Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Since 1962

Department of Artificial Intelligence and Data Science

Database Management Systems Labs Journal

Roll No.	47
Name	Yash Sarang
Class	D6AD
Subject	DBMS Lab
Grade:	

1. Lab Objectives:

	Description
1	To learn the basic concepts of Object-oriented Programming
2	To study Java Programming Language.
3	To study various concepts of Java Programming like multithreading, exception Handling, packages etc
4	To explain components of GUI based programming

2. Lab Outcome:

LO	Description
LO 1	Design ER /EER diagrams and convert to relational models for the real world application.
LO 2	Apply DDL, DML, DCL and TCL commands
LO 3	Write simple and complex queries
LO 4	Use PL / SQL Constructs.
LO 5	Demonstrate the concept of concurrent transactions execution and frontend-backend connectivity

3. LO/PO Mapping

LO	PO1	PO2	PO3	PO4	PO5	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
LO1	**1**	**3**	**2**	**1**	**1**	**1**	**2**	**1**	**1**	**2**	**2**	**1**
LO2	**-**	**3**	**3**	**2**	**2**	**1**	**2**	**1**	**2**	**3**	**2**	**2**
LO3	**-**	**3**	**3**	**2**	**2**	**1**	**2**	**1**	**2**	**3**	**1**	**3**
LO4	**-**	**2**	**2**	**1**	**2**	**-**	**1**	**1**	**-**	**1**	**-**	**2**
LO5	**1**	**3**	**2**	**2**	**2**	**1**	**2**	**1**	**1**	**3**	**1**	**2**

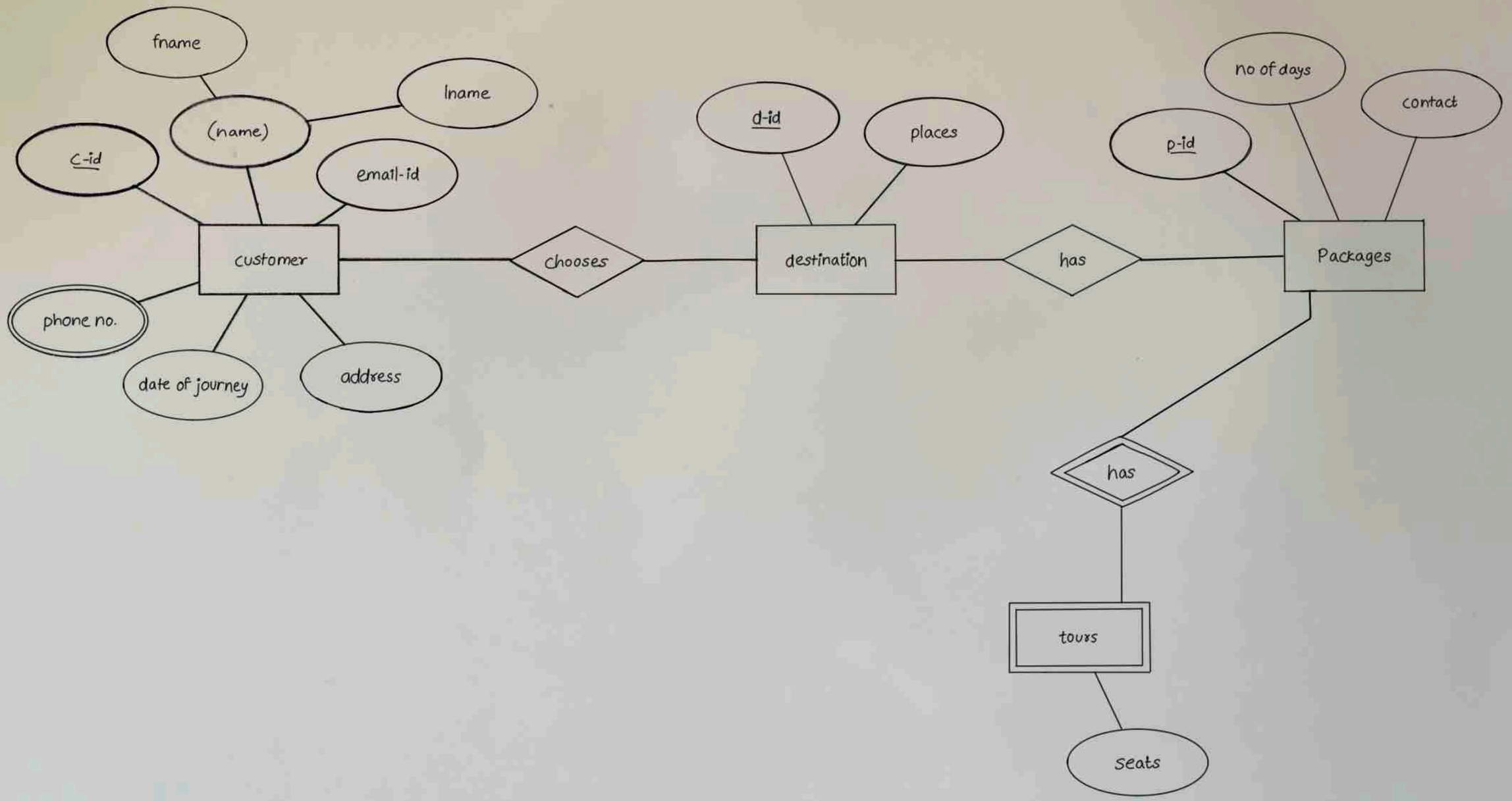
DBMS Lab

Name: Yash Sarang.

INDEX

List of Experiments:

Sr No	List of Experiments	LO
1	Identify the case study and detail **statement of problem** and draw **Entity-Relationship (ER) / Extended Entity-Relationship (EER)** Model Using Draw.io.	LO1
2.	Mapping **ER/EER to the Relational schema model and creating a schema diagram for your system.**	LO1, LO2
3	Create a database using **Data Definition Language (DDL)** and apply required **Integrity Constraints** for the specified system.	LO2, LO3
4	Populate database using **DML Commands** for your specified System.	LO2, LO3
5	Perform **Simple queries, string manipulation operations.**	LO2, LO3
6	Write **Nested queries using (in, not in some any exist nt exits, with clause)**	LO2, LO3
7	Implement various types of **Joins and Views.**	LO3
8	Demonstrate **DCL** and **TCL** commands.	LO2
9	Implementation of **Functions and Stored Procedure** in PL-SQL.	LO4
10	Implement different types **of triggers.**	LO4
11	Implementation and demonstration of **Transaction and Concurrency control** techniques using locks.	LO2, LO4, LO5
12	Implement Database Connectivity (JDBC.ODBC)	LO2, LO3, LO5


fname
lname
(name)
C-id
email-id
customer
phone no.
date of journey
address
chooses
d-id
places
destination
has
no of days
contact
p-id
Packages
has
tours
seats

ER TO RELATIONAL SCHEMA
for
TOURISM MANAGEMENT SYSTEM

GROUP NO. 5

YASH SARANG

SHREYA SINGH

PARTH SURYAVANSHI

SURABHI TAMBE

Index. Pg.

****

- Table creation using DDL commands:

1. Creating tablespace, creating a user, granting privileges to the user.

```
SQL*Plus: Release 11.2.0.1.0 Production on Tue Mar 15 23:10:48 2022

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Enter user-name: system
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> conn system
Enter password:
Connected.
SQL> Create TABLESPACE tbs_Tourism
  2  DATAFILE 'C:\DBMS\database\ermodel.dbf' SIZE 256m
  3  AUTOEXTEND ON
  4  MAXSIZE 2046m
  5  EXTENT MANAGEMENT LOCAL
  6  ONLINE;

Tablespace created.

SQL> CREATE USER NOOB IDENTIFIED BY PRO;

User created.

SQL> GRANT DBA TO NOOB;

Grant succeeded.
```

---

2. Creating destination table:

```
SQL> CREATE TABLE destination(did int  PRIMARY KEY, places varchar(255));

Table created.

SQL> desc destination
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DID                                       NOT NULL NUMBER(38)
 PLACES                                             VARCHAR2(255)
```

---

3. Creating customer table:

```
SQL> CREATE TABLE customer(cid int PRIMARY KEY,fname varchar(255),lname varchar(255),email varchar(255),address varchar(255),dofjourney date);

Table created.
```

```
SQL> ALTER TABLE customer
  2   ADD FOREIGN KEY (did) REFERENCES destination(did);

Table altered.

SQL> desc customer
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CID                                       NOT NULL NUMBER(38)
 FNAME                                              VARCHAR2(255)
 LNAME                                              VARCHAR2(255)
 EMAIL                                              VARCHAR2(255)
 ADDRESS                                            VARCHAR2(255)
 DOFJOURNEY                                         DATE
 DID                                                NUMBER(38)
```

---

4. Creating package table:

```
SQL> CREATE TABLE package(pid int  PRIMARY KEY, cost int, noofdays int,packagetype varchar(255), contact int,discount int);

Table created.
```

```
SQL> ALTER TABLE package
  2  ADD did int;

Table altered.

SQL> ALTER TABLE package
  2  ADD FOREIGN KEY (did) REFERENCES destination(did);
```

```
SQL> Alter table package
  2  drop column packagetype;

Table altered.

SQL> Alter table package
  2  drop column discount;
```

```
SQL> desc package
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PID                                       NOT NULL NUMBER(38)
 COST                                               NUMBER(38)
 NOOFDAYS                                           NUMBER(38)
 CONTACT                                            NUMBER(38)
 DID                                                NUMBER(38)
```

---

5. Creating tours table:

```
SQL> CREATE TABLE tours(tid int  PRIMARY KEY, seats int, types varchar(255), pid int);

Table created.
```

```
SQL> ALTER TABLE tours
  2  ADD FOREIGN KEY (pid) REFERENCES package(pid);
```

```
SQL> ALTER TABLE tours
  2  drop column types;
```

```
SQL> desc tours
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TID                                       NOT NULL NUMBER(38)
 SEATS                                              NUMBER(38)
 PID                                                NUMBER(38)
```

---

6. Creating phoneno table:

```
SQL> CREATE TABLE phoneno(phoneno int, places varchar(255), cid int);

Table created.
```

```
SQL> ALTER TABLE phoneno
  2  ADD FOREIGN KEY (cid) REFERENCES customer(cid);

Table altered.

SQL> desc phoneno
 Name                                       Null?    Type
 ----------------------------------------- -------- ----------------------------
 PHONENO                                            NUMBER(38)
 PLACES                                             VARCHAR2(255)
 CID                                                NUMBER(38)
```

****

- # Inserting values using DML commands:

1. Customer table:

```
SQL>  select * from customer;

  CID  FNAME                LNAME                EMAIL                          ADDRESS              DOFJOURNEY                      DID
-----  ------------------   ------------------   ----------------------------   ------------------   -----------------------------  -----
  239  Phil                 Dunphy               phildunphy09@yahoo.com         Manhattan            03-DEC-19                       3472
  240  Michael              Scott                michaels12@yahoo.com           Scranton             15-JAN-20                        987
  241  Cameron              Oldren               camoldren55@yahoo.com          New York             23-JAN-20                       6512
  242  Ted                  Mosby                tedmosby11@yahoo.com           Michigan             12-MAY-19                       4389
```

---

2. Destination table:

```
SQL> Insert into destination values (6512, 'Tokyo');

1 row created.

SQL> select * from destination;

  DID  PLACES
-----  ----------------------------
  987  Hawai
 3472  Los Angeles
 4389  Australia
 6512  Tokyo
```

---

3. Package table:

```
SQL> select* from package;

  PID       COST  NOOFDAYS      CONTACT         DID
-----  --------  --------  -----------  ----------
  674    366000         7   5058969055         987
  254    859000        12   5827777807        3472
  711    250000         6   5056725262        4389
  520    678000         4   5824002778        6512
```

---

4. Tours table:

```
SQL> select * from tours;

  TID  SEATS    PID
-----  -----  -----
 1010      3    254
 1765      4    674
 1221      2    711
 1051      3    520
```

5. Phoneno table:

```
SQL> select * from phoneno;

   PHONENO  PLACES                          CID
----------  ------------------------------  -----
2054678508  Tokyo                             241
3177510487  Hawai                             240
4255546887  Australia                         242
3054341609  Los Angeles                       239
```

****

PRACTICAL 3

Aim: Create a database using Data Definition Language (DDL) and apply integrity ~~constants~~ constraints for the specified system.

Theory: DDL Queries are used to:
a) To create structure of the database
b) To change or modify the structure
c) To delete a table from database
d) To change the name of a table
e) To delete rows of a table.

* Create Query

Syntax: create table <name>
(column1 , datatype (size),
column2 , datatype (size),
: :);

Example- for student_info (rollno, name, DOB, address, branch)
create table query would be.
create table student_info (rollno number (4),
name varchar (30),
DOB DATE,
address varchar (40),
branch varchar (4));

* Alter Query. - used to change the structure of an existing query / table.

Type - a) Alter add
b) Alter modify

- Alter add is used when we need to add a new column to an existing table. We can also add constraints.

① to add column: alter table <name>
add column name datatype(size);

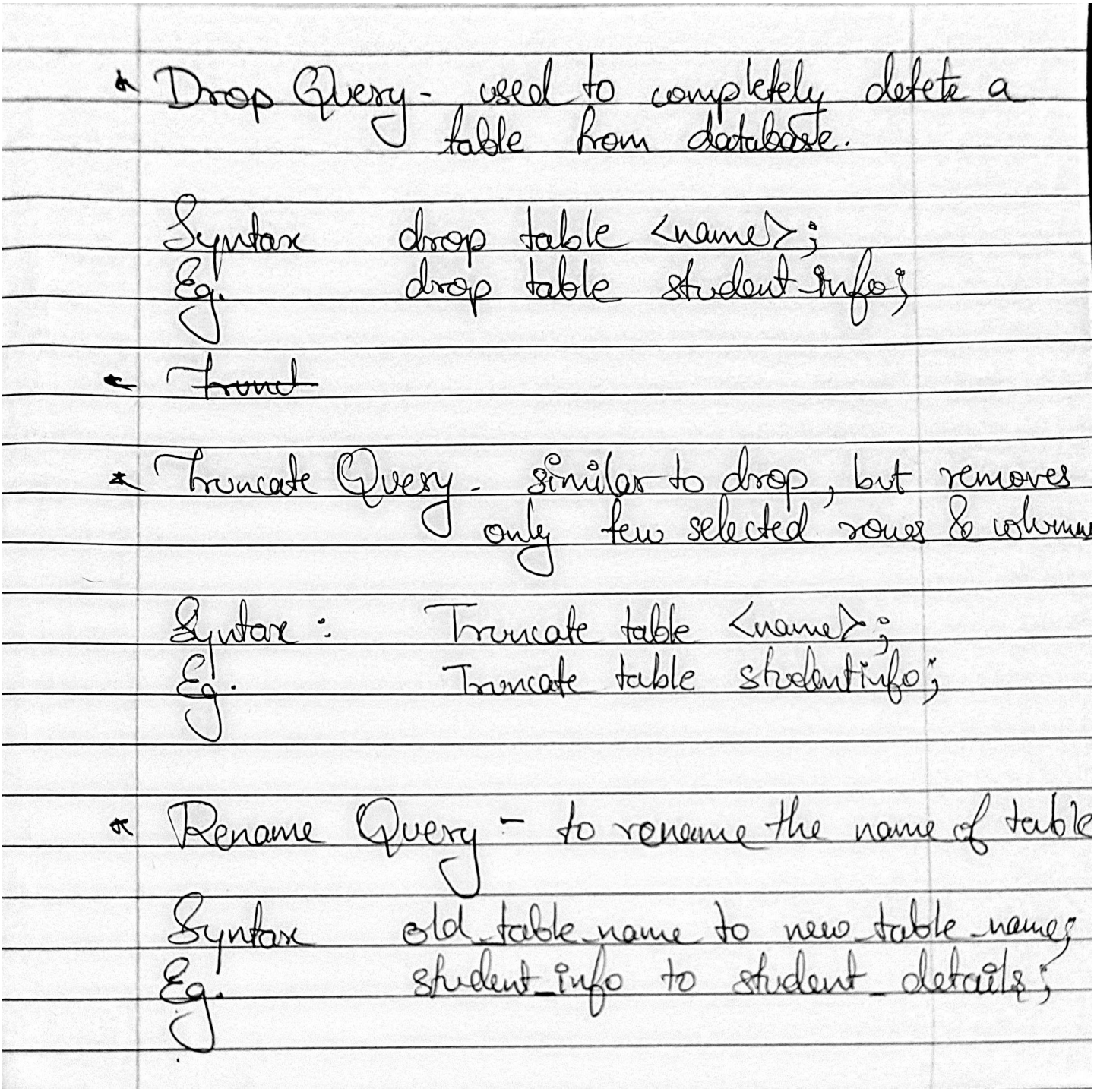
② to add constraint: alter table <name>
add constraint name type(column);

Eg. alter table student_info
add email varchar(20);

alter table student_info
add constraint student_key primary key (Rollno);

- Alter modify is used when the data type or size of an existing table is to be modified. This query can also be used to add a not null constraint.

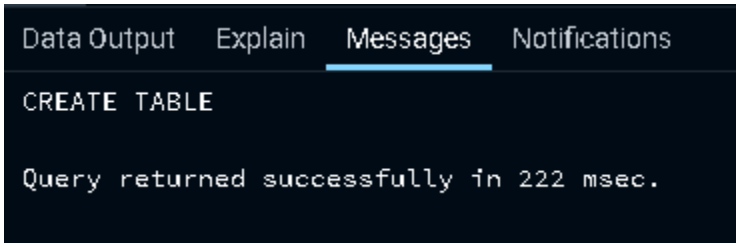
Syntax: alter table <name>
modify column_name datatype(size);



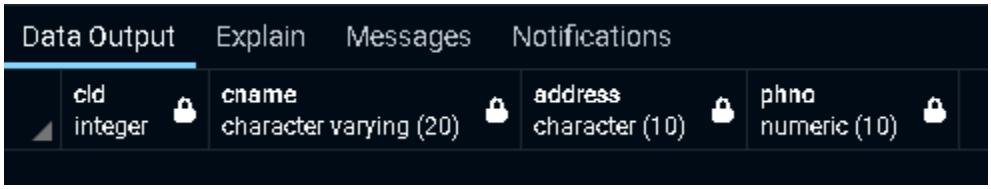
DDL Queries with Output:

1. Customer Table:

a. create table customer (cid int, cname varchar (20),address char(10));

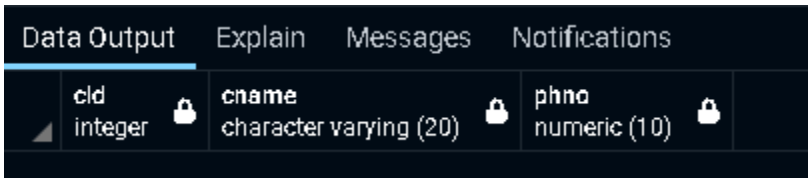


b. alter table customer add phno numeric (10);

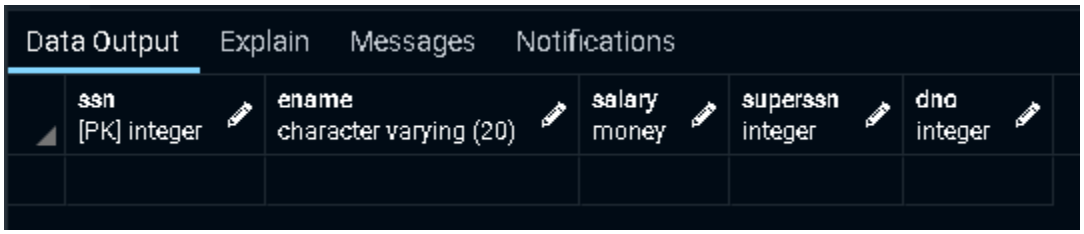


c. alter table customer modify address varchar (20);
alter table customer alter column address type varchar;

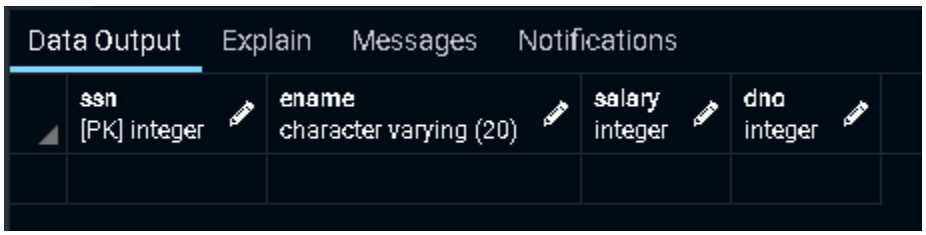
d. alter table customer drop column address;



2. Employee Table
a. create table employee (ssn int primary key, ename varchar(20) not null, salary int,dno int,Foreign key(dno) references dept(dno));

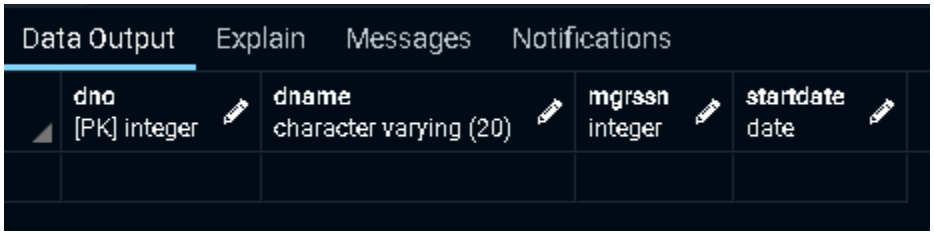


b. alter table employee add constraint fk_dno foreign key(dno) references dept1(dno);



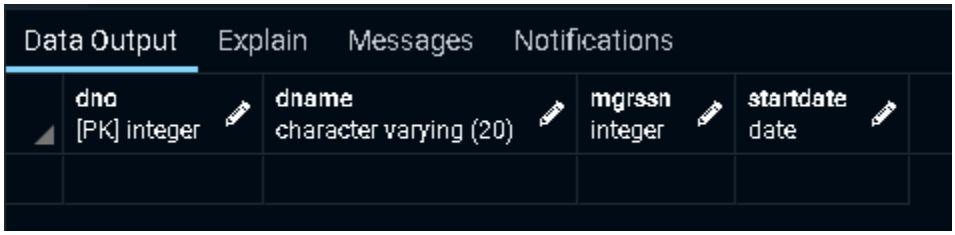
3. Department Table:

a. create table dept1(dno int primary key,dname varchar(20),mgrssn int references employee,startdate datetime);



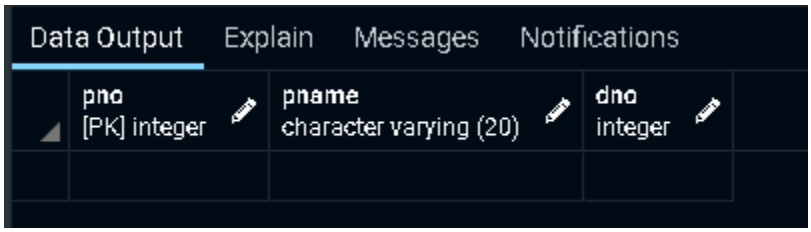
4. Deptlocation Table:

a. create table deptloc(dno int,dloc varchar(20),primary key(dno,dloc),foreign key(dno) references dept1(dno));



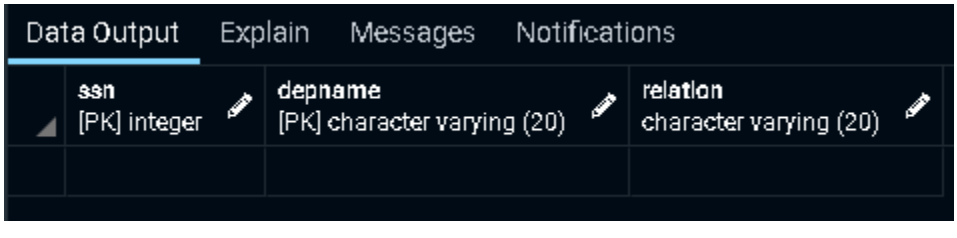
5. Project Table:

a. create table project (pno int primary key,pname varchar(20),dno int references dept1(dno));



6. Table Dependent:

a. create table dependent(ssn int,depname varchar(20),relation varchar(20), primary key(ssn,depname),foreign key(ssn) references employee(ssn));



Conclusion :- Thus we have learnt about various DDL Queries (Create, Alter, Drop, Truncate, Rename) in detail with proper Syntax and examples.

Aim: To study Data Manipulation Language commands (DML)

Theory: DML is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database.

- INSERT

  used to add records to an existing table.

  Syntax: insert into table tablename values (values);

  Example: INSERT INTO employee values ('bart', 'simpson', 12345, $45000)

- SELECT

  allows database uses to retrieve specific information from an operational database

  Syntax: Select <attribute list> from <list of table>;

  Example: SELECT * FROM Employee.

- UPDATE

  used to modify information contained within a table, either bulk or individually.

  Syntax: UPDATE tablename SET condition;

  Example: UPDATE employee SET salary = salary * 1.03;

- DELETE

  used to remove any desirable records from a functioning database.

  Syntax: DELETE FROM tablename where condition;

  Example: DELETE FROM employee WHERE employee_id = 12345;

Conclusion: Thus, we have implement different DML commands successfully.

DML Queries with Output:

```
SQL> set linesize 200;
SQL> select * from employee;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   50000        100         10
       101 Jonas                     60000        101         11
       102 SCARLET                   80000        102         12
       103 BLAIR                     45000        103         13
       104 CHARLES                   40000        100         10

SQL> select * from dept;

       DNO DNAME                STARTDATE     MGRSSN
---------- -------------------- --------- ----------
        10 FINANCE              12-NOV-21        100
        11 AUDIT                12-OCT-21        101
        12 MARKETING            01-NOV-21        102
        13 PRODUCTION           09-OCT-21        103

SQL> select * from customer;

no rows selected

SQL> insert into customer values(1,'Bart','Starbuck, Minnesota');
```

```
SQL> insert into customer values(1,'Bart','Starbuck, Minnesota');

1 row created.

SQL> insert into customer values(2,'Homer','Hammond, Louisiana');

1 row created.

SQL> insert into customer values(3,'Marge','Lockport, New York');

1 row created.

SQL> insert into customer values(4,'Lisa','Starbuck, Minnesota');

1 row created.

SQL> select * from customer;

       CID CNAME                ADDRESS
---------- -------------------- --------------------------------------------------
         1 Bart                 Starbuck, Minnesota
         2 Homer                Hammond, Louisiana
         3 Marge                Lockport, New York
         4 Lisa                 Starbuck, Minnesota

SQL> insert into dependent values(100,'FINANCE','X');

1 row created.

SQL> select * from dependent;

       SSN DEPNAME              RELATION
---------- -------------------- --------------------
       100 FINANCE              X

SQL> insert into dependent values(101,'FINANCE','X');

1 row created.
```

```
SQL> insert into dependent values(101,'FINANCE','X');

1 row created.

SQL> insert into dependent values(102,'AUDIT','Y');

1 row created.

SQL> insert into dependent values(103,'PRODUCTION','Y');

1 row created.

SQL> select * from dependent;

       SSN DEPNAME              RELATION
---------- -------------------- --------------------
       100 FINANCE              X
       101 FINANCE              X
       102 AUDIT                Y
       103 PRODUCTION           Y

SQL> insert into deptloc values(10,'SYDNEY');

1 row created.

SQL> insert into deptloc values(11,'MELBOURNE');

1 row created.

SQL> insert into deptloc values(12,'BRISBANE');

1 row created.

SQL> insert into deptloc values(12,'PERTH');

1 row created.

SQL> DELETE FROM deptloc WHERE DLOC='PERTH';
```

```
SQL> insert into workson values(100,20,4);

1 row created.

SQL> insert into workson values(101,21,8);

1 row created.

SQL> insert into workson values(102,22,6);

1 row created.

SQL> insert into workson values(103,23,5);

1 row created.

SQL> insert into workson values(104,24,7);

1 row created.

SQL> SELECT * FROM project;

       PNO PNAME                          DNO
---------- -------------------- ----------
        20 Omega                          10
        21 Nitro                          10
        22 Origin                         11
        23 Project X                      12
        24 Quadro                         13

SQL> SELECT * FROM workson;

       SSN        PNO    NOOFHRS
---------- ---------- ----------
       100         20          4
       101         21          8
       102         22          6
       103         23          5
       104         24          7
```

```
SQL> insert into project values(20,'Omega',10);

1 row created.

SQL> insert into project values(21,'Nitro',10);

1 row created.

SQL> insert into project values(22,'Origin',11);

1 row created.

SQL> insert into project values(23,'Project X',12);

1 row created.

SQL> insert into project values(24,'Quadro',13);

1 row created.

SQL> insert into workson values(100,20,4);

1 row created.

SQL> insert into workson values(101,21,8);

1 row created.

SQL> insert into workson values(102,22,6);

1 row created.

SQL> insert into workson values(103,23,5);

1 row created.

SQL> insert into workson values(104,24,7);

1 row created.
```

```
SQL> set linesize 200;
SQL> select * from employee;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   50000        100         10
       101 Jonas                     60000        101         11
       102 SCARLET                   80000        102         12
       103 BLAIR                     45000        103         13
       104 CHARLES                   40000        100         10

SQL> select * from dept;

       DNO DNAME                STARTDATE     MGRSSN
---------- -------------------- --------- ----------
        10 FINANCE              12-NOV-21        100
        11 AUDIT                12-OCT-21        101
        12 MARKETING            01-NOV-21        102
        13 PRODUCTION           09-OCT-21        103

SQL> select * from customer;

       CID CNAME                ADDRESS
---------- -------------------- --------------------------------------------------
         1 Bart                 Starbuck, Minnesota
         2 Homer                Hammond, Louisiana
         3 Marge                Lockport, New York
         4 Lisa                 Starbuck, Minnesota
```

```
SQL> select * from dependent;

       SSN DEPNAME                RELATION
---------- -------------------- --------------------
       100 FINANCE                X
       101 FINANCE                X
       102 AUDIT                  Y
       103 PRODUCTION             Y

SQL> select * from deptloc;

       DNO DLOC
---------- --------------------
        10 SYDNEY
        11 MELBOURNE
        12 BRISBANE
        13 PERTH

SQL> select * from project;

       PNO PNAME                          DNO
---------- -------------------- ----------
        20 Omega                          10
        21 Nitro                          10
        22 Origin                         11
        23 Project X                      12
        24 Quadro                         13

SQL> select * from workson;

       SSN        PNO    NOOFHRS
---------- ---------- ----------
       100         20          4
       101         21          8
       102         22          6
       103         23          5
       104         24          7
```

```
SQL> SELECT SSN,ENAME,SALARY FROM EMPLOYEE;

       SSN ENAME                    SALARY
---------- -------------------- ----------
       100 William                   50000
       101 Jonas                     60000
       102 SCARLET                   80000
       103 BLAIR                     45000
       104 CHARLES                   40000

SQL> SELECT * FROM EMPLOYEE WHERE SALARY>=50000;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   50000        100         10
       101 Jonas                     60000        101         11
       102 SCARLET                   80000        102         12

SQL> SELECT * FROM PROJECT WHERE DNO=10;

       PNO PNAME                       DNO
---------- -------------------- ----------
        20 Omega                       10
        21 Nitro                       10

SQL> SELECT * FROM PROJECT WHERE DNO=10 AND PNAME=Nitro;
SELECT * FROM PROJECT WHERE DNO=10 AND PNAME=Nitro
                                            *
ERROR at line 1:
ORA-00904: "NITRO": invalid identifier


SQL> SELECT * FROM PROJECT WHERE DNO=10 AND PNAME='Nitro';

       PNO PNAME                       DNO
---------- -------------------- ----------
        21 Nitro                       10
```

```
        21 Nitro                          10

SQL> SELECT * FROM EMPLOYEE WHERE SALARY BETWEEN 40000 AND 60000;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   50000        100         10
       101 Jonas                     60000        101         11
       103 BLAIR                     45000        103         13
       104 CHARLES                   40000        100         10

SQL>
SQL> -- UPDATE QUERY
SQL> UPDATE employee SET SALARY = SALARY * 1.03;

5 rows updated.

SQL> SELECT * FROM EMPLOYEE;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
       104 CHARLES                   41200        100         10

SQL> UPDATE employee SET SALARY = SALARY + 9000
  2  WHERE SSN = CHARLES;
WHERE SSN = CHARLES
            *
ERROR at line 2:
ORA-00904: "CHARLES": invalid identifier


SQL> UPDATE employee SET SALARY = SALARY + 9000
  2  WHERE SSN = 104;

1 row updated.
```

```
SQL>
SQL> -- UPDATE QUERY
SQL> UPDATE employee SET SALARY = SALARY * 1.03;

5 rows updated.

SQL> SELECT * FROM EMPLOYEE;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
       104 CHARLES                   41200        100         10

SQL> UPDATE employee SET SALARY = SALARY + 9000
  2  WHERE SSN = CHARLES;
WHERE SSN = CHARLES
            *
ERROR at line 2:
ORA-00904: "CHARLES": invalid identifier


SQL> UPDATE employee SET SALARY = SALARY + 9000
  2  WHERE SSN = 104;

1 row updated.

SQL> SELECT * FROM EMPLOYEE;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
       104 CHARLES                   50200        100         10
```

```
SQL> UPDATE CUSTOMER SET CNAME = 'CHERYL' WHERE CID = 3;

1 row updated.

SQL> SELECT * FROM customer;

       CID CNAME                ADDRESS
---------- -------------------- --------------------------------------------------
         1 Bart                 Starbuck, Minnesota
         2 Homer                Hammond, Louisiana
         3 CHERYL               Lockport, New York
         4 Lisa                 Starbuck, Minnesota

SQL> DELETE FROM PROJECT WHERE PNO = 20;

1 row deleted.

SQL> SELECT * FROM PROJECT;

       PNO PNAME                      DNO
---------- -------------------- ----------
        21 Nitro                        10
        22 Origin                       11
        23 Project X                    12
        24 Quadro                       13

SQL> INSERT INTO PROJECT VALUES(20,'Omega',10);

1 row created.

SQL> Delete from project where dno=(select dno from dept where dname='FINANCE');

2 rows deleted.

SQL> select * from project;

       PNO PNAME                      DNO
---------- -------------------- ----------
        22 Origin                       11
```

```
SQL> select * from project;

       PNO PNAME                          DNO
---------- -------------------- ----------
        22 Origin                         11
        23 Project X                      12
        24 Quadro                         13

SQL> INSERT INTO PROJECT VALUES(20,'Omega',10);

1 row created.

SQL> INSERT INTO PROJECT VALUES(21,'Nitro',11);

1 row created.

SQL> select * from project;

       PNO PNAME                          DNO
---------- -------------------- ----------
        22 Origin                         11
        23 Project X                      12
        24 Quadro                         13
        20 Omega                          10
        21 Nitro                          11

SQL> _
```

Conclusion: Thus we have successfully learned and implemented different DML commands in SQL.

EXPERIMENT 5

Aim: Perform Simple Queries, String Manipulation operations.

Objective: To understand simple queries and string manipulation operations

Description:

Where clause is used to select rows, eg.
select column1,... from tablename where condition.

- five basic searching ~~conductors~~ conditions.
  1) Comparision. (=, <, >, <=, >=, <>).
  2) Range (between, not between)
  3) set membership (IN / Not IN)
  4) Pattern match (Like / Not Like)
  5) NULL (ISNULL / NOT NULL)

String manipulation operators

- length (column_name) - returns length in no. of chars.
- lower (column_name) - converts all to lowercase
- upper (") - vice versa.
- trim both (both '_' from 'string') - strips starting and trailing characters.

- trimleading trim (leading '_' from 'string') - strips from front
- trim trailing trim (trailing '_' from 'string') - strips from trail.
- replace (columnname, 'from', 'to') - replaces specified string with another.
- substring (columnname from __ for __) - create a substring from start & end points.

  ↑ start ↑ end

  substring (columnname, from __) - create a substring from starting point given till the end of the whole.

SQL Queries:

```
SQL> select * from employee where salary BETWEEN 46350 and 82400;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
       104 CHARLES                   50200        100         10

SQL> select * from employee where salary BETWEEN 46350 and 51500;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       103 BLAIR                     46350        103         13
       104 CHARLES                   50200        100         10

SQL> select * from employee where DNO in '10';

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       104 CHARLES                   50200        100         10

SQL> select * from employee where DNO not in '10';

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
```

```
SQL> select * from employee where ENAME like '%as';

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       101 Jonas                     61800        101         11

SQL> select * from employee where ENAME like '%es';

no rows selected

SQL> select * from employee where ENAME like '%ES';

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       104 CHARLES                   50200        100         10

SQL> select * from employee where ENAME like 'BL%';

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       103 BLAIR                     46350        103         13
```

```
SQL> select * from employee where ENAME not like 'am%';

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
       104 CHARLES                   50200        100         10

SQL> select * from employee where ENAME is null;

no rows selected

SQL> select * from employee where ENAME IS NULL;

no rows selected

SQL> select * from employee where ENAME IS NOT NULL;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
       104 CHARLES                   50200        100         10
```

```
SQL> select * from customer;

       CID CNAME                ADDRESS
---------- -------------------- --------------------------------------------------
         1 Bart                 Starbuck, Minnesota
         2 Homer                Hammond, Louisiana
         3 Marge                Lockport, New York
         4 Lisa                 Starbuck, Minnesota

SQL> select CNAME from customer where address IS NULL;

no rows selected

SQL> select CNAME from customer where address IS NOT NULL;

CNAME
--------------------
Bart
Homer
Marge
Lisa

SQL> select * from customer where CNAME NOT IN('Starbuck, Minnesota');

       CID CNAME                ADDRESS
---------- -------------------- --------------------------------------------------
         1 Bart                 Starbuck, Minnesota
         2 Homer                Hammond, Louisiana
         3 Marge                Lockport, New York
         4 Lisa                 Starbuck, Minnesota

SQL> select * from customer where CNAME IN('Starbuck, Minnesota');

no rows selected
```

```
SQL> select * from customer where CNAME NOT IN('Starbuck, Minnesota');

       CID CNAME                ADDRESS
---------- -------------------- --------------------------------------------------
         1 Bart                 Starbuck, Minnesota
         2 Homer                Hammond, Louisiana
         3 Marge                Lockport, New York
         4 Lisa                 Starbuck, Minnesota

SQL> select * from customer where CNAME IN('Starbuck, Minnesota');

no rows selected

SQL> select * from customer where CNAME IN('Bart');

       CID CNAME                ADDRESS
---------- -------------------- --------------------------------------------------
         1 Bart                 Starbuck, Minnesota

SQL> select count(*) as 'cl' from CNAME group by CID having count(*)>1;
select count(*) as 'cl' from CNAME group by CID having count(*)>1
                   *
ERROR at line 1:
ORA-00923: FROM keyword not found where expected


SQL> desc employee;
 Name                                                                                                      Null?    Type
 --------------------------------------------------------------------------------------------------------- -------- ------------
----------------------------
 SSN                                                                                                       NOT NULL NUMBER(38)
 ENAME                                                                                                     NOT NULL VARCHAR2(20)
 SALARY                                                                                                             NUMBER(38)
 SUPERSSN                                                                                                           NUMBER(38)
 DNO                                                                                                                NUMBER(38)
```

```
SQL> select * from dept;

       DNO DNAME                STARTDATE      MGRSSN
---------- -------------------- --------- ----------
        10 FINANCE              12-NOV-21         100
        11 AUDIT                12-OCT-21         101
        12 MARKETING            01-NOV-21         102
        13 PRODUCTION           09-OCT-21         103

SQL>
```

Conclusion:

Thus, we have successfully performed, learned and implemented simple queries on strings and string manipulation.

EXPERIMENT 6

Aim: Perform Simple Queries, String manipulation operations.

Objective:- To understand simple queries & string manipulation operations.

Description:

SQL where clause is used to select rows. Satisfying given predicate.

```
SELECT Column 1...
FROM TABLE
WHERE Condition)
```

WHERE Clause consist of 5 search conditions

COMPARE → (<, >, =, <=, >=)
Range → (Between / Not Between)
Set membership → (IN/NOT IN)
Pattern Match → (LIKE/NOT LIKE)
NULL → (IS NULL / IS NOT NULL)

Simple Queries:

Between → Checks value in a range
IN → Tests whether a data value matches one of a list value.
LIKE → % represents any sequence of zero or more characters. _ (Under score) represents a single character.

String Manipulation Soft Operations.

1) ASCII → converts single character string to an ASCII value.
2) BIT LENGTH → Returns the length in bits.
3) CHAR → Convert numeric values between 0 & 255 to character values.
4) CHAR-LENGTH → returns length in no. of chars.
5) CONCAT → CONCATENATE two char strings.
6) INSERT → Insert specific character string into specified location.
7) UPPER → Converts a string to uppercase.
8) POSITION → returns numeric position of str in a character expression
9) REPEAT → repeat specific expression n times
10) SPACE → Insert blank spaces.

SQL QUERIES:

```
SQL> select * from employee;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
       104 CHARLES                   50200        100         10

SQL> select * from employee where salary>=all(select salary from employee where DNO=10);

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12

SQL> select * from employee where salary<=all(select salary from employee where DNO=10);

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       104 CHARLES                   50200        100         10
       103 BLAIR                     46350        103         13

SQL> select * from employee where salary<>all(select salary from employee where DNO=10);

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       102 SCARLET                   82400        102         12
       101 Jonas                     61800        101         11
       103 BLAIR                     46350        103         13
```

```
SQL> select * from dept where DNAME=(select DNAME from dept where DNAME ='FINANCE');

       DNO DNAME                STARTDATE     MGRSSN
---------- -------------------- --------- ----------
        10 FINANCE              12-NOV-21        100

SQL> select * from dept where exists DNAME=(select DNAME from dept where DNAME ='FINANCE');
select * from dept where exists DNAME=(select DNAME from dept where DNAME ='FINANCE')
                               *
ERROR at line 1:
ORA-00906: missing left parenthesis


SQL> select * from dept where exists(select DNAME from dept where DNAME ='FINANCE');

       DNO DNAME                STARTDATE     MGRSSN
---------- -------------------- --------- ----------
        10 FINANCE              12-NOV-21        100
        11 AUDIT                12-OCT-21        101
        12 MARKETING            01-NOV-21        102
        13 PRODUCTION           09-OCT-21        103
```

```
SQL> select * from employee;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
       104 CHARLES                   50200        100         10

SQL> select * from employee order by salary;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       103 BLAIR                     46350        103         13
       104 CHARLES                   50200        100         10
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
```

```
SQL> SELECT SUM(SALARY) AS "TOTAL SALARY" FROM EMPLOYEE GROUP BY DNO HAVING SUM(SALARY)<150000;

TOTAL SALARY
------------
       61800
       46350
      101700
       82400
```

```
SQL> select * from employee;

      SSN ENAME                    SALARY   SUPERSSN        DNO
--------- -------------------- ---------- ---------- ----------
      100 William                   51500        100         10
      101 Jonas                     61800        101         11
      102 SCARLET                   82400        102         12
      103 BLAIR                     46350        103         13
      104 CHARLES                   50200        100         10

SQL> SELECT * FROM EMPLOYEE E1 WHERE 1 =(SELECT  COUNT(DISTINCT SALARY) FROM EMPLOYEE E2 WHERE E1.DNO = E2.DNO AND E1.SALARY <=  E2.SALARY);

      SSN ENAME                    SALARY   SUPERSSN        DNO
--------- -------------------- ---------- ---------- ----------
      100 William                   51500        100         10
      101 Jonas                     61800        101         11
      102 SCARLET                   82400        102         12
      103 BLAIR                     46350        103         13

SQL> SELECT * FROM EMPLOYEE E1 WHERE E1.SALARY > (SELECT AVG(SALARY) FROM EMPLOYEE E2 WHERE E1.DNO = E2.DNO);

      SSN ENAME                    SALARY   SUPERSSN        DNO
--------- -------------------- ---------- ---------- ----------
      100 William                   51500        100         10
```

```
SQL> SELECT * FROM EMPLOYEE E1 WHERE E1.SALARY >= ALL(SELECT SALARY FROM EMPLOYEE E2 WHERE E1.DNO = E2.DNO);

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
```

Conclusion-

Thus we have successfully learned and implemented Simple queries, String manipulation operations on your own system

EXPERIMENT 7

Aim: Implement various types of Joins & views.

Objective: To combine records from two or more tables in a database.

Description: Join command is used to combine fields of tables using common values of each. It is used to retrieve data from 2 or more tables based on some logical relationship between tables.

SQL join types: inner join, left join, right join, full join, self join, cartesian join, equi join.

- inner join - returns rows when match in both tables
  eg. select ename, dname from employee e, dept d where dno = end no.
- left join - returns all rows from left table even if there are no matches in right table
- right join - vice versa of left join.
- full join - return all rows in all tables.
- self join - used to join table to itself.
  eg. select ename, e1.salary, e2.salary from employee e1, e2 where e1.dno = e2.dno.

- cartesian join - returns cartesian product of records from 2 or more joined tables.

eg. select ename, d.dname from dept d cross join employee;

- equi join - rows satisfying the selection criteria from both joined tables are selected.

eg. select ename, dname from employee e, dept d where eno = dno.

SQL QUERIES:

```
SQL> SELECT MAX(SALARY) FROM EMPLOYEE;

MAX(SALARY)
-----------
      82400

SQL> SELECT * FROM EMPLOYEE;

       SSN ENAME                    SALARY   SUPERSSN        DNO
---------- -------------------- ---------- ---------- ----------
       100 William                   51500        100         10
       101 Jonas                     61800        101         11
       102 SCARLET                   82400        102         12
       103 BLAIR                     46350        103         13
       104 CHARLES                   50200        100         10

SQL> SELECT * FROM DEPT;

       DNO DNAME                STARTDATE     MGRSSN
---------- -------------------- --------- ----------
        10 FINANCE              12-NOV-21        100
        11 AUDIT                12-OCT-21        101
        12 MARKETING            01-NOV-21        102
        13 PRODUCTION           09-OCT-21        103

SQL> SELECT ENAME,DNAME FROM EMPLOYEE E,DEPT D WHERE E.DNO = D.DNO;

ENAME                DNAME
-------------------- --------------------
William              FINANCE
CHARLES              FINANCE
Jonas                AUDIT
SCARLET              MARKETING
BLAIR                PRODUCTION
```

```
SQL> SELECT ENAME,E.DNO,DNAME FROM EMPLOYEE E,DEPT D WHERE E.DNO = D.DNO;

ENAME                       DNO DNAME
-------------------- ---------- --------------------
William                      10 FINANCE
CHARLES                      10 FINANCE
Jonas                        11 AUDIT
SCARLET                      12 MARKETING
BLAIR                        13 PRODUCTION
```

```
SQL> SELECT ENAME,E.DNO,DNAME FROM (EMPLOYEE E JOIN DEPT D ON E.DNO = D.DNO);

ENAME                       DNO DNAME
-------------------- ---------- --------------------
William                      10 FINANCE
CHARLES                      10 FINANCE
Jonas                        11 AUDIT
SCARLET                      12 MARKETING
BLAIR                        13 PRODUCTION

SQL> SELECT E.ENAME AS EMPNAME,S.ENAME AS SUPERVISORNAME FROM EMPLOYEE E,EMPLOYEE S WHERE E.SUPERSSN=S.SSN

EMPNAME              SUPERVISORNAME
-------------------- --------------------
William              William
CHARLES              William
Jonas                Jonas
SCARLET              SCARLET
BLAIR                BLAIR

SQL> SELECT E.ENAME,E.DNO,DNAME FROM (EMPLOYEE E LEFT OUTER JOIN DEPT D ON E.DNO=D.DNO);

ENAME                       DNO DNAME
-------------------- ---------- --------------------
CHARLES                      10 FINANCE
William                      10 FINANCE
Jonas                        11 AUDIT
SCARLET                      12 MARKETING
BLAIR                        13 PRODUCTION

SQL> SELECT E.ENAME,E.DNO,DNAME FROM (EMPLOYEE E RIGHT OUTER JOIN DEPT D ON E.DNO=D.DNO);

ENAME                       DNO DNAME
-------------------- ---------- --------------------
William                      10 FINANCE
CHARLES                      10 FINANCE
Jonas                        11 AUDIT
SCARLET                      12 MARKETING
BLAIR                        13 PRODUCTION
```

```
SQL> SELECT E.ENAME,E.DNO,DNAME FROM (EMPLOYEE E FULL OUTER JOIN DEPT D ON E.DNO=D.DNO);

ENAMESELECT * FROM STUD_V1    O DNAME
-------------------- ---------- --------------------
William                      10 FINANCE
Jonas                        11 AUDIT
SCARLET                      12 MARKETING
BLAIR                        13 PRODUCTION
CHARLES                      10 FINANCE
```

```
SQL> CREATE VIEW EMPLOYEE_V2 AS SELECT E.ENAME,E.DNO,DNAME FROM (EMPLOYEE E FULL OUTER JOIN DEPT D ON E.DNO=D.DNO);

View created.

SQL> SELECT * FROM EMPLOYEE_V2;

ENAME                       DNO DNAME
-------------------- ---------- --------------------
William                      10 FINANCE
Jonas                        11 AUDIT
SCARLET                      12 MARKETING
BLAIR                        13 PRODUCTION
CHARLES                      10 FINANCE
```

Conclusion:

Thus we have successfully learned and implemented joins & views in our system.

47_ YASH SARANG DBMS _ Experiment 8.

Aim: Conditional Loops & cursors in PL/SQL.

Theory: 1) PL/SQL is a block structured language. The programs are logical blocks that ~~are~~ can contain any number of nested sublocks.

2) It includes procedural language elements like conditions and loops. It allows declaration of constants, variables, procedures and functions, types and variable of those types and triggers.

3) The structure of a PL/SQL block is as follows

```
DECLARE:
    declaration section.
BEGIN:
    executable section
EXCEPTION:
    error handling section
END:
```

4) The execution section starts with the reserved keyword BEGIN and ends with END.

5) DDL is not allowed in a PL/SQL block.

6) Exception section is optional and handles any error that might occur in the program.

7) Cursors:

@ Oracle creates a certain portion in the memory for each/every SQL query that is executed.

Sundaram
FOR EDUCATIONAL USE

(b) Using Using PL/SQL this problem in the memory can be given a name of our choice.

(c) A cursor is basically a pointer to the context area and thus represents a structure in memory.

(d) Cursors are usually used to hold the data retrieved from the tables and perform and action on the data one row at a time.

(e) Cursors of two types: I) Implicit - Oracle created
II) Explicit. - User created.

SQL QUERIES:

1. To give raises to all employees earning a salary less than 1500.

```
SQL> declare
  2  cursor empcursor is select salary from employee;
  3  sal employee.salary%type;
  4  begin
  5  open empcursor;
  6  loop
  7  fetch empcursor into sal;
  8  exit when  empcursor%notfound;
  9  if sal<15000 then
 10  update employee set salary = salary + 0.15*salary where salary < 15000
 11  ;
 12  end if;
 13  end loop;
 14  close empcursor;
 15  end;
 16  /

PL/SQL procedure successfully completed.
```

```
SQL> select * from employee;

EMPLOYEE_ID EMPLOYEE_NAME                    SALARY DEPARTMENT_NO
----------- -------------------- ---------- ------------
        411 Rias                         500000
        111 Prasad                        30000          101
        112 Ash                           34000          101
        113 Tanmay                        34000          101
        114 Cray                           5290          101
        211 John                          60000          201
        212 Shweta                        40000          201
        213 Amit                          43000          201
        311 Paul                          60000          301
        312 Priya                         13225          301

10 rows selected.
```

2. To set job of all clerks with salary greater than 1300 as “Senior Clerk”

```
SQL> declare
  2  cursor empcursor is select salary,job from employee;
  3  sal employee.salary%type;
  4  job employee.job%type;
  5  begin
  6  open empcursor;
  7  loop
  8  fetch empcursor into sal,job;
  9  exit when empcursor%notfound;
 10  if sal>1300 and job like 'Clerk' then
 11  update employee set job = 'Senior Clerk' where sal<1300 and job like 'Clerk';
 12  end if;
 13  end loop;
 14  close empcursor;
 15  end;
 16  /
```

```
EMPLOYEE_ID EMPLOYEE_NAME                    SALARY DEPARTMENT_NO JOB
----------- -------------------- ---------- ------------ --------------------
        413 Pranita                       40000          401 Senior Clerk
        414 Sahil                         46000          401 Accountant
        116 Tejas                          1000          101 Programmer
```

Conclusion- Thus we have studied and successfully implemented PL/SQL.

47_ YASH SARANG EXPERIMENT 9 DBMS

Aim: Triggers / Functions and Procedure.

Theory:

TRIGGER:-

① A trigger is a piece of code that is run before or after a database table is modified.

② A trigger can be defined on:
- Tables/Views : DML (insert/update/delete)
- Schema : DDL Trigger and logon/logoff.
- Database : System event triggers (startup/shutdown)

③ A trigger is used to
- prevent invalid transaction.
- keep an audit trail of table.
- ensure rollback if database is inconsistent.

④ The different parts of a trigger are trigger statements, trigger restriction and action.

⑤ A row trigger is fired each time a row in the table is affected. It uses 'for each row' clause.

⑥ Statement triggers are fired once on behalf of the statement, independant of the number of rows the trigger statement affects.

⑦ Syntax for creating a trigger is as follows:

CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE / AFTER / INSTEAD OF} → specifies when executed.
{INSERT / UPDATE / DELETE} → specifies DML operation.
[of col_name] → specifies col_name that will be updated.

PROCEDURE & FUNCTIONS.

1) PL/SQL allows writing subprograms made up of logically grouped SQL and PL/SQL statements.

2) If we want to update data on table, a procedure is preffered while for retrieving information, a function is preffered.

3) A select SQL query can call a function but cannot call a procedure.

4) Procedure may return one or more values through parameters or may not return values at all. A function always returns a value using the return statement.

SQL QUERIES:

1. Create a trigger that fires before inserting or delete of a row in the emp table and displays the count of rows.

```
SQL> create trigger t
  2  before insert or delete on employee
  3  declare
  4  val number;
  5  begin
  6  select count(*) into val from employee;
  7  dbms_output.put_line('Number of rows are: ' || val );
  8  end;
  9  /

Trigger created.
```

```
SQL> insert into employee values(421,'Harish',5000,101,'Clerk');
Number of rows are: 17

1 row created.
```

2. Create a trigger that stops the user from entering Dept no in emp table if that dept no doesn't exist in dept table. The trigger should display the contents of dept table.

```
SQL> create trigger checker2
  2  before insert on employee
  3  for each row
  4  declare
  5  val number;
  6  begin
  7  select count(*) into val from department d where d.department_no = :new.department_no;
  8  if val = 0 then
  9  dbms_output.put_line('Department does not exist');
 10  else
 11  dbms_output.put_line('Value Inserted');
 12  end if;
 13  end;
 14  /

Trigger created.
```

```
SQL> insert into employee values(219,'Lata',7000,999,'Artist');
Number of rows are: 19
Department does not exist
```

```
SQL> insert into employee values(219,'Lata',7000,401,'Artist');
Number of rows are: 19
Value Inserted
```

3. Write a procedure that: Accepts department number and percentage of raise in sal Updates the sal of all those employees under that department

```
SQL> create procedure bonus2( deptno in number, inc in float)
  2  as
  3  begin
  4  update employee set salary = salary + salary*inc where department_no = deptno;
  5  dbms_output.put_line('Values Updated');
  6  end;
  7  /

Procedure created.
```

```
SQL> EXEC bonus2(201,0.15);
Values Updated

PL/SQL procedure successfully completed.

SQL> select * from employee where department_no = 201;

EMPLOYEE_ID EMPLOYEE_NAME            SALARY DEPARTMENT_NO JOB
----------- -------------------- ---------- ------------- --------------------
        211 John                      69000           201
        212 Shweta                     5750           201
        213 Amit                      49450           201
```

4. Write a function that accepts: Dept no and returns the total sal of all employees in that department.

```
SQL> create function dept (deptno in number)
  2  return number is totalsum employee.salary%type;
  3  begin
  4  select sum(salary) into totalsum from employee where department_no = deptno;
  5  return(totalsum);
  6  end;
  7  /

Function created.
```

```
SQL> declare
  2  answer number;
  3  deptno number;
  4  begin
  5  deptno := 101;
  6  answer := dept(deptno);
  7  dbms_output.put_line('department: '|| deptno || ' Sum is ' || answer);
  8  end;
  9  /
department: 101 Sum is 165290

PL/SQL procedure successfully completed.
```

Conclusion: Triggers / Functions and procedures have been implemented.

47 YASH SARANG - DBMS EXPERIMENT 10.

Aim: Demonstrate DCL & TCL commands.

Objective: To execute commands that control databases and transactions.

Descriptions:

Data Control Language (DCL)
Commands that control a database, including administering privileges & committing data. Used to create roles, permissions & referential integrity as well it is used to control access to databases by securing it.

Revoke, Grant are two such commands

GRANT- gives the user access privileges to the database. Only Database administrator's or owner's of the database object can provide/remove privileges on a database object. used to assign permission to users.

Eg. grant insert on emp to user1;

REVOKE: withdraws user access to database given that with the GRANT command.

Syntax- remove <permission> on <object name> from username;

GRANT:

```
SQL> grant select, update on emp to flash
  2  ;

Grant succeeded.
```

REVOKE:

```
SQL> revoke select,update on emp from flash;

Revoke succeeded.
```

TCL COMMANDS: COMMIT, ROLLBACK, SAVEPOINT---

```
SQL> select * from customer;

       CID CNAME                     PHNO
---------- -------------------- ----------
        12 ramesh                   99201
        12 ramesh                   98021
        13 suresh                   86547
        14 mahesh                  675320
```

COMMIT:

```
SQL> delete from customer
  2  where cid=12;

2 rows deleted.

SQL> commit;

Commit complete.

SQL> select * from customer;

       CID CNAME                     PHNO
---------- -------------------- ----------
        13 suresh                   86547
        14 mahesh                  675320
```

ROLLBACK:

```
SQL> delete from customer
  2  where cid=12;

2 rows deleted.

SQL> rollback;

Rollback complete.
```

SAVEPOINT:

```
SQL> insert into customer values (12,'ramesh',99201);

1 row created.

SQL> insert into customer values(12,'ramesh',98021);

1 row created.

SQL> savepoint customers;

Savepoint created.

SQL> select * from customer;

       CID CNAME                     PHNO
---------- -------------------- ----------
        12 ramesh                   99201
        13 suresh                   86547
        14 mahesh                  675320
        12 ramesh                   98021

SQL> delete from customer where cid=12;

2 rows deleted.

SQL> select * from customer;

       CID CNAME                     PHNO
---------- -------------------- ----------
        13 suresh                   86547
        14 mahesh                  675320

SQL> rollback to customers;

Rollback complete.

SQL> select * from customer;

       CID CNAME                     PHNO
---------- -------------------- ----------
        12 ramesh                   99201
        13 suresh                   86547
        14 mahesh                  675320
        12 ramesh                   98021
```

47_ YASH SARANG_ EXPERIMENT 11_ DBMS

Transition Control:

A transition can defined as a group of tasks. A single task is the min. processing unit which cannot be divided further.

Let's take an eg. of a bank employee transfering 500 Rs from A to B.

A's acc	B's acc.
Open_Acc(A) = A.balance	Open_Acc(B) = B.balance
New_Bal = Old_Bal − 500	New_Bal = Old_Bal + 500.
A.balance = New_Bal	B.balance = New_Bal.
Close_Acc(A)	Close Acc(B).

Concurrent Control.

1. Concurrency control manages the transactions simultaneously without letting them interfere with each other.
2. The main objective of concurrency control is to allow many users perform different operations at the same time.
3. Using more than one transaction concurrently improves the performance of system.
4. If we are not able to perform the operations concurrently, then there can be serious problems such as loss of data integrity & consistency.
5. It reduces waiting time of transaction.

```
SQL> select * from emp;

NAME                    SALARY        SSN        DNO   SUPERSSN
------------------- ---------- ---------- ---------- ----------


subrato                  55000          1          3          1
manas                    51750          2          2          2
jayesh                   65000          3          1          3
sarthak                46287.5          4          4          4
arnav                    50000          5          5          5
```

- CHANGING SALARY OF SSN=1
- COMMIT,SAVING IT
- CHANGING SALARY OF SSN=2
- ROLLING BACK THE CHANGE

```
SQL> set serveroutput on;
SQL>  DECLARE
  2   salary emp.salary%type;
  3  BEGIN
  4  dbms_output.put_line('Updating salary of employee,ssn=1');
  5  update emp set salary=salary+10000 where ssn=1;
  6   dbms_output.put_line('Committing transaction');
  7  COMMIT;
  8   dbms_output.put_line('Adding a savepoint');
  9  SAVEPOINT sal1;
 10  dbms_output.put_line('Updating salary of employee,ssn=2');
 11  update emp set salary=salary+10000 where ssn=2;
 12  dbms_output.put_line('Rolling back to savepoint');
 13  ROLLBACK TO sal1;
 14  END;
 15  /
Updating salary of employee,ssn=1
Committing transaction
Adding a savepoint
Updating salary of employee,ssn=2
Rolling back to savepoint

PL/SQL procedure successfully completed.
```

- WE CAN SEE THAT ONLY SALARY OF EMP SSN=1 GETS UPDATED

```
SQL> select * from emp;

NAME                    SALARY        SSN        DNO   SUPERSSN
------------------- ---------- ---------- ---------- ----------


subrato                  65000          1          3          1
manas                    51750          2          2          2
jayesh                   65000          3          1          3
sarthak                46287.5          4          4          4
arnav                    50000          5          5          5
```

```
SQL>  set serveroutput on;
SQL> DECLARE
  2   rollno student.sno%type;
  3  s_age student.age%type;
  4  snm student.sname%type;
  5  s_cr student.course%type;
  6  BEGIN
  7  rollno:=&sno;
  8  s_age:=&age;
  9  snm:='&name';
 10  s_cr:='&course':
 11  insert into student values (rollno,s_age,snm,s_cr);
 12  dbms_output.put_line('one record inserted');
 13  COMMIT;
 14  SAVEPOINT save1;
 15   rollno:=&sno;
 16   s_age:=&age;
 17  snm:='&name';
 18  s_cr:='&course':
 19  INSERT into student values (rollno,s_age,snm,s_cr);
 20   dbms_output.put_line('one record inserted');
 21  ROLLBACK to SAVEPOINT save1;
 22  END;
 23  /
```

```
Enter value for sno: 60
old   7: rollno:=&sno;
new   7: rollno:=60;
Enter value for age: 19
old   8: s_age:=&age;
new   8: s_age:=19;
Enter value for name: subrato
old   9: snm:='&name';
new   9: snm:='subrato';
Enter value for course: ai
old  10: s_cr:='&course':
new  10: s_cr:='ai':
Enter value for sno: 61
old  15:  rollno:=&sno;
new  15:  rollno:=61;
Enter value for age: 19
old  16:  s_age:=&age;
new  16:  s_age:=19;
Enter value for name: vivek
old  17: snm:='&name';
new  17: snm:='vivek';
Enter value for course: it
old  18: s_cr:='&course':
new  18: s_cr:='it':
```

```
SQL> desc employee;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPLOYEE_ID                               NOT NULL NUMBER(38)
 ENAME                                     NOT NULL VARCHAR2(20)
 SALARY                                             NUMBER
 FK_EMPLOYEE                                        NUMBER(38)
 D_NO                                               NUMBER(38)

SQL> lock table employee in share mode;

Table(s) Locked.

SQL> commit;

Commit complete.
```

```
SQL> lock table employee in exclusive mode;

Table(s) Locked.

SQL> rollback;

Rollback complete.
```

```
SQL> lock table system.Donor in exclusive mode;

Table(s) Locked.

SQL> lock table system.Receiver in exclusive mode;
lock table system.Receiver in exclusive mode
*
ERROR at line 1:
ORA-00060: deadlock detected while waiting for resource


SQL> rollback;

Rollback complete.
```

```
SQL> insert into emp values('mahi',30000,6,6,6);

1 row created.
```

```
SQL> savepoint s1;

Savepoint created.

SQL> update emp
  2  set salary=salary+1000;

13 rows updated.

SQL> select * from emp;

NAME                    SALARY        SSN        DNO   SUPERSSN
-------------------- ---------- ---------- ---------- ----------


subrato                  56000          1          3          1
manas                    52750          2          2          2
jayesh                   66000          3          1          3
sarthak                47287.5          4          4          4
arnav                    51000          5          5          5




NAME                    SALARY        SSN        DNO   SUPERSSN
-------------------- ---------- ---------- ---------- ----------
Ramesh                   11000          1          2          1
mahi                     31000          6          6          6

13 rows selected.

SQL> rollback to s1;

Rollback complete.
```

```
SQL> select * from emp;

NAME                    SALARY        SSN        DNO   SUPERSSN
-------------------- ---------- ---------- ---------- ----------


subrato                  55000          1          3          1
manas                    51750          2          2          2
jayesh                   65000          3          1          3
sarthak                46287.5          4          4          4
arnav                    50000          5          5          5




NAME                    SALARY        SSN        DNO   SUPERSSN
-------------------- ---------- ---------- ---------- ----------
Ramesh                   10000          1          2          1
mahi                     30000          6          6          6
```

47 YASH SARANG - EXPERIMENT 12 - DBMS

Aim: Implement Database Connectivity (JDBC, ODBC)

Theory:

JDBC Architecture.

In the two tier model, a Java app talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source. may be loaded on another machine to which the user is connected via network.

This is referred to as a client/server config, with user's machine as the client, & the machine housing the data source as the server. The network can be an intent.

In the 3 tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands & sends the results back to the middle tier, which then sends them to the user

Until recently, the middle tier has often been written in languages such as C or C++ which offers fast performance. However with the introduction of optimizing compilers that translate Java bytecode into efficient machine specific code & technologies such as enterprise JavaBeans, the Java platform is fast becoming a standard platform of middle tier development.

Fundamental Steps in JDBC.

1. Import JDBC packages
2. Load & register the JDBC driver.
3. Open a connection to the database.
4. Create a statement object to perform a query.
5. Execute the statement object & return a query resultset
6. Process the result set.
7. Close the result set & statement objects.
8. Close the connection.

```
import java.sql.*;
public class jdbc {

    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost:3306/";
    static final String USER = "java";
    static final String PASS = "subo";
public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{

Class.forName("com.mysql.cj.jdbc.Driver");

System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL, USER, PASS);

System.out.println("Creating database...");
stmt = conn.createStatement();

String sql1 = ("CREATE DATABASE STUDENTS1");
stmt.executeUpdate(sql1);
System.out.println("Database created successfully...");
}

catch(SQLException se){
se.printStackTrace();
}

catch(Exception e){
e.printStackTrace();
}

finally{
    try{
        if(stmt!=null)
        stmt.close();
        }

    catch(SQLException se2){
}
try{
    if(conn!=null)
    conn.close();
    }
    catch(SQLException se){
    se.printStackTrace();}
}

System.out.println("Goodbye!");
}
}
```

```
Connecting to database...
Creating database...
java.sql.SQLSyntaxErrorException: Access denied for user 'java'@'localhost' to database 'students1'
        at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:120)
        at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:122)
        at com.mysql.cj.jdbc.StatementImpl.executeUpdateInternal(StatementImpl.java:1340)
        at com.mysql.cj.jdbc.StatementImpl.executeLargeUpdate(StatementImpl.java:2089)
        at com.mysql.cj.jdbc.StatementImpl.executeUpdate(StatementImpl.java:1251)
        at jdbc.main(jdbc.java:22)
Goodbye!
```

```
Connecting to database...
Creating database...
Goodbye!
```
