DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 1

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To create database in Ms. Access and to create relationships between tables in a database.

Tools: MicroSoft Access

Introduction:

MS ACCESS: Microsoft Access is a Database Management System (DBMS) from Microsoft that combines the relational Microsoft Jet Database Engine with a graphical user interface and software development tools

- Microsoft Access is just one part of Microsoft's overall data management product strategy.
- It stores data in its own format based on the Access Jet Database Engine.
- Like relational databases, Microsoft Access also allows you to link related information easily. For example, customer and order data. However, Access 2013 also complements other database products because it has several powerful connectivity features.

Microsoft Access stores information which is called a database. To use MS Access, you will need to follow these four steps −

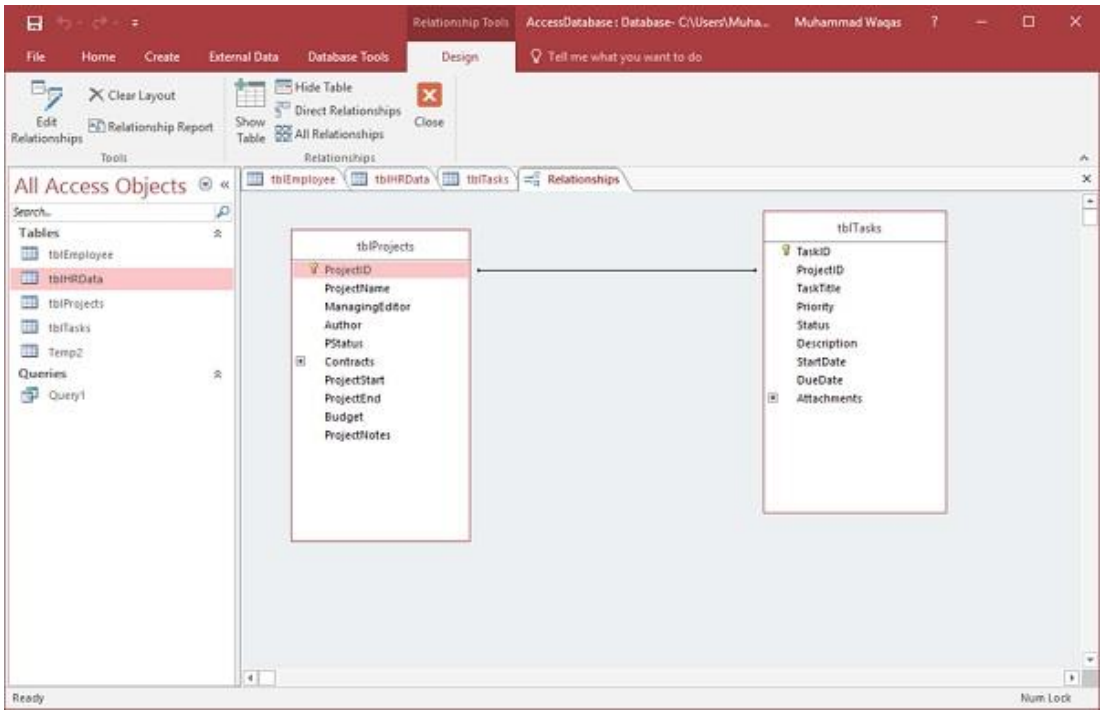
- **Database Creation** − Create your Microsoft Access database and specify what kind of data you will be storing.
- **Data Input** −After your database is created, the data of every business day can be entered into the Access database.
- **Query** −This is a fancy term to basically describe the process of retrieving information from the database.
- **Report** (optional) − Information from the database is organized in a nice presentation that can be printed in an Access Report.

Why Create Table Relationships?

MS Access uses table relationships to join tables when you need to use them in a database object. There are several reasons why you should create table relationships before you create other database objects, such as forms, queries, macros, and reports.

- To work with records from more than one table, you often must create a query that joins the tables.
- The query works by matching the values in the primary key field of the first table with a foreign key field in the second table.
- When you design a database, you divide your information into tables, each of which has a primary key and then add foreign keys to related tables that reference those primary keys.
- These foreign **key-primary key pairings** form the basis for table relationships and multi-table queries.

Example Relationships Database.



Lab Task

1. Create tables to show following one to one (1:1) relationship. Insert at least 5 records in each table. A nurse may be in-charge of a care center. A care center must have one nurse in-charge.

nurse | care_center

Field Name	Data Type
nurse_id	AutoNumber
nurse_name	Short Text
date_of_birth	Date/Time

nurse | care_center

Field Name	Data Type
center_id	AutoNumber
location	Short Text
nurse_incharge	Number
date_assigned	Date/Time

nurse | care_center

nurse_id	nurse_name	date_of_birth
1	Jogii	07/08/2020
2	Heer	29/08/2020
3	Jenna	20/08/2020
4	James	12/08/2020
5	Sara	03/08/2020

nurse | care_center

center_id	location	nurse_incha	date_assign
1	Hyd	1	30/08/2020
2	Jamshoro	2	18/08/2020
3	Karachi	3	05/08/2020
4	Dadu	4	29/08/2020
5	Larkana	5	23/08/2020
(New)		0	

2. **Create a database of University that consists of 03 tables (i.e. Student, Department, Teacher) and select appropriate primary key in each table.**

department | student | teacher | check

dept_id	dept_name	Click to Add
1	CS	
2	ES	
3	EL	
4	TL	
5	SW	
(New)		

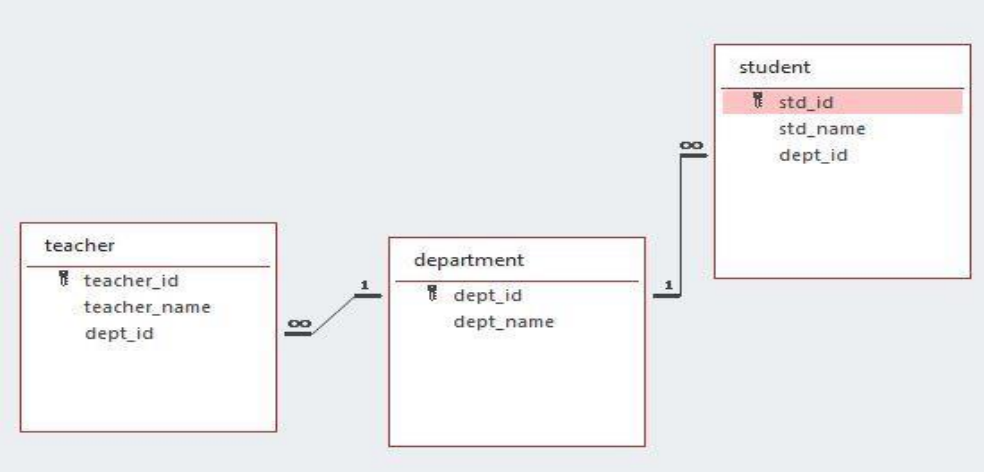
department | student | teacher | chec

teacher_id	teacher_nar	dept_id
1	Sir Rizwan	1
2	Sir Anwar	2
3	Sir Irfan	1
4	Sir Mahveer	1
5	Sir Kareem	3
(New)		0

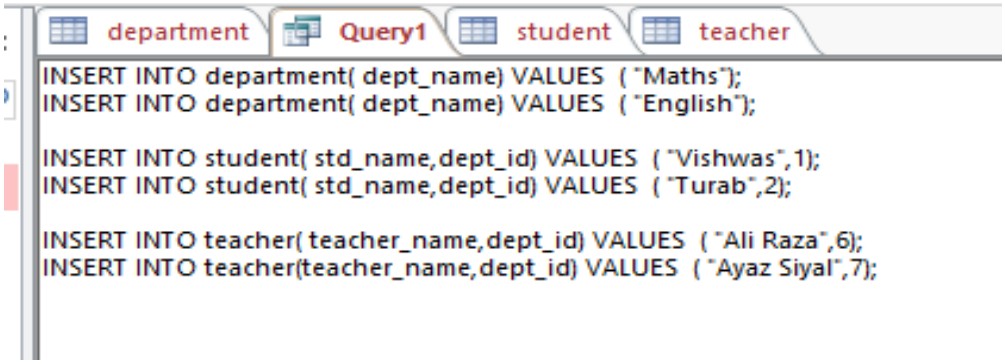
department | student | teacher | chec

std_id	std_name	dept_id
1	Ali	1
2	Akbar	1
3	Ahmed	2
4	Jan	1
5	Sarang	3
(New)		0

3. Create relationship among the three tables.

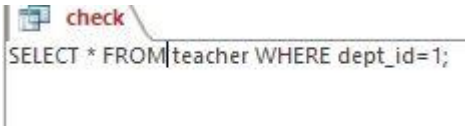


4. Populate the tables with appropriate data.

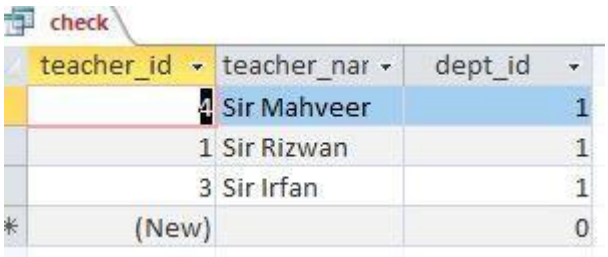


5. Write down a query of TASK#02 database, in which user can find that which teacher is teaching students of Computer System Department.

Query:



Result of Query:



teacher_id	teacher_nar	dept_id
4	Sir Mahveer	1
1	Sir Rizwan	1
3	Sir Irfan	1
(New)		0

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 2

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: Creating Form and generating Reports on database in MS Access.

Tools: MicroSoft Access

Introduction:

Create a form from an existing table or query in Access:

To create a form from a table or query in your database, in the Navigation Pane, click the table or query that contains the data for your form, and on the Create tab, click Form.

Access creates a form and displays it in Layout view. You can make design changes like adjusting the size of the text boxes to fit the data, if necessary. For more information, see the article on using the form tool.

Create a blank form in Access

1. To create a form with no controls or preformatted elements: On the Create tab, click Blank Form. Access opens a blank form in Layout view, and displays the Field List pane.
2. In the Field List pane, click the plus sign (+) next to the table or tables that contain the fields that you want to see on the form.
3. To add a field to the form, double-click it or drag it onto the form. To add several fields at once, hold down CTRL and click several fields, and then drag them onto the form at the same time.

   Note: The order of the tables in the Field List pane can change, depending on which part of the form is currently selected. If you are not able to add a field to the form, try selecting a different part of the form and then try adding the field again.

4. Use the tools in the Controls group on the Form Layout Tools tab to add a logo, title, page numbers, or the date and time to the form.
5. If you want to add a wider variety of controls to the form, click Design and use the tools in the Controls group.

Creating Reports:

Reports organize and summarize data for viewing online or for printing. A detail report displays all of the selected records. You can include summary data such as totals, counts, and percentages in a detail report. A summary report does not list the selected records but instead summarizes the data and presents totals, counts, percentages, or other summary data only. Access has several report generation tools that you can use to create both detail and summary reports quickly.

To use the Report button:

1. Open the Navigation pane.
2. Click the table or query on which you want to base your report.
3. Activate the Create tab.
4. Click the Report button in the Reports group. Access creates your report and displays your report in Layout view. You can modify the report.

To create a report by using the Report Wizard:

Open the Report Wizard

1. Activate the Create tab.
2. Click Report Wizard in the Reports group. The Report Wizard appears.

Lab Task

1. Create forms for each table created in lab#1.

department

dept_id	dept_name
1	CS
2	ES
3	EL
4	TL
5	SW
(New)	

student

std_id	std_name	dept_id
1	Ali	1
2	Akbar	1
3	Ahmed	2
4	Jan	1
5	Sarang	3
(New)		0

teacher

teacher_id	teacher_name	dept_id
1	Sir Rizwan	1
2	Sir Anwar	2
3	Sir Irfan	1
4	Sir Mahveer	1
5	Sir Kareem	3

2. **Generate reports for each table created in lab#1.**

department

dept_id	dept_name
1	CS
2	ES
3	EL
4	TL
5	SW

student

dept_id	std_id	std_name
1		
	4	Jan
	2	Akbar
	1	Ali
2		
	3	Ahmed
3		
	5	Sarang

teacher

dept_id	teacher_id	teacher_name
1		
	3	Sir Irfan
	1	Sir Rizwan
	4	Sir Mahveer
2		
	2	Sir Anwar
3		
	5	Sir Kareem

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 3

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To be familiar with Table creation and population of table.

Tools: MySql / Oracle.

Creating Databases and Tables

In order to be able to add and manipulate data, you first have to create a database. There's not much to this. You're creating just a container in which you will add tables. Creating a table is more involved and offers many choices.

There are a few basic things to decide when creating a structure for your data:

- The number of tables to include in your database, as well as the table names
- For each table, the number of columns it should contain, as well as the column names
- For each column, what kind of data is to be stored.

Creating a Database:

Creating a database is simple, mostly because there's nothing much to it. Use the SQL statement CREATE DATABASE. You will have to provide a name for the database with this SQL statement. You could call it something bland like db1.

```
CREATE DATABASE rookery;
```

Creating Tables:

The next step for structuring a database is to create tables. Although this can be complicated, we'll keep it simple to start. We'll initially create one main table and two smaller tables for reference information. The main table will have a bunch of columns, but the reference tables will have only a few columns.

```
CREATE TABLE birds (
Bird_id INT AUTO_INCREMENT PRIMARY KEY,
Scientific_name VARCHAR (255) UNIQUE,
Common_name VARCHAR (50),
Family_id INT,
Description TEXT);
```

This SQL statement creates the table birds with five fields, or columns, with commas separating the information about each column.

The names of the columns can be anything other than words that are reserved for SQL statements, clauses, and functions. Actually, you can use a reserve word, but it must always be given within quotes to distinguish it.

Adding data to your table

Let's inject some data into that table. We'll add information for a fictional member of the editorial staff. The data to be added is:

- Name: Olivia
- ID: 01
- Email: olivia@company.com

The command to add this would be:

INSERT INTO editorial (id, name, email) VALUES (01,"Olivia","olivia@company.com");
You can view the information added to the table with the command:

Lab Task

1. **Create the DEPT table with field names DEPTNO, DNAME and LOCATION. Take datatypes accordingly.**

Solution:

Result Grid | Filter Rows:

deptno	deptname	loc
1	CS	MUET
2	SW	MUET
3	ES	MUET
4	TL	MUET
5	EL	MUET
NULL	NULL	NULL

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
deptno	INT	☐	☑	☐	☐	☐	☐	☐	☐	
ename	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	☐	
job	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	☐	
mgr	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
hiredate	DATE	☐	☑	☐	☐	☐	☐	☐	☐	
sal	INT	☐	☑	☐	☐	☐	☐	☐	☐	
comm	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

2. **Create EMP table with columns EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO. Make EMPNO as primary key and DEPTNO as foreign key.**

Solution:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
deptno	INT	☐	☑	☐	☐	☐	☐	☐	☐	
ename	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	☐	
job	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	☐	
mgr	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
hiredate	DATE	☐	☑	☐	☐	☐	☐	☐	☐	
sal	INT	☐	☑	☐	☐	☐	☐	☐	☐	
comm	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

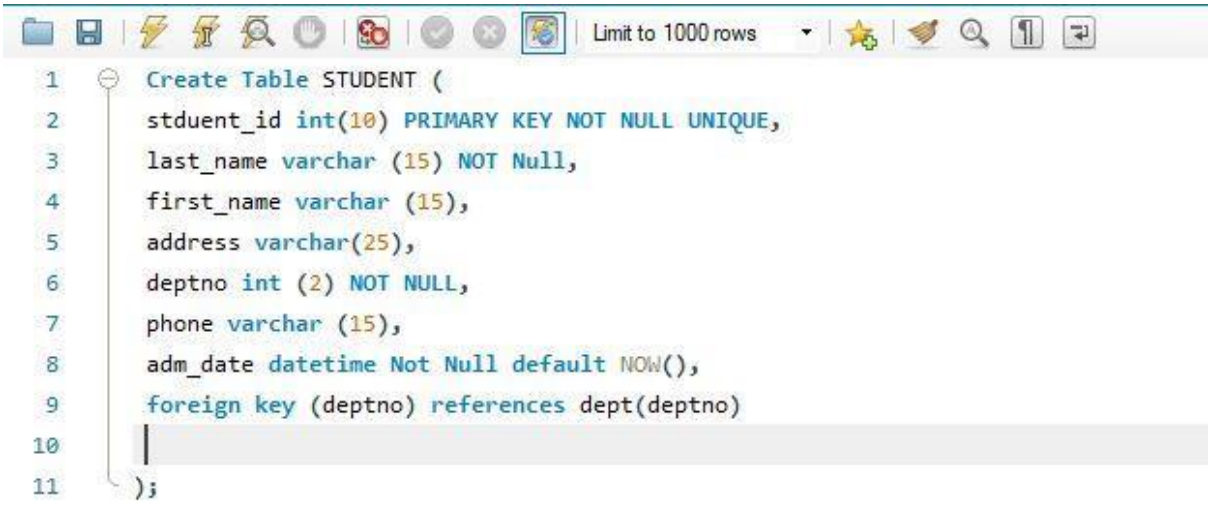
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

empno	deptno	ename	job	mgr	hiredate	sal	comm
1	1	Raza	Clerk	NULL	2020-02-02	10000	NULL
2	1	Ali	Asst:Clerk	NULL	2020-05-03	7000	NULL
3	1	Basit	Lab:Asst	NULL	2020-09-13	13000	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. **Create the STUDENT table based on the following table instance chart. Choose the appropriate data types and be sure to add integrity constraints.**

Column_ Name	STUDENT_ ID	LAST_ NAME	FIRST_NAM E	ADDRESS	DEPTNO	PHONE	ADM_ DATE
Key Type	PK				FK		
Null/ Unique	NN,U	NN			NN		NN
FK Ref Table					DEPT		
FK Ref Column					DEPTNO		
Default Value							System Date
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	NUMBER	VARCHAR2	DATE
Length	10	15	15	25	2	15	

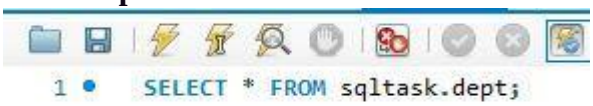
Solution:

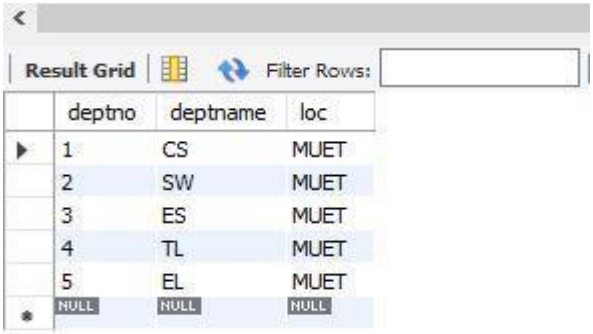


4. **Add at least 5 rows in each table by using constant values of your own choice.**

Solution:

Department





deptno	deptname	loc
1	CS	MUET
2	SW	MUET
3	ES	MUET
4	TL	MUET
5	EL	MUET
NULL	NULL	NULL

Emp Table:

```
SELECT * FROM sqltask.emp;
```

empno	deptno	ename	job	mgr	hiredate	sal	comm
1	1	Raza	Clerk	NULL	2020-02-02	10000	NULL
2	1	Ali	Asst:Clerk	NULL	2020-05-03	7000	NULL
3	1	Basit	Lab:Asst	NULL	2020-09-13	13000	NULL
4	2	Ahmed	Clerk	NULL	2020-09-12	8000	NULL
5	3	Ali Jan	Supervisor	NULL	2020-02-11	6000	NULL

Student:

```
SELECT * FROM sqltask.student;
```

stduent_id	last_name	first_name	address	deptno	phone	adm_date
1	Tahir	Mohammad	Hyd	1	0300030302	2020-02-...
2	Sarang	Mohammad	Hyd	2	0300352333	2020-02-...
3	Anees	Ahmed	Thatta	1	0300200212	2020-02-...
4	Ovais	Mohammad	Badin	1	0312232322	2020-02-...
5	Danish	Azeem	Larkana	3	0320109292	2020-02-...

student 1

5. Create the employee table based on the structure of EMP table.

Solution:

```
CREATE TABLE employee AS SELECT * FROM EMP;
```

Script Output

Task completed in 0.053 seconds

```
Table EMPLOYEE created.
```

6. **Create a table MY_EMPLOYEE based on the structure of EMP table. Include the columns Empno, Ename, and Sal. Name them in new table as Id, Name, and Salary.**

Solution:

```
1    CREATE TABLE my_employees AS SELECT empno id, deptno , ename e_name, job , mgr , hiredate , sal salary, comm  FROM emp;
```

```
1 •  SELECT * FROM sqltask.my_employees;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	deptno	e_name	job	mgr	hiredate	salary	comm
1	1	Raza	Clerk	NULL	2020-02-02	10000	NULL
2	1	Ali	Asst:Clerk	NULL	2020-05-03	7000	NULL
3	1	Basit	Lab:Asst	NULL	2020-09-13	13000	NULL
4	2	Ahmed	Clerk	NULL	2020-09-12	8000	NULL
5	3	Ali Jan	Supervisor	NULL	2020-02-11	6000	NULL

7. **Rename employee1 table to Employee.**

Solution:

```
1    Alter Table employee1 rename to employee ;
```

8. **Drop employee table.**

Solution:

emp | employee2

```
1    Drop Table employee ;
```

9. Add a comment to the student table describing the table.

Solution:

```
1 • ALTER TABLE student COMMENT 'Details of students';
2 • SELECT table_comment
3   FROM INFORMATION_SCHEMA.TABLES
4   WHERE TABLE_NAME = 'student'
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_COMMENT
Details of students

10. Add a comment on ADM_DATE column as ‘Admission Date of student’.

Solution:

```
1 • ALTER TABLE `student` CHANGE `ADM_DATE` `ADM_DATE` DATETIME NOT NULL DEFAULT NOW() COMMENT 'Admission date of student';
2 • SHOW FULL COLUMNS FROM student;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Collation	Null	Key	Default	Extra	Privileges	Comment
STUDENT_ID	int	NULL	NO	PRI	NULL		select,insert,update,references	
LAST_NAME	varchar(15)	utf8mb4_0900_ai_ci	NO		NULL		select,insert,update,references	
FIRST_NAME	varchar(15)	utf8mb4_0900_ai_ci	YES		NULL		select,insert,update,references	
ADDRESS	varchar(25)	utf8mb4_0900_ai_ci	YES		NULL		select,insert,update,references	
DEPTNO	int	NULL	NO	MUL	NULL		select,insert,update,references	
PHONE	varchar(15)	utf8mb4_0900_ai_ci	YES		NULL		select,insert,update,references	
ADM_DATE	datetime	NULL	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED	select,insert,update,references	Admission date of student

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 4

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To retrieve data from database using SQL SELECT Statement.

Tools: MySql / Oracle.

Introduction:

The SQL SELECT Statement

- The SELECT statement is used to select data from a database.
- The data returned is stored in a result table, called the result-set.

SELECT Syntax

```
SELECT column1, column2 FROM table_name;
```

Here, column1, column2 are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

SELECT Column Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table:

Example

```
SELECT CustomerName, City FROM Customers;
```

Lab Task

1. Create a query to display unique departments from EMP table.

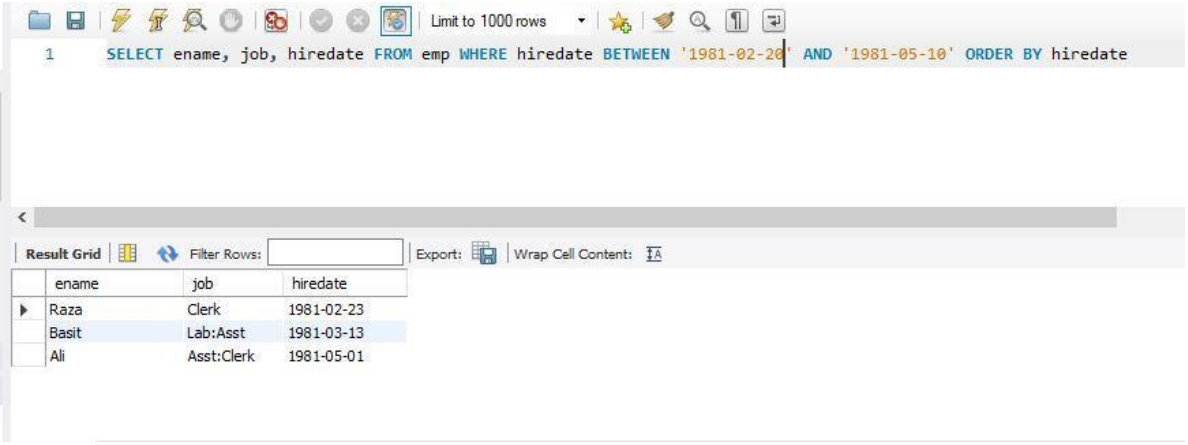
Solution:

```
1 •  SELECT DISTINCT deptno FROM emp
```

deptno
1
2
3

2. Display the employee name, job, and joining date of employees hired between February 20, 1981 and May 1, 1981. Order the query in ascending order by joining date.

Solution:



ename	job	hiredate
Raza	Clerk	1981-02-23
Basit	Lab:Asst	1981-03-13
Ali	Asst:Clerk	1981-05-01

3. Display the names of all employees who have two 'L' in their name and are in department 30 or their manager is 7782.

Solution:

```
1    SELECT ename
2    FROM emp
3    WHERE (ename LIKE '%l%l') and (deptno=1 or mgr=7782);
```

ename
Ali Ajmal

4. Display the name, salary, and commission for all employees whose commission amount is greater than their salary increased by 10%.

Solution:

```
1 •  SELECT ename "Employee", sal "Monthly Salary", comm
2    FROM emp
3    WHERE comm > sal * 1.1
4
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Employee	Monthly Salary	comm
MARTIN	1250	1400

5. Display the name and salary of employees who earn more than 1500 and are in department 10 or 30. Label the columns Employee and Monthly Salary, respectively.

Solution:

```
1 •  SELECT ename "Employee", sal "Monthly Salary"
2    FROM emp
3    WHERE sal > 1500  AND deptno IN (1,3)
4
```

Result Grid | Filter Rows: | Export: | Wrap C

Employee	Monthly Salary
Raza	10000
Ali Ajmal	7000
Basit	13000
Ali Jan	6000
KING	5000
ALLEN	1600
BLAKE	2350
CLARK	2450

6. List out the employees whose name start with S and ends with H.

Solution:

```
1 •  SELECT ename "Employee", sal "Monthly Salary"
2    FROM emp
3    WHERE sal > 1500  AND deptno IN (1,3)
4
```

Result Grid | Filter Rows: | Export: | Wrap C

Employee	Monthly Salary
Raza	10000
Ali Ajmal	7000
Basit	13000
Ali Jan	6000
KING	5000
ALLEN	1600
BLAKE	2350
CLARK	2450

7. Write a query that produces following for each employee: earns monthly but wants. Label the column Dream Salaries.

Solution:

```
3 select ename || ' earns '|| SAL || ' monthly but want '|| (SAL*3) as "Dream Salaries"
4 from emp;
5
6
7
8
```

Results | Explain | Describe | Saved SQL | History

Dream Salaries
KING earns 5000 monthly but want 15000
BLAKE earns 2850 monthly but want 8550
CLARK earns 2450 monthly but want 7350
JONES earns 2975 monthly but want 8925
SCOTT earns 3000 monthly but want 9000

8. Display the name job and salary for all employees whose job is clerk or analyst and their salary is not equal to 1000, 3000, or 5000.

Solution:

```
1  SELECT ename, job, sal
2      FROM emp
3      WHERE job IN ('CLERK', 'ANALYST')
4      AND sal NOT IN (1000, 3000, 5000)
5
```

Result Grid | Filter Rows: | Export

ename	job	sal
Raza	Clerk	10000
Ahmed	Clerk	8000
SMITH	CLERK	800
JAMES	CLERK	950

9. Display the names of all employees where the third letter of their name is an A.

Solution:

```
1  SELECT ename
2  FROM emp
3  WHERE ename LIKE '__A%'
4
```

Result Grid | Filter Rows:

ename
BLAKE
CLARK

10. Display name, salary and commission for all employees who earn commission. Sort the result in descending order of salary and commission.

Solution:

```
1  SELECT ename, sal, comm
2  FROM emp WHERE comm IS NOT NULL
3  ORDER BY sal DESC, comm DESC
4
```

Result Grid | Filter Rows: | Export:

ename	sal	comm
KING	5000	11
SCOTT	3000	
FORD	3000	
JONES	2975	
CLARK	2450	
BLAKE	2350	
ALLEN	1600	300
TURNER	1500	
WARD	1250	500
MARTIN	1250	1400
JAMES	950	
SMITH	800	

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 5

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: **To use Single row functions in SQL queries.**

Tools: MySql, Oracle.

Introduction:

Single Row functions - Single row functions are the one who work on single row and return one output per row. For example, length and case conversion functions are single row functions.

Single row functions

Single row functions can be character functions, numeric functions, date functions, and conversion functions. Note that these functions are used to manipulate data items. These functions require one or more input arguments and operate on each row, thereby returning one output value for each row. Argument can be a column, literal or an expression. Single row functions can be used in SELECT statement, WHERE and ORDER BY clause. Single row functions can be -

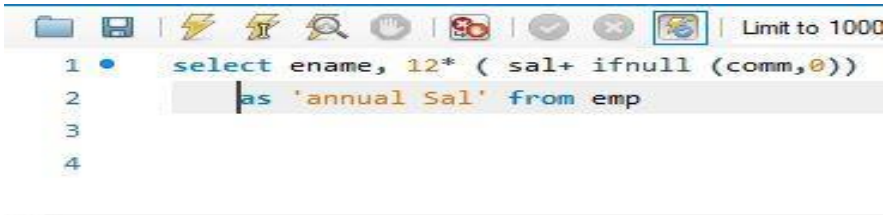
- **General functions** - Usually contains NULL handling functions. The functions under the category are NVL, NVL2, NULLIF, COALESCE, CASE, DECODE.
- **Case Conversion functions** - Accepts character input and returns a character value. Functions under the category are UPPER, LOWER and INITCAP.
    - UPPER function converts a string to upper case.
    - LOWER function converts a string to lower case.
    - INITCAP function converts only the initial alphabets of a string to upper case.

- **Character functions** - Accepts character input and returns number or character value. Functions under the category are CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD, TRIM and REPLACE.
    - CONCAT function concatenates two string values.
    - LENGTH function returns the length of the input string.
    - SUBSTR function returns a portion of a string from a given start point to an end point.
    - INSTR function returns numeric position of a character or a string in a given string.
    - LPAD and RPAD functions pad the given string Upto a specific length with a given character.
    - TRIM function trims the string input from the start or end.
    - REPLACE function replaces characters from the input string with a given character.
- **Date functions** - Date arithmetic operations return date or numeric values. Functions under the category are MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND and TRUNC.
    - MONTHS_BETWEEN function returns the count of months between the two dates.
    - ADD_MONTHS function add 'n' number of months to an input date.
    - NEXT_DAY function returns the next day of the date specified.
    - LAST_DAY function returns last day of the month of the input date.
    - ROUND and TRUNC functions are used to round and truncates the date value.
- **Number functions** - Accepts numeric input and returns numeric values. Functions under the category are ROUND, TRUNC, and MOD.
    - ROUND and TRUNC functions are used to round and truncate the number value.
    - MOD is used to return the remainder of the division operation between two numbers.

Lab Task

1. Display the employee name and their annual salary including commission amount. If the Commission is null replace it with 0.

Task:



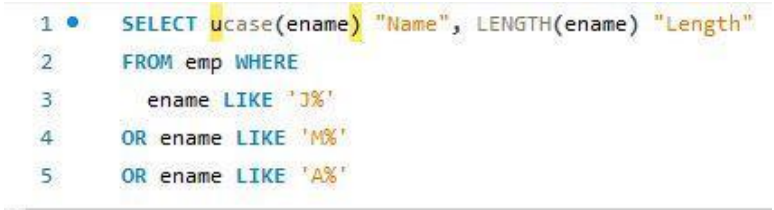
```
1  select ename, 12* ( sal+ ifnull (comm,0))
2      as 'annual Sal' from emp
3
4
```

Result Grid | Filter Rows: | Export:

ename	annual Sal
Raza	120000
Ali Ajmal	84000
Basit	156000
Ahmed	96000
Ali Jan	72000
KING	60132
TURNER	18000
SMITH	9600
ALLEN	22800
WARD	21000
JONES	35700
MARTIN	31800
BLAKE	28200
CLARK	29400
SCOTT	36000
JAMES	11400
FORD	36000

2. Display Employee's name with the first letter capitalized and all other letters lowercase and the length of their name, for all employees whose name starts with J, A, or M. Give each column appropriate names.

Task:



```
1  SELECT ucase(ename) "Name", LENGTH(ename) "Length"
2  FROM emp WHERE
3    ename LIKE 'J%'
4  OR ename LIKE 'M%'
5  OR ename LIKE 'A%'
```

Result Grid | Filter Rows: | Export: | Wrap Cell Co

Name	Length
ALI AJMAL	9
AHMED	5
ALI JAN	7
ALLEN	5
JONES	5
MARTIN	6
JAMES	5

3. Display the name, hiredate, and day of the week on which the employee started. Label the Column First Day.

Task:

```
1 •  select ename ,hiredate, dayname(hiredate)as first_day from emp
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ename	hiredate	first_day
Raza	1981-02-02	Monday
Ali Ajmal	1981-05-03	Sunday
Basit	1981-09-13	Sunday
Ahmed	2000-09-12	Tuesday
Ali Jan	2001-02-11	Sunday
KING	2000-01-11	Tuesday
TURNER	2001-02-11	Sunday
SMITH	2001-02-17	Saturday
ALLEN	2001-02-01	Thursday
WARD	2001-09-10	Monday
JONES	2001-12-11	Tuesday
MARTIN	2001-02-19	Monday
BLAKE	2001-02-11	Sunday
CLARK	2001-03-08	Thursday
SCOTT	2001-01-11	Thursday
JAMES	2002-04-11	Thursday
FORD	2003-02-11	Tuesday

4. For each employee, display the employee name and calculate the number of months between today and the date the employee was hired. Label the column MONTHS_WORKED. Order the result by the number of months employed. Round the number of months up to the closest whole number.

Task:

```
1 •  SELECT ename, ROUND(DATEDIFF (SYSDATE(), hiredate))
2        MONTHS_WORKED FROM emp
3    ORDER BY DATEDIFF(SYSDATE(), hiredate)
4
```

Result Grid | Filter Rows: | Export: | Wrap Cell Con

ename	MONTHS_WORKED
FORD	6393
JAMES	6699
JONES	6820
WARD	6912
CLARK	7098
MARTIN	7115
SMITH	7117
Ali Jan	7123
TURNER	7123
BLAKE	7123
ALLEN	7133
SCOTT	7154
Ahmed	7275
KING	7520
Basit	14214
Ali Ajmal	14347
Raza	14437

5. Display the Employee number, name, salary, and salary increase by 15% expressed as a whole number. Label the column New Salary.

Task:

```
1 •  SELECT empno, ename, sal,
2    ROUND (sal * 1.15, 0) "New Salary"
3    FROM emp
4
```

Result Grid | Filter Rows: | Export:

empno	ename	sal	New Salary
1	Raza	10000	11500
2	Ali Ajmal	7000	8050
3	Basit	13000	14950
4	Ahmed	8000	9200
5	Ali Jan	6000	6900
7339	KING	5000	5750
7344	TURNER	1500	1725
7369	SMITH	800	920
7499	ALLEN	1600	1840
7521	WARD	1250	1438
7566	JONES	2975	3421
7654	MARTIN	1250	1438
7693	BLAKE	2350	2703
7732	CLARK	2450	2818
7788	SCOTT	3000	3450
7900	JAMES	950	1093
7902	FORD	3000	3450

6. Display the employee name, hiredate, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to "Sunday, the seventh of September, 1981".

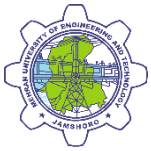
Task:

```
1  select ename,hiredate,DATE_FORMAT(DATE_ADD(DATE_ADD(hiredate,interval 6 month),
2  INTERVAL (9 - IF(DAYOFWEEK(DATE_ADD(hiredate,interval 6 month))=1,8,
3  DAYOFWEEK(DATE_ADD(hiredate,interval 6 month))))DAY),'%W, the %D of %D of %M,%Y') AS 'REVIEW'
4
5  FROM employee;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ename	hiredate	REVIEW
hania	1981-06-29	Monday, the 4th of 4th of January,1982
Lawizal	2021-09-08	Monday, the 14th of 14th of March,2022
Alia	1981-06-29	Monday, the 4th of 4th of January,1982
Maha	1981-04-19	Monday, the 26th of 26th of October,1981
Mahajabeen	1981-02-09	Monday, the 10th of 10th of August,1981
Lalital	2021-09-08	Monday, the 14th of 14th of March,2022
Aisha	1981-09-08	Monday, the 15th of 15th of March,1982
laleeta	1981-05-21	Monday, the 23rd of 23rd of November,1981
Javeria	1982-05-29	Monday, the 6th of 6th of December,1982

Result Grid | Form Editor | Field Types



DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 6

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To use Group functions in SQL queries.

Tools: MySQL, Oracle

Introduction:

Group Function: Group functions are mathematical functions to operate on sets of rows to give one result per set.

Syntax:

The general syntax for using Group functions is:
SELECT <column>, group_function (column) FROM <table>
WHERE <condition> [GROUP BY <column>]
[ORDER BY <column>]

Note that the column on which the group function is applied must exist in the SELECT column list.

Types of Group Function

Here are the different types of the Group function in SQL:

i. **AVG, MIN, MAX, and SUM function:** you can use AVG, MIN, MAX and SUM for numeric data.

Syntax:

SELECT AVG(column_name),MIN(column_name),

SUM(column_name), MAX(column_name)

FROM *t*able1 WHERE condition*;*

ii. **COUNT() Function:**

- COUNT(*) function return the number of in a table;
- COUNT (expr) returns the number of rows with non-null values for the expr.
- COUNT (DISTINCT expr) returns the number of distinct non-null values of the expr.

Syntax:

```
SELECT COUNT(Column_name)
FROM table1
WHERE condition;
```

iii. **NVL() Function:** The NVL() function forces Group functions to include the null values.

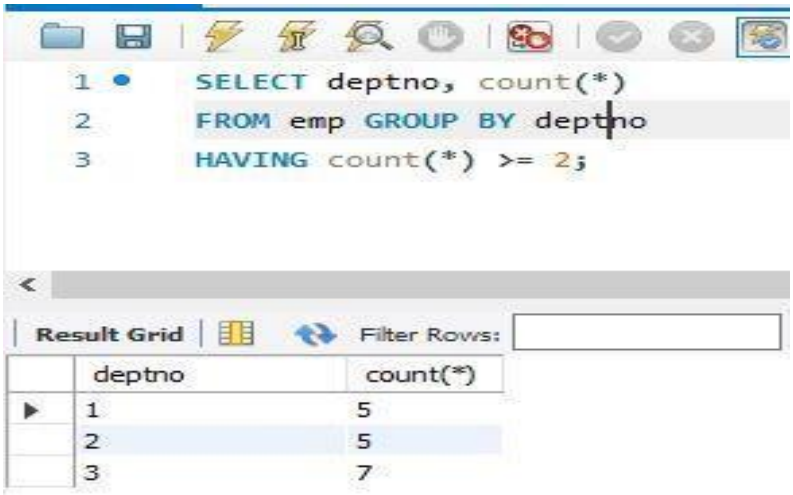
Syntax:

```
SELECT NVL(column_name)  FROM table11;
```

Lab Task

1. List out the department numbers that have at least 4 employees.

Task:



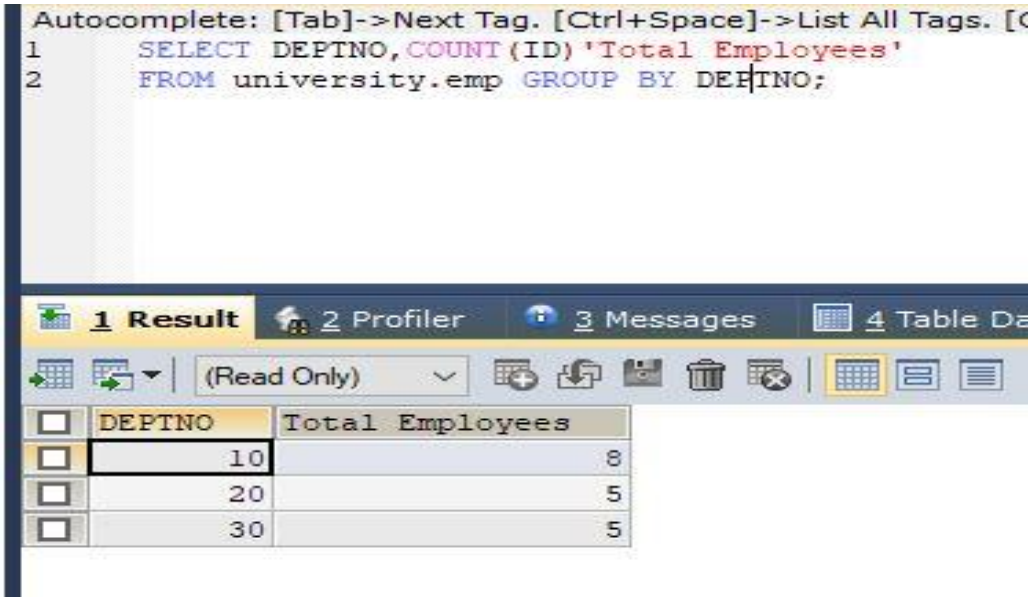
```
1  SELECT deptno, count(*)
2  FROM emp GROUP BY deptno
3  HAVING count(*) >= 2;
```

Result Grid | Filter Rows:

deptno	count(*)
1	5
2	5
3	7

2. Display the number of employees in each department.

Task:



```
Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List All Tags. [C
1   SELECT DEPTNO,COUNT(ID)'Total Employees'
2   FROM university.emp GROUP BY DEPTNO;
```

1 Result | 2 Profiler | 3 Messages | 4 Table Da

(Read Only)

DEPTNO	Total Employees
10	8
20	5
30	5

3. Display the manager no 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 1000. Sort the output in descending order of salary.

Task:

```
1 •  SELECT mgr, MIN(sal) FROM emp
2     WHERE mgr IS NOT NULL
3     GROUP BY mgr
4     HAVING MIN(sal) > 1000
5     ORDER BY MIN(sal) DESC;
```

Result Grid | Filter Rows:

mgr	MIN(sal)
103	13000
101	10000
104	8000
102	7000
105	6000
7821	5000
7566	3000
7339	2450
7839	2350

4. Find the most recently hired employee in each department.

Task:

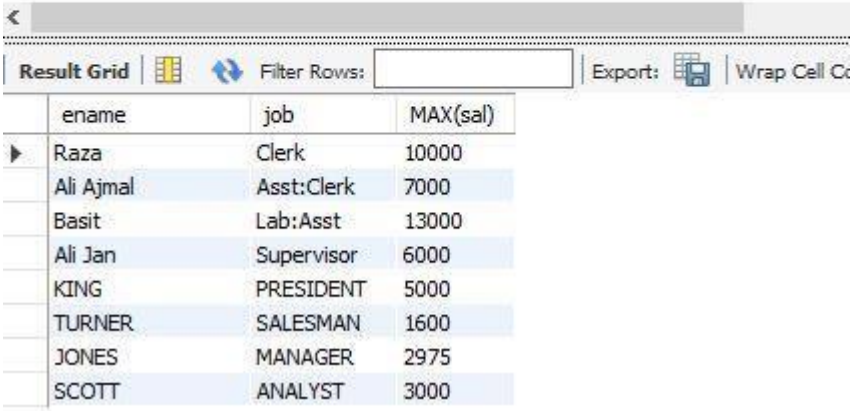
```
1 •  SELECT * FROM emp e
2    WHERE hiredate IN (SELECT max(hiredate)
3    FROM emp WHERE e.deptno = deptno )
4    ORDER BY hiredate DESC;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

empno	deptno	ename	job	mgr	hiredate	sal	comm
7902	2	FORD	ANALYST	7566	2003-02-11	3000	
7900	3	JAMES	CLERK	7698	2002-04-11	950	
7732	1	CLARK	MANAGER	7339	2001-03-08	2450	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

5. List the highest salary paid for each job.

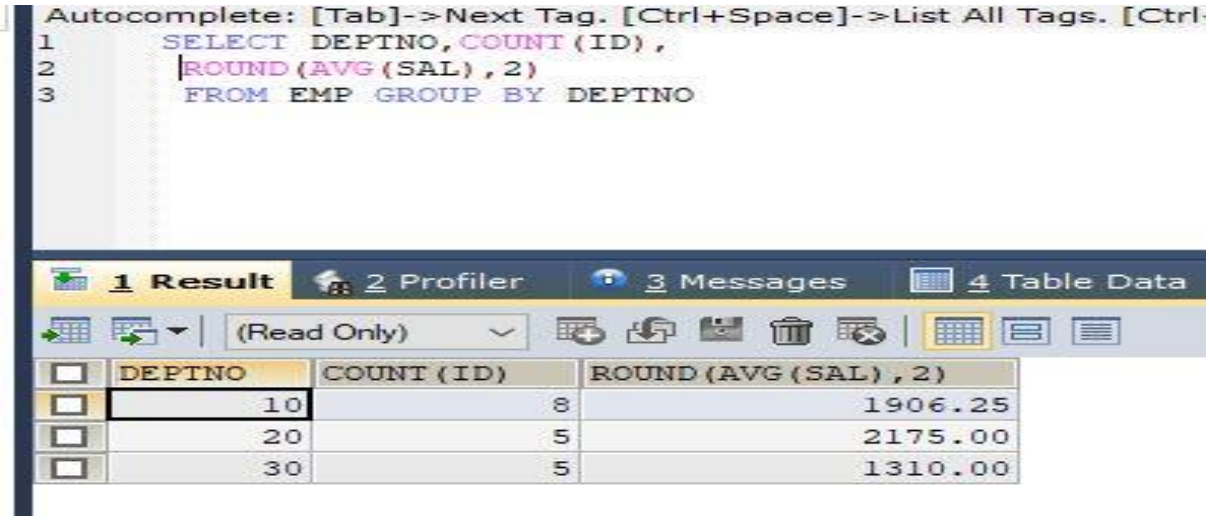
Task:



ename	job	MAX(sal)
Raza	Clerk	10000
Ali Ajmal	Asst:Clerk	7000
Basit	Lab:Asst	13000
Ali Jan	Supervisor	6000
KING	PRESIDENT	5000
TURNER	SALESMAN	1600
JONES	MANAGER	2975
SCOTT	ANALYST	3000

6. Display the department number, number of employees in that dept and the average salary for all employees in that department. Round the average salary to two decimal places.

Task:



DEPTNO	COUNT(ID)	ROUND(AVG(SAL),2)
10	8	1906.25
20	5	2175.00
30	5	1310.00

7. Write a query that will display the difference between the highest and lowest salaries. Label the column DIFFERENCE.

Task:

```
1    SELECT MAX(sal) - MIN(sal) DIFFERENCE
2    FROM emp;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

DIFFERENCE
12200

8. Display the number of employees with same job.

Task:

```
1    SELECT job, COUNT(*)
2    FROM emp
3    GROUP BY job;
4
```

Result Grid | Filter Rows:

job	COUNT(*)
Clerk	4
Asst:Clerk	1
Lab:Asst	1
Supervisor	1
PRESIDENT	1
SALESMAN	4
MANAGER	3
ANALYST	2

9. Determine the number of managers without listing them. (Hint: Use MGR column)

Task:

```
1 •  Select count(distinct mgr)"Number oF Manager"
2    from emp;
```

Result Grid | Filter Rows: | Export: | Wrap Cell

	Number oF Manager
▶	11

10. Display the job title and total monthly salary for each job title with a total payroll exceeding 5000. Exclude salespeople and sorts the list by the total monthly salary.

Task:

```
1 •  SELECT  job, SUM(sal) "Monthly Salary"
2     From emp  WHERE  job NOT LIKE 'SALES%'
3     GROUP BY  job HAVING SUM(sal)>5000
4      ORDER BY  SUM(sal) ;
```

Result Grid | Filter Rows: | Export: | Wr

	job	Monthly Salary
▶	Supervisor	6000
	ANALYST	6000
	Asst:Clerk	7000
	MANAGER	7775
	Lab:Asst	13000
	Clerk	19750

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 7

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To retrive data from multiple tables (Joins).

Tools: MySQL, Oracle

Introduction:

SQL Join: A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Let us look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

Example

```
SELECT Orders.OrderID,Customers.CustomerName,Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

and it will produce something like this:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Types of SQL Joins

Here are the different types of the JOINs in SQL:

i. **SQL Inner Join:** The INNER JOIN keyword selects records that have matching values in both tables.

Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

ii. **SQL Left Join:** The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side if there is no match.

Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

iii. **SQL Right join:** The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

Syntax:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

iv. **SQL Full join:** The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!

Syntax:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

v. **SQL Self join:** A self-JOIN is a regular join, but the table is joined with itself.

Syntax:

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and T2 are different table aliases for the same table.

Lab Task

1. List the name of the employees with the name of their immediate higher authority.

Task:

```
2     SELECT A.ENAME "EMPLOYEE", B.ENAME "REPORTS TO"
3           FROM EMP A, EMP B WHERE A.MGR=B.ID;
4
```

1 Result | 2 Profiler | 3 Messages | 4 Table Data

(Read Only)

EMPLOYEE	REPORTS TO
MERKEL	BUSH
PUTIN	BUSH
TOOSK	PUTIN
CARNEGIE	PUTIN
THATCHER	TOOSK
FORD	CARNEGIE

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

Task:

```
1     SELECT DISTINCT e.job, d.loc
2     FROM emp e, dept d
3     WHERE e.DEPTNO = d.dept_no
4     AND e.DEPTNO = 30
```

1 Result | 2 Profiler | 3 Messages

(Read Only)

job	loc
SALESMAN	CHICAGO
CLERK	CHICAGO

3. Display the employee name and department name for all employees who have an A in their name.

Task:

```
1     SELECT e.`ENAME`, d.`dname`
2       FROM emp e, dept d
3         WHERE e.`DEPTNO` = d.dept_no
4     AND e.`ENAME` LIKE '%A%'
```

1 Result | 2 Profiler | 3 Messages | 4 Table D

(Read Only)

ENAME	dname
THATCHER	RESEARCH
CARNEGIE	RESEARCH
BAROSSO	SALES
WALTON	SALES
CHIRACK	SALES
GATES	SALES

4. Display the employee name and department name for all employees who work in DALLAS.

Task:

```
1      SELECT e.ename, e.job, e.deptno, d.dname
2      FROM emp e, dept d
3      WHERE e.deptno = d.dept_no
4      AND d.loc = 'DALLAS'
5
```

1 Result | 2 Profiler | 3 Messages | 4 Table Data

(Read Only)

ename	job	deptno	dname
THATCHER	CLERK	20	RESEARCH
PUTIN	MANAGER	20	RESEARCH
CARNEGIE	ANALYST	20	RESEARCH
FORD	CLERK	20	RESEARCH
TOOSK	ANALYST	20	RESEARCH
JACK	TEACHER	20	RESEARCH

5. Display the employee name employee number along with their manager name and manager's number for all employees including KING, who has no manager. Label the columns employee, Emp#, Manager, and Mgr#, respectively.

Task:

```
1      SELECT w.ename "Employee", w.id "EMP#",
2           m.ename "Manager", m.id "Mgr#"
3        FROM emp w
4       LEFT OUTER JOIN emp m
5      ON (w.mgr = m.id);
```

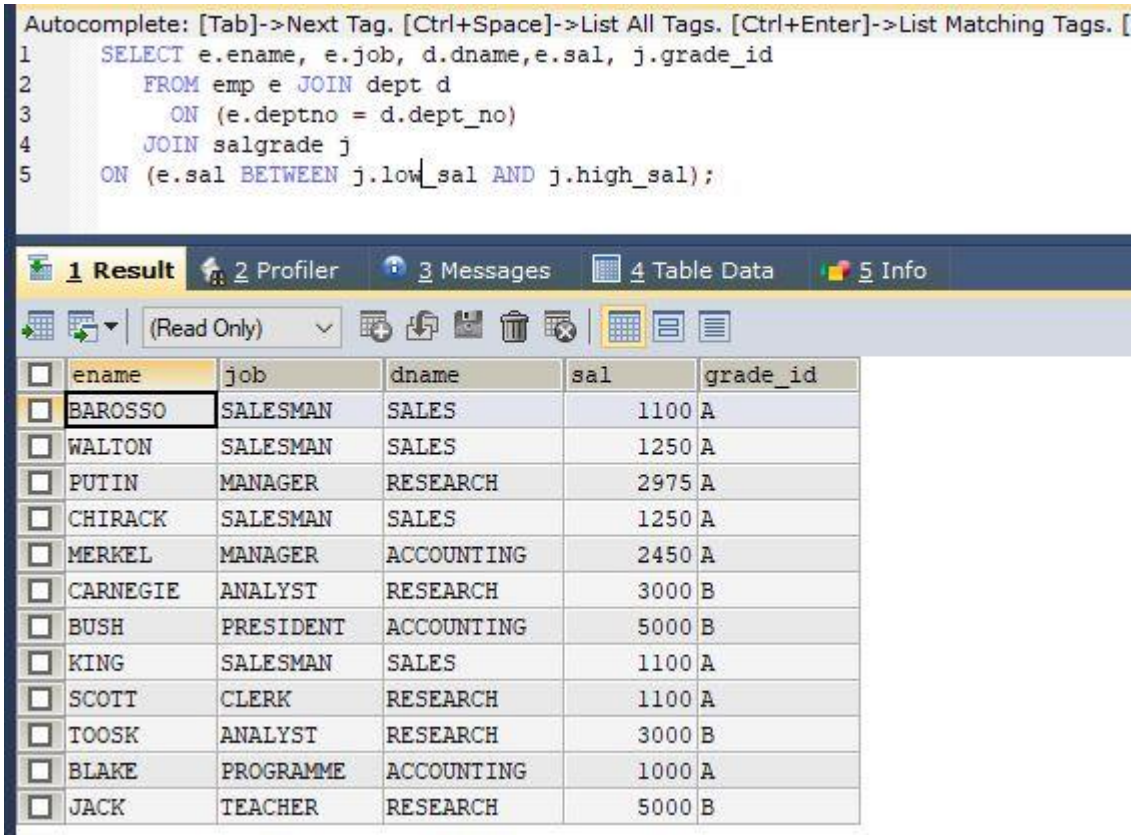
1 Result | 2 Profiler | 3 Messages | 4 Table

(Read Only)

Employee	EMP#	Manager	Mgr#
THATCHER	7369	TOOSK	7902
BAROSSO	7499	*(NULL)*	*(NULL)*
WALTON	7521	*(NULL)*	*(NULL)*
PUTIN	7566	BUSH	7839
CHIRACK	7654	*(NULL)*	*(NULL)*
MERKEL	7782	BUSH	7839
CARNEGIE	7788	PUTIN	7566
BUSH	7839	*(NULL)*	*(NULL)*
GATES	7844	*(NULL)*	*(NULL)*
FORD	7876	CARNEGIE	7788
BUFFETT	7900	*(NULL)*	*(NULL)*
TOOSK	7902	PUTIN	7566
BLAKE	7903	*(NULL)*	*(NULL)*
JACK	7904	*(NULL)*	*(NULL)*

6. Create a query that will display the name, job, department name, salary, grade for all employees.

Task:



```
Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List All Tags. [Ctrl+Enter]->List Matching Tags. [
1    SELECT e.ename, e.job, d.dname,e.sal, j.grade_id
2       FROM emp e JOIN dept d
3         ON (e.deptno = d.dept_no)
4       JOIN salgrade j
5    ON (e.sal BETWEEN j.low_sal AND j.high_sal);
```

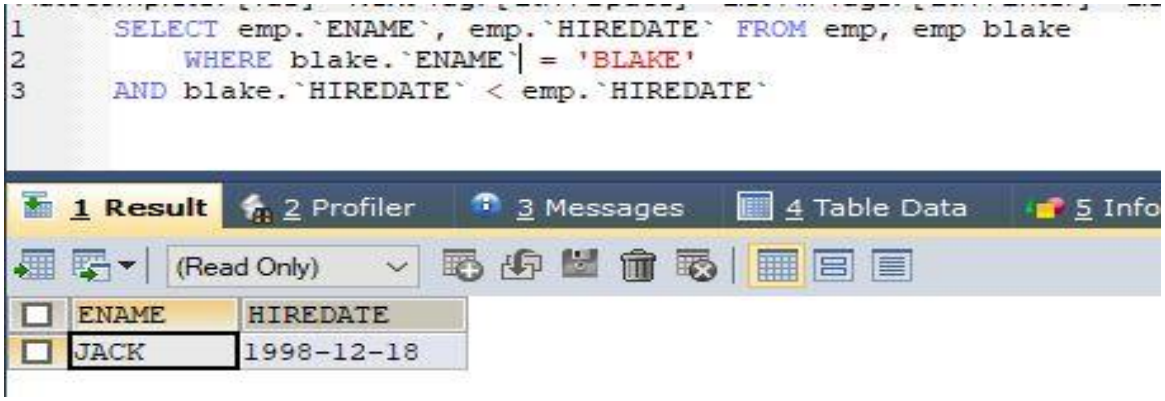
1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info

(Read Only)

ename	job	dname	sal	grade_id
BAROSSO	SALESMAN	SALES	1100	A
WALTON	SALESMAN	SALES	1250	A
PUTIN	MANAGER	RESEARCH	2975	A
CHIRACK	SALESMAN	SALES	1250	A
MERKEL	MANAGER	ACCOUNTING	2450	A
CARNEGIE	ANALYST	RESEARCH	3000	B
BUSH	PRESIDENT	ACCOUNTING	5000	B
KING	SALESMAN	SALES	1100	A
SCOTT	CLERK	RESEARCH	1100	A
TOOSK	ANALYST	RESEARCH	3000	B
BLAKE	PROGRAMME	ACCOUNTING	1000	A
JACK	TEACHER	RESEARCH	5000	B

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

Task:



```
1    SELECT emp.`ENAME`, emp.`HIREDATE` FROM emp, emp blake
2        WHERE blake.`ENAME` = 'BLAKE'
3    AND blake.`HIREDATE` < emp.`HIREDATE`
```

1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info

(Read Only)

ENAME	HIREDATE
JACK	1998-12-18

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

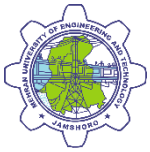
Task:

```
1   SELECT e.ename "Employee", e.hiredate "Emp Hiredate", m.ename "Manager", m.hiredate "Mgr Hiredate"
2           FROM emp e, emp m WHERE e.mgr = m.ID
3   AND e.hiredate < m.hiredate
```

1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info

(Read Only)

Employee	Emp Hiredate	Manager	Mgr Hiredate
THATCHER	1980-12-17	TOOSK	1981-12-03
PUTIN	1981-04-02	BUSH	1981-11-17
MERKEL	1981-06-09	BUSH	1981-11-17



DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 8

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To study and practice SQL sub-queries.

Tools: MySQL, Oracle

Introduction:

Sub-Query: A subquery is a SQL query nested inside a larger query.

- A subquery may occur in :
  - - A SELECT clause
  - - A FROM clause
  - - A WHERE clause
- The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.
- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.
- A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.

The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:

- Compare an expression to the result of the query.
- Determine if an expression is included in the results of the query.
- Check whether the query selects any rows.

Syntax:

```
SELECT column_name [, column_name]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
  (SELECT column_name [, column_name]
  FROM table1 [, table2]
  [WHERE])
```

- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

Lab Tasks

1. Display the employee name and hire date for all employees in the same department as BLAKE. Exclude Blake.

Task:

```
1    SELECT ename, hiredate
2    FROM emp
3    WHERE deptno = (SELECT deptno
4    FROM emp
5    WHERE ename = 'BLAKE')
6    AND ename != 'BLAKE'
```

1 Result | 2 Profiler | 3 Message

(Read Only)

ename	hiredate
MERKEL	1981-06-09
BUSH	1981-11-17

2. Display the employee name and salary for all employees who earn more than average salary.

Task:

```
1    SELECT ename, sal
2    FROM emp
3    WHERE sal > (SELECT AVG(sal)FROM emp)
4
```

1 Result | 2 Profiler | 3 Messages

(Read Only)

ename	sal
PUTIN	2975
MERKEL	2450
CARNEGIE	3000
BUSH	5000
TOOSK	3000
JACK	5000

3. Display the employee name, job and hire date for all employees who report to KING.

Task:

```
1     SELECT ENAME, JOB,HIREDATE
2     FROM emp
3   WHERE mgr = (SELECT id
4     FROM emp
5     WHERE ename = 'KING')
```

1 Result | 2 Profiler | 3 Messages | 4

(Read Only)

ENAME	JOB	HIREDATE
PUTIN	MANAGER	1981-04-02
MERKEL	MANAGER	1981-06-09

4. List the employee details whose salary is greater than the lowest salary of an employee belonging to deptno 20.

Task:

```
1   SELECT *FROM emp
2     WHERE (SAL > (SELECT MIN(SAL) FROM EMP WHERE DEPTNO = 20))AND DEPTNO=20
3     ORDER BY SAL
4     LIMIT 1;
```

1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info

(Read Only)

ID	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	SCOTT	CLERK	7788	1983-01-12	1100	*(NULL)*	20

5. Which department has the highest Monthly remuneration bill (Salaries of employees)?

Task:

```
Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List All Ta
1     SELECT deptno,SUM(sal)
2     FROM emp GROUP BY deptno
3   HAVING SUM(sal)>=ALL(SELECT SUM(sal)
4     FROM emp GROUP BY deptno);
```

1 Result | 2 Profiler | 3 Messages | 4 Tab

(Read Only)

deptno	sum(sal)
20	15875

6. Display the employees that earn a salary that is higher than the salary of all the clerks. Sort the result on salary from highest to lowest.

Task:

```
1     SELECT ename, job, sal
2   FROM emp WHERE sal > ALL (SELECT sal
3     FROM emp WHERE job = 'CLERK')
4     ORDER BY sal DESC
5
```

1 Result | 2 Profiler | 3 Messages | 4 Ta

(Read Only)

ename	job	sal
JACK	TEACHER	5000
BUSH	PRESIDENT	5000
TOOSK	ANALYST	3000
CARNEGIE	ANALYST	3000
PUTIN	MANAGER	2975
MERKEL	MANAGER	2450
BAROSSO	SALESMAN	1600
KING	SALESMAN	1500
CHIRACK	SALESMAN	1250
WALTON	SALESMAN	1250

7. Create a query to display the name, hire date and salary for all employees who have both the same salary and commission as employee SCOTT.

Task:

```
1     SELECT ename, hiredate, sal
2   FROM emp WHERE (sal, IFNULL(comm,0)) IN (SELECT sal, IFNULL(comm,0)
3     FROM emp WHERE ename = 'SCOTT')
4
```

1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info

(Read Only)

ename	hiredate	sal
KING	1981-09-08	1100
SCOTT	1983-01-12	1100

8. Display the names and salaries of those employees who earn highest salary in their department.

Task:

```
1   SELECT ename,  sal,  deptno
2   FROM emp WHERE sal IN
3     (SELECT MAX(sal) FROM emp GROUP BY deptno );
4
```

1 Result | 2 Profiler | 3 Messages | 4 Table Data

(Read Only)

ename	sal	deptno
WALTON	1250	30
CHIRACK	1250	30
BUSH	5000	10
JACK	5000	20

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 09

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To understand Indexes, Views & Sequences.

Tools: MySQL/Oracle.

Introduction:

INDEXES: A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns. The users cannot see the indexes, they are just used to speed up queries and will be used by the Database Search Engine to locate records very fast.

- The INSERT and UPDATE statements take more time on tables having indexes, whereas the SELECT statements become fast on those tables.

- The reason is that while doing insert or update, a database needs to insert or update the index values as well.

VIEWS: In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.
- Note that a view does not physically store the data.

SEQUENCE: A sequence is a list of integers generated in the ascending order**.** Many applications need sequences to generate unique numbers mainly for identification

Lab Task

1. Create a view called emp_vu based on the employee number, employee name, and department number from the EMP table. Change the heading for the employee name to EMPLOYEE.

Task

```
CREATE OR REPLACE VIEW employees_vu
    AS SELECT empno, ename , deptno
FROM emp;
```

Script Output | Task completed in 0.09 seconds

```
View EMPLOYEES_VU created.
```

2. Display the contents of the EMP_VU view.

Task

```
SELECT * FROM employees_vu;
```

Script Output | Query Result | SQL | All Rows Fetched: 14

	EMPNO	ENAME	DEPTNO
1	7839	KING	10
2	7698	BLAKE	30
3	7782	CLARK	10
4	7566	JONES	20
5	7722	SCOTT	20
6	7902	FORD	20
7	7369	SMITH	20
8	7499	ALLEN	30
9	7521	WARD	30
10	7654	MARTIN	30
11	7844	TURNER	30
12	7876	ADAMS	20
13	7900	JAMES	30
14	7934	MILLER	10

3. Select the view name and text from the data dictionary USER_views.

Task

```
SELECT view_name, text FROM user_views;
```

Script Output | Query Result | Task completed in 0.193 seconds

```
MVIEW_FILTERINSTANCE           select
                                      a.runid# as runid,
                                      a.filterid# as filterid,
                                      a.subfilter

MVIEW_LOG                      select
                                      m.runid# as id,
                                      m.filterid# as filterid,
                                      m.run_begin,

VIEW_NAME                      TEXT
------------------------------ --------------------------------
```

4. Create a view named DEPT_VU_20 that contains the employee number, employee name and department number for all employees in department 20. Label the view column Employee_Id, Employee, and Department_Id. Do not allow an employee to be reassigned to another department through the view.

Task

```
CREATE OR REPLACE VIEW dept20 AS
    SELECT  empno, ename,
deptno  FROM emp
WHERE deptno = 20
WITH CHECK OPTION CONSTRAINT emp_dept_20
```

Script Output | Query Result | Task completed in 0.045 seconds

```
View DEPT20 created.
```

5. Attempt to reassign Smith to department 30.

Task

```
UPDATE dept20 SET deptno = 30
WHERE ename = 'Smith';
```

Script Output | Query Result | Task completed in 0.056 seconds

```
0 rows updated.
```

6. Create a view Salary_vu based on the employee name, department name, salary and salary grade for all employees. Label the columns Employee, Department, Salary and Grade.

Task

```
CREATE OR REPLACE VIEW salary_vu AS SELECT e.ename "Employee",
d.dname "Department",e.sal "Salary", j.job_level "Grades"
FROM emp e, dept d, job_grades j
WHERE e.deptno = d.deptno
AND e.sal BETWEEN
j.lowest_sal and j.highest_sal;
```

Script Output | Query Result | Task completed in 0.234 seconds

```
View SALARY_VU created.
```

7. Create a sequence to be used with primary key column of the department table. The sequence should start at 60 and have a maximum value of 200. Have your sequence increment by ten numbers. Name the sequence Dept_Id_Seq.

Task

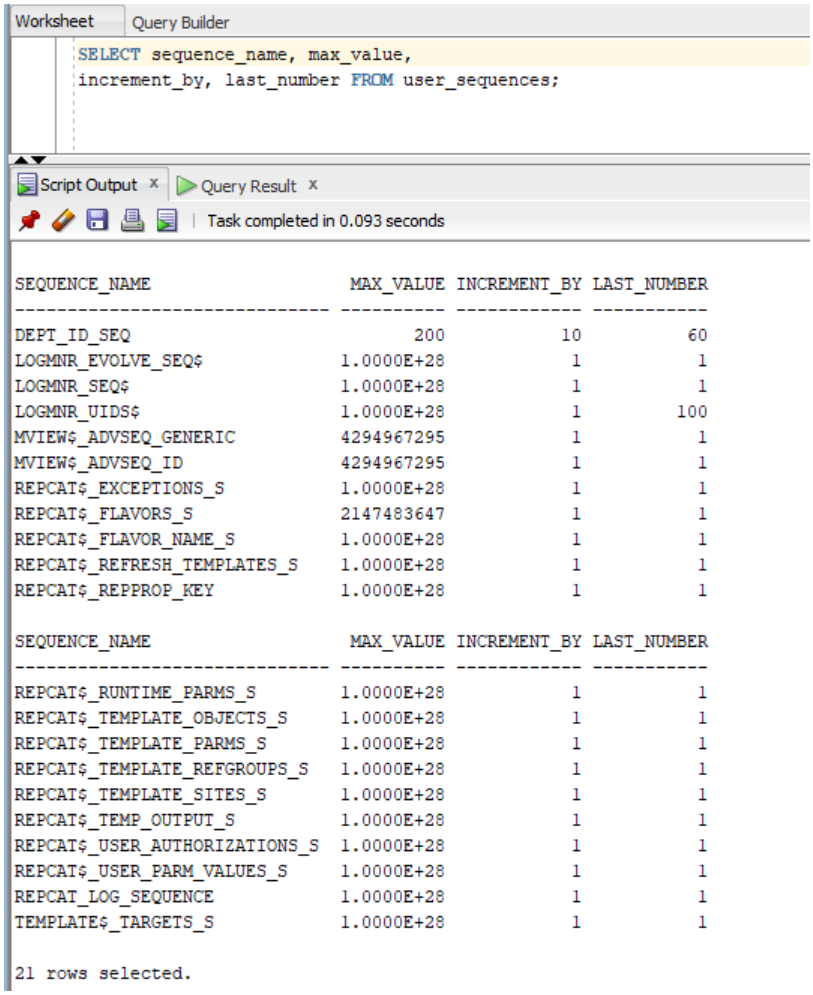
```
CREATE SEQUENCE dept_id_seq
  START WITH 60INCREMENT BY 10
MAXVALUE 200;
```

Script Output | Query Result | Task completed in 0.044 seconds

```
Sequence DEPT_ID_SEQ created.
```

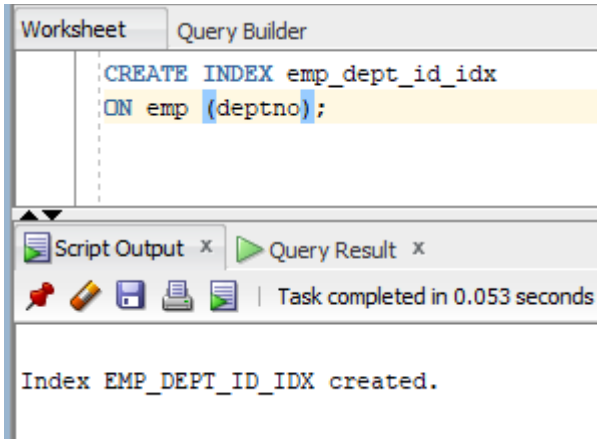
8. Display the following information about your sequences: sequence name, max value, increment size, and last number.

Task



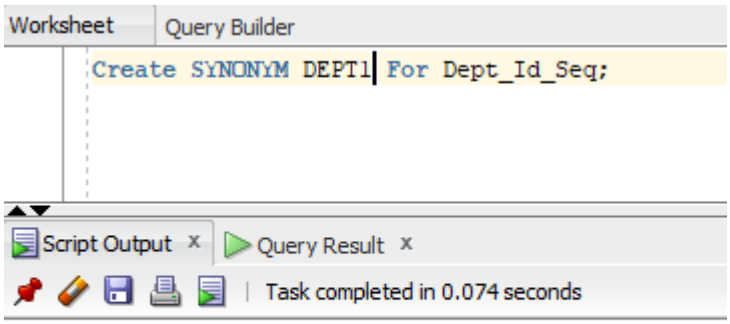
9. Create a non-unique index on the foreign key column in the employee table.

Task

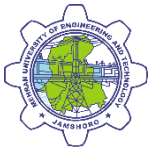


10. Create a synonym for Dept_Id_Seq.

Task




```
Synonym DEPT1 created.
```



DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 10

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: Creating database users and controlling user access.

Tools: MySQL/Oracle.

Introduction:

Create a New User

MySQL has a CREATE USER statement that lets you create a new user on your database. If you're working with a new database, you might only have a root user set up. To create users, you first need to log in with the root user or another user account that has permissions to create users.

```
mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
```

At this point *newuser* has no permissions to do anything with the databases. In fact, even if *newuser* tries to login (with the password, *password*), they will not be able to reach the MySQL shell.

Therefore, the first thing to do is to provide the user with access to the information they will need.

```
mysql> GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';
```

How to Grant Different User Permissions

There are numerous table and column level privileges that you can grant. You can also grant users the privileges to create other users or revoke other user privileges. The permissions you grant to a user should depend on what level of access you think they should have. The rule for security permissions is to give a user the least amount of privileges they need to accomplish normal tasks. For instance, you shouldn't grant all permissions to a standard user when they only need to run SELECT statements on a Customer table.

Here is a short list of other common possible permissions:

- ALL PRIVILEGES- as we saw previously, this would allow a MySQL user full access to a designated database (or if no database is selected, global access across the system)
- CREATE- allows them to create new tables or databases
- DROP- allows them to them to delete tables or databases
- DELETE- allows them to delete rows from tables
- INSERT- allows them to insert rows into tables
- SELECT- allows them to use the SELECT command to read through databases
- UPDATE- allow them to update table rows
- GRANT OPTION- allows them to grant or remove other users' privilege

This practical is an overview of managing user accounts and passwords. Keeping track of users and their permissions greatly reduces the chance that your database goes hacked without notice. By managing users, you keep granular control over your data and protect it from unauthorized access.

Lab Task

1. For a book library, create 3 users with names books_admin, books_accountant, books_user and assign passwords of your own choice.

Task:

```
--CREATE USER books_admin IDENTIFIED BY admin123 ;
--CREATE USER books_accountant IDENTIFIED BY accountant123 ;
CREATE USER books_user IDENTIFIED BY user123 ;
```

Script Output | Task completed in 0.057 seconds

```
User BOOKS_ADMIN created.


User BOOKS_ACCOUNTANT created.


User BOOKS_USER created.
```

2. Assign DBA, RESOURCE and CONNECT roles to books_admin, books_accountant, books_user, respectively.

Task:

```
--CREATE USER books_admin IDENTIFIED BY admin123 ;
--CREATE USER books_accountant IDENTIFIED BY accountant123 ;
--CREATE USER books_user IDENTIFIED BY user123 ;
GRANT CONNECT, RESOURCE, DBA TO books_admin,books_accountant,books_user;
```

Script Output | Task completed in 0.246 seconds

```
Grant succeeded.
```

3. Assign select, update, delete, and insert object privileges on books table to books_admin and books_accountant users.

Task:

Worksheet | Query Builder

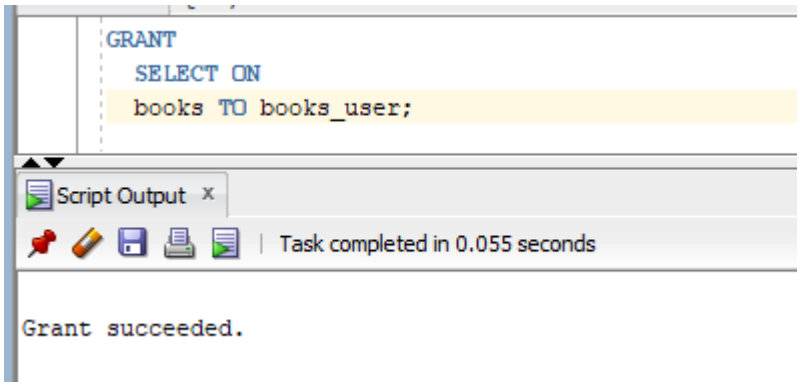
```
GRANT
  SELECT,INSERT,UPDATE,DELETE ON
  books TO books_admin;
```

Script Output | Task completed in 0.369 seconds

```
Grant succeeded.
```

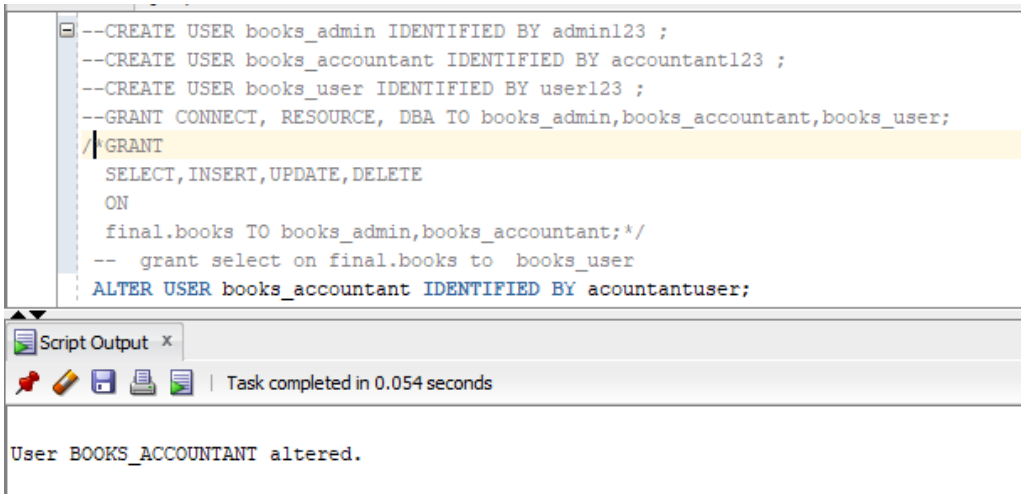
4. Assign select object privilege on books table to books_user user.

Task:



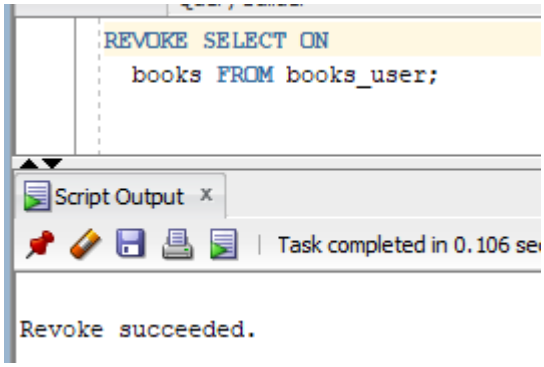
5. Change the password of user books_accountant.

Task:



6. Revoke the assigned object privilege from books_user.

Task:



7. What privilege should a user be given to log into the oracle server? Is this a system or object privilege?

Answer: The CREATE SESSION system privilege.

8. What privilege should a user be given to create tables and views?

Answer: The CREATE TABLE privilege.

9. If you create a table, who can pass along privileges to other users on your table?

Answer: You can, or anyone you have given those privileges to by using the WITH GRANT OPTION.

10. You are the DBA. You are creating many users who require the same system privileges. What would you use to make your job easier?

Answer: Create a role containing the system privileges and grant the role to the users.

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 11

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: **To understand and practice PL/SQL block structure, Control Structures and Data Types.**

Tools: MySql, Oracle.

Introduction:

PL/SQL Block Structure

In PL/SQL, as in most other procedural languages, the smallest meaningful grouping of code is known as a block. A block is a unit of code that provides execution and scoping boundaries for variable declarations and exception handling. PL/SQL allows you to create anonymous blocks (blocks of code that have no name) and named blocks, which may be packages, procedures, functions, triggers, or object types..

A PL/SQL block has up to four different sections, only one of which is mandatory:

Header

Used only for named blocks. The header determines the way the named block or program must be called. Optional.

Declaration section

Identifies variables, cursors, and subblocks that are referenced in the execution and exception sections. Optional.

Execution section

Statements the PL/SQL runtime engine will execute at runtime. Mandatory.

Exception section

Handles exceptions to normal processing (warnings and error conditions). Optional

Basic Syntax structure

Declare

Declaration of variable, constants

Begin

Execute statements in pl/sql

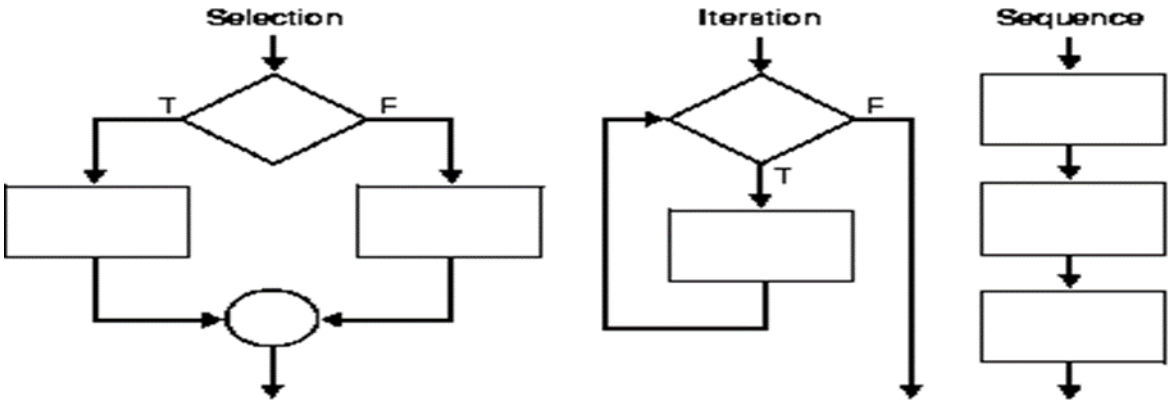
Exception

Exception handlers in pl/sql

END;

PL/SQL Control Structures

The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. A condition is any variable or expression that returns a Boolean value (TRUE or FALSE). The iteration structure executes a sequence of statements repeatedly as long as a condition holds true. The sequence structure simply executes a sequence of statements in the order in which they occur.

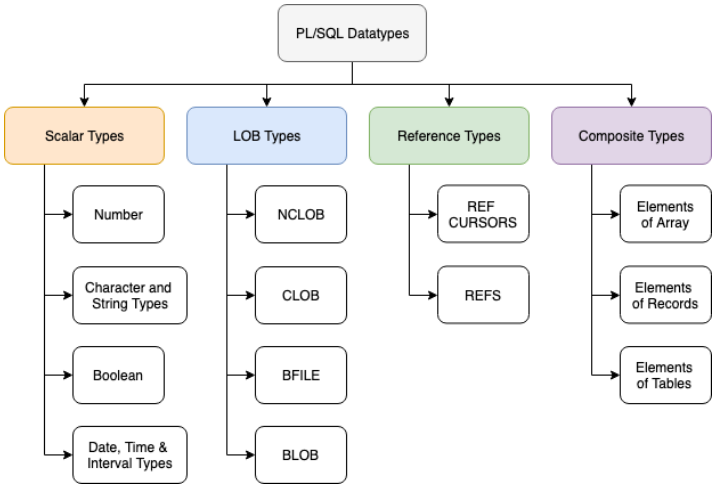


PL/SQL Data Types.

PL/SQL datatypes are not just limited to writing SQL queries but they are used in the PL/SQL block as well, just like any other programming language.

Provising a datatype specifies how any data will be stored and processed by Oracle when any PL/SQL code block is executed.

Datatype defines the type of data being used, whether it is a number or a word (string) or a single character etc. Following datatypes can be used in PL/SQL depending upon the type of data required:



So we have 4 broader categories of datatypes and they are:

1. Scalar Types: These are basic datatypes which generally holds a single value like a number or a string of characters. Scalar types have 4 different categories which are listed in the diagram above, namely Number Types, Character and String, Boolean Types and Date and Time etc.

2. **LOB Types**: This datatype deals with large objects and is used to specify location of these large objects like text files, images etc which are generally not stored outside the database.

3. **Reference Types:** This datatype is used to hold pointer values which generally stores address of other program items.

4. **Composite Types**: Last but not the least, as the name suggests this type of data is a composition of individual data which can be manipulated/processed separatel as well.

LAB TASK

1. **Write a PL/SQL block to calculate the annual salary of an employee whose ID is 7722.**

Task:

```
SET SERVEROUTPUT ON;
DECLARE
  incentive   NUMBER(8,2);
BEGIN
  SELECT SAL * 0.12 INTO incentive
  FROM EMP
  WHERE EMPNO = 7722;
DBMS_OUTPUT.PUT_LINE('Incentive  = ' || TO_CHAR(incentive));
END;
```

Script Output | Query Result | Task completed in 0.07 seconds

```
Incentive  = 360


PL/SQL procedure successfully completed.
```

2. **Write a PL/SQL block to show the operator precedence and parentheses in 5 or more complex expressions.**

Task:

```
DECLARE
  salary      NUMBER := 40000;
  commission  NUMBER := 0.15;
BEGIN
  -- Division has higher precedence than addition:

  DBMS_OUTPUT.PUT_LINE('8 + 20 / 4 = ' || (8 + 20 / 4));
  DBMS_OUTPUT.PUT_LINE('20 / 4 + 8 = ' || (20 / 4 + 8));

 -- Parentheses override default operator precedence:

  DBMS_OUTPUT.PUT_LINE('7 + 9 / 3 = ' || (7 + 9 / 3));
  DBMS_OUTPUT.PUT_LINE('(7 + 9) / 3 = ' || ((7 + 9) / 3));

  -- Most deeply nested operation is evaluated first:

  DBMS_OUTPUT.PUT_LINE('30 + (30 / 6 + (15 - 8)) = '
                       || (30 + (30 / 6 + (15 - 8))));

  -- Parentheses, even when unnecessary, improve readability:

  DBMS_OUTPUT.PUT_LINE('(salary * 0.08) + (commission * 0.12) = '
    || ((salary * 0.08) + (commission * 0.12)));

  DBMS_OUTPUT.PUT_LINE('salary * 0.08 + commission * 0.12 = '
    || (salary * 0.08 + commission * 0.12));
END;
```

Script Output | Query Result | Task completed in 0.051 seconds

```
8 + 20 / 4 = 13
20 / 4 + 8 = 13
7 + 9 / 3 = 10
(7 + 9) / 3 = 5.333333333333333333333333333333333333333
30 + (30 / 6 + (15 - 8)) = 42
(salary * 0.08) + (commission * 0.12) = 3200.018
salary * 0.08 + commission * 0.12 = 3200.018
```

3. **Write a PL/SQL program to arrange the number of two variable in such a way that the small number will store in num_small variable and large number will store in num_large variable.**

Task:

```
DECLARE
num_small NUMBER := 8;
num_large NUMBER := 5;
num_temp NUMBER;
BEGIN

IF num_small > num_large THEN
num_temp := num_small;
num_small := num_large;
num_large := num_temp;
END IF;

DBMS_OUTPUT.PUT_LINE ('num_small = '||num_small);
DBMS_OUTPUT.PUT_LINE ('num_large = '||num_large);
END;
```

Script Output | Query Result | Task completed in 0.045 seconds

```
num_small = 5
num_large = 8


PL/SQL procedure successfully completed.
```

4. **Write a PL/SQL program to count number of employees in department 30 and check whether this department have any vacancies or not. There are total 45 posts in this department.**

Task:

```
DECLARE
    CURSOR z_emp_info IS
      SELECT deptno,ename,sal
      FROM   emp where deptno=30;
    r_emp_info z_emp_info%ROWTYPE;
BEGIN
    OPEN z_emp_info;
    LOOP
        FETCH z_emp_info INTO r_emp_info;
        EXIT WHEN z_emp_info%NOTFOUND;
        dbms_output.Put_line('Name: '||r_emp_info.ename
                             ||'  SALARY: ' ||r_emp_info.sal||
                             '  DEPTNO: ' ||r_emp_info.deptno);
    END LOOP;
    dbms_output.Put_line('Total number of rows : '||z_emp_info%rowcount);
    CLOSE z_emp_info;
```

Script Output | Explain Plan | Task completed in 0.077 seconds

```
Name: BLAKE  SALARY: 3850  DEPTNO: 30
Name: ALLEN  SALARY: 2600  DEPTNO: 30
Name: WARD  SALARY: 2250  DEPTNO: 30
Name: MARTIN  SALARY: 2250  DEPTNO: 30
Name: TURNER  SALARY: 2500  DEPTNO: 30
Name: JAMES  SALARY: 1950  DEPTNO: 30
Total number of rows : 6


PL/SQL procedure successfully completed.
```

5. **Write a program in PL/SQL to check whether a number is prime or not using for loop.**

Task:

```
DECLARE
     n NUMBER := 17;
     i NUMBER:= 2 ;
     flag NUMBER:= 1;
BEGIN

 FOR i IN 2..n/2 LOOP
   IF  MOD(n,i)=0 THEN
          flag:=0; EXIT;
        END IF;
    END LOOP;

IF flag=1 THEN
        dbms_output.put_line(n||' is Prime Number');
    ELSE
        dbms_output.put_line(n||'is not Prime  Number');
    END IF;
END;
```

Script Output × Query Result ×

Task completed in 0.038 seconds

```
17 is Prime Number


PL/SQL procedure successfully completed.
```

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: **To explore Exception Handling in PL/SQL.**

Tools: MySql, Oracle.

Introduction:

An exception occurs when the PL/SQL engine encounters an instruction which it cannot execute due to an error that occurs at run-time. These errors will not be captured at the time of compilation and hence these needed to handle only at the run-time.

For example: if PL/SQL engine receives an instruction to divide any number by '0', then the PL/SQL engine will throw it as an exception. The exception is only raised at the run-time by the PL/SQL engine.

Exceptions will stop the program from executing further, so to avoid such condition, they need to be captured and handled separately. This process is called as Exception-Handling, in which the programmer handles the exception that can occur at the run time.

Exception-Handling Syntax:

Exceptions are handled at the block, level, i.e., once if any exception occurs in any block then the control will come out of execution part of that block. The exception will then be handled at the exception handling part of that block.

Syntax:

```
BEGIN
   <execution_block>
   .
   .
EXCEPTION
   WHEN <exception1_name>
   THEN
      < Exception handling code for the "exception1_name" >
   WHEN OTHERS
   THEN
      < Default exception handling code for all exceptions >
END;
```

Exception handler for "exception1_name"

Exception handler for other exception

Types of Exception:

1. Predefined Exceptions
2. User-defined Exception

Predefined Exceptions: Oracle has predefined some common exception. These exceptions have a unique exception name and error number. These exceptions are already defined in the 'STANDARD' package in Oracle. In code, we can directly use these predefined exception name to handle them.

User-defined Exception: In Oracle, other than the above-predefined exceptions, the programmer can create their own exception and handle them. They can be created at a subprogram level in the declaration part. These exceptions are visible only in that subprogram. The exception that is defined in the package specification is public exception, and it is visible wherever the package is accessible.

```
DECLARE
<exception_name> EXCEPTION;
BEGIN
<Execution block>
EXCEPTION
WHEN <exception_name> THEN
<Handler>
END;
```

- In the above syntax, the variable 'exception_name' is defined as 'EXCEPTION' type.
- This can be used as in a similar way as a predefined exception.

Lab Task

1. Write a PL/SQL block that updates description of a product and raises a user-defined exception when that product is not found.

Task:

```
Declare
    e_invaild_product EXCEPTION;
Begin
    UPDATE product set descript='&product_description'
    WHERE prodid=&product_number;

    IF SQL%NOTFOUND THEN
        RAISE e_invaild_product;
    END IF;
   COMMIT;
EXCEPTION
   WHEN e_invaild_product THEN
     DBMS_OUTPUT.PUT_LINE('INVAILD Product Number.');
END;
---18CS31---
```

Output:

```
Invalid product number.


PL/SQL procedure successfully completed.
```

2. Write a simple PL/SQL code block, to demonstrate the use of Named Exception Handler for division by zero error.

Task:

```
DECLARE
   stock_price NUMBER := 9.73;
   net_earnings NUMBER := 0;
   pe_ratio NUMBER;
BEGIN
   pe_ratio := stock_price / net_earnings;
   dbms_output.put_line('Price/earnings ratio = ' || pe_ratio);
EXCEPTION  -- exception handlers begin
   WHEN ZERO_DIVIDE THEN  -- handles 'division by zero' error
      dbms_output.put_line('Company must have had zero earnings.');
      pe_ratio := null;

   WHEN OTHERS THEN  -- handles all other errors
      dbms_output.put_line('Some other kind of error occurred.');
      pe_ratio := null;
END;  -- exception handlers and block end here
/
---18CS31-----
```

Task completed in 0.083 seconds

```
Company must have had zero earnings.


PL/SQL procedure successfully completed.
```

3. Write a PL/SQL block for displaying information of a customer ID, when the user enters an invalid ID, raise the exception invalid_id.

Task:

```
DECLARE
   c_id emp.emppno%type := &cc_id;
   c_name emp.ename%type;
   -- user defined exception
   ex_invalid_id  EXCEPTION;
BEGIN
   IF c_id <= 0 THEN
      RAISE ex_invalid_id;
   ELSE
      SELECT  ename INTO  c_name
      FROM emp
      WHERE empno = c_id;
      DBMS_OUTPUT.PUT_LINE ('EMPLOYEE name is: '||  c_name);
   END IF;
EXCEPTION
   WHEN ex_invalid_id THEN
      dbms_output.put_line('ID must be greater than zero!');
   WHEN no_data_found THEN
      dbms_output.put_line('No such customer!');
END;
/
------18CS31-----
```

Output:

```
EMPLOYEE name is KING


PL/SQL procedure successfully completed.
```

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 13

Roll No: **Date of Conduct:**

Submission Date: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To create and use Cursors in PL/SQL

Tools: MYSQL Oracle.

Introduction:

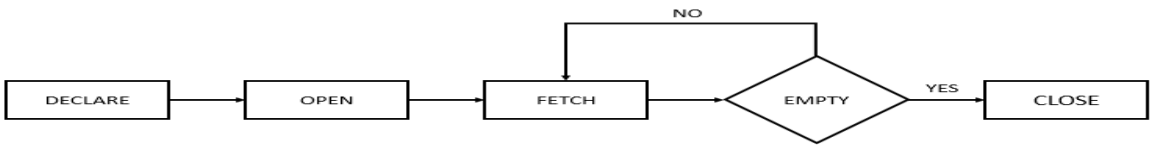
Cursors : A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors.

- Implicit cursors
- Explicit cursors

Implicit cursors: Whenever Oracle executes an SQL statement such as SELECT INTO, INSERT, UPDATE, and DELETE, it automatically creates an implicit cursor.
Oracle internally manages the whole execution cycle of implicit cursors and reveals only the cursor's information and statuses such as SQL%ROWCOUNT, SQL%ISOPEN, SQL%FOUND, and SQL%NOTFOUND.
The implicit cursor is not elegant when the query returns zero or multiple rows which cause NO_DATA_FOUND or TOO_MANY_ROWS exception respectively.

Explicit cursors: An explicit cursor is an SELECT statement declared explicitly in the declaration section of the current block or a package specification.
For an explicit cursor, you have control over its execution cycle from OPEN, FETCH, and CLOSE.
Oracle defines an execution cycle that executes an SQL statement and associates a cursor with it. The following illustration shows the execution cycle of an explicit cursor



Explicit Cursor Attributes:

A cursor has four attributes to which you can reference in the following format: `cursor_name%attribute` (where cursor_name is the name of the explicit cursor)

1. **%ISOPEN:**

   This attribute is TRUE if the cursor is open or FALSE if it is not.

2. **%FOUND:**

   This attribute has four values:

   - NULL before the first fetch
   - TRUE if a record was fetched successfully
   - FALSE if no row returned
   - INVALID_CURSOR if the cursor is not opened

3. **%NOTFOUND:**

   This attribute has four values:

   - NULL before the first fetch
   - FALSE if a record was fetched successfully
   - TRUE if no row returned
   - INVALID_CURSOR if the cursor is not opened

4. **%ROWCOUNT:**

   The %ROWCOUNT attribute returns the number of rows fetched from the cursor. If the cursor is not opened, this attribute returns INVALID_CURSOR.

Lab Task

1. **Write a program in PL/SQL to find the number of rows effected using SQL%ROWCOUNT attributes of an implicit cursor.**

Task:

```
DECLARE
  mgr_no NUMBER(6) := 7839;
BEGIN
  DELETE FROM employee WHERE mgr = mgr_no;
  DBMS_OUTPUT.PUT_LINE
    ('Number of employees deleted: ' || TO_CHAR(SQL%ROWCOUNT));
END;
/
SET SERVEROUTPUT ON;
-----18CS31------
```

Script Output | Task completed in 0.108 seconds

```
Number of employees deleted: 3


PL/SQL procedure successfully completed.
```

2. **Write a program in PL/SQL to display detail information for the employee of ID 7839 from the employees' table.**

Task:

Worksheet | Query Builder

```
DECLARE
    z_employee emp%ROWTYPE;
BEGIN
    SELECT * INTO   z_employee -- INTO clause always notifies only single row can be fetch
    FROM   employee WHERE  empno = 7839;

    dbms_output.Put_line('Employee Details :   ID:' ||z_employee.empno
                         ||'  Name: ' ||z_employee.ename
                         ||'  Salary:  ' ||z_employee.sal
                         ||'  Hire date:  ' ||z_employee.hiredate
                         ||'  DEPTNO:  ' ||z_employee.deptno);
END;
/
SET SERVEROUTPUT ON;
-----18CS31------
```

Script Output | Task completed in 0.203 seconds

```
Employee Details :   ID:7839  Name: KING  Salary:  5000  Hire date:  17-NOV-81  DEPTNO:  10


PL/SQL procedure successfully completed.
```

3. Write a program in PL/SQL to display detail information of all employees from employees table using explicit cursor.

Task:

```
DECLARE
    CURSOR z_emp_info IS
      SELECT empno,ename,JOB_ID,COMM
            HIREDATE,SAL,DEPTNO
      FROM   employee;
    r_emp_info z_emp_info%ROWTYPE;
BEGIN
    OPEN z_emp_info;

    LOOP
        FETCH z_emp_info INTO r_emp_info;
        EXIT WHEN z_emp_info%NOTFOUND;
        dbms_output.Put_line('Employee Details :   ID:' ||r_emp_info.empno
                        ||'  Name: ' ||r_emp_info.ename||'  JOB ID:  ' ||r_emp_info.JOB_ID
                        ||'  Hire date:  ' ||r_emp_info.hiredate
                        ||'  Salary:  ' ||r_emp_info.sal||'  DEPTNO:  ' ||r_emp_info.deptno);
    END LOOP;
    dbms_output.Put_line('Total number of rows : '||z_emp_info%rowcount);
    CLOSE z_emp_info;
END;
/
SET SERVEROUTPUT ON;
-----18CS31------
```

Script Output

Task completed in 0.096 seconds

```
Employee Details :   ID:7839  Name: KING  JOB ID:  PRESIDENT  Hire date:    Salary:  5000  DEPTNO:  10
Employee Details :   ID:7698  Name: BLAKE  JOB ID:  MANAGER  Hire date:    Salary:  2850  DEPTNO:  30
Employee Details :   ID:7782  Name: CLARK  JOB ID:  MANAGER  Hire date:    Salary:  2450  DEPTNO:  10
Employee Details :   ID:7566  Name: JONES  JOB ID:  MANAGER  Hire date:    Salary:  2975  DEPTNO:  20
Employee Details :   ID:7722  Name: SCOTT  JOB ID:  ANALYST  Hire date:    Salary:  3000  DEPTNO:  20
Employee Details :   ID:7902  Name: FORD  JOB ID:  ANALYST  Hire date:    Salary:  3000  DEPTNO:  20
Employee Details :   ID:7369  Name: SMITH  JOB ID:  CLERK  Hire date:    Salary:  800  DEPTNO:  20
Employee Details :   ID:7499  Name: ALLEN  JOB ID:  SALESMAN  Hire date:  300  Salary:  1600  DEPTNO:  30
Employee Details :   ID:7521  Name: WARD  JOB ID:  SALESMAN  Hire date:  500  Salary:  1250  DEPTNO:  30
Employee Details :   ID:7654  Name: MARTIN  JOB ID:  SALESMAN  Hire date:  1400  Salary:  1250  DEPTNO:  30
Employee Details :   ID:7844  Name: TURNER  JOB ID:  SALESMAN  Hire date:  0  Salary:  1500  DEPTNO:  30
Employee Details :   ID:7876  Name: ADAMS  JOB ID:  CLERK  Hire date:    Salary:  1100  DEPTNO:  20
Employee Details :   ID:7900  Name: JAMES  JOB ID:  CLERK  Hire date:    Salary:  950  DEPTNO:  30
Employee Details :   ID:7934  Name: MILLER  JOB ID:  CLERK  Hire date:    Salary:  1300  DEPTNO:  10
Total number of rows : 14
```

4. **Write a PL/SQL block that uses explicit cursors to retrieve employees one by one and displays the name and salary of those employees currently working in deptno 30.**

Task:

```
DECLARE
    CURSOR z_emp_info IS
      SELECT empno,ename,SAL,DEPTNO
      FROM   employee where deptno=30;
    r_emp_info z_emp_info%ROWTYPE;
BEGIN
    OPEN z_emp_info;

    LOOP
        FETCH z_emp_info INTO r_emp_info;
        EXIT WHEN z_emp_info%NOTFOUND;
        dbms_output.Put_line('Employee Details :   ID:' ||r_emp_info.empno
                        ||'  Name: ' ||r_emp_info.ename||'  Salary:  ' ||r_emp_info.sal
                        ||'  DEPTNO:  ' ||r_emp_info.deptno);
    END LOOP;
    dbms_output.Put_line('Total number of rows : '||z_emp_info%rowcount);
    CLOSE z_emp_info;
END;
/
SET SERVEROUTPUT ON;
-----18CS31------
```

Script Output

Task completed in 0.124 seconds

```
Employee Details :   ID:7698  Name: BLAKE  Salary:  2850  DEPTNO:  30
Employee Details :   ID:7499  Name: ALLEN  Salary:  1600  DEPTNO:  30
Employee Details :   ID:7521  Name: WARD  Salary:  1250  DEPTNO:  30
Employee Details :   ID:7654  Name: MARTIN  Salary:  1250  DEPTNO:  30
Employee Details :   ID:7844  Name: TURNER  Salary:  1500  DEPTNO:  30
Employee Details :   ID:7900  Name: JAMES  Salary:  950  DEPTNO:  30
Total number of rows : 6
```

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 14

Roll No: **Date of Conduct:**
SubmissionDate: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: Working with stored Functions and Procedures in PL/SQL.

Tools: MySql/oracle.

Introduction:

Stored Functions and Procedures in PL/SQL

Stored procedures and functions (subprograms) can be compiled and stored in an Oracle Database, ready to be executed. Once compiled, it is a schema object known as a stored procedure or stored function, which can be referenced or called any number of times by multiple applications connected to Oracle Database. Both stored procedures and functions can accept parameters when they are executed (called). To execute a stored procedure or function, you only need to include its object name.

Creating a Stored Procedure or Function

Whenever a block of code for stored procedure or function is written it is then,they are automatically compiled by the oracle engine. During compilation if any error occurs, we get a message on the screen saying that the procedure or function is created with compilation errors but actual error is not displayed.

In order to find out the compilation errors following statement can be executed:

```
SELECT * from USER_ERRORS;
```

Once it is compiled, it is then stored by the oracle engine in the database as a database object. Stored Procedure or function's block of code in PL/SQL is made up of the following three sections:

- **Declarative section:** In this section, variables, constants, cursor or exceptions that are going to be used by procedure or function are declared.

- **Executable section:** In this section, the definition of procedure or function created is written. This section also contains the SQL or PL/SQL statements assigning values, controlling execution and manipulating data.
- **Exception Handling section:** In this section, the expected exceptions are written which may arise during execution of code written in executable part. This section is optional.

Syntax for creating Stored Procedure
Below we have the basic syntax for creating a stored procedure in oracle:

```
CREATE OR REPLACE PROCEDURE <procedure_name>

(<variable_name>IN/OUT/IN OUT <datatype>,

  <variable_name>IN/OUT/IN OUT <datatype>,...) IS/AS

variable/constant declaration;

BEGIN

        -- PL/SQL subprogram body;

EXCEPTION

        -- Exception Handling block ;

END <procedure_name>;
```

Let's understand the above code,

- procedure_name is for procedure's name and variable_name is the variable name for variable used in the stored procedure.
- CREATE or REPLACE PROCEDURE is a keyword used for specifying the name of the procedure to be created.
- BEGIN, EXCEPTION and END are keywords used to indicate different sections of the procedure being created.
- IN/OUT/IN OUT are parameter modes.
- IN mode refers to READ ONLY mode which is used for a variable by which it will accept the value from the user. It is the default parameter mode.
- OUT mode refers to WRITE ONLY mode which is used for a variable that will return the value to the user.
- IN OUT mode refers to READ AND WRITE mode which is used for a variable that will either accept a value from the user or it will return the value to the user.
- At the end, <procedure_name> is optional to write, you can simply use END statement to end the procedure definition.

Syntax for creating Functions in PL/SQL

Now that we know how to create stored procedures and how to use them in another PL/SQL code block, it's time to understand how to create Functions in PL/SQL.

```
CREATE OR REPLACE FUNCTION <function_name>

(<variable_name> IN <datatype>,

<variable_name> IN <datatype>,...)

RETURN <datatype> IS/AS

variable/constant declaration;

BEGIN

        -- PL/SQL subprogram body;

EXCEPTION

        -- Exception Handling block ;

END <function_name>;
```

Let's understand the above code:

- Function_name is for defining function's name and variable_name is the variable name for variable used in the function.
- CREATE or REPLACE FUNCTION is a keyword used for specifying the name of the function to be created.
- IN mode refers to READ ONLY mode which is used for a variable by which it will accept the value from the user. It is the default parameter mode.
- RETURN is a keyword followed by a datatype specifying the datatype of a value that the function will return.

LAB TASK

1. Create a stored function to calculate the salary ranking of the employee based on the current minimum and maximum salaries for employees in the same job category. (Hint: salary ranking = (empsal-minsal)/(Maxsal-Minsal)).

Task:

```
CREATE OR REPLACE FUNCTION emp_sal_ranking (empid NUMBER) RETURN NUMBER IS
  minsal        emp.sal%TYPE; -- declare a variable same as salary
  maxsal        emp.sal%TYPE; -- declare a variable same as salary
  jobid         emp.job_id%TYPE; -- declare a variable same as job_id
  v_sal           emp.sal%TYPE; -- declare a variable same as salary
BEGIN
-- retrieve the jobid and salary for the specific employee ID
  SELECT job_id, sal INTO jobid, v_sal FROM emp WHERE empno = empid;
-- retrieve the minimum and maximum salaries for employees with the same job ID
  SELECT MIN(sal), MAX(sal) INTO minsal, maxsal FROM emp WHERE job_id = jobid;
-- return the ranking as a decimal, based on the following calculation
  RETURN ((v_sal - minsal)/(maxsal - minsal));
END emp_sal_ranking;
/
DECLARE
  empid NUMBER := 7499; -- pick an employee ID to test the function
BEGIN
  DBMS_OUTPUT.PUT_LINE('The salary ranking for employee ' || empid || ' is: '
                       || ROUND(emp_sal_ranking(empid),2));
END;
/
SET SERVEROUTPUT ON;
-------18CS31-----------
```

Script Output

Task completed in 0.141 seconds

```
Function EMP_SAL_RANKING compiled

The salary ranking for employee 7499 is: 1
```

2. Write a stored procedure that accepts radius of circle and displays area of the circle.

Task:

```
DECLARE
  PI NUMBER(3,2);
  RADIUS NUMBER(7);
  AREA NUMBER(13,2);

PROCEDURE FINDAREA(X IN NUMBER,Y IN NUMBER,Z OUT NUMBER )
 IS BEGIN
    Z:=X*POWER(Y,2);
END;

BEGIN
    PI :=3.14;
    RADIUS :=&N;
    FINDAREA(PI,RADIUS,AREA);

    DBMS_OUTPUT.PUT_LINE('AREA: '||AREA);
END;
/
SET SERVEROUTPUT ON;
-------18CS31--------
```

Script Output

Task completed in 1.718 seconds

```
BEGIN
    PI :=3.14;
    RADIUS :=9;
    FINDAREA(PI,RADIUS,AREA);

    DBMS_OUTPUT.PUT_LINE('AREA: '||AREA);
END;
AREA: 254.34
```

3. Write a stored function that accepts a number and returns its factorial.

Task:

```
DECLARE
   num1 number;
   factorial number;

FUNCTION factor(x number)
RETURN number IS
   f number;

BEGIN
   IF x=0 THEN
      f := 1;
   ELSE
      f := x * factor(x-1);
   END IF;
RETURN f;
END;

BEGIN
   num1 := 6;
   factorial := factor(num1);
   dbms_output.put_line(' Factorial '|| num1 || ' is ' || factorial);
END;
/
SET SERVEROUTPUT ON;
-------18CS31------------
```

Script Output

Task completed in 0.087 seconds

```
Factorial 6 is 720


PL/SQL procedure successfully completed.
```

DEPARTMENT OF COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING & TECHNOLOGY, JAMSHORO
Database Management Systems (4th Semester) 18CS
Lab Experiment 15

Roll No: **Date of Conduct:**

SubmissionDate: **Grade Obtained:**

Problem Recognition (0.3)	Completeness & accuracy (0.4)	Timeliness (0.3)	Score (1.0)

Objective: To understand the use of Triggers in PL/SQL.

Tools: MySql/oracle.

Introduction:

Triggers: Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP,

  Or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes −

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

The syntax for creating a trigger is −

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − this provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

LAB TASK

1. Consider Student table, in which student's marks assessment is recorded. In such schema, create a trigger so that the total and percentage of specified marks is automatically inserted whenever a record is inserted.

Task:

```
CREATE OR REPLACE TRIGGER stud_marks
BEFORE INSERT ON Student FOR EACH ROW WHEN (NEW.TID > 0 )
DECLARE
     v_subj1 student.subj1%TYPE;
     v_subj2 student.subj2%TYPE;
     v_subj3 student.subj3%TYPE;
     v_total number;
     v_percent  number;
BEGIN
   Select subj1,subj2,subj3 into v_subj1,v_subj2,v_subj3 from student where TID=1;--where into.tid ;
   v_total :=  v_subj3+ v_subj2+ v_subj3;
   v_percent := v_total * 60 / 100;
   dbms_output.put_line('TOTAL MARKS: ' || v_total);
   dbms_output.put_line('Percentage: ' || v_percent);
END;
/
insert into Student values(6, 'ABCDE', 20, 20, 20, 0, 0);
```

Script Output

Task completed in 0.091 seconds

```
Trigger STUD_MARKS compiled

TOTAL MARKS: 60
Percentage: 36


1 row inserted.
```

2. Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger should display the salary difference between the old values and new values.

Task:

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON EMP
FOR EACH ROW
WHEN (NEW.EMPNO > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.sal  - :OLD.sal;
   dbms_output.put_line('Old salary: ' || :OLD.sal);
   dbms_output.put_line('New salary: ' || :NEW.sal);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES (7890,'Tahir',75000.00);
UPDATE EMP SET sal =sal+1000 where EMPNO=7890;
----18CS31--------------
```

Script Output

Task completed in 0.198 seconds

```
1 row inserted.

Old salary: 75000
New salary: 76000
Salary difference: 1000
  Difference 1000


1 row updated.
```