DATABASE MANAGEMENT SYSTEMS LAB

FACULTY MANUAL

II Year I Semester

Prepared by

Mr. J. Karthik
Assistant Professor

Mr. M. Siva Naga Prasad
Assistant Professor

Mr. J.N.V.R.Swarup Kumar
Assistant Professor

Mr. K. Ashok Reddy
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SESHADRI RAO GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521356

SESHADRI RAO
GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institution with Permanent Affiliation to JNTUK, Kakinada)
Seshadri Rao Knowledge Village, Gudlavalleru – 521356

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE VISION & MISSION

Institute Vision

To be a leading institution of engineering education and research, preparing students for Leadership in their fields in a caring and challenging learning environment.

Institute Mission

- To produce quality engineers by providing state-of-the-art engineering education.
- To attract and retain knowledgeable, creative, motivated and highly skilled individuals whose leadership and contributions uphold the college tenets of education, creativity, research and responsible public service.
- To develop faculty and resources to impart and disseminate knowledge and information to students and also to society that will enhance educational level, which in turn, will contribute to social and economic betterment of society.
- To provide an environment that values and encourages knowledge acquisition and academic freedom, making this a preferred institution for knowledge seekers.
- To provide quality assurance.
- To partner and collaborate with industry, government, and R&D institutes to develop new knowledge and sustainable technologies and serve as an engine for facilitating the nation's economic development.
- To impart personality development skills to students that will help them to succeed and lead.
- To instil in students the attitude, values and vision that will prepare them to lead lives of personal integrity and civic responsibility.
- To promote a campus environment that welcomes and makes students of all races, cultures and civilizations feel at home.
- Putting students face to face with industrial, governmental and societal challenges.

DEPARTMENT VISION & MISSION

VISION

To be a Centre of Excellence in Computer Science and Engineering education and training to meet the challenging needs of the industry and society.

MISSION

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.

PEO2: Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations.

PEO3: Demonstrate commitment and progress in lifelong learning, professional development, leader ship and communicate effectively with professional clients and the public.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, Need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10:Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

Students will be able to

PSO1: Design, develop, test and maintain reliable software systems and intelligent systems.

PSO2: Design and develop web sites, web apps and mobile apps.

Course Objectives

- To familiarize with creation of database and formulate SQL solutions to manipulate the database.
- To disseminate knowledge on triggers and PL/SQL programs in a database environment

Course Outcomes

Upon successful completion of the course, the students will be able to

- Create relational database with the given constraints.
- Formulate simple and complex queries using features of SQL.
- Create views on relational database based on the requirements of users.
- Develop PL/SQL programs for processing multiple SQL statements.
- Implement triggers on a relational database

Mapping of Course Outcomes With Program Outcomes

Database Management Systems Lab	1	2	3	4	5	6	7	8	9	10	11	12	PSO1	PSO2
Create relational database with the given constraints.	1	1	2	1	2							1		
Formulate simple and complex queries using features of SQL	2	2	2	2	2							2		1
Create views on relational database based on the requirements of users	1	2	1	2	2									
Develop PL/SQL programs for processing multiple SQL statements	1	2	2	1	2							2		1
Implement triggers on a relational database	2	2	2	1	2							2		2

LIST OF EXPERIMENTS

S.No	Program Name	Mapping of Co's	Page No
1	Execute DDL, DML, DCL and TCL Commands on below given relational schema. EMP(Empno, Ename, Job, Salary, Mgr, Comm, Hiredate, Deptno).	CO1	9
2	Implement the following integrity constraints on the following database EMP (Empno, Ename, Job, Salary, Mgr, Comm, Hiredate, Deptno) DEPT(Deptno, Dname, Location) a. Primary Key b. Foreign Key c. Unique d. Not NULL e. Check	CO1	25
3	Execute basic SQL statements using the following a) Projection b) Selection c) arithmetic operators d) Column aliases e) Concatenation operator f) Character Strings g) Eliminating Duplicate Rows h) Limiting Rows Using • Comparison operators • LIKE,BETWEEN AND,IN operators • Logical Operators i) ORDER BY Clause • Sorting in Ascending Order • Sorting in Descending Order • Sorting by Column Alias • Sorting by Multiple Columns	CO2	32
4	Execute the following single row functions on a Relation • Character Functions o Case-manipulation functions(LOWER, UPPER, INITCAP) o Character-manipulation functions(CONCAT, SUBSTR, LENGTH, INSTR,LPAD | RPAD, TRIM, REPLACE) • Number Functions(ROUND, TRUNC, MOD) • Date functions o Months_Between o Add_Months o Next_Day o Last_Day o Round o Trunc o Arithmetic with Dates	CO2	48
5	Execute the following Multiple row functions (Aggregate Functions) on Relation • Group functions(AVG, COUNT, MAX, MIN, SUM) • DISTINCT Keyword in Count Function • Null Values in Group Functions • NVL Function with Group Functions	CO2	59
6	Create Groups of Data using Group By clause • Grouping by One Column • Grouping by More Than One Column • Illegal Queries Using Group Functions • Restricting groups using HAVING Clause • Nesting Group Functions	CO2	65

7	Retrieve Data from Multiple Tables using the following join operations • Cartesian Products • Equijoin • Non-equijoin • Outer join • Self join	CO2	72
8	Execute Set operations on various Relations. • UNION • UNION ALL • INTERSECT • MINUS	CO2	79
9	Execute Sub Queries and Co-Related Nested Queries on Relations. • Implement o Single-row subquery o Multiple-row subquery • Using Group Functions in a Subquery • Using HAVING Clause with Subqueries • Using Null Values in a Subquery • Data retrieval using Correlated Subqueries o EXISTS Operator o NOT EXISTS Operator	CO2	82
10	Perform following operations on views • Simple Views • Complex Views • Modifying a View DML Operations on a View • Denying DML Operations on view • Removing a View	CO3	88
11	Develop the following PL/SQL programs • Simple PL/SQL programs • PL/SQL programs Using Control structures. o Conditional structures o Iterative structures • PL/SQL program using the following exception handling mechanisms. o Pre defined exceptions o user defined exceptions	CO4	95
12	Implement a PL/SQL block using triggers for transaction operations of a typical application.	CO5	107

ADDITIONAL LAB EXPERIMENTS

S. No	Program Name	Mapping Of COs	Page No
1	Demonstration of database connectivity	CO1	118
2	Write a PL/SQL Code using Procedures, Functions, and Packages FORMS	CO4	122

Exercise : 1

AIM : Execute DDL,DML,DCL and TCL commands on below given relational schema.
EMP(Empno,Ename,Job,Salary,Mgr,Comm,Hiredate,Deptno).

Description **:**

SQL(Structured Query Language) is a standard language for storing, manipulating and retrieving data in databases.

The SQL uses four different languages for the commands

They are:

1. DDL – Data Definition Language.
2. DML –Data Manipulation Language.
3. DCL- Data Control Language.
4. TCL - Transaction Control Language.

Data Definition Language (DDL)

Data definition language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database

List of DDL commands :

1. CREATE
2. DROP
3. ALTER
4. TRUNCATE
5. RENAME

1. CREATE :

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

Syntax :

CREATE table table_name(column 1 domain type 1 , column 2 domain type 2 , …);

Example :

CREATE table EMP(Empno number(15), Ename varchar2(10), Job varchar2(10), Salary real, Mgr int, Comm real, Hiredate date, Deptno int);

desc emp;

```
SQL> CREATE table EMP(Empno number(15),Ename varchar2(10),Job varchar2(10),Salary real,Mgr int,Comm real,Hiredate date,Deptno int);

Table created.

SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(15)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
```

2. DROP :

This command is used to delete objects from database.

Syntax :

DROP table table_name;

Example :

DROP table EMP;

Select * from emp;

```
SQL> drop table emp;

Table dropped.

SQL> select * from emp;
select * from emp
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

3. ALTER :

This is used to alter the structure of the database.

Syntax :

ALTER TABLE - ADD COLUMN :

ALTER table table_name add column_name domain type;

ALTER TABLE – DROP COLUMN :

ALTER table table_name DROP column column_name;

ALTER TABLE – MODIFY COLUMN :

ALTER table table_name MODIFY column_name datatype;

Example :

ALTER table EMP add(fathername varchar2(20)) ;

```
SQL> ALTER table EMP add(fathername varchar2(20));

Table altered.
```

Desc emp;

```
SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(15)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
 FATHERNAME                                         VARCHAR2(20)
```

Alter table emp drop column fathername;

Desc emp;

```
SQL> Alter table emp drop column fathername;

Table altered.

SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(15)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
```

Alter table emp modify name varchar2(20);

Alter table emp modify empno int;

```
SQL> Alter table emp modify ename varchar2(50);

Table altered.

SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(15)
 ENAME                                              VARCHAR2(50)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)

SQL> Alter table emp modify empno int;

Table altered.

SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(38)
 ENAME                                              VARCHAR2(50)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
```

4.TRUNCATE :

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

Syntax :

TRUNCATE table table_name

Example:

TRUNCATE table EMP;

Select * from emp;

```
SQL> truncate table emp;

Table truncated.

SQL> select * from emp;

no rows selected
```

5. RENAME

Rename will be in two situations.

1. To change the name of the table.
2. To change the name of the column.

Syntax

i) alter table tablename rename to players.

Example

alter table player rename to players;

Table altered.

desc players;

Output

```
QL> alter table  player rename to players;

able altered.

QL> desc players;
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
ID                                                 NUMBER(10)
NAME                                               VARCHAR2(20)
EVENT                                              VARCHAR2(10)

QL>
```

ii) alter table tablename column<old-column> to <new-coloumn>

Example

alter table players rename column Event to Events;
table altered.
desc players;

Output

```
SQL> alter table  players rename column event to events;

Table altered.

SQL> desc players;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(10)
 NAME                                               VARCHAR2(20)
 EVENTS                                             VARCHAR2(10)
```

Data Manipulating Language (DML) :

The SQL commands that deals with the manipulation of data present in the data .

DML is the component of SQL statement that controls access to data and to the database.

List of DML commands :

1. SELECT – it is used to retrieve data from the database.
2. INSERT – it is used to insert data into a table.
3. UPDATE – it is used to update existing data within a table.
4. DELETE – it is used to delete records from a database table.

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

Syntax :

SELECT * FROM table_name ;

Example :

Select * from EMP;

```
SQL> select * from emp;

     EMPNO ENAME      JOB            SALARY        MGR       COMM HIREDATE
---------- ---------- ---------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7369 smith      clerk            8000       7902        800 17-DEC-80
        20

      7499 allen      sales           15000       7698        800 20-FEB-81
        30

      7521 ward       sales           15000       7698        600 22-FEB-81
        30


     EMPNO ENAME      JOB            SALARY        MGR       COMM HIREDATE
---------- ---------- ---------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7566 jones      manager         20000       7839       1000 02-APR-81
        30

      7782 clark      manager         20000       7839       1500 09-JAN-81
        40

      7788 scott      analyst         18000       7566       1200 19-APR-82
        40


6 rows selected.
```

2. INSERT : It is an SQL command used to insert new rows in a table.

Syntax :

INSERT INTO table_name values(value1 , value 2 , …);

Example :

insert into emp values(7369,'smith','clerk',8000,7902,800,'17-dec-80',20);

insert into emp values(7499,'allen','sales',15000,7698,800,'20-feb-81',30);

insert into emp values(7521,'ward','sales',15000,7698,600,'22-feb-81',30)

insert into emp values(7566,'jones','manager',20000,7839,1000,'2-apr-81',30);

insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40);

insert into emp values(7788,'scott','analyst',18000,7566,1200,'19-apr-82',40);

```
SQL> insert into emp values(7369,'smith','clerk',8000,7902,800,'17-dec-80',20);

1 row created.

SQL> insert into emp values(7499,'allen','sales',15000,7698,800,'20-feb-81',30);

1 row created.

SQL> insert into emp values(7521,'ward','sales',15000,7698,600,'22-feb-81',30);

1 row created.

SQL> insert into emp values(7566,'jones','manager',20000,7839,1000,'2-apr-81',30);

1 row created.

SQL> insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40);

1 row created.

SQL> insert into emp values(7788,'scott','analyst',18000,7566,1200,'19-apr-82',40);

1 row created.
```

3. UPDATE : It is an SQL command used to update existing rows in a table.

Syntax

UPDATE table_name

SET attribute = value

WHERE condition;

Example :

Update emp set salary = 200000 where empno = 7566;

```
SQL> Update emp set salary = 200000 where empno = 7566;

1 row updated.
```

select * from emp;

```
SQL> select * from emp;

     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7369 smith                                             clerk
      8000       7902        800 17-DEC-80         20

      7499 allen                                             sales
     15000       7698        800 20-FEB-81         30

      7521 ward                                              sales
     15000       7698        600 22-FEB-81         30


     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7566 jones                                             manager
    200000       7839       1000 02-APR-81         30

      7782 clark                                             manager
     20000       7839       1500 09-JAN-81         40

      7788 scott                                             analyst
     18000       7566       1200 19-APR-82         40


6 rows selected.
```

4. DELETE : The delete command is an SQL command used to delete existing records in a table.

Syntax :

DELETE FROM table_name WHERE condition ;

Example :

DELETE FROM EMP WHERE empno = 7566;

Select * from emp;

```
SQL> DELETE FROM EMP WHERE empno = 7566;

1 row deleted.

SQL> select * from emp;

     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7369 smith                                             clerk
      8000       7902        800 17-DEC-80         20

      7499 allen                                             sales
     15000       7698        800 20-FEB-81         30

      7521 ward                                              sales
     15000       7698        600 22-FEB-81         30


     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7782 clark                                             manager
     20000       7839       1500 09-JAN-81         40

      7788 scott                                             analyst
     18000       7566       1200 19-APR-82         40
```

Data Control Language (DCL) :

DCL commands mainly deals with the rights , permissions , and other controls of the database system.

List of DCL commands :

1. GRANT – this command gives users access privileges to the database.
2. REVOKE – this command withdraws the users access privileges given by the GRANT command.

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

Syntax

Grant privilege_name on object_name to {usesr_name};

Example :

Create user ram identified by sri;

User created.

Grant all privileges to ram;

Grant succeded.

```
SQL> create user ram identified by sri;
User created.
SQL> grant all privileges to ram;
Grant succeeded.
```

2. REVOKE : Revoke command withdraw user privileges on database objects if any granted.

Syntax :

Revoke privilege_name on onject_name from {user_name};

Example:

Revoke all privileges from ram;

```
SQL> revoke all privileges from ram;
Revoke succeeded.
```

Transaction Control Language(TCL) :

List of TCL commands :

1. COMMIT – commits a transaction.
2. ROLLBACK – rollbacks a transaction in case of any error occurs.
3. SAVEPOINT – sets a savepoint within a transaction.

1. COMMIT : Commit command is used to save all transactions to the database.

Syntax :

COMMIT;

Example :

insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40);

insert into emp values(7934,'miller','clerk',1300,7782,0,'23-jan-82',10);

Commit;

```
SQL> insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40);

1 row created.

SQL> insert into emp values(7934,'miller','clerk',1300,7782,0,'23-jan-82',10);

1 row created.

SQL> commit;

Commit complete.
```

Select * from emp;

```
SQL> select * from emp;

     EMPNO ENAME                                                JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7369 smith                                              clerk
      8000       7902        800 17-DEC-80         20

      7499 allen                                              sales
     15000       7698        800 20-FEB-81         30

      7521 ward                                               sales
     15000       7698        600 22-FEB-81         30


     EMPNO ENAME                                                JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7782 clark                                              manager
     20000       7839       1500 09-JAN-81         40

      7788 scott                                              analyst
     18000       7566       1200 19-APR-82         40

      7782 clark                                              manager
     20000       7839       1500 09-JAN-81         40


     EMPNO ENAME                                                JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7934 miller                                             clerk
      1300       7782          0 23-JAN-82         10


7 rows selected.
```

2. ROLLBACK :

It is used to undo transactions that have not already been saved to the database.

Syntax :

ROLLBACK;

Example :

insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10);

1 row created.

insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30);

1 row created.

savepoint A;

Savepoint created.

insert into emp values(7876,'adems','accounting',20000,7546,1000,'5-nov-81',10);

1 row created.

savepoint B;

Savepoint created.

insert into emp values(7844,'tunner','salesman',2000,7698,0,'5-jan-82',10);

1 row created.

rollback to savepoint B;

Rollback complete.

```
SQL> insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10);

1 row created.

SQL> insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30);

1 row created.

SQL> savepoint A;

Savepoint created.

SQL> insert into emp values(7876,'adems','accounting',20000,7546,1000,'5-nov-81',10);

1 row created.

SQL> savepoint B;

Savepoint created.

SQL> insert into emp values(7844,'tunner','salesman',2000,7698,0,'5-jan-82',10);

1 row created.

SQL> rollback to savepoint B;

Rollback complete.
```

3. SAVEPOINT : It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax :

SAVEPPOINT savepoint_name;

Example :

insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10);

1 row created.

insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30);

1 row created.

savepoint A;

```
SQL> insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10);

1 row created.

SQL> insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30);

1 row created.

SQL> savepoint A;

Savepoint created.
```

VIVA-VOCE QUESTIONS

1. List out DDL, DML, TCL and DCL commands.
2. Difference between Truncate and Drop.
3. Difference between Commit and Savepoint.
4. Creation of a table.

EXERCISE : 2

AIM : Implement the following integrity constraints on the following database EMP (Empno, Ename, Job, Salary, Mgr, Comm, Hiredate, Deptno) DEPT(Deptno, Dname, Location)
a. Primary Key b. Foreign Key c. Unique d. Not NULL e. Check

Description :

Constraints

- **KEY CONSTRAINTS**
  - **Super key :** set of one or more attributes that uniquely identifies a tuple in a relation is called as a super key.
  - **Candidate key :** minimal set of attributes that uniquely identifies a tuple in a relation is called as a candidate key.
  - **Primary key :** it is a key which uniquely identifies a tuple in a relation . the two properties of primary key are unique and not null.
  - **Alternate key:** an alternate key is a key that can be work as a primary key .basically it is a candidate key that is not a primary key.
  - **Foreign key:** ensure that referential integrity of the data in one table to match values in another table . ensure that the foregin key in the child table match with the primary key in the parent table.

- **INTEGRITY CONSTRAINTS**
  - **Unique key** : unique key is a set of one or more fields/columns of a table that uniquely identify a record in database table .it is like primary key but it can accept only one null value and it cannot have duplicate values.
  - **Check :** ensures that the value in a field meets a specified condition.
  - **Not NULL :** indicates that a field cannot store a NULL value.

- **Constraints according to the aim :**

a) Primary Key constraint :

The primary key constraint uniquely identifies each record in atable. They must contain UNIQUE values and cannot contain NULL values. A table can have only ONE primary key and in the table, this primary key can consist of single or multiple columns/fields .

Syntax :

Create table table_name(attribute name domaintype primary key , ..);

Or

By using alter :

Alter table table_name add constraint constraint_name primary key(attribute);

Example :

create table dept(deptno int primary key,dname varchar2(20),location varchar2(20));

```
SQL> create table dept(deptno int primary key,dname varchar2(20),location varchar2(20));

Table created.
```

Desc dept;

```
SQL> desc dept;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                    NOT NULL NUMBER(38)
 DNAME                                              VARCHAR2(20)
 LOCATION                                           VARCHAR2(20)
```

Select * from dept;

```
SQL> select * from dept;

    DEPTNO DNAME                LOCATION
---------- -------------------- --------------------
        10 acounting            newyork
        20 research             dallas
        30 sales                chicago
        40 operations           boston
```

b) Foreign Key Constraint:

The foreign key constraint is used to prevent actions that would destroy links between tables. A foreign key is a field or a collection of fields in one table , that refers to the primary key in another table. The table with the foreign key is called the child table , and the table with the primary key is called the referenced or parent table.

Syntax :

Create table table_name(column domain type ,… , column n domain type n , foregin key(column) , references column in parent table);

Example :

create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept);

```
SQL> create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept);

Table created.
```

Desc emp;

```
SQL> desc emp1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(38)
 ENAME                                              VARCHAR2(20)
 SALARY                                             NUMBER(10)
 MGR                                                FLOAT(63)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
```

Select * from emp;

```
SQL> select * from emp1;

     EMPNO ENAME                SALARY        MGR       COMM HIREDATE
---------- -------------------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7369 smith                      8000       7902        800 17-DEC-80
        20

      7499 allen                     15000       7698        800 20-FEB-81
        30

      7521 ward                      15000       7698        600 22-FEB-81
        30


     EMPNO ENAME                SALARY        MGR       COMM HIREDATE
---------- -------------------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7566 jones                     20000       7839       1000 02-APR-81
        30

      7782 clark                     20000       7839       1500 09-JAN-81
        10

      7788 scott                     18000       7566       1200 19-APR-82
        40


6 rows selected.
```

It has two attributes :

1. ON DELETE CASCADE :

when a primary key is deleted in the parent table then corresponding data in the child table also gets deleted.

Syntax :

Create table table_name(attribute domain type , foregin key(attribute) references parent table ON DELETE CASCADE):

Example :

create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE CASCADE);

```
SQL> create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE CASCADE);

Table created.
```

2. ON DELETE SET NULL :

When a primary key and its corresponding tuples gets deleted in parent table and then corresponding records in the child table will have the foreign key field set to null but not get deleted.

Syntax :

Create table table_name(attribute domain type , foregin key(attribute) references parent table ON DELETE SET NULL):

Example :

create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE SET NULL);

```
SQL> create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE SET NULL);

Table created.
```

3. Unique Key Constraint :

The unique constraint imposes that every value in a column or set of columns by unique. It means that no two rows of a table can have duplicate values in a specified column or set of columns.

Syntax:

While creating table :

Syntax :

Create table table_name(coloumn domain type unique);

Example :

create table dept(deptno int primary key ,dname varchar2(20),location varchar2(20) unique);

```
SQL> create table dept(deptno int primary key ,dname varchar2(20),location varchar2(20) unique);

Table created.
```

Unique constraint violation :

```
SQL> insert into dept values(10,'accounting','newyork');

1 row created.

SQL> insert into dept values(10,'accounting','newyork');
insert into dept values(10,'accounting','newyork')
*
ERROR at line 1:
ORA-00001: unique constraint (SRI.SYS_C004045) violated
```

4. NOT NULL: Indicates that a column cannot store NULL value.

Syntax :

Alter table table_name modify attribute domain type NOT NULL;

Example :

Alter table emp modify attribute varchar2(20) NOT NULL;

```
SQL> Alter table emp1 modify ename varchar2(20) NOT NULL;

Table altered.

SQL> desc emp1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(38)
 ENAME                                     NOT NULL VARCHAR2(20)
 SALARY                                             NUMBER(10)
 MGR                                                FLOAT(63)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)

SQL> insert into emp1 values(7369,' ',10000,7902,800,'17-dec-82',20);
insert into emp1 values(7369,' ',10000,7902,800,'17-dec-82',20)
*
ERROR at line 1:
ORA-02291: integrity constraint (SRI.SYS_C004047) violated - parent key not
found
```

```
SQL> insert into emp1 values(7369,' ',10000,7902,800,'17-dec-82',20);
insert into emp1 values(7369,' ',10000,7902,800,'17-dec-82',20)
*
ERROR at line 1:
ORA-00001: unique constraint (SRI.SYS_C004053) violated
```

5. Check : Ensures that the value in a column meets a specific condition.

Syntax :

Alter table table_name add constraint constraint_name check(condition);

Example :

Alter table emp add constraint s3 check(salary > 500 and salary < = 20000);

```
SQL> Alter table emp add constraint s3 check(salary > 500 and salary < = 20000);

Table altered.
```

Select * from emp;

```
SQL> select * from emp1;

     EMPNO ENAME                SALARY        MGR       COMM HIREDATE
---------- -------------------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7521 ward                  15000       7698        600 22-FEB-81
        30

      7566 jones                 15000       7839       1000 02-APR-81
        10

      7788 clark                 20000       7839       1500 09-JAN-81
        10
```

VIVA QUESTIONS

1) Define primary key.
2) Define foreign key**.**
3) What is the purpose of check and not null constraints.
4) How the primary key does differs from a candidate key? How they are similar?

EXERCISE : 3

AIM : Execute basic SQL statements using the following

a) Projection b) Selection

c) arithmetic operators d) Column aliases

e) Concatenation operator f) Character Strings

g) Eliminating Duplicate Rows

h) Limiting Rows Using

• Comparison operators • LIKE,BETWEEN AND,IN operators

• Logical Operators

i) ORDER BY Clause

• Sorting in Ascending Order • Sorting in Descending Order

• Sorting by Column Alias • Sorting by Multiple Columns

Description :

a) Projection :

It produces a new relation with only some of the attributes of R , and removes duplicate tuples.

Example :

Find the names and ages of all the sailors.

```
SQL> select s.sname,s.age from sailors s;

SNAME                          AGE
-------------------- ----------
dustin                          45
brutus                          33
lubber                        55.5
andy                          25.5
rusty                           35
horatio                         35
zorba                           16
horatio                         35
art                           25.5
bob                           63.5

10 rows selected.
```

b) Selection :

It selects all tuples that satisfy the selection conditions from a relation R.

Example :

Find all sailors with a rating above 7.

```
SQL> select * from sailors s where rating > 7;

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        31 lubber                        8       55.5
        32 andy                          8       25.5
        58 rusty                        10         35
        71 zorba                        10         16
        74 horatio                       9         35

SQL> select sname,rating,rating+5 from sailors;
```

c) Arithmetic operators :

These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in SQL .

Example:

Arithametic operators	Example / Description
+ (addition)	A+B
- (subtraction)	A-B
* (multiplication)	A*B
/ (Division)	A/B
% (Modulus)	A%B

1. Add a rating 5 to all sailors and display sname, rating, rating+5 from a sailors relation.

```
SQL> select sname,rating,rating+5 from sailors;

SNAME                     RATING   RATING+5
-------------------- ---------- ----------
dustin                         7         12
brutus                         1          6
lubber                         8         13
andy                           8         13
rusty                         10         15
horatio                        7         12
zorba                         10         15
horatio                        9         14
art                            3          8
bob                            3          8

10 rows selected.
```

2. Reduce the age of all sailors by 3 and display sid, name old age and new age of all the sailors .

```
SQL> select sid,name,age,age-3 from sailors;
select sid,name,age,age-3 from sailors
           *
ERROR at line 1:
ORA-00904: "NAME": invalid identifier


SQL> select sid,sname,age,age-3 from sailors;

       SID SNAME                      AGE      AGE-3
---------- -------------------- ---------- ----------
        22 dustin                       45         42
        29 brutus                       33         30
        31 lubber                     55.5       52.5
        32 andy                       25.5       22.5
        58 rusty                        35         32
        64 horatio                      35         32
        71 zorba                        16         13
        74 horatio                      35         32
        85 art                        25.5       22.5
        95 bob                        63.5       60.5

10 rows selected.
```

3. Multiply the rating of all the sailors by 5 and display sid, old rating and new rating from sailors.

```
SQL> select sid,rating,rating*5 from sailors;

       SID     RATING   RATING*5
---------- ---------- ----------
        22          7         35
        29          1          5
        31          8         40
        32          8         40
        58         10         50
        64          7         35
        71         10         50
        74          9         45
        85          3         15
        95          3         15

10 rows selected
```

4. Divide bid with sid and display resultant sid, bid/sid for reserves.

```
SQL> select sid,bid/sid from reserves;

       SID    BID/SID
---------- ----------
        22 4.59090909
        22 4.63636364
        22 4.68181818
        22 4.72727273
        31 3.29032258
        31 3.32258065
        64   1.578125
        64    1.59375
        31 3.35483871
        74 1.39189189

10 rows selected.
```

5. Retrieve age, age%2 using Mod operator on sailors.

```
SQL> select age,mod(age,2) from sailors;

       AGE MOD(AGE,2)
---------- ----------
        45          1
        33          1
      55.5        1.5
      25.5        1.5
        35          1
        35          1
        16          0
        35          1
      25.5        1.5
      63.5        1.5

10 rows selected.
```

d) Colmn aliases :

An alias is created with the AS keyword . it gives a table , or a column in a table a temporary name.

Example :

1. Apply the column aliasing to sailors table by aliasing rating as star rating.

```
SQL> select rating As starrating from sailors;

STARRATING
----------
         7
         1
         8
         8
        10
         7
        10
         9
         3
         3

10 rows selected.
```

e) Concatenation operator :

It is used to link columns or character strings .

Example :

1. Apply concatenation operator on dual or any relation

```
SQL> select concat(' ram ',' is a good sailor ') from dual;

CONCAT('RAM','ISAGOODSA
----------------------
 ram  is a good sailor

SQL> select sid || ' is a good sailor ' from sailors where sid = 22;

SID||'ISAGOODSAILOR'
------------------------------------------------------
22 is a good sailor
```

f) Character Strings :

A character string is any sequence of zero or more alphanumeric characters that belong to a given character set .

Example :

1. Apply Char functions on dual

```
SQL> select chr(65) from dual;

C
-
A

SQL> select chr(65) || chr(87) || chr(92) from dual;

CHR
---
AW\
```

g) Eliminating Duplicate Rows :

Deleting the duplicate rows or tuples or records from the table .

Syntax :

Select * from table_name where rowid in(select max(rowid) from table_name group by attribute);

Example :

Select * from duplicate where rowid in(select max(rowid) from duplicate group by id);

```
SQL> Select * from duplicate where rowid in(select max(rowid) from duplicate group by id);

        ID FIRSTNAME  LASTNAME
---------- ---------- ----------
       540 priya      ram
       590 vemuri     gopi
```

h) Limiting rows using

Comparison operator :

Comparison operators are used to compare according to the conditions.

Operator	Symbol
Less than	<
Gretar than	>
Equal to	=
Less than or equal to	<=
Greather than or equal to	>=
Not equal	<>

Example :

1.Find sid,ages of sailors whose age is above 45

2.Find sid,sname of sailors whose rating is below 5

3.Find sid,sname of sailors whose rating is greater than or equal to 7

4.Find sid,sname of sailors whose age is less than or equal to 32

5.Find sid,sname of sailors whose rating is equal to 7

6.Find sid,sname of sailors whose rating is not equal to 7.

```
SQL> select sid,age from sailors where age > 45;

       SID        AGE
---------- ----------
        31       55.5
        95       63.5

SQL> select sid,sname from sailors where rating < 5;

       SID SNAME
---------- --------------------
        29 brutus
        85 art
        95 bob

SQL> select sid,sname from sailors where rating > = 7;

       SID SNAME
---------- --------------------
        22 dustin
        31 lubber
        32 andy
        58 rusty
        64 horatio
        71 zorba
        74 horatio

7 rows selected.

SQL> select sid,sname from sailors where age<=32;

       SID SNAME
---------- --------------------
        32 andy
        71 zorba
        85 art

SQL> select sid,sname from sailors where rating = 7;

       SID SNAME
---------- --------------------
        22 dustin
        64 horatio
```

i) LIKE , BETWEEN , AND , IN operators :

LIKE :

Like is used to take the check whether the letter or a string starts with a particular name or character.

It is usually calculated using percentile (%) for n number of characters

underscore (_) for the single characters .

Example :

1. Find the ages of sailors whose name begins **b** and ends with **b** and has at least three characters.
2. Find the sid's, names of sailors whose name begins with **a** and has at least three characters.

```
SQL> select s.age,s.sname from sailors s where s.sname LIKE 'b%b';

       AGE SNAME
---------- ------------------
      63.5 bob

SQL> select s.sid,s.sname from sailors s where s.sname LIKE 'a%';

       SID SNAME
---------- ------------------
        32 andy
        85 art
```

Not Like : It just works opposite to the LIKE operator.

Example :

1. Find the names of sailors whose name doesn't begins with **a** and has at least three characters.

```
SQL> select s.sname from sailors s where s.sname NOT LIKE 'a%';

SNAME
------------------
dustin
brutus
lubber
rusty
horatio
zorba
horatio
bob

8 rows selected.
```

BETWEEN :

It is used to check the condition in particular range or not.

Example :

1. Find the details of sailors age between 40 and 60.

```
SQL> select age from sailors where age BETWEEN 40 AND 60;

       AGE
----------
        45
      55.5
```

AND :

And is used to compare between to conditions or it is used to combine.

Example :

1. Find the names of sailors who have reserved a red color boat.

```
SQL> select s.sname from sailors s ,reserves r,boats b where s.sid = r.sid and r.bid = b.bid and b.color = 'red';

SNAME
--------------------
dustin
dustin
lubber
horatio
lubber

SQL> select s.sname,b.color from sailors s ,reserves r,boats b where s.sid = r.sid and r.bid = b.bid and b.color = 'red';

SNAME                COLOR
-------------------- --------------------
dustin               red
dustin               red
lubber               red
horatio              red
lubber               red
```

IN :

Example :

1. Find the snames of sailors who have reserved boat 103 **(Using IN)**

```
SQL> select s.sname from sailors s where s.sid IN(select r.sid from reserves r where r.bid = 103);

SNAME
--------------------
dustin
lubber
horatio
```

NOT IN :

Example :

1. Find the snames of sailors who have not reserved boat 103 **(using NOT IN)**

```
SQL> select s.sname from sailors s where s.sid NOT IN(select r.sid from reserves r where r.bid = 103);

SNAME
--------------------
brutus
andy
rusty
horatio
zorba
art
bob

7 rows selected.
```

Logical Operators :

1. **ALL :**

It returns true if all of the subquery values meet the condition.

Syntax :

SELECT column/attribute

FROM table_name

WHERE condition_attribute = ALL (SELECT condition_attribute FROM table_name2 WHERE condition);

Example :

Select sname from sailors where sid = all(select sid from reserves where bid = 103);

```
SQL> Select sname from sailors where sid = all(select sid from reserves where bid = 103);

no rows selected
```

2. **AND :**

It rerurns true if all the conditions separated by and is true.

Syntax :

SELECT * FROM table_name

WHERE attribute = value AND attribute = value;

Example :

Select * from sailors where sname = ‘lubber’ and sid = 31;

```
SQL> Select * from sailors where sname = 'lubber' and sid = 31;

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        31 lubber                        8       55.5
```

3. **ANY :**

It returns true if any of the subquery values meet the condition.

Syntax :

SELECT * FROM table_name

WHERE condition ANY (SELECT attribute FROM table_name2 WHERE condition);

Example :

Find the sailors whose rating is better than soe sailor called Horatio

```
SQL> select s.sid from sailors s where s.rating>ANY(select s2.rating
from sailors s2 where s2.sname='Horatio');

       SID
----------
        58
        71
        74
        31
        32
```

4. **EXIST :**

It returns true if the subquery returns one or more records.

Syntax :

SELECT column

FROM table_name

WHERE EXIST (SELECT attribute FROM table_name2 WHERE condition AND condition);

Example :

Find the names of sailors who have reserved boat number 103

```
SQL> select s.sname from sailors s where EXISTS(select * from reserve
s r where r.bid=103 and r.sid=s.sid);

SNAME
----------
Dustin
Lubber
Horatio
```

5. **NOT :**

Displays a record if the condition is not true.

Syntax :

SELECT * FROM table_name

WHERE attribute NOT LIKE 'string';

Example :

Select * from sailors where sname not like ‘a%’;

```
SQL> Select * from sailors where sname not like 'a%';

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        22 dustin                        7         45
        29 brutus                        1         33
        31 lubber                        8       55.5
        58 rusty                        10         35
        64 horatio                       7         35
        71 zorba                        10         16
        74 horatio                       9         35
        95 bob                           3       63.5

8 rows selected.
```

6. **OR :**

It returns true if any of the conditions separated by or is true.

Syntax :

SELECT * FROM table_name

WHERE attribute = value OR attribute = value**;**

Example :

Select * from sailors where sname = ‘lubber’ or sid = 103;

```
SQL> Select * from sailors where sname = 'lubber' or sid = 103;

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        31 lubber                        8       55.5
```

I) **ORDER by clause :**

ORDER BY

The ORDER BY statement in sql is used to sort the fetched data in either ascending or descending according to one or more columns.

- By default ORDER BY sorts the data in ascending order.
- We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

Syntax of all ways of using ORDER BY is shown below:

- Sort according to one column: To sort in ascending or descending order we can use the keywords ASC or DESC respectively.

  Syntax:

  SELECT * FROM table_name ORDER BY column_name ASC|DESC;
- Sort according to multiple columns: To sort in ascending or descending order we can use the keywords ASC or DESC respectively. To sort according to multiple columns, separate the names of columns by (,) operator.

  Syntax:
- SELECT * FROM table_name ORDER BY column1 ASC|DESC , column2 ASC|DESC;

Sorting in ascending order :

Example :

1. Find the details of sailors, ordered by sid

```
SQL> select * from sailors order by sid asc;

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        22 dustin                        7         45
        29 brutus                        1         33
        31 lubber                        8       55.5
        32 andy                          8       25.5
        58 rusty                        10         35
        64 horatio                       7         35
        71 zorba                        10         16
        74 horatio                       9         35
        85 art                           3       25.5
        95 bob                           3       63.5

10 rows selected.
```

Sorting in descending order :

Example :

1. Find rating and sid's of sailors ordered by rating in descending order

```
SQL> select * from sailors order by rating desc;

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        58 rusty                        10         35
        71 zorba                        10         16
        74 horatio                       9         35
        31 lubber                        8       55.5
        32 andy                          8       25.5
        64 horatio                       7         35
        22 dustin                        7         45
        95 bob                           3       63.5
        85 art                           3       25.5
        29 brutus                        1         33

10 rows selected.
```

Sorting by column alias :

Example :

```
SQL> select rating As starrating from sailors order by starrating;

STARRATING
----------
         1
         3
         3
         7
         7
         8
         8
         9
        10
        10

10 rows selected.
```

Sorting by multiple columns :

Example :

1. Find the details of all sailors ordered by sname and sid .

```
SQL> select * from sailors order by sname,sid;

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        32 andy                          8       25.5
        85 art                           3       25.5
        95 bob                           3       63.5
        29 brutus                        1         33
        22 dustin                        7         45
        64 horatio                       7         35
        74 horatio                       9         35
        31 lubber                        8       55.5
        58 rusty                        10         35
        71 zorba                        10         16

10 rows selected.
```

VIVA QUESTIONS

1. What is Projection?
2. Define Selection?
3. List the logical Operators
4. Outline the various Comparison Operators

EXERCISE : 4

AIM : Execute the following single row functions on a Relation

- Character Functions
  - Case-manipulation functions(LOWER, UPPER, INITCAP)
  - Character-manipulation functions(CONCAT, SUBSTR, LENGTH, INSTR,LPAD | RPAD, TRIM, REPLACE)
- Number Functions(ROUND, TRUNC, MOD)
- Date functions
  - Months_Between
  - Add_Months
  - Next_Day
  - Last_Day
  - Round
  - Trunc
  - Arithmetic with Dates

Description :

a) Character functions :

Case – manipulation functions (LOWER , UPPER , INITCAP) :

1. lower (): this function converts the uppercase letters to lower case letters what you are passed to the function.

Syntax:

lower(message)

Example

select lower('SRGEC') as low from dual;

```
SQL> select  lower('SRGEC') as low from dual;

LOW
-----
srgec
```

2.upper(): this function is used to convert the lower case letters into uppercase letters.

Syntax:

upper(message)

Example

select upper('database') as upper1 from dual;

```
SQL> select upper('database') as upper1 from dual;

UPPER1
--------
DATABASE
```

3. initcap():

It make initial letter to capital letter what you have passed to the function.

Syntax:

initcap(message)

Example :

select initcap('srgec') from dual;

```
SQL> select  initcap('srgec') from dual;

INITC
-----
Srgec
```

Character manipulation functions (CONCAT , SUBSTR , LENGTH , INSTR , LPAD | RPAD , TRIM , REPLACE) :

1. **lpad():** This function is used for attaching a new word to the original one at left side.

   Syntax:

   lpad(word1,length,word2)

   Example:

   select lpad('gec',’6','cse') as lpad1 from dual;

```
SQL> select lpad('gec','6','cse') as lpad1 from dual;

LPAD1
------
csegec
```

2. **rpad():** This function is used for attaching a new word to the original one at right side.

   Syntax:

   rpad(word1,length,word2)

   Example: select Rpad('CSE',10,'GEC') from dual;

```
SQL> select Rpad('CSE',10,'GEC') from dual;

RPAD('CSE'
----------
CSEGECGECG
```

3. **ltrm():**This function is used for left trimming i.e, it delete(cut) the left most letter.

Syntax:

ltrim('message','character')

Example:

select ltrim('computerscience','c') as msg from dual;

```
SQL> select ltrim('computerscience','c') as msg from dual;

MSG
-------------
omputerscience
```

4. **rtrim()**

This function is used for right trimming.

Syntax:

rtrim('message','character')

Example:

select rtrim('computerscience','e') as rtrim1 from dual;

```
SQL> select rtrim('computerscience','e') as rtrim1 from dual;

RTRIM1
-------------
computerscienc
```

5. **concat():**This function is used to add two strings.

Syntax:

Concat('string1' , 'string 2')

Example :

select concat('ABC','DEF') from dual;

```
SQL> select concat('ABC','DEF') from dual;

CONCAT
------
ABCDEF
```

6. Replace : This function is used to replace a particular character from a string.

Syntax :

Replace('string', 'replaceable char',char);

Example :

select replace ('jack and jue','j','bl') from dual;

```
SQL> select replace ('jack and jue','j','bl') from dual;

REPLACE('JACKA
--------------
black and blue
```

7. substring : This function is used to extract the substring from a main string .

syntax :

substr(string , indexing , size);

Example :

SQL> select substr('srgec is a college',12,7) from dual;

```
SQL> select substr('srgec is a college',12,7) from dual;

SUBSTR(
-------
college
```

8. length : This function is used to find the length of a given string.

Syntax :

Length(string);

Example:

SQL> select length('srgec') from dual;

```
SQL> select length('srgec') from dual;

LENGTH('SRGEC')
---------------
              5
```

9. INSTR():The INSTR() function returns the position of the first occurrence of a string in another string. This function performs a case-insensitive search.

Syntax: INSTR(string1, string2)

Parameter Values

Parameter	Description
string1	Required. The string that will be searched
string2	Required. The string to search for in *string1*. If *string2* is not found, this function returns 0

Example: SQL> SELECT INSTR('srgec','e') from dual;

Output:

```
SQL> SELECT INSTR('srgec','e') from dual;

INSTR('SRGEC','E')
------------------
                 4
```

b) Number functions (ROUND , TRUNCATE , MOD) :

1. ROUND :

The ROUND() function rounds a number to a specified number of decimal places.

SYNTAX :

ROUND(*number*, *decimals*, *operation*)

Example :

```
SQL> select round(17.77,1) from dual;

ROUND(17.77,1)
--------------
          17.8

SQL> select round(17.74,1) from dual;

ROUND(17.74,1)
--------------
          17.7

SQL> select round(171.77,-1) from dual;

ROUND(171.77,-1)
----------------
             170

SQL> select round(177.77,-1) from dual;

ROUND(177.77,-1)
----------------
             180

SQL> select round(1789.77,1) from dual;

ROUND(1789.77,1)
----------------
          1789.8
```

2. TRUNCATE :

The TRUNCATE() function truncates a number to the specified number of decimal places.

Syntax :

TRUNCATE(*number*, *decimals*)

Example :

Output :

```
SQL> select trunc(17.32) from dual;

TRUNC(17.32)
------------
          17

SQL> select trunc(17.3217,2) from dual;

TRUNC(17.3217,2)
----------------
           17.32

SQL> select trunc(17.3217,7) from dual;

TRUNC(17.3217,7)
----------------
         17.3217

SQL> select trunc(17.32,-1) from dual;

TRUNC(17.32,-1)
---------------
             10

SQL> select trunc(17.32,-2) from dual;

TRUNC(17.32,-2)
---------------
              0

SQL> select trunc(173.32,-2) from dual;

TRUNC(173.32,-2)
----------------
             100

SQL> select trunc(1546.3236,-2) from dual;

TRUNC(1546.3236,-2)
-------------------
               1500
```

3. MOD :

The MOD() function returns the remainder of a number divided by another number.

Syntax :

MOD(x , y)

Example :

select mod(17,3) from dual;

```
SQL> select mod(17,3) from dual;

 MOD(17,3)
----------
         2
```

c) Date functions :

Months _ Between : It gives the number of months between specified two dates.

Result value	Months_between(date-exp1,date-exp2)
Negative result	If date-exp1 is earlier than date-exp2
Integer result	If date-exp1 and date-exp2 have the same day,or both specify the last day of the month.
Decimal result	If days are different and they are not both specify the last day of the month
Fractional part	Always calcilated as the difference between days divided by 31 despite the number of days in the month.

Syntax:

months_between(date1,date2)

Example:

select months_between('28-aug-17','1-jan-17') as mon from dual;

```
SQL> select months_between('28-aug-17','1-jan-17') as mon from dual;

       MON
----------
7.87096774
```

Add _Months : This function is used to add the 'n' number of months to a given date.

Example:

select add_months('28-sep-1997',5) from dual;

```
SQL> select  add_months('28-sep-1997',5) from dual;

ADD_MONTH
---------
28-FEB-98
```

Next_Day :

Syntax :

next_day(date,dayname)

EXAMPLE :

SQL> select sysdate,next_day(sysdate,'monday') from dual;

```
SQL> select sysdate,next_day(sysdate,'monday') from dual;

SYSDATE   NEXT_DAY(
--------- ---------
21-NOV-21 22-NOV-21
```

Last _ Day : It gives the last day of the specified month in a date.

Syntax:

last_date(date)

Example:

select last_day('28-sep-2017') as lastday from dual;

```
SQL> select last_day('28-sep-2017') as lastday from dual;

LASTDAY
---------
30-SEP-17
```

Round :

The Round() Returns the date rounded by the specified format unit

Example :

select round(to_date('10-oct-1998'),'MM') "nearest month" from dual;

```
SQL> select round(to_date('10-oct-1998'),'MM') "nearest month" from dual;

nearest m
---------
01-OCT-98
```

Trunc : Truncates the specified date of its time portion according to the format unit provided**.**

Example :

```
SQL> select trunc(to_date('29-oct-1998'),'MM') "nearest month" from dual;

nearest m
---------
01-OCT-98
```

Arithmetic with Dates:

- Add or subtract a number to or from a date for a resultant date value
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24
- Since the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates. You can perform the following operations:

Operation	Result	Description
Date + number	Date	Adds a number of days to a date
Date - number	Date	Subtracts a number of days from a date
Date – date	Number of days	Subracts one date from another
Date + number/24	Date	Adds a number of hours to a date

```
SQL> select SYSDATE from dual;

SYSDATE
---------
21-NOV-21

SQL> select SYSDATE+1 from dual;

SYSDATE+1
---------
22-NOV-21

SQL> select SYSDATE+9 from dual;

SYSDATE+9
---------
30-NOV-21

SQL> select SYSDATE-2 from dual;

SYSDATE-2
---------
19-NOV-21
```

VIVA QUESTIONS

1. List various Case-manipulation functions

2. Outline the date functions

3. Illustrate Aggregate Functions

EXERCISE : 5

AIM : Execute the following Multiple row functions (Aggregate Functions) on Relation

• Group functions(AVG, COUNT, MAX, MIN, SUM)

• DISTINCT Keyword in Count Function

• Null Values in Group Functions

• NVL Function with Group Functions.

Description :

AGGREGATE FUNCTIONS

In data base management system ,an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

The aggregate functions are:

1) MAX(): It returns the max value in the given column.

2) MIN(): It returns the max value in the given column.

3) SUM(): It returns the sum of all numeric values in the given column.

4) AVG(): It returns the average of all values in the given column.

5) COUNT():It returns the total number of all values in the given column(excluding null values).

6) COUNT(*):It returns the number of all rows in the given table(including null values).

Group Functions(AVG , COUNT , MAX , MIN , SUM) :

Example :

```
SELECT * FROM sailors;
```

```
SQL> select * from sailors;

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        22 dustin                        7         45
        29 brutus                        1         33
        31 lubber                        8       55.5
        32 andy                          8       25.5
        58 rusty                        10         35
        64 horatio                       7         35
        71 zorba                        10         16
        74 horatio                       9         35
        85 art                           3       25.5
        95 bob                           3       63.5

10 rows selected.
```

AVERAGE (AVG):

Example :

select avg(s.age) from sailors s;

```
SQL> select avg(s.age) from sailors s;

AVG(S.AGE)
----------
      36.9
```

select avg(s.age) from sailors s where s.rating=10;

```
SQL> select avg(s.age) from sailors s where s.rating=10;

AVG(S.AGE)
---------
     25.5
```

MAXIMUN (MAX):

Example:

select max(s.age) from sailors s;

```
SQL> select max(s.age) from sailors s;

MAX(S.AGE)
----------
      63.5
```

MINIMUM (MIN):

Example:

select min(s.age) from sailors s;

```
SQL> select min(s.age) from sailors s;

MIN(S.AGE)
----------
        16
```

SUM:

Example:

select sum(distinct s.rating) from sailors s;

```
SQL> select sum(distinct s.rating) from sailors s;

SUM(DISTINCTS.RATING)
---------------------
                   38
```

COUNT:

Example:

select count(*) from sailors;

```
SQL> select count(*) from sailors;

  COUNT(*)
----------
        10
```

DISTINCT keyword in count function :

EXAMPLE :

select count(Distinct sname) from sailors;

```
SQL> select count(Distinct sname) from sailors;

COUNT(DISTINCTSNAME)
--------------------
                   9
```

Null values in group functions :

EXAMPLE :

Create table table1 (id int, col int);

Select * from table1;

```
SQL> select * from table1;

        ID         COL
---------- ----------
         1
         2
         3
         4
```

select count(*) from table1;

```
SQL> select count(*) from table1;

  COUNT(*)
----------
         5
```

select count(id) from table1;

```
SQL> select count(id) from table1;

 COUNT(ID)
----------
         4
```

NVL function with group functions :

EXAMPLE :

create table gg(fname varchar2(20),lname varchar2(20),country varchar2(10));

SELECT * FROM GG;

```
SQL> select * from gg;

FNAME                LNAME                COUNTRY
-------------------- -------------------- ----------
karthik              j                    INDIA
krishna              m                    INDIA
sri
                     rithu                uk
```

select fname,NVL(fname,'noname') from gg;

```
SQL> select fname,NVL(fname,'noname') from gg;

FNAME                NVL(FNAME,'NONAME')
-------------------- --------------------
karthik              karthik
krishna              krishna
sri                  sri
                     noname
```

select lname,NVL(lname,'empty') from gg;

```
SQL> select lname,NVL(lname,'empty') from gg;

LNAME                NVL(LNAME,'EMPTY')
-------------------- --------------------
j                    j
m                    m
                     empty
rithu                rithu
```

select country,NVL(country,'no country') from gg;

```
SQL> select country,NVL(country,'no country') from gg;

COUNTRY    NVL(COUNTR
---------- ----------
INDIA      INDIA
INDIA      INDIA
           no country
uk         uk
```

select country,NVL(country,'0') from gg;

```
SQL> select country,NVL(country,'0') from gg;

COUNTRY    NVL(COUNTR
---------- ----------
INDIA      INDIA
INDIA      INDIA
           0
uk         uk
```

select * from gg;

```
SQL> select * from gg;

FNAME                LNAME                COUNTRY
-------------------- -------------------- ---------
karthik              j                    INDIA
krishna              m                    INDIA
sri
                     rithu                uk
```

select lname,NVL(lname,fname) from gg;

```
SQL> select lname,NVL(lname,fname) from gg;

LNAME                NVL(LNAME,FNAME)
-------------------- --------------------
j                    j
m                    m
                     sri
rithu                rithu
```

EXERCISE : 6

AIM : Create Groups of Data using Group By clause

• Grouping by One Column

• Grouping by More Than One Column

• Illegal Queries Using Group Functions

• Restricting groups using HAVING Clause

• Nesting Group Functions.

Description :

GROUP by clause :

- The GROUP BY clause is used in a SELECT statement to collect data across multiple records and group the results by one or more columns.
- Sometimes it is required to get information not about each row, but about each group.
- Related rows can be grouped together by the GROUP BY clause by specifying a column as a grouping column.
- In the output table all the rows with an identical value in the grouping column will be grouped together. Hence, the number of rows in the output is equal to the number of distinct values of the grouping column.

Grouping by More Than One Column

EXAMPLE :

create table employee(eno int,ename varchar2(10),job varchar2(10),salary int,deptno int);

select * from employee;

```
SQL> select * from employee;

       ENO ENAME      JOB            SALARY     DEPTNO
---------- ---------- ---------- ---------- ----------
         1 ram        Asst.prof       40000         10
         3 rahat      prof            80000         10
         4 suresh     Asst.prof       40000         20
         5 ruhani     Asst.prof       45000         20
         7 Ankit      Asst.prof       50000         30
         9 Amit       prof            90000         30
         9 Amit       prof            90000         30

7 rows selected.
```

Query: Total salary paid to each job in each department

select Deptno,job,salary from employee;

```
SQL> select deptno,job,salary from employee;

    DEPTNO JOB           SALARY
---------- ---------- ----------
        10 Asst.prof       40000
        10 Assoc.prof      60000
        10 prof            80000
        20 Asst.prof       40000
        20 Asst.prof       45000
        20 Assoc.prof      65000
        30 Asst.prof       50000
        30 Assoc.prof      70000
        30 prof            90000
        30 prof            90000

10 rows selected.
```

select deptno,job,sum(salary) Total_Salary from employee group by deptno,job;

```
SQL> select deptno,job,sum(salary) from employee group by deptno,job;

    DEPTNO JOB        SUM(SALARY)
---------- ---------- -----------
        10 prof             80000
        30 Assoc.prof       70000
        10 Assoc.prof       60000
        30 Asst.prof        50000
        20 Asst.prof        85000
        10 Asst.prof        40000
        30 prof            185000
        20 Assoc.prof       65000

8 rows selected.
```

Query: Total Salary paid to each job in each department excluding Assoc Prof

```
SQL> delete from employee where job='Assoc.prof';

3 rows deleted.

SQL> select * from employee;

       ENO ENAME      JOB            SALARY     DEPTNO
---------- ---------- ---------- ---------- ----------
         1 ram        Asst.prof       40000         10
         3 rahat      prof            80000         10
         4 suresh     Asst.prof       40000         20
         5 ruhani     Asst.prof       45000         20
         7 Ankit      Asst.prof       50000         30
         9 Amit       prof            90000         30
        10 vikas      prof            95000         30

7 rows selected.

SQL> select Deptno,job,salary from employee;

    DEPTNO JOB            SALARY
---------- ---------- ----------
        10 Asst.prof       40000
        10 prof            80000
        20 Asst.prof       40000
        20 Asst.prof       45000
        30 Asst.prof       50000
        30 prof            90000
        30 prof            95000

7 rows selected.
```

Grouping by one column :

EXAMPLE :

select deptno,sum(salary) from employee group by deptno;

```
SQL> select deptno,sum(salary) from employee group by deptno;

    DEPTNO SUM(SALARY)
---------- ----------
        30     230000
        20      85000
        10     120000
```

select deptno,count(*) from employee group by deptno;

```
> select deptno,count(*) from employee group by deptno;

 DEPTNO   COUNT(*)
------- ----------
     30          3
     20          2
     10          2
```

Illegal Queries using Group functions :

EXAMPLE :

select deptno,ename,count(*) from employee group by deptno;

```
SQL> select deptno,ename,count(*) from employee group by deptno;
select deptno,ename,count(*) from employee group by deptno
              *
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

select deptno,job,count(*) from employee group by deptno,job;

```
SQL> select deptno,job,count(*) from employee group by deptno,job;

    DEPTNO JOB          COUNT(*)
---------- ---------- ----------
        10 prof                1
        30 Asst.prof           1
        20 Asst.prof           2
        10 Asst.prof           1
        30 prof                2
```

select eno,job,count(*) from employee group by deptno,job;

```
SQL> select eno,job,count(*) from employee group by deptno,job;
select eno,job,count(*) from employee group by deptno,job
       *
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

Restricting groups using HAVING clause :

EXAMPLE :

select deptno,sum(salary) from employee group by deptno having sum(salary)>85000;

```
SQL> select deptno,sum(salary) from employee group by deptno having sum(salary)>85000;

    DEPTNO SUM(SALARY)
---------- -----------
        30      230000
        10      120000
```

select deptno,sum(salary) from employee group by deptno having sum(salary)>5000;

```
SQL> select deptno,sum(salary) from employee group by deptno having sum(salary)>5000;

    DEPTNO SUM(SALARY)
---------- -----------
        30      230000
        20       85000
        10      120000
```

Nesting group functions :

EXAMPLE :

select deptno,avg(slary) from employee group by deptno;

```
SQL> select deptno,avg(salary) from employee group by deptno;

    DEPTNO AVG(SALARY)
---------- -----------
        30  76666.6667
        20       42500
        10       60000
```

select max(avg(salary)) from employee group by deptno;

```
SQL> select max(avg(salary)) from employee group by deptno;

MAX(AVG(SALARY))
----------------
      76666.6667
```

select Min(avg(salary)) from employee group by deptno;

```
SQL>  select Min(avg(salary)) from employee group by deptno;

MIN(AVG(SALARY))
----------------
           42500
```

select deptno,sum(salary) from employee group by deptno;

```
SQL> select deptno,sum(salary) from employee group by deptno;

    DEPTNO SUM(SALARY)
---------- -----------
        30      230000
        20       85000
        10      120000
```

select Min(sum(salary)) from employee group by deptno;

```
SQL> select Min(sum(salary)) from employee group by deptno;

MIN(SUM(SALARY))
----------------
           85000
```

select Max(sum(salary)) from employee group by deptno;

```
SQL> select Max(sum(salary)) from employee group by deptno;

MAX(SUM(SALARY))
----------------
          230000
```

select deptno,count(salary) from employee group by deptno;

```
SQL> select deptno,count(salary) from employee group by deptno;

    DEPTNO COUNT(SALARY)
---------- -------------
        30             3
        20             2
        10             2
```

select max(count(salary)) from employee group by deptno;

```
SQL> select max(count(salary)) from employee group by deptno;

MAX(COUNT(SALARY))
------------------
                 3
```

VIVA QUESTIONS:

1. Illustrate the process of Grouping by One Column
2. Examples of Illegal Queries Using Group Functions
3. How to Rest groups using HAVING Clause?
4. How to use Nesting Group Functions?

EXERCISE : 7

AIM : Retrieve Data from Multiple Tables using the following join operations

- Cartesian Products
- Equijoin
- Non-equijoin
- Outer join
- Self join.

Description :

Cartesian product :

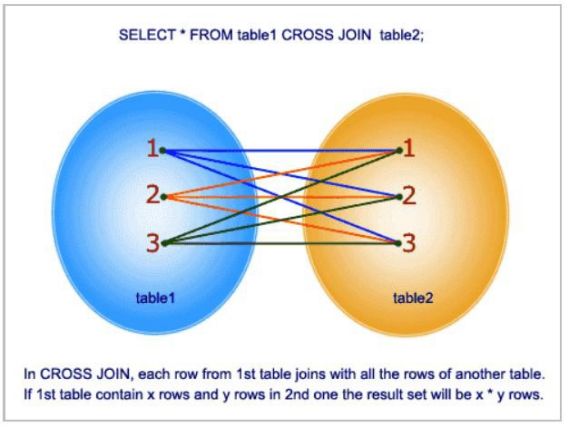
Join:-A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN. This kind of result is called as Cartesian product.

If WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN**.**

Syntax:- SELECT * FROM table1 CROSS JOIN table2;

Pictorial Presentation of Cross Join syntax



Example:

Select * from student, enroll;

Output:

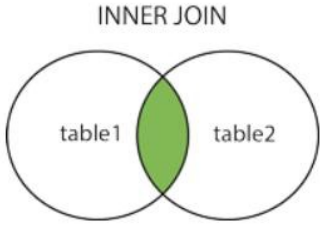
```
SQL> select * from student;

       SID NAME                AGE
---------- ---------- ----------
       101 cse                 15
       102 ece                 16
       103 civil               17
       104 eee                 18
       105 ME                  19

SQL> select * from enroll;

       SID CID
---------- ----------
       101 cs101
       103 cs103
       105 cs105

SQL> select * from student cross join enroll;

       SID NAME                AGE        SID CID
---------- ---------- ---------- ---------- ----------
       101 cse                 15        101 cs101
       102 ece                 16        101 cs101
       103 civil               17        101 cs101
       104 eee                 18        101 cs101
       105 ME                  19        101 cs101
       101 cse                 15        103 cs103
       102 ece                 16        103 cs103
       103 civil               17        103 cs103
       104 eee                 18        103 cs103
       105 ME                  19        103 cs103
       101 cse                 15        105 cs105

       SID NAME                AGE        SID CID
---------- ---------- ---------- ---------- ----------
       102 ece                 16        105 cs105
       103 civil               17        105 cs105
       104 eee                 18        105 cs105
       105 ME                  19        105 cs105

15 rows selected.
```

Equijoin:

The INNER JOIN keyword selects records that have matching values in both tables

Syntax:

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



Example: select * from student INNER JOIN enroll on student.sid=enroll.sid;

Output:

```
SQL> select * from student INNER JOIN enroll on student.sid=enroll.sid;

       SID NAME              AGE        SID CID
---------- ---------- ---------- ---------- ----------
       101 cse                15        101 cs101
       103 civil              17        103 cs103
       105 ME                 19        105 cs105

SQL> select s.sid,s.name,e.cid from student s INNER JOIN enroll e on s.sid=e.sid;

       SID NAME       CID
---------- ---------- ----------
       101 cse        cs101
       103 civil      cs103
       105 ME         cs105
```

Non-equijoin:

Theta Join allows you to merge two tables based on the condition represented by theta. Theta joins work for all comparison operators.

The general case of JOIN operation is called a Theta join. It is denoted by symbol θ

Syntax: Select column_names from table1 join table2 on table1.col1>table2.col1

Example: select * from student join enroll on student.sid>enroll.sid;

Output:

```
SQL> select * from student join enroll on student.sid>enroll.sid;

       SID NAME              AGE        SID CID
---------- ---------- ---------- ---------- ----------
       102 ece                16        101 cs101
       103 civil              17        101 cs101
       104 eee                18        101 cs101
       105 ME                 19        101 cs101
       104 eee                18        103 cs103
       105 ME                 19        103 cs103

6 rows selected.

SQL> select s.sid,s.age,e.cid from student s join enroll e on s.sid<e.sid;

       SID        AGE CID
---------- ---------- ----------
       101         15 cs103
       102         16 cs103
       101         15 cs105
       102         16 cs105
       103         17 cs105
       104         18 cs105

6 rows selected.
```

Outer join:

1. When performing an inner join, rows from either table that are unmatched in the other table are not returned.

2. In an outer join, unmatched rows in one or both tables can be returned. There are a few types of outer joins:

- ✓ LEFT JOIN returns only unmatched rows from the left table.
- ✓ RIGHT JOIN returns only unmatched rows from the right table.
- ✓ FULL OUTER JOIN returns unmatched rows from both tables

LEFT JOIN(left outer join):-

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

Syntax:-

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

Examples:

```
SQL> select * from student left join enroll on student.sid=enroll.sid;

       SID NAME              AGE        SID CID
---------- ---------- ---------- ---------- ----------
       101 cse                15        101 cs101
       103 civil              17        103 cs103
       105 ME                 19        105 cs105
       102 ece                16
       104 eee                18

SQL> select s.sid,s.name,e.sid,e.cid from student s left outer join enroll e on s.sid=e.sid;

       SID NAME              SID CID
---------- ---------- ---------- ----------
       101 cse               101 cs101
       103 civil             103 cs103
       105 ME                105 cs105
       102 ece
       104 eee
```

RIGHT JOIN(right outer join):-

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

Syntax:-

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

Examples:

select * from student right join enroll on student.sid=enroll.sid;

insert into enroll values (107,'cs107');

```
SQL> select * from student right join enroll on student.sid=enroll.sid;

       SID NAME              AGE        SID CID
---------- ---------- ---------- ---------- ----------
       101 cse                15        101 cs101
       103 civil              17        103 cs103
       105 ME                 19        105 cs105

SQL> insert into enroll values (107,'cs107');

1 row created.

SQL> select * from student right join enroll on student.sid=enroll.sid;

       SID NAME              AGE        SID CID
---------- ---------- ---------- ---------- ----------
       101 cse                15        101 cs101
       103 civil              17        103 cs103
       105 ME                 19        105 cs105
                                        107 cs107

SQL> select s.sid,s.name,e.sid,e.cid from student s right join enroll e on s.sid=e.sid;

       SID NAME              SID CID
---------- ---------- ---------- ----------
       101 cse               101 cs101
       103 civil             103 cs103
       105 ME                105 cs105
                             107 cs107
```

Full Outer Join:

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

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

Syntax:-

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

Example: select * from student full join enroll on student.sid=enroll.sid;

Output:

```
SQL> select * from student full join enroll on student.sid=enroll.sid;

       SID NAME              AGE        SID CID
---------- ---------- ---------- ---------- ----------
       101 cse                15        101 cs101
       103 civil              17        103 cs103
       105 ME                 19        105 cs105
       102 ece                16
       104 eee                18
                                        107 cs107

6 rows selected.
```

Self join: self JOIN is a regular join, but the table is joined with itself.

Syntax:

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

Note:-*T1* and *T2* are different table aliases for the same table.

Example:

Output:

```
SQL> select s1.sid,s2.age from student s1,student s2 where s1.sid=s2.sid;

       SID        AGE
---------- ----------
       101         15
       102         16
       103         17
       104         18
       105         19

SQL> select s1.sid,s2.age from student s1,student s2 where s1.sid>s2.sid;

       SID        AGE
---------- ----------
       102         15
       103         15
       104         15
       105         15
       103         16
       104         16
       105         16
       104         17
       105         17
       105         18

10 rows selected.
```

VIVA QUESTIONS:

1. What is Cartesian Product?
2. Differentiate is Equi join & Non-Equi Join?
3. Describe about Self join

EXERCISE : 8

AIM : Execute Set operations on various Relations.

• UNION • UNION ALL • INTERSECT

• MINUS.

Description :

Set operations in sql:

UNION :

Let R and S are two union compatible relations then, union operation returns the tuples that are present in R or s or both.

- Two relational instances are said to be union compatible if the following conditions are hold.
    1) They have the same number of columns.
    2) Corresponding columns taken in order from left to right have same data type.

1. Find the names of sailors who have reserved red or green boat.

Query

```
select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color =
'red'
UNION
 select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color =
'green';
```

Output

```
SQL> select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'red' UNION select s.sname from sailor1 s,reserve1 r,boat1 b w
here s.sid = r.sid and r.bid = b.bid and b.color = 'green';

SNAME
----------
dustin
horatio
lubber
```

2. Find all sid's of sailors who have rating of 10 or reserved boat no.104.

Query

```
 select s.sid from sailor1 s where s.rating=10
UNION
 select r.sid from reserve1 r where r.bid = 104;
```

Output

```
select s.sid from sailor1 s where s.rating=10 UNION  select r.sid from reserve1 r where r.bid = 104;

  SID
-----
   22
   31
   58
   71
```

UNION ALL :

The UNION ALL command combines the result set of two or more SELECT statements (allows duplicate values).

Syntax :

SELECT column_name(s) FROM table1

UNION ALL:

SELECT column_name(s) FROM table2;

Example :

```
SQL> select s.sid,s.sname from sailors s,reserves r,boats b where s.sid=r.sid and r.bid=b.bid and b.color='red'
  2  union all
  3  select s2.sid,s2.sname from sailors s2,reserves r2,boats b2 where s2.sid=r2.sid and r2.bid=b2.bid and b2.color='green'
  4  ;

       SID SNAME
---------- ----------
        22 Dustin
        22 Dustin
        31 Lubber
        31 Lubber
        64 Horatio
        22 Dustin
        31 Lubber
        74 Horatio

8 rows selected.
```

INTERSECT :

Let R and S are two union compatible relations then, intersect operation returns the tuples that are common in both the relations.

1. Find the names of sailors who have reserved red and green boat.

Query

select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'red'

INTERSECT

select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'green';

Output

```
SQL> select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'red'
  2  INTERSECT
  3   select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'green';

SNAME
----------
dustin
horatio
lubber
```

MINUS :

Let R and S are two union compatible relations then, intersect operation returns the tuples that are present in R but not in S.

1. Find the sid's of sailors who have reserved red but not green boat.

Query

select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color = 'red' MINUS select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color ='green';

Output

```
SQL> select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color = 'red' MINUS  select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color ='green';

     SID
---------
      64
```

VIVA QUESTION:

1. List various set Operations
2. Differentiate UNION and UNION ALL

EXERCISE : 9

AIM : Execute Sub Queries and Co-Related Nested Queries on Relations.

• Implement

o Single-row subquery o Multiple-row subquery

• Using Group Functions in a Subquery

• Using HAVING Clause with Subqueries

• Using Null Values in a Subquery

• Data retrieval using Correlated Subqueries

o EXISTS Operator o NOT EXISTS Operator .

Description :

NESTED QUERIES

A query embedded inside another query is called a sub query. Inner query executes initially only once and that result will be used by all the tuples of outer query.

Co-Related nested queries: Correlated subquery is a query in which the inner query is executed for each row of the outer query.

Implement :

- **Single row subquery**:

  A single row subquery returns zero or one row to the outer SQL statement. You can place a subquery in a WHERE clause, a HAVING clause, or a FROM clause of a SELECT statement.

EXAMPLE :

1. Find the name and age of the oldest sailor.

select s.sname,s.age from sailors s where age=(select max(s2.age) from sailors s2);

```
SQL> select s.sname,s.age from sailors s where age=(select max(s2.age) from sailors s2);

SNAME                          AGE
------------------- ----------
bob                          63.5
```

- **Multiple row subquery :**
  - ✓ Multiple-row subqueries are nested queries that can return more than one row of results to the parent query. Multiple-row subqueries are used most commonly in WHERE and HAVING clauses.
  - ✓ Since it returns multiple rows, it must be handled by set comparison operators (IN, ALL, ANY).
  - ✓ While IN operator holds the same meaning as discussed in the earlier chapter, ANY operator compares a specified value to each value returned by the subquery while ALL compares a value to every value returned by a subquery.
  - ✓ The below query will show the error because single-row subquery returns multiple rows

EXAMPLE :

1. Find sailors whose rating is better than some sailor called Horatio

select s.sid from sailors s where s.rating > ANY(select s2.rating from sailors s2 where s2.sname='Horatio');

```
SQL> select s.sid from sailors s where s.rating>ANY(select s2.rating from sailors s2 where s2.sname='Hor
atio');

       SID
----------
        58
        71
        74
        31
        32
```

Using Group functions in a subqueries :

EXAMPLE :

1. Find the names of sailors who are older than oldest sailor with a rating of 10.

select s.sname from sailors s where s.age>(select max(s2.age) from sailors s2 where s2.rating=10);

```
SQL> select s.sname from sailors s where s.age>(select max(s2.age) from sailors s2 where s2.rating=10);

SNAME
--------------------
dustin
lubber
bob
```

Using HAVING clauses with subqueries :

- The HAVING clause is used along with the GROUP BY clause. The HAVING clause can be used to select and reject row groups.
- The format of the HAVING clause is similar to the WHERE clause, consisting of the keyword HAVING followed by a search condition.
- The HAVING clause thus specifies a search condition for groups.

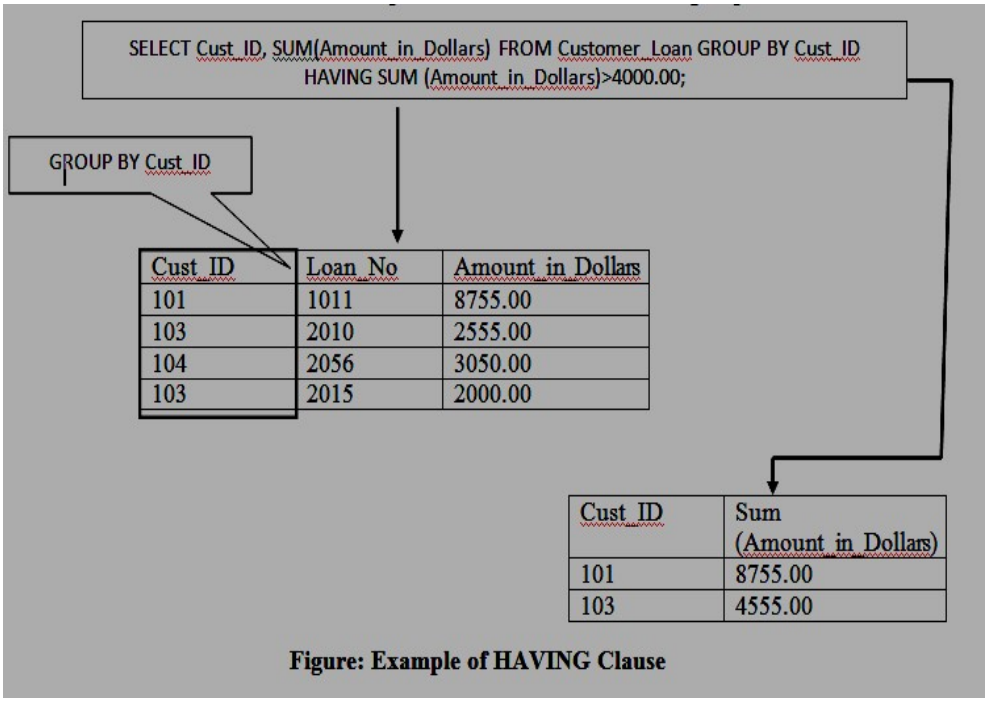


Figure: Example of HAVING Clause

EXAMPLE :

1. Find the average age of sailors for each rating level that has at least two sailors.

select s.rating,avg(s.age) from sailors s group by s.rating having 1<(select count(*) from sailors s2 where s.rating=s2.rating);

```
SQL> select s.rating,avg(s.age) from sailors s group by s.rating having 1<(select count(*) from sailors s2 where s.rating=s2.rating);

    RATING AVG(S.AGE)
---------- ----------
         8       40.5
         7         40
         3       44.5
        10       25.5
```

2. Find the average age of sailors who are of voting age (i.e., at least 18 years old) for each rating level that has at least two sailors.

select s.rating,avg(s.age) from sailors s where s.age>=18 group by s.rating having 1<(select count(*) from sailors s2 where s.rating=s2.rating);

```
SQL> select s.rating,avg(s.age) from sailors s where s.age>=18 group by s.rating having 1<(select count(*) from sailors s2 where s.rating=s2.rating);

    RATING AVG(S.AGE)
---------- ----------
         8       40.5
         7         40
         3       44.5
        10         35
```

Using NULL values in a subquery :

A field with a NULL value is a field with no value.

Syntax :

a. **IS NULL :**

Syntax

SELECT column_names

FROM table_name

WHERE column_name IS NULL;

IS NOT NULL :

Syntax

SELECT column_names

FROM table_name

WHERE column_name IS NOT NULL;

Example :

Select sid from sailors where sid is null;

```
SQL> Select sid from sailors where sid is null;

no rows selected
```

Select sid from sailors where sid is not null;

```
SQL> Select sid from sailors where sid is not null;

       SID
----------
        22
        29
        31
        32
        58
        64
        71
        74
        85
        95

10 rows selected.
```

Data retrieval using correlated sub queries :

- **EXISTS operator :**

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

Example :

1.Find the names of sailors who have reserved boat number 103.

select s.sname from sailors s where EXISTS(select * from reserves r where r.bid=103 and r.sid=s.sid);

```
SQL> select s.sname from sailors s where EXISTS(select * from reserves r where r.bid=103 and r.sid=s.sid);

SNAME
--------------------
dustin
lubber
horatio
```

- **NOT EXISTS operator : Negated version of EXISTS**

Example:

1. Find the sid's and names of sailors who have not reserved boat number 103.

select s.sid,s.sname from sailors s where not exists(select * from reserves r where r.bid=103 and r.sid=s.sid);

```
SQL> select s.sid,s.sname from sailors s where not exists(select * from reserves r where r.bid=103 and r.sid=s.sid);

       SID SNAME
---------- --------------------
        71 zorba
        85 art
        64 horatio
        58 rusty
        32 andy
        29 brutus
        95 bob

7 rows selected.
```

VIVA QUESTIONS

1. What is nested query?
2. What is co-related nested query?
3. Differentiate Single row sub query and Multiple row sub query.

EXERCISE : 10

AIM : Perform following operations on views

• Simple Views

• Complex Views

• Modifying a View DML Operations on a View

• Denying DML Operations on view

• Removing a View

Description :

a) Simple views :

It is the view created by involving only single table.

Syntax :

CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;

Example:

create view sailorsv As select * from sailors**;**

Output:

```
SQL> create view sailorsv As select * from sailors;

View created.

SQL> select * from sailorsv;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 Dustin              7         45
        29 Brutus              1         33
        31 Lubber              8       55.5
        32 Andy                8       25.5
        58 Rusty              10         35
        64 Horatio             7         35
        71 Zorba              10         16
        74 Horatio             9         35
        85 Art                 3       25.5
        95 Bob                 3       63.5

10 rows selected.
```

b) Complex views :

When the view is created based on multiple tables then it is known as a complex view in SQL Server. The most important point that we need to remember is, on a complex view in SQL Server, we may or may not perform the DML operations and more ever the complex view may not update the data correctly on the underlying database tables.

1. In a Complex View, if your update statement affects one base table, then the update succeeded but it may or may not update the data correctly.

2. if your update statement affects more than one table, then the update failed and we will get an error message stating "**View or function 'vwEmployeesByDepartment'** is not updatable because the modification affects multiple base tables".

Example:

```
create view dept_stat1
 As
select dname,sum(sal) Total_Salary,Min(sal) Min_sal,Max(sal) Max_Sal
 from emp e,dept d
 where e.deptno=d.deptno
group by dname;
```

output:

```
SQL> create view dept_stat1
  2   As
  3  select dname,sum(sal) Total_Salary,Min(sal) Min_sal,Max(sal) Max_Sal
  4   from emp e,dept d
  5   where e.deptno=d.deptno
  6  group by dname;

View created.

SQL> select * from dept_stat1;

DNAME      TOTAL_SALARY    MIN_SAL    MAX_SAL
---------- ------------ ---------- ----------
accounting         7450       2450       5000
sales              6950       1250       2850
research           6775        800       3000
```

c) Modifying a view :

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

d) DML operations on a view :

The DML operations INSERT, UPDATE, and DELETE can be performed on simple views. These operations can be used to change the data in the underlying base tables.
If you create a view that allows users to view restricted information using the WHERE clause, users can still perform DML operations on all columns of the view.

INSERT

Synatx:

Example: insert into sailorsv values(105,'srgec',4,21);

```
SQL> insert into sailorsv values(105,'srgec',4,21);

1 row created.
```

select * from sailorsv;

```
SQL> select * from sailorsv;

       SID SNAME          RATING          AGE
---------- ---------- ---------- ----------
        22 Dustin              7         45
        29 Brutus              1         33
        31 Lubber              8       55.5
        32 Andy                8       25.5
        58 Rusty              10         35
        64 Horatio             7         35
        71 Zorba              10         16
        74 Horatio             9         35
        85 Art                 3       25.5
        95 Bob                 3       63.5
       105 srgec               4         21

11 rows selected.
```

```
SQL> select * from sailors;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 Dustin              7         45
        29 Brutus              1         33
        31 Lubber              8       55.5
        32 Andy                8       25.5
        58 Rusty              10         35
        64 Horatio             7         35
        71 Zorba              10         16
        74 Horatio             9         35
        85 Art                 3       25.5
        95 Bob                 3       63.5
       105 srgec               4         21

11 rows selected.
```

UPDATE:

Example:

update sailorsv set sname='gec' where sid=105;

Output:

```
SQL> update sailorsv set sname='gec' where sid=105;

1 row updated.

SQL> select * from sailorsv;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 Dustin              7         45
        29 Brutus              1         33
        31 Lubber              8       55.5
        32 Andy                8       25.5
        58 Rusty              10         35
        64 Horatio             7         35
        71 Zorba              10         16
        74 Horatio             9         35
        85 Art                 3       25.5
        95 Bob                 3       63.5
       105 gec                 4         21

11 rows selected.

SQL> select * from sailors;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 Dustin              7         45
        29 Brutus              1         33
        31 Lubber              8       55.5
        32 Andy                8       25.5
        58 Rusty              10         35
        64 Horatio             7         35
        71 Zorba              10         16
        74 Horatio             9         35
        85 Art                 3       25.5
        95 Bob                 3       63.5
       105 gec                 4         21

11 rows selected.
```

e) Denying DML operations on view :

- The deny statement denies permission to a principal for accessing the securable. For example, if we do not want a user to perform a delete operation on a table, DBA can deny delete permission on the object.
- The WITH READ ONLY option ensures that no DML operations occur through the view.
- Any attempt to execute an INSERT, UPDATE, or DELETE statement will result in an Oracle server error.

Syntax :

```
CREATE OR REPLACE VIEW view_dept50
AS SELECT department_id, employee_id, first_name, last_name, salary
FROM employees
WHERE department_id = 50
WITH READ ONLY;
```

INSERT EXAMPLE:

```
SQL> create view sailorsv12 As select * from sailors with read only;

View created.

SQL> select * from sailorsv12;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 Dustin              7         45
        29 Brutus              1         33
        31 Lubber              8       55.5
        32 Andy                8       25.5
        58 Rusty              10         35
        64 Horatio             7         35
        71 Zorba              10         16
        74 Horatio             9         35
        85 Art                 3       25.5
        95 Bob                 3       63.5
       105 gec                 4         21

11 rows selected.

SQL> insert into sailorsv12 values(106,'cse',5,22);
insert into sailorsv12 values(106,'cse',5,22)
*
ERROR at line 1:
ORA-01733: virtual column not allowed here
```

DELETE:

EXAMPLE:

```
SQL> DELETE FROM SAILORSV12 WHERE SID=105;
DELETE FROM SAILORSV12 WHERE SID=105
            *
ERROR at line 1:
ORA-01752: cannot delete from view without exactly one key-preserved table
```

f) Removing a view :

Syntax : Drop view viewname;

Example: Drop view sailorsv;

Output:

```
SQL> drop view sailorsv;

View dropped.
```

VIVA QUESTIONS

1. What is view?
2. Is view updatable?
3. What are the advantages of views?

EXERCISE : 11

AIM : Develop the following PL/SQL programs

• Simple PL/SQL programs

• PL/SQL programs Using Control structures.

o Conditional structures o Iterative structures

• PL/SQL program using the following exception handling mechanisms.

o Pre defined exceptions o user defined exceptions.

Description :

Simple PL/SQL programs .:

1. **Aim:-Sum of two numbers**

Sourcecode:-

```
SQL> declare
        x integer;
        y integer;
        z integer;
    begin
        x:=10;
        y:=20;
        z:=x+y;
     dbms_output.put_line('sum is' ||Z);
    end;
    /
```

Output:-

```
sum is30
PL/SQL procedure successfully completed.
```

2. Aim:-Sum of two numbers reading input from user

Sourcecode:-

```
SQL> declare
        x integer;
        y integer;
        z integer;
    begin
        x:=&x;
        y:=&y;
```

```
        z:=x+y;
    dbms_output.put_line(x||'+'||y||'='||z);
    end;
    /
```

Output:-

```
Enter value for x: 2
old   6: x:=&x;
new   6: x:=2;
Enter value for y: 2
old   7: y:=&y;
new   7: y:=2;
2+2=4
PL/SQL procedure successfully completed.
```

PL/SQL programs using control structures :

- **Conditional structures :**

IF-THEN-ELSIF statement

1. **Aim:-Greatest of three numbers**

Sourcecode;-

```
SQL> declare
        a number:=46;
        b number:=67;
        c number:=21;
    begin
         if  a>b and a>c then
        dbms_output.put_line('greatest number is'||a);
        elsif b>a and b>c then
        dbms_output.put_line('greatest number is'||b);
        else
        dbms_output.put_line('greatest number is'||c);
        end if;
     end;
    /
```

OUTPUT:-

greatest number is67

PL/SQL procedure successfully completed.

CASE

```
DECLARE
  grade char(1) := 'C';
BEGIN
  CASE grade
    when 'A' then dbms_output.put_line('Distinction');
    when 'B' then dbms_output.put_line('First class');
    when 'C' then dbms_output.put_line('Second class');
    when 'D' then dbms_output.put_line('Pass class');
    else dbms_output.put_line('Failed');
  END CASE;
END;
/
```

- **Iterative structures :**

  Exit loop.

```
DECLARE
        i NUMBER := 1;
BEGIN
LOOP
        EXIT WHEN i>5;
        dbms_output.put_line(i);
        i := i+1;
        END LOOP;
END;
1
2
3
4
5
```

PL/SQL procedure successfully completed.

While loop

1. AIM:-SUM OF EVEN NUMBERS USER INPUT DYNAMICALLY

SOURCODE:-

```
SQL> declare
        x integer:=2;
        y integer;
        s integer:=0;
    begin
        y:=&y;
        while x<=y loop
            dbms_output.put_line(x);
        s:=s+x;
        x:=x+2;
        end loop;
    dbms_output.put_line('sum of even numbers is' || s);
    end;
    /
```

Output:-

```
Enter value for y: 10
old   6:    y:=&y;
new   6:    y:=10;
2
4
6
8
10
sum of even numbers is30
PL/SQL procedure successfully completed.
```

For loop

1. **Aim:-**TO PRINT NATURAL NUMBERS

Sourcecode:-

```
SQL> declare
        a integer;
    begin
        for a in 10 .. 20 loop
        dbms_output.put_line('value of a:'||a);
        end loop;
        end;
        /
```

Output:-

```
value of a:10
value of a:11
value of a:12
value of a:13
value of a:14
value of a:15
value of a:16
value of a:17
value of a:18
value of a:19
value of a:20

PL/SQL procedure successfully completed.
```

PL/SQL GOTO Statement

```
DECLARE
  a number(2) := 50;
BEGIN
  <<loopstart>>
  -- while loop execution
  WHILE a < 60LOOP
    dbms_output.put_line ('value of a: ' || a);
```

```
    a := a + 1;
    IF a = 55 THEN
      a := a + 1;
      GOTO loopstart;
    END IF;
  END LOOP;
END;
/
```

Output:

value of a: 50

value of a: 51

value of a: 52

value of a: 53

value of a: 54

value of a: 56

value of a: 57

value of a: 58

value of a: 59

PL/SQL program using the following exception handling mechanisms .

- Exceptions are runtime errors or unexpected events that occur during the execution of a PL/SQL code block.
- The oracle engine is the first one to identify such an exception and it immediately tries to resolve it by default exception handler.
- The default exception handler is a block of code predefined in the memory to take the appropriate action against exceptions.
- Exception handling can be done in the EXCEPTION part of PL/SQL program code block. Following is the syntax for it:

```
DECLARE
        -- Declaration statements;
BEGIN
```

```
        -- SQL statements;
        -- Procedural statements;
EXCEPTION
        -- Exception handling statements;
END;
```

There are two types of exceptions:

System (pre-defined) Exceptions

User-defined Exceptions

Let's cover both types of exceptions one by one.

In order to handle common exceptions that occur while running PL/SQL code, there are two types of exception handlers in oracle:

- Named Exception Handler
- Numbered Exception Handler

Named Exception Handling

Such exceptions are the predefined names given by oracle for those exceptions that occur most commonly.

Following is the syntax for handling named exception:

```
EXCEPTION
        WHEN<exception_name>THEN
```

Named Exception	Meaning
LOGIN_DENIED	Occurs when invalid username or invalid password is given while connecting to Oracle.
TOO_MANY_ROWS	Occurs when select statement returns more than one row.
VALUE_ERROR	Occurs when invalid datatype or size is given by the user.
NO_DATA_FOUND	Occurs when no records are found.
DUP_VAL_ON_INDEX	Occurs when a unique constraint is applied on some column and execution of Insert or Update leads to creation of duplicate records for that column.

PROGRAM_ERROR	Occurs when internal error arise in program.
ZERO_DIVIDE	Occurs when the division of any variable value is done by zero.

```
-- take action
```

There are number of pre-defined named exceptions available by default. Few of them are shown in the table below, along with their meanings:

Below we have a simple PL/SQL code block, to demonstrate the use of **Named Exception Handler,**

```
set serveroutput on;
DECLARE
        a int;
        b int;
        c int;
BEGIN
        a := &a;
        b := &b;
        c := a/b;
        dbms_output.put_line('RESULT=' || c);
EXCEPTION
        when ZERO_DIVIDE then
                dbms_output.put_line('Division by 0 is not possible');
END;
```

Output:-

```
Enter the value for a:10
Enter the value for b:0
Division by 0 is not possible
PL/SQL procedure successfully completed.
```

Numbered Exception Handling

In oracle, some of the pre-defined exceptions are numbered in the form of four integers preceded by a hyphen symbol. To handle such exceptions we should assign a name to them before using them. This can be done by using the Pragma exception technique in which a numbered exception

handler is bound to a name. For this purpose, we use a keyword in PL/SQL program and write a statement that binds a name to a numbered exception using the following syntax and this statement is written in the DECLARE section of program:

pragma exception_init(exception_name, exception _number);

where, pragma exception_init(case doesn't matter) is a keyword indicating Pragma exception technique with two arguments:

exception_name, which is a user-defined name given to a predefined numbered exception if it occurs.

exception_number, is the number allotted to the exception by oracle.

Below we have a table with Student's data in it.

ROLLNO	SNAME	AGE	COURSE
11	Anu	20	BSC
12	Asha	21	BCOM
13	Arpit	18	BCA
14	Chetan	20	BCA
15	Nihal	19	BBA

In the PL/SQL program below, we will be using the above table student to demonstrate the use of Numbered Exception,

```
set serveroutput on;

DECLARE
        sno student.rollno%type;
        snm student.sname%type;
        s_age student.age%type;
        cr student.course%type;
        -- Exception name declared below
        already_exist EXCEPTION;
        -- pragma statement to provide name to numbered exception
```

```
        pragma exception_init(already_exist, -1);
BEGIN
        sno:=&rollno;
        snm:='&sname';
        s_age:=&age;
        cr:='&course';
        INSERT into student values(sno, snm, s_age, cr);
        dbms_output.put_line('Record inserted');
        EXCEPTION
                WHEN already_exist THEN
                        dbms_output.put_line('Record already exist');
END;
```

```
Enter the value for sno:11
Enter the value for snm:heena
Enter the value for s_age:20
Enter the value for cr:bsc
Record already exist
PL/SQL procedure successfully completed.
```

Description: In the above program, whenever a primary key concept(records should be unique and not null) is violated oracle generates a numbered exception by -1 and that is why when rollno entered by user during execution of above program was 11. The exception section of the program comes into action and message is displayed before the user Record already exist.
Using pragma keyword in the declare section of the program already_exist string is mapped to a numbered exception -1.

User-defined Exception

In any program, there is a possibility that a number of errors can occur that may not be considered as exceptions by oracle. In that case, an exception can be defined by the programmer while writing the code such type of exceptions are called User-defined exception. User defined exceptions are in general defined to handle special cases where our code can generate exception due to our code logic.

Also, in your code logic, you can explicitly specify to genrate an exception using the RAISE keyword and then handle it using the EXCEPTION block.

Following is the syntax for it,

```
DECLARE
        <exception name> EXCEPTION
BEGIN
        <sql sentence>
        If <test_condition> THEN
                RAISE <exception_name>;
        END IF;
        EXCEPTION
                WHEN <exception_name> THEN
                        -- some action
END;
```

Let's take an example to understand how to use user-defined exception. Below we have a simple example,

ROLLNO	SNAME	Total_Courses
11	Anu	2
12	Asha	1
13	Arpit	3
14	Chetan	1

In the PL/SQL program below, we will be using the above table student to demonstrate the use of User-defined Exception,

```
set serveroutput on;

DECLARE
        sno student.rollno%type;
        snm student.sname%type;
        crno student.total_course%type;
        invalid_total EXCEPTION;
BEGIN
        sno := &rollno;
        snm := '&sname';
        crno:=total_courses;
        IF (crno > 3) THEN
                RAISE invalid_total;
        END IF;
        INSERT into student values(sno, snm, crno);
        EXCEPTION
                WHEN invalid_total THEN
                        dbms_output.put_line('Total number of courses cannot be more than 3');
END;
```

Output:

```
Enter the value for sno:15
Enter the value for snm:Akash
Enter the value for crno:5
Total number of courses cannot be more than 3
PL/SQL procedure successfully completed.
```

User-defined exception called invalid_total is used which is generated when total number of courses is greater than 3(when a student can be enrolled maximum in 3 courses)

VIVA QUESTIONS

1. What is PL/SQL?
2. What is Exception Handling?

EXERCISE :12

AIM : Implement a PL/SQL block using triggers for transaction operations of a typical application

Description :

Trigger:

Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs.

Example: a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]

create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
[before | after]: This specifies when the trigger will be fired.
{insert | update | delete}: This specifies the DML operation.
on [table_name]: This specifies the name of the table associated with the trigger.
[for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
[trigger_body]: This provides the operation to be performed as trigger is fired Each trigger is attached to a single, specified table in the database.

SQL> create table customers(id int,name varchar2(10),age int,address varchar2(10),salary int);
Table created.
SQL> select * from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	ramesh	23	allahabad	20000
2	suresh	22	kanpur	22000

3	mahesh	24	ghaziabad	24000
4	chandan	25	noida	26000
5	alex	21	paris	28000
6	sunita	20	delhi	30000

6 rows selected.

SQL> set serveroutput on;

Create trigger:

The following trigger will display the salary difference between the old values and new values:

```
SQL> CREATE OR REPLACE TRIGGER display_salary_changes
        BEFORE DELETE OR INSERT OR UPDATE
        ON customers
        FOR EACH ROW
        WHEN (NEW.ID > 0)
        DECLARE
                sal_diff number;
        BEGIN
                sal_diff := :NEW.salary  - :OLD.salary;
                dbms_output.put_line('Old salary: ' || :OLD.salary);
                dbms_output.put_line('New salary: ' || :NEW.salary);
                dbms_output.put_line('Salary difference: ' || sal_diff);
         END;
        /
```

Trigger created.

Above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using DELETE operation on the table.

```
SQL> DECLARE
        total_rows number(2);
        BEGIN
                UPDATE  customers
                SET salary = salary + 5000;
                IF sql%notfound THEN
                        dbms_output.put_line('no customers updated');
```

```
        ELSIF sql%found THEN
                total_rows := sql%rowcount;
        dbms_output.put_line( total_rows || ' customers updated ');
        END IF;
END;
/
```

```
Old salary: 20000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
Old salary: 26000
New salary: 31000
Salary difference: 5000
Old salary: 28000
New salary: 33000
Salary difference: 5000
Old salary: 30000
New salary: 35000
Salary difference: 5000

6 customers updated
PL/SQL procedure successfully completed.
```

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table:

```
INSERT INTO customers VALUES (7,'Karthik',22,'GEC',47500);
```

When a record is created in CUSTOMERS table, above create trigger display_salary_changes will be fired and it will display the following result:

Output:-

SQL> INSERT INTO customers VALUES (7,'Karthik',22,'GEC',47500);

Old salary:

New salary: 47500

Salary difference:

1 row created.

Descrpition:-

SQL (Implicit) Cursor Attribute

1. A SQL (implicit) cursor is opened by the database to process each SQL statement that is not associated with an explicit cursor.
2. Every SQL (implicit) cursor has six attributes, each of which returns useful information about the execution of a data manipulation statement.

Keyword and Parameter Descriptions

%BULK_ROWCOUNT

A composite attribute designed for use with the FORALL statement. This attribute acts like an index-by table. Its ith element stores the number of rows processed by the ith execution of an UPDATE or DELETE statement. If the ith execution affects no rows, %BULK_ROWCOUNT(i) returns zero.

%BULK_EXCEPTIONS

An associative array that stores information about any exceptions encountered by a FORALL statement that uses the SAVE EXCEPTIONS clause.

You must loop through its elements to determine where the exceptions occurred and what they were. For each index value i between 1 and SQL%BULK_EXCEPTIONS.COUNT, SQL%BULK_EXCEPTIONS(i).ERROR_INDEX specifies which iteration of the FORALL loop caused an exception. SQL%BULK_EXCEPTIONS(i).ERROR_CODE specifies the Oracle Database error code that corresponds to the exception.

%FOUND

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

%ISOPEN

Always returns FALSE, because the database closes the SQL cursor automatically after executing its associated SQL statement.

%NOTFOUND

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

%ROWCOUNT

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

2. Convert employee name into uppercase whenever an employee record is inserted or updated.

Trigger to fire before the insert or update.

```
SQL> create table Employeee(ID VARCHAR2(4 BYTE) NOT NULL,
        First_Name VARCHAR2(10 BYTE),
        Last_Name VARCHAR2(10 BYTE),
        Start_Date DATE,
         End_Date DATE,
        Salary NUMBER(8,2),
        City VARCHAR2(10 BYTE),
        Description VARCHAR2(15 BYTE) )
 /
```

Table created.

```
SQL> CREATE OR REPLACE TRIGGER employeee_insert_update
        BEFORE INSERT OR UPDATE
        ON employeee
        FOR EACH ROW
        DECLARE
        dup_flag INTEGER;
        BEGIN
         --Force all employee names to uppercase.
        :NEW.first_name := UPPER(:NEW.first_name);
        END;
        /
Trigger created.
SQL> insert into Employeee(ID,First_Name, Last_Name, Start_Date, End_Date, Salary, City,
Description)
 values ('01','karthik', 'jayavarapu', to_date('20200101','YYYYMMDD'),
to_date('20200110','YYYYMMDD'), 27500, 'gec', 'Professor')
  /
1 row created.
SQL> insert into Employeee(ID,First_Name, Last_Name, Start_Date, End_Date, Salary, City,
Description)
 values ('02','siva', 'prasad', to_date('20200101','YYYYMMDD'),
to_date('20200110','YYYYMMDD'), 97500, 'gec', 'Professor')
  /
1 row created.
SQL> set pagesize 100;
SQL> set linesize 100;
SQL> select * from employeee;
ID   FIRST_NAME LAST_NAME  START_DAT END_DATE      SALARY CITY
DESCRIPTION
---- ---------- ---------- --------- --------- ---------- ---------- ---------------
01   KARTHIK    jayavarapu 01-JAN-20 10-JAN-20      27500 gec        Professor
02   SIVA       prasad     01-JAN-20 10-JAN-20      97500 gec        Professor
```

EXAMPLE-2

```
SQL>  create table empa(id number(3),name varchar2(10),income number(4),expence
number(3),savings number(3));
Table created.
SQL> insert into empa values(2,'kumar',2500,150,650);
1 row created.
SQL>  insert into empa values(3,'venky',5000,900,950);
1 row created.
SQL>  insert into empa values(4,'anish',9999,999,999);
1 row created.
SQL>  select * from empa;

        ID NAME          INCOME    EXPENCE    SAVINGS
---------- ---------- ---------- ---------- ----------
         2 kumar          2500        150        650
         3 venky          5000        900        950
         4 anish          9999        999        999
```

TYPE 1- TRIGGER AFTER UPDATE

```
SQL> CREATE OR REPLACE TRIGGER after_update
        AFTER UPDATE OR INSERT OR DELETE ON empa
        FOR EACH ROW
        BEGIN
        IF UPDATING THEN
        DBMS_OUTPUT.PUT_LINE('TABLE IS UPDATED');
        ELSIF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('TABLE IS INSERTED');
        ELSIF DELETING THEN
        DBMS_OUTPUT.PUT_LINE('TABLE IS DELETED');
        END IF;
        END;
        /
Trigger created.
SQL> SET SERVEROUTPUT ON;
```

```
SQL> update empa set income =9000 where name='kumar';
TABLE IS UPDATED
1 row updated.
SQL> insert into empa values(40,'Chandru',700,250,80);
TABLE IS INSERTED
1 row created.
SQL>DELETE FROM EMPA WHERE ID = 4;
TABLE IS DELETED
```

Create a Trigger to check the age valid or not Using Message Alert

```
SQL> CREATE TABLE TRIG(NAME CHAR(10),AGE NUMBER(3));
Table created.
SQL> DESC TRIG;
 Name                          Null?    Type
 ----------------------------------------- -------- ----------------------------
 NAME                                      CHAR(10)
 AGE                                       NUMBER(3)

SQL> CREATE TRIGGER TRIGNEW
        AFTER INSERT OR UPDATE OF AGE ON TRIG
        FOR EACH ROW
        BEGIN
                IF(:NEW.AGE<0) THEN
                        DBMS_OUTPUT.PUT_LINE('INVALID AGE');
                ELSE
                        DBMS_OUTPUT.PUT_LINE('VALID AGE');
                END IF;
        END;
        /
Trigger created.
SQL>  insert into trig values('abc',15);
VALID AGE
1 row created.
SQL>  insert into trig values('xyz',-12);
INVALID AGE
```

```
1 row created.
SQL> SELECT * FROM TRIG;
NAME            AGE
---------- ----------
abc              15
xyz             -12
```

Create a Trigger to check the age valid and Raise appropriate error code and error message.

```
SQL>  create table data(name char(10),age number(3));
Table created.

SQL>  desc data;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NAME                                               CHAR(10)
 AGE                                                NUMBER(3)

SQL> CREATE TRIGGER DATACHECK
        AFTER INSERT OR UPDATE OF AGE ON DATA
        FOR EACH ROW
        BEGIN
        IF(:NEW.AGE<0) THEN
  RAISE_APPLICATION_ERROR(-20000,'NO NEGATIVE AGE ALLOWED');
        END IF;
        END;
        /
Trigger created.

SQL> INSERT INTO DATA VALUES('ABC',10);
1 row created.
SQL>  INSERT INTO DATA VALUES ('DEF',-15);
 INSERT INTO DATA VALUES ('DEF',-15)
           *
ERROR at line 1:
ORA-20000: NO NEGATIVE AGE ALLOWED
```

ORA-06512: at "JK.DATACHECK", line 3

ORA-04088: error during execution of trigger 'JK.DATACHECK'

SQL> SELECT * FROM DATA;

NAME AGE

---------- ----------

ABC 10

2. Statementlevel Trigger:

Description:

- Statement level triggers executes only once for each single transaction.
- Used for enforcing all additional security on the transactions performed on the table.
- "FOR EACH ROW" clause is omitted in CREATE TRIGGER command
- If 1500 rows are to be inserted into a table, the statement level trigger would execute only once.

Sql>create table student(id number(10) primary key,marks number(10));

Output:Table created.

Sql>create table count(des varchar(20));

Output:Table created.

Sql>create or replace trigger st_lev

After

insert or update or delete on student

begin

insert into count values('stmt fixed');

end;

/

Output:Trigger created.

Sql>insert into student values(1,80);

Output:1 row inserted.

Sql>insert into student values(2,45);

1 row created

Sql>select * from count;

VIVA QUESTIONS:

1. What is Trigger?
2. Syntax for creating a trigger.
3. How many types of triggers are there and what are they?

Additional Experiment-1

Aim:-Demonstration of database connectivity

Java Program to connect to oracle DB:

```
import java.sql.*;
class ConnectToOraDB {
        public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");
            System.out.println(".....Connected to Oracle
Database....."+con.getClass());
        }
}
```

Output:

```
E:\acedamic\f\j2ee\jdbc>javac ConnectToOraDB.java

E:\acedamic\f\j2ee\jdbc>java ConnectToOraDB
.....Connected to Oracle Database.....class oracle.jdbc.driver.OracleConne
Ction
```

Java program to create table in oracle DB:

```
import java.sql.*;
class CreateTable {
        public static void main(String args[])throws Exception{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");
            System.out.println(".....Connected to Oracle Database.....");
            Statement stmt=con.createStatement();
            String sql="create table std(sid number(4),sname varchar2(10),marks
number(3))";
            stmt.executeUpdate(sql);
        }
}
```

Output:

E:\acedamic\f\j2ee\jdbc>javac CreateTable.java

E:\acedamic\f\j2ee\jdbc>java CreateTable
.....Connected to Oracle Database.....

E:\acedamic\f\j2ee\jdbc>
Output:

SQL> desc std;

Name	Null?	Type
SID		NUMBER(4)
SNAME		VARCHAR2(10)
MARKS		NUMBER(3)

SQL>

Java program to insert rows into a table:

```
import java.sql.*;
class InsertUsingStatement {
                public static void main(String args[])throws Exception{
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");
                        System.out.println(".....Connected to Oracle Database.....");
                        Statement stmt=con.createStatement();
                        String sql1="insert into std values(1001,'sone',60)";
                        String sql2="insert into std values(1002,'stwo',70)";
                        String sql3="insert into std values(1003,'sthr',80)";
                        stmt.executeUpdate(sql1);
                        stmt.executeUpdate(sql2);
                        stmt.executeUpdate(sql3);
```

```
        }
}
```

Output:

```
E:\acedamic\f\j2ee\jdbc>java InsertUsingStatement
.....Connected to Oracle Database.....

SQL> select *from std;

      SID SNAME          MARKS
---------- ---------- ----------
      1001 sone             60
      1002 stwo             70
      1003 sthr             80

SQL>
```

Java program to select rows from table:

```
import java.sql.*;
class SelectingRowsUsingResultSet {
        public static void main(String args[])throws Exception{
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");
                System.out.println(".....Connected to Oracle Database.....");
                String sql="select *from std";
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery(sql);
                while(rs.next()){
                        System.out.println("row.....>"+rs.getRow());
                        System.out.print(rs.getInt(1)+"        ");
                        System.out.print(rs.getString(2)+"        ");
                        System.out.println(rs.getInt(3)+"        ");
                }

        }
```

```
}
```

Output:

```
E:\acedamic\f\j2ee\jdbc>java SelectingRowsUsingResultSet
.....Connected to Oracle Database.....
row.....>1
1001        sone        60
row.....>2
1002        stwo        70
row.....>3
1003        sthr        80

E:\acedamic\f\j2ee\jdbc>
```

Additional Experiment-2

Aim: Write a PL/SQL Code using Procedures, Functions, and Packages FORMS

Procedures:

a. procedure to compute factorial of a number

```
SQL> create or replace procedure factorial(n number) IS
  2  f number:=1;
  3  i number;
  4  e exception;
  5  begin
  6  if n<0 then
  7  raise e;
  8  end if;
  9  for i in 1..n
 10  loop
 11  f:=f*i;
 12  end loop;
 13  dbms_output.put_line('factorial of'||n||'is'||f);
 14  exception
 15  when e then
 16  dbms_output.put_line('factorial for negative numbers donot exist');
 17  end factorial;
 18  /

Procedure created.
SQL> /
Procedure created.

SQL> set serveroutput on;
SQL> execute factorial(4);
factorial of4is24
PL/SQL procedure successfully completed.
SQL> execute factorial(-9);
Factorial for Negative Number donot exists
```

```
PL/SQL procedure successfully completed.
SQL> execute factorial(-6);
Factorial for Negative Number donot exists

PL/SQL procedure successfully completed.
```

b.procedure to increment emp sal

```
SQL> create or replace procedure a(eid number) IS
  2  incr e1.sal%type;
  3  net e1.sal%type;
  4  veno e1.eno%type;
  5  vsal e1.sal%type;
  6  vcomm e1.comm%type;
  7  begin
  8  select eno,sal,NVL(comm,0) into veno,vsal,vcomm from e1
  9  where eno=eid;
 10  net:=vsal+vcomm;
 11  if vsal<=3000 then
 12  incr:=0.20*net;
 13  elsif vsal>=3000 and vsal<=6000 then
 14  incr:=0.40*net;
 15  else
 16  incr:=0.40*net;
 17  end if;
 18  update e1 set sal=sal+incr where eno=eid;
 19  exception
 20  when no_data_found then
 21  dbms_output.put_line('emp doesnot exist');
 22  end a;
 23  /
Procedure created.
SQL> select eno,sal from e1 where eno=7369;
      ENO        SAL
---------- ----------
      7369        960
```

```
SQL> execute a(7369);
PL/SQL procedure successfully completed.

SQL> select eno,sal from e1 where eno=7369;
      ENO          SAL
---------- ----------
     7369         1152
SQL> select eno,sal from emp where eno=7934;
      ENO          SAL
---------- ----------
     7934         1300

SQL> select eno,sal from emp where eno=7521;
      ENO          SAL
---------- ----------
     7521         1250
SQL> execute a(7521);
PL/SQL procedure successfully completed.

SQL> execute a(7934);
PL/SQL procedure successfully completed.
SQL> execute a(5414);
emp not exists
PL/SQL procedure successfully completed.
```

Functions:

a.

```
 1  create or replace function fact(n number,f in out number) return number IS
 2  i number;
 3  begin
 4  if n<0 then
 5  return -1;
 6  end if;
 7  for i in 1..n
```

```
 8   loop
 9  f:=f*i;
10   end loop;
11  return f;
12* end fact;
SQL> /
```

Function created.

```
   1  declare
 2  n number:=&n;
 3   f number:=1;
 4  k number;
 5   begin
 6   k:=fact(n,f);
 7   if k<0 then
 8   dbms_output.put_line('factorial for negative numbers donot exist');
 9   else
10   dbms_output.put_line('factorial is'||k);
11  end if;
12*  end;
13  /
Enter value for n: 4
old   2: n         number:=&n;
new   2: n         number:=4;
Factorial is24
PL/SQL procedure successfully completed.

SQL> /
Enter value for n: 6
old   2: n         number:=&n;
new   2: n         number:=6;
Factorial is720
PL/SQL procedure successfully completed.
```

```
SQL> /
Enter value for n: -6
old   2: n         number:=&n;
new   2: n         number:=-6;
Factorial for Negative Numbers not exist
PL/SQL procedure successfully completed.
```

b.

```
 1  create or replace function a1(eid number) return number IS
 2  incr e1.sal%type;
 3  net e1.sal%type;
 4  veno e1.eno%type;
 5  vsal e1.sal%type;
 6  vcomm e1.comm%type;
 7  begin
 8  select eno,sal,NVL(comm,0) into veno,vsal,vcomm from e1
 9  where eno=eid;
10  net:=vsal+vcomm;
11  if vsal<=3000 then
12  incr:=0.20*net;
13  elsif vsal>=3000 and vsal<=6000 then
14  incr:=0.30*net;
15  else
16  incr:=0.40*net;
17  end if;
18  return incr;
19* end a1;
20  /

Function created.

SQL> ed;
Wrote file afiedt.buf
```

```
 1  declare
 2  eid number:=&eid;
 3  k e1.sal%type;
 4  begin
 5  k:=a1(eid);
 6  update e1 set sal=sal+k where eno=eid;
 7* end;
 8  /
Enter value for eid: 7369
old   2: eid number:=&eid;
new   2: eid number:=7369;

PL/SQL procedure successfully completed.

SQL> select eno,sal from e1 where eno=7369;

      ENO          SAL
---------- ----------
     7369    1990.66
```

Packages:

a.

```
1  create or replace package p1 is
 2  procedure p_empdept1(dn NUMBER);
 3* end p1;
SQL> /

Package created.

SQL> ed;
Wrote file afiedt.buf

 1  create or replace package body p1 is
 2  procedure p_empdept1(dn in number) is dummydept number;
```

```
  3  begin
  4  select distinct dn into dummydept from emp where deptno=dn;
  5  dbms_output.put_line('dept number present');
  6  exception
  7  when no_data_found then
  8  dbms_output.put_line('dept number not present');
  9  end;
 10* end p1;
SQL> /
Package body created.

SQL> execute p1.p_empdept1(20);
dept number present

PL/SQL procedure successfully completed.

SQL> execute p1.p_empdept1(50);
dept number not present

PL/SQL procedure successfully completed.
```

b.

```
SQL> create or replace package p2 is procedure fact3(n number);
  2  end p2;
  3  /

Package created.

SQL> create or replace package body p2 is procedure fact3(n in number) is
  2  f number:=1;
  3  i number;
  4  e exception;
  5  begin
  6  if n<0 then raise e;
  7  end if;
```

```
 8  for i in 1..n
 9  loop
10  f:=f*i;
11  end loop;
12  dbms_output.put_line('factorial of'||n||'is'||f);
13  exception
14  when e then
15  dbms_output.put_line('not possible');
16  end fact3;
17  end p2;
18  /
Package body created.
SQL> set serveroutput on;
SQL> execute p2.fact3(4);
factorial of4is24
PL/SQL procedure successfully completed.

SQL> execute p2.fact3(-4);
not possible
PL/SQL procedure successfully completed.
SQL> execute p2.b(6);
factorial 6is720
PL/SQL procedure successfully completed.
SQL> execute p2.b(-6);
Not possible
PL/SQL procedure successfully completed.
SQL> execute p2.b(7);
factorial 7is5040
  PL/SQL procedure successfully completed.
SQL> execute p2.b(-7);
Not possible
PL/SQL procedure successfully completed.
```