DBMS Lab Manual

Revision History

Document	Version	Date	Author(s)
ASIET_DBMS_Lab_v1.0	First draft version	30/3/23	

Expt No 1

ER diagram and Relational Schema

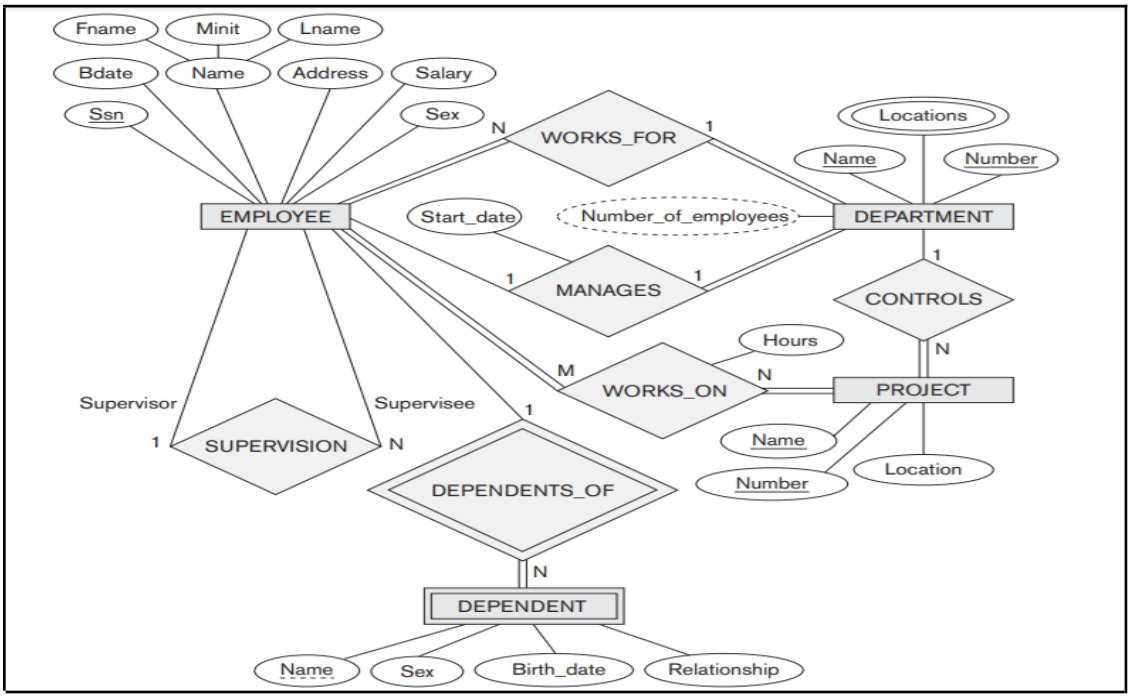
AIM:Design a database schema for an application with ER diagram from a problem description

Problem Description:

The COMPANY database keeps track of a company's employees, departments, and projects. Suppose that after the requirements collection and analysis phase, the database designers provide the following description of the miniworld—the part of the company that will be represented in the database.

- The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- We store each employee's name, Social Security number,address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).
- We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

ER Diagram:



Entities:

- An entity type DEPARTMENT with attributes Name, Number, Locations, Manager, and Manager_start_date. Locations is the only multivalued attribute. We can specify that both Name and Number are (separate) key attributes because each was specified to be unique.
- An entity type PROJECT with attributes Name, Number, Location, and Controlling_department. Both Name and Number are (separate) key attributes.
- An entity type EMPLOYEE with attributes Name, Ssn, Sex, Address, Salary, Birth_date, Department, and Supervisor. Both Name and Address may be composite attributes; however, this was not specified in the requirements.
- An entity type DEPENDENT with attributes Employee, Dependent_name, Sex, Birth_date, and Relationship (to the employee).

Relationship types:

- MANAGES, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. DEPARTMENT participation is not clear from the requirements. We assume that a department must have a manager at all times, which implies total participation.The attribute Start_date is assigned to this relationship type.
- WORKS_FOR, a 1:N relationship type between DEPARTMENT and EMPLOYEE. Both participations are total.
- CONTROLS, a 1:N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total, whereas that of DEPARTMENT is determined to be partial, assuming that some departments may control no projects.

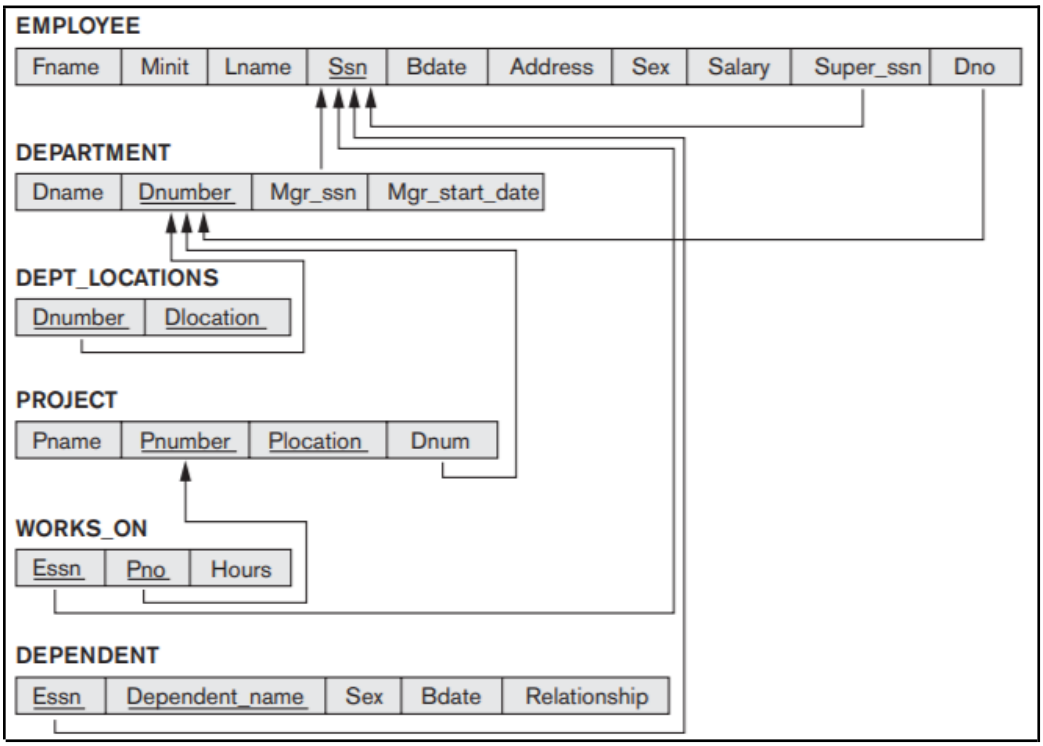
- SUPERVISION, a 1:N relationship type between EMPLOYEE (in the supervisor role) and EMPLOYEE (in the supervisee role). Both participations are determined to be partial, assuming that not every employee is a supervisor and not every employee has a supervisor.
- WORKS_ON, determined to be an M:N relationship type with attribute Hours, after the users indicate that a project can have several employees working on it. Both participations are determined to be total. DEPENDENTS_OF, a 1:N relationship type between EMPLOYEE and DEPENDENT, which is also the identifying relationship for the weak entity type DEPENDENT. The participation of EMPLOYEE is partial, whereas that of DEPENDENT is total

Summary :

Figure above displays the COMPANY ER database schema as an ER diagram. We now review the full ER diagram notation.

- Entity types such as EMPLOYEE, DEPARTMENT, and PROJECT are shown in rectangular boxes. Relationship types such as WORKS_FOR, MANAGES, CONTROLS, and WORKS_ON are shown in diamond-shaped boxes attached to the participating entity types with straight lines.
- Attributes are shown in ovals, and each attribute is attached by a straight line to its entity type or relationship type.
- Component attributes of a composite attribute are attached to the oval representing the composite attribute, as illustrated by the Name attribute of EMPLOYEE.
- Multivalued attributes are shown in double ovals, as illustrated by the Locations attribute of DEPARTMENT.
- Key attributes have their names underlined.
- Derived attributes are shown in dotted ovals, as illustrated by the Number_of_employees attribute of DEPARTMENT.
- Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds, as illustrated by the DEPENDENT entity type and the DEPENDENTS_OF identifying relationship type. The partial key of the weak entity type is underlined with a dotted line.
- The cardinality ratio of each binary relationship type is specified by attaching a 1, M, or N on each participating edge. The cardinality ratio of DEPARTMENT:EMPLOYEE in MANAGES is 1:1, whereas it is 1:N for DEPARTMENT: EMPLOYEE in WORKS_FOR, and M:N for WORKS_ON.
- The participation constraint is specified by a single line for partial participation and by double lines for total participation (existence dependency).
- In Figure we show the role names for the SUPERVISION relationship type because the same EMPLOYEE entity type plays two distinct roles in that relationship. Notice that the cardinality ratio is 1:N from supervisor to supervisee because each employee in the role of supervisee has at most one direct supervisor, whereas an employee in the role of supervisor can supervise zero or more employees

Relational Schema:



ER-to-Relational Schema Mapping:

Step 1: Mapping of Regular Entity Types

We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in Figure to correspond to the regular entity types EMPLOYEE, DEPARTMENT, and PROJECT. We choose Ssn, Dnumber, and Pnumber as primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT, respectively

Step 2: Mapping of Weak Entity Types

We create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. We include the primary key Ssn of the EMPLOYEE relation—which corresponds to the owner entity type—as a foreign key attribute of DEPENDENT; We rename it Essn. The primary key of the DEPENDENT relation is the combination {Essn, Dependent_name}, because Dependent_name is the partial key of DEPENDENT

Step 3: Mapping of Binary 1:1 Relationship Types.

We map the 1:1 relationship type MANAGES from Figure by choosing the participating entity type DEPARTMENT to serve in the role of S because its participation in the MANAGES relationship type is total (every department has a manager). We include the primary key of the EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it Mgr_ssn. We also include the simple attribute

Start_date of the MANAGES relationship type in the DEPARTMENT relation and rename it Mgr_start_date

Step 4: Mapping of Binary 1:N Relationship Types

We now map the 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION from Figure. For WORKS_FOR we include the primary key Dnumber of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it Dno. For SUPERVISION we include the primary key of the EMPLOYEE relation as foreign key in the EMPLOYEE relation itself—because the relationship is recursive—and call it Super_ssn. The CONTROLS relationship is mapped to the foreign key attribute Dnum of PROJECT, which references the primary key Dnumber of the DEPARTMENT relation.

Step 5: Mapping of Binary M:N Relationship Types

We map the M:N relationship type WORKS_ON from the ER diagram by creating the relation WORKS_ON in relational schema. We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them Pno and Essn, respectively. We also include an attribute Hours in WORKS_ON to represent the Hours attribute of the relationship type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {Essn, Pno}.

Step 6: Mapping of Multivalued Attributes

We create a relation DEPT_LOCATIONS. The attribute Dlocation represents the multivalued attribute LOCATIONS of DEPARTMENT, while Dnumber—as foreign key—represents the primary key of the DEPARTMENT relation. The primary key of DEPT_LOCATIONS is the combination of {Dnumber, Dlocation}. A separate tuple will exist in DEPT_LOCATIONS for each location that a department has.

RESULT:

We successfully created an ER diagram from the problem description given. We were also able to extract the relational schema from the ER diagram.

Expt No 2

APPLICATION OF DDL COMMANDS USING UI AND SQL

AIM : Creation, modification, configuration, and deletion of databases using UI and SQL Commands.

QUERY

Create database students and execute various commands on it.

create database students;
show databases;
use students;

Create a table students with the fields student id, name, email and phone number.

create table Student(stud_id INT AUTO_INCREMENT PRIMARY KEY,stud_fname VARCHAR(20),stud_lname VARCHAR(20),stud_email VARCHAR(20),stud_ph VARCHAR(10));

Create a table subject to store the list of subjects.

create table Subject(sub_id INT AUTO_INCREMENT PRIMARY KEY,sub_name VARCHAR(20));

Create a table marks to store marks of students for various subjects.

create table Marks(sub_id INT,stud_id INT,marks INT, PRIMARY KEY(sub_id,stud_id));
show tables;

Display the details of the tables created.

desc Student;
desc Subject;
desc Marks;

Alter the tables to include foreign keys.

alter table Marks ADD FOREIGN KEY (stud_id) REFERENCES Student(stud_id);
alter table Marks ADD FOREIGN KEY (sub_id) REFERENCES Subject(sub_id);

Alter the tables to initialize auto increment value.

alter table Student AUTO_INCREMENT=100;
alter table Subject AUTO_INCREMENT=200;

Alter the tables to add new column.

alter table Student add address char(50);
describe Student;

Alter the tables to modify datatype of a column.

alter table Student modify column address varchar(50);
describe Student;

Alter the tables to rename column.

alter table Student rename column address to stud_address;
describe Student;

Alter the tables to delete column.

alter table Student drop column stud_address;
describe Student;

Truncate command

To test truncate command, we have to populate table.

insert into Student (stud_fname, stud_lname, stud_email, stud_ph) values ('Alice', 'Tom', ' alice@alice.com', '9889975555');

insert into Student (stud_fname, stud_lname, stud_email, stud_ph) values ('Bob', 'John', ' bob@bob.com', '9889975556');

*select * from Student;*

Test Truncate command
SET FOREIGN_KEY_CHECKS=0; # Used to disable referential integrity
truncate table Student;
*select * from Student; # Observe entire data is cleared from table.*

Drop the tables created.

drop table Student;
drop table Subject;
drop table Marks;
drop database students;

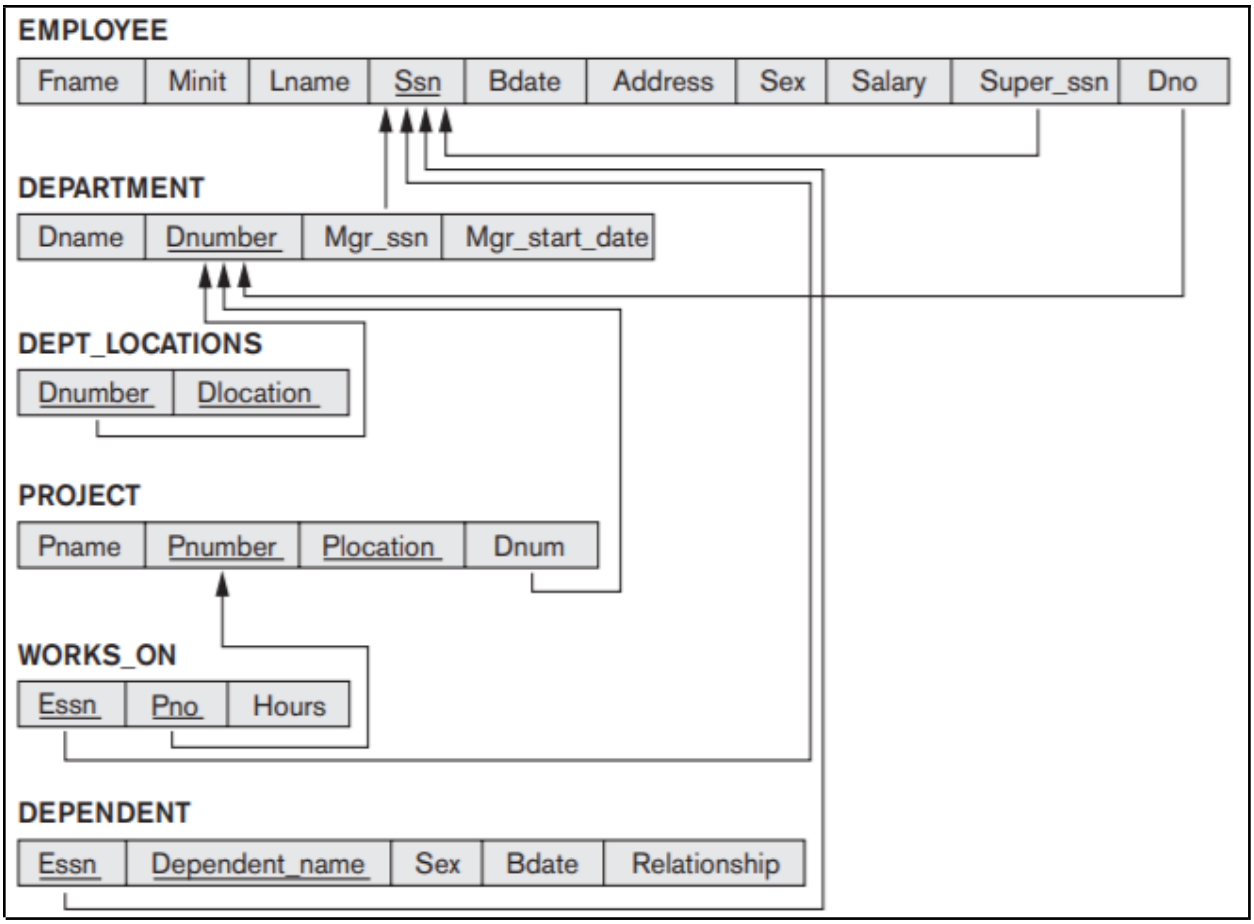
RESULT: Successfully executed the queries using MySQL Workbench.

Expt No 3

CREATION OF DATABASE SCHEMA AND EXTRACTION OF ER DIAGRAM

AIM: Creation of database schema - DDL (create tables, set constraints, enforce relationships, create indices, delete and modify tables). Export ER diagram from the database and verify relationships (with the ER diagram designed in step 1).

Create a database schema for the below diagram.



QUERY:

```
CREATE TABLE EMPLOYEE
( Fname          VARCHAR(10)   NOT NULL,
  Minit          CHAR,
  Lname          VARCHAR(20)      NOT NULL,
  Ssn            CHAR(9)          NOT NULL,
  Bdate          DATE,
  Address        VARCHAR(30),
  Sex            CHAR(1),
  Salary         DECIMAL(5),
  Super_ssn      CHAR(9),
  Dno            INT              NOT NULL,
PRIMARY KEY   (Ssn));

CREATE TABLE DEPARTMENT
( Dname          VARCHAR(15)      NOT NULL,
  Dnumber        INT              NOT NULL,
  Mgr_ssn        CHAR(9)          NOT NULL,
  Mgr_start_date  DATE,
PRIMARY KEY (Dnumber),
FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

```
CREATE TABLE DEPT_LOCATIONS
( Dnumber        INT             NOT NULL,
  Dlocation      VARCHAR(15)     NOT NULL,
PRIMARY KEY (Dnumber, Dlocation),
FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );

CREATE TABLE PROJECT
( Pname          VARCHAR(15)     NOT NULL,
  Pnumber        INT             NOT NULL,
  Plocation      VARCHAR(15),
  Dnum           INT             NOT NULL,
PRIMARY KEY (Pnumber),
FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );

CREATE TABLE WORKS_ON
( Essn           CHAR(9)         NOT NULL,
  Pno            INT             NOT NULL,
  Hours          DECIMAL(3,1)    NOT NULL,
PRIMARY KEY (Essn, Pno),
FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );

CREATE TABLE DEPENDENT
( Essn           CHAR(9)         NOT NULL,
  Dependent_name VARCHAR(15)     NOT NULL,
  Sex            CHAR,
  Bdate          DATE,
  Relationship   VARCHAR(8),
PRIMARY KEY (Essn, Dependent_name),
FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );

ALTER TABLE DEPARTMENT
 ADD CONSTRAINT Dep_emp FOREIGN KEY (Mgr_ssn) REFERENCES
EMPLOYEE(Ssn);

ALTER TABLE EMPLOYEE
 ADD CONSTRAINT Emp_emp FOREIGN KEY (Super_ssn) REFERENCES
EMPLOYEE(Ssn);

ALTER TABLE EMPLOYEE
 ADD CONSTRAINT Emp_dno FOREIGN KEY  (Dno) REFERENCES
DEPARTMENT(Dnumber);

ALTER TABLE EMPLOYEE
 ADD CONSTRAINT Emp_super FOREIGN KEY  (Super_ssn) REFERENCES
EMPLOYEE(Ssn);
```

EER diagram: <SCREENSHOT>

Export ER Diagram from Workbench

RESULT:

We successfully created the Database Schema from the problem description given. We were also able to export the ER diagram from the database and relationships are verified.

Expt No 4

APPLICATION OF DML COMMANDS USING SQL

AIM :Insertion, updation, deletion, and selection of databases using SQL Commands

1. Insert data into the Employee schema created in Expt No 3.

```
SET FOREIGN_KEY_CHECKS=0;

INSERT INTO EMPLOYEE
VALUES     ('John','B','Smith',123456789,'1965-01-09','731 Fondren, Houston
TX','M',30000,333445555,5),
        ('Franklin','T','Wong',333445555,'1965-12-08','638 Voss, Houston
TX','M',40000,888665555,5),
        ('Alicia','J','Zelaya',999887777,'1968-01-19','3321 Castle, Spring
TX','F',25000,987654321,4),
        ('Jennifer','S','Wallace',987654321,'1941-06-20','291 Berry, Bellaire
TX','F',43000,888665555,4),
        ('Ramesh','K','Narayan',666884444,'1962-09-15','975 Fire Oak, Humble
TX','M',38000,333445555,5),
        ('Joyce','A','English',453453453,'1972-07-31','5631 Rice, Houston
TX','F',25000,333445555,5),
        ('Ahmad','V','Jabbar',987987987,'1969-03-29','980 Dallas, Houston
TX','M',25000,987654321,4),
        ('James','E','Borg',888665555,'1937-11-10','450 Stone, Houston
TX','M',55000,null,1);

INSERT INTO DEPARTMENT VALUES ('Research',5,333445555,'1988-05-22');
INSERT INTO DEPARTMENT VALUES ('Administration',4,987654321,'1995-01-01');
INSERT INTO DEPARTMENT VALUES ('Headquarters',1,888665555,'1981-06-19');

INSERT INTO DEPT_LOCATIONS (Dnumber, Dlocation) VALUES (1,'Houston');
INSERT INTO DEPT_LOCATIONS (Dnumber, Dlocation) VALUES (4,'Stafford');
INSERT INTO DEPT_LOCATIONS (Dnumber, Dlocation) VALUES (5,'Bellaire');
INSERT INTO DEPT_LOCATIONS (Dnumber, Dlocation) VALUES (5,'Sugarland');
INSERT INTO DEPT_LOCATIONS (Dnumber, Dlocation) VALUES (5,'Houston');
```

```
INSERT INTO PROJECT
VALUES      ('ProductX',1,'Bellaire',5),
            ('ProductY',2,'Sugarland',5),
            ('ProductZ',3,'Houston',5),
            ('Computerization',10,'Stafford',4),
            ('Reorganization',20,'Houston',1),
            ('Newbenefits',30,'Stafford',4);

INSERT INTO WORKS_ON
VALUES      (123456789,1,32.5),
            (123456789,2,7.5),
            (666884444,3,40.0),
            (453453453,1,20.0),
            (453453453,2,20.0),
            (333445555,2,10.0),
            (333445555,3,10.0),
            (333445555,10,10.0),
            (333445555,20,10.0),
            (999887777,30,30.0),
            (999887777,10,10.0),
            (987987987,10,35.0),
            (987987987,30,5.0),
            (987654321,30,20.0),
            (987654321,20,15.0),
            (888665555,20,16.0);

INSERT INTO DEPENDENT
VALUES      (333445555,'Alice','F','1986-04-04','Daughter'),
            (333445555,'Theodore','M','1983-10-25','Son'),
            (333445555,'Joy','F','1958-05-03','Spouse'),
            (987654321,'Abner','M','1942-02-28','Spouse'),
            (123456789,'Michael','M','1988-01-04','Son'),
            (123456789,'Alice','F','1988-12-30','Daughter'),
            (123456789,'Elizabeth','F','1967-05-05','Spouse');
```

2. <u>UPDATE QUERY</u>

- **Update Salary of all employee by 1000 $**

```
update EMPLOYEE Set Salary = Salary+1000;
```

- **Update Address of Ssn 666884444 to "100 Centre, Stafford TX 77477"**

```
update EMPLOYEE set Address = '100 Centre, Stafford TX 77477' where Ssn =
'666884444';
```

3. SELECT QUERIES:

- **Write a query to get the details of a Employee whose Ssn = 666884444.**

```
select *
from EMPLOYEE
where Ssn = '666884444';
```

- **Write a query to get the Address of Employee Ramesh Narayan**

```
select Address
from EMPLOYEE
where Fname = 'Ramesh' and Lname = 'Narayan';
```

- **Write a query to get the list of employees working in Department No = 5**

```
select Fname, Lname
from EMPLOYEE
where Dno = 5;
```

- **Write a query to get the list of Employees working in Research Department.**

```
select Fname, Lname
from EMPLOYEE, DEPARTMENT
where Dno = Dnumber and Dname = 'Research';
```

- **Write a query to get the Manager's Ssn of "Research" department.**

```
select Mgr_ssn
from DEPARTMENT
where Dname = 'Research';
```

- **Write a query to get the Manager's Name of "Research" department.**

```
select Fname, Lname
from EMPLOYEE, DEPARTMENT
where Dno = Dnumber and Dname = 'Research';
```

3. DELETE QUERIES:

- **Delete the details of Research department from DEPARTMENT tables**

```
delete from DEPARTMENT where Dname = 'Research';
```

- **Delete the contents of DEPARTMENT Table**

```
delete from DEPARTMENT;
```

4. VIEW

- **Create a view Emp(Ssn , Fname, Lname, Sex, Salary,Dno) from EMPLOYEE Table**

  CREATE VIEW Emp AS SELECT Ssn,Fname, Lname, Sex, Salary, Dno FROM EMPLOYEE;

- **Display the contents of View**

  *select * from Emp;*

RESULT: Successfully executed the queries using SQL DML Commands

Expt No 5

IMPLEMENTATION OF BUILT IN FUNCTIONS

AIM: Implementation of built in functions in RDBMS

A. Create a table store. Fields are order no, code, item, quantity, price, discount, mrp

QUERY

```
Create table store (order_no int primary key, code int, item char(15), quantity varchar(8), price int,
discount varchar(7), mrp int);
Insert into store values(‘1’, ‘1’, ‘soap’, ‘5’, ‘75’, ‘2%’, ‘72’,);
1 row created;
Insert into store values(‘2’, ‘2’, ‘chilly powder’, ‘2’, ‘24’, ‘3%’, ‘20’,);
1 row created;
Insert into store values(‘3’, ‘3’, ‘atta’, ‘2’, ‘70’, ‘3%’, ‘78’,);
1 row created;
Insert into store values(‘4’, ‘4’, ‘pepper’, ‘5’, ‘524’, ‘5%’, ‘520’,);
1 row created;
Insert into store values(‘5’, ‘5’, ‘salt’, ‘4’, ‘40’, ‘2%’, ‘39’,);
1 row created;
```

B. Display the table;

QUERY

```
Select * from store;
```

OUTPUT

Oder_no	code	item	quantity	price	discount	mrp
1	1	soap	5	75	2%	72
2	2	chilly powder	2	24	3%	20
3	3	atta	2	70	3%	78
4	4	pepper	5	524	5%	520
5	5	salt	4	40	2%	39

c. Write an SQL query to display the reminder, if the amount of an each item in store is divided by 9.

QUERY

Select item, mod(mrp,9) from store;

OUTPUT

Item	mod(mrp)
Soap	0
chilly powder	2
atta	6
pepper	7
salt	3

d. Write SQL query to display the amount in store and its square.

QUERY

Select price, power(price,2) as power from store;

OUTPUT

Price	Power
75	5625
24	576
70	49000
524	274576
40	1600

e. Program to divide the amount in stock of each item by 7 in store table and display the resut round to the nearest integer.

QUERY

Select price, round(price/7,0) as round from store;

OUTPUT

Price	Round
75	11
24	3
70	10
524	75
40	6

RESULT: Successfully executed the queries using SQL DML Commands.

EXPT NO:-6

AGGREGATE FUNCTIONS

AIM:-Implementation of various aggregate functions in SQL

1. Find the number of Employee in the organization.

select count(*) from EMPLOYEE;

2. **Find the sum of the salaries of all employees, the maximum salary,the minimum salary, and the average salary.**

SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
FROM EMPLOYEE;

3) **Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.**

SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
FROM (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
WHERE Dname='Research';

4)**Retrieve the total number of employees in the company and the number of employees in the 'Research' department**

SELECT COUNT (*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research';

5)**Count the number of distinct salary values in the database.**

SELECT COUNT (DISTINCT Salary)
FROM EMPLOYEE;

RESULT: Successfully executed the queries using SQL DML Commands.

EXPT NO:-7

ORDER BY,GROUP BY AND HAVING CLAUSE

AIM:-Implementation of order by,group by and having clause

1. For each department, retrieve the department number, the number of employees in the department.

```
select Dno, count(*)
from EMPLOYEE
group by Dno;
```

2. For each department, retrieve the department name, the number of employees in the department, and their average salary.

```
select Dname, count(Ssn), avg(Salary)
from EMPLOYEE , DEPARTMENT
where Dno = Dnumber
group by Dname;
```

3. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

```
desc PROJECT;
desc WORKS_ON;

select Pnumber, Pname, count(Essn)
from PROJECT, WORKS_ON
where Pnumber = Pno
group by Pno;
```

4. For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
select Pnumber, Pname, count(Essn)
from PROJECT, WORKS_ON where Pnumber = Pno
group by Pno
having count(Essn) > 2;
```

5. For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

```
select Pnumber, Pname, count(Essn)
from PROJECT, WORKS_ON where Pnumber = Pno AND Dnum = 5
```

group by Pno;

6. For each department that has more than two employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
select Dno, count(Ssn)
from EMPLOYEE
where Salary < 40000
group by Dno having count(Ssn) > 2;
```

7. For each department that has more than two employees, retrieve the department number , department name and the number of its employees who are making more than $40,000.

```
select Dno, Dname, count(Ssn)
from EMPLOYEE, DEPARTMENT
where Dno = Dnumber AND  Salary < 40000
group by Dno
having count(Ssn) > 2;
```

8. List the total salary paid to employees in each department, but only for departments with a total salary greater than $100000.

```
SELECT Dname, SUM(Salary) as total_salary
FROM DEPARTMENT , EMPLOYEE
where Dnumber = Dno
GROUP BY Dname HAVING total_salary > 100000;
```

9. List all employees name and salary in the Research department, ordered by their last name

```
select Lname, Dname, Salary
from EMPLOYEE, DEPARTMENT
where Dno = Dnumber and Dname = 'Research'
order by Lname;
```

10. Select all staff members SSN, Fname, DepartmentName, Salary in ascending order by their Depatment, then by their salary in Descending order:

```
select Ssn, Fname, Dname , Salary
from DEPARTMENT, EMPLOYEE
where Dno = Dnumber
order by Dname ASC, Salary DESC;
```

11. What is the name of the department with the highest department number?

SELECT Dname , Dnumber
FROM DEPARTMENT
ORDER BY Dnumber DESC LIMIT 1;

12. Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name

SELECT D.Dname, E.Lname, E.Fname, P.Pname
FROM DEPARTMENT D, EMPLOYEE E, WORKS_ON W, PROJECT P
WHERE D.Dnumber= E.Dno AND E.Ssn= W.Essn AND W.Pno= P.Pnumber
ORDER BY D.Dname, E.Lname, E.Fname;

RESULT: Successfully executed the queries using SQL DML Commands.

ExpNo:8

NESTED QUERIES , JOIN QUERIES AND SET OPERATORS

AIM: To perform nested Queries , joining Queries and set operations using DML command

QUERIES

1. Display all employee names and salary whose salary is greater than minimum salary of the company

select Fname,Lname,Salary
from EMPLOYEE
where Salary>(select min(Salary) from EMPLOYEE);

2. Issue a query to display information about employees who earn more than any employee in dept no 5

*select * from EMPLOYEE*
where Salary>(select min(Salary) from EMPLOYEE where Dno=5);

3. Display the details of those who draw the salary greater than the average salary.

*select distinct **
from EMPLOYEE
where Salary >= (select avg(Salary) from EMPLOYEE);

4. Write SQL Query which retrieves the name and address of every employee who

works for the Research Department

```
select Fname, Lname, Address
from EMPLOYEE, DEPARTMENT
where Dno = Dnumber and Dname = 'Research';
```

5. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Select E.Fname, E.Lname
From EMPLOYEE as E
where E.Ssn in ( Select Essn From DEPENDENT as D where
E.Fname=D.Dependent_Name and E.Sex=D.Sex );
```

6. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
select Pno from WORKS_ON, EMPLOYEE
where Essn = Ssn and Lname = 'Smith'
UNION
select Pnumber from PROJECT P, DEPARTMENT D, EMPLOYEE E where
P.Dnum = D.Dnumber and D.Mgr_ssn = E.Ssn and E.Lname = 'Smith';
```

7. Write a query to display the name for all employees who work in a department with any employee whose Fname contains the letter 'h'

```
Select Fname from EMPLOYEE where Dno IN (Select Dno from EMPLOYEE
where Fname LIKE '%h%');
```

8 Retrieve all employees whose address Starts with Houston.

```
select Fname, Lname, Address
from EMPLOYEE
where Address LIKE 'Houston%';
```

9. Retrieve all employees whose address is Ends with Houston..

```
select Fname, Lname, Address
from EMPLOYEE
where Address LIKE '%Houston';
```

10. Find all employees who were born during the 1960s.

```
select Fname, Lname
from EMPLOYEE
where Bdate LIKE '__6_____';
```

11. Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

 # This is the use of in between

```
SELECT *  FROM EMPLOYEE
WHERE (Salary BETWEEN 30000 AND 40000) AND Dno = 5;
```

 # this is euquqlent to <= and > =

```
SELECT *  FROM EMPLOYEE
WHERE (Salary >=  30000 AND Salary <= 40000) AND Dno = 5;
```

12. Write a SQL query to find those employees who work in the same department where 'Ramesh' works.

 # Exclude all those records where first name is 'Ramesh'. Return first name, last name

```
select Fname, Lname, Dno  from EMPLOYEE where dno = (select dno from
EMPLOYEE where Fname = 'Ramesh') and Fname <> 'Ramesh';
```

13 Display all the dept numbers available in Emp and not in dept tables

Minus is no more supported in mysql

```
select Dno
from EMPLOYEE left join  DEPARTMENT on Dno = Dnumber
where Dnumber is NULL;
```

14. Display all the dept numbers available in dept and not in Emp tables

```
select Dnumber
from EMPLOYEE right join  DEPARTMENT on Dno = Dnumber
where Dno is NULL;
```

15. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
select Pnumber, Dnum, Lname, Address, Bdate
from PROJECT, DEPARTMENT, EMPLOYEE
where Dnum=Dnumber and Mgr_ssn=Ssn and Plocation='Stafford';
```

16. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
# only employees who have a supervisor are included in the result
# this is SELF JOIN
select E.Fname, E.Lname, S.Fname, S.Lname
from  EMPLOYEE AS E, EMPLOYEE AS S
where E.Super_ssn = S.Ssn;
```

17. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor, including those who have no immediate supervisors

```
select E.Fname, E.Lname, S.Fname, S.Lname
from  EMPLOYEE AS E left join EMPLOYEE AS S on  E.Super_ssn = S.Ssn;
```

18. List the details of employees having no immediate supervisor.

```
select *
from EMPLOYEE
where Super_ssn IS NULL;
```

19. Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

#This is use of arithmetic expression in select clause

```
select E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
from  EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
where E.Ssn=W.Essn AND W.Pno=P.Pnumber AND  P.Pname='ProductX';
```

20. List the first name and last name of all employees who work in the same department as the manager with last name 'Wong',

select E.Fname, E.Lname from EMPLOYEE E where E.Dno = (select D.Dnumber from DEPARTMENT D where D.Mgr_ssn = (select E2.Ssn from EMPLOYEE E2 where E2.Lname = 'Wong'));

RESULT

The query was executed and output was successfully obtained.

Exp No-9
TCL COMMANDS

AIM: Implementation of SQL TCL commands Rollback, Commit, Savepoint.

TRANSACTIONAL CONTROL LANGUAGE (TCL): A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be mode permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either role back or commit statement.

COMMIT

The basic syntax for using a COMMIT command in SQL is as follows :

BEGIN;

{a set of SQL statements};

COMMIT;

A more simplified version of syntax for other relational databases like MYSQL is as follows :

{a set of SQL statements};

COMMIT;

SAVE POINT & ROLL BACK:

Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with role back.

Syntax: SQL>SAVE POINT ID;

Example: SQL>SAVE POINT xyz;

ROLL BACK:

A role back command is used to undo the current transactions. We can role back the entire transaction so that all changes made by SQL statements are undo (or) role back a transaction to a save point so that the SQL statements after the save point are role back.

Syntax:

ROLE BACK(current transaction can be role back)

ROLE BACK to save point ID;

Experiment 1. Commit

/*

In MySQL, the statement `SET autocommit=0;` is used to disable the autocommit feature. By default, autocommit is **enabled**, which means that each SQL statement is automatically committed as a separate transaction.

***/**

SET autocommit=0;

/*

Create Employee table

*/

```
CREATE TABLE Emp(EMP_no int primary key,
                 Emp_name varchar(10),
                 Job varchar(10),
                 Hiredata date,
                 Salary float,
                 Comm Float,
                 Depno int references Dept(department_id));
```

```
INSERT INTO Emp VALUES(1,'Steven', 'Marketing', STR_TO_DATE('06-jan-1995', '%d-%M-%Y'),24000, NULL,2);
INSERT INTO Emp VALUES(2,'Neena', 'FI_ACCOUNT', STR_TO_DATE('06-feb-1987', '%d-%M-%Y'),34000, NULL,1);
INSERT INTO Emp VALUES(3,'Lex', 'FI_MGR', STR_TO_DATE('06-jan-1980', '%d-%M-%Y'),240000, NULL,1);
INSERT INTO Emp VALUES(4,'Alexander', 'Sa_Rep', STR_TO_DATE('06-jun-1987', '%d-%M-%Y'),20000, NULL,4);
INSERT INTO Emp VALUES(5,'Bruce', 'IT_PROG',STR_TO_DATE('06-jul-1990', '%d-%M-%Y'),24000, NULL,4);
```

start transaction ;

```
INSERT INTO Emp VALUES(6,'Jack','Clerk', STR_TO_DATE('06-aug-
1980', '%d-%M-%Y'),240000, NULL,5);
UPDATE Emp SET Job = 'FI_MGR' where EMP_no = 5;
```

commit;

```
select * from Emp;
```

Experiment 2. Savepont & Rollback

```
SET autocommit=0;
start transaction;
update Emp set Salary = Salary + 1000 where EMP_no = 6;
# create savepoint
SAVEPOINT emp_save_point1;
INSERT INTO Emp VALUES(7,'Girish','Clerk', STR_TO_DATE('06-aug-
1980', '%d-%M-%Y'),240000, NULL,5);
# verify Girish is added ;
select * from Emp;
# rollback
ROLLBACK TO SAVEPOINT emp_save_point1;
#commit transaction
commit;
# verify Girish is removed on rollback;
select * from Emp;
```

Experiment 3. Commit

```
SET autocommit=0;
start transaction;
update Emp set Salary = Salary + 1000 where EMP_no = 6;
# create savepoint
SAVEPOINT emp_save_point1;
INSERT INTO Emp VALUES(7,'Girish','Clerk', STR_TO_DATE('06-aug-
1980', '%d-%M-%Y'),240000, NULL,5);
# verify Girish is added ;
select * from Emp;
#commit transaction
commit;
# verify Girish is NOT removed on rollback;
# rollback
ROLLBACK TO SAVEPOINT emp_save_point1;
```

Output

mysql> ROLLBACK TO SAVEPOINT emp_save_point1;
ERROR 1305 (42000): SAVEPOINT emp_save_point1 does not exist

select * from Emp;

This will list all employee including Girish, because Girish is already commintted.

Exp No-10

DCL COMMANDS FOR GRANT AND REVOKE

AIM:Implementation of DCL commands Grant and Revoke

A privilege is a right to execute an SQL statement or to access another user's object.

For eg, In Oracle, there are two types of privileges

System Privileges

Object Privileges

System Privileges

are those through which the user can manage the performance of database actions. It is normally granted by DBA to users. Eg: Create Session,Create Table,Create user etc.

Object Privileges

allow access to objects or privileges on object, i.e. tables, table columns.tables,views etc..It includes alter,delete,insert,select update etc. (After creating the user, DBA grant specific system privileges to user)

GRANT Command

It is employed to grant a privilege to a user. GRANT command allows specified users to perform specified tasks

Syntax

GRANT privilege_name on objectname to user;

REVOKE Command

It is employed to remove a privilege from a user. REVOKE helps the owner to cancel previously granted permissions.

Syntax

REVOKE privilege_name on objectname from user

check current user

```
select user();
# create new user
create database testdcl;
use testdcl;

# create a table
create table Emp(EMP_no int primary key, Emp_name varchar(10), Job varchar(10), Hiredata date, Salary float, Comm Float, Depno int );

show tables;

# create a new user
create user 'user_test@localhost' identified by "Test@123";

# list users list and verify user_test exists.
select user from mysql.user;

# check what all privileges allotted by default on creating new user.
show grants for 'user_test@localhost';

# grant all privileges to new user
grant select on *.* to 'user_test@localhost';

# grant privileges to a specific table for (insert)
grant insert on testdcl.Emp to 'user_test@localhost';

# grant update privilege to a specific attribute on a table
grant update (Emp_name) on testdcl.Emp to 'user_test@localhost';

#show grants for new user
show grants for 'user_test@localhost';

#Verify
# logoff mysql and login using below command, it will prompt for password and enter the same password.
mysql -u 'user_test@localhost' -p

# verify user is user_test
slect user();
```

```
# list the grants
show grants for 'user_test@localhost';

# select databse and list tables
show databases;
use testdcl;
show tables;

# insert into tables
INSERT INTO Emp VALUES(1,'Steven', 'Marketing', STR_TO_DATE('06-jan-1995', '%d-%M-%Y'),24000, NULL,2);

select * from Emp;

# logoff mysql and login to super user,
mysql -u root -p

#change password of new user
alter user 'user_test@localhost' identified by '123456#';

# verify by logging with new password.
#logoff and login with user_test with newly modified password.
mysql -u 'user_test@localhost' -p
```

Experiment 11: Views in SQL

AIM: Demonstrate View in SQL

1. Create Table / May be you can use the same Table as that of **Expt No 4**

```
SET FOREIGN_KEY_CHECKS=0;

# creating base table

CREATE TABLE EMPLOYEE(
                Fname       VARCHAR(10)   NOT NULL,
                 Minit      CHAR,
```

```
Lname          VARCHAR(20)      NOT NULL,
Ssn            CHAR(9)          NOT NULL,
Bdate          DATE,
Address        VARCHAR(30),
Sex            CHAR(1),
Salary         DECIMAL(5),
Super_ssn      CHAR(9),
Dno            INT              NOT NULL,
PRIMARY KEY    (Ssn));
```

2. Populate table/ May be you can use same data of **Expt no 4**

```
INSERT INTO EMPLOYEE VALUES
('John','B','Smith',123456789,'1965-01-09','731 Fondren, Houston TX','M',30000,333445555,5),
('Franklin','T','Wong',333445555,'1965-12-08','638 Voss, Houston TX','M',40000,888665555,5),
('Alicia','J','Zelaya',999887777,'1968-01-19','3321 Castle, Spring TX','F',25000,987654321,4),
('Jennifer','S','Wallace',987654321,'1941-06-20','291 Berry, Bellaire TX','F',43000,888665555,4),
('Ramesh','K','Narayan',666884444,'1962-09-15','975 Fire Oak,  TX','M',38000,333445555,5),
('Joyce','A','English',453453453,'1972-07-31','5631 Rice, Houston TX','F',25000,333445555,5),
('Ahmad','V','Jabbar',987987987,'1969-03-29','980 Dallas, Houston TX','M',25000,987654321,4),
('James','E','Borg',888665555,'1937-11-10','450 Stone, Houston TX','M',55000,null,1);
```

3. Create View

#View-1. create view Emp(Ssn, Fname, Salary, Dno)

```
CREATE VIEW Emp AS SELECT Ssn,Fname,Salary, Dno FROM EMPLOYEE;
```

4. Check data in view Emp

```
select * from Emp;
```

5. Create another View

#View-2 create view Emp_sal(Ssn, Fname, Salary) where salary > 25000

```
CREATE VIEW Emp_sal AS SELECT Ssn,Fname,Salary FROM EMPLOYEE where salary >
25000;
```

6. Check data in view Emp-Sal

```
select * from Emp_sal;
```

7. Search view

select employee details of Fname = John

select * from Emp where Fname = 'John';

8. UODATE Emp View

update view Emp , change Dno to 1 for Emp name =john

update Emp SET Dno = 1 where Fname = 'John';

9. Check the updated data in view

select * from Emp where Fname = ‘ John’;

10. Check if the source table is updated or not

select * from EMPLOYEE where Fname = ‘ John’;

Note base table got changed..!

11. Update Base table, and check if View is updated or not

Change DNo of John to 4

update EMPLOYEE SET Dno = 4 where Fname = 'John';

12. Check if View got changes

select * from Emp where Fname = ‘ John’;

Note view got changed..!

13. Delete View

Drop view Emp_sal;

14. Drop Base Table

Drop table EMPLOYEE;

15. Verify deletion of base table will automatically delete view

```
SHOW CREATE VIEW Emp_sal;
```

CYCLE 2 – PL/SQL

Exp No:12
PL/SQL Introduction.

AIM: To implement various various control structures like IF-THEN,IF-THEN-ELSE,IF-THEN ELSIF,CASE ,WHILE USING PL/SQL

Procedural Language/Structured Query Language (PL/SQL) is an extension of SQL

Basic Syntax of PL/SQL

```
DECLARE
/* Variables can be declared here */
BEGIN
/* Executable statements can be written here */
EXCEPTION
/* Error handlers can be written here. */
END;
```

As we want output of PL/SQL Program on screen, before Starting writing anything type (Only Once per session)

SET SERVEROUTPUT ON

Sample 1: Hello World Program

```
DECLARE
   age integer;
   name VARCHAR(20);
BEGIN
   dbms_output.put_line('Hello world');
   --dbms_output.put_line('age = ' || age);
   --dbms_output.put_line('name = ' || name);
   --insert into Stud values(&rollno, '&name', &mark1, &mark2, &mark3);
END;
/
```

Sampe 2: **Find the largest of two integers.**

(Use of “If else” in PL/SQL)

```
DECLARE
   a integer := &a;
   b integer := &b;
BEGIN
  if (a > b) then
      dbms_output.put_line(a || ' is the largest number');
```

```
  else
     dbms_output.put_line(b || ' is the largest number');
  end if;
END;
```

Sample 3: **Print the range of two integers.**

(Use of if elsif ladder)

```
DECLARE
  c integer := &c;
BEGIN
  if (c >= 0 and c < 10) then
     dbms_output.put_line(' is less than 10');
  elsif (c >= 10 and c < 20) then
     dbms_output.put_line(' is less than 20');
  elsif (c >= 20 and c < 30) then
     dbms_output.put_line(' is less than 30');
  else
     dbms_output.put_line(' is grater than or equal 30');
  end if;
END;
```

Sample 4*: *Print the performance rating.

(Use of case statement)

```
DECLARE
  c char(1) := '&c';
BEGIN
  case c
     when 'A' then dbms_output.put_line('Excellent');
     when 'B' then dbms_output.put_line('Very good');
     when 'C' then dbms_output.put_line('Well done');
     when 'D' then dbms_output.put_line('You passed');
     when 'F' then dbms_output.put_line('Better try again');
     else dbms_output.put_line('No such grade');
  end case;
END;
```

Sample 5: Use of Array and Loops in PL/SQL

Please note:

- Default index starts from 1

- Declared using the TYPE keyword

```
DECLARE
   type intArray IS VARRAY(10) OF INTEGER;
   type namesArray IS VARRAY(5) OF VARCHAR2(5);

   arr intArray;
   names namesArray;
   i integer;

BEGIN
   arr :=  intArray(1,5,2,3,6,7,4,8,9,10);
   names := namesArray('Alice', 'Bob', 'Cindy', 'Sam', 'Eric');

   i := 1;
   /*While loop...! */
   while( i <= 10) loop
      dbms_output.put_line('arr[' || i ||'] =' ||arr(i));
      i := i+1;
   end loop;

   /*For loop...! */
   for i in 1 .. 10 loop
      dbms_output.put_line('arr[' || i ||'] =' ||arr(i));
   end loop;
  /* while loop */
   i := 1;
   while( i <= 5) loop
      dbms_output.put_line('names[' || i ||'] =' ||names(i));
      i := i+1;
   end loop;

   /*For loop...! */
   for i in 1 .. 5 loop
      dbms_output.put_line('names[' || i ||'] =' ||names(i));
   end loop;
END;
```

Question 1: Write a plsql program to check whether a given number is ODD or EVEN

```
DECLARE
   number integer;
BEGIN
   -- get role no from user
```

```
    number := &number;

    --calculate & print result
    if (mod(number,2) = 0) then
        dbms_output.put_line('EVEN Number');
    else
        dbms_output.put_line('ODD Number');
    end if;
END;
/
```

Question 2: Write a PL/SQL block to find the maximum number from given three numbers.

```
DECLARE
    number1 integer := &number1;
    number2 integer := &number2;
    number3 integer := &number3;

BEGIN
    --calculate & print result
    if (number1 > number2 and number1>number3) then
        dbms_output.put_line('Greatest is ' || number1);
    elsif (number2 > number2 and number2>number3) then
        dbms_output.put_line('Greatest is ' || number2);
    else
        dbms_output.put_line('Greatest is ' || number3);
    end if;
END;
/
```

Question 3: Write a program to accept a number and find the sum of the digits

```
DECLARE
    num integer :=&num;
    total integer:=0;
    digit integer:=0;
BEGIN
    while (num !=0) loop
        digit := mod(num,10);
        total := total + digit;
        num := trunc(num/10);
    end loop;
    dbms_output.put_line('sum of digits of given number is '||total);
```

```
END;
/
```

Question 4: Write a program to accept a number and find the sum of the digits

```
DECLARE
   num integer :=&num;
   total integer:=0;
   digit integer:=0;
BEGIN
   while (num !=0) loop
      digit := mod(num,10);
      total := total + digit;
      num := trunc(num/10);
   end loop;
   dbms_output.put_line('sum of digits of given number is '||total);
END;
/
```

Question 5: Program to print the days names in the week.

```
DECLARE
d number:=&num1;
BEGIN
case d
  when 1 then
    dbms_output.put_line('sunday');
  when 2 then
    dbms_output.put_line('monday');
  when 3 then
    dbms_output.put_line('tuesday');
  when 4 then
    dbms_output.put_line('wednesday');
  when 5 then
    dbms_output.put_line('thursday');
  when 6 then
    dbms_output.put_line('friday');
  when 7 then
    dbms_output.put_line('saturday');
  else
    dbms_output.put_line('invalid day');
end case;
END;
/
```

Exp No-13
Creation of Procedures, Triggers and Functions

Creation of Triggers, Procedures and Functions

AIM: Design and implement Procedure, Triggers, and Functions

I. PROCEDURE
In PL/SQL, a procedure is a named block of code that can be stored and executed in an Oracle database. It is a subroutine that performs a specific task or a series of tasks.

Features
1. Procedure Syntax
2. No return value
3. Parameters

a. **Procedure Syntax**

1. **Create a procedure**

```
CREATE [OR REPLACE] PROCEDURE procedure_name  [ (parameter [,parameter]) ]
IS
[declaration_section]
BEGIN
executable_section
[EXCEPTION
exception_section]
END [procedure_name];
```

2. **Delete a procedure**

```
DROP PROCEDURE procedure_name;
```

b. Parameters

There are three types of parameters that can be declared:
1. IN - The parameter can be referenced by the procedure or function.
   The value of the parameter can not be overwritten by the procedure or function.
2. OUT - The parameter can not be referenced by the procedure or function,
   but the value of the parameter can be overwritten by the procedure or function.
3. IN OUT - The parameter can be referenced by the procedure or function and the
   value of the parameter can be overwritten by the procedure or function.

c. How differ from functions

functions in PL/SQL are similar to procedures, but they return a value to the calling program.

Functions are defined using the CREATE FUNCTION statement

d. How to use
a. Compile the procedure from a .sql script
b. Write a pl/sql program in another script and call procedure from it.

Question 1: Write a PL/SQL Procedure to find largest of two Numbers

```
CREATE OR REPLACE PROCEDURE FindLargest (
   num1 in integer,
   num2 in integer,
   largest out integer
) AS
BEGIN
   IF num1 > num2 THEN
      largest := num1;
   ELSE
      largest := num2;
   END IF;
END;
/
```

-- How to use this procedure from a pl/sql program

```
DECLARE
   num1 integer := 10;
   num2 integer := 20;
   result integer;
BEGIN
   -- invoke procedure, just like function call
   FindLargest(num1, num2, result);
   dbms_output.put_line('The largest number is: ' || result);
END;
/
```

-- Delete Procedure

```
DROP PROCEDURE FindLargest;
/
```

Question 2: Create a table EMPLOYEE(Eid,Ename,salary,Dept,Wef). Write a procedure to accept two arguments Eid and Salary increment(in %). update the EMPLOYEE table with the salary increment also record the effective date.

```
-- create table
CREATE TABLE EMPLOYEE (Eid INT, Ename VARCHAR2(100), Salary NUMBER,
Dept VARCHAR2(100), Wef DATE);

-- insert few data
INSERT INTO EMPLOYEE (Eid, Ename, Salary, Dept, Wef)
VALUES (1, 'Alice', 5000, 'Finance', TO_DATE('2022-06-11', 'YYYY-MM-DD'));

INSERT INTO EMPLOYEE (Eid, Ename, Salary, Dept, Wef)
VALUES (2, 'Bob', 6000, 'HR', TO_DATE('2022-06-11', 'YYYY-MM-DD'));

INSERT INTO EMPLOYEE (Eid, Ename, Salary, Dept, Wef)
VALUES (3, 'Cindy', 7000, 'Sales', TO_DATE('2022-06-11', 'YYYY-MM-DD'));

INSERT INTO EMPLOYEE (Eid, Ename, Salary, Dept, Wef)
VALUES (4, 'Sam', 5500, 'Finance', TO_DATE('2022-06-11', 'YYYY-MM-DD'));
```

```
INSERT INTO EMPLOYEE (Eid, Ename, Salary, Dept, Wef)
VALUES (5, 'Eric', 7500, 'HR', TO_DATE('2022-06-11', 'YYYY-MM-DD'));

-- default name format 'DD-MON-YY' which is different from mysql
INSERT INTO EMPLOYEE (Eid, Ename, Salary, Dept, Wef)
VALUES (5, 'Alex', 7500, 'HR', '11-MAY-22');

-- Procedure definition
CREATE OR REPLACE PROCEDURE UpdateSalary(
   v_Eid IN EMPLOYEE.Eid%TYPE,
   v_Increment IN NUMBER
) AS
BEGIN
   UPDATE EMPLOYEE
   SET Salary = Salary + (Salary * v_Increment / 100), Wef = SYSDATE
   WHERE Eid = v_Eid;
   dbms_output.put_line('UpdateSalary:: Salary updated for Employee ID: ' || v_Eid);
END;

-- How to use this procedure from a pl/sql program
DECLARE
   v_Eid NUMBER := &v_Eid;
   Percentage NUMBER := &Percentage;
   v_Salary EMPLOYEE.Salary % TYPE;
   v_Ename EMPLOYEE.Ename % TYPE;
BEGIN
   UpdateSalary(v_Eid, Percentage);
   select Salary, Ename into v_Salary, v_Ename from EMPLOYEE where Eid = v_Eid;
   dbms_output.put_line('Emp Name : ' || v_Ename);
   dbms_output.put_line('New Salary : ' || v_Salary);
END;
/
```

Question 3: Print all employees in a given department.

```
CREATE OR REPLACE PROCEDURE PrintEmployeesByDepartment(
   v_Department IN EMPLOYEE.Dept%TYPE
) AS
BEGIN
   FOR emp IN (select * from EMPLOYEE where Dept = v_Department) LOOP
      dbms_output.put_line('Employee ID: ' || emp.Eid);
      dbms_output.put_line('Employee Name: ' || emp.Ename);
      dbms_output.put_line('Salary: ' || emp.Salary);
      dbms_output.put_line('Department: ' || emp.Dept);
      dbms_output.put_line('Effective Date: ' || TO_CHAR(emp.Wef, 'DD-MON-YYYY'));
      dbms_output.put_line('--------------------------');
   END LOOP;
END;
/
```

-- How to use this procedure from a pl/sql program

```
DECLARE
   v_Department EMPLOYEE.Dept%TYPE := &v_Department;
BEGIN
   PrintEmployeesByDepartment(v_Department);
END;
```

II. TRIGGER

A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA.. A database that has a set of associated triggers is called an Active Database.

A trigger description contains three parts:

Event : A change to the database that activates the trigger.
Condition : A query or test that is run when the trigger is activated.
Action : A procedure that is executed when the trigger is activated and its condition is true.

An insert, delete, or update statement could activate a trigger, regardless of which user or application invoked the activating statement; users may not even be aware that a trigger was executed as a side effect of their program.

Question 1. create a trigger which shows the salary difference of a particular employee whenever salary is getting updated

-- trigger definition

```
CREATE OR REPLACE TRIGGER Salary_Update_Trigger
BEFORE UPDATE OF Salary ON EMPLOYEE
FOR EACH ROW
DECLARE
  old_salary NUMBER;
  new_salary NUMBER;
BEGIN
  IF :OLD.Eid = :NEW.Eid THEN
    old_salary := :OLD.Salary;
    new_salary := :NEW.Salary;
      DBMS_OUTPUT.PUT_LINE('Salary difference for Employee ' || :NEW.Eid || ': ' ||
(new_salary - old_salary));
  END IF;
END;
/
```

-- How to test

```
-- Reuse the EMPLOYEE data in last program,
-- If data set is empty, insert data as in last program
-- Execute below pl.sql program, that update employee salary
```

```
DECLARE
   e_Eid EMPLOYEE.Eid%TYPE := &e_Eid;
   newSalary EMPLOYEE.Salary % TYPE := &newSalary;

BEGIN
   update  EMPLOYEE SET Salary = newSalary WHERE Eid = e_Eid;
END;
```

Question 2: Create a table THEATER(MovieId, movieName, Language, ReviewRatings). Whenever rating goes below 5 , the movies has to be removed from THEATER table and add to table OUTDATED_MOVIES with attributes MovieId and MovieName. Write a trigger for above problem

```
-- create table
CREATE TABLE THEATER (MovieId INT, MovieName VARCHAR(255), Language
VARCHAR(255),
ReviewRatings NUMBER);

-- Insert values into THEATER table
INSERT INTO THEATER (MovieId, MovieName, Language, ReviewRatings)
VALUES (101, 'Junglebook', 'English', 6);

INSERT INTO THEATER (MovieId, MovieName, Language, ReviewRatings)
VALUES (501, 'Parava', 'Malayalam', 8);

INSERT INTO THEATER (MovieId, MovieName, Language, ReviewRatings)
VALUES (601, 'OSO', 'Hindi', 9);

INSERT INTO THEATER (MovieId, MovieName, Language, ReviewRatings)
VALUES (701, 'Avengers', 'English', 9);

INSERT INTO THEATER (MovieId, MovieName, Language, ReviewRatings)
VALUES (801, 'Hobbit', 'English', 7);

INSERT INTO THEATER (MovieId, MovieName, Language, ReviewRatings)
VALUES (901, 'Don', 'Hindi', 6);

CREATE TABLE OUTDATED_MOVIES (MovieId INT, MovieName VARCHAR(255));

/* Triger */
CREATE OR REPLACE TRIGGER Remove_From_Theater
AFTER DELETE ON THEATER
FOR EACH ROW

BEGIN
   DBMS_OUTPUT.PUT_LINE('Movie id : ' || :OLD.MovieId);
   DBMS_OUTPUT.PUT_LINE('MovieName : ' || :OLD.MovieName);
   insert into OUTDATED_MOVIES values (:OLD.MovieId, :OLD.MovieName);
END;
```

```
-- How to use this trigger
DECLARE
   v_MovieId number := &v_MovieId;
   v_Rating number := &v_Rating;

BEGIN
   if (v_Rating < 5) then
      delete from THEATER where MovieId = v_MovieId;
   end if;
END;
/
```

Question 3

Create a database trigger to calculate the fine based on the rules given below.

- **After 1 month 5% of price**
- **After 2 month 10% of price**
- **After 3 month 20% of price.**

Schema:

Book_avail (bookid, title, no_of _copies, price)
Student (st_id,name,class,fine)
Issue_tab (st_id, book_id, issuedate, returndate)

```
-- create table
create table Book_avail(bookid int primary key, title varchar(20), no_of_copies int, price
int);
create table Student(st_id int primary key, name varchar(20), class varchar(10), fine int);
create table Issue_tab(st_id int, book_id int, issue_date date , return_date date, primary
key(st_id, book_id));

-- insert few data
insert into Student values(100, 'Alice', 'CSE', 0);
insert into Student values(101, 'Bob', 'CSE', 0);
insert into Book_avail values(1, 'Data Structure', 10, 1000);
insert into Book_avail values(2, 'Java - Complete ref', 10, 1000);

insert into Issue_tab values(100, 1, TO_DATE('2022/01/01','%yyyy-%mm-%dd'),
TO_DATE('2022/02/01','yyyy-%mm-%dd'));
insert into Issue_tab values(101, 2, TO_DATE('2022/01/01','%yyyy-%mm-%dd'),
TO_DATE('2022/03/01','yyyy-%mm-%dd'));
select * from Issue_tab;
select * from Student;
select * from Book_avail;

/* trigger */
CREATE OR REPLACE TRIGGER Calc_Late_Fine
AFTER UPDATE ON Issue_tab
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
DECLARE
   v_bprice int;
   v_months int;
```

```
    v_latefine int;
    v_bookid int;
BEGIN
    v_bookid := :OLD.book_id;

    -- get book price from book table
    select price into v_bprice from Book_avail where bookid = v_bookid;

    dbms_output.put_line('v_bookid = ' || v_bookid);
    dbms_output.put_line('v_bprice = ' || v_bprice);

    -- get months after student returning book
    v_months:=months_between(:new.return_date,:old.issue_date);
    dbms_output.put_line('v_months = ' || v_months);

    -- Calculate fine while update issue_tab
    if v_months>=1 and v_months<2 then
        v_latefine := v_bprice*0.05;
    else if v_months>=2 and v_months<3 then
        v_latefine := v_bprice*0.01;
    else if v_months>=3 then
        v_latefine := v_bprice*0.2;
    end if;
    end if;
    end if;

    -- Update fine into Student table while update issue_tab
    dbms_output.put_line('v_latefine = ' || v_latefine);
    update Student set fine=v_latefine where st_id=:old.st_id;
END;
/
```

-- How to test this trigger

```
update Issue_tab set return_date=TO_DATE('2022/04/02','%yyyy-%mm-%dd')
where st_id=100;
```

Question 4

consider the bank databse

account (account_number, branch_name , balance);
depositor (customer_name, account_number);
customer (customer_name, customer_address);

Write SQL Trigger to carryout following action

On delete of an account, for each customer-owner of the account, check if the owner has any remaining account, and if she/he does not, delete her/him from depositor relation.

-- create tables & populate table

```
CREATE TABLE account (account_number INT PRIMARY KEY, branch_name
VARCHAR(255), balance int);
CREATE TABLE depositor (customer_name VARCHAR(255), account_number INT,
```

```
FOREIGN KEY (account_number) REFERENCES account(account_number));
CREATE TABLE customer (customer_name VARCHAR(255) PRIMARY KEY,
customer_address VARCHAR(255));

insert into account values(1,'TVM',100000);
insert into account values(2,'TVM',200000);
insert into account values(3,'TVM',300000);
insert into account values(4,'TVM',400000);
insert into account values(5,'Cochin',500000);

insert into customer values('Alice', 'Address1');
insert into customer values('Bob', 'Address2');
insert into customer values('Cindy', 'Address3');

insert into depositor values('Alice', 1);
insert into depositor values('Alice', 2);
insert into depositor values('Alice', 3);
insert into depositor values('Bob', 4);
insert into depositor values('Cindy', 5);

DELIMITER //
CREATE  TRIGGER Trigger1234
AFTER DELETE ON account
FOR EACH ROW
BEGIN
declare
        c_name varchar(30);
        select customer_name into c_name from depositor where  account_number =
old.account_number;

        if (select count(*) from depositor d where d.customer_name = c_name ) > 1 then
                signal sqlstate '45000' set message_text = 'Error..!  Delete Not Allowed in
depositor table: Customer has more accounts ';
        else
                DELETE FROM depositor WHERE account_number =
old.account_number;
        end if;
END
//
DELIMITER ;

-- Testing trigger
DELETE FROM account WHERE account_number = 2;

-- output
mysql> DELETE FROM account WHERE account_number = 2;
ERROR 1644 (45000): Error..!  Delete Not Allowed in depositor table: Customer has
more accounts
```

III FUNCTIONS

AIM: To implement programs using functions.

A standalone function is created using the CREATE FUNCTION statement.

The syntax is given by

```
CREATE [OR REPLACE] FUNCTION function_name ((parameter_name {IN} type
        {,  ...}))
RETURN return_datatype
{IS | AS}
        <declaration section>
BEGIN
        < function_body >
END;
```

Where,

- function-name specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside The function must contain a return statement.
- The RETURN clause specifies the data type you are going to return from the function. · Function-body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.
- While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function.
To call a function, you simply need to pass the required parameters along with the function name and if the function returns a value, then you can store the returned value.

Question 1: findMax value of two integers.

```
-- function definition
CREATE OR REPLACE FUNCTION findMax(a IN NUMBER, b IN NUMBER)
RETURN NUMBER IS
BEGIN
   IF a > b THEN
      RETURN a;
   ELSE
      RETURN b;
   END IF;
END;
/

-- how to test this function
DECLARE
   max_value NUMBER;
   a int := &a;
   b int := &b;
```

```
BEGIN
   -- Test cases
   max_value := findMax(10, 20);
   dbms_output.put_line('Max of 10 and 20: ' || max_value);

   max_value := findMax(30, 15);
   dbms_output.put_line('Max of 30 and 15: ' || max_value);

   max_value := findMax(-5, -10);
   dbms_output.put_line('Max of -5 and -10: ' || max_value);

   max_value := findMax(0, 0);
   dbms_output.put_line('Max of 0 and 0: ' || max_value);

   max_value := findMax(a, b);
   dbms_output.put_line('Max of ' || a || ' and ' || b || ': ' || max_value);
END;
/
```

Exp No-14
Creation of Packages

AIM: *Creation of Packages*

Packages
A package is a schema object that groups logically related PL/SQL types, variables, constants, subprograms, cursors, and exceptions. A package is compiled and stored in the database, where many applications can share its contents.

A package always has a specification, which declares the public items that can be referenced from outside the package. If the public items include cursors or procedures or functions, then the package must also have a body. The body must define queries for public cursors and code for public functions and procedures. The body can also declare and define private items that cannot be referenced from outside the package, but are necessary for the internal workings of the package. You can change the body without changing the specification or the references to the public items; therefore, you can think of the package body as a black box.

Question 1
Create a PL/SQL Package with one procedure and one function.

Procedure: will take id and return the name corresponding to id as out parameter from Customer table
Function: will take id and return the annual salary of employee corresponding to id from Customer table

Ans:

-- Create Customer table and populate table

```
create table CUSTOMER (Cid int primary key, Cname varchar(30), Sex char(10), Age int,
Sal int);

INSERT INTO CUSTOMER (Cid, Cname, Sex, Age)
VALUES (1, 'Alice', 'Female', 25, 10000),
     (2, 'Bob', 'Male', 30, 20000),
     (3, 'Cindy', 'Female', 35, 30000),
     (4, 'Sam', 'male', 28, 40000);
-- package definition
-- package specification
CREATE OR REPLACE PACKAGE Customer_package AS
PROCEDURE Get_emp_name(c_id IN NUMBER,
                c_name OUT VARCHAR);
FUNCTION Get_annual_salary(c_id IN NUMBER) RETURN NUMBER;
END Customer_package;
/
-- package body
CREATE OR REPLACE PACKAGE BODY Customer_package AS
-- first procedure definition
PROCEDURE Get_emp_name(c_id IN NUMBER, c_name OUT VARCHAR)IS
BEGIN
        SELECT name INTO c_name from CUSTOMER where cid = c_id;
        END Get_emp_name;

-- second function definition
FUNCTION Get_annual_salary (c_id IN NUMBER)
RETURN NUMBER
IS
        c_sal NUMBER :=0;
        annual_sal NUMBER;
BEGIN
        SELECT Sal INTO c_sal FROM CUSTOMER WHERE cid = c_id;
        annual_sal := 12 * c_sal;
        RETURN (annual_sal);
END;
END Customer_package;
/

-- Test program
declare
        e_id number := &a1;
        e_name varchar(10);
        sal number :=0;
begin
        Customer_package.Get_emp_name(e_id, e_name);
        dbms_output.put_line('Customer corresponding to '||e_id || ' is '||
        e_name);
        sal := Customer_package.Get_annual_salary(e_id);
```

```
        dbms_output.put_line('Annual Salary of '||e_name || ' is : '|| sal);
end;
```

Exp No-15

Creation of Cursor

AIM Create a PL/SQL Cursors

Cursors

A cursor is a named control structure used by an application program to point to and select a row of data from a result set. Instead of executing a query all at once, you can use a cursor to read and process the query result set one row at a time.

Implicit Cursors:

An implicit cursor is a session cursor that is constructed and managed by PL/SQL. PL/SQL opens an implicit cursor every time you run a SELECT or DML statement. You cannot control an implicit cursor, but you can get information from its attributes.

Explicit Cursors:

Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

How to create Explicit Cursor:

Declare Cursor Object.

Syntax : DECLARE cursor_name CURSOR FOR SELECT * FROM table_name

DECLARE s1 CURSOR FOR SELECT * FROM studDetails

Open Cursor Connection.

Syntax : OPEN cursor_connection

OPEN s1

Fetch Data from cursor.

There are total 6 methods to access data from cursor. They are as follows :

FIRST is used to fetch only the first row from cursor table.

LAST is used to fetch only last row from cursor table.

NEXT is used to fetch data in forward direction from cursor table.

PRIOR is used to fetch data in backward direction from cursor table.ABSOLUTE n is used to fetch the exact nth row from cursor table.

RELATIVE n is used to fetch the data in incremental way as well as decremental way.

Syntax : FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor_name

FETCH FIRST FROM s1

FETCH LAST FROM s1

FETCH NEXT FROM s1

FETCH PRIOR FROM s1

FETCH ABSOLUTE 7 FROM s1
FETCH RELATIVE -2 FROM s1

Close cursor connection.

Syntax : CLOSE cursor_name
CLOSE s1

Deallocate cursor memory.

Syntax : DEALLOCATE cursor_name
DEALLOCATE s1

Question 1: Consider the table

Customer (accout no, customer name,balance amount,date of join).

Write a pl/sql program using cursor to display the id, name, age, Balance, and date of join of all employees in Customer table.

```
set serveroutput on;
create table Customer(id int primary key,  name varchar(20), age int, address varchar(30), balance int, doj date);
insert into Customer values(1, 'Alice', 20, 'Address1', 150000, TO_DATE('2009/01/01','%yyyy-%mm-%dd'));
insert into Customer values(2, 'Bob', 30, 'Address2', 200000, TO_DATE('2007/01/01','%yyyy-%mm-%dd'));
insert into Customer values(3, 'Cindy', 40, 'Address3', 30000, TO_DATE('2019/01/01','%yyyy-%mm-%dd'));
insert into Customer values(4, 'Sam', 50, 'Address4', 40000, TO_DATE('2018/01/01','%yyyy-%mm-%dd'));
insert into Customer values(5, 'Eric', 60, 'Address5', 11000, TO_DATE('2008/01/01','%yyyy-%mm-%dd'));
insert into Customer values(6, 'Tom', 20, 'Address6', 60000, TO_DATE('2016/01/01','%yyyy-%mm-%dd'));
insert into Customer values(7, 'John', 30, 'Address7', 70000, TO_DATE('2015/01/01','%yyyy-%mm-%dd'));
insert into Customer values(8, 'Sari', 40, 'Address8', 80000, TO_DATE('2014/01/01','%yyyy-%mm-%dd'));
insert into Customer values(9, 'Timo', 50, 'Address9', 90000, TO_DATE('2013/01/01','%yyyy-%mm-%dd'));
insert into Customer values(10, 'Skove', 60, 'Address10', 100000,
TO_DATE('2012/01/01','%yyyy-%mm-%dd'));
select * from  Customer;


DECLARE CURSOR cursor0 IS
SELECT id, name, age, balance , doj FROM Customer;
   --variable definition
   e_name Customer.name%type;
   e_id Customer.id%type;
   e_bal Customer.balance%type;
   e_age Customer.id%type;
   e_doj Customer.doj%type;
BEGIN
   -- open cursor
   OPEN cursor0;
   LOOP
      FETCH cursor0 INTO e_id, e_name, e_age, e_bal, e_doj;
      -- exit condition
      EXIT WHEN cursor0%NOTFOUND;
```

```
    -- print employee info
    dbms_output.put_line('Customer Id:'||e_id);
    dbms_output.put_line('Customer Name:'||e_name);
    dbms_output.put_line('Customer Age:'||e_age);
    dbms_output.put_line('Balance :'||e_bal);
    dbms_output.put_line('Date of Join:'||to_char(e_doj));
    dbms_output.put_line('============================');
  END LOOP;
  -- close cursor
  CLOSE cursor0;
END;
```

Question 2:
Consider the table Customer (accout no, customer name,balance amount,date of join). Implement a PL/SQL block to insert those customers who have
current balance greater than 1 Lakh
and date of join before 1 january 2010 into the table premium customer
who doesnt meet above criteria are to be inserted into table nonpremium customer .

```
create table Premium_Customers(id int primary key, name varchar(20), age int, address
varchar(30), balance int, doj date);

create table Nonpremium_Customers(id int primary key, name varchar(20), age int, address
varchar(30), balance int, doj date);

DECLARE CURSOR cursor1 IS
SELECT * FROM Customer;
BEGIN
  -- open cursor
  FOR c1 in cursor1
    LOOP
    if c1.balance > 100000 and c1.doj  < '01-JAN-2010' then
      dbms_output.put_line('=======Premium_Customers=======');
      dbms_output.put_line('Employee Name:'||c1.name);
      insert into Premium_Customers values(c1.id, c1.name, c1.age,
            c1.address, c1.balance, c1.doj);
    else
      dbms_output.put_line('=======Nonpremium_Customers=======');
      dbms_output.put_line('Employee Name:'||c1.name);
      insert into Nonpremium_Customers values(c1.id, c1.name, c1.age,
            c1.address, c1.balance, c1.doj);
    end if;
  END LOOP;
END;

select * from Premium_Customers;
```

*select * from Nonpremium_Customers;*

Question 3:
Consider the table Account(Customer name,account number,date_last transaction,amount).

Implement a PL/SQL block to perform the following action on the table .
Calculate the interest of each person if it satisfies the condition
a)if the last transaction is not on the current month
insert the records into inactive customer
b)otherwise check the balance amount and display the interest amount
i)if the balance amount is less than 50000 interest rate is 5% of the amount
ii)if it is between 250000 and 5 Lakhs interest rate is 10%
iii)if the amount is greater than 5 lakh interest rate is 15%

```
create table AccDetails(accno number(10),cname varchar(20),lastdate date,amount number(7));

insert into AccDetails values(101,'Alice','08-JAN-2017',50000);
insert into AccDetails values(102,'Bob','10-FEB-2017',100000);
insert into AccDetails values(103,'Cindy','17-MAR-2017',25000);
insert into AccDetails values(104,'Sam','06-APR-2017',300000);
insert into AccDetails values(105,'Eric','15-MAY-2017',650000);

create table InactiveCustomer(accno number(10),cname varchar(20));

DECLARE CURSOR cursor2 IS
SELECT * FROM AccDetails;
 iRow AccDetails %rowtype;
 v_month number;
 interest number;
BEGIN
   -- open cursor
   OPEN cursor2;
   LOOP
      FETCH cursor2 INTO iRow.accno, iRow.cname, iRow.lastdate, iRow.amount;
      -- exit condition
      EXIT WHEN cursor2%NOTFOUND;
      v_month := months_between(sysdate, iRow.lastdate);

      dbms_output.put_line('iRow.cname '||iRow.cname);
      dbms_output.put_line('iRow.lastdate '||iRow.lastdate);
      dbms_output.put_line('iRow.amount '||iRow.amount);
      dbms_output.put_line('v_month '||v_month);

      if v_month>=1 then
         dbms_output.put_line('Moving to InactiveCustomer ...!');
         insert into InactiveCustomer values(iRow.accno, iRow.cname);
      else
         if iRow.amount < 250000 then
            interest := iRow.amount*0.05;
```

```
            dbms_output.put_line('interest of '||iRow.cname|| ' is :'||interest);
        elsif iRow.amount>250000 and iRow.amount<500000 then
            interest := iRow.amount*0.1;
            dbms_output.put_line('interest of '||iRow.cname|| ' is :'||interest);
        elsif iRow.amount>500000 then
            interest := iRow.amount*0.15;
            dbms_output.put_line('interest of '||iRow.cname|| ' is :'||interest);
        else
            dbms_output.put_line('Error...!');
        end if;
      end if;
   END LOOP;
   -- close cursor
   CLOSE cursor2;
END;

select * from AccDetails;
select * from InactiveCustomer;
```

Exp No:16

Creation of plsql blocks for Exception Handling

AIM:To implement plsql blocks for exception handling

QUESTION

Sales of different products in one week is recorded
Product (productid,productname,grade)
Sales (prdctname,salesamount,salesday)

Do the following
a) write a function that displays the product name and grade of the given product b)whenever the product sales is grater than the target value it is given a A grade, if there is no sale for a product an exception to be raised

Query

```
create table product(pid number(5),pname varchar(10),pgrade varchar(3));

insert into product values(1,'HDD','a');
insert into product values(2,'GoPro','b');
insert into product values(3,'laptop','b');
insert into product values(4,'mobile','b');
insert into product values(5,'DVD','c');

create table sales(pid number(5),samount number(10),sdate date,sday
varchar(15)); insert into sales values(1,1000,'2-10-2017','Monday');
insert into sales values(3,1500,'4-10-2017','Wednesday');
insert into sales values(3,2000,'5-10-2017','Thursday');
insert into sales values(1,3500,'7-10-2017','Saturday');
insert into sales values(3,4000,'3-10-2017','Tuesday');

create function funct(a1 in number,a3 in number)
return number
as
        q number;
        z number;
        r number;
        x varchar(10);
        y varchar(10);
begin
        select pname into x from product where pid=a1;
        select pgrade into y from product where pid=a1;
```

```
        dbms_output.put_line('name of product :'||x);
        dbms_output.put_line('grade of product :'||y);

        select sum(samount) into q from sales where pid=a1;
        dbms_output.put_line('sum of sales of '||a1||' is'||q);
if(q>a3) then
        update product set pgrade='a' where pid=a1;
end if;
select count(pid) into z from sales where pid=a1;

if(z<1) then
        r:=0;
else
        r:=1;
end if;
        return(r);
end;

set serveroutput on;
declare
        d1 number;
        e1 number;
        f1 number;
        pnull exception;
begin
        d1:=&d1;
        e1:=&e1;
        f1:=funct(d1,e1);
If f1=0 then
        raise pnull;
end if;
        exception
        when pnull then
        dbms_output.put_line('no sales corresponding this
pid ');
        when no_data_found then
        dbms_output.put_line('no data found:');
end;
```

-- How to Test

```
select * from product;
anonymous block completed
Enter the value of d1 : 3
Enter the value of e1 : 5000
name of product :laptop
```

grade of product :b sum of sales of 3 is 7500 table updated.

*select * from product;*
anonymous block completed
Enter the value of d1 : 2
Enter the value of e1 : 1000
name of product :GoPro
grade of product :b
sum of sales of 2 is
no sales corresponding to this pid

Exp No:17

Familioarisation of NoSQL Database and CRUD Operations

Create the tables mentioned below using crude operation – MongoDB.

There is no "create" command in the MongoDB Shell.

In order to create a database, you will first need to switch the context to a non-existing database using the **use** command:

> show databases
#Or
> show dbs

switch to Bookstore
> use Bookstore

Create a collection - Books
However, In MongoDB, create a collection is not mandatory.
MongoDB creates collection automatically, when you insert some document.
Here we use createCollection explicitely..!
> db.createCollection("Books");

Verify if collection is created or not
> show collections

insert one document to Books
> db.Books.insertOne({"title": "Book3", "author": "author3", "pages": 400, "genere": ["magical", "realism"], "price": 400, "copies":2});

insert more than one one document to Books
> db.Books.insertMany([{"title": "Book4", "author": "author4", "pages": 400, "genere": ["magical", "realism"], "price": 400, "copies":5}, {"title": "Book5", "author": "author5", "pages": 400, "genere": ["magical", "realism"], "price": 400, "copies":5}]);

Create a new collection - "Notes" without using **createCollection** command.
MongoDB creates collection automatically, when you insert some document.
> db.Notes.insertOne({"title": "Book3", "author": "author3", "pages": 400, "genere": ["magical", "realism"], "price": 400, "copies":2})

List all collections
> show collections

Delete Notes from database using drop()
> db.Notes.drop()

verify drop is done for "Notes"
> show collections

insert one more document to Books
> db.Books.insertOne({"title": "Harry Potter 2", "author": "J K Rowling", "pages": 400, "genere": ["magical", "realism"], "price": 400, "copies":2})

Query1, find() method is used to Find all the documents present in the collection:.
> db.Books.find()

Query2 - finding all document with author = "J K Rowling"
> db.Books.find({"author": "J K Rowling"})

Query3 - finding all document with genere = "magical" and "realism" both
> db.Books.find({"genere": ["magical", "realism"]})

Query4 - finding all document with price = 400
> db.Books.find({"price": 400})

Query5 - finding all document with price = 400 and author = "J K Rowling"
> db.Books.find({author: "J K Rowling", price: 1000})

Query6 - finding document with unque object id
> db.Books.find({_id: ObjectId("63bb2e9593b418ae2251d144")})

Delete operation: Delete one document identified with unique object id
> db.Books.deleteOne({_id: ObjectId("63bb2e9593b418ae2251d144")})

Delete operation: Delete more than one document identified with author = "J K Rowling"
> db.Books.deleteMany({"author": "J K Rowling"})
‘
verify documents have been deleted..!
> db.Books.find()

Update operation: Update author to J K Rowling in the ducument with given unique object id

> db.Books.updateOne({_id: ObjectId("63bb2e9593b418ae2251d145")}, {$set: { "author": "J K Rowling"}})

verify the documents with given unique object id has been updated..!

> db.Books.find({_id: ObjectId("63bb2e9593b418ae2251d145")})