18CSC303J - Database Management Systems

Lab Experiments

S.No	Date	Name of the Topic	Page No.	Signature (Faculty)

Experiment -1

Aim: SQL Data Definition Language commands

Procedure:

- Table - they are the database object that hold the data in the relational database.

  A database can be equipped with one or more table can be model as relational. The "CREATE TABLE" statement is used to create new table in a database

- DDL: Data definition language is a subset of SQL and a part of DBMS. DDL consists of commands like, CREATE, ALTER, TRUNCATE, and DROP. These commands are used to create or modify the tables in SQL

DDL commands:

- Create
- Alter
- truncate
- drop

- Create:- used to create a new table in SQL, the user has to provide table name, column names, and their data types.

Syntax: CREATE Table table_name(column1 datatype, column2 datatype ...);

- Alter :- this command is used to add, delete or change columns in table.

Syntax: Alter table table_name add/delete column_name.

- Truncate :- The command is used to remove all rows from the table but the structure

Syntax : Truncate table table_name;

- Drop :- used to remove an existing table along with its structure

Syntax : Drop table_name

Output -

```
SQL> alter table department rename column total to sum;

Table altered.

SQL> desc edd256;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SNO                                                NUMBER(38)
 NAME                                               VARCHAR2(10)
 M1                                                 NUMBER(38)
 M2                                                 NUMBER(38)
 SUM                                                NUMBER(38)

SQL> alter table edd256 add(percentage int);

Table altered.

SQL> desc edd256
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SNO                                                NUMBER(38)
 NAME                                               VARCHAR2(10)
 M1                                                 NUMBER(38)
 M2                                                 NUMBER(38)
 SUM                                                NUMBER(38)
 PERCENTAGE                                         NUMBER(38)
```

```
SQL> desc edd256;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SNO                                                NUMBER(38)
 NAME                                               VARCHAR2(10)
 M1                                                 NUMBER(38)
 M2                                                 NUMBER(38)

SQL> alter table edd256 add(total int);

Table altered.

SQL> desc edd256;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SNO                                                NUMBER(38)
 NAME                                               VARCHAR2(10)
 M1                                                 NUMBER(38)
 M2                                                 NUMBER(38)
 TOTAL                                              NUMBER(38)
```

Result : SQL DDL commands were successfully studied and implemented.

Experiment - 2

Aim :- SQL Data Manipulation Language commands

Procedure :

1) Open the sqlplus.exe file to connect with the SQL instance
2) then to connect, put in the username
3) After connection, write SPOOL ON & SPOOL lab2.lst. to ~~cat~~ store all the commands being run
4) Create a new employee table using CREATE DDL commands.
5) then insert values into the table using the following syntax

   Insert into EMP values ('empno', 'name', 'position', salary);

6) Then, to see all the values, type

   select * from EMP.

7) then we use the following commands to ~~retrieve~~ update anything in the table.

   UPDATE
   ~~update~~ EMP SET EMPNO = 10008 WHERE SAL = 50000;

8) Then we use the following commands to retrieve anything from the table using the where clause

```
SELECT ENAME FROM EMP WHERE SAL = 20000.
```

9) Then we use the order by clause to retrieve into in ascending order

```
SELECT ENAME FROM EMP ORDER BY DOJ;
```

10) We then write SPOOL OFF.

□

Output -

```
SQL> insert into ed256 values(1,'Eeshan',99,100);

1 row created.

SQL> insert into ed256 values(2, 'Aakash',41,69);

1 row created.

SQL> insert into ed256 values(3, 'Ayush',55,66);

1 row created.

SQL> select * from ed256;

       SNO NAME        M1  M2
---------- ---------- ---------- ----------
         1 Eeshan             99 100
         2 Aakash             41  69
         3 Ayush      55  66
```

```
SQL> update ed256 set sno=2 where m1=41;

1 row updated.

SQL> select * from ed256;

       SNO NAME        M1  M2
---------- ---------- ---------- ----------
         1 Eeshan             99 100
         2 Aakash             41  69
         3 Ayush      55  66
```

```
SQL> delete from ed256 where m1 = 55;

1 row deleted.

SQL> select * from ed256;

       SNO NAME        M1  M2
---------- ---------- ---------- ----------
         1 Eeshan               99 100
         2 Aakash               41  69
         4 Harsh        89  67
```

Result :- The study of DML commands was successfully completed.

Experiment - 3

Aim: To write SQL queries to execute different DCL and TCL commands.

Procedure:

1) Then open the sqlplus.exe file to connect
2) Then to connect, put in the username.
3) After connection, write SPOOL ON and SPOOL lab 3.lst to store all the commands that will be done
4) To create a table, we use CREATE TABLE command
5) Then to insert values in to the table we use the following command

   - insert into <tablename> values (<values>);

6) Use save point sp1 ; to create a savepoint within a transaction so that we can use the ROLLBACK command to roll back to it.
7) use Grant ALL ON students TO PUBLIC WITH GRANT OPTION to give user access.
8) Use REVOKE ALL ON STUDENTS FROM PUBLIC; to revoke the user's privileges. and write SPOOL OFF command.

Output -

```
SQL> select * from eddu256;

       SNO NAME        M1  M2
---------- ---------- ---------- ----------
         1 Eeshan              99 100
         2 Aakash              67  89
         3 Harsh       87  85
         4 Krish       67  78
         5 Shivansh            80  81

SQL> SAVEPOINT SP1;

Savepoint created.

SQL> DELETE FROM eddu256 where m1=87;

1 row deleted.

SQL> SAVEPOINT SP2;

Savepoint created.

SQL> select * from eddu256;

       SNO NAME        M1  M2
---------- ---------- ---------- ----------
         1 Eeshan              99 100
         2 Aakash              67  89
         4 Krish       67  78
         5 Shivansh            80  81

SQL> Rollback to SP1;

Rollback complete.

SQL>  select * from eddu256;

       SNO NAME        M1  M2
---------- ---------- ---------- ----------
         1 Eeshan              99 100
         2 Aakash              67  89
         3 Harsh       87  85
         4 Krish       67  78
         5 Shivansh            80  81
```

```
SQL> select * from eddu256;

       SNO NAME        M1  M2
---------- ---------- ---------- ----------
         1 Eeshan              99 100
         2 Aakash              67  89
         3 Harsh       87  85
         4 Krish       67  78

SQL> GRANT SELECT ON eddu256 TO public;

Grant succeeded.

SQL> REVOKE SELECT ON eddu256 FROM public;

Revoke succeeded.

SQL> insert into eddu256 values (5,'Shivansh',80,81);

1 row created.

SQL> Commit;

Commit complete.
```

Result: All DCL and TCL commands were executed successfully.

Experiment - 4

Aim: To study inbuilt functions in SQL.

Procedure:

working on the following inbuilt functions -

INITCAP - Capitalize the initial of a string

SELECT INITCAP(' soham ') FROM dual

LENGTH (< str >) -

Converts all characters in the specified string to uppercase

SELECT upper (' soham ') FROM dual;

LTRIM (< str1 > [, < str2 >]) -

- removes all space characters from the left-hand side of a String

- SELECT ltrim(' soham ') FROM dual;

RTRIM (< str > [, < str2 >]) -

Removes all spaces characters from the right-hand side of a side.

ABS() - returns the absolute value of no.

ACOS() - Returns the arc cosine of a no.

CEIL() - Returns the smallest int value that is greater than a no.

FLOOR() - Returns the largest int. value that is equal to or less than a no.

MOD() - Returns the remainder.

POWER() - Returns m raised to the nth power

Output -

```
SQL> select ascii('A'),ascii('a') from dual;

ASCII('A') ASCII('A')
---------- ----------
        65         97

SQL> select chr(65),chr(97) from dual;

C C
- -
A a

SQL> select concat('eeshan',' dutta') from dual;

CONCAT('EESH
------------
eeshan dutta

SQL> select initcap('eeshan dutta') from dual;

INITCAP('EES
------------
Eeshan Dutta
```

```
SQL> Select INSTR('Burger King','K') SOLN from dual;

      SOLN
----------
         8

SQL> select length('eeshan dutta') as len from dual;

       LEN
----------
        12

SQL> select lower('EESHAN') from dual;

LOWER(
------
eeshan

SQL> select upper('eeshan') from dual;

UPPER(
------
EESHAN
```

```
SQL> select abs(-20) from dual;

  ABS(-20)
----------
        20

SQL> select acos(.50) from dual;

 ACOS(.50)
----------
1.04719755

SQL> select asin(0.50) from dual;

ASIN(0.50)
----------
.523598776

SQL> select atan(0.3) from dual;

 ATAN(0.3)
----------
.291456794

SQL> select ceil(3.7) from dual;

 CEIL(3.7)
----------
         4
```

```
SQL> SELECT SYSDATE, ADD_MONTHS(SYSDATE,2), ADD_MONTHS(SYSDATE,2) FROM DUAL;

SYSDATE   ADD_MONTH ADD_MONTH
--------- --------- ---------
08-FEB-22 08-APR-22 08-APR-22

SQL> SELECT SYSDATE, LAST_DAY(SYSDATE) END_OF_MONTH FROM DUAL;

SYSDATE   END_OF_MO
--------- ---------
08-FEB-22 28-FEB-22

SQL> select next_day('31-Aug-18','SUN') from dual;

NEXT_DAY(
---------
02-SEP-18

SQL> select sysdate,round(sysdate,'MM') from dual;

SYSDATE   ROUND(SYS
--------- ---------
08-FEB-22 01-FEB-22

SQL> SELECT MONTHS_BETWEEN
   (TO_DATE('02-02-1995','MM-DD-YYYY'),
    TO_DATE('01-01-1995','MM-DD-YYYY') ) "Months"
    FROM DUAL;  2    3    4

    Months
----------
1.03225806
```

Result: All inbuilt functions studied & executed successfully

Experiment - 5

Aim: To construct an ER diagram using drawio.

Procedure:

ER diagram:- It is basically called Entity relationship diagram which describes a proper structure of a wholedatabase with a help of diagram which is called ER diagram.

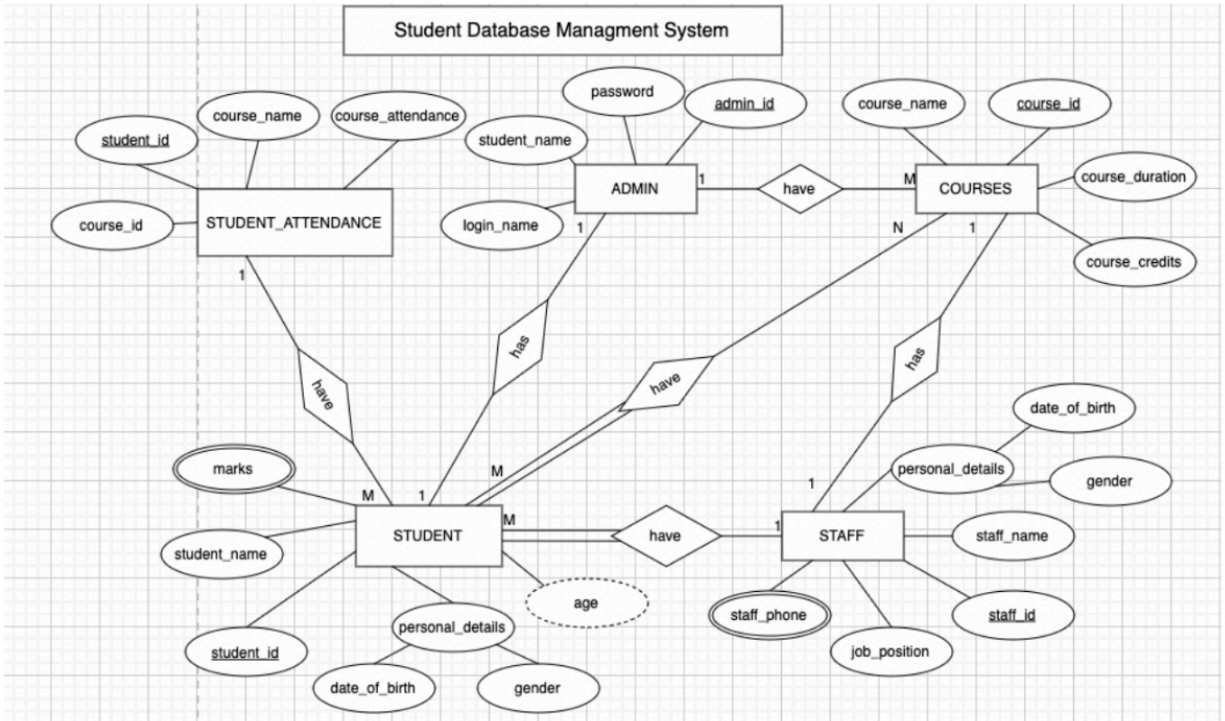
It is a design or blueprint of a database that can later be implemented as a database. components are: entity set and relationship set.

Geometric shapes used in ER diagram:

→ Rectangle: Represents entity sets
→ Ellipses: Attributes
→ Diamonds: Relationship set
→ Double Ellipses: Multivalued attributes
→ Dashed Ellipses: Derived Attributes
→ Double rectangles: weak entity sets
→ Double lines: Total participation of an entity.

Components of ER diagram:

1. Entity
   - weak
2. Attribute
   - key
   - composite
   - Multivalued
   - Derived
3. Relationship
   - one to one
   - one to many
   - Many to one
   - Many to many.



Relationship - Binary
Attributes - course_name, course_attendance, student_name, login_name, password, course_credits, course_credits, date_of_birth, personal_details, gender, staff_name, job_position

Derived Attributes - age
Composite Attributes - personal_details
Multivalued Attributes - staff_phone, marks
Key Attributes - student_id, staff_id, course_id, admin_id
Cardinality Mapping - one to one, one to many, many to one, many to many

Total Participation - student -> staff relationship
Partial Participation - staff -> student relationship

Result: ER diagram using draw.io was constructed.

Experiment - 6

Aim: To convert ER diagram to Relational Tables.

Procedures:

After designing the ER diagram of system, we need to convert it to Relational Model which can directly be implemented by any RDBMS like oracle, MySQL etc.

Relational Model represents how data is stored in Relational Databases.

step-wise procedure:

Create tables for all higher-level entities

Create tables for lower-level entities.

Add primary keys of higher-level entities in the table of lower-level.

Declare foreign key constraints and other constraints.

Output -

```
SQL> create view DetailsView as select name,address from eedu_stu where st_id<4;

View created.

SQL> select * from DetailsView;

NAME                 ADDRESS
-------------------- --------------------
Eeshan               West Bengal
Aakash               Delhi
Krish                Bangalore
```

```
SQL> create view SampleView as select name,address from eedu_stu where st_id>2;

View created.

SQL> select * from SampleView;

NAME                 ADDRESS
-------------------- --------------------
Krish                Bangalore
Shivansh             Nepal

SQL> Drop View SampleView;

View dropped.

SQL> Drop View DetailsView;

View dropped.
```

```
SQL> create view marksview as select eedu_stu.name, eedu_stu.address, eedu_marks
.marks from eedu_stu, eedu_marks where eedu_stu.name = eedu_marks.name;

View created.

SQL> select * from MarksView;

NAME                 ADDRESS                MARKS
-------------------- -------------------- ----------
Eeshan               West Bengal                99
Aakash               Delhi                      85
Krish                Bangalore                  92
Shivansh             Nepal                      35
```

Result: Hence, the entity relationship diagram was implemented & successfully converted to respective relational tables.

Experiment - 7

Aim: To implement SQL joins (inner, right, left, outer)

Procedure:

1) Create some tables to perform joining operation.

⇒ CREATE TABLE <TABLE NAME>(ATTRIBUTES);

2) Insert values in to the tables

⇒ INSERT INTO <TABLE NAME> VALUES (ATTRIBUTES VALUES);

3) Inner join - Returns records that have matching values in both tables

4) outer join - Returns all records then there is a match in either left or right table

5) Left join - Returns all the records joined from left table plus the ones that matched on right table

6) Right join - Returns all the records joined from right table plus the ones that matched on left table

Output

```
SQL> select persons.lastname, persons.firstname, orders.orderno from persons left join orders on
persons.p_id = orders.p_id order by persons.lastname;

LASTNAME      FIRSTNAME                 ORDERNO
------------------- -------------------- ----------
Hansen                Ola                   22456
Hansen                Ola                   24562
Pettersen             Kari                          44678
Pettersen             Kari                          77895
Svendson      Tove

SQL> select persons.lastname, persons.firstname, orders.orderno from persons full outer join orders
on persons.p_id = orders.p_id order by persons.lastname;

LASTNAME      FIRSTNAME                 ORDERNO
------------------- -------------------- ----------
Hansen                Ola                   22456
Hansen                Ola                   24562
Pettersen             Kari                          44678
Pettersen             Kari                          77895
Svendson      Tove                                  34764
```

```
SQL> select * from persons;

      P_ID LASTNAME              FIRSTNAME            ADDRESS
---------- -------------------- -------------------- --------------------
CITY
--------------------
         1 Hansen               Ola                  Timoteivn 10
sandnes

         2 Svendson             Tove                 Borgan 23
sandnes

         2 Pettersen            Kari                 Storgt 20
Stavanger

SQL> select persons.lastname, persons.firstname, orders.orderno from persons left join orders on
persons.p_id = orders.p_id order by persons.lastname;

LASTNAME      FIRSTNAME                 ORDERNO
------------------- -------------------- ----------
Hansen                Ola                   22456
Hansen                Ola                   24562
Pettersen             Kari
Svendson      Tove
```

Result : SQL joins were successfully implemented.

Experiment - 8

Aim: To study the various SQL view operations on the database.

Procedure:

1) CREATE VIEW command is used to define a view

2) INSERT command is used to insert a new row into the view

3) DELETE command is used to change a value in a tuple without changing all values in the tuple

4) Create view of the table

⇒ CREATE VIEW view-name as SELECT * FROM Table-name WHERE <condition>;

Output -

```
Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production

SQL>
SQL> spool cursor.lst
SQL> set serveroutput on
SQL> select * from employees
  2
SQL> select * from employees;

      E_ID NAME                           SALARY     DEPTNO
---------- ------------------------------ ---------- ----------
         1 Aakash                          20000         20
         2 Harsh                           10000         10
         3 Ayush                           15000         18
         4 Abhishek                         9000         20

SQL> Declare
  2  num number(5);
  3  Begin
  4  update employees set salary = salary + salary*0.15 where deptno=20; if SQL%NOTFOUND then
  5  dbms_output.put_line('none of the salaries were updated');
  6
  7  elsif SQL%FOUND then
  8  num := SQL%ROWCOUNT;
  9  dbms_output.put_line('salaries for ' || num || 'employees are updated');
 10  end if;
 11  End;
 12  /
salaries for 2employees are updated

PL/SQL procedure successfully completed.
```

```
salaries for 2employees are updated

PL/SQL procedure successfully completed.
```

Result : various SQL view operations on the database were studied

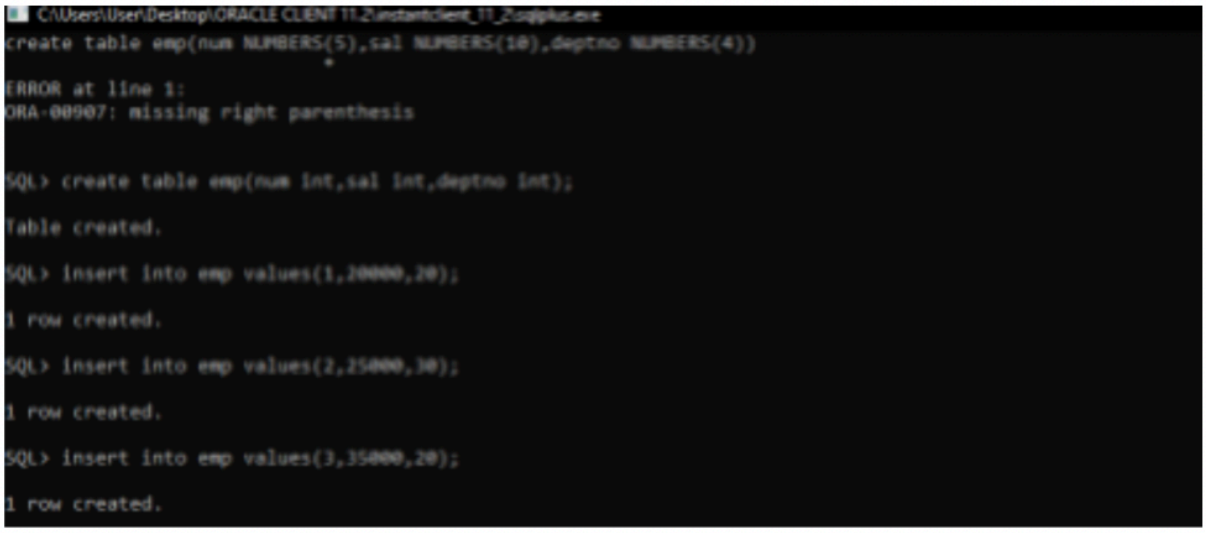
Experiment - 10

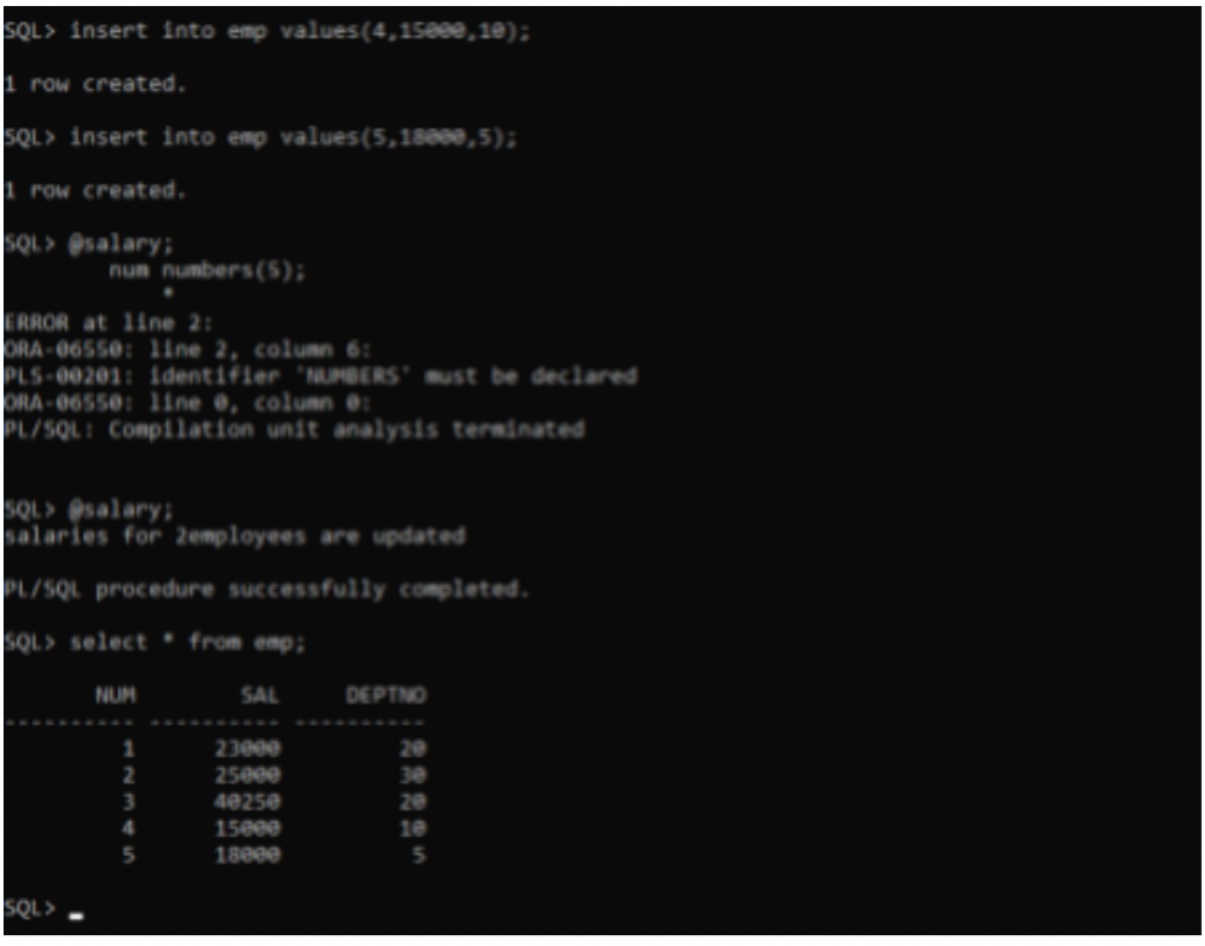
Aim: To write a program on PL/SQL procedures on sample exercises.

Procedure:

```
CREATE OR REPLACE PROCEDURE adjust_salary (
 in_employee_id IN EMPLOYEES.EMPLOYEE_ID%TYPE,
 in_percent IN NUMBER) IS
BEGIN
  -- update employee's salary
   UPDATE employees
   SET salary = salary + salary * in_percent / 100
   WHERE employee_id = in_employee_id ;
END.
```

Output -





Result: PL/SQL procedures were successfully executed.

Experiment - 11

Aim : To write a program on PL/SQL functions

Procedure

```
Declare
  a number;
  b number;
  c number;
  FUNCTION findMax (x IN number, y IN NUMBER);
  RETURN number;
  IS
     z number;
  BEGIN
     IF x>y THEN
        z:=x;
     ELSE
        z:=y;
     END IF
        RETURN z;
  END
  BEGIN
     a:=23;
     b:=45;
     c:= findMax(a,b);
```

```
dbms_output.put_line('Maximum of (23,45)
:'||c);
END
/
```

Output Screenshots -

```
Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production

SQL> CREATE OR REPLACE FUNCTION get_total_sales(
  2      in_year PLS_INTEGER
  3  )
  4  RETURN NUMBER
  5  IS
  6      l_total_sales NUMBER := 0;
  7  BEGIN
  8      -- get total sales
  9      SELECT SUM(unit_price * quantity)
 10      INTO l_total_sales
 11      FROM order_items
 12      INNER JOIN orders USING (order_id)
 13      WHERE status = 'Shipped'
 14      GROUP BY EXTRACT(YEAR FROM order_date)
 15      HAVING EXTRACT(YEAR FROM order_date) = in_year;
 16
```

```
SQL> CREATE OR REPLACE FUNCTION totalCustomers
  2  RETURN number IS
  3     total number(2) := 0;
  4  BEGIN
  5     SELECT count(*) into total
  6     FROM customer;
  7
  8     RETURN total;
  9  END;
 10  /

Function created.

SQL>
SQL> DECLARE
  2     c number(2);
  3  BEGIN
  4     c := totalCustomers();
  5     dbms_output.put_line('Total no. of Customers: ' || c);
  6  END;
  7  /

PL/SQL procedure successfully completed.

SQL>
```

```
SQL> CREATE OR REPLACE FUNCTION totalCustomers
  2  RETURN number IS
  3  total number(2):=0;
  4  BEGIN
  5  SELECT count(*) into total
  6  FROM customer;
  7  RETURN total;
  8  END;
  9  /

Function created.

SQL> DECLARE
  2  c number(2);
  3  BEGIN
  4  c:=totalCustomers();
  5  dbms_output.put_line('Total no. of Customers:'||c);
  6  END;
  7  /
Total no. of Customers:6

PL/SQL procedure successfully completed.
```

Result: PL/SQL Function was successfully executed.

Experiment - 13

Aim: To write Program on PL/SQL Execution handeling.

Procedure:

```
declare
n number;
grade varchar(1);
begin
n:=-1;
findGrade(n, grade);
if grade is not null then
dbms_output.put_line('Grade:' || grade);
end if;
end;
/
```

Output Screenshots -

```
SQL> spool exception.lst
SQL> set serveroutput on;
SQL> create table geeks(g_id int,g_name varchar(20),marks int);

Table created.

SQL> insert into geeks values(1,'Suraj',100);

1 row created.

SQL> insert into geeks values(2,'Praveen',97);

1 row created.

SQL> insert into geeks values(3,'Jessie',99);

1 row created.
```

```
SQL> DECLARE
  2  temp varchar(20);
  3
  4  BEGIN
  5  SELECT g_id into temp from geeks where g_name='GeeksforGeeks';
  6
  7  exception
  8  WHEN no_data_found THEN
  9  dbms_output.put_line('ERROR');
 10  dbms_output.put_line('there is no name as');
 11  dbms_output.put_line('GeeksforGeeks in geeks table');
 12  end;
 13
 14  /
ERROR
there is no name as
GeeksforGeeks in geeks table

PL/SQL procedure successfully completed.

SQL> DECLARE
  2  temp varchar(20);
```

```
  4  BEGIN
  5  SELECT g_id into temp from geeks where g_name='GeeksforGeeks';
  6
  7  exception
  8  WHEN no_data_found THEN
  9  dbms_output.put_line('ERROR');
 10  dbms_output.put_line('there is no name as');
 11  dbms_output.put_line('GeeksforGeeks in geeks table');
 12  end;
 13
 14  /
ERROR
there is no name as
GeeksforGeeks in geeks table

PL/SQL procedure successfully completed.

SQL> DECLARE
  2  temp varchar(20);
  3
  4  BEGIN
  5
  6  -- raises an exception as SELECT
  7  -- into trying to return too many rows
  8  SELECT g_name into temp from geeks;
  9  dbms_output.put_line(temp);
 10
 11  EXCEPTION
 12  WHEN too_many_rows THEN
 13  dbms_output.put_line('error trying to SELECT too many rows');
 14
 15  end;
 16  /
error trying to SELECT too many rows

PL/SQL procedure successfully completed.

SQL>
```

Result : PL/SQL Exception Handling was successfully executed

Experiment - 14

Aim : To write program on PL/SQL Trigger.

Procedure :

```
CREATE OR REPLACE TRIGGER invalid_Age;
BEFORE INSERT ON STUDENT D49
FOR EACH ROW
BEGIN
IF(:NEW.Age >100) THEN
raise_application_error(-20000,'inserting invalid ag
END IF;
END;
/
```

Output Screenshots -

```
SQL> spool trigger.lst
SQL> set serveroutput on;
SQL> desc customer;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(38)
 NAME                                               VARCHAR2(20)
 AGE                                                NUMBER(38)
 ADDRESS                                            VARCHAR2(20)
 SALARY                                             NUMBER(38)

SQL> select * from customer;

        ID NAME                      AGE ADDRESS                SALARY
---------- -------------------- ---------- -------------------- ----------
         1 Ramesh                       32 Ahmedabad                  2000
         2 Khilan                       25 Delhi                      1500
         3 Kaushik                      23 Kota                       2000
         4 Chaitali                     25 Mumbai                     6500
         5 Hardik                       27 Bhopal                     8500
         6 Komal                        22 MP                         4500

6 rows selected.
```

```
  7  BEGIN
  8     sal_diff := :NEW.salary  - :OLD.salary;
  9     dbms_output.put_line('Old salary: ' || :OLD.salary);
 10     dbms_output.put_line('New salary: ' || :NEW.salary);
 11     dbms_output.put_line('Salary difference: ' || sal_diff);
 12  END;
 13  /
BEFORE DELETE OR INSERT OR UPDATE ON customers
                                    *
ERROR at line 2:
ORA-00942: table or view does not exist


SQL> CREATE OR REPLACE TRIGGER display_salary_changes
  2  BEFORE DELETE OR INSERT OR UPDATE ON customer
  3  FOR EACH ROW
  4  WHEN (NEW.ID > 0)
  5  DECLARE
  6     sal_diff number;
  7  BEGIN
  8     sal_diff := :NEW.salary  - :OLD.salary;
  9     dbms_output.put_line('Old salary: ' || :OLD.salary);
 10     dbms_output.put_line('New salary: ' || :NEW.salary);
 11     dbms_output.put_line('Salary difference: ' || sal_diff);
 12  END;
 13  /

Trigger created.
```

```
Trigger created.

SQL> insert into customer (ID,NAME,AGE,ADDRESS,SALARY
  2  VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
Old salary:
New salary: 7500
Salary difference:

1 row created.

SQL> update customer
  2  SET salary = salary + 500
  3  WHERE id = 2;
Old salary: 1500
New salary: 2000
Salary difference: 500

1 row updated.

SQL>
```

Result: PL/SQL Trigger was successfully executed.