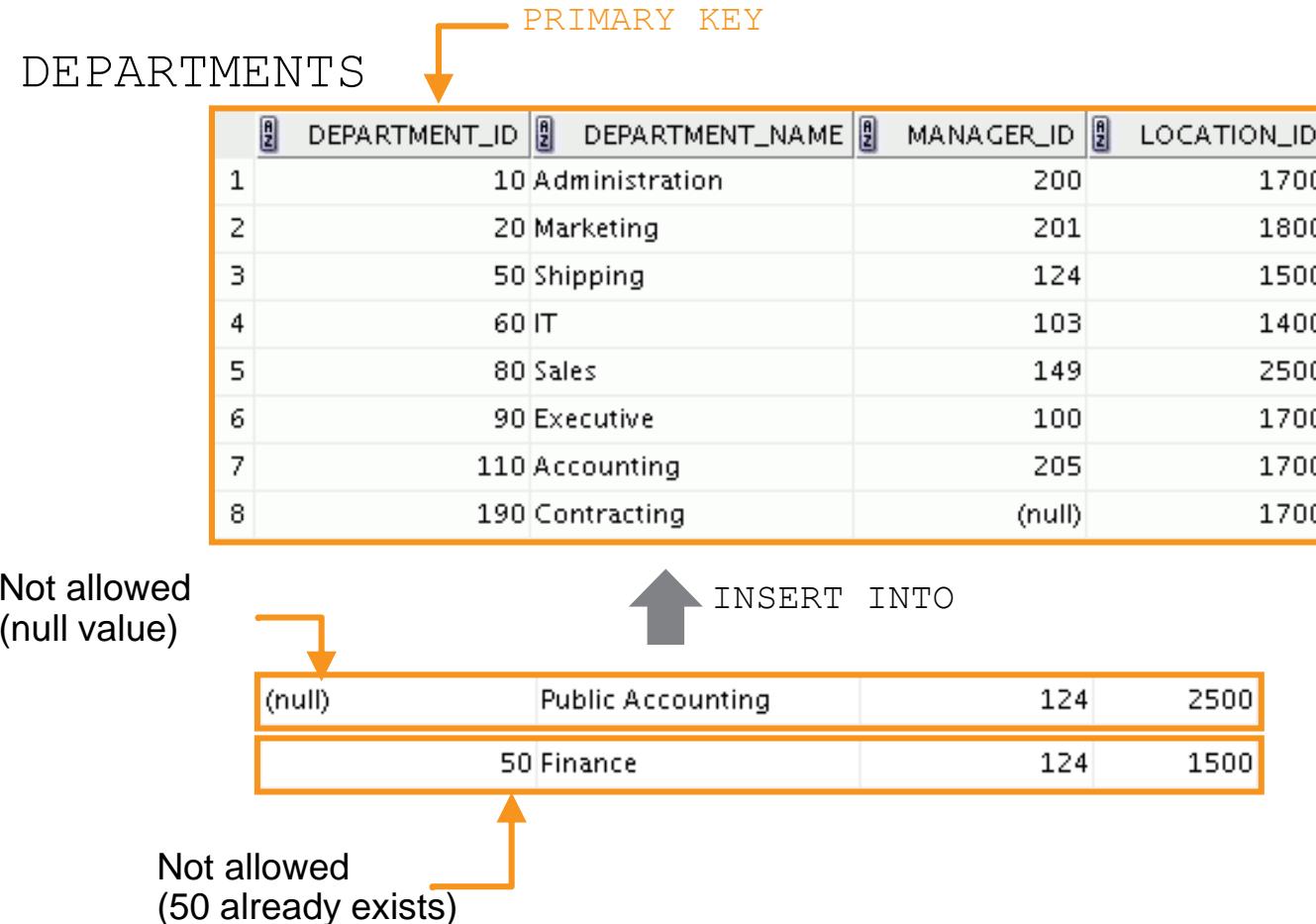


# UNIQUE Constraint

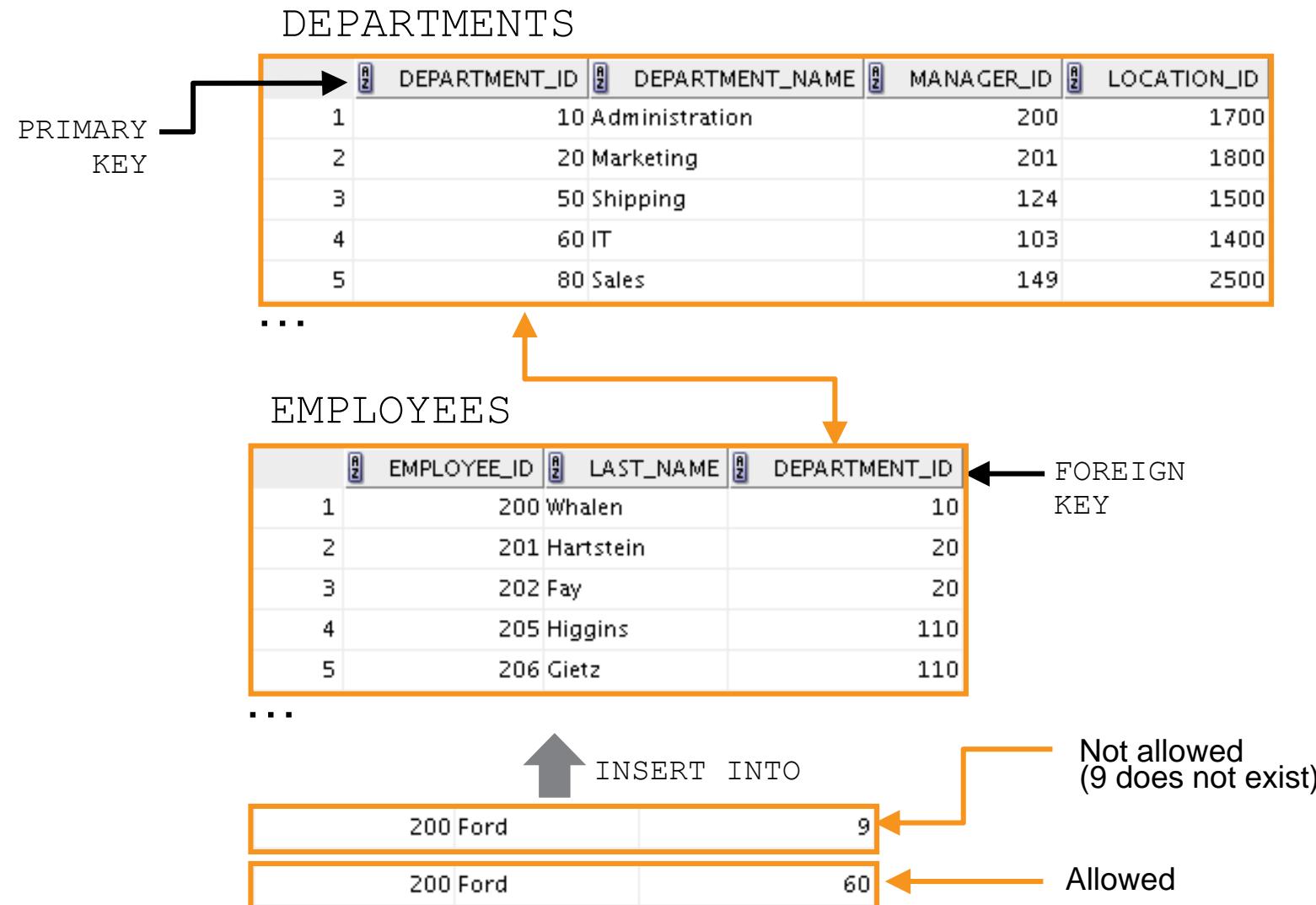
Define at either the table level or the column level:

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

# PRIMARY KEY Constraint



# FOREIGN KEY Constraint



# FOREIGN KEY Constraint

Define at either the table level or the column level:

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

# 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

- Defines a condition that each row must satisfy
- 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 (TEACH_A.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 (TEACH_A.EMP_DEPT_FK) violated - child record found  
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"  
*Cause:    attempted to delete a parent key value that had a foreign  
            dependency.  
*Action:   delete dependencies first then parent or disable constraint.
```

# 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

- 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

```
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	126000	29-JAN-16	(null)
2	174	Abel	132000	11-MAY-12	(null)
3	176	Taylor	103200	24-MAR-14	(null)
4	206	Gietz	99600	07-JUN-10	(null)



# Modifying a Column

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

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

```
Table DEPT80 altered.
```

Size of the last\_name column is modified.

- A change to the default value of a column affects only subsequent insertions to the table



# Dropping a Column

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

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

```
Table DEPT80 altered.
```

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	149	Zlotkey	126000	29-JAN-16
2	174	Abel	132000	11-MAY-12
3	176	Taylor	103200	24-MAR-14
4	206	Gietz	99600	07-JUN-10



# 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.

# SET UNUSED Option

```
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

Identify three actions that you perform by using constraints.

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

# 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

# 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

# Cloud

- Oracle Cloud Overview

# Lesson Objectives

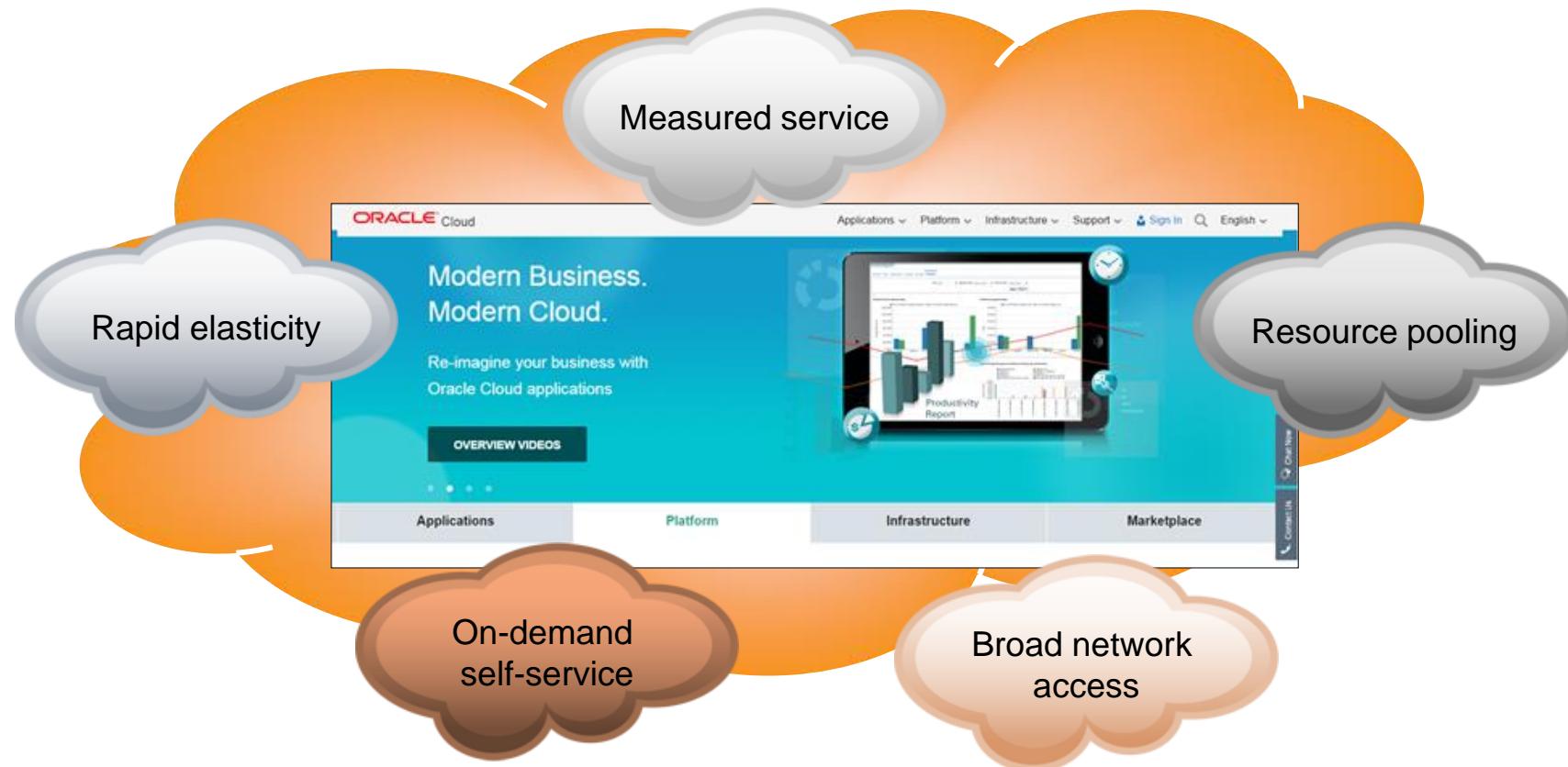
- After completing this lesson, you should be able to do the following:
  - Describe the salient features of Oracle Cloud
  - Discuss the features of Oracle Database Exadata Express Cloud Service

# Lesson Agenda

- Overview of Oracle Cloud
- Working with Oracle Database Exadata Express Cloud Service

# Introduction to Oracle Cloud

- Any business can now use the enterprise cloud provided by Oracle.
- You can access the Oracle Cloud from [cloud.oracle.com](http://cloud.oracle.com).



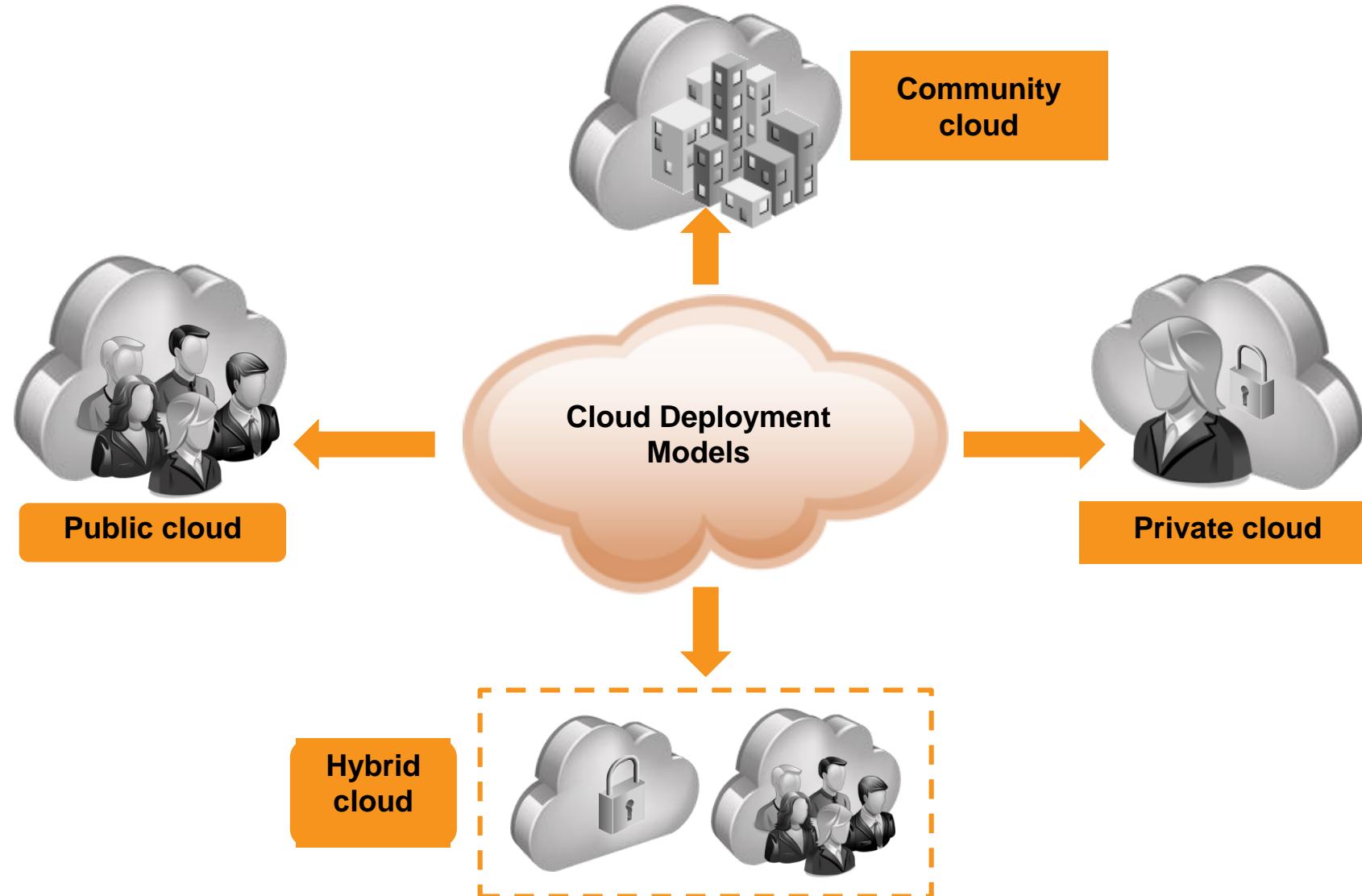
# Oracle Cloud Services

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

- Overview of Oracle Cloud
- Working with Oracle Database Exadata Express Cloud Service

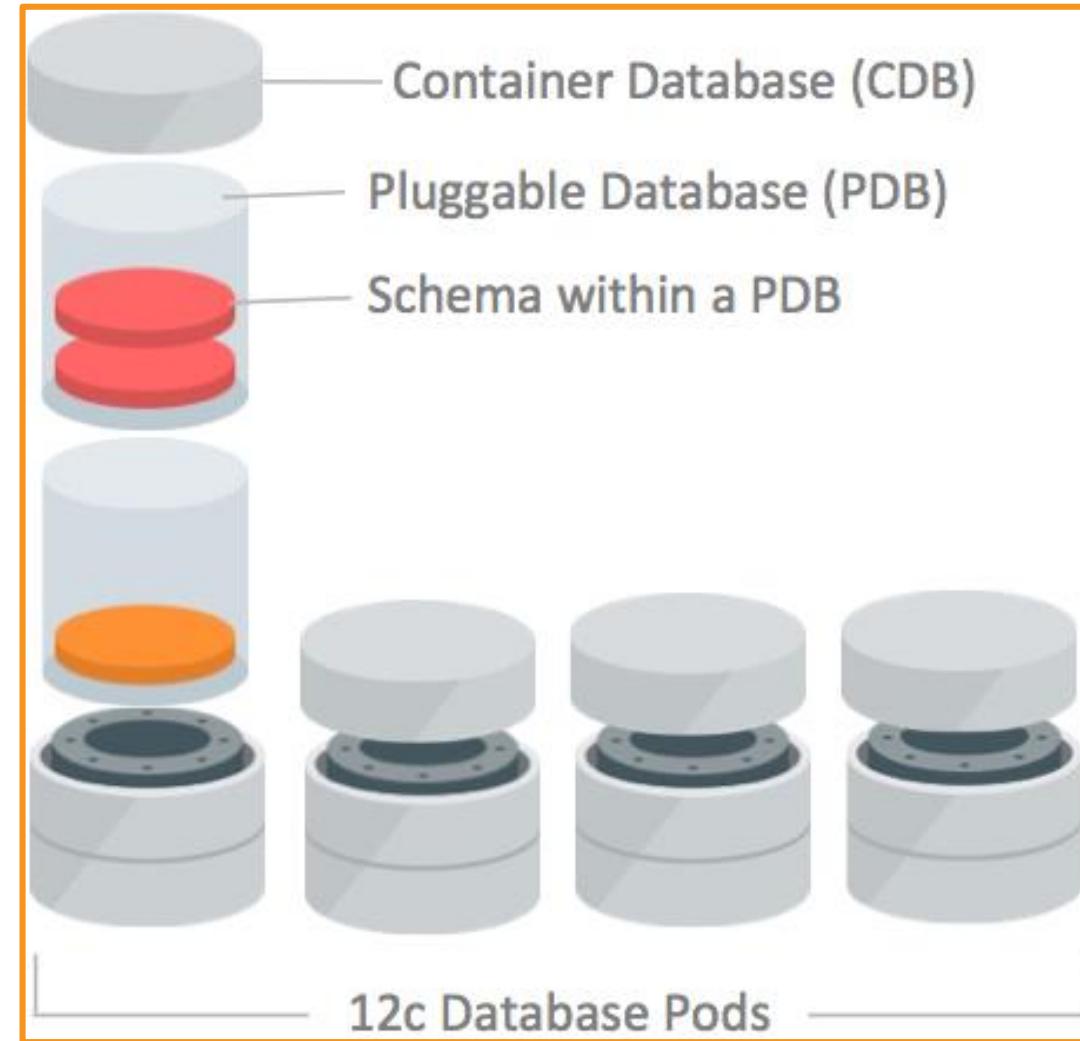
# What is in Exadata Express?

- A fully managed database service
- Provides powerful yet elastic database cloud service for developers
- Provides on-demand access to a shared pool of database resources
- Comes with built-in tools for rapid application development
  - APEX for web application development
  - Compatibility with clients such as SQL Developer, SQLcl

# Exadata Express for Users

- Oracle manages the service as multiple Container databases(CDBs), also known as database pods
- Each CDB can accommodate upto 1000 Pluggable databases(PDBs).
- Each user is provisioned with a PDB on subscribing to the service, where the user can create several schemas.

# Exadata Express for Users



# Exadata Express for Developers

- Developers can connect with a wide range of data sources for their applications
    - JSON Document Storage
    - Document Style data access
    - Oracle Rest Data Services



# Getting Started with Exadata Express

1. Purchase a subscription.
2. Activate and verify the service.
3. Verify activation.
4. Learn about users and roles.
5. Create accounts for your users and assign them appropriate privileges and roles.
6. Set the password for the database user authorized to perform administrative tasks for your service (PDB\_ADMIN).

# Getting Started with Exadata Express

- **Note:** You can refer to Using Oracle Database Exadata Express Cloud Service (<https://docs.oracle.com/cloud/latest/exadataexpress-cloud/CSDBP/toc.htm>) for details on the subscription process.

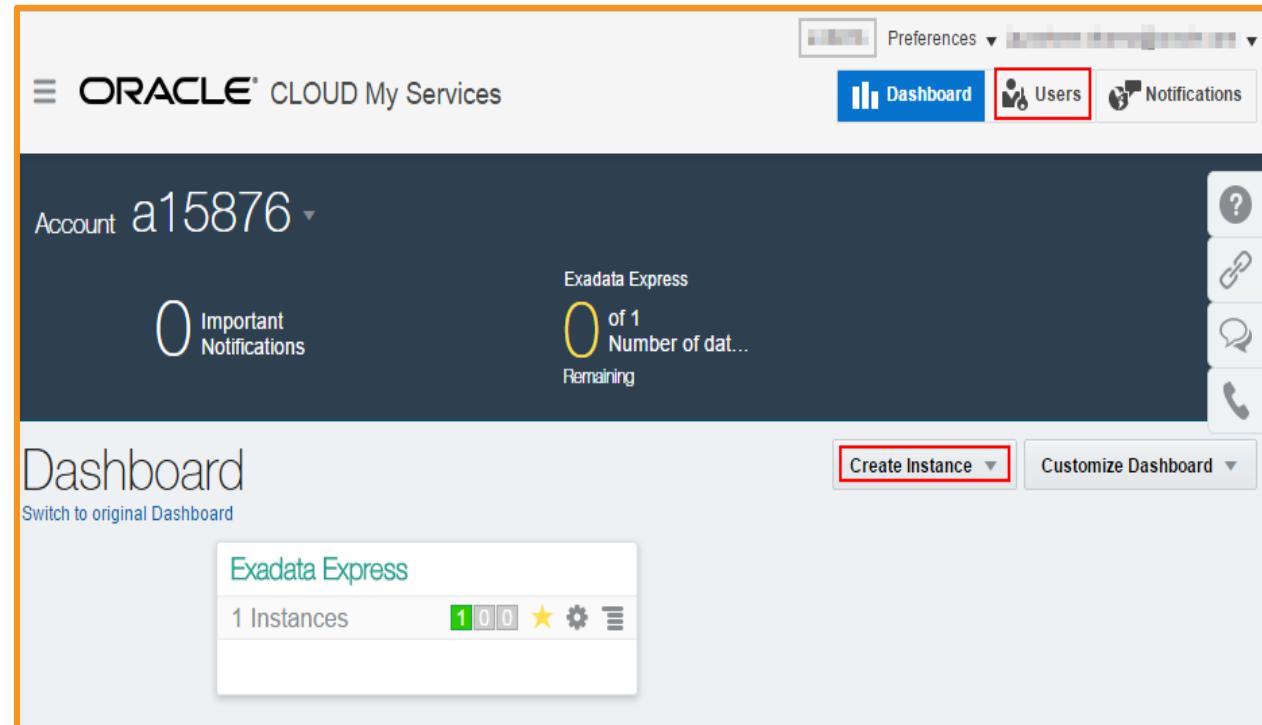
# Oracle Exadata Express Cloud Service

You can refer to Working with Oracle Database Exadata Express Cloud Service

(<http://oukc.oracle.com/public/redir.html?type=player&offid=1984115860>) to gain an introduction to the service and its features.

# Getting Started with Exadata Express

- On signing into the service, you get access to the dashboard.
- Dashboard allows you to create database instances and users.
- The number of instances you create is limited by the amount of resources you have access to.



# Managing Exadata

**Service Instances**

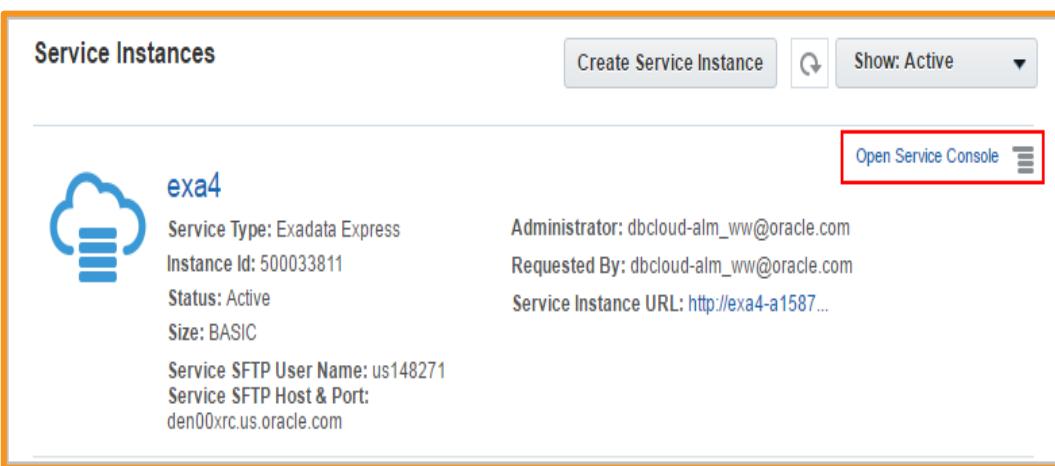
Create Service Instance Show: Active ▾

**exa4**

Service Type: Exadata Express  
Instance Id: 500033811  
Status: Active  
Size: BASIC  
Service SFTP User Name: us148271  
Service SFTP Host & Port: den00xrc.us.oracle.com

Administrator: dbcloud-alm\_www@oracle.com  
Requested By: dbcloud-alm\_www@oracle.com  
Service Instance URL: <http://exa4-a1587...>

Open Service Console




**Service Console: exa4**

 <b>Web Access</b> Develop database and web apps using Oracle Application Express (APEX) <a href="#">Learn More</a> <a href="#">Watch Video</a>	<a href="#"><b>Go to SQL Workshop</b></a> Run SQL commands, execute SQL scripts and browse database objects	<a href="#"><b>Develop with App Builder</b></a> Declaratively develop and deploy data-driven apps
 <b>Client Access</b> Enable database client access, then connect using drivers and tools <a href="#">Learn More</a> <a href="#">Watch Video</a>	<a href="#"><b>Define REST Data Services</b></a> Create and manage RESTful web service interfaces to your database	<a href="#"><b>Install Productivity Apps</b></a> Browse and install productivity apps
 <b>Administration</b> Manage your cloud database <a href="#">Learn More</a> <a href="#">Watch Video</a>	<a href="#"><b>Download Client Credentials</b></a> Download a zip file containing your security credentials and network configuration files	<a href="#"><b>Download Drivers</b></a> Get database drivers for Java, .NET, Node.js, Python, PHP, Ruby, C, C++, Instant Client and more
 <b>Create Database Schema</b> Create a new schema for database objects	<a href="#"><b>Disable Client Access</b></a> Disable SQL*Net access and invalidate all existing client credential files	<a href="#"><b>Download Tools</b></a> Get SQL*Plus command-line and developer tools including SQL Developer and JDeveloper
 <b>Create Document Store</b> Enable or disable a schema-less document-style interface, with JSON storage and access	 <b>Set Administrator Password</b> Set or reset your database's privileged user (PDB_ADMIN) account password	<a href="#"><b>Manage Application Express</b></a> Use Application Express (APEX) administrative options

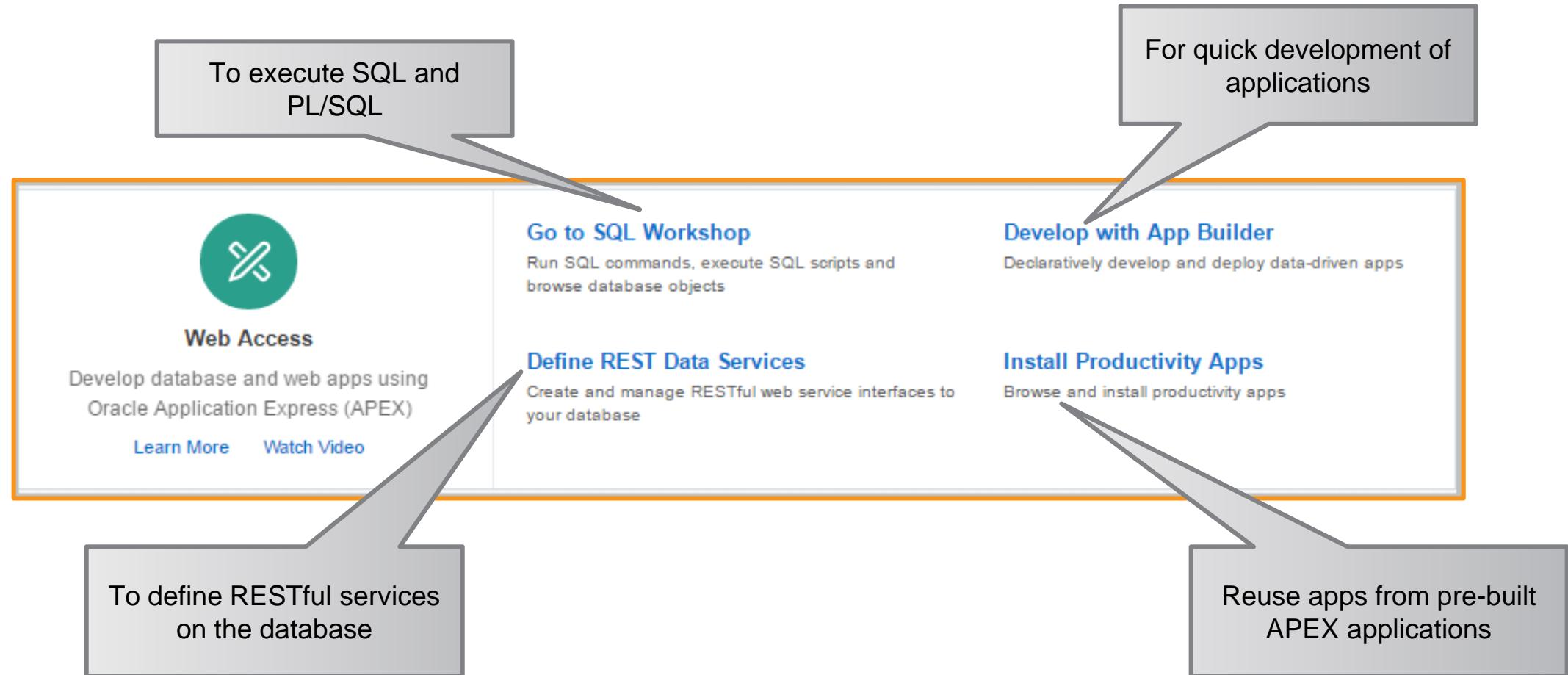
# Service Console

- Service Console is the interface to use and manage the Exadata service
- It provides three different perspectives of the instance
  - Web Access
  - Client Access
  - Administration

The screenshot shows the Service Console interface for an instance named 'exa4'. The interface is divided into three main sections: Web Access, Client Access, and Administration.

- Web Access:** This section includes a 'Web Access' icon, a brief description of using Oracle Application Express (APEX) to develop database and web apps, and links to 'Learn More' and 'Watch Video'.
- Client Access:** This section includes a 'Client Access' icon, a brief description of enabling database client access, and links to 'Learn More' and 'Watch Video'. It also includes links to 'Download Client Credentials' and 'Disable Client Access'.
- Administration:** This section includes an 'Administration' icon, a brief description of managing the cloud database, and links to 'Learn More' and 'Watch Video'. It also includes links to 'Create Database Schema', 'Set Administrator Password', and 'Manage Application Express'.
- Additional Options:** To the right of the main sections, there are links to 'Go to SQL Workshop', 'Define REST Data Services', 'Install Productivity Apps', 'Download Drivers', 'Download Tools', 'Create Document Store', and 'Manage Application Express'.

# Web Access through Service Console



# Client Access Configuration through Service Console



### Client Access

Enable database client access, then connect using drivers and tools

[Learn More](#) [Watch Video](#)

**Enable Client Access**  
Enable client SQL\*Net access to your database. You must first enable before seeing other options





### Client Access

Enable database client access, then connect using drivers and tools

[Learn More](#) [Watch Video](#)

**Download Client Credentials**  
Download a zip file containing your security credentials and network configuration files

**Disable Client Access**  
Disable SQL\*Net access and invalidate all existing client credential files

**Download Drivers**  
Get database drivers for Java, .NET, Node.js, Python, PHP, Ruby, C, C++, Instant Client and more

**Download Tools**  
Get SQL\*Plus command-line and developer tools including SQL Developer and JDeveloper

# Database Administration through Service Console

The screenshot shows the Oracle Database Service Console interface. On the left, there's a navigation bar with a gear icon labeled "Administration" and the subtext "Manage your cloud database". Below this are "Learn More" and "Watch Video" links. The main content area has three main sections:

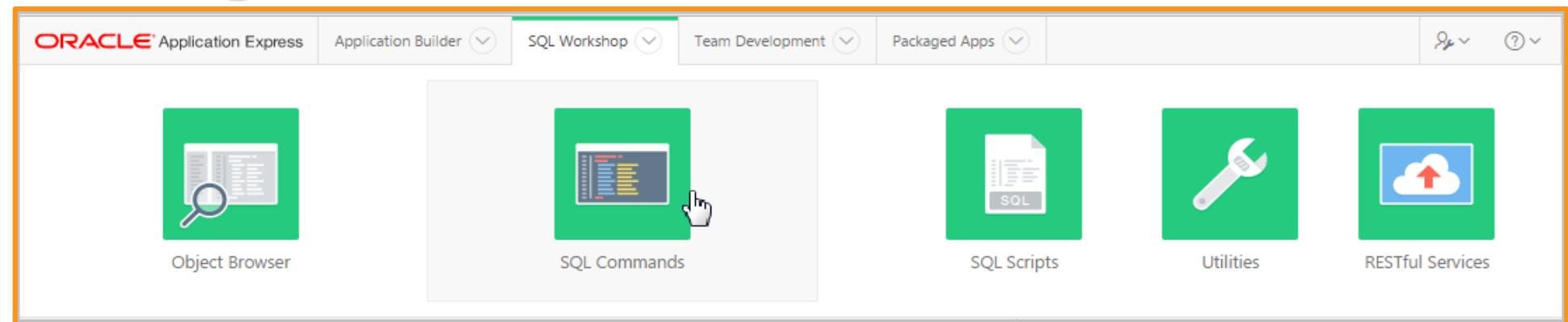
- Create Database Schema**: A link to "Create a new schema for database objects". A callout box above it says "Create a new schema for database objects".
- Set Administrator Password**: A link to "Set or reset your database's privileged user (PDB\_ADMIN) account password". A callout box below it says "Set or reset password for admin".
- Create Document Store**: A link to "Enable or disable a schema-less document-style interface, with JSON storage and access". A callout box to its right says "To create a document store using a schema".
- Manage Application Express**: A link to "Use Application Express (APEX) administrative options". A callout box below it says "To manage tasks such as archiving APEX schemas and association among APEX schema".

# SQL Workshop

The screenshot shows the Oracle Database Home page with several options:

- Web Access**: Develop database and web apps using Oracle Application Express (APEX). Includes "Learn More" and "Watch Video" links.
- Go to SQL Workshop**: Run SQL commands, execute SQL scripts and browse database objects.
- Develop with App Builder**: Declaratively develop and deploy data-driven apps.
- Define REST Data Services**: Create and manage RESTful web service interfaces to your database.
- Install Productivity Apps**: Browse and install productivity apps.

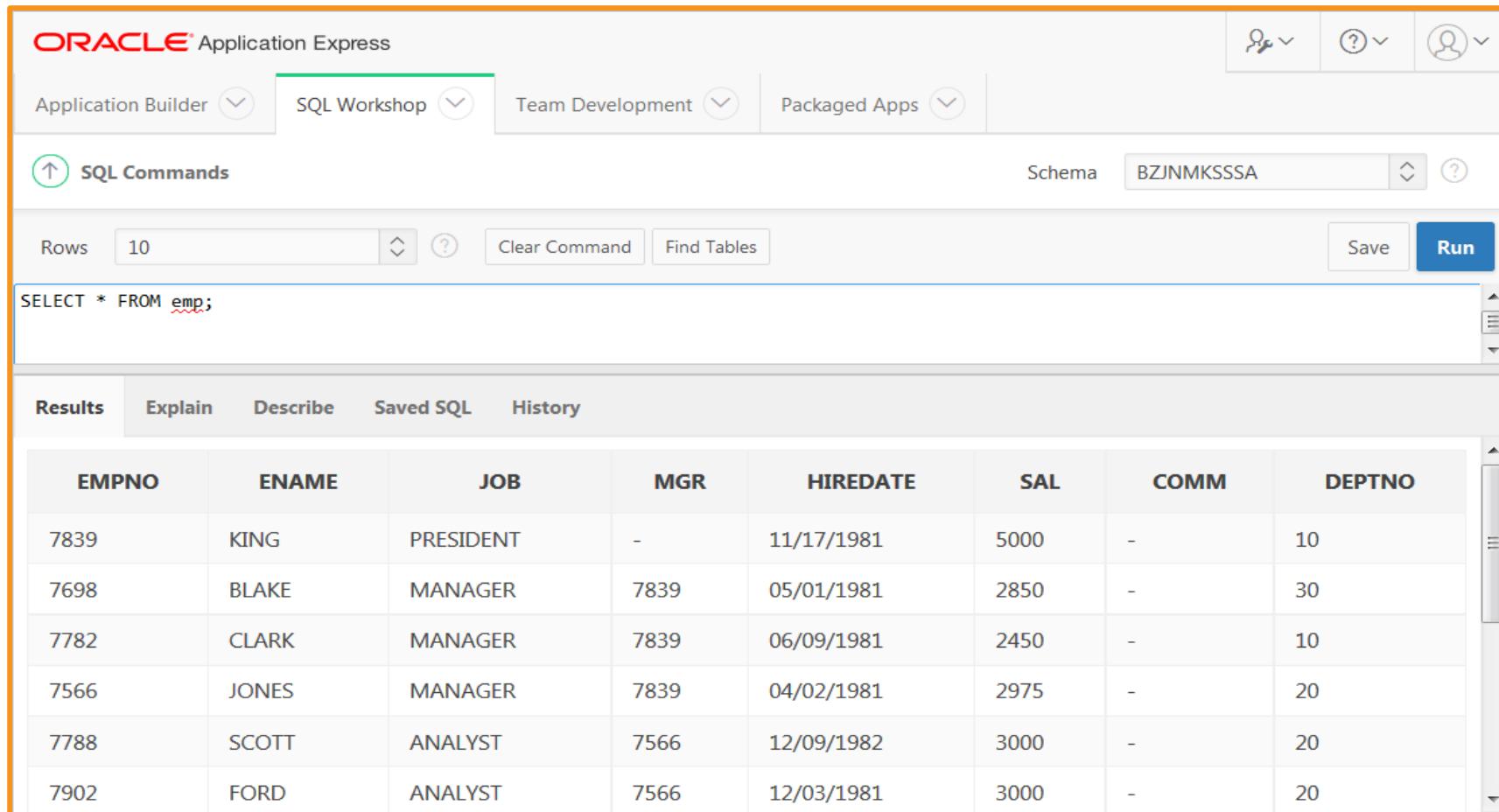
1 Clicking on SQL workshop will lead you to APEX interface



2 To run SQL or PL/SQL you can use the SQL commands utility

# SQL Workshop

You can run SQL statements in the editor.



The screenshot shows the Oracle Application Express SQL Workshop interface. The top navigation bar includes tabs for Application Builder, SQL Workshop (which is selected), Team Development, and Packaged Apps. Below the tabs, there's a section for SQL Commands with a 'Rows' dropdown set to 10, a 'Schema' dropdown set to BZJNMKSSA, and buttons for Save and Run. A command line contains the SQL statement: `SELECT * FROM emp;`. The results pane displays the output of this query:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	-	11/17/1981	5000	-	10
7698	BLAKE	MANAGER	7839	05/01/1981	2850	-	30
7782	CLARK	MANAGER	7839	06/09/1981	2450	-	10
7566	JONES	MANAGER	7839	04/02/1981	2975	-	20
7788	SCOTT	ANALYST	7566	12/09/1982	3000	-	20
7902	FORD	ANALYST	7566	12/03/1981	3000	-	20

# Connecting through Database Clients

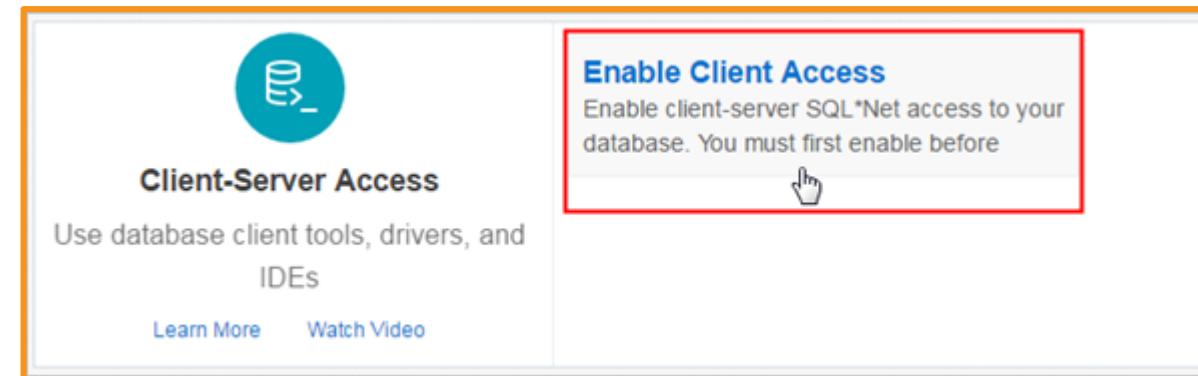
You can connect to Exadata Express through various database clients.

Some of the database clients include:

- SQL\*Plus
- SQLcl
- SQL Developer
- .Net and Visual Studio
- JDBC Thin Client

# Enabling SQL\*Net Access for Client Applications

Enable SQL\*Net Access in the Service Console to obtain the various Database Client options.



# Downloading Client Credentials

Client-Server Access

Use database client tools, drivers, and IDEs

Learn More Watch Video

**Download Client Credentials**

Download a zip file containing your security credentials and network configuration files

**Disable Client Access**

Disable SQL\*Net access and invalidate all existing client credential files

1

2

**Download Client Credentials**

Provide a password for your client credentials, then click **Download** below.

>Password \*

Confirm Password \*

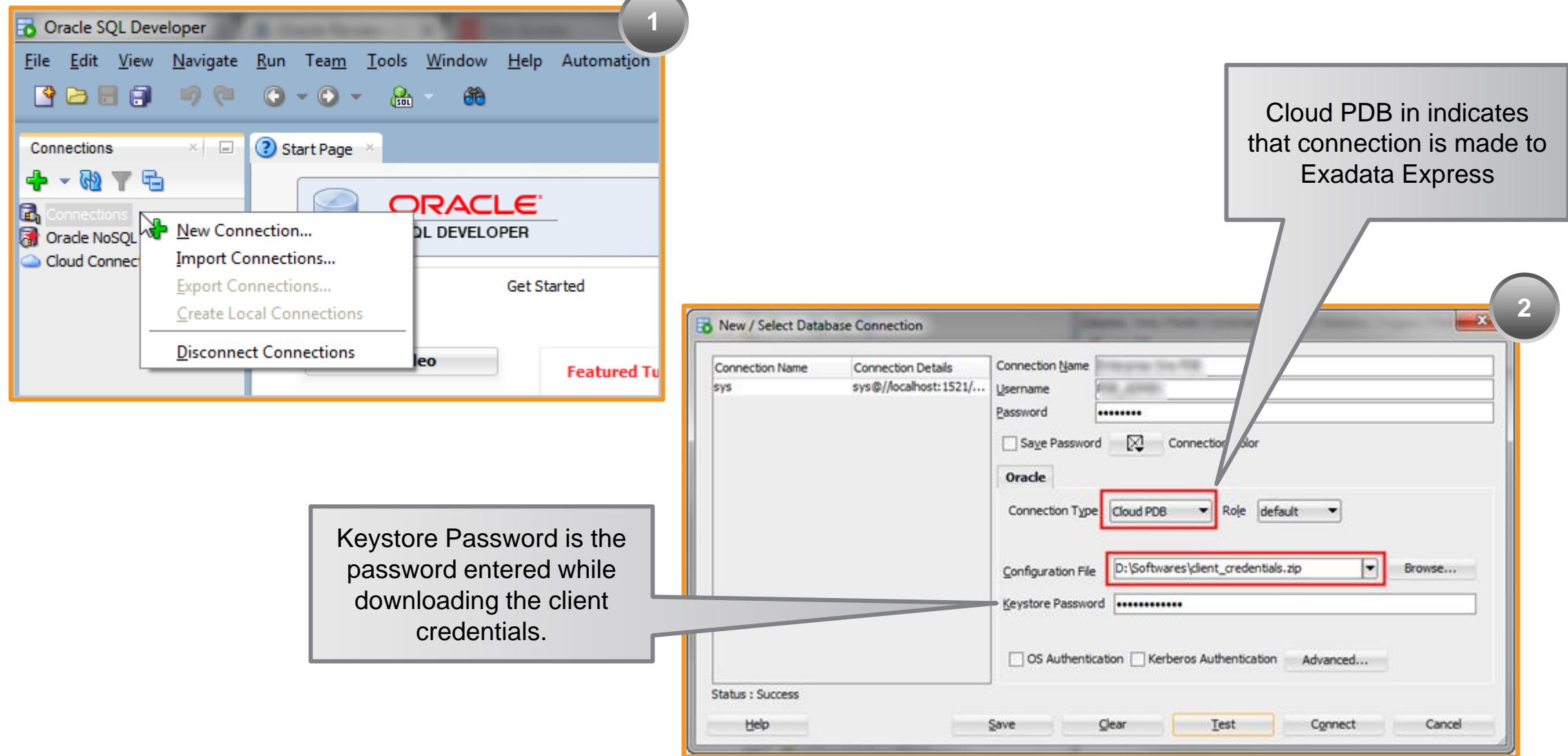
Passwords match!

The client credentials file is required in addition to entering database username and password to access your cloud database. This zip file includes an Oracle Wallet and Java Keystore containing a client certificate. You may wish to share it with authorized users who need to connect to your cloud database. Please keep it secure to avoid unauthorized access.

Cancel

Download

# Connecting Oracle SQL Developer



# Connecting Oracle SQLcl

```
D:\PDB Service\SQL CL\sqlcl-no-jre-latest\sqlcl\bin>sql /nolog  
  
SQLcl: Release 4.2.0.16.160.2007 RC on Thu Sep 08 12:18:07 2016  
  
Copyright (c) 1982, 2016, Oracle. All rights reserved.  
SQL>
```

1

```
SQL> set cloudconfig client_credentials.zip  
Wallet Password: *****  
Using temp directory:C:\Users\APOTHU~1.ORA\AppData\Local\Temp\  
oracle_cloud_config6707346342028726502
```

2

```
SQL> conn pdb_admin/welcome1@dbaccess  
Connected.  
SQL>
```

3

# Summary

- In this lesson, you should have learned about:
  - The salient features of Oracle Cloud
  - Oracle Database Exadata Express Cloud Service