EX NO.: 01 DDL COMMANDS

AIM:

To work with DDL commands

DDL (Data Definition Language):

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. DDL is a set of SQL commands used to create, modify, and delete database structures but not data.

PROCEDURE:

Step 1: Open Run SQL on Command line and connect to SQL

Step 2: Then work with database using SQL queries.

CREATE:

This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).

SYNTAX:

CREATE TABLE table_name

(

column_Name1 data_type (size of the column) ,

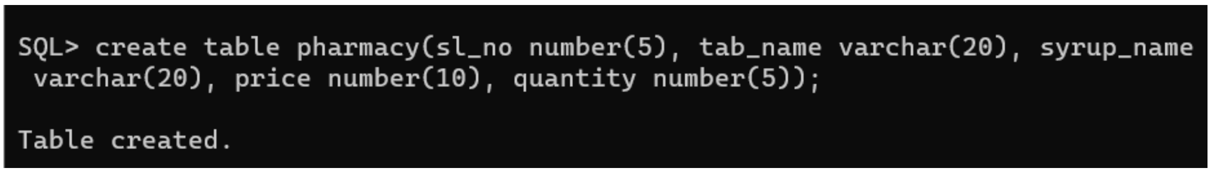
column_Name2 data_type (size of the column) ,

...

column_NameN data_type (size of the column)

) ;

OUTPUT:



```
SQL> create table pharmacy(sl_no number(5), tab_name varchar(20), syrup_name
 varchar(20), price number(10), quantity number(5));

Table created.
```

ALTER:

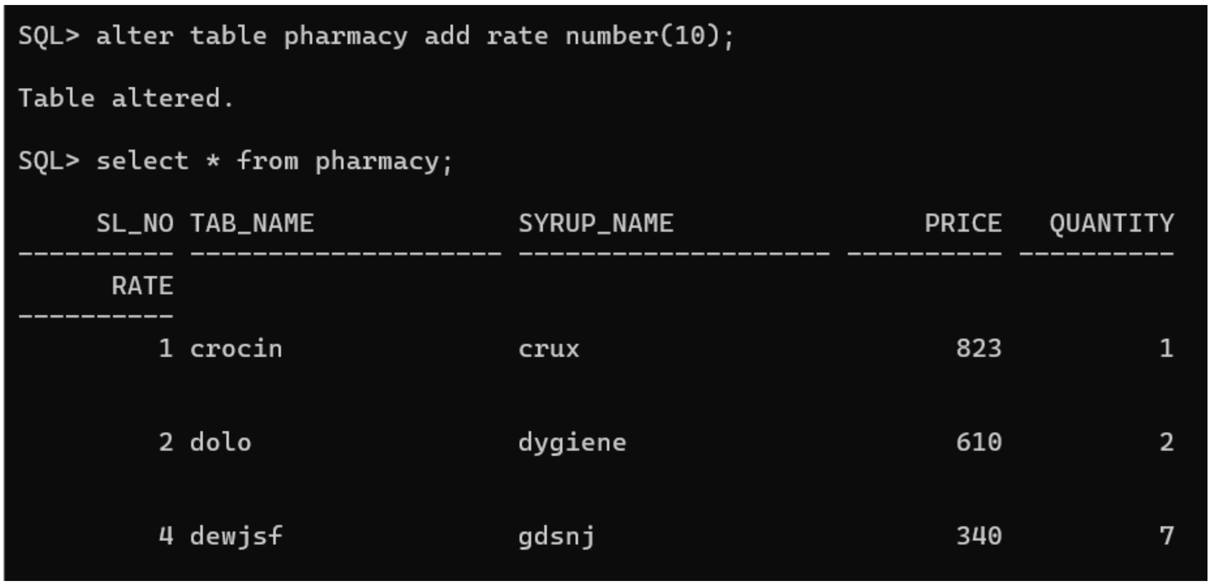
This is used to alter the structure of the database.

ALTER ADD:

Syntax to add a new field in the table:

ALTER TABLE name_of_table ADD column_name column_definition;

OUTPUT:



```
SQL> alter table pharmacy add rate number(10);

Table altered.

SQL> select * from pharmacy;

     SL_NO TAB_NAME             SYRUP_NAME                PRICE   QUANTITY
---------- -------------------- -------------------- ---------- ----------
      RATE
----------
         1 crocin               crux                        823          1


         2 dolo                 dygiene                     610          2


         4 dewjsf               gdsnj                       340          7
```

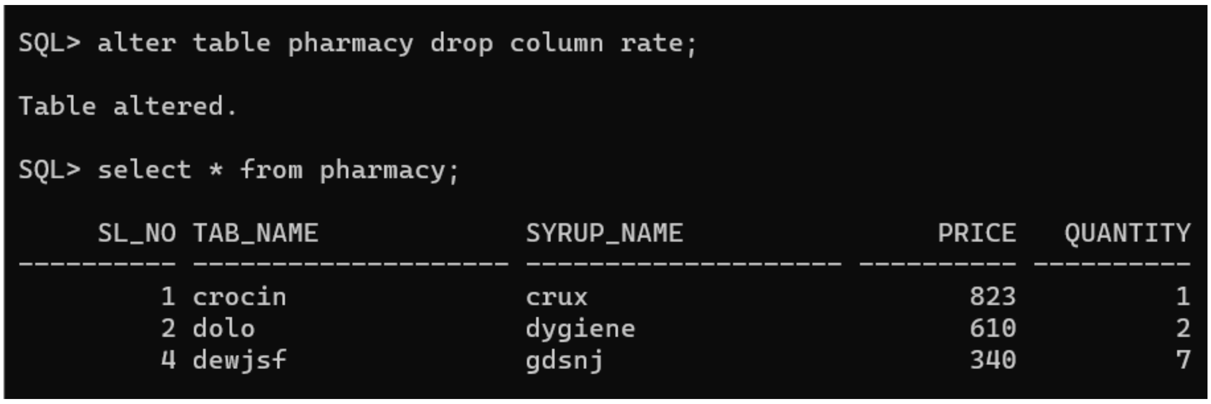
ALTER DROP:

Syntax to remove a column from the table:

ALTER TABLE name_of_table DROP

Column_Name_1 , column_Name_2 , ….., column_Name_N;

OUTPUT:



```
SQL> alter table pharmacy drop column rate;

Table altered.

SQL> select * from pharmacy;

     SL_NO TAB_NAME             SYRUP_NAME                PRICE   QUANTITY
---------- -------------------- -------------------- ---------- ----------
         1 crocin               crux                        823          1
         2 dolo                 dygiene                     610          2
         4 dewjsf               gdsnj                       340          7
```

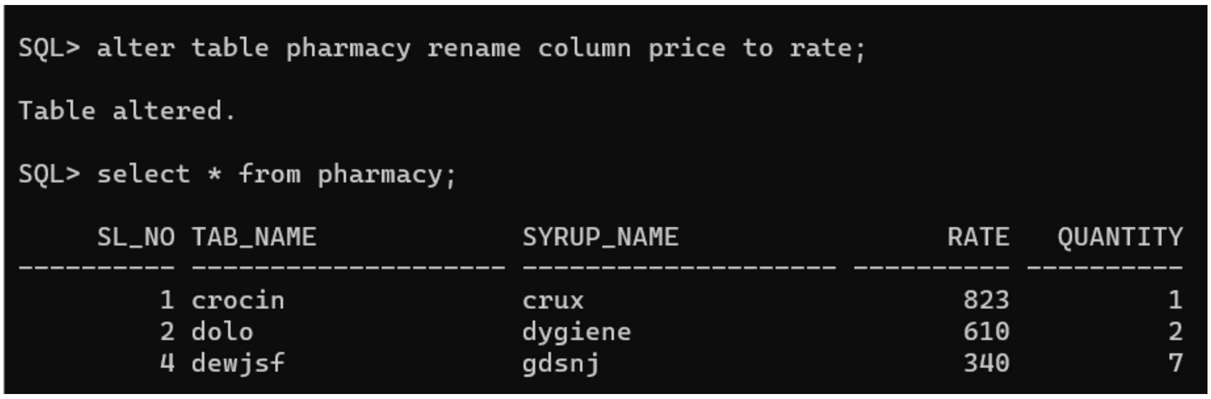
ALTER RENAME:

Syntax to Rename a column from the table:

ALTER TABLE name_of_table RENAME

Old Column_Name to New Column Name;

OUTPUT:



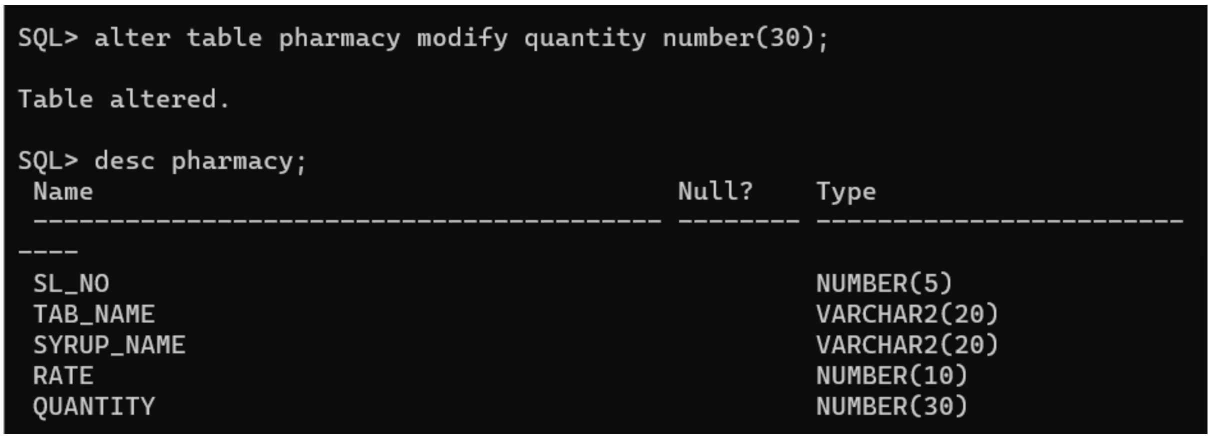
```
SQL> alter table pharmacy rename column price to rate;

Table altered.

SQL> select * from pharmacy;

     SL_NO TAB_NAME             SYRUP_NAME                 RATE   QUANTITY
---------- -------------------- -------------------- ---------- ----------
         1 crocin               crux                        823          1
         2 dolo                 dygiene                     610          2
         4 dewjsf               gdsnj                       340          7
```

ALTER MODIFY:

Syntax to modify the column of the table:

ALTER TABLE table_name MODIFY (column_name column_datatype(size));

OUTPUT:



```
SQL> alter table pharmacy modify quantity number(30);

Table altered.

SQL> desc pharmacy;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------------
----
 SL_NO                                              NUMBER(5)
 TAB_NAME                                           VARCHAR2(20)
 SYRUP_NAME                                         VARCHAR2(20)
 RATE                                               NUMBER(10)
 QUANTITY                                           NUMBER(30)
```

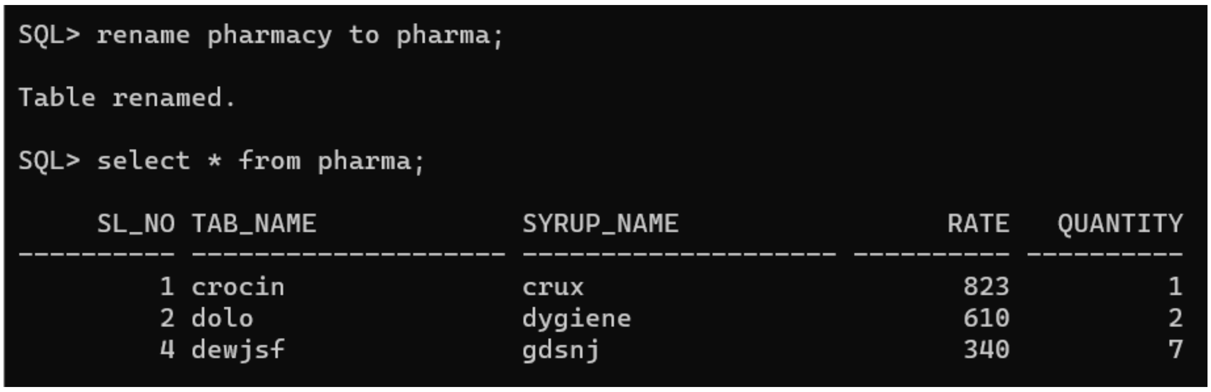
RENAME:

This is used to rename an object existing in the database.

Syntax of rename command:

RENAME TABLE Old_Table_Name TO New_Table_Name;

OUTPUT:



```
SQL> rename pharmacy to pharma;

Table renamed.

SQL> select * from pharma;

     SL_NO TAB_NAME             SYRUP_NAME                 RATE   QUANTITY
---------- -------------------- -------------------- ---------- ----------
         1 crocin               crux                        823          1
         2 dolo                 dygiene                     610          2
         4 dewjsf               gdsnj                       340          7
```

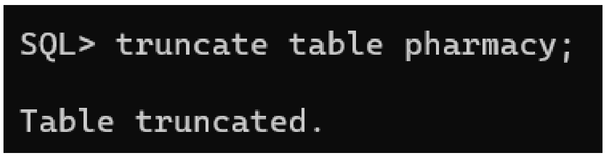
TRUNCATE:

This is used to remove all records from a table, including all spaces allocated for the records are removed.

Syntax of TRUNCATE command:

TRUNCATE TABLE Table_Name;

OUTPUT:



```
SQL> truncate table pharmacy;

Table truncated.
```

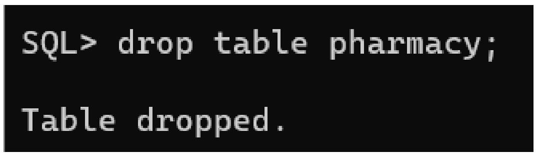
DROP:

This command is used to delete objects from the database.

Syntax to remove a table:

DROP TABLE Table_Name;

OUTPUT:



```
SQL> drop table pharmacy;

Table dropped.
```

RESULT:

The queries for DDL commands were successfully executed and the output is noted.

EX NO.: 02 DML COMMANDS

AIM:

To work with DML commands

DML (Data Manipulation Language):

The DML commands in Structured Query Language change the data present in the SQL database. We can easily access, store, modify, update and delete the existing records from the database using DML commands

PROCEDURE:

Step 1: Open Run SQL on Command line and connect to SQL

Step 2: Then work with database using SQL queries.

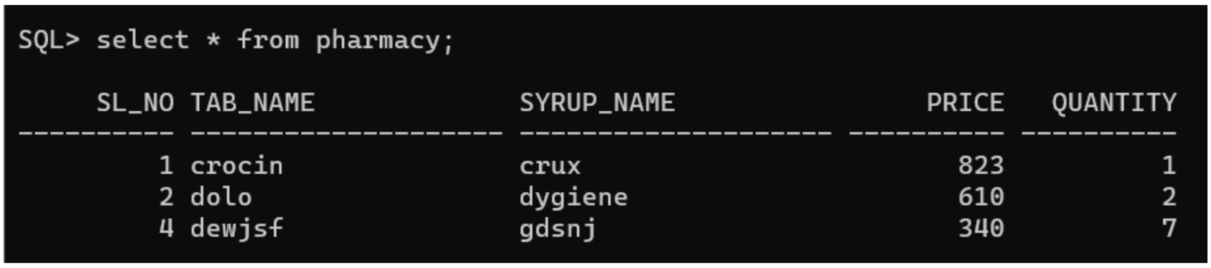
SELECT:

SELECT is the most important data manipulation command in Structured Query Language. The SELECT command shows the records of the specified table. It also shows the particular record of a particular column by using the WHERE clause.

Syntax of SELECT DML command

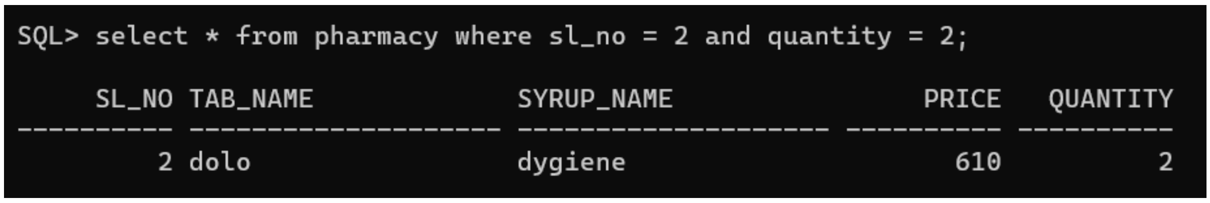
SELECT column_Name_1, column_Name_2, ….., column_Name_N FROM Name_of_table;

OUTPUT:



```
SQL> select * from pharmacy;

     SL_NO TAB_NAME             SYRUP_NAME                PRICE   QUANTITY
---------- -------------------- -------------------- ---------- ----------
         1 crocin               crux                        823          1
         2 dolo                 dygiene                     610          2
         4 dewjsf               gdsnj                       340          7
```

OUTPUT for SELECT using WHERE:



```
SQL> select * from pharmacy where sl_no = 2 and quantity = 2;

     SL_NO TAB_NAME             SYRUP_NAME                PRICE   QUANTITY
---------- -------------------- -------------------- ---------- ----------
         2 dolo                 dygiene                     610          2
```

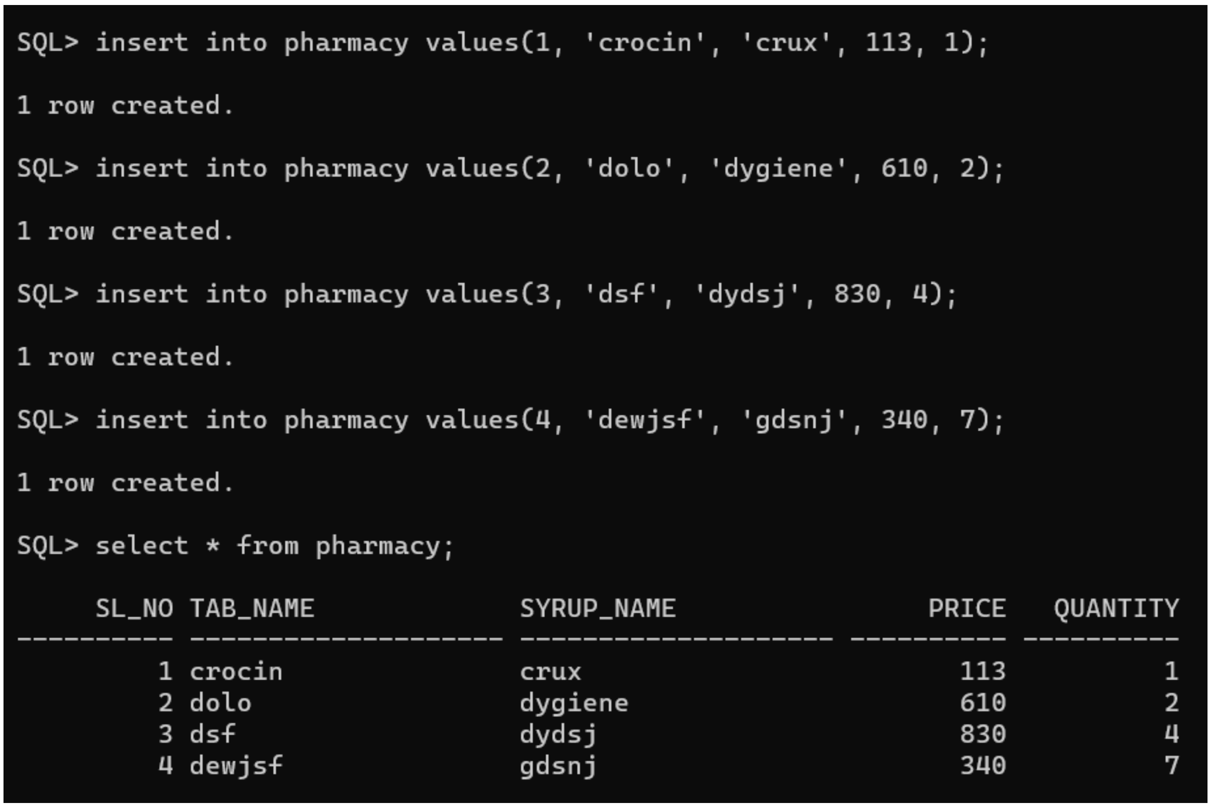
INSERT:

INSERT is another most important data manipulation command in Structured Query Language, which allows users to insert data in database tables.

Syntax of INSERT Command

INSERT INTO TABLE_NAME (column_Name1 , column_Name2 , column_Name3 , ... column _ NameN) **VALUES** (value_1, value_2, value_3, ... value_N) ;

OUTPUT:



```
SQL> insert into pharmacy values(1, 'crocin', 'crux', 113, 1);

1 row created.

SQL> insert into pharmacy values(2, 'dolo', 'dygiene', 610, 2);

1 row created.

SQL> insert into pharmacy values(3, 'dsf', 'dydsj', 830, 4);

1 row created.

SQL> insert into pharmacy values(4, 'dewjsf', 'gdsnj', 340, 7);

1 row created.

SQL> select * from pharmacy;

     SL_NO TAB_NAME             SYRUP_NAME                PRICE   QUANTITY
---------- -------------------- -------------------- ---------- ----------
         1 crocin               crux                        113          1
         2 dolo                 dygiene                     610          2
         3 dsf                  dydsj                       830          4
         4 dewjsf               gdsnj                       340          7
```

UPDATE:

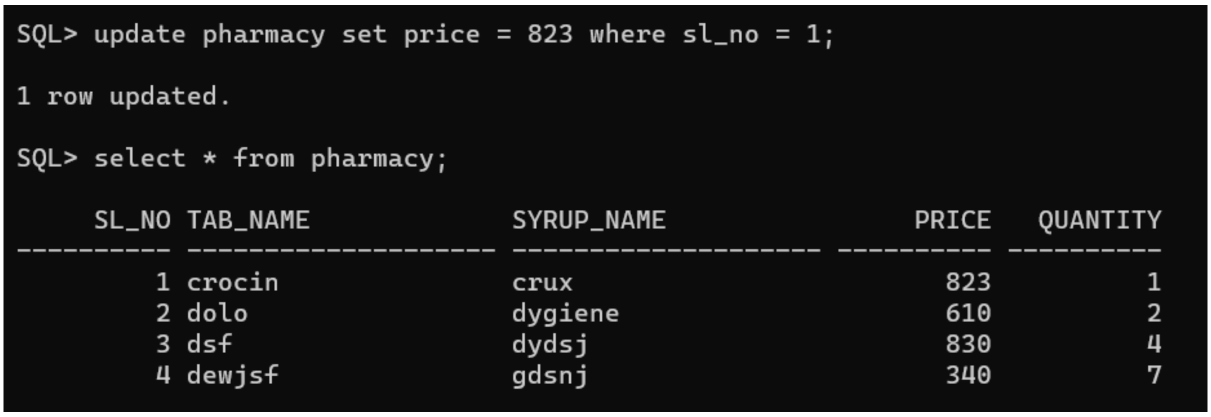
UPDATE is another most important data manipulation command in Structured Query Language, which allows users to update or modify the existing data in database tables.

Syntax of UPDATE Command

UPDATE Table_name **SET** [column_name1= value_1, ….., column_nameN = value_N]

WHERE CONDITION;

OUTPUT:



```
SQL> update pharmacy set price = 823 where sl_no = 1;

1 row updated.

SQL> select * from pharmacy;

     SL_NO TAB_NAME             SYRUP_NAME                PRICE   QUANTITY
---------- -------------------- -------------------- ---------- ----------
         1 crocin               crux                        823          1
         2 dolo                 dygiene                     610          2
         3 dsf                  dydsj                       830          4
         4 dewjsf               gdsnj                       340          7
```

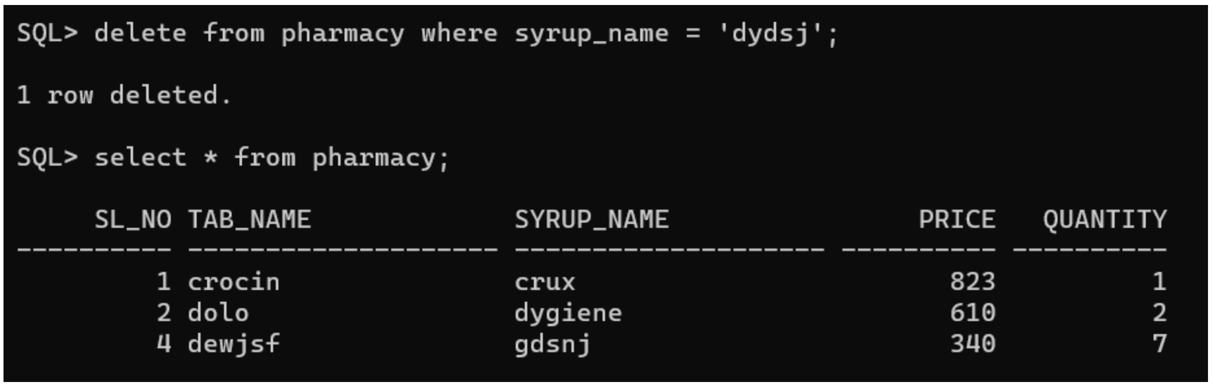
DELETE:

DELETE is a DML command which allows SQL users to remove single or multiple existing records from the database tables.

Syntax of DELETE Command

DELETE FROM Table_Name **WHERE** condition;

OUTPUT:



```
SQL> delete from pharmacy where syrup_name = 'dydsj';

1 row deleted.

SQL> select * from pharmacy;

     SL_NO TAB_NAME             SYRUP_NAME                PRICE   QUANTITY
---------- -------------------- -------------------- ---------- ----------
         1 crocin               crux                        823          1
         2 dolo                 dygiene                     610          2
         4 dewjsf               gdsnj                       340          7
```

RESULT:

The queries for DML commands were successfully executed and the output is noted.

EX NO: 03 DCL COMMANDS

AIM:

To work with DCL commands

PROCEDURE:

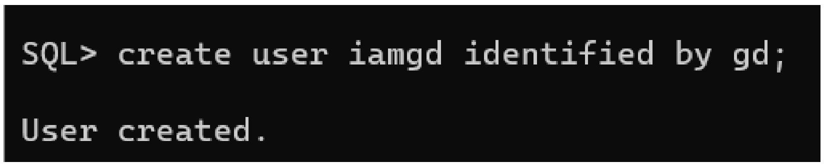
Step 1: Open Run SQL on Command line and connect to SQL

Step 2: Then work with database using SQL queries.

(DCL)Data control language:

Data control language is used to access the stored data. It is mainly used for revoke and to grant the user the required access to a database. In the database, this language does not have the feature of rollback.

USER CREATION:



```
SQL> create user iamgd identified by gd;

User created.
```

1. GRANT:

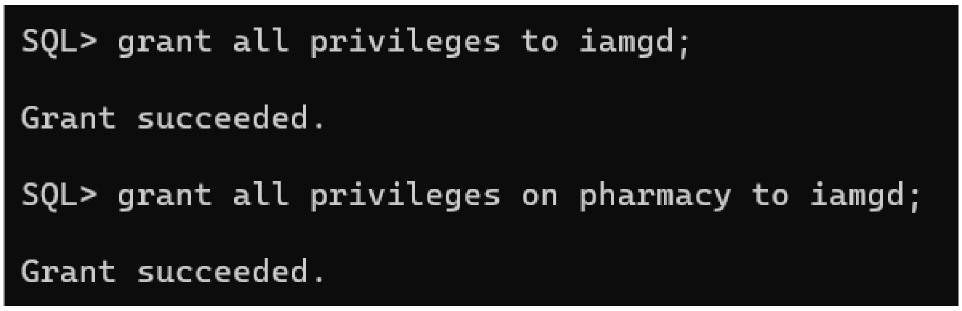
SQL Grant command is specifically used to provide privileges to database objects for a user. This command also allows users to grant permissions to other users too.

Syntax:

grant privilege_name on object_name

to {user_name | public | role_name}

OUTPUT:



```
SQL> grant all privileges to iamgd;

Grant succeeded.

SQL> grant all privileges on pharmacy to iamgd;

Grant succeeded.
```

2. REVOKE:

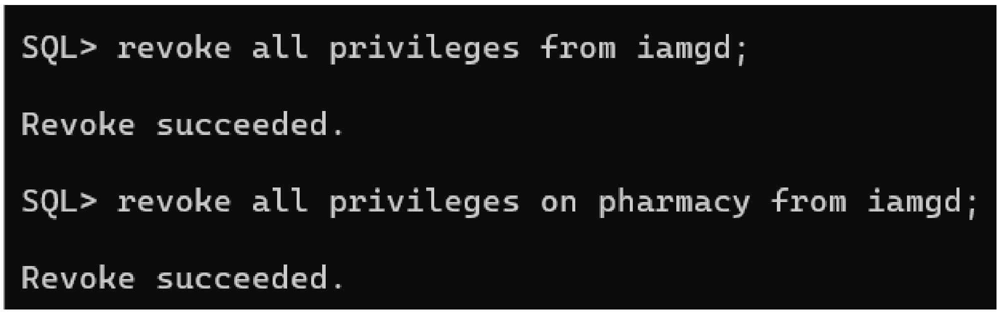
Revoke command withdraw user privileges on database objects if any granted. It does operations opposite to the Grant command. When a privilege is revoked from a particular user U, then the privileges granted to all other users by user U will be revoked.

Syntax:

revoke privilege_name on object_name

from {user_name | public | role_name}

OUTPUT:



```
SQL> revoke all privileges from iamgd;

Revoke succeeded.

SQL> revoke all privileges on pharmacy from iamgd;

Revoke succeeded.
```

RESULT:

The queries for DCL commands were successfully executed and the output is noted.

EX NO.: 04 SUB QUERIES AND JOINS

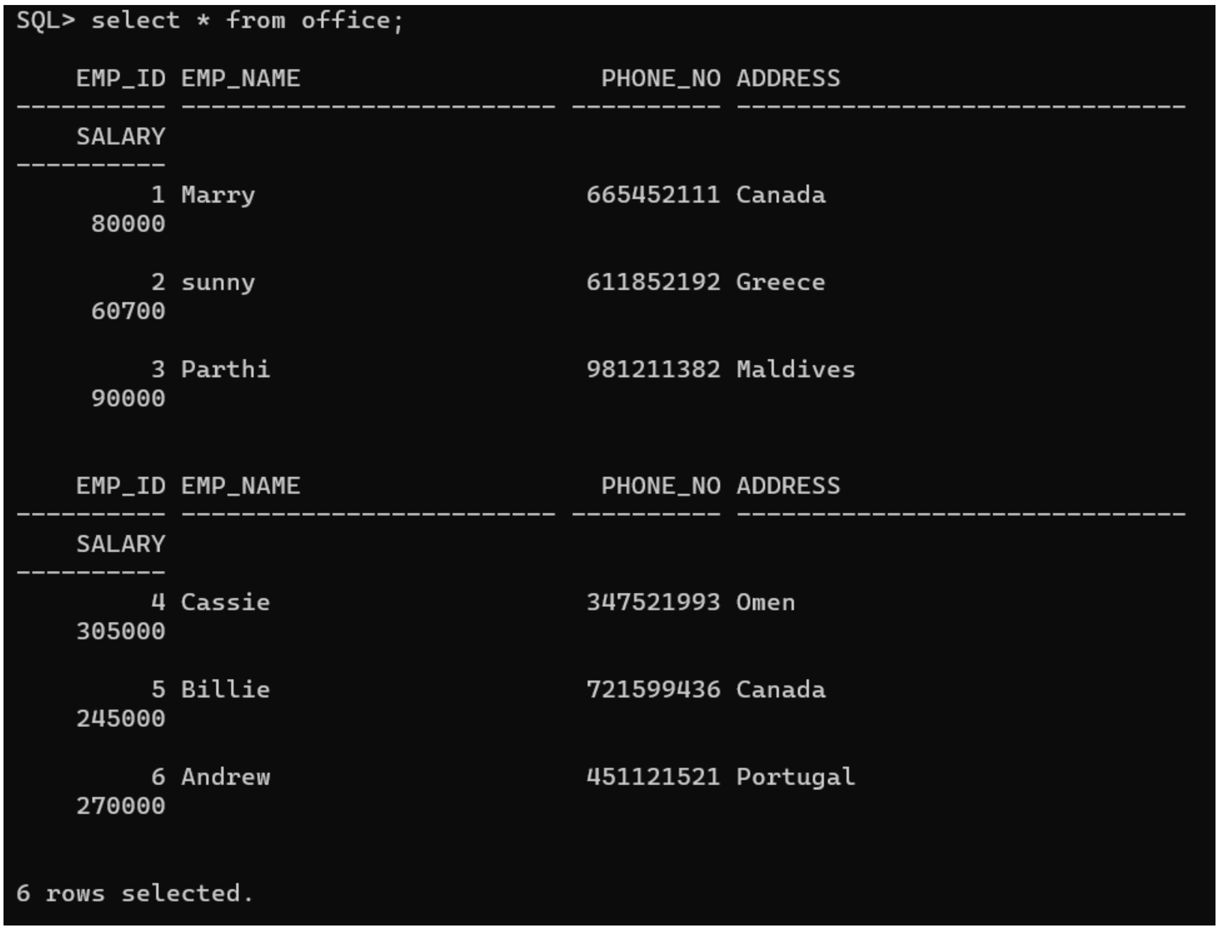
AIM:

To work with Sub queries and joins

SUB QUERIES:

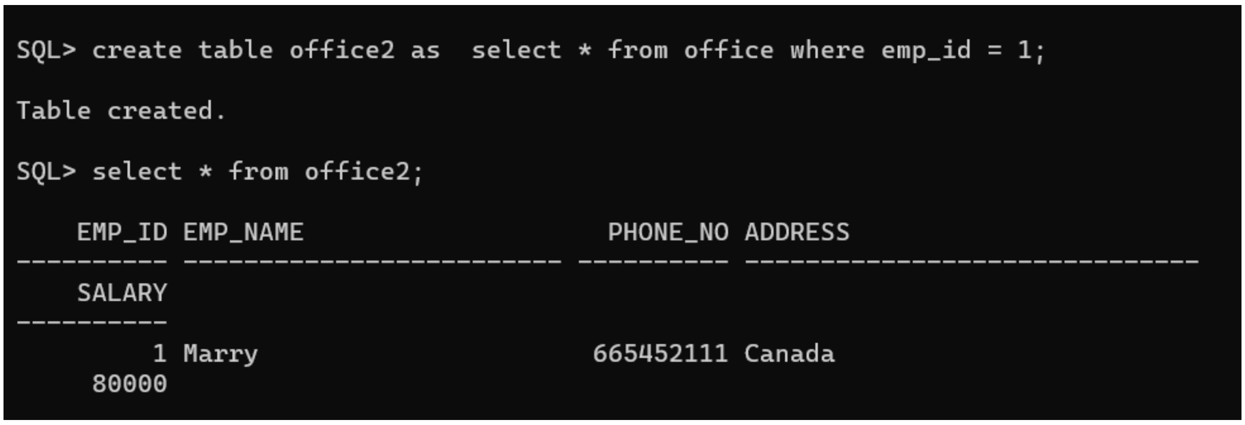
In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query.

TABLE 1 – OFFICE:



```
SQL> select * from office;

    EMP_ID EMP_NAME                    PHONE_NO ADDRESS
---------- ------------------------- ---------- ------------------------------
    SALARY
----------
         1 Marry                      665452111 Canada
     80000

         2 sunny                      611852192 Greece
     60700

         3 Parthi                     981211382 Maldives
     90000


    EMP_ID EMP_NAME                    PHONE_NO ADDRESS
---------- ------------------------- ---------- ------------------------------
    SALARY
----------
         4 Cassie                     347521993 Omen
    305000

         5 Billie                     721599436 Canada
    245000

         6 Andrew                     451121521 Portugal
    270000


6 rows selected.
```

TABLE 2 – OFFICE 2:



```
SQL> create table office2 as  select * from office where emp_id = 1;

Table created.

SQL> select * from office2;

    EMP_ID EMP_NAME                     PHONE_NO ADDRESS
---------- ------------------------- ---------- ------------------------------
    SALARY
----------
         1 Marry                       665452111 Canada
     80000
```

SUB QUERIES WITH SELECT STATEMENT:

Subqueries are most frequently used with the SELECT statement.

The basic syntax is as follows –

SELECT column_name [, column_name]

FROM   table1 [, table2]

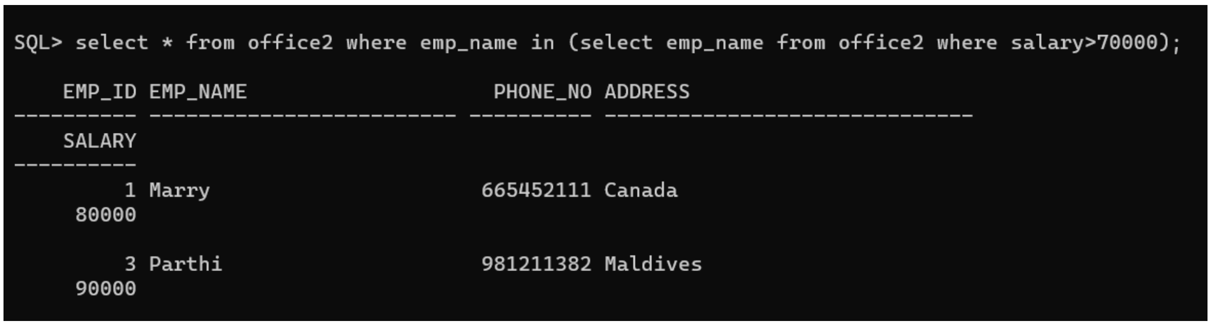
WHERE column_name OPERATOR

  (SELECT column_name [, column_name]

  FROM table1 [, table2]

  [WHERE])

OUTPUT:



```
SQL> select * from office2 where emp_name in (select emp_name from office2 where salary>70000);

    EMP_ID EMP_NAME                     PHONE_NO ADDRESS
---------- ------------------------- ---------- ------------------------------
    SALARY
----------
         1 Marry                       665452111 Canada
     80000

         3 Parthi                      981211382 Maldives
     90000
```

SUB QUERIES WITH INSERT STATEMENT:

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows.

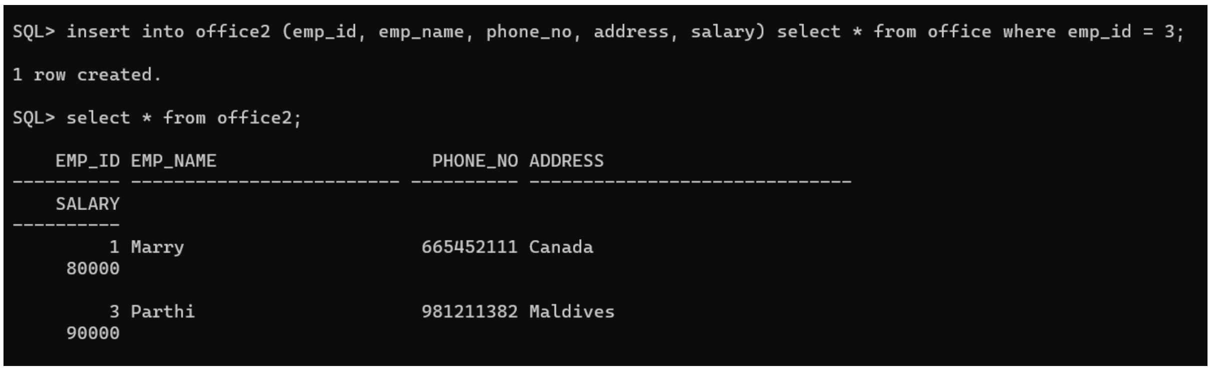
INSERT INTO table_name [(column1 [, column2])]

SELECT [*|column1 [, column2]

FROM table1 [, table2]

[WHERE VALUE OPERATOR]

OUTPUT:



```
SQL> insert into office2 (emp_id, emp_name, phone_no, address, salary) select * from office where emp_id = 3;

1 row created.

SQL> select * from office2;

    EMP_ID EMP_NAME                    PHONE_NO ADDRESS
---------- ------------------------- ---------- ------------------------------
    SALARY
----------
         1 Marry                      665452111 Canada
     80000

         3 Parthi                     981211382 Maldives
     90000
```

SUB QUERIES WITH UPDATE STATEMENT:

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows

UPDATE table

SET column_name = new_value

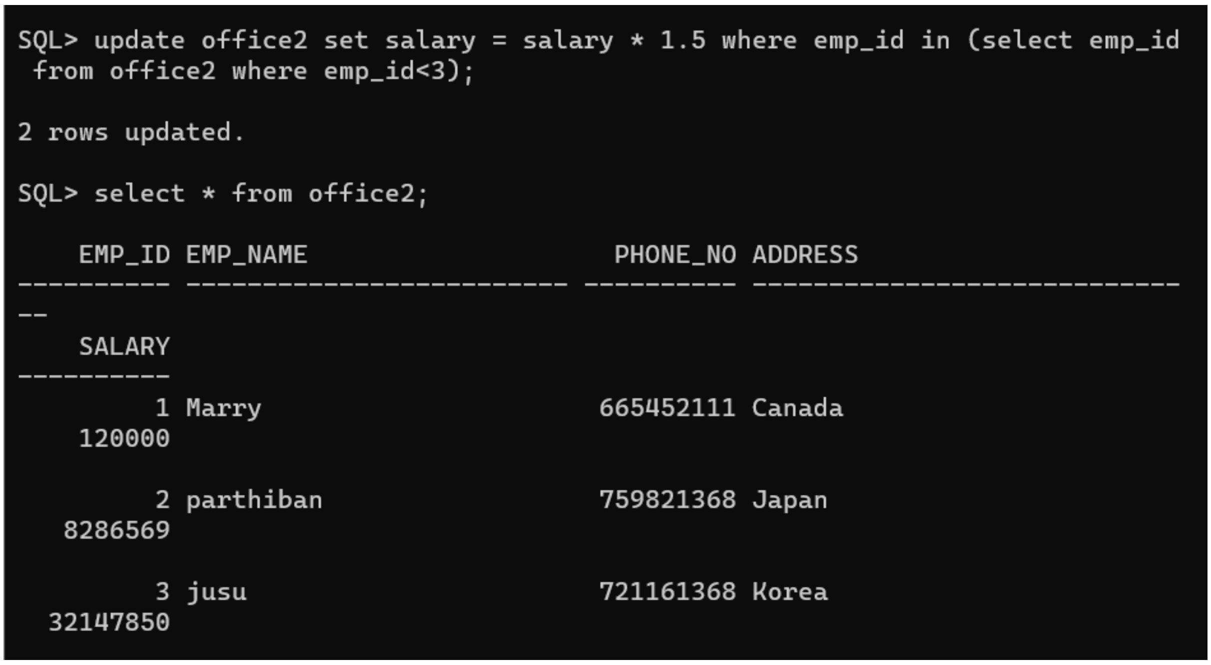
[WHERE OPERATOR [VALUE]

(SELECT COLUMN_NAME

FROM TABLE_NAME)

[WHERE)]

OUTPUT:



```
SQL> update office2 set salary = salary * 1.5 where emp_id in (select emp_id
 from office2 where emp_id<3);

2 rows updated.

SQL> select * from office2;

    EMP_ID EMP_NAME                      PHONE_NO ADDRESS
---------- ------------------------- ---------- ----------------------------
--
    SALARY
----------
         1 Marry                      665452111 Canada
    120000

         2 parthiban                  759821368 Japan
   8286569

         3 jusu                       721161368 Korea
  32147850
```

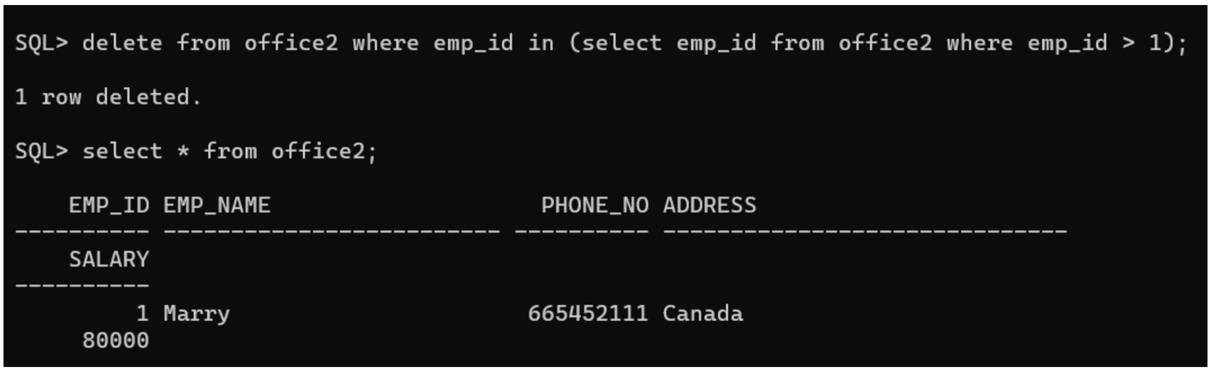
SUB QUERIES WITH DELETE STATEMENT:

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows.

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

OUTPUT:



```
SQL> delete from office2 where emp_id in (select emp_id from office2 where emp_id > 1);

1 row deleted.

SQL> select * from office2;

    EMP_ID EMP_NAME                      PHONE_NO ADDRESS
---------- ------------------------- ---------- ------------------------------
    SALARY
----------
         1 Marry                      665452111 Canada
     80000
```

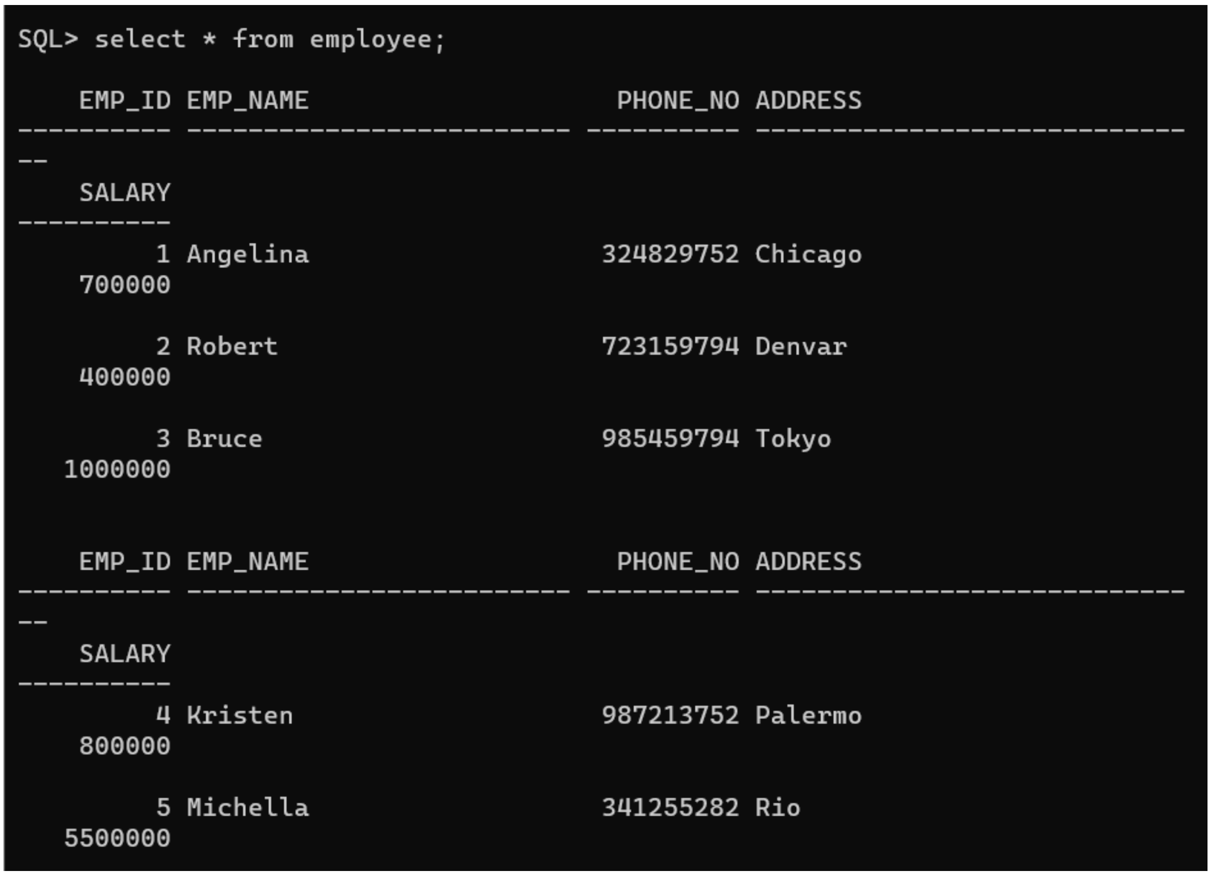
JOINS:

Different types of Joins are as follows:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

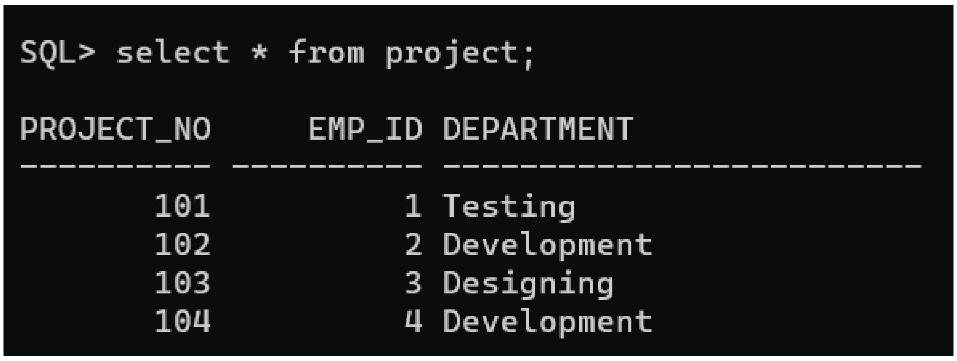
Consider the two tables below:

EMPLOYEE TABLE:



```
SQL> select * from employee;

    EMP_ID EMP_NAME                    PHONE_NO ADDRESS
---------- ------------------------- ---------- ----------------------------
--
    SALARY
----------
         1 Angelina                   324829752 Chicago
    700000

         2 Robert                     723159794 Denvar
    400000

         3 Bruce                      985459794 Tokyo
   1000000


    EMP_ID EMP_NAME                    PHONE_NO ADDRESS
---------- ------------------------- ---------- ----------------------------
--
    SALARY
----------
         4 Kristen                    987213752 Palermo
    800000

         5 Michella                   341255282 Rio
   5500000
```

PROJECT TABLE:



```
SQL> select * from project;

PROJECT_NO     EMP_ID DEPARTMENT
---------- ---------- -------------------------
       101          1 Testing
       102          2 Development
       103          3 Designing
       104          4 Development
```

A. INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

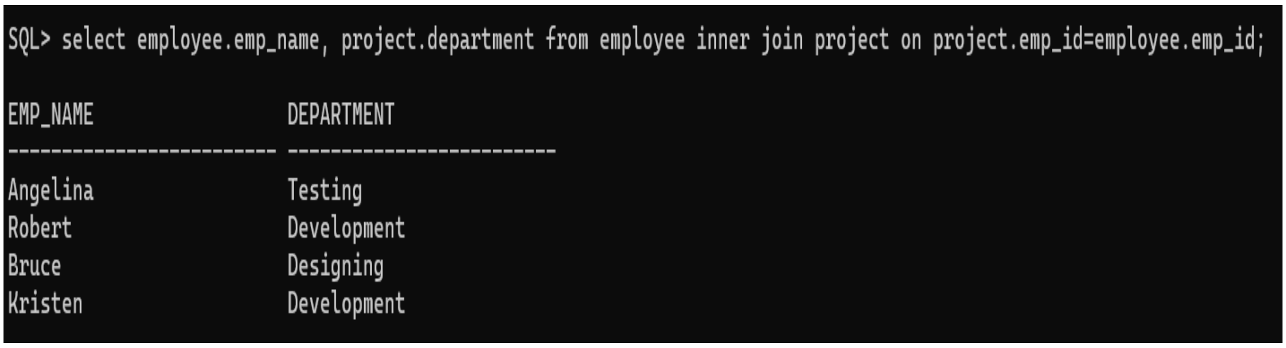
ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

OUTPUT:



```
SQL> select employee.emp_name, project.department from employee inner join project on project.emp_id=employee.emp_id;

EMP_NAME                  DEPARTMENT
------------------------- -------------------------
Angelina                  Testing
Robert                    Development
Bruce                     Designing
Kristen                   Development
```

B. LEFT JOIN

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

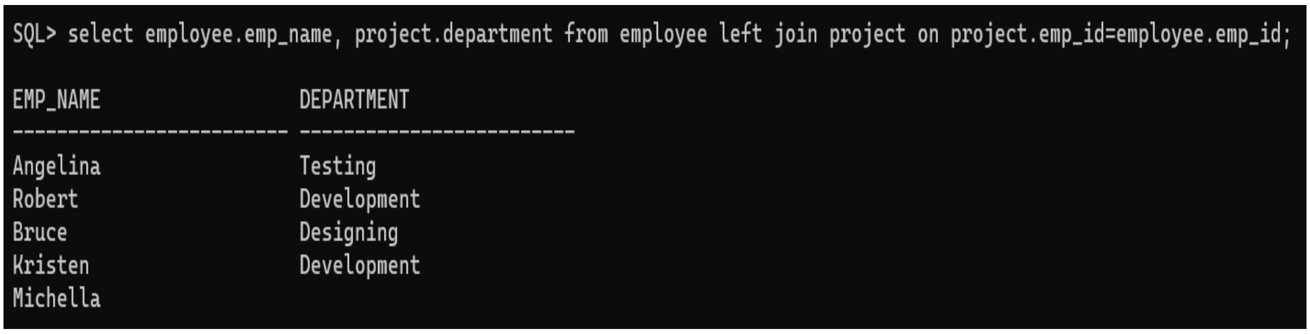
ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

OUTPUT:



```
SQL> select employee.emp_name, project.department from employee left join project on project.emp_id=employee.emp_id;

EMP_NAME                 DEPARTMENT
------------------------ ------------------------
Angelina                 Testing
Robert                   Development
Bruce                    Designing
Kristen                  Development
Michella
```

C. RIGHT JOIN

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

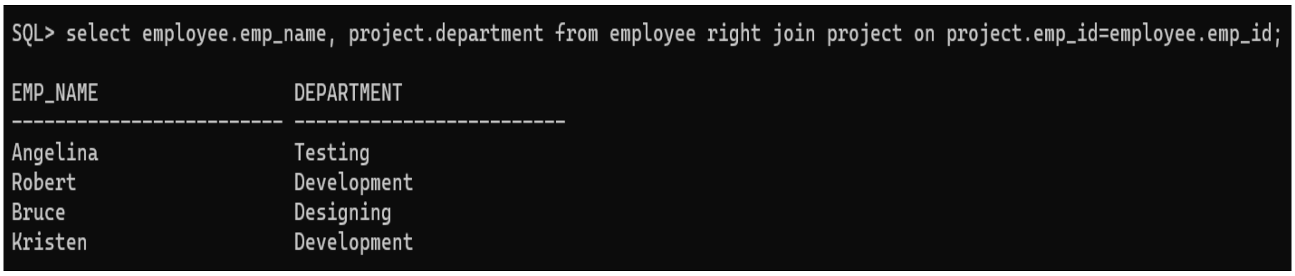
ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

OUTPUT:



```
SQL> select employee.emp_name, project.department from employee right join project on project.emp_id=employee.emp_id;

EMP_NAME                 DEPARTMENT
------------------------ ------------------------
Angelina                 Testing
Robert                   Development
Bruce                    Designing
Kristen                  Development
```

D. FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain *NULL* values.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

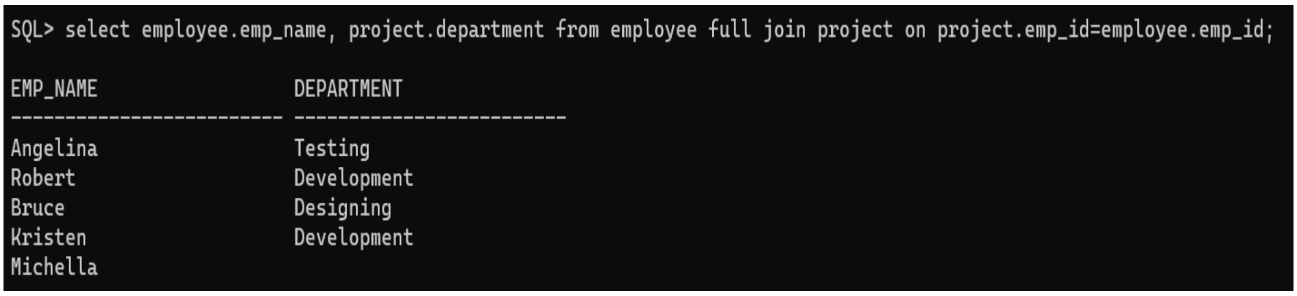
ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

OUTPUT:



```
SQL> select employee.emp_name, project.department from employee full join project on project.emp_id=employee.emp_id;

EMP_NAME                 DEPARTMENT
------------------------ ------------------------
Angelina                 Testing
Robert                   Development
Bruce                    Designing
Kristen                  Development
Michella
```

RESULT:

The queries for Sub queries and Joins were successfully executed and the output is noted.

EX NO.: 05 PL/SQL

AIM:

To work with PL/SQL commands

PROCEDURE:

Step1: Open Run SQL on Command line and connect to SQL .

Step 2: Then work with database using PL/SQL Block commands

i. Declare
ii. Begin
iii. Exception
iv. End

SYNTAX:

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling>
END;
```

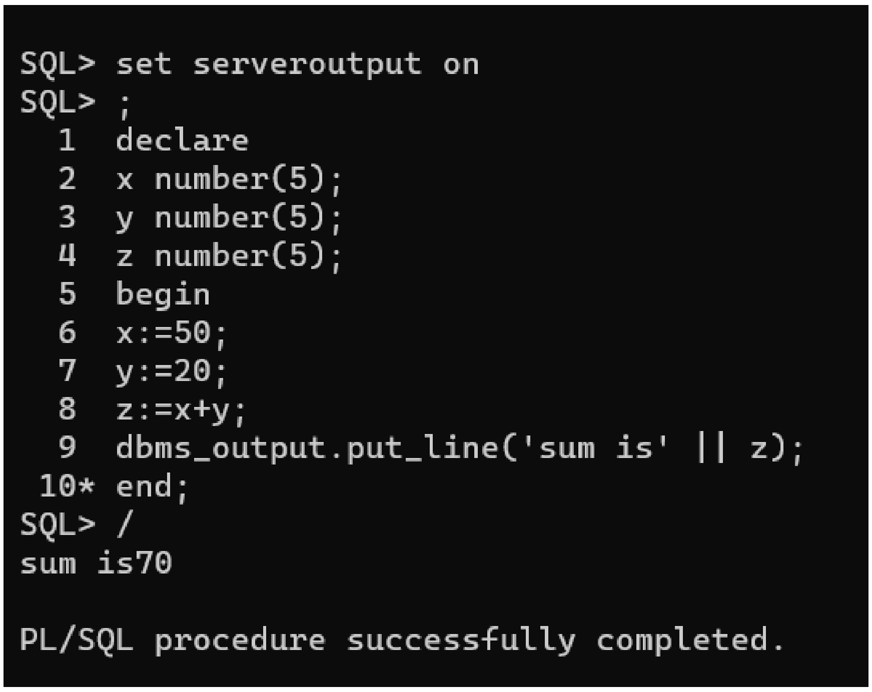
EXAMPLE:

1. **ADDITION OF TWO NUMBERS:**

PROGRAM CODE:

```
SQL> SET SERVEROUTPUT ON;
SQL> declare
x number(5);
y number(5);
z number(5);
begin
x:=50;
y:=20;
z:=x+y;
dbms_output.put_line('sum is'||z);
end;
/
```

OUTPUT:



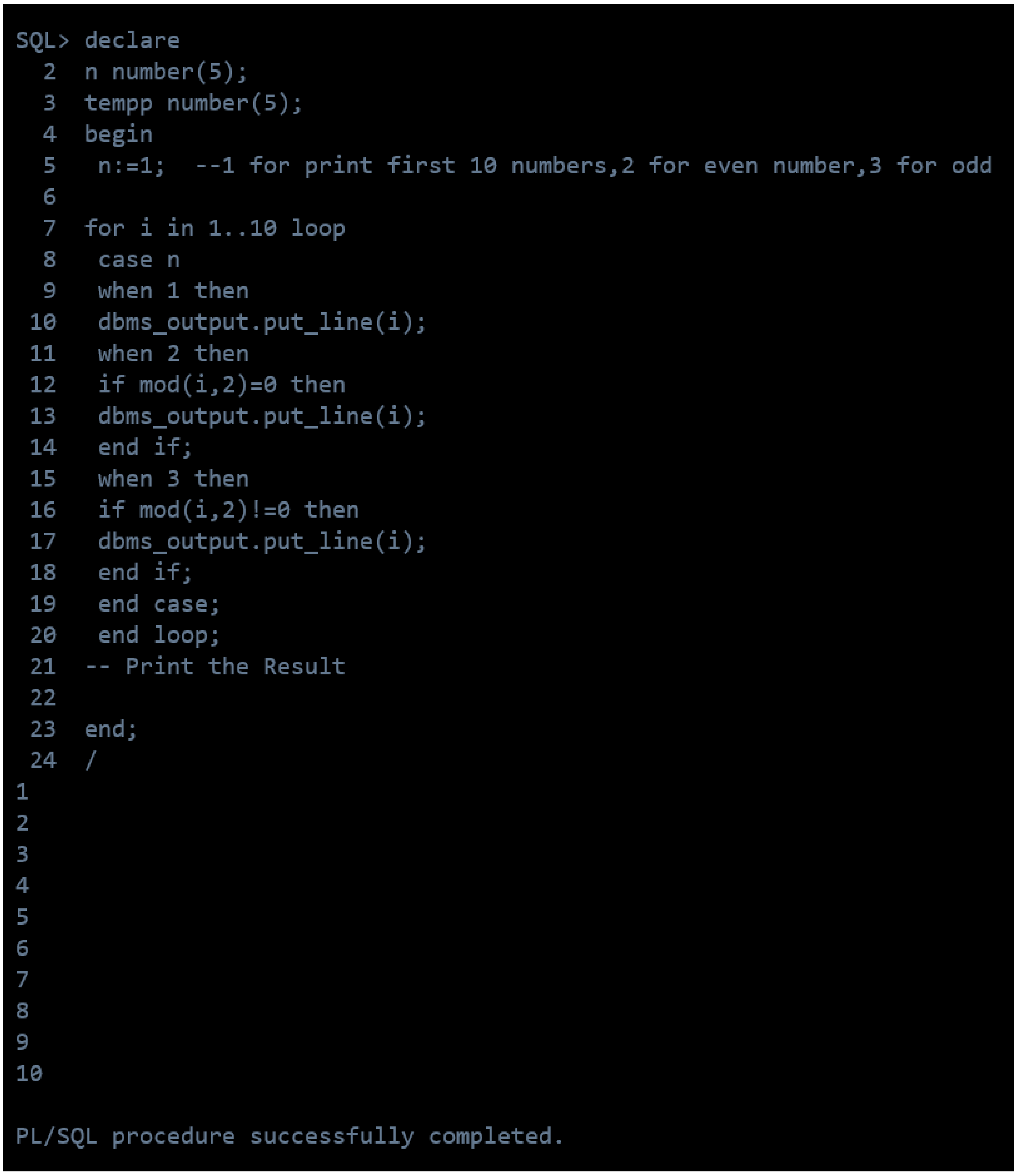
```
SQL> set serveroutput on
SQL> ;
declare
x number(5);
y number(5);
z number(5);
begin
x:=50;
y:=20;
z:=x+y;
dbms_output.put_line('sum is' || z);
end;
SQL> /
sum is70

PL/SQL procedure successfully completed.
```

2. **GENERATING SERIES:**

PROGRAM CODE:

```
SQL> SET SERVEROUTPUT ON;
SQL> declare
n number(5);
tempp number(5);
begin
 n:=1;  --1 for print first 10 numbers,2 for even number,3 for odd
for i in 1..10 loop
 case n
when 1 then
 dbms_output.put_line(i);
 when 2 then
 if mod(i,2)=0 then
 dbms_output.put_line(i);
 end if;
 when 3 then
 if mod(i,2)!=0 then
 dbms_output.put_line(i);
 end if;
 end case;
 end loop;
end;
/
```

OUTPUT:



```
SQL> declare
  2  n number(5);
  3  tempp number(5);
  4  begin
  5   n:=1;  --1 for print first 10 numbers,2 for even number,3 for odd
  6
  7  for i in 1..10 loop
  8   case n
  9   when 1 then
 10   dbms_output.put_line(i);
 11   when 2 then
 12   if mod(i,2)=0 then
 13   dbms_output.put_line(i);
 14   end if;
 15   when 3 then
 16   if mod(i,2)!=0 then
 17   dbms_output.put_line(i);
 18   end if;
 19   end case;
 20   end loop;
 21  -- Print the Result
 22
 23  end;
 24  /
1
2
3
4
5
6
7
8
9
10

PL/SQL procedure successfully completed.
```

RESULT:

The PL/SQL queries were successfully executed and the output is noted.

EX NO.: 06 CURSOR PROCEDURE FUNCTIONS

AIM:

To write a SQL program to work with cursor, procedure and functions.

PROCEDURE:

Step 1: Open Run SQL on Command line and connect to SQL

Step 2: Then work with database using SQL queries.

PL/SQL PROCEDURE:

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

- **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
- **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

Syntax for creating procedure:

```
CREATE [OR REPLACE] PROCEDURE procedure_name

   [ (parameter [,parameter]) ]

IS

   [declaration_section]

BEGIN

   executable_section

[EXCEPTION

   exception_section]

END [procedure_name];
```

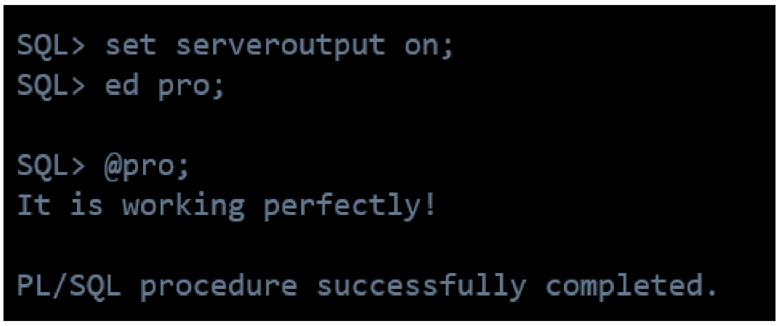
TABLE QUERY:

```
create table employee(emp_id number(5)primary key, emp_name varchar2(20), city varchar2(20), salary number(7), age number(5));

 insert into employee values (1, 'Raju', 'Pdy', 800000, 20);

 insert into employee values (2, 'Niteesh', 'Pdy', 790000, 21);

 insert into employee values (3, 'Punith', 'AP', 750000, 20);

 insert into employee values (4, 'Sidharth', 'MP', 650000, 21);

 insert into employee values (5, 'Mantu', 'Delhi', 900000, 22);
```

PROGRAM CODE:

```
DECLARE

PROCEDURE pro

AS

BEGIN

   dbms_output.put_line('It is working perfectly!');

END;

BEGIN

pro();

END;

/
```

OUTPUT:



```
SQL> set serveroutput on;
SQL> ed pro;

SQL> @pro;
It is working perfectly!

PL/SQL procedure successfully completed.
```

PL/SQL – CURSORS:

A cursor is used to referred to 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 CURSOR:

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement.

1 %FOUND

Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

2 %NOTFOUND

The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.

3 %ISOPEN

Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.

4 %ROWCOUNT

Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

EXPLICIT CURSOR:

Explicit cursors are programmer-defined cursors for gaining more control over the context area.

The syntax for creating an explicit cursor is −

CURSOR cursor_name IS select_statement;

Working with an explicit cursor includes the following steps −

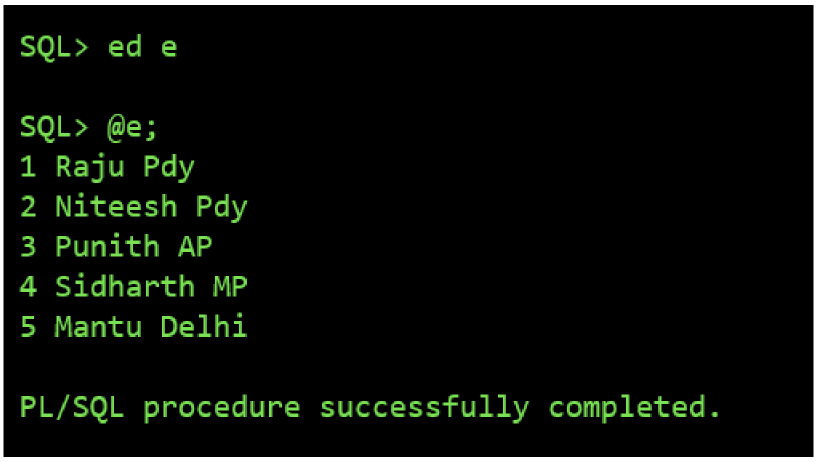
- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

PROGRAM CODE:

```
DECLARE

   e_id employee.emp_id%type;

   e_name employee.emp_name%type;

   e_city employee.city%type;

   cursor e_employee is

   select emp_id, emp_name, city from employee;

begin

   open e_employee;

   loop
```

```
  fetch e_employee into e_id, e_name, e_city;

  exit when e_employee%notfound;

  dbms_output.put_line(e_id || ' ' || e_name || ' ' || e_city);

  end loop;

  close e_employee;

end;

/
```

OUTPUT:



```
SQL> ed e

SQL> @e;
1 Raju Pdy
2 Niteesh Pdy
3 Punith AP
4 Sidharth MP
5 Mantu Delhi

PL/SQL procedure successfully completed.
```

PL/SQL FUNCTION:

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value.

Syntax to create a function:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]

[(parameter_name [IN | OUT | IN OUT] type [, ...])]
```

```
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

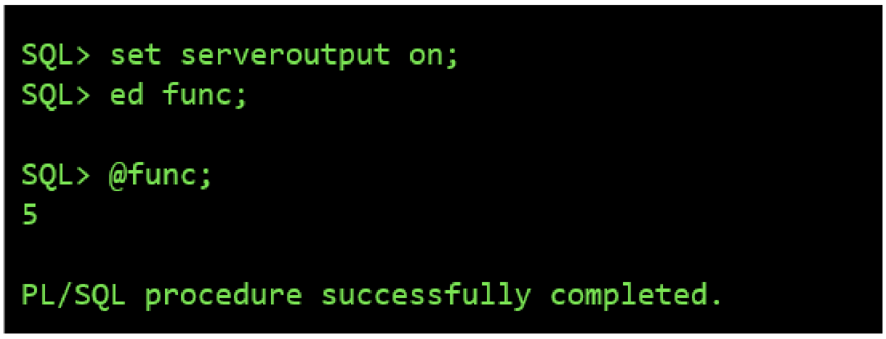
PROGRAM CODE:

```
DECLARE
n number;
t number;
FUNCTION func
RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total
  FROM employee;
   RETURN total;
END;
BEGIN
n:=2;
      t:=func();
      dbms_output.put_line(t);
```

END;

/

OUTPUT:



```
SQL> set serveroutput on;
SQL> ed func;

SQL> @func;
5

PL/SQL procedure successfully completed.
```

RESULT:

The queries for Procedure, Cursors and Functions were successfully executed and the output is noted.

EX NO.: 07 TRIGGERS

AIM:

To create and work with triggers.

PROCEDURE:

Step 1: Open Run SQL on Command line and connect to SQL

Step 2: Then work with database using SQL queries.

Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]
```

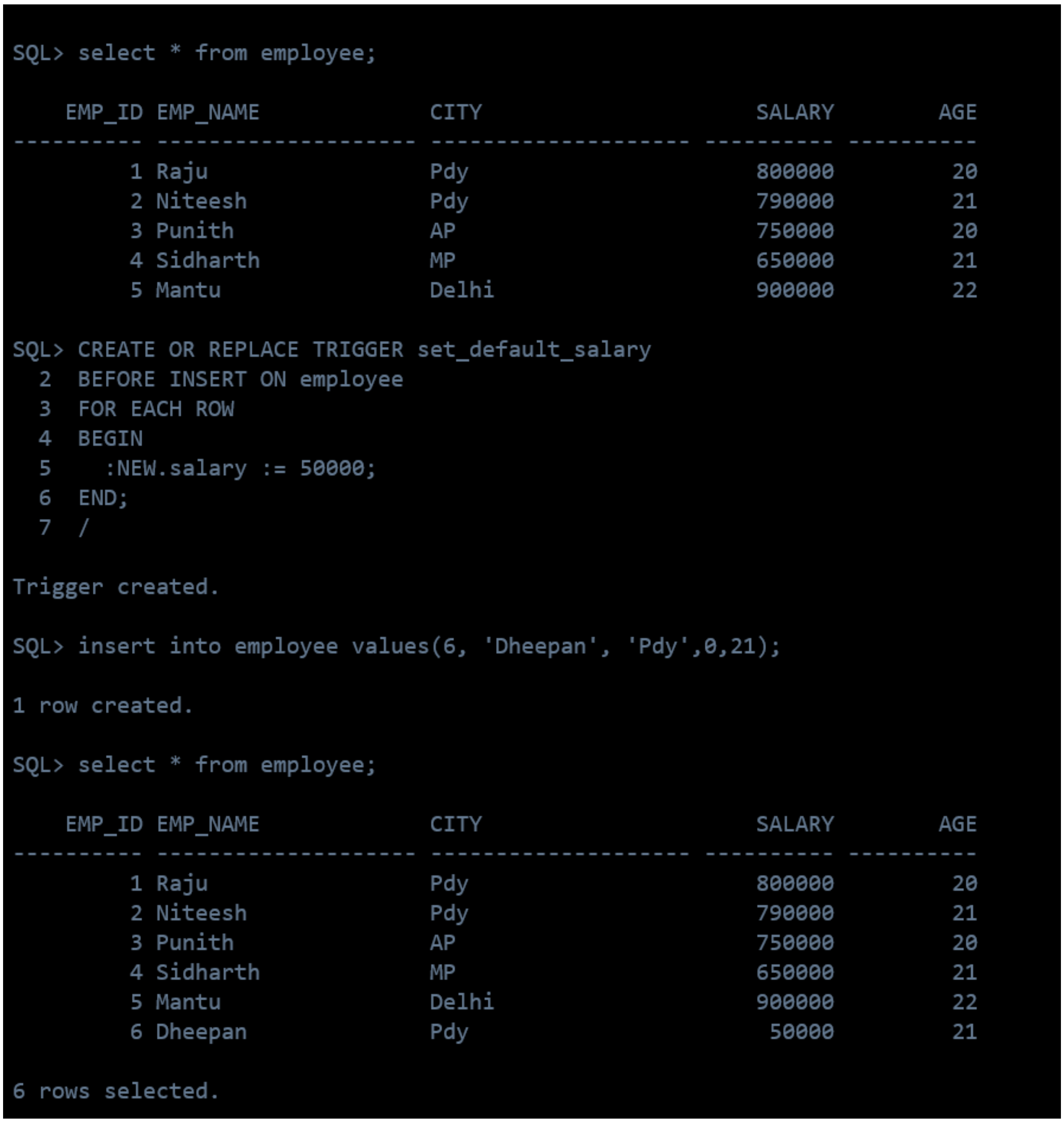
PROGRAM CODE:

```
SQL> CREATE OR REPLACE TRIGGER set_default_salary
  2  BEFORE INSERT ON employee
  3  FOR EACH ROW
  4  BEGIN
  5    :NEW.salary := 50000;
  6  END;
  7  /
```

This trigger program creates a trigger called **set_default_salary** that fires before an insert operation is performed on the **employee** table. The trigger sets the **salary** value of the new employee being inserted to 50000.

Note that the **BEFORE INSERT** clause specifies that the trigger should fire before the insert operation is performed, and the **FOR EACH ROW** clause specifies that the trigger should execute once for each row that is inserted. Finally, the **:NEW** keyword is used to reference the values being inserted into the table.

OUTPUT:



```
SQL> select * from employee;

    EMP_ID EMP_NAME             CITY                     SALARY        AGE
---------- -------------------- -------------------- ---------- ----------
         1 Raju                 Pdy                      800000         20
         2 Niteesh              Pdy                      790000         21
         3 Punith               AP                       750000         20
         4 Sidharth             MP                       650000         21
         5 Mantu                Delhi                    900000         22

SQL> CREATE OR REPLACE TRIGGER set_default_salary
  2  BEFORE INSERT ON employee
  3  FOR EACH ROW
  4  BEGIN
  5    :NEW.salary := 50000;
  6  END;
  7  /

Trigger created.

SQL> insert into employee values(6, 'Dheepan', 'Pdy',0,21);

1 row created.

SQL> select * from employee;

    EMP_ID EMP_NAME             CITY                     SALARY        AGE
---------- -------------------- -------------------- ---------- ----------
         1 Raju                 Pdy                      800000         20
         2 Niteesh              Pdy                      790000         21
         3 Punith               AP                       750000         20
         4 Sidharth             MP                       650000         21
         5 Mantu                Delhi                    900000         22
         6 Dheepan              Pdy                       50000         21

6 rows selected.
```

RESULT:

The queries for Triggers were successfully executed and the output is noted.

APPLICATION PROJECT

BLOOD DONATION APP

ABSTRACT:

- ❖ The blood donation website is a platform designed to encourage and facilitate blood donation.
- ❖ This website aims to educate the public about the importance of donating blood and to provide information about the donation process.
- ❖ It also serves as a means for individuals to register as blood donors and for blood banks to manage their inventory.
- ❖ The website features a user-friendly interface that allows donors to schedule appointments, track their donation history, and receive notifications about upcoming blood drives.
- ❖ Additionally, the website provides resources for individuals who may have questions or concerns about the donation process.
- ❖ Overall, the blood donation website is a valuable tool for promoting and supporting the lifesaving act of blood donation.

MODULES:

The project consists of 2 modules, which are

- ❖ Become a donor
- ❖ Need a blood donor

SOFTWARE REQUIREMENTS:

- ❖ Operating system – Windows 10
- ❖ Web Server : XAMPP [5.6.4]
- ❖ Database : MYSQL [5.0.21]
- ❖ Coding Language : Web Tech (HTML, CSS, Java Script, PHP)

LIMITATION OF EXISTING SYSTEM:

- ❖ Time - consuming
- ❖ Inefficient
- ❖ Limited reach

- Lack of privacy
- Safety concerns

PROGRAM CODE:

//SAVE_DATA

```
<?php

$name=$_POST['fullname'];

$number=$_POST['mobileno'];

$email=$_POST['emailid'];

$age=$_POST['age'];

$gender=$_POST['gender'];

$blood_group=$_POST['blood'];

$address=$_POST['address'];

$conn=mysqli_connect("localhost","root","","blood_donation") or die("Connection error");

$sql= "INSERT INTO
donor_details(donor_name,donor_number,donor_mail,donor_age,donor_gender,donor_bl
ood,donor_address)
values('{$name}','{$number}','{$email}','{$age}','{$gender}','{$blood_group}','{$address}')";

$result=mysqli_query($conn,$sql) or die("query unsuccessful.");

echo "<script>alert('Successfully Requested... ');</script>";

header("Location: http://localhost:7070/blood/index.php");

mysqli_close($conn);

 ?>
```

//INDEX.PHP

```
<html>

<head>

  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <meta name="description" content="">

  <meta name="author" content="">
```

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>

<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</head>

<body>

  <div id="page-container" style="margin-top:50px; position: relative;min-height: 84vh;">

    <div class="container">

    <div id="content-wrap" style="padding-bottom:50px;">

  <div class="row">

      <div class="col-lg-6">

          <h1 class="mt-4 mb-3">Need Blood</h1>

        </div>

  </div>

  <form name="needblood" action="" method="post">

  <div class="row">

  <div class="col-lg-4 mb-4">

  <div class="font-italic">Blood Group<span style="color:red">*</span></div>

  <div><select name="blood" class="form-control" required>

    <option value=""selected disabled>Select</option>

    <?php

      include 'conn.php';

      $sql= "select * from blood";

      $result=mysqli_query($conn,$sql) or die("query unsuccessful.");

    while($row=mysqli_fetch_assoc($result)){

     ?>

     <option value=" <?php echo $row['blood_id'] ?>"> <?php echo $row['blood_group'] ?>
</option>
```

```
  <?php } ?>
</select>
</div>
</div>
<div class="col-lg-4 mb-4">
<div class="font-italic">Reason, why do you need blood?<span
style="color:red">*</span></div>
<div><textarea class="form-control" name="address" required></textarea></div></div>
</div>
<div class="row">
<div class="col-lg-1">
<div><input type="submit" name="search" class="btn btn-primary" value="Search"
style="cursor:pointer"></div>
</div>
<div class="col-lg-4 mb-4">
                                        Willing To DonateBlood,
                                        <a href="donate_blood.php">
                                            click here to Donate
                                        </a>
                                    </div>
</div><div class="row">
<?php if(isset($_POST['search'])){
 $bg=$_POST['blood'];
 $sql= "select * from donor_details join blood where
donor_details.donor_blood=blood.blood_id AND donor_blood='{$bg}' order by rand() limit
5";
 $result=mysqli_query($conn,$sql) or die("query unsuccessful.");
  if(mysqli_num_rows($result)>0)   {
  while($row = mysqli_fetch_assoc($result)) {
   ?>
   <div class="col-lg-4 col-sm-6 portfolio-item" ><br>
```

```
      <div class="card" style="width:300px">
        <img class="card-img-top" src="blood_drop_logo.jpg" alt="Card image"
style="width:100%;height:300px">
        <div class="card-body">
          <h3 class="card-title"><?php echo $row['donor_name']; ?></h3>
          <p class="card-text">
            <b>Blood Group : </b> <b><?php echo $row['blood_group']; ?></b><br>
            <b>Mobile No. : </b> <?php echo $row['donor_number']; ?><br>
            <b>Gender : </b><?php echo $row['donor_gender']; ?><br>
            <b>Age : </b> <?php echo $row['donor_age']; ?><br>
            <b>Address : </b> <?php echo $row['donor_address']; ?><br>
          </p>
        </div>
      </div>
  </div>
  <?php
    }
  }
    else
    {
        echo '<div class="alert alert-danger">No Donor Found For your search Blood group
</div>';
    }
} ?>
</div>
</div>
</div>
</div>
</body>
</html>
```

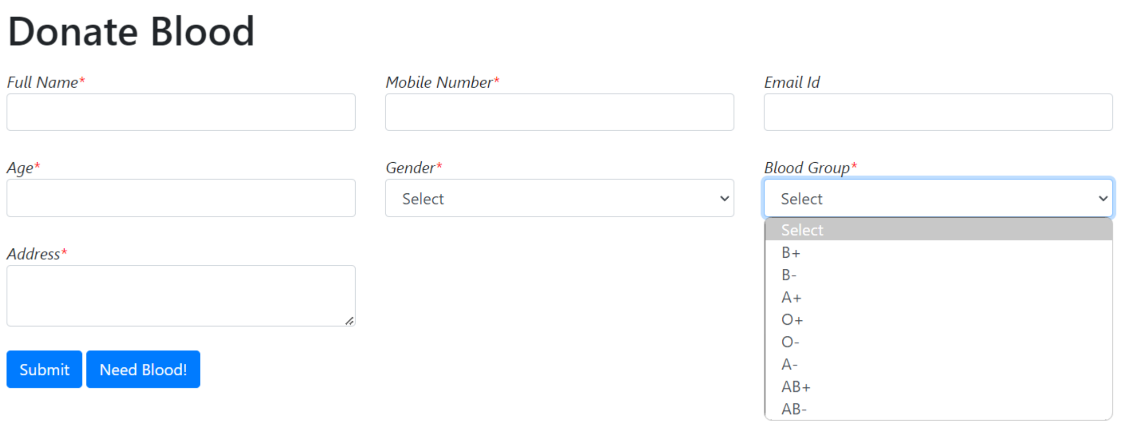
//BLOOD_DONATION

```
-- Database: `blood_donation`

-- Table structure for table `blood`

CREATE TABLE `blood` (

  `blood_id` int(11) NOT NULL,

  `blood_group` varchar(10) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;

-- Dumping data for table `blood`

INSERT INTO `blood` (`blood_id`, `blood_group`) VALUES

(1, 'B+'),

(2, 'B-'),

(3, 'A+'),

(4, 'O+'),

(5, 'O-'),

(6, 'A-'),

(7, 'AB+'),

(8, 'AB-');

-- Table structure for table `donor_details`

CREATE TABLE `donor_details` (

  `donor_id` int(11) NOT NULL,

  `donor_name` varchar(50) NOT NULL,

  `donor_number` varchar(10) NOT NULL,

  `donor_mail` varchar(50) DEFAULT NULL,

  `donor_age` int(60) NOT NULL,

  `donor_gender` varchar(10) NOT NULL,

  `donor_blood` varchar(10) NOT NULL,

  `donor_address` varchar(100) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;
```

```
-- Dumping data for table `donor_details`


INSERT INTO `donor_details` (`donor_id`, `donor_name`, `donor_number`, `donor_mail`,
`donor_age`, `donor_gender`, `donor_blood`, `donor_address`) VALUES
(7, 'gd', '9585718037', 'gd@gmail.com', 20, 'Male', ' 2', 'Pondy'),
(8, 'gs', '9585718037', 'gs@gmail.com', 18, 'Male', ' 2', 'Pdy'),
(9, 'gs', '9585718037', 'gs@gmail.com', 18, 'Male', ' 2', 'Pdy');
-- Indexes for dumped tables
-- Indexes for table `blood`
ALTER TABLE `blood`
  ADD PRIMARY KEY (`blood_id`);
-- Indexes for table `donor_details`
ALTER TABLE `donor_details`
  ADD PRIMARY KEY (`donor_id`);
-- AUTO_INCREMENT for dumped tables
-- AUTO_INCREMENT for table `blood`
ALTER TABLE `blood`
  MODIFY `blood_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;
-- AUTO_INCREMENT for table `donor_details`
ALTER TABLE `donor_details`
  MODIFY `donor_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10;
COMMIT;
```

OUTPUT:

1) DONATE BLOOD:



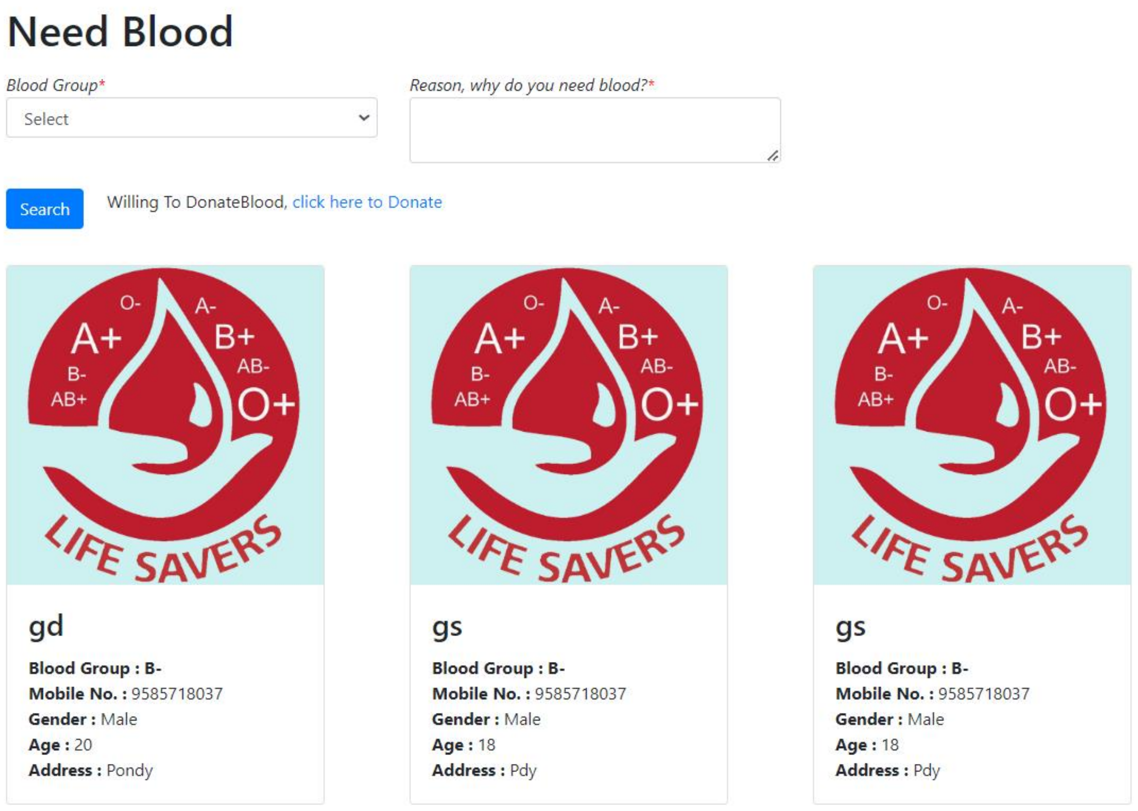
Donate Blood

Full Name*	Mobile Number*	Email Id
Age*	Gender* Select	Blood Group* Select
Address*		

Submit | Need Blood!

Select
B+
B-
A+
O+
O-
A-
AB+
AB-

2) NEED BLOOD:



Need Blood

Blood Group* Select

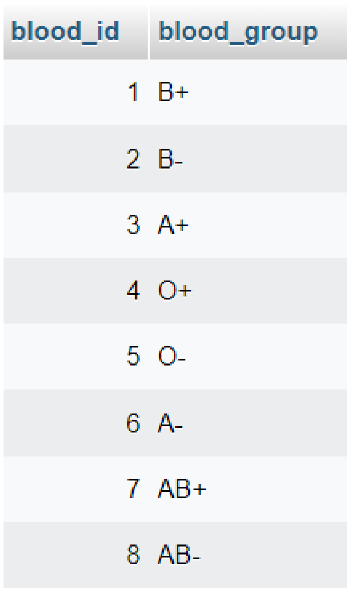
Reason, why do you need blood?*

Search   Willing To DonateBlood, click here to Donate

gd	gs	gs
Blood Group : B-	**Blood Group : B-**	**Blood Group : B-**
Mobile No. : 9585718037	**Mobile No. :** 9585718037	**Mobile No. :** 9585718037
Gender : Male	**Gender :** Male	**Gender :** Male
Age : 20	**Age :** 18	**Age :** 18
Address : Pondy	**Address :** Pdy	**Address :** Pdy

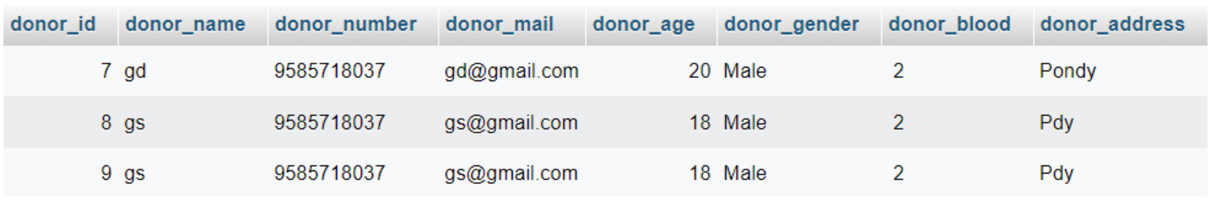
DATABASE OUTPUT:

BLOOD GROUP TABLE:



blood_id	blood_group
1	B+
2	B-
3	A+
4	O+
5	O-
6	A-
7	AB+
8	AB-

BLOOD DONOR DETAILS:



donor_id	donor_name	donor_number	donor_mail	donor_age	donor_gender	donor_blood	donor_address
7	gd	9585718037	gd@gmail.com	20	Male	2	Pondy
8	gs	9585718037	gs@gmail.com	18	Male	2	Pdy
9	gs	9585718037	gs@gmail.com	18	Male	2	Pdy