MEERUT INSTITUTE OF ENGINEERING & TECHNOLOGY,MEERUT

N.H. 58, Delhi-Roorkee Highway, Baghpat Road Bypass Crossing,
Meerut-250005, UP(India)

LAB MANUAL

DATABASE MANAGEMENT SYSTEM
RCS-551

Department of Information Technology
Odd Semester

Submitted by :

Ms. Sonika Jalhotra

INDEX

LAB PRE-REQUISITE

The basic pre-requisite for Database Management System is the basic programming skills and knowledge about relational database model.

Developing and managing efficient and effective database applications require understanding the fundamentals of database management systems, techniques for the design of databases, and principles of database administration.

LAB MANUAL OBJECTIVE

This D.B.M.S. manual is primarily written according to the unified syllabus of "DATABASE MANAGEMENT SYSTEMS " of Third year B.tech(CSE) A.K.T.U. syllabus.

This manual clearly explains the concepts of D.B.M.S. and the principles deployed behind databases.

The **Objective** in writing of this lab manual is to produce a general , comprehensive text that treats all the essential core area of D.B.M.S.

The **Goal** is to describe the real world constraint and realities of Database Management Systems.

I have done my best to hunt down and eradicate all errors in this manual.

Lab OUTCOME

After undergoing this laboratory module, the students should be able to:

- Understand and effectively explain the underlying concepts of database technologies
- Design and implement a database schema for a given problem-domain
- Normalize a database
- Querying a database using SQL DML/DDL commands.
- Declare and enforce integrity constraints on a database using a state-of-the-art RDBMS
- Programming PL/SQL including stored procedures, stored functions, cursors, packages

PROGRAM LIST

1. Write SQL queries to implement DDL commands.
2. Write SQL queries to implement DML commands.
3. Write SQL queries to implement TCL commands.
4. Write SQL queries to implement Keys constraints.
5. Write SQL queries to implement Joins.
6. Write SQL queries to implement Views.
7. Write SQL queries to SELECT data using SET UNION ,INTERSECTION and MINUS operations.
8. Write SQL queries using AGGREGATE functions.
9. Write SQL queries to implement Nested queries.
10. Write a PL/SQL code block to find factorial of a number.
11. Write a PL/SQL code block to implement Indexes.
12. Write a PL/SQL code block to implement Procedures.
13. Write a PL/SQL code block to implement Triggers.

VALUE ADDITION

1. Implementation of cursors
2. Program for Student Grade Calculation
3. Database Design using E-R Model and Normalization.
4. Database Design and Implementation of Pay Roll Processing
5. Design and Implementation of Banking System.

…….
NAME & SIGNATURE OF CONCERNED FACULTY

BASIC CONCEPTS OF PROGRAMS

1. Write SQL queries to implement DDL commands.

It is used to communicate with database. DDL is used to:

- o Create an object
- o Alter the structure of an object
- o To drop the object created.

The commands used are:

- Create
- Alter
- Drop
- Truncate
- Rename

1. COMMAND NAME: CREATE

COMMAND DESCRIPTION: CREATE command is used to create objects in the database.

2. COMMAND NAME: DROP

COMMAND DESCRIPTION: DROP command is used to delete the object from the database.

3. COMMAND NAME: TRUNCATE

COMMAND DESCRIPTION: TRUNCATE command is used to remove all the records from the table

4. COMMAND NAME: ALTER

COMMAND DESCRIPTION: ALTER command is used to alter the structure of database

5. COMMAND NAME: RENAME

COMMAND DESCRIPTION: RENAME command is used to rename the objects.

2. **Write SQL queries to implement DML commands**.

DML (DATA MANIPULATION LANGUAGE)

- SELECT
- INSERT
- DELETE
- UPDATE

1. COMMAND NAME: INSERT

COMMAND DESCRIPTION: INSERT command is used to Insert objects in the database.

2. COMMAND NAME: SELECT

COMMAND DESCRIPTION: SELECT command is used to SELECT the object from the database.

3. COMMAND NAME: UPDATE

COMMAND DESCRIPTION: UPDATE command is used to UPDATE the records from the table

4. COMMAND NAME: DELETE

COMMAND DESCRIPTION: DELETE command is used to DELETE the Records form the table

3. Write SQL queries to implement TCL commands.

TCL (TRANSACTION CONTROL LANGUAGE)

- COMMIT
- ROLL BACK
- SAVE POINT

1. COMMAND NAME: COMMIT

COMMAND DESCRIPTION: COMMIT command is used to save the Records.

2. COMMAND NAME: ROLLBACK

COMMAND DESCRIPTION: ROLL BACK command is used to undo the Records.

3. COMMAND NAME: SAVE POINT

COMMAND DESCRIPTION: SAVE POINT command is used to undo the Records in a particular transaction.

4. Write SQL queries to implement Keys constraints.

Constraints are part of the table definition that limits and restriction on the value entered into its columns.

TYPES OF CONSTRAINTS:

1) **Primary key** : A **PRIMARY KEY** constraint is a unique identifier for a row within a database table. Every table should have a primary key constraint to uniquely identify each row and only one primary key constraint can be created for each table. The primary key constraints are used to enforce entity integrity.

2) **Foreign key/references** : A **FOREIGN KEY** constraint prevents any actions that would destroy link between tables with the corresponding data values. A foreign key in one table points to a primary key in another table. Foreign keys prevent actions that would leave rows with foreign key values when there are no primary keys with that value. The foreign key constraints are used to enforce referential integrity.

3) **Check** : A **CHECK** constraint is used to limit the values that can be placed in a column. The check constraints are used to enforce domain integrity.

4) **Unique** : A **UNIQUE** constraint enforces the uniqueness of the values in a set of columns, so no duplicate values are entered. The unique key constraints are used to enforce entity integrity as the primary key constraints.

5) **Not null** : A **NOT NULL** constraint enforces that the column will not accept null values. The not null constraints are used to enforce domain integrity, as the check constraints.

6) **Default** : The **DEFAULT** constraint provides a default value to a column when the **INSERT INTO** statement does not provide a specific value.

CONSTRAINTS CAN BE CREATED IN THREE WAYS:

1) Column level constraints
2) Table level constraints
3) Using DDL statements-alter table command

5.Write SQL queries to implement Joins.

SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables.

SQL COMMANDS

COMMAND NAME: INNER JOIN

COMMAND DESCRIPTION: The INNER JOIN keyword return rows when there is at least one match in both tables.

COMMAND NAME LEFT JOIN

COMMAND DESCRIPTION: The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

COMMAND NAME : RIGHT JOIN

COMMAND DESCRIPTION: The RIGHT JOIN keyword Return all rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

COMMAND NAME : FULL JOIN

COMMAND DESCRIPTION: The FULL JOIN keyword return rows when there is a match in one of the tables.

6.Write SQL queries to implement Views.

Views Helps to encapsulate complex query and make it reusable.

• Provides user security on each view - it depends on your data policy security.
• Using view to convert units - if you have a financial data in US currency, you can create view to convert them into Euro for viewing in Euro currency.

SQL COMMANDS

1. COMMAND NAME: CREATE VIEW
COMMAND DESCRIPTION: CREATE VIEW command is used to define a view.

2. COMMAND NAME: INSERT IN VIEW
COMMAND DESCRIPTION: INSERT command is used to insert a new row into the view.

3. COMMAND NAME: DELETE IN VIEW
COMMAND DESCRIPTION: DELETE command is used to delete a row from the view.

4. COMMAND NAME: UPDATE OF VIEW
COMMAND DESCRIPTION: UPDATE command is used to change a value in a tuple without changing all values in the tuple.

5. COMMAND NAME: DROP OF VIEW
COMMAND DESCRIPTION: DROP command is used to drop the view table

7.Write SQL queries to SELECT data using SET UNION , INTERSECTION and MINUS operations.

UNION

It returns a union of two select statements. It is returning unique (distinct) values of them.

```
SELECT * FROM table1
UNION
SELECT * FROM table2
```

MINUS

MINUS (also known as EXCEPT) returns the difference between the first and second SELECT statement. It is the one where we need to be careful which statement will be put first, cause we will get only those results that are in the first SELECT statement and not in the second.

```
SELECT * FROM table1
MINUS
SELECT * FROM table2;
```

INTERSECT

INTERSECT is opposite from MINUS as it returns us the results that are both to be found in first and second SELECT statement.

```
SELECT * FROM table1
INTERSECT
SELECT * FROM table2;
```

8.Write SQL queries using AGGREGATE functions.

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 or measurement such as a set, a bag or a list.

The functions include:

- **count()** - counts a number of rows
- **sum()** - compute sum
- **avg()** - compute average
- **min()** - compute minimum
- **max()** - compute maximum

9.Write SQL queries to implement Nested queries.

Nested Query can have more than one level of nesting in one single query. A SQL nested query is a SELECT query that is nested inside a SELECT, UPDATE, INSERT, or DELETE SQL query.

SQL COMMANDS

1. COMMAND NAME: SELECT
COMMAND DESCRIPTION: SELECT command is used to select records from the table.

2. COMMAND NAME: WHERE
COMMAND DESCRIPTION: WHERE command is used to identify particular elements.

3. COMMAND NAME: HAVING
COMMAND DESCRIPTION: HAVING command is used to identify particular elements.

4. COMMAND NAME: MIN (SAL)
COMMAND DESCRIPTION: MIN (SAL) command is used to find minimum salary.

10Write a PL/SQL code block to find factorial of a number.

```
declare
n number;
i number;
f number:=1;
begin
n:=&n;
for i in 1..n
loop
f:=f*i;
end loop;
dbms_output.put_line(n||'! = '||f);
end;
```

Output:

```
Enter value for n: 5
old 6: n:=&n;
new 6: n:=5;
5! = 120
PL/SQL procedure successfully completed.
```

11.Write a PL/SQL code block to implement Indexes.

, . . . ,column_nameN)
[COMPUTE STATISTICS];

A collection is an ordered group of elements having the same data type. Each element is identified by a unique subscript that represents its position in the collection.

PL/SQL provides three collection types:

- Index-by tables or Associative array
- Nested table
- Variable-size array or Varray

An **index-by** table (also called an associative array) is a set of **key-value** pairs. Each key is unique and is used to locate the corresponding value. The key can be either an integer or a string.

Syntax for creating an INDEX in Oracle SQL / PLSQL is:

CREATE [UNIQUE] INDEX index_name
ON TABLE table_name (column_name1,column_name2

12.Write a PL/SQL code block to implement Procedures.

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows modifying an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

13.Write a PL/SQL code block to implement Triggers.

Trigger automatically associated with DML statement, when DML statement execute trigger implicitly execute.

We can create trigger using the CREATE TRIGGER statement. If trigger activated, implicitly fire DML statement and if trigger deactivated can't fire.

PL/SQL trigger define using CREATE TRIGGER statement.

```
CREATE [OR REPLACE] TRIGGER trigger_name
        BEFORE | AFTER
        [INSERT, UPDATE, DELETE [COLUMN NAME..]
        ON table_name

        Referencing [ OLD AS OLD | NEW AS NEW ]
        FOR EACH ROW | FOR EACH STATEMENT [ WHEN Condition ]

DECLARE
  [declaration_section
          variable declarations;
          constant declarations;
  ]

BEGIN
        [executable_section
                PL/SQL execute/subprogram body
  ]

EXCEPTION
  [exception_section
                PL/SQL Exception block
  ]

END;
```

VALUE ADDITION

IMPLEMENTATION OF CURSORS

CURSOR PROGRAM FOR ELECTRICITY BILL CALCULATION:

SQL> create table bill(name varchar2(10), address varchar2(20), city varchar2(20), unit number(10));

Table created.

SQL> insert into bill values('&name','&addess','&city','&unit');

Enter value for name: yuva

Enter value for addess: srivi

Enter value for city: srivilliputur

Enter value for unit: 100

old 1: insert into bill values('&name','&addess','&city','&unit')

new 1: insert into bill values('yuva','srivi','srivilliputur','100')

1 row created.

SQL> /

Enter value for name: nithya

Enter value for addess: Lakshmi nagar

Enter value for city: sivakasi

Enter value for unit: 200

old 1: insert into bill values('&name','&addess','&city','&unit')

new 1: insert into bill values('nithya','Lakshmi nagar','sivakasi','200')

1 row created.

Enter value for name: maya

Enter value for addess: housing board

Enter value for city: sivakasi

Enter value for unit: 300

old 1: insert into bill values('&name','&addess','&city','&unit')

new 1: insert into bill values('maya','housing board','sivakasi','300')

1 row created.

SQL> /

Enter value for name: jeeva

Enter value for addess: RRR nagar

Enter value for city: sivaganagai

Enter value for unit: 400

old 1: insert into bill values('&name','&addess','&city','&unit')

new 1: insert into bill values('jeeva','RRR nagar','sivaganagai','400')

1 row created.

SQL> select * from bill;

NAME ADDRESS CITY UNIT

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

yuva	srivi	srivilliputur	100
nithya	Lakshmi nagar	sivakasi	200
maya	housing board	sivakasi	300

```
SQL> declare
 2  cursor c is select * from bill;
 3  b bill %ROWTYPE;
 4  begin
 5  open c;
 6 dbms_output.put_line('Name  Address      city  Unit      Amount');
 7  loop
 8  fetch c into b;
 9  if(c % notfound) then
10 exit;
11 else
12 if(b.unit<=100) then
13 dbms_output.put_line(b.name||' '||b.address||' '||b.city||' '||b.unit||' '||b.uni t*1);

14 elsif(b.unit>100 and b.unit<=200) then
15 dbms_output.put_line(b.name||' '||b.address||' '||b.city||' '||b.unit||' '||b. unit*2);

16 elsif(b.unit>200 and b.unit<=300) then
17 dbms_output.put_line(b.name||' '||b.address||' '||b.city||' '||b.unit||' '||b. unit*3);

18 elsif(b.unit>300 and b.unit<=400) then
 19 dbms_output.put_line(b.name||'   '||b.address||'   '||b.city||'    '||b.unit||'      '||b.unit*
 4);
 20 else
 21 dbms_output.put_line(b.name||'   '||b.address||'   '||b.city||'    '||b.unit||'      '||b.unit*
 5);
 22 end if;
```

```
23  end if;
24  end loop;
25  close c;
26  end;
27  /
```

Name Address city Unit Amount

yuva srivi srivilliputur 100 100

nithya Lakshmi nagar sivakasi 200 400

maya housing board sivakasi 300 900

jeeva RRR nagar sivaganagai 400 1600

PL/SQL procedure successfully completed.

PROGRAM FOR STUDENT GRADE CALCULATION

SQL> create table std(name varchar(10), rollno number(3),mark1 number(3), mark2 number(3), mark3 nu

mber(3));

Table created.

SQL> insert into std values('&name','&rollno','&mark1','&mark2','&mark3');

Enter value for name: gowri

Enter value for rollno: 101

Enter value for mark1: 78

Enter value for mark2: 89

Enter value for mark3: 99

old 1: insert into std values('&name','&rollno','&mark1','&mark2','&mark3')

new 1: insert into std values('gowri','101','78','89','99')

1 row created.

SQL> /

Enter value for name: prem

Enter value for rollno: 102

Enter value for mark1: 88

Enter value for mark2: 99

Enter value for mark3: 90

old 1: insert into std values('&name','&rollno','&mark1','&mark2','&mark3')

new 1: insert into std values('prem','102','88','99','9’)

1 row created.

SQL> /

Enter value for name: ravathi

Enter value for rollno: 103

Enter value for mark1: 67

Enter value for mark2: 89

Enter value for mark3: 99

old 1: insert into std values('&name','&rollno','&mark1','&mark2','&mark3')

new 1: insert into std values('ravathi','103','67','89','99')

1 row created.

SQL> /

Enter value for name: arun

Enter value for rollno: 104

Enter value for mark1: 56

Enter value for mark2: 66

Enter value for mark3: 77

old 1: insert into std values('&name','&rollno','&mark1','&mark2','&mark3')

new 1: insert into std values('arun','104','56','66','77')

1 row created.

SQL> set serveroutput on;

```
SQL> declare
  2  tot number;
  3  average number;
  4  cursor c is select * from std;
  5  s std %ROWTYPE;
  6  begin
  7  open c;
  8  dbms_output.put_line('Name Rollno Mark1 Mark2 Mark3 Total Average Grade');

  9  loop
 10  fetch c into s;
 11  tot:=s.mark1+s.mark2+s.mark3;
 12  average:=floor(tot/3);
 13  if(c % notfound)then
 14  exit;
 15  else
 16  if(s.mark1<50 or s.mark2<50 or s.mark3<50)then
  17 dbms_output.put_line(s.name||'   '||s.rollno||'   '||s.mark1||'   '||s.mark2||'    '||s.mark3||
'   '||tot||'   '||average||'   '||'F');
  18 elsif(average>=90 and average<=100)then
  19 dbms_output.put_line(s.name||'   '||s.rollno||'   '||s.mark1||'   '||s.mark2||'    '||s.mark3||
'   '||tot||'   '||average||'   '||'S');
  20 elsif(average>=80 and average<90)then
  21 dbms_output.put_line(s.name||'   '||s.rollno||'   '||s.mark1||'   '||s.mark2||'    '||s.mark3||
'   '||tot||'   '||average||'   '||'A+');
 22  elsif(average>=70 and average<80)then
 23  dbms_output.put_line(s.name||'  '||s.rollno||'  '||s.mark1||'  '||s.mark2||'  '||s.mark3||
```

```
'  '||tot||'   '||average||'   '||'B');
  24 elsif(average>=60 and average<70)then
  25 dbms_output.put_line(s.name||'   '||s.rollno||'   '||s.mark1||'   '||s.mark2||'    '||s.mark3||
'  '||tot||'   '||average||'   '||'C');
  26 else
  27 dbms_output.put_line(s.name||'   '||s.rollno||'   '||s.mark1||'   '||s.mark2||'    '||s.mark3||
'   '||tot||' '||average||' '||'D');
 28 end if;
 29 end if;
 30 end loop;
 31 close c;
 32 end;
 33 /
```

Name	Rollno	Mark1	Mark2	Mark3	Total	Average	Grade
gowri	101	78	89	99	266	88	A+
prem	102	88	99	90	277	92	S
ravathi	103	67	89	99	255	85	A+
arun	104	56	66	77	199	66	C

PL/SQL procedure successfully completed.

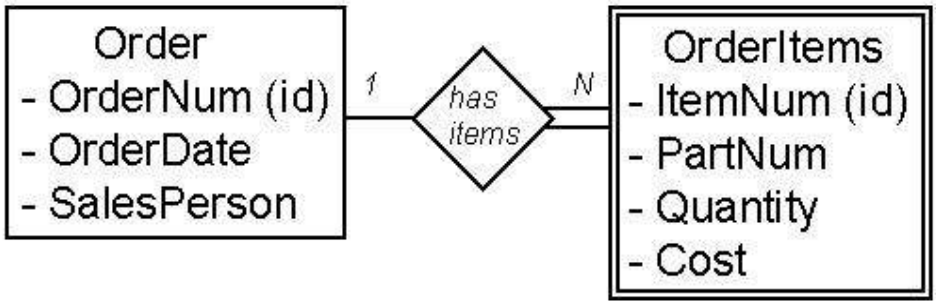
RESULT:

Thus the program to implement cursors was executed and output was verified successfully.

Database design using E-R model and Normalization

ER diagram:

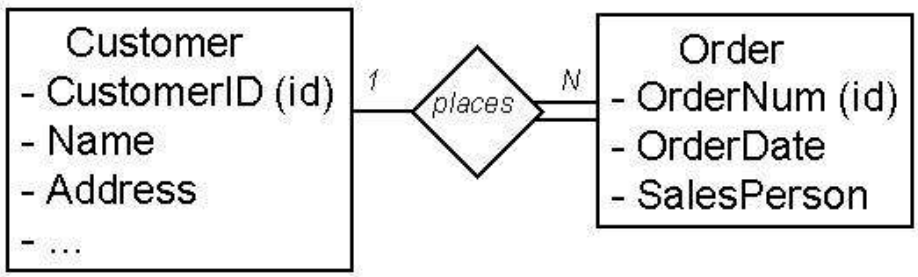
Chen Notation



- **ORDER** (OrderNum (key), OrderDate, SalesPerson)
  ORDERITEMS (OrderNum (key)(fk) , ItemNum (key), PartNum, Quantity, Cost)

- In the above example, in the ORDERITEMS Relation: OrderNum is the *Foreign Key* and OrderNum plus ItemNum is the *Composite Key*.

Chen Notation



In the ORDER Relation: OrderNum is the *Key*.

Representing Relationships

- **1:1** Relationships. The key of one relation is stored in the second relation. Look at example queries to determine which key is queried most often.

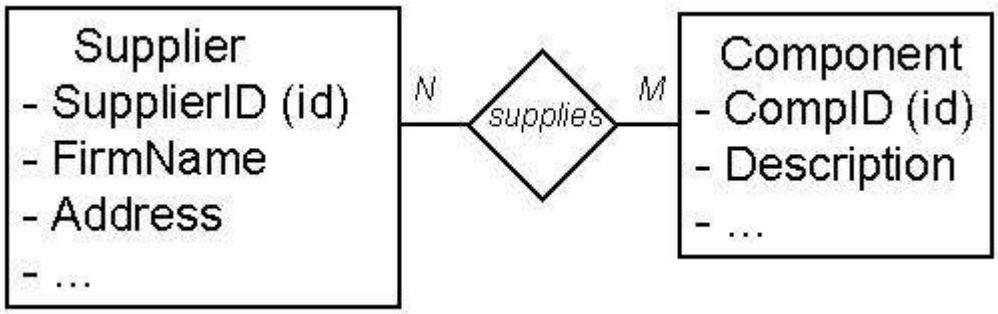
- **1:N** Relationships.
  Parent - Relation on the "1" side.
  Child - Relation on the "Many" side.

- Represent each Entity as a relation.
  Copy the key of the parent into the child relation.

- **CUSTOMER (<u>CustomerID</u> (key), Name, Address, ...)**
  ORDER (<u>OrderNum</u> (key), OrderDate, SalesPerson, CustomerID (fk))

- **M:N** Relationships. Many to Many relationships can not be directly implemented in relations.
- Solution: Introduce a third *Intersection relation* and copy keys from original two relations.

Chen Notation



- SUPPLIER (SupplierID (key), FirmName, Address, ...)
  COMPONENT (CompID (key), Description, ...)
  SUPPLIER_COMPONENT (SupplierID (key), CompID (key))
- Note that this can also be shown in the ER diagram. Also, look for potential added attributes in the intersection relation.

RESULT:

Thus the ER Database design using E-R model and Normalization was implemented successfully.

DATABASE DESIGN AND IMPLEMENTATION PAY ROLL PROCESSING

STEPS:

1. Create a database for payroll processing which request the using SQL
2. Establish ODBC connection
3. In the administrator tools open data source ODBC
4. Click add button and select oracle in ORA home 90, click finish
5. A window will appear given the data source home as oracle and select TNS source name as lion and give the used id as SWTT
6. ADODC CONTROL FOR SALARY FORM:-
7. The above procedure must be follow except the table , A select the table as salary
8. Write appropriate Program in form each from created in VB from each from created in VB form project.

SQL>create table emp(eno number primary key,enamr varchar(20),age number,addr

varchar(20),DOB date,phno number(10));

Table created.

SQL>create table salary(eno number,edesig varchar(10),basic number,da number,hra number,pf number,mc number,met number,foreign key(eno) references emp); Table created.

TRIGGER to calculate DA,HRA,PF,MC

SQL> create or replace trigger employ

2 after insert on salary

3 declare

4 cursor cur is select eno,basic from salary;

5 begin

6 for cur1 in cur loop

7 update salary set

8 hra=basic*0.1,da=basic*0.07,pf=basic*0.05,mc=basic*0.03 where hra=0; 9 end loop;

10 end;

11 / Trigger created.

PROGRAM FOR FORM 1

```
Private Sub emp_Click() Form
2.Show End
Sub Private
Sub exit_Click()
Unload Me
End Sub Private
Sub salary_Click()
Form3.Show
End Sub
```

PROGRAM FOR FORM 2

```
Private Sub add_Click()
Adodc1.Recordset.AddNew MsgBox "Record added"
End Sub Private
Sub clear_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
End Sub Private Sub delte _Click()
Adodc1.Recordset.Delete MsgBox "Record
Deleted" If Adodc1.Recordset.EOF = True Then
Adodc1.Recordset.MovePrevious
End If
End
```

```
Sub Private Sub exit_Click()
```

```
Unload Me
End Sub
Private Sub main_Click()
Form1.Show
End Sub
Private Sub modify_Click()
Adodc1.Recordset.Update
End Sub
```

PROGRAM FOR FORM 3

```
Private Sub add_Click()
Adodc1.Recordset.AddNew MsgBox "Record added"
End Sub
Private Sub
clear_Click()
Text1.Text = ""
```

```
Text2.Text = ""

Text3.Text = ""

Text4.Text = ""

Text5.Text = ""
```

DESIGN AND IMPLEMENTATION OF BANKING SYSTEM

DETAILS OF THE STEP

1.Create the DB for banking system source request the using SQL

2.Establishing ODBC connection

3.Click add button and select oracle in ORA home 90 click finished

4.A window will appear give the data source name as oracle and give the user id as scott

5.Now click the test connection a window will appear with server and user name give user as scott and password tiger Click ok

6.VISUAL BASIC APPLICATION:-

Create standard exe project in to and design ms from in request format

To add ADODC project select component and check ms ADO data control click ok
Now the control is added in the tool book

Create standard exe project in to and design ms from in request format

7.ADODC CONTEOL FOR ACCOUNT FROM:- Click customs and property window and window will appear and select ODBC data source name as oracle and click apply as the some window.

CREATE A TABLE IN ORACLE

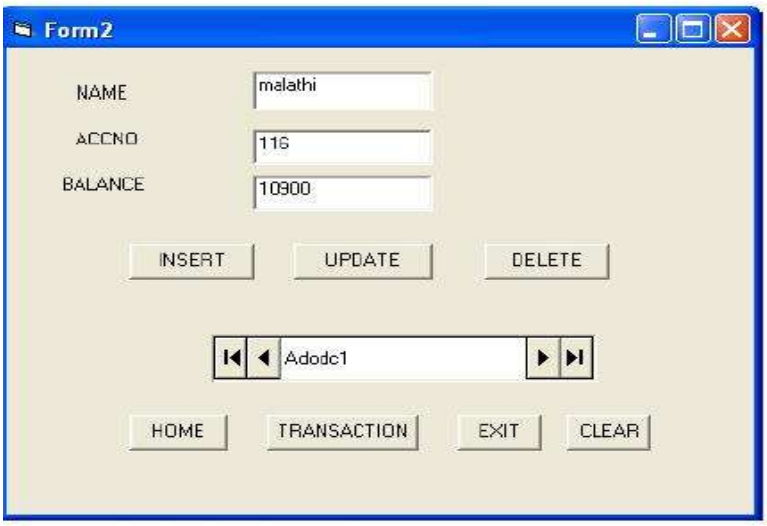
SQL>create table account(cname varchar(20),accno number(10),balance
number); Table Created
SQL> insert into account values('&cname',&accno,&balance);
Enter value for cname: Mathi
Enter value for accno: 1234
Enter value for balance: 10000
old 1: insert into account values('&cname',&accno,&balance)
new 1: insert into emp values('Mathi',1234,10000) 1 row created.

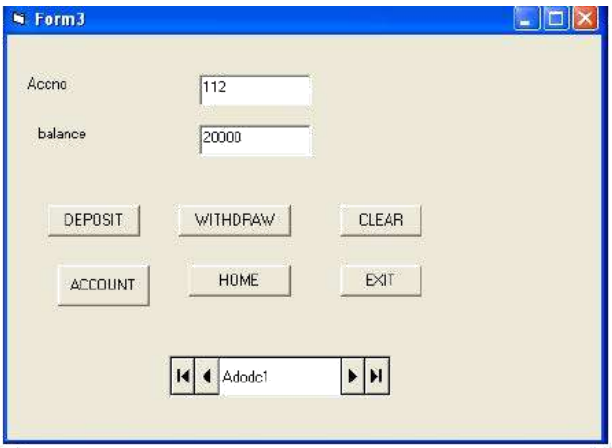
SOURCE CODE FOR FORM1

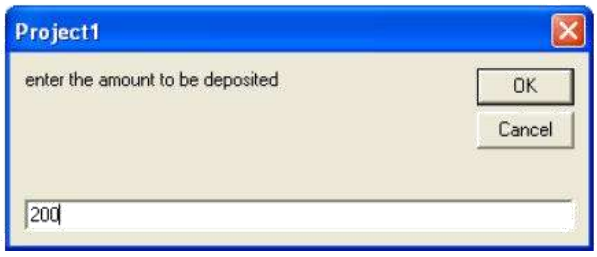
```
Private Sub ACCOUNT_Click()
Form2.Show
End Sub
Private Sub
EXIT_Click()
Unload Me
End Sub
Private Sub
TRANSACTION_Click()
Form3.Show
```

```
End Sub
SOURCE CODE FOR FORM 2
Private Sub CLEAR_Click()
Text1.Text = ""
 Text2.Text = ""
Text3.Text = ""
End Sub
Private Sub
DELETE_Click()
Adodc1.Recordset.DELETE MsgBox "record deleted"
Adodc1.Recordset.MoveNext If Adodc1.Recordset.EOF = True Then
Adodc1.Recordset.MovePrevious
End If
End Sub
Private Sub EXIT_Click()
Unload Me
End Sub
Private Sub
HOME_Click()
Form1.Show
End Sub
Private Sub
INSERT_Click() Adodc1.Recordset.AddNew
End Sub
Private Sub
TRANSACTION_Click()
Form3.Show
End Sub
Private Sub UPDATE_Click() Adodc1.Recordset.UPDATE MsgBox "record updated
successfully"
End Sub
SOURCE CODE FOR FORM 3
Private Sub ACCOUNT_Click()
Form2.Show
End Sub
Private Sub CLEAR_Click()
Text1.Text = ""
Text2.Text = ""
End Sub
```

```
Private Sub
DEPOSIT_Click()
Dim s As String s = InputBox("enter the amount to be deposited")
Text2.Text = Val(Text2.Text) + Val(s) A = Text2.Text MsgBox "CURRENT BALANCE IS
Rs" + Str(A) Adodc1.Recordset.Save Adodc1.Recordset.UPDATE
End Sub
Private Sub
EXIT_Click()
Unload Me
End Sub
Private Sub
HOME_Click()
Form1.Show End
Sub Private Sub
WITHDRAW_Click()
Dim s As String s = InputBox("enter the amount to be deleted")
Text2.Text = Val(Text2.Text) - Val(s) A = Text2.Text MsgBox "current balance is Rs" +
Str(A)
Adodc1.Recordset.Save
Adodc1.Recordset.UPDATE
End Sub
```







Result:

Thus the banking system was designed and implemented successfully.

APPENDIX

1. Write SQL queries to implement DDL commands.

DDL commands used are:

- o Create
- o Alter
- o Drop
- o Truncate

QUERY: 01 Write a query to create a table employee with empno, ename, designation, and salary.

Syntax for creating a table:

SQL: CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), COLUMN NAME.1 <DATATYPE> (SIZE) ……);

```
SQL>CREATE TABLE EMP (EMPNO NUMBER (4),
ENAME VARCHAR2 (10),
DESIGNATIN VARCHAR2 (10),
SALARY NUMBER (8,2));
Table created.
```

QUERY: 02 Write a query to display the column name and datatype of the table employee.

Syntax for describe the table:

SQL: DESC <TABLE NAME>;

```
SQL> DESC EMP;

Name          Null?          Type
----------------------------------------- -------- ------------
EMPNO                        NUMBER(4)
ENAME                        VARCHAR2(10)
DESIGNATIN                   VARCHAR2(10)
SALARY                        NUMBER(8,2)
```

ALTER & MODIFICATION ON TABLE

QUERY: 03 Write a Query to Alter the column EMPNO NUMBER (4) TO EMPNO NUMBER(6).

Syntax for Alter & Modify on a Single Column:

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME> <DATATYPE> (SIZE);

SQL>ALTER TABLE EMP MODIFY EMPNO NUMBER (6);
Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(6)
ENAME		VARCHAR2(10)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)

QUERY: 04 Write a Query to Alter the table employee with multiple columns (EMPNO, ENAME.)

Syntax for alter table with multiple column:

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME1> <DATATYPE> (SIZE), MODIFY <COLUMN NAME2> <DATATYPE> (SIZE) …….;

SQL>ALTER TABLE EMP MODIFY (EMPNO NUMBER (7), ENAME VARCHAR2(12));
Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2);

QUERY: 05 Write a query to add a new column in to employee

Syntax for add a new column:

SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME> <DATA TYPE> <SIZE>);

SQL> ALTER TABLE EMP ADD QUALIFICATION VARCHAR2(6);
Table altered.
SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR2(6)

QUERY: 06 Write a query to add multiple columns in to employee

Syntax for add a new column:

SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME1> <DATA TYPE> <SIZE>,(<COLUMN NAME2> <DATA TYPE> <SIZE>, ……);

SQL>ALTER TABLE EMP ADD (DOB DATE, DOJ DATE);
Table altered

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR2(6)
DOB		DATE
DOJ		DATE

REMOVE / DROP

QUERY: 07 Write a query to drop a column from an existing table employee

Syntax for add a new column:

SQL> ALTER TABLE <TABLE NAME> DROP COLUMN <COLUMN NAME>;

SQL> ALTER TABLE EMP DROP COLUMN DOJ;
Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR2(6)
DOB		DATE

QUERY: 08 Write a query to drop multiple columns from employee

Syntax for add a new column:

SQL> ALTER TABLE <TABLE NAME> DROP <COLUMN NAME1>,<COLUMN NAME2>,…….. ;

SQL> ALTER TABLE EMP DROP (DOB, QUALIFICATION);
Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)

REMOVE

QUERY: 09 Write a query to rename table emp to employee

Syntax for add a new column:

SQL> ALTER TABLE RENAME <OLD NAM E> TO <NEW NAME>

SQL> ALTER TABLE EMP RENAME EMP TO EMPLOYEE;

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)

2. **Write SQL queries to implement DML commands**.

DML (DATA MANIPULATION LANGUAGE)

- SELECT
- INSERT

- DELETE
- UPDATE

INSERT

QUERY: 01 Write a query to insert the records in to employee.

Syntax for Insert Records in to a table:

SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',…..);

INSERT A RECORD FROM AN EXISTING TABLE:

SQL>INSERT INTO EMP VALUES(101,'NAGARAJAN','LECTURER',15000);
1 row created.

SELECT

QUERY: 02 Write a query to display the records from employee.

Syntax for select Records from the table:

SQL> SELECT * FROM <TABLE NAME>;

DISPLAY THE EMP TABLE:

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATION	SALARY
101	NAGARAJAN	LECTURER	15000

INSERT A RECORD USING SUBSITUTION METHOD

QUERY: 03 . Write a query to insert the records in to employee using substitution method.

Syntax for Insert Records into the table:

SQL :> INSERT INTO <TABLE NAME> VALUES< '&column name', '&column name 2',…..);

```
SQL> INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY');

Enter value for empno: 102
Enter value for ename: SARAVANAN
Enter value for designatin: LECTURER
Enter value for salary: 15000
old 1: INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')
new 1: INSERT INTO EMP VALUES(102,'SARAVANAN','LECTURER','15000')
1 row created.

SQL> /
Enter value for empno: 103
Enter value for ename: PANNERSELVAM
Enter value for designatin: ASST. PROF
Enter value for salary: 20000
old 1: INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')
new 1: INSERT INTO EMP VALUES(103,'PANNERSELVAM','ASST. PROF','20000')
1 row created.

SQL> /

Enter value for empno: 104
Enter value for ename: CHINNI
Enter value for designatin: HOD, PROF
Enter value for salary: 45000
old 1: INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')
new 1: INSERT INTO EMP VALUES(104,'CHINNI','HOD, PROF','45000')
1 row created.

SQL> SELECT * FROM EMP;
 EMPNO      ENAME       DESIGNATION          SALARY
----------  ----------  ----------           ----------
 101        NAGARAJAN   LECTURER             15000
 102        SARAVANAN   LECTURER             15000
 103        PANNERSELVAM ASST. PROF          20000
```

104	CHINNI	HOD, PROF	45000

UPDATE

QUERY: 04 Write a query to update the records from employee.

Syntax for update Records from the table:

SQL> UPDATE <<TABLE NAME> SET <COLUMNANE>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

SQL> UPDATE EMP SET SALARY=16000 WHERE EMPNO=101;
1 row updated.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATION	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	LECTURER	15000
103	PANNERSELVAM	ASST. PROF	20000
104	CHINNI	HOD, PROF	45000

UPDATE MULTIPLE COLUMNS

QUERY: 05 Write a query to update multiple records from employee.

Syntax for update multiple Records from the table:

SQL> UPDATE <<TABLE NAME> SET <COLUMNANE>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

SQL>UPDATE EMP SET SALARY = 16000, DESIGNATIN='ASST. PROF' WHERE EMPNO=102;
1 row updated.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATOIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	16000

103	PANNERSELVAM	ASST. PROF	20000
104	CHINNI	HOD, PROF	45000

DELETE

QUERY: 06 . Write a query to delete records from employee.

Syntax for delete Records from the table:

SQL> DELETE <TABLE NAME> WHERE <COLUMN NAME>=<VALUE>;

SQL> DELETE EMP WHERE EMPNO=103;
1 row deleted.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATOIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	16000
104	CHINNI	HOD, PROF	45000

3. Write SQL queries to implement TCL commands.

TCL (TRANSACTION CONTROL LANGUAGE)

- COMMIT
- ROLL BACK
- SAVE POINT

SAVEPOINT:

QUERY: 01 Write a query to implement the save point.

Syntax for save point:

SQL> SAVEPOINT <SAVE POINT NAME>;

SQL> SAVEPOINT S1;
Savepoint created.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATION	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	16000
104	CHINNI	HOD, PROF	45000

SQL> INSERT INTO EMP VALUES(105,'PARTHASAR','STUDENT',100);
1 row created.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATION	SALARY
105	PARTHASAR	STUDENT	100
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	16000
104	CHINNI	HOD, PROF	45000

ROLL BACK

QUERY: 02 Write a query to implement the Rollback.

Syntax for save point:

SQL> ROLL BACK <SAVE POINT NAME>;

SQL> ROLL BACK S1;
Rollback complete.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	16000
103	PANNERSELVAM	ASST. PROF	20000
104	CHINNI	HOD, PROF	45000

COMMIT

QUERY: 03 Write a query to implement the Rollback.

Syntax for commit:

SQL> COMMIT;

SQL> COMMIT;

Commit complete.

4. Write SQL queries to implement Keys constraints.

Constraints are part of the table definition that limits and restriction on the value entered into its columns.

TYPES OF CONSTRAINTS:

1) Primary key

2) Foreign key/references

3) Check

4) Unique

5) Not null

6) Null

7) Default

CONSTRAINTS CAN BE CREATED IN THREE WAYS:

1) Column level constraints

2) Table level constraints

3) Using DDL statements-alter table command

Column level constraints Using Primary key

QUERY 1. Write a query to create primary constraints with column level

Syntax for Column level constraints Using Primary key:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE)<TYPE OF CONSTRAINTS> , COLUMN NAME.1 <DATATYPE> (SIZE) ……);

```
SQL>CREATE TABLE EMPLOYEE(EMPNO NUMBER(4) PRIMARY  KEY,
ENAME VARCHAR2(10),
JOB VARCHAR2(6),
SAL NUMBER(5),
DEPTNO NUMBER(7));
```

Column level constraints Using Primary key with naming convention

QUERY 2. Write a query to create primary constraints with column level with naming convention

Syntax for Column level constraints Using Primary key:

SQL: >CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE)CONSTRAINTS <NAME OF THE CONSTRAINTS> <TYPE OF THE

CONSTRAINTS> , COLUMN NAME.1 <DATATYPE> (SIZE) ……);

SQL>CREATE TABLE EMPLOYEE(EMPNO NUMBER(4) CONSTRAINT EMP_EMPNO_PK PRIMARY KEY,
ENAME VARCHAR2(10),
JOB VARCHAR2(6),
SAL NUMBER(5),
DEPTNO NUMBER(7));

Table Level Primary Key Constraints

QUERY 3. Write a query to create primary constraints with table level with naming convention

Syntax for Table level constraints Using Primary key:

SQL: >CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE) , COLUMN NAME.1 <DATATYPE> (SIZE), CONSTRAINTS <NAME OF THE CONSTRAINTS> <TYPE OF THE CONSTRAINTS>);

SQL>CREATE TABLE EMPLOYEE (EMPNO NUMBER(6),
ENAME VARCHAR2(20),
JOB VARCHAR2(6),
SAL NUMBER(7),
DEPTNO NUMBER(5),
CONSTRAINT EMP_EMPNO_PK PRIMARY
KEY(EMPNO));

Table level constraint with alter command (primary key):

QUERY 4. Write a query to create primary constraints with alter command

Syntax for Column level constraints Using Primary key:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), COLUMN NAME.1 <DATATYPE> (SIZE));

SQL> ALTER TABLE <TABLE NAME> ADD CONSTRAINTS <NAME OF THE CONSTRAINTS> <TYPE OF THE CONSTRAINTS> <COLUMN NAME>);

SQL>CREATE TABLE EMPLOYEE(EMPNO NUMBER(5),
ENAME VARCHAR2(6),
JOB VARCHAR2(6),

SAL NUMBER(6),
DEPTNO NUMBER(6));

SQL>ALTER TABLE EMP3 ADD CONSTRAINT EMP3_EMPNO_PK PRIMARY KEY (EMPNO);

Reference /foreign key constraint

Column level foreign key constraint:

QUERY 5. Write a query to create foreign key constraints with column level
Parent Table:

Syntax for Column level constraints Using Primary key:
SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE)<TYPE OF CONSTRAINTS> , COLUMN NAME.1 <DATATYPE> (SIZE) ……);
Child Table:
Syntax for Column level constraints Using foreign key:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), COLUMN NAME2 <DATATYPE> (SIZE) REFERENCES <TABLE NAME> (COLUMN NAME> ……);

SQL>CREATE TABLE DEPT(DEPTNO NUMBER(2) PRIMARY KEY,
DNAME VARCHAR2(20),
LOCATION VARCHAR2(15));

SQL>CREATE TABLE EMP4
(EMPNO NUMBER(3),
DEPTNO NUMBER(2) REFERENCES DEPT(DEPTNO),
DESIGN VARCHAR2(10));

Table Level Foreign Key Constraints

QUERY 6. Write a query to create foreign key constraints with Table level
Parent Table:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE)<TYPE OF CONSTRAINTS> , COLUMN NAME.1 <DATATYPE> (SIZE) ……);

Child Table:

Syntax for Table level constraints using foreign key:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), COLUMN NAME2 <DATATYPE> (SIZE), CONSTRAINT <CONST. NAME> REFERENCES <TABLE NAME> (COLUMN NAME>);

SQL>CREATE TABLE DEPT
(DEPTNO NUMBER(2) PRIMARY KEY,
DNAME VARCHAR2(20),
LOCATION VARCHAR2(15));
SQL>CREATE TABLE EMP5
(EMPNO NUMBER(3),
DEPTNO NUMBER(2),
DESIGN VARCHAR2(10)CONSTRAINT ENP2_DEPTNO_FK FOREIGN
KEY(DEPT NO)REFERENCESDEPT(DEPTNO));

Table Level Foreign Key Constraints with Alter command

QUERY 7. Write a query to create foreign key constraints with Table level with alter command.

Parent Table:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE)<TYPE OF CONSTRAINTS> , COLUMN NAME.1 <DATATYPE> (SIZE) ……);

Child Table:

Syntax for Table level constraints using foreign key:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE) , COLUMN NAME2 <DATATYPE> (SIZE));

SQL> ALTER TABLE <TABLE NAME> ADD CONSTRAINT <CONST. NAME> REFERENCES <TABLE NAME> (COLUMN NAME>);

SQL>CREATE TABLE DEPT
(DEPTNO NUMBER(2) PRIMARY KEY,
DNAME VARCHAR2(20),
LOCATION VARCHAR2(15));
SQL>CREATE TABLE EMP5
(EMPNO NUMBER(3),

DEPTNO NUMBER(2),
DESIGN VARCHAR2(10));

SQL>ALTER TABLE EMP6 ADD CONSTRAINT EMP6_DEPTNO_FK FOREIGN KEY(DEPTNO)REFERENCES DEPT(DEPTNO);

Check constraint

Column Level Check Constraint

QUERY 8. Write a query to create Check constraints with column level

Syntax for column level constraints using Check:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE) CONSTRAINT <CONSTRAINTS NAME> <TYPE OF CONSTRAINTS> (CONSTRAITNS CRITERIA) , COLUMN NAME2 <DATATYPE> (SIZE));

SQL>CREATE TABLE EMP7(EMPNO NUMBER(3),
ENAME VARCHAR2(20),
DESIGN VARCHAR2(15),
SAL NUMBER(5)CONSTRAINT EMP7_SAL_CK CHECK(SAL>500 AND SAL<10001),
DEPTNO NUMBER(2));

Table Level Check Constraint:

Query 9. Write a query to create Check constraints with table level

Syntax for Table level constraints using Check:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), (COLUMN NAME2 <DATATYPE> (SIZE), CONSTRAINT <CONSTRAINTS NAME> <TYPE OF CONSTRAINTS> (CONSTRAITNS CRITERIA)) ;

SQL>CREATE TABLE EMP8(EMPNO NUMBER(3),
ENAME VARCHAR2(20),
DESIGN VARCHAR2(15),
SAL NUMBER(5),DEPTNO NUMBER(2),
CONSTRAINTS EMP8_SAL_CK CHECK(SAL>500 AND SAL<10001));

Check Constraint with Alter Command

Query 10. Write a query to create Check constraints with table level using alter command.

Syntax for Table level constraints using Check:

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), (COLUMN NAME2 <DATATYPE> (SIZE), CONSTRAINT <CONSTRAINTS NAME> <TYPE OF CONSTRAINTS> (CONSTRAITNS CRITERIA)) ;

SQL>CREATE TABLE EMP9(EMPNO NUMBER,
ENAME VARCHAR2(20),
DESIGN VARCHAR2(15),
SAL NUMBER(5));

SQL>ALTER TABLE EMP9 ADD CONSTRAINTS EMP9_SAL_CK CHECK(SAL>500 AND SAL<10001);

Unique Constraint

Column Level Constraint

Query 11. Write a query to create unique constraints with column level

Syntax for Column level constraints with Unique:

SQL :> CREATE <OBJ.TYPE> <OBJ.NAME> (<COLUMN NAME.1> <DATATYPE> (SIZE) CONSTRAINT <NAME OF CONSTRAINTS> <CONSTRAINT TYPE>, (COLUMN NAME2 <DATATYPE> (SIZE)) ;

SQL>CREATE TABLE EMP10(EMPNO NUMBER(3),
ENAME VARCHAR2(20),
DESGIN VARCHAR2(15)CONSTRAINT EMP10_DESIGN_UK UNIQUE,
SAL NUMBER(5));

Table Level Constraint Alter Command

Query 12. Write a query to create unique constraints with table level
Syntax for Table level constraints with Check Using Alter
SQL :> CREATE <OBJ.TYPE> <OBJ.NAME> (<COLUMN NAME.1> <DATATYPE> (SIZE), (COLUMN NAME2 <DATATYPE> (SIZE)) ;

SQL> ALTER TABLE ADD <CONSTRAINTS> <CONSTRAINTS NAME> <CONSTRAINTS TYPE>(COLUMN NAME);

```
SQL>CREATE TABLE EMP12
(EMPNO NUMBER(3),
ENAME VARCHAR2(20),
DESIGN VARCHAR2(15),
SAL NUMBER(5));

SQL>ALTER TABLE EMP12 ADD CONSTRAINT EMP12_DESIGN_UK
UNIQUE(DESING);
```

Not Null

Column Level Constraint

Query 13. Write a query to create Not Null constraints with column level

Syntax for Column level constraints with Not Null:

SQL :> CREATE <OBJ.TYPE> <OBJ.NAME> (<COLUMN NAME.1> <DATATYPE> (SIZE) CONSTRAINT <NAME OF CONSTRAINTS> <CONSTRAINT TYPE>, (COLUMN NAME2 <DATATYPE> (SIZE)) ;

```
SQL>CREATE TABLE EMP13
(EMPNO NUMBER(4),
ENAME VARCHAR2(20) CONSTRAINT EMP13_ENAME_NN NOT NULL,
DESIGN VARCHAR2(20),
SAL NUMBER(3));
```

Null

Column Level Constraint

QUERY 14. Write a query to create Null constraints with column level
Syntax for Column level constraints with Null:

SQL :> CREATE <OBJ.TYPE> <OBJ.NAME> (<COLUMN NAME.1> <DATATYPE> (SIZE) CONSTRAINT <NAME OF CONSTRAINTS> <CONSTRAINT TYPE>, (COLUMN NAME2 <DATATYPE> (SIZE)) ;

```
SQL>CREATE TABLE EMP13
(EMPNO NUMBER(4),
ENAME VARCHAR2(20) CONSTRAINT EMP13_ENAME_NN NULL,
DESIGN VARCHAR2(20),
SAL NUMBER(3));
```

5. Write SQL queries to implement Joins.

SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables.

SQL commands are :

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

LEFT JOIN or LEFT OUTTER JOIN

Table:1 - ORDERS

```
SQL> CREATE table orders(O_Id number(5),
Orderno number(5),
P_Id number(3));
Table created.

SQL> DESC orders;

 Name          Null?          Type
 ---------------------- -------- -----------
 O_ID                          NUMBER(5)
 ORDERNO                       NUMBER(5)
 P_ID                          NUMBER(3)
```

INSERTING VALUES INTO ORDERS

```
SQL> INSERT into orders values(&O_Id,&Orderno,&P_Id);
Enter value for o_id: 1
Enter value for orderno: 77895
Enter value for p_id: 3
old 1: INSERT into orders values(&O_Id,&Orderno,&P_Id)
new 1: INSERT into orders values(1,77895,3)
1 row created.

SQL> INSERT into orders values(&O_Id,&Orderno,&P_Id);
Enter value for o_id: 2
```

```
Enter value for orderno: 44678
Enter value for p_id: 3
old 1: INSERT into orders values(&O_Id,&Orderno,&P_Id)
new 1: INSERT into orders values(2,44678,3)
1 row created.

SQL> INSERT into orders values(&O_Id,&Orderno,&P_Id);
Enter value for o_id: 3
Enter value for orderno: 22456
Enter value for p_id: 1
old 1: INSERT into orders values(&O_Id,&Orderno,&P_Id)
new 1: INSERT into orders values(3,22456,1)
1 row created.

SQL> INSERT into orders values(&O_Id,&Orderno,&P_Id);
Enter value for o_id: 4
Enter value for orderno: 24562
Enter value for p_id: 1
old 1: INSERT into orders values(&O_Id,&Orderno,&P_Id)
new 1: INSERT into orders values(4,24562,1)
1 row created.

SQL> INSERT into orders values(&O_Id,&Orderno,&P_Id);
Enter value for o_id: 5
Enter value for orderno: 34764
Enter value for p_id: 15
old 1: INSERT into orders values(&O_Id,&Orderno,&P_Id)
new 1: INSERT into orders values(5,34764,15)
1 row created.
```

TABLE SECTION:

```
SQL> SELECT * FROM orders;

 O_ID        ORDERNO    P_ID
---------- ---------- ----------
 1          77895      3
 2          44678      3
 3          22456      1
 4           24562             1
 5           34764             15
```

TABLE -2: PERSONS

```
SQL> CREATE table persons(p_Id number(5),
LASTNAME varchar2(10),
Firstname varchar2(15), Address varchar2(20),
city varchar2(10));
Table created.

SQL> INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city');
Enter value for p_id: 1
Enter value for lastname: Hansen
Enter value for firstname: Ola
Enter value for address: Timoteivn 10
Enter value for city: sadnes
old 1: INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city')
new 1: INSERT into persons values(1,'Hansen','Ola','Timoteivn 10','sadnes')
1 row created.

SQL> INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city');
Enter value for p_id: 2
Enter value for lastname: Svendson
Enter value for firstname: Tove
Enter value for address: Borgn 23
Enter value for city: Sandnes
old 1: INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city')
new 1: INSERT into persons values(2,'Svendson','Tove','Borgn 23','Sandnes')
1 row created.

SQL> INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city');
Enter value for p_id: 3
Enter value for lastname: Pettersen
Enter value for firstname: Kari
Enter value for address: Storgt 20
Enter value for city: Stavanger
old 1: INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city')
new 1: INSERT into persons values(3,'Pettersen','Kari','Storgt 20','Stavanger')
1 row created.

SQL> SELECT * FROM persons;
```

P_ID	LASTNAME	FIRSTNAME	ADDRESS	CITY
1	Hansen	Ola	Timoteivn 10	sandnes
2	Svendson	Tove	Borgn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

LEFT JOIN SYNTAX

```
SQL> SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

LEFT JOIN EXAMPLE

```
SQL> SELECT persons.lastname,persons.firstname,orders.orderno
 FROM persons
 LEFT JOIN orders
 ON persons.p_Id = orders.p_Id
ORDER BY persons.lastname;
```

OUTPUT

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

FULL OUTTER JOIN

```
SQL> SELECT * FROM persons;

 P_ID   LASTNAME  FIRSTNAME          ADDRESS              CITY
---------- --------------- -------------------- ----------------              ----------
 1         Hansen          Ola               Timoteivn 10          sandnes
 2         Svendson         Tove              Borgn 23              Sandnes
 3         Pettersen         Kari              Storgt 20              Stavanger

SQL> SELECT * FROM orders;

 O_ID  ORDERNO     P_ID
---------- ----------       ----------
 1         77895           3
 2         44678           3
 3         22456           1
4          24562                    1
 5         34764           15
```

FULL OUTER JOIN SYNTAX

SQL>SELECT column_name(s)
FROM table_name1
FULL JOIN table_name2
ON table_name1.column_name=table_name2.column_name

FULL OUTER JOIN EXAMPLE

```
SQL> SELECT persons.lastname,persons.firstname,orders.orderno
 FROM persons
 FULL OUTER JOIN orders
 ON persons.p_Id = orders.p_Id
 ORDER BY persons.lastname;
```

RIGHT OUTTER JOIN

RIGHT OUTTER JOIN SYNTAX

SQL>SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
RIGHT JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName

RIGHT OUTTER JOIN EXAMPLE

```
SQL> SELECT persons.lastname,persons.firstname,orders.orderno
 FROM persons
 RIGHT OUTER JOIN orders
 ON persons.p_Id = orders.p_Id
 ORDER BY persons.lastname;
```

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

INNER JOIN

SQL>SELECT column_name(s)

```
FROM table_name1
INNER JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

INNER JOIN EXAMPLE

```
SQL> SELECT persons.lastname,persons.firstname,orders.orderno
 2 FROM persons
 3 INNER JOIN orders
 4 ON persons.p_Id = orders.p_Id
 5 ORDER BY persons.lastname;
```

LASTNAME	FIRSTNAME	ORDERNO
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

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

6 rows selected.

6. Write SQL queries to implement Views.

Views Helps to encapsulate complex query and make it reusable.

SQL COMMANDS

- CREATE VIEW
- INSERT IN VIEW
- DELETE IN VIEW
- UPDATE OF VIEW
- DROP OF VIEW

CREATION OF TABLE

```
SQL> CREATE TABLE EMPLOYEE (
EMPLOYEE_NAMEVARCHAR2(10),
EMPLOYEE_NONUMBER(8),
DEPT_NAME VARCHAR2(10),
DEPT_NO NUMBER (5),DATE_OF_JOIN DATE);
Table created.

SQL> DESC EMPLOYEE;

 NAME          NULL?                    TYPE
 ----------------------------- -------- ------------------------
 EMPLOYEE_NAME                          VARCHAR2(10)
 EMPLOYEE_NO                NUMBER(8)
 DEPT_NAME                              VARCHAR2(10)
 DEPT_NO                     NUMBER(5)
 DATE_OF_JOIN               DATE
```

SYNTAX FOR CREATION OF VIEW

SQL> CREATE <VIEW> <VIEW NAME> AS SELECT <COLUMN_NAME_1>, <COLUMN_NAME_2> FROM <TABLE NAME>;

SQL> CREATE VIEW EMPVIEW AS SELECT EMPLOYEE_NAME, EMPLOYEE_NO, DEPT_NAME, DEPT_NO, DATE_OF_JOIN FROM
EMPLOYEE;
VIEW CREATED.

DESCRIPTION OF VIEW

```
SQL> DESC EMPVIEW;
NAME                                      NULL?    TYPE
 ----------------------------------------- -------- ----------------------------
 EMPLOYEE_NAME                                      VARCHAR2(10)
 EMPLOYEE_NO                               NUMBER(8)
 DEPT_NAME                                          VARCHAR2(10)
 DEPT_NO                                   NUMBER(5)
```

DISPLAY VIEW:

```
SQL> SELECT * FROM EMPVIEW;

EMPLOYEE_N EMPLOYEE_NO DEPT_NAME DEPT_NO
---------- ----------- ---------- ----------
RAVI            124          ECE          89
VIJAY                345          CSE          21
RAJ             98           IT           22
GIRI            100          CSE           67
```

INSERTION INTO VIEW

SQL> INSERT INTO <VIEW_NAME> (COLUMN NAME1,……) VALUES(VALUE1,….);

```
SQL> INSERT INTO EMPVIEW VALUES ('SRI', 120,'CSE', 67,'16-NOV-1981');
1 ROW CREATED.

SQL> SELECT * FROM EMPVIEW;

EMPLOYEE_N EMPLOYEE_NO DEPT_NAME DEPT_NO
---------- ----------- ---------- ----------
RAVI            124          ECE          89
VIJAY                345          CSE          21
RAJ             98           IT           22
GIRI            100          CSE           67
SRI             120          CSE           67

SQL> SELECT * FROM EMPLOYEE;


EMPLOYEE_N EMPLOYEE_NO DEPT_NAME DEPT_NO DATE_OF_J
---------- ----------- ---------- ---------- ---------
```

RAVI	124	ECE	89	15-JUN-05
VIJAY	345	CSE	21	21-JUN-06
RAJ	98	IT	22	30-SEP-06
GIRI	100	CSE	67	14-NOV-81
SRI	120	CSE	67	16-NOV-81

DELETION OF VIEW:

SQL> DELETE <VIEW_NMAE>WHERE <COLUMN NMAE> =’VALUE’;

SQL> DELETE FROM EMPVIEW WHERE EMPLOYEE_NAME='SRI';
1 ROW DELETED.

SQL> SELECT * FROM EMPVIEW;

EMPLOYEE_N	EMPLOYEE_NO	DEPT_NAME	DEPT_NO
RAVI	124	ECE	89
VIJAY	345	CSE	21
RAJ	98	IT	22
GIRI	100	CSE	67

UPDATE STATEMENT:

SQL>UPDATE <VIEW_NAME> SET< COLUMN NAME> = <COLUMN NAME> <VIEW> WHERE <COLUMNNAME>=VALUE;

SQL> UPDATE EMPKAVIVIEW SET EMPLOYEE_NAME='KAVI' WHERE EMPLOYEE_NAME = 'RAVI';
1 ROW UPDATED.

SQL> SELECT * FROM EMPKAVIVIEW;

EMPLOYEE_N	EMPLOYEE_NO	DEPT_NAME	DEPT_NO
KAVI	124	ECE	89
RAJ	98	IT	22
GIRI	100	CSE	67

DROP A VIEW:

SQL> DROP VIEW <VIEW_NAME>

SQL>DROP VIEW EMPVIEW;
VIEW DROPED

7. Write SQL queries to SELECT data using SET UNION ,INTERSECTION and MINUS operations.

```
select empno,ename,sal from emp
UNION
select empno,ename,salary from oldemp

SELECT empno FROM emp
INTERSECT
SELECT empno FROM oldemp;

SELECT empno FROM emp
MINUS
SELECT empno FROM oldemp;
```

8. **Write SQL queries using AGGREGATE functions**.

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM
(Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

```
SELECT Shippers.ShipperName, Employees.LastName,
COUNT(Orders.OrderID) AS NumberOfOrders
FROM ((Orders
INNER JOIN Shippers
ON Orders.ShipperID=Shippers.ShipperID)
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID)
GROUP BY ShipperName,LastName;
```

9. Write SQL queries to implement Nested queries.

Nested Query can have more than one level of nesting in one single query. A SQL nested query is a SELECT query that is nested inside a SELECT, UPDATE, INSERT, or DELETE SQL query.

SQL COMMANDS:

- SELECT
- WHERE
- HAVING
- MIN(SALARY)

SQL: CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), COLUMN NAME.1 <DATATYPE> (SIZE) ……);

```
SQL> CREATE TABLE EMP2(EMPNO NUMBER(5),
ENAME VARCHAR2(20),
JOB VARCHAR2(20),
SAL NUMBER(6),
MGRNO NUMBER(4),
DEPTNO NUMBER(3));
```

SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',…..);

```
SQL> INSERT INTO EMP2 VALUES(1001,'MAHESH','PROGRAMMER',15000,1560,200);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1002,'MANOJ','TESTER',12000,1560,200);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1003,'KARTHIK','PROGRAMMER',13000,1400,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1004,'NARESH','CLERK',1400,1400,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1005,'MANI','TESTER',13000,1400,200);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1006,'VIKI','DESIGNER',12500,1560,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1007,'MOHAN','DESIGNER',14000,1560,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1008,'NAVEEN','CREATION',20000,1400,201);
1 ROW CREATED.
```

SQL> INSERT INTO EMP2 VALUES(1009,'PRASAD','DIR',20000,1560,202);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1010,'AGNESH','DIR',15000,1400,200);
1 ROW CREATED.

SQL> SELECT * FROM <TABLE NAME>;

SQL> SELECT *FROM EMP2;

EMPNO	ENAME	JOB	SAL		MGRNO	DPTNO
1001	MAHESH	PROGRAMMER	15000		1560	200
1002	MANOJ	TESTER	12000		1560	200
1003	KARTHIK	PROGRAMMER	13000		1400	201
1004	NARESH	CLERK		1400	1400	201
1005	MANI	TESTER	13000		1400	200
1006	VIKI	DESIGNER	12500		1560	201
1007	MOHAN	DESIGNER	14000		1560	201
1008	NAVEEN	CREATION	20000		1400	201
1009	PRASAD	DIR	20000		1560	202
1010	AGNESH	DIR	15000		1400	200

SQL: CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), COLUMN NAME.1 <DATATYPE> (SIZE) ……);

SQL> CREATE TABLE DEPT2(DEPTNO NUMBER(3),
DEPTNAME VARCHAR2(10),
LOCATION VARCHAR2(15));
Table created.

SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',…..);

SQL> INSERT INTO DEPT2 VALUES(107,'DEVELOP','ADYAR');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(201,'DEBUG','UK');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(200,'TEST','US');
SQL> INSERT INTO DEPT2 VALUES(201,'TEST','USSR');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(108,'DEBUG','ADYAR');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(109,'BUILD','POTHERI');
1 ROW CREATED.

SQL> SELECT * FROM <TABLE NAME>;

SQL> SELECT *FROM DEPT2;

DEPTNO	DEPTNAME	LOCATION
107	DEVELOP	ADYAR
201	DEBUG	UK
200	TEST	US
201	TEST	USSR
108	DEBUG	ADYAR
109	BUILD	POTHERI

6 rows selected.

GENERAL SYNTAX FOR NESTED QUERY:

SELECT "COLUMN_NAME1"
FROM "TABLE_NAME1"
WHERE "COLUMN_NAME2" [COMPARISON OPERATOR]
(SELECT "COLUMN_NAME3"
FROM "TABLE_NAME2"
WHERE [CONDITION])

SYNTAX NESTED QUERY STATEMENT:

SQL> SELECT <COLUMN_NAME> FROM FRORM <TABLE _1> WHERE <COLUMN_NAME> <RELATIONAL _OPERATION> 'VALUE'
(SELECT (AGGRECATE FUNCTION) FROM <TABLE_1> WHERE <COLUMN NAME> = 'VALUE'
(SELECT <COLUMN_NAME> FROM <TABLE_2> WHERE <COLUMN_NAME= 'VALUE'));

NESTED QUERY STATEMENT:

SQL> SELECT ENAME FROM EMP2 WHERE SAL>
(SELECT MIN(SAL) FROM EMP2 WHERE DPTNO=
(SELECT DEPTNO FROM DEPT2 WHERE LOCATION='UK'));

Nested Query Output:

ENAME

MAHESH
MANOJ

KARTHIK
MANI
VIKI
MOHAN
NAVEEN
PRASAD
AGNESH

10.Write a PL/SQL code block to find factorial of a number.

```
declare
    n number;
    i number;
    f number:=1;
    begin
    n:=&n;
    for i in 1..n
    loop
    f:=f*i;
    end loop;
    dbms_output.put_line(n||'! = '||f);
    end;
```

Output:

```
Enter value for n: 5
old 6: n:=&n;
new 6: n:=5;
5! = 120
PL/SQL procedure successfully completed.
```

11.Write a PL/SQL code block to implement Indexes.

```
SQL> SELECT * FROM vendor_parts
  2    WHERE part_no = 457 AND vendor_id = 1012
  3     ORDER BY vendor_id;


VENDOR_ID PART_NO  UNIT_COST
--------- ------- ----------
1012      457          4.95


1 row selected.
```

```
SQL> CREATE INDEX ind_vendor_id
  2    ON vendor_parts (part_no, vendor_id);


Index created.
```

12.Write a PL/SQL code block to implement Procedures.

```
Begin
i:=1;
tot:=0;
while(i<=n)
loop
j:=1;
c:=0;
while(j<=i)
loop
if(mod(I,j)=0) then
c:=c+1;
end if;
j:=j+1;
end loop;
if(c=2) then
dbms_output.put_line(i);
tot:=tot+1;
end if;
i:=i+1;
end loop;
end;
/
Sql>procedure created.
declare
```

t number;

begin

prime_proc(10,t);

dbms_output.put_line('the total prime no .are'||t);

end;

Valid Test Data:

sql>set serveroutput on

OUTPUT

sql>/

2

3

5

7

The total prime no.are 4

Pl/sql procedure successfully completed.

13.Write a PL/SQL code block to implement Triggers

```
Begin
2 For I in 1.. 10 loop
3 If mod(i,2) <> 0 then
4 Dbms_output.put_line(' I am an odd number :'||i);
5 else
6 Dbms_output.put_line(' I am an even number :'||i);
7 End if;
8 End loop;
9* end;
SQL> /
I am an odd number :1
I am an even number :2
I am an odd number :3
I am an even number :4
I am an odd number :5
I am an even number :6
I am an odd number :7
I am an even number :8
I am an odd number :9
I am an even number :10

PL/SQL procedure successfully completed.
```