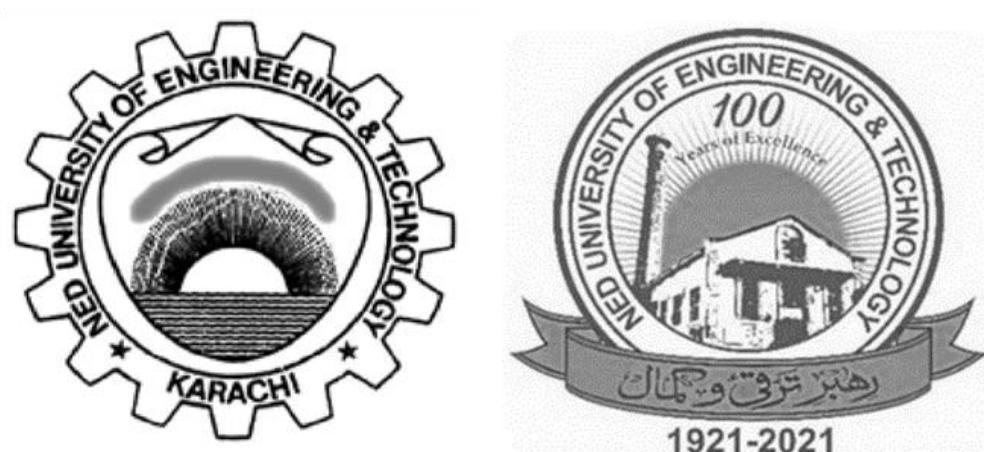


# **Practical Workbook CT-**

**261**

## **DATABASE MANAGEMENT SYSTEMS**



Name: Fizza Ahmed

Roll No: Dt-23304

Batch: 2023

Department: Data Science

# **Practical Workbook**

## **Database Management Systems**

**(CT – 261)**

Prepared by  
Dr. Umer Farooq  
CS & IT

Approved by

Chairman

Department of Computer Science & Information Technology NED  
University of Engineering & Technology

# Table of Contents

S. No.	Object	Date
01	Basic SQL SELECT Statements	13-Jan-2025
02	Restricting and Sorting Data	13-Jan-2025
03	Single-Row Functions	19-Jan-2025
04	Displaying Data from Multiple Tables	26-Jan-2025
05	Aggregating Data Using Group Functions	03-Feb-2025
06	Using Sub queries	17-Feb-2025
07	Manipulating Data	7-Apr-2025
08	Creating and Managing Tables	14-Apr-2025
09	Constraints	21-Apr-2025
10	Creating Views	28-Apr-2025
11	PL/SQL Declaring Variables	5-May-2025
12	PL/SQL Writing Executable Statements	12-May-2025

## **Tables Used in the Course**

### **EMPLOYEES, DEPARTMENTS, JOB\_GRADES**

The following main tables are used in this course:

- EMPLOYEES table, which gives details of all the employees
- DEPARTMENTS table, which gives details of all the departments
- JOB\_GRADES table, which gives details of salaries for various grades

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5600	
141	Trenna	Rajs	TRAJS	17-OCT-95	ST_CLERK	3500	
142	Curtis	Davies	CDAMES	29-JAN-97	ST_CLERK	3100	
143	Randall	Matos	RMATOS	15-MAR-98	ST_CLERK	2600	
144	Peter	Vargas	PVARGAS	09-JUL-98	ST_CLERK	2500	
149	Eleni	Zlotkey	EZLOTKEY	29-JAN-00	SA_MAN	10500	2
174	Ellen	Abel	EABEL	11-MAY-96	SA REP	11000	3
176	Jonathon	Taylor	JTAYLOR	24-MAR-98	SA REP	8600	2
178	Kimberely	Grant	KGRAFT	24-MAY-98	SA REP	7000	15
200	Jennifer	Whalen	JWHALEN	17-SEP-87	AD_ASST	4400	
201	Michael	Hartstein	MHARTSTE	17-FEB-96	MKT_MAN	13000	
202	Pat	Fay	PFAY	17-AUG-97	MKT REP	8000	
205	Shelley	Higgins	SHIGGINS	07-JUN-94	AC_MGR	12000	
206	William	Gietz	WGRIETZ	07-JUN-94	AC_ACCOUNT	8300	

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		1700

GRADE	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	59999
D	10000	14999
E	15000	24999
F	25000	40000

# 1-Writing Basic SQL SELECT Statements

## 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
- Differentiate between SQL statements and iSQL\*Plus commands

### Capabilities of SQL SELECT Statements

**Projection   Selection      Join**

A SELECT statement retrieves information from the database. Using a SELECT statement, you can do the following:

- Projection: You can use the projection capability in SQL to choose the columns in a table that you want returned by your query. You can choose as few or as many columns of the table as you require.
- Selection: You can use the selection capability in SQL to choose the rows in a table that you want returned by a query. You can use various criteria to restrict the rows that you see.
- Joining: You can use the join capability in SQL to bring together data that is stored in different tables by creating a link between them. You learn more about joins in a later lesson.

### Basic SELECT Statement

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

- **SELECT identifies what columns**
- **FROM identifies which table**

**Selecting All Columns**

**SELECT \* FROM departments;**

**SELECT department\_id, department\_name, manager\_id, location\_id FROM departments;**

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

LOCATION_ID	DEPARTMENT_ID
1700	10
1800	20
1500	50

8 rows selected.

### Selecting Specific Columns

`SELECT department_id, location_id FROM departments;`

### Arithmetic Expressions

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

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

### Using Arithmetic Operators

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

### Operator Precedence

`/ *+ - •`

Multiplication and division take priority over addition and subtraction.

- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements.

### Operator Precedence

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

LAST_NAME	SALARY	12*(SALARY+100)
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200
Hunold	9000	109200
Ernst	6000	73200
Lorentz	4200	51600
Gietz	8300	100800

20 rows selected.

## Using Parentheses

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

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
King	AD_PRES	24000	
Kochhar	AD_VP	17000	
Zlotkey	SA_MAN	10500	.2
Abel	SA_REP	11000	.3
Taylor	SA_REP	8600	.2
Higgins	AC_MGR	12000	
Gietz	AC_ACCOUNT	8300	

20 rows selected.

## Defining a Null Value

- A null is a value that is unavailable, unassigned, unknown, or inapplicable.
  - A null is not the same as zero or a blank space.
- `SELECT last_name, job_id, salary, commission_pct  
FROM employees;`

LAST_NAME	12*SALARY*COMMISSION_PCT
King	
Kochhar	
Zlotkey	25200
Abel	39600
Taylor	20640
Higgins	
Gietz	

20 rows selected.

## Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null.

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

## 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 alias

- Requires double quotation marks if it contains spaces or special characters or is case sensitive

NAME	COMM
King	
Kochhar	
Higgins	
Gietz	

20 rows selected

Name	Annual Salary
King	286000
Kochhar	204000
Higgins	144000
Gietz	96000

20 rows selected.

## Using Column Aliases

```
SELECT last_name AS name, commission_pct comm
```

```
FROM employees;
```

```
SELECT last_name "Name",salary*12 "Annual Salary"
```

```
FROM employees;
```

A concatenation operator:

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

Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP
HunoldIT_PROG
GietzAC_ACCOUNT

20 rows selected

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

## Using the Concatenation Operator

### Literal Character Strings

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

Employee Details
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Gietz is a AC_ACCOUNT

20 rows selected.

MONTHLY
King: 1 Month salary = 24000
Kochhar: 1 Month salary = 17000
De Haan: 1 Month salary = 17000
Hunold: 1 Month salary = 9000
Ernst: 1 Month salary = 6000
Lorentz: 1 Month salary = 4200
Mourgos: 1 Month salary = 5800
Rajs: 1 Month salary = 3500

20 rows selected.

### Using Literal Character Strings

**SELECT last\_name ||' is a '|job\_id AS "Employee Details"  
FROM employees;**

DEPARTMENT_ID
90
90
90
60
60
60
50
50

20 rows selected.

### Duplicate Rows

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

**SELECT department\_id FROM employees;**

DEPARTMENT_ID
10
20
50
60
80
90
110

8 rows selected.

DEPARTMENT_ID	JOB_ID
10	AD_ASST
20	MK_MAN
20	MK_REP
50	ST_CLERK
50	ST_MAN
60	IT_PROG
80	SA_MAN
80	SA_REP

13 rows selected.

### Eliminating Duplicate Rows

Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause.

**SELECT DISTINCT department\_id FROM employees;**

### SQL and iSQL\*Plus Interaction

SQL statements iSQL\*Plus Oracle Internet server Browser

Query results iSQL\*Plus commands Formatted report Client

SQL and iSQL\*Plus SQL is a command language for communication with the Oracle server from any tool or application. Oracle SQL contains many extensions.

iSQL\*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle server for execution and contains its own command language.

Features of SQL

- Can be used by a range of users, including those with little or no programming experience
- Is a nonprocedural language
- Reduces the amount of time required for creating and maintaining systems
- Is an English-like language

Features of iSQL\*Plus

- Accessed from a browser
- Accepts ad hoc entry of statements
- Provides online editing for modifying SQL statements
- Controls environmental settings
- Formats query results into a basic report
- Accesses local and remote databases

### **SQL Statements versus iSQL\*Plus Commands**

SQL iSQL\*Plus

- A language
  - An environment
  - ANSI standard
  - Oracle proprietary
  - Keyword cannot be
  - Keywords can be abbreviated abbreviated
  - Statements manipulate
  - Commands do not allow data and table definitions manipulation of values in in the database the database
  - Runs on a browser
  - Centrally loaded, does not have to be implemented on each machine
- SQL iSQL\*Plus statements commands
- SQL and iSQL\*Plus (continued)

**Paper-Based Questions**

For questions 2–4, circle either True or False.

**Practice 1**

1. Initiate an iSQL\*Plus session using the user ID and password provided by the instructor.

2.iSQL\*Plus commands access the database.

True/False

3. The following SELECT statement executes successfully:

`SELECT last_name, job_id, salary AS Sal FROM employees;`

True/False

4. The following SELECT statement executes successfully:

`SELECT * FROM job_grades;`

True/False

5. There are four coding errors in this statement. Can you identify them?

`SELECT employee_id, last_name sal x 12 ANNUAL SALARY`

`FROM employees;`

6. Show the structure of the DEPARTMENTS table. Select all data from the table.

7. Show the structure of the

EMPLOYEES table. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first. Save your SQL statement to a file named lab1\_7.sql.

8. Run your query in the file lab1\_7.sql.

9. Create a query to display unique job codes from the EMPLOYEES table.

If you have time, complete the following exercises:

10. Copy the statement from lab1\_7.sql into the iSQL\*Plus Edit window. Name the column headings Emp #,Employee,Job, and Hire Date, respectively. Run your query again.

<b>Employee and Title</b>
King, AD_PRES
Kochhar, AD_VP
De Haan, AD_VP
Hunold, IT_PROG
Ernst, IT_PROG
Lorentz, IT_PROG
Mourgos, ST_MAN
Rajs, ST_CLERK
Davies, ST_CLERK

Gietz, AC_ACCOUNT
-------------------

20 rows selected.

THE_OUTPUT
100,Steven,King,SKING,515.123.4567,AD_PRES,,17-JUN-87,24000,,90
101,Neena,Kochhar,NKOCHHAR,515.123.4568,AD_VP,100,21-SEP-89,17000,,90
102,Lex,De Haan,LDEHAAN,515.123.4569,AD_VP,100,13-JAN-93,17000,,90
103,Alexander,Hunold,AHUNOLD,590.423.4567,IT_PROG,102,03-JAN-90,9000,,60
104,Bruce,Ernst,BERNST,590.423.4568,IT_PROG,103,21-MAY-91,6000,,60
107,Diana,Lorentz,DLORENTZ,590.423.5567,IT_PROG,103,07-FEB-99,4200,,60
124,Kevin,Mourgos,KMOURGOS,650.123.5234,ST_MAN,100,16-NOV-99,5800,,50
141,Trenna,Rajs,TRAJS,650.121.8009,ST_CLERK,124,17-OCT-95,3500,,50
206,William,Gietz,WGIETZ,515.123.8181,AC_ACCOUNT,205,07-JUN-94,8300,,110

20 rows selected.

11. Display the last name concatenated with the job ID, separated by a comma and space, and name the column Employee and Title.

If you want an extra challenge, complete the following exercise:

12. Create a query to display all the data from the EMPLOYEES table. Separate each column by a comma. Name the column THE\_OUTPUT.

## LAB 1

### WRITING BASIC SQL SELECT STATEMENTS

1. Initiate an iSQL\*Plus session using the user ID and password provided by the instructor.

2. ISQL\*Plus commands access the database.

TRUE

3. The following SELECT statement executes successfully:

`SELECT last_name, job_id, salary AS Sal FROM employees;`

TRUE

4. The following SELECT statement executes successfully:

`SELECT * FROM job_grades;`

TRUE

5. There are four coding errors in this statement. Can you identify them?

`SELECT employee_id, last_name sal x 12 ANNUAL SALARY  
FROM employees;`

**CORRECTED:** `SELECT employee_id, last_name, salary * 12 as  
ANNUAL_SALARY FROM employees;`

## 6. Show the structure of the DEPARTMENTS table. Select all data from the table.

Language SQL Rows 10 Clear Command Find Tables Save Run

```
1 DESCRIBE OEHR_DEPARTMENTS
```

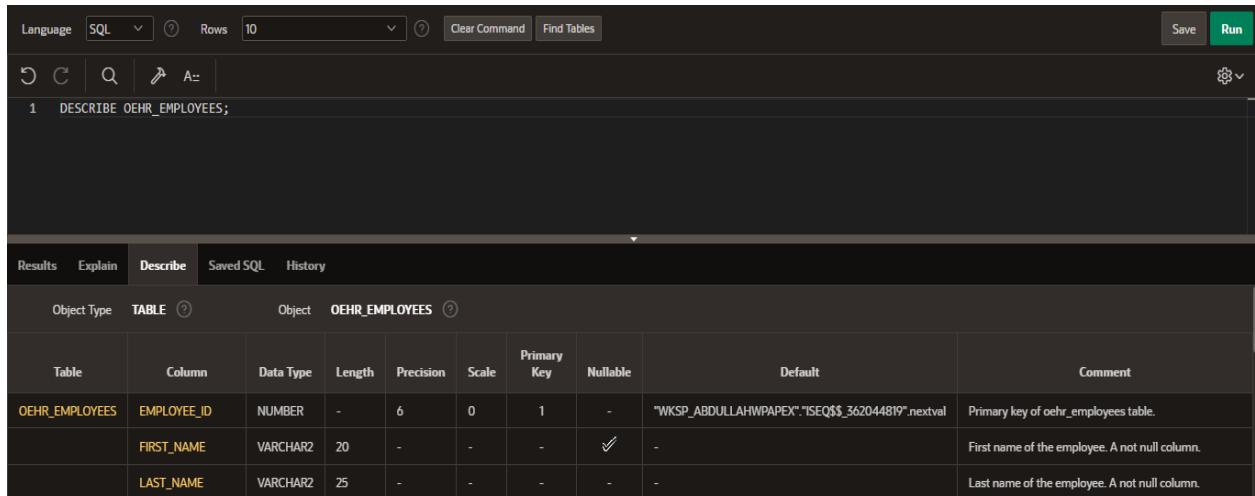
Object Type	TABLE	Object	OEHR_DEPARTMENTS						
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
OEHR_DEPARTMENTS	DEPARTMENT_ID	NUMBER	-	4	0	1	-	"WKSP_ABDULLAHWPAPEX"."ISEQ\$\$_362044813".nextval	Primary key column of oehr_departments table.
	DEPARTMENT_NAME	VARCHAR2	30	-	-	-	-	-	A not null column that shows name of a department. Administration, Marketing, Purchasing, Human Resources, Shipping, IT, Executive, Public Relations, Sales, Finance, and

Language SQL Rows 10 Clear Command Find Tables Save Run

```
1 SELECT * FROM OEHR_DEPARTMENTS;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500

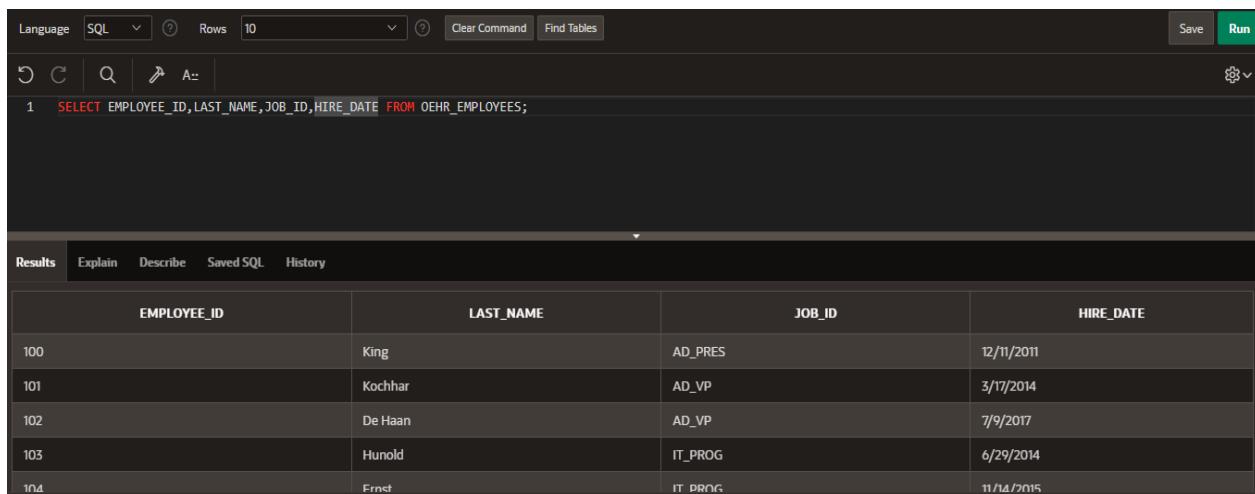
7. Show the structure of the EMPLOYEES table. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first. Save your SQL statement to a file named lab1\_7.sql.



The screenshot shows a database interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Clear Command, Find Tables, Save, Run
- Query:** 1 DESCRIBE OEHR\_EMPLOYEES;
- Table Structure:**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
OEHR_EMPLOYEES	EMPLOYEE_ID	NUMBER	-	6	0	1	-	"WKSP_ABDULLAHWPAPEX"."ISEQ\$\$_362044819".nextval	Primary key of oehr_employees table.
	FIRST_NAME	VARCHAR2	20	-	-	-	✓	-	First name of the employee. A not null column.
	LAST_NAME	VARCHAR2	25	-	-	-	-	-	Last name of the employee. A not null column.



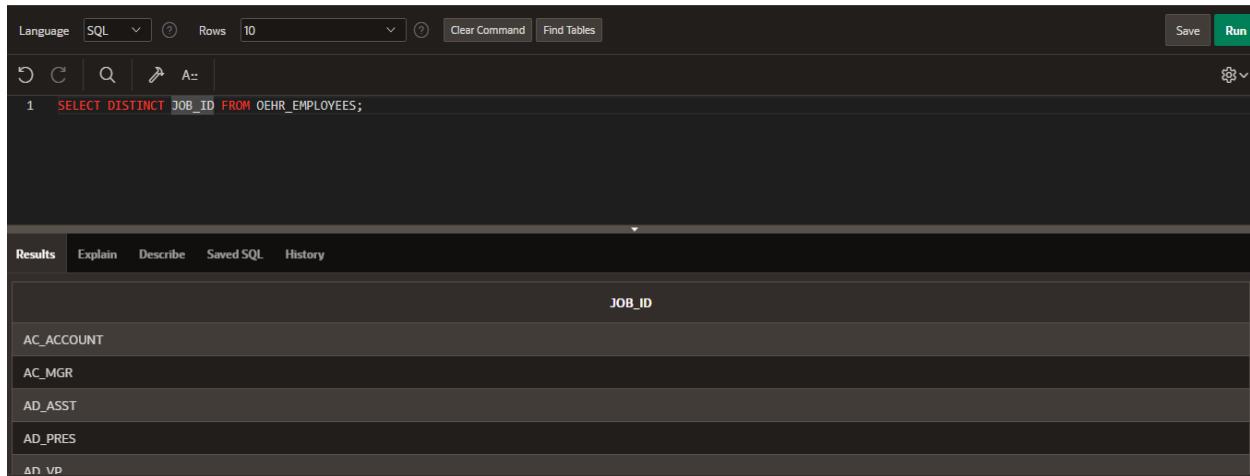
The screenshot shows a database interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Clear Command, Find Tables, Save, Run
- Query:** 1 SELECT EMPLOYEE\_ID, LAST\_NAME, JOB\_ID, HIRE\_DATE FROM OEHR\_EMPLOYEES;
- Table Data:**

EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE
100	King	AD_PRES	12/11/2011
101	Kochhar	AD_VP	3/17/2014
102	De Haan	AD_VP	7/9/2017
103	Hunold	IT_PROG	6/29/2014
104	Ernst	IT_PROG	11/14/2015

8. Run your query in the file lab1\_7.sql.

9. Create a query to display unique job codes from the EMPLOYEES table.

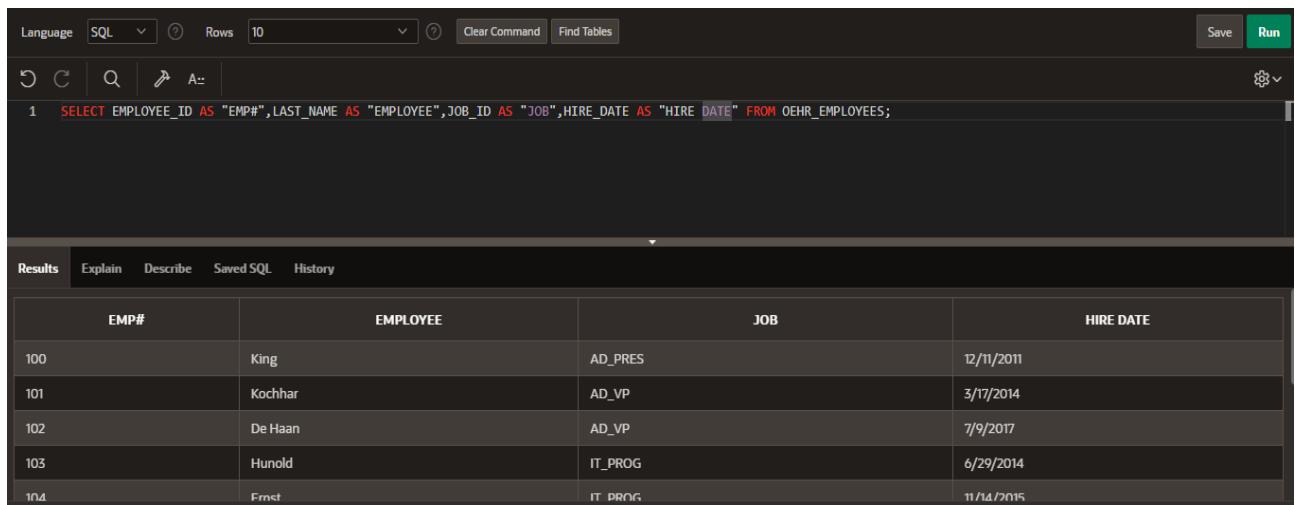


The screenshot shows the iSQL\*Plus interface with the following details:

- Language:** SQL
- Rows:** 10
- Command:** SELECT DISTINCT JOB\_ID FROM OEHR\_EMPLOYEES;
- Results:** The results table has one column labeled "JOB\_ID" containing the following values:
 

AC_ACCOUNT
AC_MGR
AD_ASST
AD_PRES
AD_VP

10. Copy the statement from lab1\_7.sql into the iSQL\*Plus Edit window. Name the column headings Emp #,Employee,Job, and Hire Date, respectively. Run your query again.



The screenshot shows the iSQL\*Plus interface with the following details:

- Language:** SQL
- Rows:** 10
- Command:** SELECT EMPLOYEE\_ID AS "EMP#", LAST\_NAME AS "EMPLOYEE", JOB\_ID AS "JOB", HIRE\_DATE AS "HIRE DATE" FROM OEHR\_EMPLOYEES;
- Results:** The results table has four columns with the specified headings:
 

EMP#	EMPLOYEE	JOB	HIRE DATE
100	King	AD_PRES	12/11/2011
101	Kochhar	AD_VP	3/17/2014
102	De Haan	AD_VP	7/9/2017
103	Hunold	IT_PROG	6/29/2014
104	Ernst	IT_PROG	11/14/2015

11. Display the last name concatenated with the job ID, separated by a comma and space, and name the column Employee and Title.

The screenshot shows a SQL query interface with the following details:

- Language: SQL
- Rows: 10
- SQL Command:

```
1 SELECT LAST_NAME||' , '||JOB_ID AS "EMPLOYEE AND TITLE"FROM OEHR_EMPLOYEES;
```
- Results tab selected.
- Output table header: EMPLOYEE AND TITLE
- Output data:

EMPLOYEE AND TITLE
Abel , SA_REP
Ande , SA_REP
Atkinson , ST_CLERK
Austin , IT_PROG
Rae , PR_RFP

12. Create a query to display all the data from the EMPLOYEES table. Separate each column by a comma. Name the column THE\_OUTPUT.

The screenshot shows a SQL query interface with the following details:

- Language: SQL
- Rows: 10
- SQL Command:

```
1 SELECT EMPLOYEE_ID||','||FIRST_NAME||','||LAST_NAME||','||EMAIL||','||PHONE_NUMBER||','||HIRE_DATE||','||JOB_ID||','||SALARY||','||COMMISSION_PCT||','||MANAGER_ID||','||DEPARTMENT_ID "THE_OUTPUT" FROM OEHR_EMPLOYEES;
```
- Results tab selected.
- Output table header: THE\_OUTPUT
- Output data:

THE_OUTPUT
100 , Steven , King , 5151234567 , 12/11/2011 , AD_PRES , 24000 , , 90
101 , Neena , Kochhar , NKOCHHAR , 5151234568 , 3/17/2014 , AD_VP , 17000 , , 100 , 90
102 , Lex , De Haan , LDEHAAN , 5151234569 , 7/9/2017 , AD_VP , 17000 , , 100 , 90
103 , Alexander , Hunold , AHUNOLD , 5904234567 , 6/29/2014 , IT_PROG , 9000 , , 102 , 60
104 , Bruce , Ernst , RERNST , 5904234568 , 11/14/2015 , IT_PROG , 6000 , , 103 , 60

## 2- Restricting and Sorting Data Objectives

### Objectives:

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

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

### Limiting Rows Using a Selection

EMPLOYEES “retrieve all employees in department 90”

Limiting Rows Using a Selection

### Limiting the Rows Selected

- Restrict the rows returned by using the WHERE clause.

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

- The WHERE clause follows the FROM clause.

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

### Using the WHERE Clause

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

### Character Strings and Dates

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

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

## Comparison Conditions

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

### Example

... WHERE hire\_date='01-JAN-95'

... WHERE salary>=6000

... WHERE last\_name='Smith'

An alias cannot be used in the WHERE clause.

**Note:** The symbol != and ^= can also represent the *not equal to* condition.

LAST_NAME	SALARY
Matos	2600
Vargas	2500

## Using Comparison Conditions

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

## Other Comparison Conditions

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

## Using the BETWEEN Condition

Use the BETWEEN condition 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
The BETWEEN Condition
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	6800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

8 rows selected.

### Using the IN Condition

Use the IN membership condition to test for values in a list.

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

### Using the LIKE Condition

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

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

LAST_NAME
Kochhar
Lorentz
Mourgos

EMPLOYEE_ID	LAST_NAME	JOB_ID
149	Zlotkey	SA_MAN
174	Abel	SA_REP
176	Taylor	SA_REP
178	Grant	SA_REP

### Using the LIKE Condition

- You can combine pattern-matching characters.
- ```
SELECT last_name FROM employees
WHERE last_name LIKE '_o%';
```
- You can use the ESCAPE identifier to search for the Actual % and \_ symbols.

| LAST_NAME | JOB_ID  | COMMISSION_PCT |
|-----------|---------|----------------|
| King      | AD_PRES |                |
| Kochhar   | AD_VP   |                |
| De Haan   | AD_VP   |                |

| LAST_NAME | MANAGER_ID |
|-----------|------------|
| King      |            |

| Gietz | AC_ACCOUNT |
|-------|------------|
|       |            |

16 rows selected.

## Using the NULL Conditions

Test for nulls with the IS NULL operator.

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

## Logical Conditions

| Meaning | Operator | Description                    |
|---------|----------|--------------------------------|
|         |          | Returns TRUE if both component |
| AND     |          | conditions are true            |
| OR      |          | Returns TRUE                   |
|         |          | If either component            |
|         |          | condition is true              |
| NOT     |          | Returns TRUE if the following  |
|         |          | condition is false             |

## Using the AND Operator

AND requires both conditions to be true.

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

## Using the OR Operator

OR requires either condition to be true.

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >= 10000
```

**OR job\_id LIKE '%MAN%';**

| LAST_NAME | JOB_ID     |
|-----------|------------|
| King      | AD_PRES    |
| Kochhar   | AD_VP      |
| De Haan   | AD_VP      |
| Mourgos   | ST_MAN     |
| Zlotkey   | SA_MAN     |
| Whalen    | AD_ASST    |
| Hartstein | MK_MAN     |
| Fay       | MK_REP     |
| Higgins   | AC_MGR     |
| Gietz     | AC_ACCOUNT |

10 rows selected.

### Using the NOT Operator

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

### Rules of Precedence

Order Evaluated Operator

- 1 Arithmetic operators
- 2 Concatenation operator
- 3 Comparison conditions
- 4 IS [NOT] NULL , LIKE , [NOT] IN
- 5 [NOT] BETWEEN
- 6 NOT logical condition
- 7 AND logical condition
- 8 OR logical condition

Override rules of precedence by using parentheses.

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;
```

| LAST_NAME | JOB_ID  | SALARY |
|-----------|---------|--------|
| King      | AD_PRES | 24000  |

### Use parentheses to force priority.

```
SELECT last_name, job_id, salary
FROM employees
WHERE (job_id = 'SA_REP'
OR job_id = 'AD_PRES')
```

**AND salary > 15000;**

20 rows selected.

| LAST_NAME | JOB_ID  | DEPARTMENT_ID | HIRE_DATE |
|-----------|---------|---------------|-----------|
| King      | AD_PRES | 90            | 17-JUN-87 |
| Whalen    | AD_ASST | 10            | 17-SEP-87 |
| Kochhar   | AD_VP   | 90            | 21-SEP-89 |
| Hunold    | IT_PROG | 60            | 03-JAN-90 |
| Ernst     | IT_PROG | 60            | 21-MAY-91 |
| De Haan   | AD_VP   | 90            | 13-JAN-93 |

## ORDER BY Clause

- Sort rows with the ORDER BY clause
- ASC : ascending order (the default order)
- DESC : descending order
- The ORDER BY clause comes last in the SELECT statement.

**SELECT last\_name, job\_id, department\_id, hire\_date FROM employees  
ORDER BY hire\_date;**

### Syntax

**SELECT expr FROM table [WHERE condition(s) ] [ORDER BY {column,  
Expr } [ASC|DESC]];**

| LAST_NAME | JOB_ID   | DEPARTMENT_ID | HIRE_DATE |
|-----------|----------|---------------|-----------|
| Zlotkey   | SA_MAN   | 80            | 29-JAN-00 |
| Mourgos   | ST_MAN   | 50            | 16-NOV-99 |
| Grant     | SA_REP   |               | 24-MAY-99 |
| Lorentz   | IT_PROG  | 60            | 07-FEB-99 |
| Vargas    | ST_CLERK | 50            | 09-JUL-98 |
| Taylor    | SA_REP   | 80            | 24-MAR-98 |
| Matos     | ST_CLERK | 50            | 15-MAR-98 |
| Fay       | MK_REP   | 20            | 17-AUG-97 |
| Davies    | ST_CLERK | 50            | 29-JAN-97 |
| Abel      | SA_REP   | 80            | 11-MAY-96 |

|      |         |    |           |
|------|---------|----|-----------|
| King | AD_PRES | 90 | 17-JUN-87 |
|------|---------|----|-----------|

20 rows selected.

## Sorting in Descending Order

**SELECT last\_name, job\_id, department\_id, hire\_date  
FROM employees  
ORDER BY hire\_date DESC;**

| EMPLOYEE_ID | LAST_NAME | ANNSAL |
|-------------|-----------|--------|
| 144         | Vergas    | 30000  |
| 143         | Matos     | 31200  |
| 142         | Davies    | 37200  |
| 141         | Rajs      | 42000  |
| 107         | Lorentz   | 50400  |
| 200         | Whalen    | 52800  |
| 124         | Mourgos   | 69600  |
| 104         | Ernst     | 72000  |
| 202         | Fay       | 72000  |
| 178         | Grant     | 84000  |
| 206         | Gietz     | 96000  |
|             |           |        |
| 100         | King      | 288000 |

20 rows selected.

### Sorting by Column Alias

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

**Sorting by Column Aliases**

You can use a column alias in the ORDER BY clause. The slide example sorts the data by annual salary.

|         |     |       |
|---------|-----|-------|
| Higgins | 110 | 12000 |
| Gietz   | 110 | 8300  |
| Grant   |     | 7000  |

20 rows selected.

| LAST_NAME | DEPARTMENT_ID | SALARY |
|-----------|---------------|--------|
| Whalen    | 10            | 4400   |
| Hartstein | 20            | 13000  |
| Fay       | 20            | 6000   |
| Mourgos   | 50            | 5800   |
| Rajs      | 50            | 3900   |

### Sorting by Multiple Columns

- The order of ORDER BY list is the order of sort.

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

- You can sort by a column that is not in the SELECT list.

### Practice 1

1. Create a query to display the last name and salary of employees earning more than \$12,000.

Place your SQL statement in a text file named lab2\_1.sql . Run your query.

2. Create a query to display the employee last name and department number for employee number 176.

3. Modify lab2\_1.sql to display the last name and salary for all employees whose salary is not in the range of \$5,000 and \$12,000. Place your SQL statement in a text file named lab2\_3.sql

4. Display the employee last name, job ID, and start date of employees hired between February 20, 1998, and May 1, 1998. Order the query in ascending order by start date.

5. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.

6. Modify lab2\_3.sql to list the last name and salary of employees who earn between \$5,000 and \$12,000, and are in department 20 or 50. Label the columns Employee and Monthly Salary , respectively. Resave lab2\_3.sql as lab2\_6.sql . Run the statement in lab2\_6.sql .

7. Display the last name and hire date of every employee who was hired in 1994.

8. Display the last name and job title of all employees who do not have a manager.

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

If you have time, complete the following exercises:

10. Display the last names of all employees where the third letter of the name is an *a*.

11. Display the last name of all employees who have an *a* and an *e* in their last name.

If you want an extra challenge, complete the following exercises:

12. Display the last name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to \$2,500, \$3,500, or \$7,000.

13. Modify lab2\_6.sql to display the last name, salary, and commission for all employees whose commission amount is 20%. Resave lab2\_6.sql as lab2\_13.sql . Rerun the statement in lab2\_13.sql.

## LAB:02

1. Create a query to display the last name and salary of employees earning more than \$12,000.

Place your SQL statement in a text file named lab2\_1.sql . Run your query

The screenshot shows a SQL query execution interface. The query is:

```
1 SELECT LAST_NAME , SALARY FROM OEHR_EMPLOYEES
2 WHERE SALARY > 12000;
3
```

The results table has two columns: LAST\_NAME and SALARY. The data is:

| LAST_NAME | SALARY |
|-----------|--------|
| Hartstein | 13000  |
| King      | 24000  |
| Kochhar   | 17000  |
| De Haan   | 17000  |

2. Create a query to display the employee last name and department number for employee number 176.

The screenshot shows a SQL query execution interface. The query is:

```
1 SELECT LAST_NAME,DEPARTMENT_ID FROM OEHR_EMPLOYEES
2 WHERE EMPLOYEE_ID='176';
```

The results table has two columns: LAST\_NAME and DEPARTMENT\_ID. The data is:

| LAST_NAME | DEPARTMENT_ID |
|-----------|---------------|
| Taylor    | 80            |

1 rows returned in 0.00 seconds    Download

3. Modify lab2\_1.sql to display the last name and salary for all employees whose salary is not in the range of \$5,000 and \$12,000. Place your SQL statement in a text file named lab2\_3.sql

```
1  SELECT LAST_NAME , SALARY FROM OEHR_EMPLOYEES
2  WHERE SALARY NOT BETWEEN 5000 AND 12000;
3
```

| LAST_NAME | SALARY |
|-----------|--------|
| OConnell  | 2600   |
| Grant     | 2600   |
| Whalen    | 4400   |
| Hartstein | 13000  |
| King      | 24000  |

4. Display the employee last name, job ID, and start date of employees hired between February 20, 1998,

and May 1, 1998. Order the query in ascending order by start date

```
Language SQL Rows 10 Clear Command Find Tables Save Run
SELECT LAST_NAME,JOB_ID,HIRE_DATE FROM OEHR_EMPLOYEES
WHERE HIRE_DATE BETWEEN TO_DATE('20-FEB-98', 'DD-MON-YY') AND TO_DATE('01-MAY-98', 'DD-MON-YY')
ORDER BY HIRE_DATE ASC;
```

| LAST_NAME | JOB_ID   | HIRE_DATE |
|-----------|----------|-----------|
| King      | AD_PRES  | 20-FEB-98 |
| Allen     | ST_CLERK | 20-FEB-98 |
| Ernest    | ST_CLERK | 20-FEB-98 |
| Margolin  | ST_CLERK | 20-FEB-98 |
| Hardy     | ST_CLERK | 20-FEB-98 |
| James     | ST_CLERK | 20-FEB-98 |
| King      | AD_VP    | 01-MAY-98 |
| King      | AD_ASST  | 01-MAY-98 |
| King      | AD_ASST  | 01-MAY-98 |
| King      | AD_ASST  | 01-MAY-98 |

5. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.

```

Language SQL Rows 10 Clear Command Find Tables Save Run
SELECT LAST_NAME,DEPARTMENT_ID FROM OEHR_EMPLOYEES
WHERE DEPARTMENT_ID='20' AND DEPARTMENT_ID='50'
ORDER BY LAST_NAME ASC;

Results Explain Describe Saved SQL History
no data found

```

6. Modify lab2\_3.sql to list the last name and salary of employees who earn between \$5,000 and \$12,000, and are in department 20 or 50. Label the columns Employee and Monthly Salary , respectively. Resave lab2\_3.sql as lab2\_6.sql . Run the statement in lab2\_6.sql

```

Language SQL Rows 10 Clear Command Find Tables Save Run
SELECT LAST_NAME AS Employee, SALARY AS "Monthly Salary"
FROM OEHR_EMPLOYEES
WHERE SALARY BETWEEN 5000 AND 12000 AND DEPARTMENT_ID IN (20, 50);

Results Explain Describe Saved SQL History

```

| EMPLOYEE | Monthly Salary |
|----------|----------------|
| Fay      | 6000           |
| Weiss    | 8000           |
| Fripp    | 8200           |

Activate Windows  
ahmed4630136@cloud.neduet.edu.pk    pizza\_dbms  
Copyright © 1999, 2024, Oracle and/or its affiliates.  
Go to Settings to activate Windows  
Oracle APEX 24.2.1

Q7. Display the last name and hire date of every employee who was hired in 1994.

The screenshot shows a SQL query interface with the following details:

- Language: SQL
- Rows: 10
- SQL Command:

```
1  SELECT LAST_NAME,HIRE_DATE FROM OEHR_EMPLOYEES
2  WHERE HIRE_DATE BETWEEN TO_DATE('1-JAN-94', 'DD-MON-YY') AND TO_DATE('31-DEC-94', 'DD-MON-YY');
```
- Results tab selected.
- Output: "no data found"

Q8. Display the last name and job title of all employees who do not have a manager.

The screenshot shows a SQL query interface with the following details:

- Language: SQL
- Rows: 10
- SQL Command:

```
1  SELECT LAST_NAME,JOB_ID FROM OEHR_EMPLOYEES
2  WHERE MANAGER_ID IS NULL;
```
- Results tab selected.
- Output:

| LAST_NAME | JOB_ID  |
|-----------|---------|
| King      | AD_PRES |

1 rows returned in 0.01 seconds    Download

Q9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions

The screenshot shows a SQL query being run in a database environment. The query selects last names, salaries, and commission percentages from the OEHR\_EMPLOYEES table where commission is not null, ordered by salary and commission percentage in descending order. The results are displayed in a table with columns LAST\_NAME, SALARY, and COMMISSION\_PCT.

| LAST_NAME | SALARY | COMMISSION_PCT |
|-----------|--------|----------------|
| Russell   | 14000  | .4             |
| Partners  | 13500  | .3             |
| Errazuriz | 12000  | .3             |

Q10. Display the last names of all employees where the third letter of the name is an a.

The screenshot shows a SQL query being run in a database environment. The query selects last names from the OEHR\_EMPLOYEES table where the third character of the last name is an 'a'. The results are displayed in a table with a single column LAST\_NAME.

| LAST_NAME |
|-----------|
| Grant     |
| Grant     |
| Whalen    |

Q11. Display the last name of all employees who have an a and an e in their last name.

A screenshot of a SQL query interface. The top bar shows 'Language' set to 'SQL', 'Rows' set to '10', and buttons for 'Save' and 'Run'. The query window contains the following code:

```
2 FROM OEHR_EMPLOYEES
3 WHERE LAST_NAME LIKE '%a%' AND LAST_NAME LIKE '%e%';
4
```

The results tab is selected, showing a table with a single column 'LAST\_NAME' containing three rows: 'Baer', 'Bates', and 'Colmenares'.

Q12. display the last name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to \$2,500, \$3,500, or \$7,000

A screenshot of a SQL query interface. The top bar shows 'Language' set to 'SQL', 'Rows' set to '10', and buttons for 'Save' and 'Run'. The query window contains the following code:

```
1 SELECT LAST_NAME ,JOB_ID, SALARY
2 FROM OEHR_EMPLOYEES
3 WHERE JOB_ID IN('ST_CLERK,SA_REP') AND (SALARY NOT IN(2500,3500,7500));
4
```

The results tab is selected, showing the message 'no data found'.

Q13. Modify lab2\_6.sql to display the last name, salary, and commission for all employees whose commission amount is 20%. Resave lab2\_6.sql as lab2\_13.sql . Rerun the statement in lab2\_13.sql.

The screenshot shows a SQL query being run in a database environment. The query selects the last name, salary, and commission percentage for employees where the commission percentage is 0.20. The results are displayed in a table with columns for LAST\_NAME, SALARY, and COMMISSION\_PCT.

```
1  SELECT LAST_NAME, SALARY, COMMISSION_PCT FROM OEHR_EMPLOYEES
2  WHERE COMMISSION_PCT=.20;
```

| LAST_NAME | SALARY | COMMISSION_PCT |
|-----------|--------|----------------|
| Zlotkey   | 10500  | .2             |
| Olsen     | 8000   | .2             |
| Cambrault | 7500   | .2             |

## 3-Single-Row Functions

### Objectives

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

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

### SQL Functions

#### Input Output Function

Function performs action arg 1

arg 2 Result value arg n

#### Note:

Most of the functions described in this lesson are specific to Oracle Corporation's version of SQL.

### Two Types of SQL Functions

Functions

Single-row functions

### Single-Row Functions

Single row functions:

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

function\_name[(arg1, arg2,...)]

Single-Row Functions

### Single-Row Functions

Character

General

Number

Single-row

functions

Conversion

Date

### Character Functions

Character functions

Case-manipulation functions

Character-manipulation functions

LOWER  
 CONCAT  
 UPPER  
 SUBSTR  
 INITCAP  
 LENGTH  
 INSTR  
 LPAD | RPAD  
 TRIM  
 REPLACE

**Note:** The functions discussed in this lesson are only some of the available functions.

### Case Manipulation Functions

These functions convert case for character strings.

Function Result

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

### Using Case Manipulation 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';
no rows selected
SELECT employee_id, last_name, department_id
FROM employees
WHERE LOWER(last_name) = 'higgins';
```

### Character-Manipulation Functions

These functions manipulate character strings:

| Function                    | Result     |
|-----------------------------|------------|
| CONCAT('Hello', 'World')    | HelloWorld |
| SUBSTR('HelloWorld', 1,5)   | Hello      |
| LENGTH('HelloWorld')        | 10         |
| INSTR('HelloWorld', 'W')    | 6          |
| LPAD(salary,10,'*')         | *****24000 |
| RPAD(salary, 10, '*')       | 24000****  |
| TRIM('H' FROM 'HelloWorld') | elloWorld  |

| EMPLOYEE_ID | NAME            | JOB_ID | LENGTH(LAST_NAME) | Contains 'a'? |
|-------------|-----------------|--------|-------------------|---------------|
| 174         | Ellen Abel      | SA_REP | 4                 | 0             |
| 176         | Jonathon Taylor | SA_REP | 6                 | 2             |
| 178         | Kimberely Grant | SA_REP | 5                 | 3             |
| 202         | Pat Fay         | MK_REP | 3                 | 2             |

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

## Using the Character-Manipulation Functions

```
SELECT employee_id, CONCAT(first_name, last_name) NAME, job_id,
       LENGTH(last_name), INSTR(last_name, 'a') "Contains 'a'?"
  FROM employees
 WHERE SUBSTR(job_id, 4) = 'REP';
```

## Number Functions

- **ROUND** : Rounds value to specified decimal

ROUND(45.926, 2) 45.93

- **TRUNC** : Truncates value to specified decimal

TRUNC(45.926, 2) 45.92

- **MOD** : Returns remainder of division

MOD(1600, 300) 100

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

## Using the ROUND Function

```
SELECT ROUND(45.923,2), ROUND(45.923,0),ROUND(45.923,-1)
  FROM DUAL;
```

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

| TRUNC(45.923,2) | TRUNC(45.923) | TRUNC(45.923,-2) |
|-----------------|---------------|------------------|
| 45.92           | 45            | 0                |

## Using the TRUNC Function

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

| LAST_NAME | SALARY | MOD(SALARY,5000) |
|-----------|--------|------------------|
| Abel      | 11000  | 1000             |
| Taylor    | 8600   | 3600             |
| Grant     | 7000   | 2000             |

## Using the MOD Function

Calculate the remainder of a salary after it is divided by 5000 for all employees whose job title is sales representative.

```
SELECT last_name, salary, MOD(salary, 5000)
  FROM employees
```

**WHERE job\_id = 'SA\_REP';**

| LAST_NAME | HIRE_DATE |
|-----------|-----------|
| Gietz     | 07-JUN-94 |
| Grant     | 24-MAY-99 |

### Working with Dates

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

**SELECT last\_name, hire\_date FROM employees  
WHERE last\_name like 'G%';**

| SYSDATE   |
|-----------|
| 08-MAR-01 |

### Working with Dates

**SYSDATE is a function that returns:**

- Date
- Time

**Example** Display the current date using the DUAL table.

**SELECT SYSDATE FROM DUAL;**

### Arithmetic with Dates

- Add or subtract a number to or 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.

Operation Result Description

date + number Date Adds a number of days to a date

date - number Date Subtracts a number of days from a date

date - date Number of days Subtracts one date from another

date + number/24 Date Adds a number of hours to a date

| LAST_NAME | WEEKS      |
|-----------|------------|
| King      | 716.227963 |
| Kochhar   | 598.084706 |
| De Haan   | 425.227963 |

### Using Arithmetic Operators with Dates

**SELECT last\_name, (SYSDATE-hire\_date)/7 AS WEEKS**

```
FROM employees
WHERE department_id = 90;
```

**Note:** SYSDATE is a SQL function that returns the current date and time. Your results may differ from the example. If a more current date is subtracted from an older date, the difference is a negative number.

| EMPLOYEE_ID | HIRE_DATE | TENURE     | REVIEW    | NEXT_DAY() | LAST_DAY() |
|-------------|-----------|------------|-----------|------------|------------|
| 107         | 07-FEB-99 | 25.0548529 | 07-AUG-99 | 12-FEB-99  | 28-FEB-99  |
| 124         | 16-NOV-99 | 15.7645303 | 16-MAY-00 | 19-NOV-99  | 30-NOV-99  |
| 143         | 15-MAR-98 | 35.7967884 | 15-SEP-98 | 20-MAR-98  | 31-MAR-98  |
| 144         | 09-JUL-98 | 31.9903368 | 09-JAN-99 | 10-JUL-98  | 31-JUL-98  |
| 149         | 29-JAN-00 | 13.3451755 | 29-JUL-00 | 04-FEB-00  | 31-JAN-00  |
| 176         | 24-MAR-98 | 35.5064658 | 24-SEP-98 | 27-MAR-98  | 31-MAR-98  |
| 178         | 24-MAY-99 | 21.5064658 | 24-NOV-99 | 28-MAY-99  | 31-MAY-99  |

7 rows selected.

## Date Functions

### Using Date Functions

- MONTHS\_BETWEEN ('01-SEP-95','11-JAN-94')  
19.6774194
- ADD\_MONTHS ('11-JAN-94',6) '11-JUL-94'
- NEXT\_DAY ('01-SEP-95','FRIDAY') '08-SEP-95'
- LAST\_DAY('01-FEB-95') '28-FEB-95'

| EMPLOYEE_ID | HIRE_DATE | ROUND(HIR) | TRUNC(HIR) |
|-------------|-----------|------------|------------|
| 142         | 29-JAN-97 | 01-FEB-97  | 01-JAN-97  |
| 202         | 17-AUG-97 | 01-SEP-97  | 01-AUG-97  |

### Using Date Functions

Assume SYSDATE = '25-JUL-95':

- ROUND(SYSDATE,'MONTH') 01-AUG-95
- ROUND(SYSDATE , 'YEAR') 01-JAN-96
- TRUNC(SYSDATE , 'MONTH') 01-JUL-95
- TRUNC(SYSDATE , 'YEAR') 01-JAN-95

**Example**

Compare the hire dates for all employees who started in 1997. Display the employee number, hire date, and month started using the ROUND and TRUNC functions.

```
SELECT employee_id, hire_date, ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH') FROM employees WHERE hire_date LIKE '%97';
```

**Practice 3, Part 1**

This practice is designed to give you a variety of exercises using different functions available for character, number, and date data types.

Complete questions 1 through 5 of Practice 3, found at the end of this lesson.

**Conversion Functions****Data-type conversion****Implicit data-type conversion****Explicit data-type conversion**

**Note:** Although implicit data-type conversion is available, it is recommended that you do explicit data type conversion to ensure the reliability of your SQL statements.

**Implicit Data-Type Conversion**

For assignments, the Oracle server can automatically convert the following:

| From             | To       |
|------------------|----------|
| VARCHAR2 or CHAR | NUMBER   |
| VARCHAR2 or CHAR | DATE     |
| NUMBER           | VARCHAR2 |
| DATE             | VARCHAR2 |
| VARCHAR2 or CHAR | DATE     |

**Note:** CHAR to NUMBER conversions succeed only if the character string represents a valid number.

**Explicit Data-Type Conversion**

**TO\_NUMBER**

**TO\_DATE**

**NUMBER**

**DATE**

**CHARACTER**

**TO\_CHAR**

**TO\_CHAR**

**Function****Purpose**

**TO\_CHAR(** *number | date , [fmt ]*, **VARCHAR2 [nlsparams ])** Converts a number or date value to a character string with format model *fmt*.

**TO\_NUMBER(** *char,[fmt]*, Converts a character string containing digits to a *[nlsparams ]* number in the format specified by the optional format model *Fmt*. The *nlsparams* parameter has the same purpose in this function as in the **TO\_CHAR** function for number conversion.

**TO\_DATE(char , [fmt ],[ nlsparams ])** Converts a character string representing a date to a date value according to the *fmt* specified. If *fmt* is omitted, the format is DD-MON-YY. The *nlsparams* parameter has the same purpose in this function as in the TO\_CHAR function for date conversion.

**Note:** The list of functions mentioned in this lesson includes only some of the available conversion functions.

| EMPLOYEE_ID | MONTH |
|-------------|-------|
| 205         | 06/94 |

## Using the TO\_CHAR Function with Dates

**TO\_CHAR(date, 'format\_model')**

The format model:

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

**SELECT employee\_id, TO\_CHAR(hire\_date, 'MM/YY') Month\_Hired**

**FROM employees**

**WHERE last\_name = 'Higgins';**

## Elements of the Date Format Model

YYYY Full year in numbers

YEAR Year spelled out

MM Two-digit value for month

MONTH Full name of the month

MON Three-letter abbreviation of the

Month Three-letter abbreviation of the

DY day of the week

DAY Full name of the day of the week

DD Numeric day of the month

## Elements of the Date Format Model

- Time elements format the time portion of the date.

HH24:MI:SS AM 15:45:32 PM

- Add character strings by enclosing them in double quotation marks.

DD "of" MONTH 12 of OCTOBER

- Number suffixes spell out numbers.

ddspth fourteenth

## Using the TO\_CHAR Function with Dates

**SELECT last\_name, TO\_CHAR(hire\_date, 'fmDD Month YYYY') HIREDATE**

**FROM employees;**

```
SELECT last_name, TO_CHAR(hire_date, 'fmDdspth "of" Month YYYY fmHH:MI:SS
AM') HIREDATE FROM employees;
```

Notice that the month follows the format model specified: in other words, the first letter is capitalized and the rest are lowercase.

## Using the TO\_CHAR

### Function with Numbers

**TO\_CHAR( number, ' format\_model ')**

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

#### Element Description Example Result

9 Numeric position (number of 9s determine display 999999 1234 width)

0 Display leading zeros 099999 001234

\$ Floating dollar sign \$999999 \$1234

L Floating local currency symbol L999999 FF1234

. Decimal point in position specified 999999.99 1234.00

, Comma in position specified 999,999 1,234

MI Minus signs to right (negative values) 999999MI 1234-

PR Parenthesize negative numbers 999999PR <1234>

EEEE Scientific notation (format must specify four Es) 99.999EEEE 1.234E+03

V Multiply by 10 n times ( n = number of 9s after V) 9999V99 123400 B Display zero

| SALARY     |
|------------|
| \$6,000.00 |

values as blank, not 0 B9999.99 1234.00

## Using the TO\_CHAR

### Function with Numbers

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
```

```
FROM employees
```

```
WHERE last_name = 'Ernst';
```

## Using the TO\_NUMBER

### And TO\_DATE Functions

- Convert a character string to a number format using the TO\_NUMBER

function: TO\_NUMBER( char [,format\_model'])

- Convert a character string to a date format using the TO\_DATE function:

TO\_DATE( char [, 'format\_model'])

- These functions have an fx modifier. This modifier specifies the exact matching for the

character

argument and date format model of a TO\_DATE function.

#### **Example**

Display the names and hire dates of all the employees who joined on May 24, 1999. Because the fx modifier is used, an exact match is required and the spaces after the word "May" are not recognized.

```
SELECT last_name, hire_date FROM employees
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY')
```

#### **Example of RR Date Format**

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

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM employees
WHERE hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
SELECT last_name, TO_CHAR(hire_date, 'DD -Mon-yyyy')
FROM employees
WHERE TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';
no rows selected
```

#### **Nesting Functions**

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

F3(F2(F1(col,arg1),arg2),arg3)

**Step 1 = Result 1**

**Step 2 = Result 2**

**Step 3 = Result 3**

#### **Nesting Functions**

Single-row functions can be nested to any depth. Nested functions are evaluated from the innermost level to the outermost level. Some examples follow to show you the flexibility of these functions.

| LAST_NAME | NVL(TO_CHAR(MANAGER_ID), 'No Manager') |
|-----------|----------------------------------------|
| King      | No Manager                             |

#### **Nesting Functions**

```
SELECT last_name,
NVL(TO_CHAR(manager_id), 'No Manager')
FROM employees
WHERE manager_id IS NULL;
```

#### **Example**

Display the date of the next Friday that is six months from the hire date. The resulting date should appear as Friday, August 13th, 1999. Order the results by hire date.

```

SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS
(hire_date, 6), 'FRIDAY'),
'fmDay, Month DDth, YYYY')
"Next 6 Month Review"
FROM employees
ORDER BY hire_date;

```

## General Functions

These functions work with any data type and pertain to using null value.

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

| Function | Description                                                                                                                                                             |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NVL      | Converts a null value to an actual value                                                                                                                                |
| NVL2     | If expr1 is not null, NVL2 returns expr2.                                                                                                                               |
| NULLIF   | If expr1 is null, NVL2 returns. The argument can have any data type. expr3 expr1                                                                                        |
| COALESCE | Compares two expressions and returns null if they are equal, or the first expression if they are not equal Returns the first non-null expression in the expression list |

## NVL Function

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

NVL Conversions for Various Data Types

Data Type Conversion Example

NUMBER NVL(*number\_column*,9)  
DATE NVL(*date\_column*, '01-JAN-95')

|         |       |    |        |  |
|---------|-------|----|--------|--|
| Vargas  | 2500  |    |        |  |
| Zlotkey | 10500 | .2 | 151200 |  |
| Abel    | 11000 | .3 | 171600 |  |
| Taylor  | 8600  | .2 | 123840 |  |

CHAR or VARCHAR2 NVL(*character\_column*, 'Unavailable')

| LAST_NAME | SALARY | NVL(COMMISSION_PCT,0) | AN_SAL |
|-----------|--------|-----------------------|--------|
| King      | 24000  | 0                     | 288000 |
| Kochhar   | 17000  | 0                     | 204000 |
| De Haan   | 17000  | 0                     | 204000 |
| Hunold    | 9000   | 0                     | 108000 |
| Ernst     | 6000   | 0                     | 72000  |
| Lorentz   | 4200   | 0                     | 50400  |
| Mourgos   | 5800   | 0                     | 69600  |
| Rajs      | 3500   | 0                     | 42000  |
| Davies    | 3100   | 0                     | 37200  |
| Matos     | 2600   | 0                     | 31200  |
| Vargas    | 2500   | 0                     | 30000  |
| Zlotkey   | 10500  | .2                    | 151200 |
| Abel      | 11000  | .3                    | 171600 |

20 rows selected.

| LAST_NAME | SALARY | COMMISSION_PCT | AN_SAL |
|-----------|--------|----------------|--------|
|-----------|--------|----------------|--------|

20 rows selected.

### Using the NVL Function

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

### The NVL Function

To calculate the annual compensation of all employees, you need to multiply the monthly salary by 12 and then add the commission percentage to it.

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

| LAST_NAME | SALARY | COMMISSION_PCT | INCOME   |
|-----------|--------|----------------|----------|
| Zlotkey   | 10500  | .2             | SAL+COMM |
| Abel      | 11000  | .3             | SAL+COMM |
| Taylor    | 8600   | .2             | SAL+COMM |
| Mourgos   | 5800   | 0              | SAL      |
| Rajs      | 3500   | 0              | SAL      |
| Davies    | 3100   | 0              | SAL      |
| Matos     | 2600   | 0              | SAL      |
| Vargas    | 2500   | 0              | SAL      |

8 rows selected.

### Using the NVL2 Function

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

| FIRST_NAME | expr1 | LAST_NAME | expr2 | RESULT |
|------------|-------|-----------|-------|--------|
| William    | 7     | Gietz     | 5     | 7      |
| Shelley    | 7     | Higgins   | 7     |        |
| Pat        | 3     | Fay       | 3     |        |
| Michael    | 7     | Hartstein | 9     | 7      |
| Jennifer   | 8     | Whalen    | 6     | 8      |
| Kimberely  | 9     | Grant     | 5     | 9      |
| Jonathon   | 8     | Taylor    | 6     | 8      |
| Ellen      | 5     | Abel      | 4     | 5      |
| Eleni      | 6     | Zlotkey   | 7     | 5      |
| Peter      | 5     | Vargas    | 6     | 5      |
| Randall    | 7     | Matos     | 5     | 7      |
| Curtis     | 6     | Davies    | 6     |        |
| Trenna     | 6     | Rajs      | 4     | 6      |
| Kevin      | 5     | Mourgos   | 7     | 5      |

20 rows selected.

### Using the NULLIF Function

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

**Note:**

The NULLIF function is logically equivalent to the following CASE expression. The CASE expression is discussed in a subsequent page:  
CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END

### Practice 2, Part 1 (continued)

5. Write a query that displays the employee's last names with the first letter capitalized and all other letters lowercase and the length of the names, for all employees whose name starts with *J* , *A* , or *M* . Give each column an appropriate label. Sort the results by the employees' last names.

### Practice 2, Part 2

6. For each employee, display the employee's last name, and calculate the number of months between today and the date the employee was hired. Label the column MONTHS\_WORKED . Order your results by the number of months employed. Round the number of months up to the closest whole number.

**Note:**

Your results will differ.

|                                                       |
|-------------------------------------------------------|
| Gietz earns \$8,300.00 monthly but wants \$24,900.00. |
|-------------------------------------------------------|

20 rows selected.

|       |                          |
|-------|--------------------------|
| Gietz | \$\$\$\$\$\$\$\$\$\$8300 |
|-------|--------------------------|

20 rows selected.

### Practice 3, Part 2 (continued)

7. Write a query that produces the following for each employee:

<employee last name> earns <salary> monthly but wants <3 times salary>. Label the column Dream Salaries . If you have time, complete the following exercises:

8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with \$. Label the column SALARY .

9. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear similar to "Monday, the Thirty-First of July, 2000."

| EMPLOYEE_AND THEIR SALARIES |       |
|-----------------------------|-------|
| King                        | ***** |
| Kochhar                     | ***** |
| De Haan                     | ***** |
| Hartstei                    | ***** |
| Higgins                     | ***** |
| Abel                        | ***** |
| Zlotkey                     | ***** |
| Hunold                      | ***** |
| Taylor                      | ***** |
| Gietz                       | ***** |
| Grant                       | ***** |

10. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY . Order the results by the day of the week starting with Monday.

If you want an extra challenge, complete the following exercises:

11. Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, put "No Commission." Label the column COMM.
12. Create a query that displays the employees' last names and indicates the amounts of their annual salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column EMPLOYEES\_AND\_THEIR\_SALARIES.
13. Using the DECODE function, write a query that displays the grade of all employees based on the value of the column JOB\_ID , as per the following data:

**Job Grade**

A *AD\_PRES*

B *ST\_MAN*

C *IT\_PROG*

D *SA REP*

E *ST\_CLERK*

F None of the above 0

14. Rewrite the statement in the preceding question using the CASE syntax.

## LAB :03

**Q1:**Write a query that displays the employee's last names with the first letter capitalized and all other letters lowercase and the length of the names, for all employees whose name starts with J , A , or M . Give each column an appropriate label. Sort the results by the employees

```
1 -- select * from emp;
2 SELECT INITCAP(ENAME) AS FormattedName, LENGTH(ENAME) AS NameLength
3 FROM EMP
4 WHERE ENAME LIKE 'A%' OR ENAME LIKE 'J%' OR ENAME LIKE 'M%';
5
```

| Results |   | Explain | Describe | Saved SQL | History       |
|---------|---|---------|----------|-----------|---------------|
|         |   |         |          |           | FORMATTEDNAME |
|         |   |         |          |           | NAMELENGTH    |
| Allen   | 5 |         |          |           |               |
| Jones   | 5 |         |          |           |               |
| Martin  | 6 |         |          |           |               |
| Adams   | 5 |         |          |           |               |

**Question 2:**each employee, display the employee's last name, and calculate the number of months between today and the date the employee was hired. Label the column MONTHS\_WORKED . Order your results by the number of months employed. Round the number of months up to the closest whole number.

**Note:**

Your results will differ.

```
1 SELECT ENAME, HIREDATE, ROUND((SYSDATE-HIREDATE)/12) AS "MONTHS_WORKED" FROM EMP;
2
```

| Results |            | Explain | Describe | Saved SQL | History |
|---------|------------|---------|----------|-----------|---------|
|         |            |         |          |           |         |
|         |            | ENAME   |          | HIREDATE  |         |
| SMITH   | 12/17/1980 |         |          | 1343      |         |
| ALLEN   | 2/20/1981  |         |          | 1337      |         |
| WARD    | 2/22/1981  |         |          | 1337      |         |
| JONES   | 4/2/1981   |         |          | 1334      |         |
| MARTIN  | 9/28/1981  |         |          | 1319      |         |
| BLAKE   | 5/1/1981   |         |          | 1334      |         |

**Q3:Write a query that produces the following for each employee:**

**< employee last named earns <salary> monthly but wants <3 times salary >. Label the column Dream Salaries . If you have time, complete the following exercises:**

```
2   SELECT ENAME, SAL, SAL*3 AS "DREAM SALARIES" FROM EMP;
3
```

| Results |       | Explain | Describe       | Saved SQL | History |
|---------|-------|---------|----------------|-----------|---------|
|         | ENAME | SAL     | DREAM SALARIES |           |         |
| SMITH   |       | 800     | 2400           |           |         |
| ALLEN   |       | 1600    | 4800           |           |         |
| WARD    |       | 1250    | 3750           |           |         |
| JONES   |       | 2975    | 8925           |           |         |
| MARTIN  |       | 1250    | 3750           |           |         |
| BLAKE   |       | 2850    | 8550           |           |         |

**Q4:Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with \$. Label the column SALARY**

```
3   SELECT ENAME, LPAD(SAL, 15, '$') AS "SALARY" FROM EMP;
4
```

| Results |       | Explain                         | Describe | Saved SQL | History |
|---------|-------|---------------------------------|----------|-----------|---------|
|         | ENAME | SALARY                          |          |           |         |
| SMITH   |       | \$\$\$\$\$\$\$\$\$\$\$\$\$\$800 |          |           |         |
| ALLEN   |       | \$\$\$\$\$\$\$\$\$\$\$\$\$1600  |          |           |         |
| WARD    |       | \$\$\$\$\$\$\$\$\$\$\$\$\$1250  |          |           |         |
| JONES   |       | \$\$\$\$\$\$\$\$\$\$\$\$\$2975  |          |           |         |
| MARTIN  |       | \$\$\$\$\$\$\$\$\$\$\$\$\$1250  |          |           |         |
| BLAKE   |       | \$\$\$\$\$\$\$\$\$\$\$\$\$2850  |          |           |         |

**Q5: Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear similar to “Monday, the Thirty-First of July, 2000.”.**

```
5   SELECT HIREDATE, TO_CHAR(NEXT_DAY(ADD_MONTHS(HIREDATE, 6), 'MONDAY'), 'Day, "the" FMDDth "of" Month, YYYY') AS "INC_MONTH" FROM EMP;
```

Results Explain Describe Saved SQL History

| HIREDATE   | INC_MONTH                         |
|------------|-----------------------------------|
| 12/17/1980 | Monday , the 22ND of June, 1981   |
| 2/20/1981  | Monday , the 24TH of August, 1981 |
| 2/22/1981  | Monday , the 24TH of August, 1981 |

**Q6:Display the last name, hire date, and day of the week on which the employee started. Label the column DAY . Order the results by the day of the week starting with Monday.**

```
6   SELECT ENAME, HIREDATE, TO_CHAR(HIREDATE, 'Day') AS "DAY" FROM EMP ORDER BY MOD(TO_CHAR(HIREDATE, 'D') + 5, 7);
```

Results Explain Describe Saved SQL History

| ENAME  | HIREDATE   | DAY     |
|--------|------------|---------|
| MARTIN | 9/28/1981  | Monday  |
| KING   | 11/17/1981 | Tuesday |
| SCOTT  | 9/8/1981   | Tuesday |

**Q7:Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, put "No Commission." Label the column COMM.**

```
2   SELECT EMPNO, ENAME, NVL(TO_CHAR(COMM), 'NO COMMISSION') AS "COMM" FROM EMP;
```

Results Explain Describe Saved SQL History

| EMPNO | ENAME  | COMM          |
|-------|--------|---------------|
| 7369  | SMITH  | NO COMMISSION |
| 7499  | ALLEN  | 300           |
| 7521  | WARD   | 500           |
| 7566  | JONES  | NO COMMISSION |
| 7654  | MARTIN | 1400          |

**Q8:**Create a query that displays the employees' last names and indicates the amounts of their annual salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column **EMPLOYEES AND THEIR SALARIES**.

```
13  SELECT
14    ENAME,
15    RPAD('*', FLOOR(SAL / 1000), '*') AS EMPLOYEES_AND THEIR_SALARIES
16  FROM EMP
17  ORDER BY SAL DESC;
18
```

| Results | Explain | Describe                     | Saved SQL | History |
|---------|---------|------------------------------|-----------|---------|
| ENAME   |         | EMPLOYEES_AND_THEIR_SALARIES |           |         |
| ASIYA   |         | -                            |           |         |
| KING    |         | *****                        |           |         |
| FORD    |         | ***                          |           |         |
| SCOTT   |         | ***                          |           |         |

**Q9:** Using the DECODE function, write a query that displays the grade of all employees based on the value of the column JOB ID , as per the following data:

```
4  SELECT EMPNO, ENAME,
5    DECODE(JOB,
6      'CLERK', 'C',
7      'SALESMAN', 'S',
8      'MANAGER', 'M',
9      'ANALYST', 'A',
10     'PRESIDENT', 'P') AS "JOB_TITLE"
11   FROM EMP;
```

| Results | Explain | Describe | Saved SQL | History   |
|---------|---------|----------|-----------|-----------|
|         |         | EMPNO    | ENAME     | JOB_TITLE |
| 7369    |         | SMITH    |           | C         |
| 7499    |         | ALLEN    |           | S         |
| 7521    |         | WARD     |           | S         |

## 4-Displaying Data from Multiple Tables

### Objectives

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

- Write SELECT statements to access data from more than one table using equality and nonequality joins
- View data that generally does not meet a join condition by using outer joins
- Join a table to itself by using a self join

### Obtaining Data from Multiple Tables

#### EMPLOYEES DEPARTMENTS

Sometimes you need to use data from more than one table. In the example, the report displays data from two separate tables.

- Employee IDs exist in the EMPLOYEES table.
- Department IDs exist in both the EMPLOYEES and DEPARTMENTS tables.
- Location IDs exist in the DEPARTMENTS table.

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

### Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition in a WHERE clause.

Cartesian Products

### Generating a Cartesian Product

**EMPLOYEES**

**(20 rows)**

**DEPARTMENTS**

**(8 rows)**

**Cartesian**

**product:**

**20x8=160 rows**

`SELECT last_name, department_name dept_name`

FROM employees, departments;

### **Types of Joins**

Oracle Proprietary SQL: 1999

Compliant Joins: Joins (8i and prior):

- Equijoin
  - Cross joins
  - Nonequijoin
  - Natural joins
  - Outer join
- Using clause

- Self join
- Full or two sided outer joins
- Arbitrary join conditions for outer joins

### **Joining Tables Using Oracle Syntax**

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

**SELECT**

**table1.column, table2.column**

**FROM**

**table1, table2**

**WHERE**

**table1.column1 = table2.column2;**

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

### **What Is an Equijoin?**

## **EMPLOYEES DEPARTMENTS**

### **Foreign key Primary key**

### **Equijoins**

To determine an employee's department name, you compare the value in the DEPARTMENT\_ID column in the EMPLOYEES table with the DEPARTMENT\_ID values in the DEPARTMENTS table. The relationship between the EMPLOYEES and DEPARTMENTS tables is an *equijoin*, that is, values in the DEPARTMENT\_ID column on both tables must be equal. Frequently, this type of join involves primary and foreign key complements.

**Note:** Equijoins are also called *simple joins* or *inner joins*

.

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_ID | LOCATION_ID |
|-------------|-----------|---------------|---------------|-------------|
| 200         | Whalen    | 10            | 10            | 1700        |
| 201         | Hartstein | 20            | 20            | 1800        |
| 202         | Fay       | 20            | 20            | 1800        |
| 124         | Mourgos   | 50            | 50            | 1500        |
| 141         | Rajs      | 50            | 50            | 1500        |
| 142         | Davies    | 50            | 50            | 1500        |
| 143         | Matos     | 50            | 50            | 1500        |
| 205         | Higgins   | 110           | 110           | 1700        |
| 206         | Gietz     | 110           | 110           | 1700        |

19 rows selected.

## Retrieving Records with Equijoins

```
SELECT employees.employee_id, employees.last_name,
employees.department_id, departments.department_id,
departments.location_id
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

## Additional Search Conditions

### Using the AND Operator

#### EMPLOYEES DEPARTMENTS

```
SELECT last_name, employees.department_id,
department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND last_name = 'Matos';
```

### Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Improve performance by using table prefixes.
- Distinguish columns that have identical names but reside in different tables by using column aliases.

### Using Table Aliases

```
•Simplify queries by using table aliases
•Improve performance by using table prefixes
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

## Joining More than Two Tables

EMPLOYEES

DEPARTMENTS

LOCATIONS

To join n tables together, you need a minimum of  
n -1

join conditions. For example, to join three tables, a  
minimum of two joins is required.

```
SELECT e.last_name, d.department_name, l.city
FROM employees e, departments d, locations l
WHERE e.department_id = d.department_id
AND d.location_id = l.location_id;
```

## Nonequijoins

EMPLOYEES JOB\_GRADES

Salary in the EMPLOYEES

table must be between

lowest salary and highest

salary in the JOB\_GRADES table.

## Retrieving Records with Nonequijoins

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

## Outer Joins

DEPARTMENTS EMPLOYEES

There are no employees in

department 190.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

## Outer Joins Syntax

- You use an outer join to also see rows that do not meet the join condition.

- The outer join operator is the plus sign (+).

```
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column(+) = table2.column;
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column = table2.column(+);
```

## Using Outer Joins

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id(+) = d.department_id;
```

### **Self Joins**

**EMPLOYEES (WORKER) EMPLOYEES (MANAGER)**  
**MANAGER\_ID in the WORKER table is equal to**  
**EMPLOYEE\_ID in the MANAGER**  
**table.**

### **Joining a Table to Itself**

```
SELECT worker.last_name || ' works for '
|| manager.last_name
FROM employees worker, employees manager
WHERE worker.manager_id = manager.employee_id;
```

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

```
SELECT table1.column, table2.column
FROM table1
[CROSS JOIN table2] | [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 )];
```

### **Creating Cross Joins**

- The CROSS JOIN clause produces the cross-product of two tables.
- This is the same as a Cartesian product between the two tables.

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

### **Creating Cross Joins**

The example in the slide gives the same results as the following:

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

### **Creating Natural Joins**

- The NATURAL JOIN clause is based on all columns in the two tables that have the same name.
- 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, then an error is returned.

Note: The join can happen only on columns having the same names and data types in both the tables. If the columns have the same name, but different data types, then the

NATURAL JOIN syntax causes an error.

### **Retrieving Records with Natural Joins**

```
SELECT department_id, department_name,
location_id, city
FROM departments
NATURAL JOIN locations;
```

```
SELECT department_id, department_name,
departments.location_id, city
FROM departments, locations
WHERE departments.location_id = locations.location_id;
```

```
SELECT department_id, department_name,
location_id, city
FROM departments
NATURAL JOIN locations
WHERE department_id IN (20, 50);
```

### **Creating Joins with the USING Clause**

- If several columns have the same names but the data types do not match, the NATURAL JOIN clause can be modified with the USING clause to specify the columns that should be used for an equijoin.

Note: Use the USING clause to match only one column when more than one column matches.

- Do not use a table name or alias in the referenced columns.
- The NATURAL JOIN and USING clauses are mutually exclusive.

For example, this statement is valid:

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

This statement is invalid because the  
**LOCATION\_ID**

is qualified in the where clause:

```
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  
The same restriction applies to NATURAL  
joins also. Therefore columns that have the same name in both

tables have to be used without any qualifiers.

### **Retrieving Records with the USING Clause**

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

```
SELECT employee_id, last_name,
employees.department_id, location_id
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

### **Creating Joins with the**

#### **ON Clause**

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

```
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);
```

### **INNER versus OUTER Joins**

- In SQL: 1999, the join of two tables returning only matched rows is an inner join.
- A join between two tables that returns the results of the inner join as well as unmatched rows left (or right) tables is 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.

Joins: Comparing SQL: 1999 to Oracle Syntax

### **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);
```

Example of

### **LEFT OUTER JOIN**

This query retrieves all rows in the EMPLOYEES

table, which is the left table even if there is no match in the DEPARTMENTS table.

This query was completed in earlier releases as follows:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id (+) = e.department_id;
```

### **RIGHT OUTER JOIN**

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

#### **Example of**

### **RIGHT OUTER JOIN**

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

This query was completed in earlier releases as follows:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id = e.department_id (+);
```

### **FULL OUTER JOIN**

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

#### **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;
```

|         |     |            |
|---------|-----|------------|
| Higgins | 110 | Accounting |
| Gietz   | 110 | Accounting |

19 rows selected.

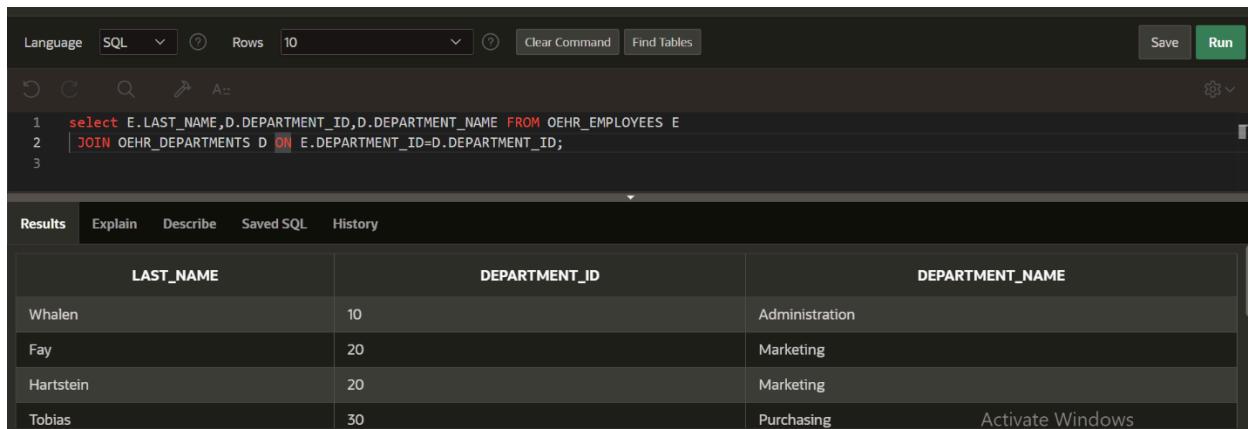
### Practice 3, Part 1

1. Write a query to display the last name, department number, and department name for all employees.
2. Create a unique listing of all jobs that are in department 30. Include the location of department 90 in the output.
3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.
4. Display the employee last name and department name for all employees who have an *a* (lowercase) in their last names. Place your SQL statement in a text file named lab4\_4.sql.
5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.
6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns Employee , Emp# , Manager , and Mgr# , respectively.  
Place your SQL statement in a text file named lab4\_6.sql.
7. Modify lab4\_6.sql to display all employees including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named lab4\_7.sql . Run the query in lab4\_7.sql.  
If you have time, complete the following exercises:
  8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label.
  9. Show the structure of the JOB\_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees.  
If you want an extra challenge, complete the following exercises:
    10. Create a query to display the name and hire date of any employee hired after employee Davies.
    11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee , Emp Hired , Manager , and Mgr Hired , respectively.

## LAB:04

### QUESTION:01

Write a query to display the last name, department number, and department name for all employees.



The screenshot shows a SQL query being run in a database interface. The query is:

```

1 select E.LAST_NAME,D.DEPARTMENT_ID,D.DEPARTMENT_NAME FROM OEHR_EMPLOYEES E
2 JOIN OEHR_DEPARTMENTS D ON E.DEPARTMENT_ID=D.DEPARTMENT_ID;
3

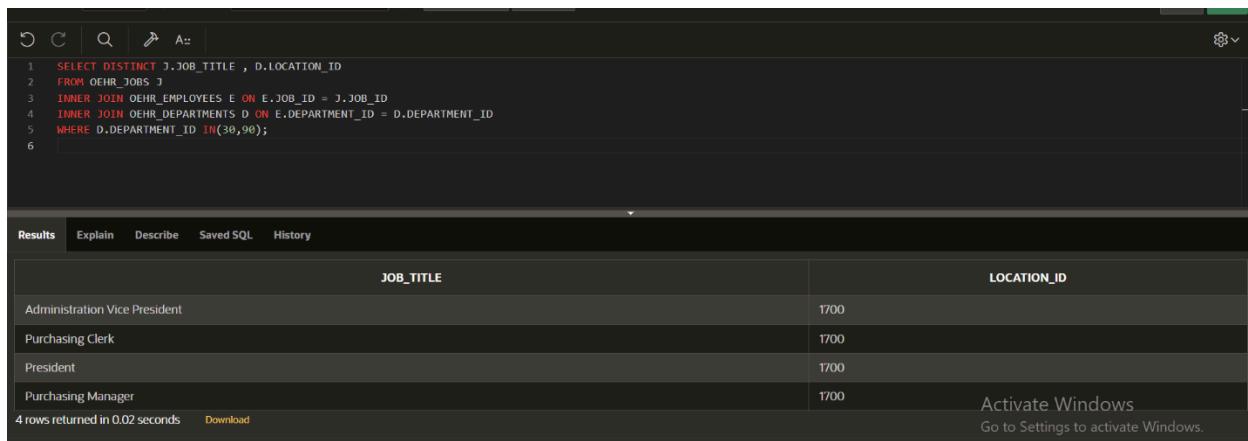
```

The results table has three columns: LAST\_NAME, DEPARTMENT\_ID, and DEPARTMENT\_NAME. The data is:

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----------|---------------|-----------------|
| Whalen    | 10            | Administration  |
| Fay       | 20            | Marketing       |
| Hartstein | 20            | Marketing       |
| Tobias    | 30            | Purchasing      |

### QUESTION:2

Create a unique listing of all jobs that are in department 30. Include the location of department 90 in the output



The screenshot shows a SQL query being run in a database interface. The query is:

```

1 SELECT DISTINCT J.JOB_TITLE , D.LOCATION_ID
2 FROM OEHR_JOBS J
3 INNER JOIN OEHR_EMPLOYEES E ON E.JOB_ID = J.JOB_ID
4 INNER JOIN OEHR_DEPARTMENTS D ON E.DEPARTMENT_ID = D.DEPARTMENT_ID
5 WHERE D.DEPARTMENT_ID IN(30,90);
6

```

The results table has two columns: JOB\_TITLE and LOCATION\_ID. The data is:

| JOB_TITLE                     | LOCATION_ID |
|-------------------------------|-------------|
| Administration Vice President | 1700        |
| Purchasing Clerk              | 1700        |
| President                     | 1700        |
| Purchasing Manager            | 1700        |

4 rows returned in 0.02 seconds    Download

Activate Windows  
Go to Settings to activate Windows.

### QUESTION: 3

Write a query to display the employee's last name, department name, location ID, and city of all employees who earn a commission

```

Language SQL Rows 10 Clear Command Find Tables Save Run
1 SELECT E.last_name, D.department_name, L.location_id, L.city FROM OEHR_EMPLOYEES E
2 INNER JOIN OEHR_DEPARTMENTS D ON E.department_id = D.department_id
3 INNER JOIN OEHR_LOCATIONS L ON L.location_id = D.location_id
4 WHERE E.commission_pct IS NOT NULL;

```

| Results                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Explain         | Describe        | Saved SQL        | History |         |       |      |        |          |       |      |        |           |       |      |        |     |  |      |                  |  |  |  |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|-----------------|------------------|---------|---------|-------|------|--------|----------|-------|------|--------|-----------|-------|------|--------|-----|--|------|------------------|--|--|--|--|
| <table border="1"> <thead> <tr> <th>LAST_NAME</th> <th>DEPARTMENT_NAME</th> <th>LOCATION_ID</th> <th>CITY</th> </tr> </thead> <tbody> <tr> <td>Russell</td> <td>Sales</td> <td>2500</td> <td>Oxford</td> </tr> <tr> <td>Partners</td> <td>Sales</td> <td>2500</td> <td>Oxford</td> </tr> <tr> <td>Errazuriz</td> <td>Sales</td> <td>2500</td> <td>Oxford</td> </tr> <tr> <td>...</td> <td></td> <td>----</td> <td>Activate Windows</td> </tr> </tbody> </table> | LAST_NAME       | DEPARTMENT_NAME | LOCATION_ID      | CITY    | Russell | Sales | 2500 | Oxford | Partners | Sales | 2500 | Oxford | Errazuriz | Sales | 2500 | Oxford | ... |  | ---- | Activate Windows |  |  |  |  |
| LAST_NAME                                                                                                                                                                                                                                                                                                                                                                                                                                                       | DEPARTMENT_NAME | LOCATION_ID     | CITY             |         |         |       |      |        |          |       |      |        |           |       |      |        |     |  |      |                  |  |  |  |  |
| Russell                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Sales           | 2500            | Oxford           |         |         |       |      |        |          |       |      |        |           |       |      |        |     |  |      |                  |  |  |  |  |
| Partners                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Sales           | 2500            | Oxford           |         |         |       |      |        |          |       |      |        |           |       |      |        |     |  |      |                  |  |  |  |  |
| Errazuriz                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Sales           | 2500            | Oxford           |         |         |       |      |        |          |       |      |        |           |       |      |        |     |  |      |                  |  |  |  |  |
| ...                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                 | ----            | Activate Windows |         |         |       |      |        |          |       |      |        |           |       |      |        |     |  |      |                  |  |  |  |  |

#### Question :4

Display the employee last name and department name for all employees who have an a (lowercase) in their last names.

```

SELECT E.last_name, D.department_name FROM OEHR_EMPLOYEES E
INNER JOIN OEHR_DEPARTMENTS D ON E.department_id = D.department_id
WHERE E.last_name LIKE '%a%';

```

| Results                                                                                                                                                                                                                                                                                                   | Explain         | Describe        | Saved SQL | History        |     |           |           |           |     |      |  |  |  |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|-----------------|-----------|----------------|-----|-----------|-----------|-----------|-----|------|--|--|--|--|
| <table border="1"> <thead> <tr> <th>LAST_NAME</th> <th>DEPARTMENT_NAME</th> </tr> </thead> <tbody> <tr> <td>Whalen</td> <td>Administration</td> </tr> <tr> <td>Fay</td> <td>Marketing</td> </tr> <tr> <td>Hartstein</td> <td>Marketing</td> </tr> <tr> <td>...</td> <td>----</td> </tr> </tbody> </table> | LAST_NAME       | DEPARTMENT_NAME | Whalen    | Administration | Fay | Marketing | Hartstein | Marketing | ... | ---- |  |  |  |  |
| LAST_NAME                                                                                                                                                                                                                                                                                                 | DEPARTMENT_NAME |                 |           |                |     |           |           |           |     |      |  |  |  |  |
| Whalen                                                                                                                                                                                                                                                                                                    | Administration  |                 |           |                |     |           |           |           |     |      |  |  |  |  |
| Fay                                                                                                                                                                                                                                                                                                       | Marketing       |                 |           |                |     |           |           |           |     |      |  |  |  |  |
| Hartstein                                                                                                                                                                                                                                                                                                 | Marketing       |                 |           |                |     |           |           |           |     |      |  |  |  |  |
| ...                                                                                                                                                                                                                                                                                                       | ----            |                 |           |                |     |           |           |           |     |      |  |  |  |  |

#### Question :5

Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

The screenshot shows a SQL query results window. The query selects employee last names, department names, location IDs, and job titles from four tables: OEHR\_EMPLOYEES, OEHR\_DEPARTMENTS, OEHR\_LOCATIONS, and OEHR\_JOBS, where the location is 'Toronto' and the department is 'Marketing'. The results show two rows: Hartstein (Marketing Manager) and Fay (Marketing Representative).

```

1 SELECT E.LAST_NAME,D.DEPARTMENT_NAME,D.LOCATION_ID, J.JOB_TITLE FROM OEHR_EMPLOYEES E
2 INNER JOIN OEHR_DEPARTMENTS D ON E.DEPARTMENT_ID = D.DEPARTMENT_ID
3 INNER JOIN OEHR_LOCATIONS L ON L.LOCATION_ID = D.LOCATION_ID
4 INNER JOIN OEHR_JOBS J ON J.JOB_ID=E.JOB_ID
5 WHERE L.CITY='Toronto';

```

| LAST_NAME | DEPARTMENT_NAME | LOCATION_ID | JOB_TITLE                |
|-----------|-----------------|-------------|--------------------------|
| Hartstein | Marketing       | 1800        | Marketing Manager        |
| Fay       | Marketing       | 1800        | Marketing Representative |

2 rows returned in 0.01 seconds [Download](#) Activate Windows

### Question 6:

Display the employee last name and employee number along with their manager's last name and manager number.

The screenshot shows a SQL query results window. The query selects employee last names, employee IDs, manager last names, and manager IDs from the OEHR\_EMPLOYEES and OEHR\_EMPLOYEES tables, using a left join to include all employees even if they don't have a manager. The results show five rows: Fay, Gietz, Hartstein, Kochhar, and De Haan, each with their respective manager information.

```

1 SELECT
2     E.LAST_NAME AS "Employee",
3     E.EMPLOYEE_ID AS "Emp#",
4     M.LAST_NAME AS "Manager",
5     M.MANAGER_ID AS Mgr#
6 FROM OEHR_EMPLOYEES E
7 LEFT JOIN OEHR_EMPLOYEES M ON E.MANAGER_ID = M.EMPLOYEE_ID;

```

| Employee  | Emp# | Manager   | MGR# |
|-----------|------|-----------|------|
| Fay       | 202  | Hartstein | 100  |
| Gietz     | 206  | Higgins   | 101  |
| Hartstein | 201  | King      | -    |
| Kochhar   | 101  | King      |      |
| De Haan   | 102  | King      |      |

Activate Windows  
Go to Settings to activate Windows.

### Question 7:

Display the employee last name and employee number along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively.

```

1 SELECT
2     E.LAST_NAME AS "Employee",
3     E.EMPLOYEE_ID AS "Emp#",
4     M.LAST_NAME AS "Manager",
5     M.MANAGER_ID AS Mgr#
6 FROM OEHR_EMPLOYEES E
7 LEFT JOIN OEHR_EMPLOYEES M ON E.MANAGER_ID = M.EMPLOYEE_ID
8 ORDER BY E.EMPLOYEE_ID;
9

```

The screenshot shows a SQL query in a code editor. The query performs a self-join on the OEHR\_EMPLOYEES table to find managers and their employees. The results are displayed in a table with four columns: Employee, Emp#, Manager, and MGR#. The table contains five rows of data.

| Employee | Emp# | Manager | MGR# |
|----------|------|---------|------|
| King     | 100  | -       | -    |
| Kochhar  | 101  | King    | -    |
| De Haan  | 102  | King    | -    |
| Hunold   | 103  | De Haan | 100  |
| Ernst    | 104  | Hunold  | 102  |

### Question 8:

Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label.

```

1 SELECT
2     E.LAST_NAME AS "employee_name",
3     E.DEPARTMENT_ID AS "department_number",
4     E2.LAST_NAME AS "colleague_name"
5 FROM OEHR_EMPLOYEES E
6 JOIN OEHR_EMPLOYEES E2
7     ON E.DEPARTMENT_ID = E2.DEPARTMENT_ID
8     AND E.EMPLOYEE_ID != E2.EMPLOYEE_ID;
9

```

The screenshot shows a SQL query in a code editor. The query performs a self-join on the OEHR\_EMPLOYEES table to find colleagues in the same department as a given employee. The results are displayed in a table with three columns: employee\_name, department\_number, and colleague\_name. The table contains five rows of data.

| employee_name | department_number | colleague_name |
|---------------|-------------------|----------------|
| Grant         | 50                | OConnell       |
| Weiss         | 50                | OConnell       |
| Fripp         | 50                | OConnell       |
| Kaufling      | 50                | OConnell       |
| Vollman       | 50                | OConnell       |

### Question 9:

Show the structure of the JOB\_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees

The screenshot shows a SQL query being run in Oracle SQL Developer. The query retrieves employee details including last name, job title, department name, salary, and job ID. The results are displayed in a grid:

| LAST_NAME  | JOB_TITLE                | DEPARTMENT_NAME | SALARY | JOB_ID   |
|------------|--------------------------|-----------------|--------|----------|
| Whalen     | Administration Assistant | Administration  | 4400   | AD_ASST  |
| Hartstein  | Marketing Manager        | Marketing       | 13000  | MK_MAN   |
| Fay        | Marketing Representative | Marketing       | 6000   | MK_REP   |
| Colmenares | Purchasing Clerk         | Purchasing      | 2500   | PU_CLERK |
| Khoo       | Purchasing Clerk         | Purchasing      | 3100   | PU_CLERK |
| Baida      | Purchasing Clerk         | Purchasing      | 2900   | PU_CLERK |
| Tobias     | Purchasing Clerk         | Purchasing      | 2800   | PU_CLERK |

### Question 10:

Create a query to display the name and hire date of any employee hired after employee Davies.

The screenshot shows a SQL query being run in Oracle SQL Developer. The query selects employee names and hire dates from employees hired after Davies. The results show 'no data found'.

| NAME          | HIRE_DATE |
|---------------|-----------|
| no data found |           |

### Question 11:

Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp Hired , Manager , and Mgr Hired respectively.

The screenshot shows a SQL query being run in a database environment. The query retrieves information from two tables: OEHR\_EMPLOYEES (Employee) and OEHR\_EMPLOYEES (Manager). It selects the Employee's last name, hire date, and Manager's last name and hire date, where the Employee was hired before their Manager. The results are displayed in a table with four columns: Employee, Emp hired, Manager, and Mgr Hired.

```
1  SELECT
2      E.LAST_NAME AS "Employee",
3      E.HIRE_DATE AS "Emp hired",
4      M.LAST_NAME AS "Manager",
5      M.HIRE_DATE AS "Mgr Hired"
6  FROM OEHR_EMPLOYEES E
7  LEFT JOIN OEHR_EMPLOYEES M ON E.MANAGER_ID = M.EMPLOYEE_ID
8  where E.HIRE_DATE < M.HIRE_DATE ;
9
```

| Employee | Emp hired | Manager   | Mgr Hired                                                                |
|----------|-----------|-----------|--------------------------------------------------------------------------|
| Whalen   | 3/12/2012 | Kochhar   | 3/17/2014                                                                |
| Hunold   | 6/29/2014 | De Haan   | 7/9/2017                                                                 |
| Faviet   | 2/9/2019  | Greenberg | 2/10/2019                                                                |
| Marlow   | 8/12/2021 | Frapp     | 10/4/2021                                                                |
| Sarchand | 7/22/2020 | Frapp     | 10/4/2021 Activate Windows<br>10/4/2021 to Settings to activate Windows. |

## 5-Aggregating Data Using Group Functions

### Objectives

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

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

### What Are Group Functions?

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

**EMPLOYEES**

The maximum salary in the EMPLOYEES table.

### Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- SUM

### Group Functions Syntax

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

### Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

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

### Using the MIN and MAX Functions

You can use MIN and MAX for any data type.

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

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

**Note:** AVG , SUM , VARIANCE , and STDDEV functions can be used only with numeric data types.

| COUNT(*) |
|----------|
| 5        |

### Using the COUNT Function COUNT(\*)

returns the number of rows in a table.

```
SELECT COUNT(*)
```

```
FROM employees
```

```
WHERE department_id = 50;
```

### Using the COUNT Function

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

- Display the number of department values in the EMPLOYEES table, excluding the null values.

```
SELECT COUNT(commission_pct)
```

```
FROM employees
```

```
WHERE department_id = 80;
```

```
SELECT COUNT(department_id)
```

```
FROM employees;
```

### Using the DISTINCT Keyword

- COUNT(DISTINCT expr ) returns the number of distinct nonnull values of the expr .

- Display the number of distinct department values in the EMPLOYEES table.

```
SELECT COUNT(DISTINCT department_id)
```

```
FROM employees;
```

### Group Functions and Null Values

Group functions ignore null values in the column.

```
SELECT AVG(commission_pct)
```

```
FROM employees;
```

### Using the NVL Function

#### with Group Functions

The NVL function forces group functions to include null values.

```
SELECT AVG(NVL(commission_pct, 0))
```

```
FROM employees;
```

**Creating Groups of Data****EMPLOYEES****4400****9500**

The average salary 3500 in EMPLOYEES

Table 6400 for each department.

**Creating Groups of Data:****GROUP BY Clause Syntax****SELECT column , group\_function(column)****FROM table****[WHERE condition ]****[GROUP BY group\_by\_expression ]****[ORDER BY column ];**

Divide rows in a table into smaller groups

**Using the GROUP BY Clause**

All 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;****SELECT department\_id, AVG(salary)****FROM employees****GROUP BY department\_id****ORDER BY AVG(salary);****Grouping by More Than One Column****EMPLOYEES**

Add up the salaries in the

EMPLOYEES table

for each job,

grouped by

department.

## Using the GROUP BY Clause on Multiple Columns

```
SELECT department_id dept_id, job_id, SUM(salary)
FROM employees
GROUP BY department_id, job_id;
```

### Illegal Queries

#### Using Group Functions

Any column or expression in the

SELECT list that is not an aggregate function must be in the GROUP BY clause.

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

**ERROR at line 1:**

**ORA-00937: not a single-group group function**

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

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

### Illegal Queries

#### Using Group Functions

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

```
SELECT department_id, AVG(salary)
```

```
FROM employees
```

```
WHERE AVG(salary) > 8000
```

```
GROUP BY department_id;
```

```
WHERE AVG(salary) > 8000
```

```
*
```

**ERROR at line 3:**

**ORA-00934: group function is not allowed here**

#### Illegal Queries Using Group Functions (continued)

```
SELECT department_id, AVG(salary)
```

```
FROM employees
```

```
HAVING AVG(salary) > 8000
```

```
GROUP BY department_id;
```

### Excluding Group Results

#### EMPLOYEES

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

## **Excluding Group Results: The HAVING Clause**

Use the HAVING clause to restrict groups:

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 ];
```

| DEPARTMENT_ID | Avg(Salary) |
|---------------|-------------|
| 20            | 9500        |
| 80            | 10033.3333  |
| 90            | 19333.3333  |
| 110           | 10150       |

### **Using the HAVING Clause**

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

### **Using the HAVING Clause**

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

| JOB_ID  | PAYROLL |
|---------|---------|
| IT_PROG | 19200   |
| AD_PRES | 24000   |
| AD_VP   | 34000   |

### 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);
```

| MAX(AVG(SALARY)) |
|------------------|
| 19333.3333       |

### Nesting Group Functions

Display the maximum average salary.

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

## Paper-Based Questions

For questions 1 through 3, circle either True or False.

**Note:**

Column aliases are used for the queries.

### Practice 4

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result per group.

True/False

2. Group functions include nulls in calculations.

True/False

3. The WHERE clause restricts rows prior to inclusion in a group calculation

True/False

4. Display the highest, lowest, sum, and average salary of all employees. Label the columns Maximum , Minimum , Sum , and Average , respectively. Round your results to the nearest whole number. Place your SQL statement in a text file named lab5\_6.sql.

5. Modify the query in lab5\_4.sql to display the minimum, maximum, sum, and average salary for each job type. Resave lab5\_4.sql To lab5\_5.sql. Run the statement in lab5\_5.sql.

6. Write a query to display the number of people with the same job.

7. Determine the number of managers without listing them. Label the column Number of Managers .

**Hint:**Use the MANAGER\_ID column to determine the number of managers.

8. Write a query that displays the difference between the highest and lowest salaries. Label the column DIFFERENCE.

If you have time, complete the following exercises:

9. Display the manager number and the salary of the lowest paid employee for that manager.

Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is less than \$6,000. Sort the output in descending order of salary.

10. Write a query to display each department's name, location, number of employees, and the average salary for all employees in that department. Label the columns Name , Location , Number of People , and Salary , respectively. Round the average salary to two decimal places.

If you want an extra challenge, complete the following exercises:

11. Create a query that will display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

12. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

## LAB:05

QUESTION:01

**TRUE**

QUESTION:2

**FALSE**

QUESTION: 3

**FALSE (“HAVING” CLAUSE DO IT)**

Question :4

Display the highest, lowest, sum, and average salary of all employees. Label the columns **Maximum**, **Minimum**, **Sum**, and **Average**, respectively. Round your results to the nearest whole number.

|                                                                                                     |         |        |         |
|-----------------------------------------------------------------------------------------------------|---------|--------|---------|
| 62    SELECT MAX(SALARY) MAXIMUM , MIN(SALARY) MINIMUM, SUM(SALARY) SUM, ROUND(AVG(SALARY)) AVERAGE |         |        |         |
| 63    FROM OEHR_EMPLOYEES;                                                                          |         |        |         |
| Results Explain Describe Saved SQL History                                                          |         |        |         |
| MAXIMUM                                                                                             | MINIMUM | SUM    | AVERAGE |
| 24000                                                                                               | 2100    | 691400 | 6462    |
| rows returned in 0.01 seconds    Download                                                           |         |        |         |

Question :5

Modify the query to display the minimum, maximum, sum, and average salary for each job type..

|                                                                                                             |         |         |       |         |
|-------------------------------------------------------------------------------------------------------------|---------|---------|-------|---------|
| 65    SELECT JOB_ID, MAX(SALARY) MAXIMUM , MIN(SALARY) MINIMUM, SUM(SALARY) SUM, ROUND(AVG(SALARY)) AVERAGE |         |         |       |         |
| 66    FROM OEHR_EMPLOYEES                                                                                   |         |         |       |         |
| 67    GROUP BY JOB_ID;                                                                                      |         |         |       |         |
| Results Explain Describe Saved SQL History                                                                  |         |         |       |         |
| JOB_ID                                                                                                      | MAXIMUM | MINIMUM | SUM   | AVERAGE |
| AD_VP                                                                                                       | 17000   | 17000   | 34000 | 17000   |
| FI_ACCOUNT                                                                                                  | 9000    | 6900    | 39600 | 7920    |
| PU_CLERK                                                                                                    | 3100    | 2500    | 13900 | 2780    |
| CU_CLERK                                                                                                    | 6000    | 3500    | 13700 | 3425    |

Question 6:

DT-23304

Fizza Ahmed

Write a query to display the number of people with the same job.

```
69  SELECT JOB_ID, COUNT(EMPLOYEE_ID) "NUMBER OF PEOPLE"  
70  FROM OEHR_EMPLOYEES  
71  GROUP BY JOB_ID;
```

| Results | Explain | Describe | Saved SQL        | History |
|---------|---------|----------|------------------|---------|
|         |         | JOBS     |                  |         |
|         |         | JOB_ID   | NUMBER OF PEOPLE |         |
|         |         | HR_REP   | 1                |         |
|         |         | IT_PROG  | 5                |         |
|         |         | MK_MAN   | 1                |         |

### Question 7:

Determine the number of managers without listing them. Label the column **Number of Managers**.

**Hint:** Use the **MANAGER\_ID** column to determine the number of managers.

```
73  SELECT COUNT(MANAGER_ID) "NUMBER OF MANAGERS"  
74  FROM OEHR_EMPLOYEES;
```

| Results                       | Explain | Describe | Saved SQL | History |
|-------------------------------|---------|----------|-----------|---------|
| NUMBER OF MANAGERS            |         |          |           |         |
| 106                           |         |          |           |         |
| rows returned in 0.00 seconds |         |          | Download  |         |

### Question 8:

Write a query that displays the difference between the highest and lowest salaries. Label the column **DIFFERENCE**.

```
76  SELECT MAX(SALARY) MAXIMUM , MIN(SALARY) MINIMUM,  
77  MAX(SALARY) - MIN(SALARY) DIFFERENCE  
78  FROM OEHR_EMPLOYEES;
```

| Results                       | Explain                  | Describe | Saved SQL | History    |
|-------------------------------|--------------------------|----------|-----------|------------|
|                               |                          | MAXIMUM  | MINIMUM   | DIFFERENCE |
| 24000                         |                          | 2100     |           | 21900      |
| rows returned in 0.00 seconds | <a href="#">Download</a> |          |           |            |

### Question 9:

Display the manager number and the salary of the lowest paid employee for that manager.

- Exclude anyone whose manager is not known.
- Exclude any groups where the minimum salary is less than \$6,000.
- Sort the output in descending order of salary.

```
80 SELECT MANAGER_ID, MIN(SALARY)
81 FROM OEHR_EMPLOYEES
82 WHERE MANAGER_ID IS NOT NULL
83 GROUP BY MANAGER_ID
84 HAVING MIN(SALARY) >= 6000
85 ORDER BY MIN(SALARY) DESC;
```

| Results | Explain    | Describe | Saved SQL | History     |
|---------|------------|----------|-----------|-------------|
|         | MANAGER_ID |          |           | MIN(SALARY) |
| 102     |            |          |           | 9000        |
| 205     |            |          |           | 8300        |
| 145     |            |          |           | 7000        |
| ...     |            |          |           | 6000        |

#### Question 10:

Write a query to display each department's name, location, number of employees, and the average salary for all employees in that department.

- Label the columns **Name**, **Location**, **Number of People**, and **Salary**, respectively.
- Round the average salary to two decimal places.

```
91 SELECT D.DEPARTMENT_NAME NAME , L.CITY LOCATION , COUNT(E.EMPLOYEE_ID) "NUMBER OF PEOPLE" , ROUND(AVG(SALARY), 2) SALARY
92 FROM OEHR_EMPLOYEES E
93 INNER JOIN OEHR_DEPARTMENTS D ON E.DEPARTMENT_ID = D.DEPARTMENT_ID
94 INNER JOIN OEHR_LOCATIONS L ON D.LOCATION_ID = L.LOCATION_ID
95 GROUP BY D.DEPARTMENT_NAME, L.CITY;
96
```

| Results        | Explain | Describe | Saved SQL        | History |
|----------------|---------|----------|------------------|---------|
|                | NAME    | LOCATION | NUMBER OF PEOPLE | SALARY  |
| Executive      | Seattle | 3        | 10333.33         |         |
| Sales          | Oxford  | 34       | 8955.88          |         |
| Marketing      | Toronto | 2        | 9500             |         |
| Administration | Seattle | 1        | 4400             |         |

#### Question 11:

Create a query that will display the total number of employees and, of that total, the number of employees hired in **1995, 1996, 1997, and 1998**.

- Create appropriate column headings.

```

97  SELECT COUNT(*) AS "TOTAL EMPLOYEES", TO_CHAR(HIRE_DATE, 'YYYY')
98  FROM OEHR_EMPLOYEES
99  WHERE TO_CHAR(HIRE_DATE, 'YYYY') IN(2015, 2016, 2017, 2018)
100 GROUP BY TO_CHAR(HIRE_DATE, 'YYYY');
101
102
103

```

| Results         | Explain | Describe | Saved SQL | History                   |
|-----------------|---------|----------|-----------|---------------------------|
| TOTAL EMPLOYEES |         |          |           | TO_CHAR(HIRE_DATE,'YYYY') |
| 4               |         |          |           | 2018                      |
| 1               |         |          |           | 2017                      |
| 1               |         |          |           | 2015                      |

3 rows returned in 0.01 seconds. [Download](#)

### Question 12:

Create a matrix query to display:

- The job
- The salary for that job based on department number
- The total salary for that job
- Only include departments **20, 50, 80, and 90**
- Give each column an appropriate heading.

```

11  SELECT JOB_ID, AVG(SALARY) AS "AVG_SALARY", SUM(SALARY) AS
12  "TOTAL_SALARY"
13  FROM OEHR_EMPLOYEES
14  WHERE DEPARTMENT_ID IN (20, 50, 80, 90)
15  GROUP BY JOB_ID;

```

| Results  | Explain | Describe   | Saved SQL | History      |
|----------|---------|------------|-----------|--------------|
| JOB_ID   |         | AVG_SALARY |           | TOTAL_SALARY |
| AD_VP    | 17000   |            |           | 34000        |
| SH_CLERK | 3215    |            |           | 64300        |
| ST_CLERK | 2785    |            |           | 55700        |
| SA MAN   | 12200   |            |           | 61000        |

## 6- Subqueries

### Objectives

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

- Describe the types of problem that subqueries can solve
- Define subqueries
- List the types of subqueries
- Write single-row and multiple-row subqueries

### Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?

Main Query: Which employees have salaries greater than Abel's salary?

Subquery: ?

What is Abel's salary?

### Subquery Syntax

**SELECT select\_list**

**FROM table**

**WHERE expr operator**

**(SELECT select\_list**

**FROM table );**

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

| LAST_NAME |
|-----------|
| King      |
| Kochhar   |
| De Haan   |
| Hartstein |
| Higgins   |

output is used to complete the query condition for the main or outer query.

### Using a Subquery

**SELECT last\_name**

**FROM employees**

**11000**

**WHERE salary >**

**(SELECT salary**

**FROM employees**

**WHERE last\_name = 'Abel');**

## Types of Subqueries

- Single-row subquery Main query returns ST\_CLERK Subquery
- Multiple-row subquery Main query ST\_CLERK returns Subquery

| LAST_NAME | JOB_ID   |
|-----------|----------|
| Rajs      | ST_CLERK |
| Davies    | ST_CLERK |
| Matos     | ST_CLERK |
| Vargas    | ST_CLERK |

SA\_MAN

## 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             |

### Example

Display the employees whose job ID is the same as that of employee 141.

```
SELECT last_name, job_id FROM employees WHERE job_id =
(SELECT job_id FROM employees WHERE employee_id = 141);
```

| LAST_NAME | JOB_ID   | SALARY |
|-----------|----------|--------|
| Rajs      | ST_CLERK | 3600   |
| Davies    | ST_CLERK | 3100   |

## Executing Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id =
ST_CLERK
(SELECT job_id
FROM employees
WHERE employee_id = 141)
AND salary >
2600
```

```
(SELECT salary
FROM employees
WHERE employee_id = 143);
```

| LAST_NAME | JOB_ID   | SALARY |
|-----------|----------|--------|
| Margas    | ST_CLERK | 2500   |

### Using Group Functions in a Subquery

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

| DEPARTMENT_ID | MIN(SALARY) |
|---------------|-------------|
| 10            | 4400        |
| 20            | 6000        |
|               | 7000        |

7 rows selected.

### The HAVING Clause with Subqueries

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

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

### Example

Find the job with the lowest average salary.

```
SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) = (SELECT MIN(AVG(salary))
FROM employees
GROUP BY job_id);
```

**What Is Wrong with This Statement?**

```

SELECT employee_id, last_name
FROM employees
WHERE salary =
(SELECT MIN(salary)
FROM employees
GROUP BY department_id);
ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row

```

**Will This Statement Return Rows?**

```

SELECT last_name, job_id
FROM employees
WHERE job_id =
(SELECT job_id
FROM employees
WHERE last_name = 'Haas');
no rows selected

```

**Multiple-Row Subqueries**

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

| <b>Operator</b> | <b>Meaning</b>                                              |
|-----------------|-------------------------------------------------------------|
| IN              | <b>Equal to any member in the list</b>                      |
| ANY             | <b>Compare value to each value returned by the subquery</b> |
|                 | <b>Compare value to every value returned</b>                |
| ALL             | <b>by the subquery</b>                                      |

```

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

```

**Example**

Find the employees who earn the same salary as the minimum salary for each department. The inner query is executed first, producing a query result. The main query block is then processed and uses the values returned by the inner query to complete its search condition. In fact, the main query would

look like the following to the Oracle Server:

```

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

```

**Using the ANY Operator  
in Multiple-Row Subqueries**

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

| EMPLOYEE_ID | LAST_NAME | JOB_ID   | SALARY |
|-------------|-----------|----------|--------|
| 141         | Rajs      | ST_CLERK | 3500   |
| 142         | Davies    | ST_CLERK | 3100   |
| 143         | Matos     | ST_CLERK | 2800   |
| 144         | Margas    | ST_CLERK | 2500   |

### **Using the ALL Operator in Multiple-Row Subqueries**

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL
9 00 0 , 6 0 00 , 42 0 0
(SELECT salary
FROM employees
WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

### **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);
no rows selected
```

### Practice 5

1. Write a query to display the last name and hire date of any employee in the same department as Zlotkey. Exclude Zlotkey.
2. Create a query to display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in ascending order of salary.
3. Write a query that displays the employee numbers and last names of all employees who work in a department with any employee whose last name contains a *u*. Place your SQL statement in a text file named lab6\_3.sql. Run your query.
4. Display the last name, department number, and job ID of all employees whose department location ID is 1700.
5. Display the last name and salary of every employee who reports to King.
6. Display the department number, last name, and job ID for every employee in the Executive department.

If you have time, complete the following exercises:

7. Modify the query in lab6\_3.sql to display the employee numbers, last names, and salaries of all employees who earn more than the average salary and who work in a department with any employee with a *u* in their name. Resave lab6\_3.sql to lab6\_7.sql . Run the statement in lab6\_7.sql.

## LAB:06

### QUESTION:01

Write a query to display the last name and hire date of any employee in the same department as Zlotkey. Exclude Zlotkey.

```

1  SELECT LAST_NAME, HIRE_DATE
2  FROM OEHRS.EMPLOYEES
3  WHERE DEPARTMENT_ID = (
4    SELECT DEPARTMENT_ID
5    FROM OEHRS.EMPLOYEES
6    WHERE LAST_NAME = 'Zlotkey')
7  AND LAST_NAME <> 'Zlotkey';

```

| LAST_NAME | HIRE_DATE |
|-----------|-----------|
| Russell   | 4/3/2021  |
| Partners  | 7/8/2021  |
| Erazuriz  | 9/10/2021 |
| Cambrault | 4/16/2024 |

### QUESTION:2

Create a query to display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in ascending order of salary.

```

1  SELECT EMPLOYEE_ID , LAST_NAME , SALARY
2  FROM OEHRS.EMPLOYEES
3  WHERE SALARY > (
4    SELECT AVG(SALARY) FROM OEHRS.EMPLOYEES
5  )
6  ORDER BY SALARY;
7

```

| EMPLOYEE_ID | LAST_NAME | SALARY |
|-------------|-----------|--------|
| 123         | Vollman   | 6500   |
| 205         | Mavris    | 6500   |
| 165         | Lee       | 6800   |
| 113         | Popp      | 6900   |

### QUESTION: 3

Write a query that displays the employee numbers and last names of all employees who work in a department with any employee whose last name contains a u.

```

1  SELECT EMPLOYEE_ID , LAST_NAME
2  FROM OEHR_EMPLOYEES
3  WHERE DEPARTMENT_ID IN (
4      SELECT DEPARTMENT_ID
5      FROM OEHR_EMPLOYEES
6      WHERE LAST_NAME LIKE '%u%'
7  )
8  ORDER BY SALARY;

```

| Results |     | Explain     | Describe | Saved SQL  | History |
|---------|-----|-------------|----------|------------|---------|
|         |     | EMPLOYEE_ID |          | LAST_NAME  |         |
|         | 132 |             |          | Olson      |         |
|         | 128 |             |          | Markle     |         |
|         | 136 |             |          | Philtanker |         |
|         | 135 |             |          | Gee        |         |

#### Question :4

Display the last name, department number, and job ID of all employees whose department location ID is 1700.

```

1  SELECT LAST_NAME , DEPARTMENT_ID, JOB_ID
2  FROM OEHR_EMPLOYEES
3  WHERE DEPARTMENT_ID IN (
4      SELECT DEPARTMENT_ID
5      FROM OEHR_DEPARTMENTS
6      WHERE LOCATION_ID = 1700
7  );
8

```

| Results |           | Explain   | Describe      | Saved SQL | History |
|---------|-----------|-----------|---------------|-----------|---------|
|         |           | LAST_NAME | DEPARTMENT_ID | JOB_ID    |         |
|         | King      |           | 90            |           | AD_PRES |
|         | Kochhar   |           | 90            |           | AD_VP   |
|         | De Haan   |           | 90            |           | AD_VP   |
|         | Greenberg |           | 100           |           | FI_MGR  |

#### Question :5

Display the last name and salary of every employee who reports to King.

```

1  SELECT LAST_NAME , SALARY
2  FROM OEHR_EMPLOYEES
3  WHERE MANAGER_ID = ANY (
4      SELECT EMPLOYEE_ID
5      FROM OEHR_EMPLOYEES
6      WHERE LAST_NAME = 'King'
7  );
8

```

| Results |          | Explain   | Describe | Saved SQL | History |
|---------|----------|-----------|----------|-----------|---------|
|         |          | LAST_NAME |          | SALARY    |         |
|         | Kochhar  |           |          | 17000     |         |
|         | De Haan  |           |          | 17000     |         |
|         | Raphaely |           |          | 11000     |         |
|         | Weiss    |           |          | 8000      |         |

**Question 6:**

Display the department number, last name, and job ID for every employee in the **Executive** department.

```

1  SELECT DEPARTMENT_ID, LAST_NAME, JOB_ID
2  FROM OEHR_EMPLOYEES
3  WHERE DEPARTMENT_ID = (
4      SELECT DEPARTMENT_ID
5      FROM OEHR_DEPARTMENTS
6      WHERE DEPARTMENT_NAME = 'Executive'
7  );
8

```

| Results       |           |         |
|---------------|-----------|---------|
| DEPARTMENT_ID | LAST_NAME | JOB_ID  |
| 90            | King      | AD_PRES |
| 90            | Kochhar   | AD_VP   |
| 90            | De Haan   | AD_VP   |

3 rows returned in 0.02 seconds    [Download](#)

**Question 7:**

Modify the query to display the employee numbers, last names, and salaries of all employees who earn more than the average salary and who work in a department with any employee with a **u** in their name.

```

1  SELECT EMPLOYEE_ID , LAST_NAME , SALARY
2  FROM OEHR_EMPLOYEES
3  WHERE SALARY > (
4      SELECT AVG(SALARY)
5      FROM OEHR_EMPLOYEES
6      WHERE DEPARTMENT_ID = ANY (
7          SELECT DEPARTMENT_ID
8          FROM OEHR_EMPLOYEES
9          WHERE LAST_NAME LIKE '%u%'
10     )
11   )
12 ORDER BY SALARY;

```

| Results     |           |        |
|-------------|-----------|--------|
| EMPLOYEE_ID | LAST_NAME | SALARY |
| 124         | Mougos    | 5800   |
| 202         | Fay       | 6000   |
| 104         | Ernst     | 6000   |
| 173         | Kumar     | 6100   |
| 167         | Banda     | 6200   |

## 7-Manipulating Data Objectives

### **Objective:**

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

- Describe each DML statement
- Insert rows into a table
- Update rows in a table
- Delete rows from a table
- Merge rows in a table
- Control transactions

### **Data Manipulation Language**

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

### **Adding a New Row to a Table**

New row DEPARTMENTS

Insert a new row into the  
DEPARMENTS  
table.

### **The INSERT Statement Syntax**

- Add new rows to a table by using the INSERT statement.
- ```
INSERT INTO table[(  
column[, column...])]  
VALUES(value[, value...]);
```
- Only one row is inserted at a time with this syntax.

### **Adding a New Row to a Table (continued)**

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

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

## 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 created.**
- 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 created.**

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

**INSERT INTO departments**  
**VALUES (100, 'Finance', NULL, NULL);**  
**1 row created.**

EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
113	Popp	AC_ACCOUNT	12-MAR-01	

## 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, 100);**  
**1 row created.**

## Confirming Additions to the Table

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

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

## Inserting Specific Date Values

- Add a new employee.

```
INSERT INTO employees
VALUES (114,
'Den', 'Raphealy',
'DRAPHEAL', '515.127.4561',
TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
'AC_ACCOUNT', 11000, NULL, 100, 30);
1 row created.
```

- Verify your addition.

```
INSERT INTO departments
(department_id, department_name, location_id)
VALUES (&department_id, '&department_name', &location);
40
Human Resources
2500
1 row created.
```

## Copying Rows from Another Table

- Write your INSERT statement with a subquery.

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
4 rows created.
• Do not use the VALUES clause.
• Match the number of columns in the INSERT clause to those in the subquery.
INSERT INTO copy_emp SELECT *
FROM employees;
```

## The UPDATE Statement Syntax

- Modify existing rows with the

```
UPDATE
statement.
```

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

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

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

LAST_NAME	DEPARTMENT_ID
King	110
Kochhar	110
De Haan	110
Hunold	110
Ernst	110
Lorentz	110
Mourgos	110
Gietz	110

22 rows selected.

### Updating Rows in a Table

- Specific row or rows are modified if you specify The WHERE clause.

**UPDATE employees**

```
SET department_id = 70
WHERE employee_id = 113;
```

**1 row updated.**

- All rows in the table are modified if you omit the WHERE clause.

**UPDATE copy\_emp**

```
SET department_id = 110;
```

**22 rows updated.**

### Updating Two Columns with a Subquery

Update employee 114's job and department to match that of employee 205.

**UPDATE employees**

```
SET job_id = (SELECT job_id
```

```
FROM employees
```

```
WHERE employee_id = 205),
```

```
salary = (SELECT salary
```

```
FROM employees
```

```
WHERE employee_id = 205)
```

```
WHERE employee_id = 114;
```

**1 row updated.**

### Updating Rows Based

on Another Table

**Use subqueries in UPDATE statements to update rows in a table based on values from another table.**

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

### Updating Rows:

#### **Integrity Constraint Error**

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

**ERROR at line 1:**

**ORA-02291: integrity constraint (HR.EMP\_DEPT\_FK) violated - parent key not found**

#### **Integrity Constraint Error**

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
100	Finance		
80	Sales	149	2500
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

### **Removing a Row from a Table**

#### **DEPARTMENTS**

Delete a row from the DEPARTMENTS table.

## The DELETE Statement

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

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

**Note:**If no rows are deleted, a message “  
0 rows deleted .” is returned:

### 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.**

#### Example

Remove rows identified in the WHERE clause.

**DELETE FROM employees**

**WHERE employee\_id = 114;**

**1 row deleted.**

**DELETE FROM departments**

**WHERE department\_id IN (30, 40);**

**2 rows deleted.**

### Deleting Rows Based on Another Table

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

**DELETE FROM employees**

**WHERE department\_id =**

**(SELECT department\_id**

**FROM departments**

**WHERE department\_name LIKE '%Public%');**

**1 row deleted.**

### Deleting Rows:

**Integrity Constraint Error**

**DELETE FROM departments**

**WHERE department\_id = 60;**

**DELETE FROM departments**

**\***

**ERROR at line 1:**

**ORA-02292: integrity constraint (HR.EMP\_DEPT\_FK) violated - child record found**

### Integrity Constraint Error

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

```
DELETE FROM departments WHERE department_id = 70;
1 row deleted.
```

### Using a Subquery in an

#### INSERT Statement

**INSERT INTO**

```
(SELECT employee_id, last_name,
email, hire_date, job_id, salary,
department_id
FROM employees
WHERE department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
TO_DATE('07-JUN-99', 'DD-MON-RR'),
'ST_CLERK', 5000, 50);
```

**1 row created.**

EMPLOYEE_ID	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
124	Mourgos	KMOURGOS	16-NOV-98	ST_MAN	5000	50
141	Rajs	TRAJS	17-OCT-85	ST_CLERK	3500	50
142	Davies	CDAVIES	29-JAN-97	ST_CLERK	3100	50
143	Matos	RMATOS	16-MAR-98	ST_CLERK	2600	50
144	Vargas	PVARGAS	09-JUL-98	ST_CLERK	2500	50
98999	Taylor	DTAYLOR	07-JUN-99	ST_CLERK	5000	50

6 rows selected

### Using a Subquery in an

#### INSERT Statement

- Verify the results

```
SELECT employee_id, last_name, email, hire_date,
job_id, salary, department_id
FROM employees
WHERE department_id = 50;
```

### Using the WITH CHECK OPTION

#### Keyword on DML Statements

- A subquery is used to identify the table and columns of the DML statement.
- The WITH CHECK OPTION

keyword prohibits you from changing rows that are not in the subquery.

```
INSERT INTO (SELECT employee_id, last_name, email,
hire_date, job_id, salary
FROM employees
```

```

WHERE department_id = 50 WITH CHECK OPTION)
VALUES (99998, 'Smith', 'JSMITH',
TO_DATE('07-JUN-99', 'DD-MON-RR'),
'ST_CLERK', 5000);
INSERT INTO
*
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation

```

## **The MERGE Statement**

- Provides the ability to conditionally update or insert data into a database table
- Performs an UPDATE if the row exists and an INSERT if it is a new row:
  - Avoids separate updates
  - Increases performance and ease of use
  - Is useful in data warehousing applications

## **MERGE**

### **Statement Syntax**

You can conditionally insert or update rows in a table by using the MERGE statement.

```

MERGE INTO table_name
AS table_alias
USING ( table|view|sub_query) AS alias
ON (join condition)
WHEN MATCHED THEN
UPDATE SET
col1 = col_val1,
col2 = col2_val
WHEN NOT MATCHED THEN
INSERT (column_list)
VALUES (column_values);

```

### **Merging Rows**

Insert or update rows in the COPY\_EMP table to match the EMPLOYEES table.

```

MERGE INTO copy_emp AS c
USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET

```

```

c.first_name = e.first_name,
c.last_name = e.last_name,
...
c.department_id = e.department_id
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);

```

**Merging Rows: Example**

```

MERGE INTO copy_emp AS c
USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
c.email = e.email,
c.phone_number = e.phone_number,
c.hire_date = e.hire_date,
c.job_id = e.job_id,
c.salary = e.salary,
c.commission_pct = e.commission_pct,
c.manager_id = e.manager_id,
c.department_id = e.department_id
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);

```

**Merging Rows**

```

SELECT *
FROM COPY_EMP;
no rows selected
MERGE INTO copy_emp c
USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
...
WHEN NOT MATCHED THEN
INSERT VALUES...;
SELECT *
FROM COPY_EMP;
20 rows selected.

```

## **Database Transactions**

A database transaction consists of one of the following:

- DML statements which constitute one consistent change to the data
- One DDL statement
- One DCL statement

## **Database Transactions**

- 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 iSQL\*Plus
  - The system crashes

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

statements, you can:

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

Controlling Transactions

COMMIT Time

Transaction

DELETE

SAVEPOINT A

INSERT

UPDATE

SAVEPOINT B

INSERT

ROLLBACK

ROLLBACK

ROLLBACK

to SAVEPOINT A

to SAVEPOINT B

Note:SAVEPOINT is not ANSI standard SQL.

## **Rolling Back Changes to a Marker**

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

```

UPDATE... SAVEPOINT update_done;
Savepoint created.
INSERT...
ROLLBACK TO update_done;
Rollback complete.

```

### **Implicit Transaction Processing**

- An automatic commit occurs under the following circumstances:
  - DDL statement is issued
  - DCL statement is issued
  - Normal exit from iSQL\*Plus, without explicitly Issuing COMMIT or ROLLBACK statements
  - An automatic rollback occurs under an abnormal termination of iSQL\*Plus or a system failure.

Implicit Transaction Processing

Status Circumstances

### **State of the Data Before COMMIT Or ROLLBACK**

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

### **State of the Data After COMMIT**

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

### **Committing Data**

- Make the changes.

**DELETE FROM employees**

**WHERE employee\_id = 99999;**

**1 row deleted.**

**INSERT INTO departments**

**VALUES (290, 'Corporate Tax', NULL, 1700);**

**1 row inserted.**

- Commit the changes.

**COMMIT;**

**Commit complete.**

**Example**

Remove departments 290 and 300 in the DEPARTMENTS table, and update a row in the COPY\_EMP table. Make the data change permanent.

**DELETE FROM departments**

**WHERE department\_id IN (290, 300);**

2 rows deleted.

**UPDATE copy\_emp**

**SET department\_id = 80**

**WHERE employee\_id = 206;**

1 row updated.

**COMMIT;**

Commit Complete.

## **State of the Data After ROLLBACK**

Discard all pending changes by using the

**ROLLBACK**

statement:

- Data changes are undone.

- Previous state of the data is restored.

- Locks on the affected rows are released.

**DELETE FROM copy\_emp;**

**22 rows deleted.**

**ROLLBACK;**

**Rollback complete.**

**Rolling Back Changes**

**DELETE FROM test;**

25,000 rows deleted.

**ROLLBACK;**

Rollback complete.

**DELETE FROM test**

**WHERE id = 100;**

**1 row deleted.**

**SELECT \***

**FROM test**

**WHERE id = 100;**

**No rows selected.**

**COMMIT;**

Commit complete.

## **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.

### **Read Consistency**

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with 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

### **Implementation of Read Consistency**

User A Data

**UPDATE employees blocks**

**SET salary = 7000**

**WHERE last\_name = 'Goyal';**

**Undo segments changed and**

**SELECT \***

**Read unchanged**

**FROM userA.employees;**

Data consistent before image change: “old” data

User B

### **Locking**

In an Oracle database, locks:

- Prevent destructive interaction between concurrent transactions
- Require no user action
- Use the lowest level of restrictiveness
- Are held for the duration of the transaction
- Are of two types: explicit locking and implicit locking

### **What Are Locks?**

Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource, either a user object (such as tables or rows) or a system object not visible to users (such as shared data structures and data dictionary rows).

### **How the Oracle Database Locks Data**

Locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested. Implicit locking occurs for all SQL statements except SELECT.

The users can also lock data manually, which is called explicit locking.

### **Implicit Locking**

- Two lock modes:

- Exclusive: Locks out other users
- Share: Allows other users to access the server
- High level of data concurrency:
- DML: Table share, row exclusive
- Queries: No locks required
- DDL: Protects object definitions
- Locks held until commit or rollback

## DML Locking

When performing data manipulation language (DML) operations, the Oracle Server provides data concurrency through DML locking. DML locks occur at two levels:

- A share lock is automatically obtained at the table level during DML operations. With share lock mode, several transactions can acquire share locks on the same resource.
- An exclusive lock is acquired automatically for each row modified by a DML statement. Exclusive locks prevent the row from being changed by other transactions until the transaction is committed or rolled back. This lock ensures that no other user can modify the same row at the same time and overwrite changes not yet committed by another user.

**Note:** DDL locks occur when you modify a database object such as a table.

## Practice 7

Insert data into the MY\_EMPLOYEE table.

1. Run the statement in the lab8\_1.sql script to build the MY\_EMPLOYEE table to be used for the lab.
2. Describe the structure of the MY\_EMPLOYEE table to identify the column names.
3. Add the first row of data to the MY\_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause.

**ID LAST\_NAME FIRST\_NAME USERID SALARY**

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

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

5. Confirm your addition to the table.

6. Write an INSERT statement in a text file named loademp.sql

to load rows into the MY\_EMPLOYEE table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the user ID.

7. Populate the table with the next two rows of sample data by running the INSERT statement in the script that you created.

8. Confirm your additions to the table.

9. Make the data additions permanent.

Update and delete data in the MY\_EMPLOYEE table.

10. Change the last name of employee 3 to Drexler.

11. Change the salary to 1000 for all employees with a salary less than 900.

12. Verify your changes to the table.

13. Delete Betty Dancs from the MY\_EMPLOYEE table.

14. Confirm your changes to the table.

15. Commit all pending changes. Control data transaction to the MY\_EMPLOYEE table.

16. Populate the table with the last row of sample data by modifying the statements in the script that you created in step 6. Run the statements in the script.

17. Confirm your addition to the table.

18. Mark an intermediate point in the processing of the transaction.

19. Empty the entire table.

20. Confirm that the table is empty.

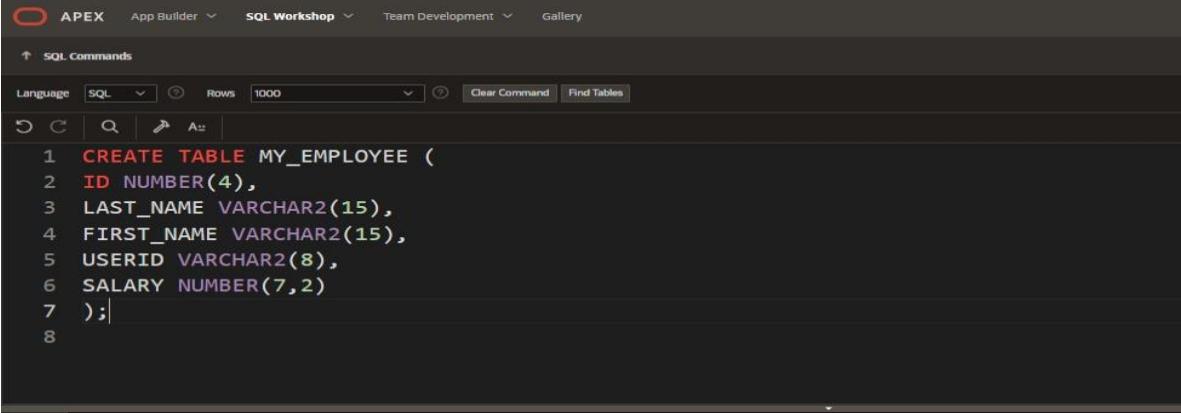
21. Discard the most recent DELETE operation without discarding the earlier INSERT operation.

22. Confirm that the new row is still intact.

23. Make the data addition permanent.

# LAB : 07

## Question: 01

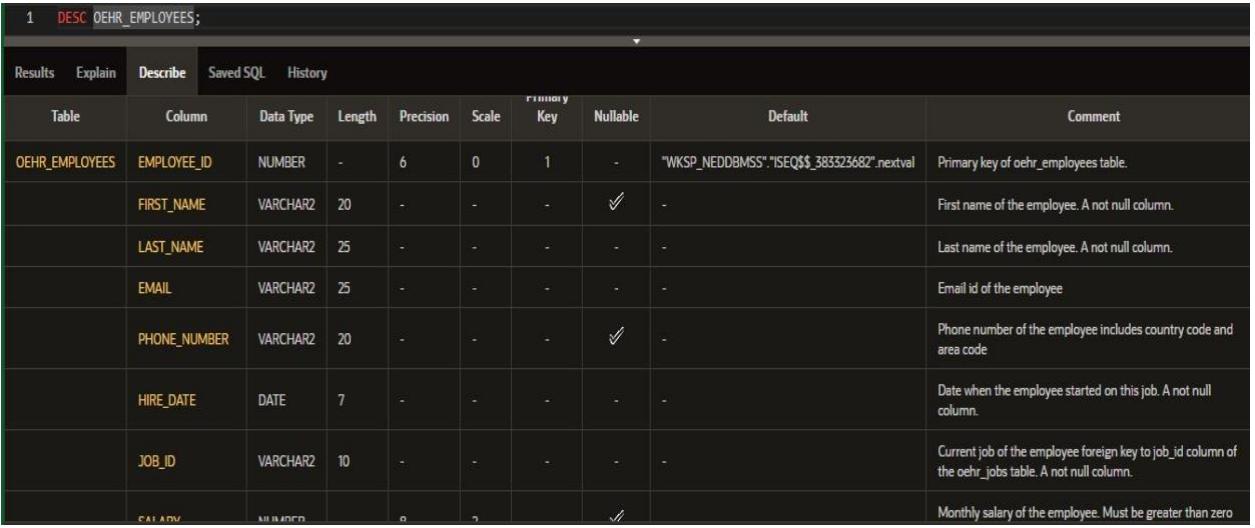


```

APEX App Builder SQL Workshop Team Development Gallery
SQL Commands
Language SQL Rows 1000 Clear Command Find Tables
SQL
1 CREATE TABLE MY_EMPLOYEE (
2   ID NUMBER(4),
3   LAST_NAME VARCHAR2(15),
4   FIRST_NAME VARCHAR2(15),
5   USERID VARCHAR2(8),
6   SALARY NUMBER(7,2)
7 );
8

```

## Question: 02



OEHR_EMPLOYEES										
Results	Explain	Describe	Saved SQL	History						
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment	
OEHR_EMPLOYEES	EMPLOYEE_ID	NUMBER	-	6	0	1	-	"WKSP_NEDDBMS"."SEQ\$\$_383323682".nextval	Primary key of oehr_employees table.	
	FIRST_NAME	VARCHAR2	20	-	-	-	✓	-	First name of the employee. A not null column.	
	LAST_NAME	VARCHAR2	25	-	-	-	-	-	Last name of the employee. A not null column.	
	EMAIL	VARCHAR2	25	-	-	-	-	-	Email id of the employee	
	PHONE_NUMBER	VARCHAR2	20	-	-	-	✓	-	Phone number of the employee includes country code and area code	
	HIRE_DATE	DATE	7	-	-	-	-	-	Date when the employee started on this job. A not null column.	
	JOB_ID	VARCHAR2	10	-	-	-	-	-	Current job of the employee foreign key to job_id column of the oehr_jobs table. A not null column.	
	SALARY	NUMBER	-	0	0	-	✓	-	Monthly salary of the employee. Must be greater than zero	

### Question: 03

```
1 INSERT INTO OEHR_EMPLOYEES
2 VALUES (300, 'Patel', 'Ralph', 'rpatel@example.com', '515.123.4567', TO_DATE('2025-04-13', 'YYYY-MM-DD'), 'AD_PRES', 895, NULL, 100, 90);
3 |
```

Results Explain Describe Saved SQL History

1 row(s) inserted.

### Question: 04

```
1 INSERT INTO OEHR_EMPLOYEES (
2   EMPLOYEE_ID, LAST_NAME, FIRST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID
3 )
4 VALUES (301, 'Dancs', 'Betty', 'bdancs@example.com', '515.555.1234', TO_DATE('2025-04-13', 'YYYY-MM-DD'), 'AD_VP', 860, NULL, 100, 90);
5 |
```

### Question: 05

```
1 SELECT * FROM OEHR_EMPLOYEES WHERE EMPLOYEE_ID IN (300, 301);
2 |
```

Results Explain Describe Saved SQL History

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
300	Patel	Ralph	rpatel@example.com	515.123.4567	4/13/2025	AD_PRES	895	-	100	90
301	Betty	Dancs	bdancs@example.com	515.555.1234	4/13/2025	AD_VP	860	-	100	90

2 rows returned in 0.01 seconds. Download

## Question: 06

```

1 INSERT INTO OEHR_EMPLOYEES (
2   EMPLOYEE_ID, LAST_NAME, FIRST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID
3 )
4 VALUES (302, 'Biri', 'Ben', 'bbiri@example.com', '515.123.1234', SYSDATE, 'IT_PROG', 1100, NULL, 100, 60);
5
6 INSERT INTO OEHR_EMPLOYEES (
7   EMPLOYEE_ID, LAST_NAME, FIRST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID
8 )
9 VALUES (303, 'Newman', 'Chad', 'cnewman@example.com', '515.234.2345', SYSDATE, 'IT_PROG', 750, NULL, 100, 60);

```

Results Explain Describe Saved SQL History

1 row(s) inserted.

## Question: 07

```

1 SELECT * FROM OEHR_EMPLOYEES WHERE EMPLOYEE_ID IN (302, 303);
2
3

```

Results Explain Describe Saved SQL History

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPART
302	Ben	Biri	bbiri@example.com	515.123.1234	4/12/2025	IT_PROG	1100	-	100	60
303	Chad	Newman	cnewman@example.com	515.234.2345	4/12/2025	IT_PROG	750	-	100	60

rows returned in 0.00 seconds [Download](#)

## Question: 08

The screenshot shows a SQL command window with the following details:

- SQL Commands:** The language is set to SQL, and the rows limit is set to 1000.
- Query:** The command entered is `SELECT * FROM MY_EMPLOYEE;`.
- Results:** The results are displayed in a table with columns: ID, LAST\_NAME, FIRST\_NAME, USERID, and SALARY.
- Data:** There are two rows of data:
 

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	serise	pro	reptile	895
2	Denics	Betty	balancs	860

### Question :09

COMMIT

### Question :10

```
1 UPDATE OEHR_EMPLOYEES
2 SET LAST_NAME = 'Drexler'
3 WHERE EMPLOYEE_ID = 302;
4
```

Results	Explain	Describe	Saved SQL	History
1 row(s) updated.				

### Question:11

```
1 UPDATE OEHR_EMPLOYEES
2 SET SALARY = 1000
3 WHERE SALARY < 900;
4
```

Results	Explain	Describe	Saved SQL	History
3 row(s) updated.				

### Question: 12

The screenshot shows a SQL query editor with the following details:

- Language: SQL
- Rows: 1000
- Clear Command, Find Tables buttons
- Run button
- SQL command: `1 SELECT * FROM MY_EMPLOYEE;`
- Results tab selected
- Table output:

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	sarisa	pro	reptile	1000
2	Drexler	Betty	bdanics	1000

- Text at the bottom: "2 rows returned in 0.01 seconds. Download"

### Question: 13

The screenshot shows a SQL query editor with the following details:

- SQL command:  
1 **DELETE FROM OEHR\_EMPLOYEES**  
2 **WHERE EMPLOYEE\_ID = 301;**  
3
- Results tab selected
- Text output:  
1 row(s) deleted.

### Question: 14

The screenshot shows a SQL command window with the following details:

- SQL Commands:** The window title.
- Language:** SQL.
- Rows:** 1000.
- Scheme:** WKSP\_DBASELAB.
- Commands:** 1. `SELECT * FROM MY_EMPLOYEE;`, 2. (empty).
- Run Button:** A green "Run" button is visible.
- Results Tab:** Selected tab.
- Table Data:** A result set table with columns: ID, LAST\_NAME, FIRST\_NAME, USERID, SALARY. One row is present: ID 2, LAST\_NAME Drexler, FIRST\_NAME Betty, USERID bdancs, SALARY 1000.
- Message:** 1 rows returned in 0.00 seconds.

### Question: 15

The screenshot shows a SQL command window with the following details:

- SQL Commands:** The window title.
- Commands:** 1. (empty), 2. `DELETE FROM OEHR_EMPLOYEES WHERE EMPLOYEE_ID >= 300;`.
- Results Tab:** Selected tab.
- Message:** 3 row(s) deleted.

### Question: 16

The screenshot shows a SQL command window with the following details:

- SQL Commands:** The window title.
- Commands:** 2. `SELECT * FROM OEHR_EMPLOYEES WHERE EMPLOYEE_ID >= 300;`.
- Results Tab:** Selected tab.
- Message:** no data found.

# 8- Creating and Managing Tables

## Objectives

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

- Describe the main database objects
- Create tables
- Describe the data types that can be used when specifying column definition
- Alter table definitions
- Drop, rename, and truncate tables

## The CREATE TABLE Statement

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

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

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

## Referencing Another User's Tables

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

```
SELECT *
FROM user_b.employees;
```

## Creating Tables

- Create the table.

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

**Table created.**

- Confirm creation of the table.

```
DESCRIBE dept
```

## Tables in the Oracle Database

### • User tables:

- Are a collection of tables created and maintained by the user

- Contain user information

### • Data dictionary:

- Is a collection of tables created and maintained by the Oracle Server

- Contain database information

### **Querying the Data Dictionary**

- See the names of tables owned by the user.

```
SELECT table_name FROM user_tables;
```

- View distinct object types owned by the user.

```
SELECT DISTINCT object_type
```

```
FROM user_objects;
```

- View tables, views, synonyms, and sequences owned by the user.

```
SELECT * FROM user_catalog;
```

### **Creating a Table**

#### **by Using a Subquery Syntax**

- 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.

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

### **The ALTER TABLE Statement**

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

### **The 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 );  
**The ALTER TABLE**

### **Adding a Column**

- Use the ADD clause to add columns.

**ALTER TABLE** dept80  
**ADD** (job\_id VARCHAR2(9));  
Table altered.

### **Modifying a Column**

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

**ALTER TABLE** dept80  
**MODIFY** (last\_name VARCHAR2(30));  
Table altered.

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

### **Dropping a Column**

Use the **DROP COLUMN** clause to drop columns you no longer need from the table.

**ALTER TABLE** dept80  
**DROP COLUMN** job\_id;  
Table altered.

### **The 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.

**ALTER TABLE** table **SET UNUSED** ( column );  
**OR**  
**ALTER TABLE** table **COLUMN** column;  
**SET UNUSED** **ALTER TABLE** table  
**DROP UNUSED COLUMNS**;

**ALTER TABLE** dept80  
**SET UNUSED** (last\_name);  
Table altered.  
**ALTER TABLE** dept80  
**DROP UNUSED COLUMNS**;

**Table altered.**

### **Dropping a Table**

- All data and structure in the table is deleted.
  - Any pending transactions are committed.
  - All indexes are dropped.
  - You cannot roll back the DROP TABLE statement.
- DROP TABLE dept80;  
Table dropped.

### **Changing the Name of an Object**

- To change the name of a table, view, sequence, or synonym, execute the RENAME statement.
- RENAME dept TO detail\_dept;  
Table renamed.
- You must be the owner of the object.

### **Truncating a Table**

- The TRUNCATE TABLE statement:
    - Removes all rows from a table
    - Releases the storage space used by that table
- TRUNCATE TABLE detail\_dept;  
Table truncated.
- You cannot roll back row removal when using TRUNCATE.
  - Alternatively, you can remove rows by using the DELETE statement.

### **Adding Comments to a Table**

- You can add comments to a table or column by using The COMMENT statement.
- COMMENT ON TABLE employees  
IS 'Employee Information';  
Comment created.
- Comments can be viewed through the data dictionary views:
    - ALL\_COL\_COMMENTS
    - USER\_COL\_COMMENTS
    - ALL\_TAB\_COMMENTS
    - USER\_TAB\_COMMENTS

## Practice 8

1. Create the DEPT table based on the following table instance chart. Place the syntax in a script called lab9\_1.sql , then execute the statement in the script to create the table.

Confirm that the table is created.

**ID NAME Column Name Key Type Nulls/Unique FK Table FK Column**

**NUMBER VARCHAR2 Data type 7 25 Length**

2. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.

3. Create the EMP table based on the following table instance chart. Place the syntax in a script called lab9\_3.sql , and then execute the statement in the script to create the table. Confirm that the table is created.

**ID LAST\_NAME FIRST\_NAME DEPT\_ID**

**Column Name**

**Key Type**

**Nulls/Unique**

**FK Table**

**FK Column**

**NUMBER VARCHAR2 VARCHAR2 NUMBER**

**Data type Length 7 25 25 7**

4. Modify the EMP table to allow for longer employee last names. Confirm your modification.

5. Confirm that both the DEPT and EMP tables are stored in the data dictionary.  
**(Hint:USER\_TABLES)**

6. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the

EMPLOYEE\_ID , FIRST\_NAME , LAST\_NAME , SALARY , and  
DEPARTMENT\_ID columns. Name the columns in your new table  
ID , FIRST\_NAME , LAST\_NAME , SALARY , and DEPT\_ID ,  
respectively.

7. Drop the EMP table.

8. Rename the EMPLOYEES2 table as EMP.

9. Add a comment to the DEPT and EMP table definitions describing the tables. Confirm your additions in the data dictionary.

10. Drop the FIRST\_NAME column from the EMP table. Confirm your modification by checking the description of the table.

11. In the EMP table, mark the DEPT\_ID column in the EMP table as  
UNUSED . Confirm your modification by checking the description of the table.

12. Drop all the UNUSED columns from the EMP table. Confirm your modification by checking the description of the table.

# Lab 08

## Database Management System

### Exercise

1. Create the DEPT table based on the following table instance chart.

```
1  CREATE TABLE DEPT2 (
2    ID NUMBER(7) PRIMARY KEY,
3    NAME VARCHAR2(25)
4 );
```

2. Populate the DEPT table with data from the DEPARTMENTS table.

```
8  INSERT INTO DEPT2 (ID, NAME)
9  SELECT DEPARTMENT_ID, DEPARTMENT_NAME
10 FROM DEPARTMENTS;
```

3. Create the EMP table based on the following table instance chart. Place the syntax in a script called lab9\_3.sql , and then execute the statement in the script to create the table.

Confirm that the table is created. ID LAST\_NAME FIRST\_NAME DEPT\_ID

```
13
14  CREATE TABLE EMP (
15    ID NUMBER(7) PRIMARY KEY,
16    LAST_NAME VARCHAR2(25),
17    FIRST_NAME VARCHAR2(25),
18    DEPT_ID NUMBER(7),
19    CONSTRAINT emp_dept2_fk FOREIGN KEY (DEPT_ID) REFERENCES DEPT2(ID)
20 );
```

4. Modify the EMP table to allow for longer employee last names. Confirm your modification.

```
23  
24  ALTER TABLE EMP  
25    MODIFY LAST_NAME VARCHAR2(50);
```

5. Confirm that both the DEPT and EMP tables are stored in the data dictionary.  
(Hint:USER\_TABLES)

```
28  
29  SELECT table_name  
30  FROM user_tables  
31  WHERE table_name IN ('DEPT2', 'EMP');
```

6. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table.

```
34  
35  CREATE TABLE EMPLOYEES2 AS  
36  SELECT  
37    EMPLOYEE_ID AS ID,  
38    FIRST_NAME,  
39    LAST_NAME,  
40    SALARY,  
41    DEPARTMENT_ID AS DEPT_ID  
42  FROM EMPLOYEES;
```

**7. Drop the EMP table.**

```
44
45  DROP TABLE EMP;
```

**8. Rename the EMPLOYEES2 table as EMP.**

```
47  ALTER TABLE EMPLOYEES2
48    RENAME TO EMP;
```

**9. Add a comment to the DEPT and EMP table definitions describing the tables.**

```
48  RENAME TO EMP;
49
50  COMMENT ON TABLE DEPT2 IS 'Department information table';
51  COMMENT ON TABLE EMP IS 'Employee information table';
```

**10. Drop the FIRST\_NAME column from the EMP table. Confirm your modification by checking the description of the table.**

```
55
56  ALTER TABLE EMP
57    DROP COLUMN FIRST_NAME;
```

**11. In the EMP table, mark the DEPT\_ID column in the EMP table as**

```
59
60  ALTER TABLE EMP
61    SET UNUSED (DEPT_ID);
```

## 8- Constraints

### Objectives

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

- Describe constraints
- Create and maintain constraints

### What Are Constraints?

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies.

#### • The following constraint types are valid:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

### Defining Constraints

```
CREATE TABLE [schema.]table(column datatype
[DEFAULTexpr][column_constraint],...[table_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));
```

### Defining Constraints

#### • Column constraint level:

Column [CONSTRAINT constraint\_name ]  
constraint\_type ,

#### • Table constraint level:

column,...[CONSTRAINT constraint\_name]  
constraint\_type(column, ...),

### The NOT NULL Constraint

Ensures that null values are not permitted for the

#### The NOT NULL Constraint

The NOT NULL constraint ensures that the column contains no null values. Columns without the NOT NULL constraint can contain null values by default.

## The NOT NULL Constraint

Is defined at the column level

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  System
  last_name    VARCHAR2(25) NOT NULL,
  named
  salary       NUMBER(8,2),
  commission_pct NUMBER(2,2),
  hire_date    DATE
```

## The UNIQUE Constraint

**UNIQUE constraint**

**EMPLOYEES**  
**INSERT INTO**

**Allowed**

**Not allowed:**

**already exists**

## The UNIQUE Constraint

Is defined at either the table level or the column level

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

## The PRIMARY KEY Constraint

**DEPARTMENTS PRIMARY KEY**

**Not allowed (null value)**

**INSERT INTO**

**Not allowed**

**(50 already exists)**

Is defined at either the table level or the column level

```
CREATE TABLE departments(
  department_id  NUMBER(4),
  department_name  VARCHAR2(30)
CONSTRAINT dept_name_nn NOT NULL,
  manager_id     NUMBER(6),
  location_id    NUMBER(4),
CONSTRAINT dept_id_pk PRIMARY KEY(department_id));
```

**The FOREIGN KEY Constraint****DEPARTMENTS PRIMARY KEY****EMPLOYEES FOREIGN KEY****Not allowed****INSERT INTO****(9 does not exist)****Allowed****The FOREIGN KEY Constraint**

Is defined at either the table level or the column level

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

**FOREIGN KEY Constraint Keywords**

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

**The CHECK Constraint**

- Defines a condition that each row must satisfy
  - The following expressions are not allowed:
    - References to CURRVAL,NEXTVAL,LEVEL, and ROWNUM pseudocolumns
    - Calls to SYSDATE ,UID,USER, and USERENV functions
    - Queries that refer to other values in other rows
- ..., salary NUMBER(2)  
**CONSTRAINT emp\_salary\_min**  
**CHECK (salary > 0),...**

**Adding a Constraint Syntax**

Use the ALTER TABLE statement to:

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

**ALTER TABLE table**

**ADD [CONSTRAINT  
constraint]type(column);**

### **Adding a Constraint**

Add a FOREIGN KEY constraint to the EMPLOYEES table to indicate that a manager must already exist as a valid employee in the EMPLOYEES table.

```
ALTER TABLE employees  
ADD CONSTRAINT emp_manager_fk  
FOREIGN KEY(manager_id)  
REFERENCES employees(employee_id);  
Table altered.
```

### **Dropping a Constraint**

- Remove the manager constraint from the EMPLOYEES table.

```
ALTER TABLE employees  
DROP CONSTRAINT emp_manager_fk;  
Table altered.
```

- Remove the PRIMARY KEY constraint on the DEPARTMENTS table and drop the associated FOREIGN KEY constraint on the EMPLOYEES.DEPARTMENT\_ID column.

```
ALTER TABLE departments DROP PRIMARY KEY CASCADE;  
Table altered.
```

### **Disabling Constraints**

- Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint.
- Apply the CASCADE option to disable dependent integrity constraints.

```
ALTER TABLE employees  
DISABLE CONSTRAINT emp_emp_id_pk CASCADE;  
Table altered.
```

## **Enabling Constraints**

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

**ALTER TABLE employees**

**ENABLE CONSTRAINT emp\_emp\_id\_pk;**

**Table altered.**

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

## **Cascading Constraints**

- The CASCADE CONSTRAINTS clause is used along with the DROP COLUMN clause.
- The CASCADE CONSTRAINTS clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.
- The CASCADE CONSTRAINTS clause also drops all multicolumn constraints defined on the dropped columns.

## **Cascading Constraints**

### **Example**

**ALTER TABLE test1**

**DROP (pk) CASCADE CONSTRAINTS;**

**Table altered.**

**ALTER TABLE test1**

**DROP (pk, fk, col1) CASCADE CONSTRAINTS;**

**Table altered.**

## **Viewing Constraints**

Query the USER\_CONSTRAINTS table to view all constraint definitions and names.

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

### **Viewing Constraints**

## **Viewing the Columns Associated with Constraints**

View the columns associated with the constraint names in the USER\_CONS\_COLUMNS view.

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

### Practice 8

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my\_emp\_id\_pk.

**Hint:** The constraint is enabled as soon as the ALTER TABLE command executes successfully.

2. Create a PRIMARY KEY constraint to the DEPT table using the ID column. The constraint should be named at creation. Name the constraint my\_deptid\_pk .

**Hint:**

The constraint is enabled as soon as the ALTER TABLE command executes successfully.

3. Add a column DEPT\_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my\_emp\_dept\_id\_fk.

4. Confirm that the constraints were added by querying the USER\_CONSTRAINTS view. Note the types and names of the constraints. Save your statement text in a file called lab10\_4.sql.

5. Display the object names and types from the USER\_OBJECTS data dictionary view for the EMP and DEPT tables. Notice that the new tables and a new index were created.

If you have time, complete the following exercise:

6. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.

## Lab 09 Database Management System

### Exercise

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my\_emp\_id\_pk.

```
67
68  ALTER TABLE EMP
69    ADD CONSTRAINT my_emp_id_pk PRIMARY KEY (ID);
```

2. Create a PRIMARY KEY constraint to the DEPT table using the ID column. The constraint should be named at creation. Name the constraint my\_deptid\_pk .

```
71
72  ALTER TABLE DEPT2
73    ADD CONSTRAINT my_deptid_pk PRIMARY KEY (ID);
```

3. Add a column DEPT\_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my\_emp\_dept\_id\_fk.

```
75
76  ALTER TABLE EMP
77    ADD DEPT_ID NUMBER(7);
78
79  ALTER TABLE EMP
80    ADD CONSTRAINT my_emp_dept_id_fk FOREIGN KEY (DEPT_ID)
81      REFERENCES DEPT2(ID);
```

**4. Confirm that the constraints were added by querying the USER\_CONSTRAINTS view.**

```
83
84  SELECT constraint_name, constraint_type, table_name
85  FROM user_constraints
86  WHERE table_name IN ('EMP', 'DEPT2');
```

**5. Display the object names and types from the USER\_OBJECTS data dictionary view for the EMP and DEPT tables**

```
90  SELECT object_name, object_type
91  FROM user_objects
92  WHERE object_name IN ('EMP', 'DEPT2');
```

**6. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.**

```
96
97  ALTER TABLE EMP
98  ADD COMMISSION NUMBER(2,2);
99
100 ALTER TABLE EMP
101 ADD CONSTRAINT emp_commission_ck
102 CHECK (COMMISSION > 0);
```

## 9- Creating Views

### Objectives

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

- Describe a view
- Create, alter the definition of, and drop a view
- Retrieve data through a view
- Insert, update, and delete data through a view

### Creating a View

- Create a view, EMPVU80, that contains details of employees in department 80.

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

• Describe the structure of the view by using the iSQL*Plus DESCRIBE command.
```

```
DESCRIBE empvu80
```

### Creating a View

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

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

• Select the columns from this view by the given alias names.
```

### Retrieving Data from a View

```
SELECT *
FROM salvu50;
```

### Querying a View

Oracle Server iSQL\*Plus USER\_VIEWS

```
SELECT *
EMPVU80
FROM empvu80;
SELECT employee_id,
last_name, salary
FROM employees
WHERE department_id=80;
```

## Modifying a View

- Modify the EMPVU80 view by using

CREATE OR REPLACE VIEW

clause. Add an alias for each column name.

```
CREATE OR REPLACE VIEW empvu80
```

```
(id_number, name, sal, department_id)
```

```
AS SELECT employee_id, first_name || ' ' || last_name,
```

```
salary, department_id
```

```
FROM employees
```

```
WHERE department_id = 80;
```

View created.

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

## Creating a Complex View

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

```
CREATE VIEW dept_sum_vu
```

```
(name, minsal, maxsal, avgsal)
```

```
AS SELECT d.department_name, MIN(e.salary),
```

```
MAX(e.salary), AVG(e.salary)
```

```
FROM employees e, departments d
```

```
WHERE e.department_id = d.department_id
```

```
GROUP BY d.department_name;
```

View created.

## Rules for Performing

### DML Operations on a View

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

## Denying DML Operations

```
CREATE OR REPLACE VIEW empvu10
```

```
(employee_number, employee_name, job_title)
```

```
AS SELECT employee_id, last_name, job_id
```

```
FROM employees
```

```
WHERE department_id = 10
```

```
WITH READ ONLY;
```

View created.

## Removing a View

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

```
DROP VIEW view;  
DROP VIEW empvu80;  
View dropped.
```

**Practice 09**

1. Create a view called EMPLOYEES\_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table.

Change the heading for the employee name to EMPLOYEE.

2. Display the contents of the EMPLOYEES\_VU view.

3. Select the view name and text from the USER.Views data dictionary view.

**Note:** Another view already exists. The EMP\_DETAILS\_VIEW was created as part of your schema.

**Note:** To see more contents of a LONG column, use the *iSQL\*Plus* command

SET LONG *n*, where *n* is the value of the number of characters of the LONG column that you want to see.

4. Using your EMPLOYEES\_VU view, enter a query to display all employee names and department numbers.

5. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE, and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

6. Display the structure and contents of the DEPT50 view.

7. Attempt to reassign Matos to department 80.

If you have time, complete the following exercise:

8. Create a view called SALARY\_VU based on the employee last names, department names, salaries,

and salary grades for all employees. Use the EMPLOYEES,DEPARTMENTS, and

JOB\_GRADES tables. Label the columns Employee,Department,Salary, and Grade, respectively.

# LAB: 10

## CREATING VIEWS

### Question 1

Create a view called EMPLOYEES\_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.

```
1 CREATE VIEW EMPLOYEES_VU AS
2 SELECT EMPLOYEE_ID, LAST_NAME EMPLOYEE, DEPARTMENT_ID
3 FROM OEHR_EMPLOYEES;
```

Results Explain Describe Saved SQL History

View created.

0.04 seconds

### Question 2

Display the contents of the EMPLOYEES\_VU view.

Results Explain Describe Saved SQL History

EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60
105	Austin	60
106	Pataballa	60

## Question 3

Select the view name and text from the USER\_VIEWS data dictionary view.

**Note:** Another view already exists. The EMP\_DETAILS\_VIEW was created as part of your schema.

**Note:** To see more contents of a LONG column, use the iSQL\*Plus command SET LONG n, where n is the value of the number of characters of the LONG column that you want to see.

```

1  SET LONG 600
2  SELECT VIEW_NAME, TEXT
3  FROM USER_VIEWS;

```

VIEW_NAME	TEXT
EMPLOYEES_VU	SELECT EMPLOYEE_ID, LAST_NAME EMPLOYEE, DEPARTMENT_ID FROM OEHR_EMPLOYEES
EMP_DEPT_V	select e.empno , eENAME , e.job , (select m.ENAME from emp m where e.MGR = m.empno) MGR , e.hiredate , e.SAL , e.comm , d.deptno , d.DNAME , d.LOC from emp e , dept d where e.deptno = d.deptno (+)
OEHR_ACCOUNT_MANAGERS	select c.account_mgr_id acct_mgr, cr.region_id region, c.country_id country, c.state_province province, count(*) num_customers from oehr_customers c, oehr_countries cr where c.country_id = cr.country_id group by rollup (c.account_mgr_id, cr.region_id, c.country_id, c.state_province)
OEHR_BOMBAY_INVENTORY	select p.product_id , p.product_name , i.quantity_on_hand from oehr_inventories i , oehr_warehouses w , oehr_products p where p.product_id = i.product_id and i.warehouse_id = w.warehouse_id and w.warehouse_name = 'Bombay'
OEHR_CUSTOMERS_VIEW	select c.customer_id, c.cust_first_name, c.cust_last_name, c.street_address, c.postal_code, c.city, c.state_province, co.country_id, co.country_name, co.region_id, c.nls_language,

## Question 4

Using your EMPLOYEES\_VU view, enter a query to display all employee names and department numbers.

```

1  SELECT EMPLOYEE, DEPARTMENT_ID
2  FROM EMPLOYEES_VU;

```

EMPLOYEE	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Austin	60
Pataballa	60

## Question 5

Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE, and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

```
1 CREATE VIEW DEPT50 AS
2 SELECT EMPLOYEE_ID EMPNO, LAST_NAME EMPLOYEE, DEPARTMENT_ID DEPTNO
3 FROM OHR_EMPLOYEES
4 WHERE DEPARTMENT_ID = 50
5 WITH CHECK OPTION CONSTRAINT EMP_DEPT_50;
```

Results Explain Describe Saved SQL History

View created.

0.05 seconds

## Question 6

Display the structure and contents of the DEPT50 view.

```
1 SELECT COLUMN_NAME, DATA_TYPE,
2 DATA_LENGTH, NULLABLE
3 FROM ALL_TAB_COLUMNS
4 WHERE TABLE_NAME = 'DEPT50';
```

Results Explain Describe Saved SQL History

COLUMN_NAME	DATA_TYPE	DATA_LENGTH	NULLABLE
EMPNO	NUMBER	22	N
EMPLOYEE	VARCHAR2	25	N
DEPTNO	NUMBER	22	Y

3 rows returned in 1.01 seconds Download

```
1 SELECT * FROM DEPT50;
```

Results Explain Describe Saved SQL History

EMPNO	EMPLOYEE	DEPTNO
120	Weiss	50
121	Fripp	50
122	Kaufling	50
123	Vollman	50
124	Mourgos	50
125	Nayer	50
126	Mikkilineni	50

EMPNO	EMPLOYEE	DEPTNO
120	Weiss	50
121	Fripp	50
122	Kaufling	50
123	Vollman	50
124	Mourgos	50
125	Nayer	50
126	Mikkilineni	50

EMPNO	EMPLOYEE	DEPTNO
120	Weiss	50
121	Fripp	50
122	Kaufling	50
123	Vollman	50
124	Mourgos	50
125	Nayer	50
126	Mikkilineni	50

## Question 7

Attempt to reassign Matos to department 80.

The screenshot shows a SQL editor window in Oracle SQL Developer. The code entered is:

```
1 UPDATE DEPT50
2 SET DEPTNO = 88
3 WHERE EMPLOYEE = 'Matos';
```

Below the code, the results tab is selected, showing an error message in a yellow box: "ORA-01402: view WITH CHECK OPTION where-clause violation".

## Question 8

Create a view called SALARY\_VU based on the employee last names, department names,salaries, and salary grades for all employees. Use the EMPLOYEES,DEPARTMENTS, and JOB\_GRADES tables. Label the columns Employee,Department,Salary, andGrade, respectively.

The screenshot shows a SQL editor window in Oracle SQL Developer. The code entered is:

```
1 CREATE OR REPLACE VIEW salary_vu AS
2 SELECT
3     e.LAST_NAME AS Employee,
4     d.DEPARTMENT_NAME AS Department,
5     e.SALARY AS Salary
6 FROM
7     OEHR_EMPLOYEES e
8 JOIN
9     OEHR_DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID;
```

Below the code, the results tab is selected, showing the message "View created".

On the right, the view details are displayed under the heading "SALARY\_VU". The table shows the following columns:

Column Name	Data Type	Nullable	Comment
EMPLOYEE	VARCHAR2(25 BYTE)	N	
DEPARTMENT	VARCHAR2(30 BYTE)	N	
SALARY	NUMBER(8,2)	Y	

## **PL/SQL Section**

**1-Declaring Variables**

**2-Writing Executable Statements**

**3-Interacting with the Oracle Server**

**4-Writing Control Structures**

## 1-Declaring Variables

### PL/SQL Block Structure

```

DECLARE (Optional)
Variables, cursors, user-defined exceptions
BEGIN (Mandatory)
– SQL statements
– PL/SQL statements
EXCEPTION (Optional)
Actions to perform when errors occur
END; (Mandatory)

```

### Executing Statements and PL/SQL Blocks

```

DECLARE
v_variable VARCHAR2(5);
BEGIN
SELECT column_name
INTO v_variable
FROM table_name;
EXCEPTION
WHEN exception_name THEN
...
END;

```

### Declaring PL/SQL Variables

```

identifier [CONSTANT] datatype [NOT NULL]
[:= | DEFAULT expr];

```

```

DECLARE
v_hiredate DATE;
v_deptno NUMBER(2) NOT NULL := 10;
v_location VARCHAR2(13) := 'Atlanta';
c_comm CONSTANT NUMBER := 1400;

```

### Base Scalar Data Types

- **CHAR [(*maximum\_length*)]**
- **VARCHAR2 (*maximum\_length*)**
- **LONG**
- **LONG RAW**

- NUMBER [(precision, scale)]

- BINARY\_INTEGER

- PLS\_INTEGER

- BOOLEAN

```
DECLARE
v_job VARCHAR2(9);
v_count BINARY_INTEGER := 0;
v_total_sal NUMBER(9,2) := 0;
v_orderdate DATE := SYSDATE + 7;
c_tax_rate CONSTANT NUMBER(3,2) := 8.25;
v_valid BOOLEAN NOT NULL := TRUE;
```

### Declaring Variables with the %TYPE Attribute

```
identifier Table.column_name%TYPE;

...
v_name employees.last_name%TYPE;
v_balance NUMBER(7,2);
v_min_balance v_balance%TYPE := 10;
...
```

### B i n d   V a r i a b l e s

```
VARIABLE g_salary NUMBER
BEGIN
SELECT salary
INTO :g_salary
FROM employees
WHERE employee_id = 178;
END;
/
PRINT g_salary
```

D B M S \_ O U T P U T . P U T \_ L I N

```

SET SERVEROUTPUT ON
DEFINE p_annual_sal = 60000

DECLARE
  v_sal NUMBER(9,2) := &p_annual_sal;
BEGIN
  v_sal := v_sal/12;
  DBMS_OUTPUT.PUT_LINE ('The monthly salary is ' ||
    TO_CHAR(v_sal));
END;
/

```

### Practice 1

1. Evaluate each of the following declarations. Determine which of them are *not* legal and explain why.
  - a. DECLARE v\_id NUMBER(4);
  - b. DECLARE v\_x, v\_y, v\_z VARCHAR2(10);
  - c. DECLARE v\_birthdate DATE NOT NULL;
  - d. DECLARE v\_in\_stock BOOLEAN := 1;
2. In each of the following assignments, indicate whether the statement is valid and what the valid data type of the result will be.

a.v\_days\_to\_go := v\_due\_date - SYSDATE;  
 b.v\_sender := USER || ':' || TO\_CHAR(v\_dept\_no);  
 c.v\_sum := \$100,000 + \$250,000;  
 d.v\_flag := TRUE;  
 e.v\_n1 := v\_n2 > (2 \* v\_n3);  
 f.v\_value := NULL;

If you have time, complete the following exercise:

4. Create a block that declares two variables. Assign the value of these PL/SQL variables to SQL\*Plus host variables and print the results of the PL/SQL variables to the screen. Execute your PL/SQL block. Save your PL/SQL block in a file named p1q4.sql , by clicking the Save Script button. Remember to save the script with a .sql extension.

V\_CHAR Character (variable length) V\_NUM Number

Assign values to these variables as follows:

Variable Value

---

V\_CHAR The literal '42' is the answer'

V\_NUM The first two characters from V\_CHAR

# PL/SQL

## Section

### 1-Declaring Variables

**DECLARE**

```
v_employee_name  
VARCHAR2(100);
```

```
v_salary NUMBER(10,  
2) := 5000.00;
```

**BEGIN**

```
-- Executable  
statements
```

**END;**

### 2-Writing Executable Statements

**DECLARE**

```
v_employee_name VARCHAR2(100);
```

**BEGIN**

```
SELECT employee_name INTO  
v_employee_name
```

```
FROM employees
```

```
WHERE employee_id = 101;
```

```
DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_employee_name);
```

```
END;
```

### **3-Interacting with the Oracle Server**

```
DECLARE
```

```
v_employee_name VARCHAR2(100) :=  
'John Doe';
```

```
v_employee_salary NUMBER(10, 2) := 6000;
```

```
BEGIN
```

```
INSERT INTO employees  
(employee_name, salary)
```

```
VALUES (v_employee_name,  
v_employee_salary);
```

```
COMMIT;
```

```
END;
```

### **4-Writing Control Structures**

```
DECLARE
```

```
v_grade CHAR(1) := 'A';
```

```
BEGIN
```

```
CASE v_grade
```

```
WHEN 'A' THEN
```

```
DBMS_OUTPUT.PUT_LINE('Excellent');
```

```
WHEN 'B' THEN
```

```
DBMS_OUTPUT.PUT_LINE('Good');      WHEN 'C' THEN
```

```
DBMS_OUTPUT.PUT_LINE('Average');
```

```
ELSE      DBMS_OUTPUT.PUT_LINE('Fail'); END CASE; END;
```

## 1-Declaring Variables

### PL/SQL Block Structure

```

DECLARE
  (Optional)
  Variables, cursors, user-defined exceptions
BEGIN (Mandatory)
  - SQL statements -
  PL/SQL statements
EXCEPTION
  (Optional)
  Actions to perform when errors occur END;
  (Mandatory)

```

### Executing Statements and PL/SQL Blocks

```

DECLARE v variable VARCHAR2 (5)
; BEGIN
SELECT Column name
INTO v variable
FROM table name;
EXCEPTION
WHEN exception_name THEN
END;

```

### Declaring PL/SQL Variables

```

identifier [CONSTANT] datatype [NOT NULL] [:=
| DEFAULT expr];

DECLARE
v hiredate DATE; v deptno NUMBER (2) NOT
NULL := 10; v location VARCHAR2 (13) :=
'Atlanta'; c comm CONSTANT NUMBER := 1400;

```

### Base Scalar Data Types

- CHAR  
[(*maximum\_length*)]
- VARCHAR2 (*maximum\_length*)
- LONG
- LONG RAW

87

Database Management Systems CT-261

- NUMBER [(*precision, scale*)]
- BINARY\_INTEGER
- PLS\_INTEGER
- BOOLEAN

```

DECLARE
    v_job VARCHAR2 (9);
    v_count BINARY_INTEGER := 0;
    total_sal NUMBER (9,2) := 0;
    v_orderdate DATE := SYSDATE + 7;
    ..c_tax_rate CONSTANT NUMBER (3,2) := 8.25;
    v_valid BOOLEAN NOT NULL := TRUE;

```

### Declaring Variables with the %TYPE Attribute

```

identifier Table.column name%TYPE;
v_name employees. last name%TYPE; v_balance
NUMBER (7,2);

v_min_balance v_balance TYPE = 10;

```

### Bind Variables

```

VARIABLE g_salary NUMBER
BEGIN SELECT
salary
INTO :g_salary
FROM employees
WHERE employee_id = 178;
END;
/
PRINT g_salary

DBMS_OUTPUT.PUT_LIN

```

88

Database Management Systems CT-261

```

SET SERVEROUTPUT ON

DEFINE p_annual_sal = 60000

DECLARE

    v_sal NUMBER (9,2) := &p_annual_sal;
BEGIN
    v_sal := v_sal/12;

    DBMS_OUTPUT.PUT_LINE ('The monthly salary is
TO CHAR (v_sal));
END;
/

```

### Practice 1

1. Evaluate each of the following declarations. Determine which of them are *not* legal and explain why.
  - a. DECLARE v\_id NUMBER(4);
  - b. DECLARE v\_x, v\_y, v\_z VARCHAR2(10);
  - c. DECLARE v\_birthdate DATE NOT NULL;
  - d. DECLARE v\_in\_stock BOOLEAN := 1;
2. In each of the following assignments, indicate whether the statement is valid and what the valid data type of the result will be.

```

a.v_days_to_go := v_due_date - SYSDATE;
b.v_sender := USER || ':' ||
TO_CHAR(v_dept_no); c.v_sum = $100,000+
$250,000;
d.v_flag := TRUE;
e.v_n1 := v_n2 > (2 * v_n3);
f.v value = NULL;

```

If you have time, complete the following exercise:

4. Create a block that declares two variables. Assign the value of these PL/SQL variables to iSQL\*Plus host **variables** and print the results of the PL/SQL variables to the screen. Execute your PL/SQL block. Save your PL/SQL block in a file named **plq4.sql**, by clicking the Save Script button.  
Remember to save the script with a .sql extension. V\_CHAR Character (variable length)  
V\_NUM Number

Assign values to these variables as follows:

Variable	Value
----------	-------

V CHAR The literal '42 is the answer'

V\_NUM The first two characters from V\_CHAR

## 2-Writing Executable Statements

### Data Type Conversion

Conversion functions:

- **TO\_CHAR**
- **TO\_DATE**
- **TO\_NUMBER**

```
DECLARE
v_date DATE := TO_DATE('12-JAN-2001', 'DD-MON-YYYY');
BEGIN
    . . .
END;
```

**Practice 2****PL/SQL Block**

```

DECLARE
v_weight NUMBER(3) := 600;
v_message VARCHAR2(255) := 'Product 10012';
BEGIN
DECLARE
v_weight NUMBER(3) := 1;
v_message VARCHAR2(255) := 'Product 11001';
v_new_locn VARCHAR2(50) := 'Europe';
BEGIN
v_weight := v_weight + 1;
v_new_locn := 'Western ' || v_new_locn;
1
END;
v_weight := v_weight + 1;
v_message := v_message || ' is in stock';
v_new_locn := 'Western ' || v_new_locn;
2
END;
/

```

1. Evaluate the PL/SQL block above and determine the data type and value of each of the following variables according to the rules of scoping.

- The value of V\_WEIGHT at position 1 is:
- The value of V\_NEW\_LOCN at position 1 is:
- The value of V\_WEIGHT at position 2 is:
- The value of V\_MESSAGE at position 2 is:
- The value of V\_NEW\_LOCN at position 2 is:

**Scope Example**

```

DECLARE
v_customer VARCHAR2(50) := 'Womansport';
v_credit_rating VARCHAR2(50) := 'EXCELLENT';
BEGIN
DECLARE
v_customer NUMBER(7) := 201;
v_name VARCHAR2(25) := 'Unisports';
BEGIN
v_customer v_name v_credit_rating
END;
v_customer v_name v_credit_rating
END;
/

```

2. Suppose you embed a subblock within a block, as shown above. You declare two variables,

V\_CUSTOMER and V\_CREDIT\_RATING , in the main block. You also declare two variables,

V\_CUSTOMER and V\_NAME , in the subblock. Determine the values and data types for

each of  
the following cases.

- a. The value of V\_CUSTOMER in the subblock is:
- b. The value of V\_NAME in the subblock is:
- c. The value of V\_CREDIT\_RATING in the subblock is:
- d. The value of V\_CUSTOMER in the main block is:
- e. The value of V\_NAME in the main block is:
- f. The value of V\_CREDIT\_RATING in the main block is:

3. Create and execute a PL/SQL block that accepts two numbers through iSQL\*Plus substitution variables.

- a. Use the DEFINE command to provide the two values.

```
DEFINE p_num1 =2
DEFINE p_num2 =4
```

b. Pass the two values defined in step a above, to the PL/SQL block through iSQL\*Plus substitution variables. The first number should be divided by the second number and have the second number added to the result. The result should be stored in a PL/SQL variable and printed on the screen.

**Note:**

SET VERIFY OFF in the PL/SQL block.

4. Build a PL/SQL block that computes the total compensation for one year.

- a. The annual salary and the annual bonus percentage values are defined using the DEFINE command.
- b. Pass the values defined in the above step to the PL/SQL block through iSQL\*Plus substitution variables. The bonus must be converted from a whole number to a decimal (for example, 15 to .15). If the salary is null, set it to zero before computing the total compensation. Execute the PL/SQL block.

*Reminder:* Use the NVL function to handle null values.

**Note:** Total compensation is the sum of the annual salary and the annual bonus.

To test the NVL function, set the DEFINE variable equal to NULL.

```
DEFINE p_salary =50000
DEFINE p_bonus =10
```

# LAB 12

## Question 01

```
1  DECLARE
2      v_weight NUMBER(3) := 600;
3      v_message VARCHAR2(255) := 'Product 10012';
4  BEGIN
5      DECLARE
6          v_weight NUMBER(3) := 1;
7          v_message VARCHAR2(255) := 'Product 11001';
8          v_new_locn VARCHAR2(50) := 'Europe';
9      BEGIN
10         v_weight := v_weight + 1; -- Position 1
11         v_new_locn := 'Western ' || v_new_locn;
12         DBMS_OUTPUT.PUT_LINE('Inner Block - v_weight: ' || v_weight);
13         DBMS_OUTPUT.PUT_LINE('Inner Block - v_new_locn: ' || v_new_locn);
14     END;
15
16     v_weight := v_weight + 1; -- Position 2
17     v_message := v_message || ' is in stock';
18     -- v_new_locn is not declared in this scope
19
20     DBMS_OUTPUT.PUT_LINE('Outer Block - v_weight: ' || v_weight);
21     DBMS_OUTPUT.PUT_LINE('Outer Block - v_message: ' || v_message);
22 END;
23
```

Results	Explain	Describe	Saved SQL	History
Inner Block - v_weight: 2 Inner Block - v_new_locn: Western Europe Outer Block - v_weight: 601 Outer Block - v_message: Product 10012 is in stock  statement processed.				

## Answers:

a. The value of `v_weight` at position 1 is:

- Inside the inner block:  $v\_weight = 1 + 1 = 2$

b. The value of `v_new_locn` at position 1 is:

- Inside the inner block: 'Western Europe'

Fizza Ahmed  
DT-23304

c. The value of v\_weight at position 2 is:

- From outer block:  $600 + 1 = 601$

d. The value of v\_message at position 2 is:

- 'Product 10012 is in stock'

e. The value of v\_new\_locn at position 2 is:

- Error: v\_new\_locn is not declared in the outer block; it's local to the inner block and out of scope here

## Question 02

```
1  DECLARE
2      v_customer VARCHAR2(50) := 'Womansport';
3      v_credit_rating VARCHAR2(50) := 'EXCELLENT';
4  BEGIN
5      DECLARE
6          v_customer NUMBER(7) := 201;
7          v_name VARCHAR2(25) := 'Unisports';
8      BEGIN
9          DBMS_OUTPUT.PUT_LINE('Subblock - v_customer: ' || v_customer);
10         DBMS_OUTPUT.PUT_LINE('Subblock - v_name: ' || v_name);
11         DBMS_OUTPUT.PUT_LINE('Subblock - v_credit_rating: ' || v_credit_rating);
12     END;
13
14     DBMS_OUTPUT.PUT_LINE('Main block - v_customer: ' || v_customer);
15     DBMS_OUTPUT.PUT_LINE('Main block - v_credit_rating: ' || v_credit_rating);
16
17     -- v_name not accessible here
18 END;
```

Results	Explain	Describe	Saved SQL	History
Subblock - v_customer: 201 Subblock - v_name: Unisports Subblock - v_credit_rating: EXCELLENT Main block - v_customer: Womansport Main block - v_credit_rating: EXCELLENT  statement processed.				

## Answers:

a. V\_CUSTOMER in the subblock:

- Value: 201
- Type: NUMBER(7)

Fizza Ahmed  
DT-23304

b. **V\_NAME in the subblock:**

- Value: 'Unisports'
- Type: VARCHAR2(25)

c. **V\_CREDIT\_RATING in the subblock:**

- Value: 'EXCELLENT'
- Type: VARCHAR2(50) (outer scope)

d. **V\_CUSTOMER in the main block:**

- Value: 'Womansport'
- Type: VARCHAR2(50)

e. **V\_NAME in the main block:**

- **Not accessible** — declared only in subblock

f. **V\_CREDIT\_RATING in the main block:**

- Value: 'EXCELLENT'
- Type: VARCHAR2(50)

### Question 03

```
1  DECLARE
2      v_result NUMBER;
3  BEGIN
4      v_result := :P1_NUM1 / :P1_NUM2 + :P1_NUM2;
5      :P1_RESULT := v_result;
6  END;
7  
```

Enter Bind Variables - Profile 1 - Microsoft Edge

https://apex.oracle.com/pls/apex/f?p=4500:138:111174685979679:::

Submit

Bind Variable	Value
:P1_NUM1	1
:P1_NUM2	2
:P1_RESULT	

### Question 04

```
1  DECLARE
2      v_salary NUMBER := NVL(:P1_SALARY, 0);
3      v_bonus_pct NUMBER := NVL(:P1_BONUS, 0) / 100;
4      v_total_comp NUMBER;
5  BEGIN
6      v_total_comp := v_salary + (v_salary * v_bonus_pct);
7      :P1_TOTAL_COMP := v_total_comp;
8  END;
9  
```

Fizza Ahmed  
DT-23304

✉ Enter Bind Variables - Profile 1 - Microsoft Edge

https://apex.oracle.com/pls/apex/f?p=4500:138:111174685979679:::

174

Bind Variable	Value
:P1_SALARY	<input type="text"/>
:P1_BONUS	<input type="text"/>
:P1_TOTAL_COMP	<input type="text"/>

Submit