1(a). MANIPULATE A DATABASE BY CREATING, INSERTING, DELETING, UPDATING AND RETRIEVING TABLES.

COMMANDS:

SQL> CREATEDATABASETest;
Database Created

SQL> CREATE TABLE Employee(EmployeeNo char(4), EmployeeName varchar2(30), EmployeeSalnumber(10,2), EmployeeCity varchar2(30), EmployeeDob date);

Table Created

SQL> INSERT INTO Employee values('2', 'Santosh', 5000, 'Delhi','23-DEC-1994');

1 row inserted

SQL>select * from Employee;

EMPLOYEENO	EMPLOYEENAME	EMPLOYEESAL	EMPLOYEECITY	EMPLOYEEDOB
2	Santosh	5000	Delhi	23-DEC-94

SQL> UPDATE Employee SET EmployeeName='KASHISH' WHERE EmployeeNo=1;

SQL>SELECT * from Employee;

EMPLOYEENO	EMPLOYEENAME	EMPLOYEESAL	EMPLOYEECITY	EMPLOYEEDOB
2	KASHISH	5000	Delhi	23-DEC-94

SQL>DELETE * from Employee;

0 row(s) deleted

1(b).IMPLEMENTATION OF DDL COMMANDS TO CREATE, ALTER AND DROP TABLE

COMMANDS:

Consider the below table:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL>ALTER TABLE CUSTOMERS ADD SEX char(1);

SQL>SELECT * FORM CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY	SEX
1	Ramesh	32	Ahmedabad	2000.00	NULL
2	Ramesh	25	Delhi	1500.00	NULL
3	kaushik	23	Kota	2000.00	NULL
4	kaushik	25	Mumbai	6500.00	NULL
5	Hardik	27	Bhopal	8500.00	NULL
6	Komal	22	MP	4500.00	NULL
7	Muffy	24	Indore	10000.00	NULL

SQL>ALTER TABLE CUSTOMERS DROP SEX;

SQL>SELECT * FORM CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Ramesh	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	kaushik	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL>TRUNCATE TABLE CUSTOMERS;

SQL>SELECT * FORM CUSTOMERS;

Empty set (0.00 sec)

SQL>DROP TABLE CUSTOMERS;

Query OK, 0 rows affected (0.01 sec)

SQL>DESC CUSTOMERS;

ERROR 1146 (42S02): Table 'TEST.CUSTOMERS' doesn't exist

2. IMPLEMENTATION OF DML COMMANDS FOR DATA INSERTION USING DIFFERENT WAYS, INTEGRITY CONSTRAINTS AND TRUNCATE

COMMANDS:

SQL>**CREATE DATABASE** Organization;

Database Created

SQL>CREATE TABLE Persons

(

PersonIDint,

LastNamevarchar(255),

FirstNamevarchar(255),

Address varchar(255),

City varchar(255)

);

SQL>INSERT INTO Persons (PersonID, LastName, FirstName, Address, City) VALUES ('101', 'Erichsen', 'Tom', 'Street no-21', 'New York');

SQL>INSERT INTO Persons (PersonID, LastName, FirstName, Address, City)

VALUES ('102', 'Johnson', 'Marry', 'Old Street Road-43', 'California');

SQL>INSERT INTO Persons (PersonID, LastName,FirstName)

VALUES ('103', 'Steve','Rossy')

SQL>INSERT INTO Persons VALUES ('104', 'Allen', 'Ketty', 'South Side Road', 'U.S.');

SQL>**select** * **from** persons

SQL>**TRUNCATE TABLE Persons;**
SQL>SELECT * FORM Persons;

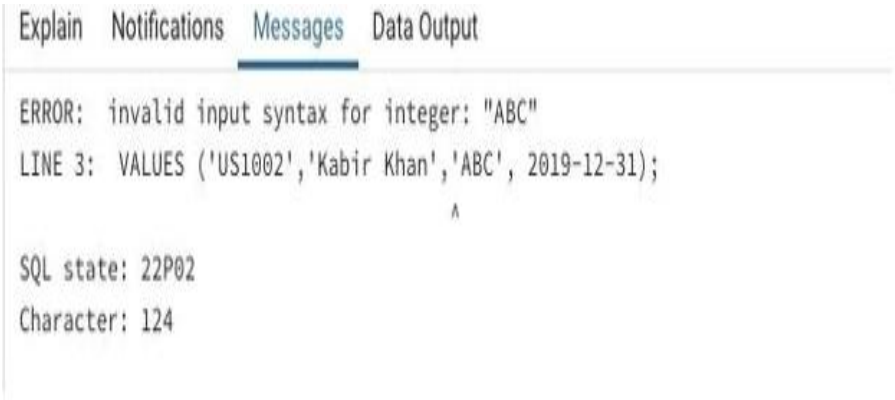
Empty set (0.00 sec)

SQL> CREATE TABLE customer_details(
customer_id character varying(255) NOT NULL,
customer_name character varying(255) NOT NULL,
quantity integer NOT NULL,
date_purchased date
);

Table Created.

SQL> INSERT INTO public.customer_details(
customer_id, customer_name, quantity, date_purchased)
VALUES ('US1002','Kabir Khan','ABC', 2019-12-31);

OUTPUT:



SQL> CREATE TABLE Students(
Student_IDint NOT NULL,
Student_Namevarchar(255) NOT NULL,
Class_Namevarchar(255) UNIQUE,
Age int,
PRIMARY KEY (Student_ID)
);

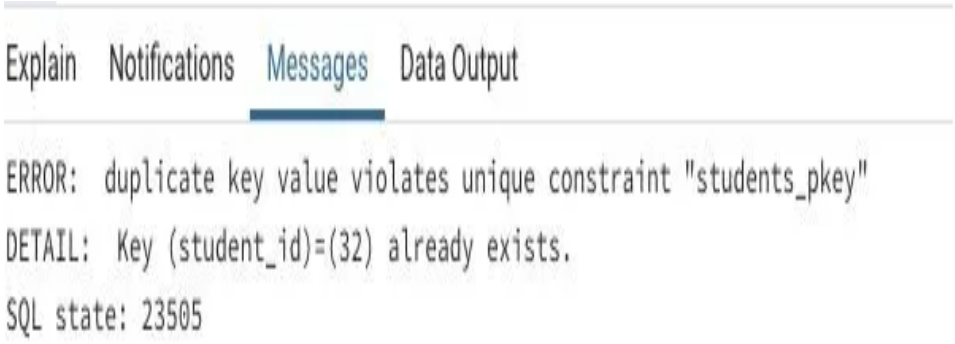
Table Created.

SQL> INSERT INTO public.students(
student_id, student_name, class_name, age)

VALUES (32,'ABC','V',12),(32,'XYZ','V',11);

SQL> CREATE TABLE Department(
Department_IDint NOT NULL,
Department_Namevarchar(255) NOT NULL,
PRIMARY KEY(Department_ID)
);

OUTPUT:

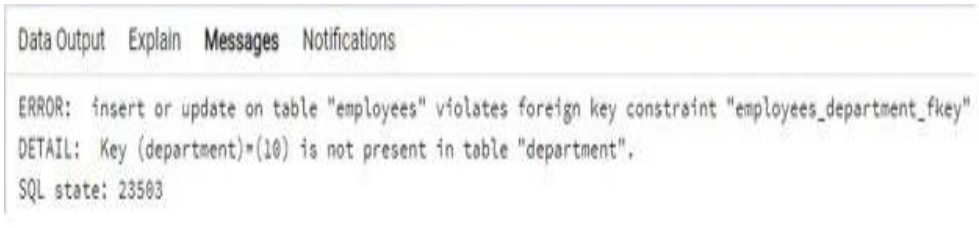


SQL>CREATE TABLE Employees(
Employee_IDint NOT NULL,
Employee_Namevarchar(255) NOT NULL,
Department int NOT NULL,
Age int,
FOREIGN KEY (Department) REFERENCES Department(Department_ID)
);

Table Created.

SQL> INSERT INTO public.employees(
employee_id, employee_name, department, age)
VALUES (1002,'K K Davis',10,43);

OUTPUT:


Data Output
Explain
Messages
Notifications
ERROR: insert or update on table "employees" violates foreign key constraint "employees_department_fkey"
DETAIL: Key (department)=(10) is not present in table "department".
SQL state: 23503

3. MANIPULATE TABLES IN A DATABASE USING SIMPLE QUERIES, NESTED QUERIES, SUB QUERIES AND JOINS

COMMANDS:

NESTED QUERY

SQL>create table student(id number(10), name varchar2(20),classID number(10), marks varchar2(20));

SQL>insert into student values(1,'pinky',3,2.4);

SQL>insert into student values(2,'bob',3,1.44);

SQL>insert into student values(3,'Jam',1,3.24);

SQL>insert into student values(4,'lucky',2,2.67);

SQL>insert into student values(5,'ram',2,4.56);

SQL>select * from student;

Id	Name	classID	Marks
1	Pinky	3	2.4
2	Bob	3	1.44
3	Jam	1	3.24
4	Lucky	2	2.67
5	Ram	2	4.56

SQL>Create table teacher(id number(10), name varchar(20), subject varchar2(10), classID number(10), salary number(30));

SQL>insert into teacher values(1,’bhanu’,’computer’,3,5000);

SQL>insert into teacher values(2,'rekha','science',1,5000);

SQL>insert into teacher values(3,'siri','social',NULL,4500);

SQL>insert into teacher values(4,'kittu','mathsr',2,5500);

SQL>select * from teacher;

Id	Name	Subject	classID	Salary
1	Bhanu	Computer	3	5000
2	Rekha	Science	1	5000
3	Siri	Social	NULL	4500
4	Kittu	Maths	2	5500

SQL>Create table class(id number(10), grade number(10), teacherID number(10), noofstudents number(10));

SQL>insert into class values(1,8,2,20);

SQL>insert into class values(2,9,3,40);

SQL>insert into class values(3,10,1,38);

SQL>select * from class;

Id	Grade	teacherID	No.ofstudents
1	8	2	20
2	9	3	40
3	10	1	38

SQL> Select AVG(noofstudents) from class where teacherID IN(Select id from teacher Where subject='science' OR subject='maths');

20.0

SQL> SELECT * FROM student WHERE classID = (SELECT id FROM class WHERE 2 noofstudents = (SELECT MAX(noofstudents) FROM class));

4|lucky |2|2.67

5|ram |2|4.56

JOINS:

EQUIJOIN.

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1|Ramesh|32|Ahmedabad|2000.00|
|2|Khilan|25|Delhi|1500.00|
|3|kaushik|23|Kota|2000.00|
|4|Chaitali|25|Mumbai|6500.00|
|5|Hardik|27|Bhopal|8500.00|
|6|Komal|22| MP        |4500.00|
|7|Muffy|24|Indore|10000.00|
+----+----------+-----+-----------+----------+
```

SQL>SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS INNER JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+----+----------+--------+---------------------+
```

LEFT JOIN

SQL>SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

Output:

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  1 | Ramesh   |   NULL | NULL                |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   NULL | NULL                |
|  6 | Komal    |   NULL | NULL                |
|  7 | Muffy    |   NULL | NULL                |
+----+----------+--------+---------------------+
```

RIGHT JOIN

SQL>SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

Output:

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

FULL JOIN

SQL>SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS FULL JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

Output:

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

SELF JOIN:

SQL>SELECT a.ID, b.NAME, a.SALARY FROM CUSTOMERS a, CUSTOMERS b

WHERE a.SALARY<b.SALARY;

Output:

```
+----+----------+---------+
| ID | NAME     | SALARY  |
+----+----------+---------+
|  2 | Ramesh   | 1500.00 |
|  2 | kaushik  | 1500.00 |
|  1 | Chaitali | 2000.00 |
|  2 | Chaitali | 1500.00 |
|  3 | Chaitali | 2000.00 |
|  6 | Chaitali | 4500.00 |
|  1 | Hardik   | 2000.00 |
|  2 | Hardik   | 1500.00 |
|  3 | Hardik   | 2000.00 |
|  4 | Hardik   | 6500.00 |
```

```
|   6 | Hardik   | 4500.00 |
|   1 | Komal    | 2000.00 |
|   2 | Komal    | 1500.00 |
|   3 | Komal    | 2000.00 |
|   1 | Muffy    | 2000.00 |
|   2 | Muffy    | 1500.00 |
|   3 | Muffy    | 2000.00 |
|   4 | Muffy    | 6500.00 |
|   5 | Muffy    | 8500.00 |
|   6 | Muffy    | 4500.00 |
+-----+----------+---------+
```

CARTESIAN JOIN :

SQL>SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS, ORDERS;

Output:

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  1 | Ramesh   |   3000 | 2009-10-08 00:00:00 |
|  1 | Ramesh   |   1500 | 2009-10-08 00:00:00 |
|  1 | Ramesh   |   1560 | 2009-11-20 00:00:00 |
|  1 | Ramesh   |   2060 | 2008-05-20 00:00:00 |
|  2 | Khilan   |   3000 | 2009-10-08 00:00:00 |
|  2 | Khilan   |   1500 | 2009-10-08 00:00:00 |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  2 | Khilan   |   2060 | 2008-05-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   2060 | 2008-05-20 00:00:00 |
|  4 | Chaitali |   3000 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   1560 | 2009-11-20 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   3000 | 2009-10-08 00:00:00 |
|  5 | Hardik   |   1500 | 2009-10-08 00:00:00 |
|  5 | Hardik   |   1560 | 2009-11-20 00:00:00 |
|  5 | Hardik   |   2060 | 2008-05-20 00:00:00 |
|  6 | Komal    |   3000 | 2009-10-08 00:00:00 |
|  6 | Komal    |   1500 | 2009-10-08 00:00:00 |
|  6 | Komal    |   1560 | 2009-11-20 00:00:00 |
|  6 | Komal    |   2060 | 2008-05-20 00:00:00 |
|  7 | Muffy    |   3000 | 2009-10-08 00:00:00 |
|  7 | Muffy    |   1500 | 2009-10-08 00:00:00 |
|  7 | Muffy    |   1560 | 2009-11-20 00:00:00 |
|  7 | Muffy    |   2060 | 2008-05-20 00:00:00 |
+----+----------+--------+---------------------+
```

4. IMPLENTATION OF AGGREGATION FUNCTIONS, GROUPING AND ORDERING COMMANDS TO MANIPULATE TABLES IN A DATABASE

COMMANDS:

Let's consider an employee table. We will perform the calculations on this table by using aggregate functions.

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000
E004	XYZ	24	Mumbai	4000
E005	STU	32	Bangalore	25000

SQL> select AVG(salary) from employee;

16800

SQL> select MAX(salary) from employee;

30000

SQL> select MIN (salary) from employee;

4000

SQL>select SUM (salary) from employee where city='Pune';

50000

SQL> select COUNT(Empid) from employee;

5

SQL> select COUNT(*) from employee;

5

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
   GROUP BY NAME;
+----------+-------------+
| NAME     | SUM(SALARY) |
+----------+-------------+
| Chaitali |     6500.00 |
| Hardik   |     8500.00 |
| kaushik  |     2000.00 |
| Khilan   |     1500.00 |
| Komal    |     4500.00 |
| Muffy    |    10000.00 |
| Ramesh   |     2000.00 |
+----------+-------------+
```

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY NAME, SALARY;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
```

```
+----+----------+-----+-----------+----------+
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   |  27 | Bhopal    |  8500.00 |
| 3 | kaushik  |  23 | Kota      |  2000.00 |
| 2 | Khilan   |  25 | Delhi     |  1500.00 |
| 6 | Komal    |  22 | MP        |  4500.00 |
| 7 | Muffy    |  24 | Indore    | 10000.00 |
| 1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
SQL> SELECT * FROM CUSTOMERS
  ORDER BY NAME DESC;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
| 7 | Muffy    |  24 | Indore    | 10000.00 |
| 6 | Komal    |  22 | MP        |  4500.00 |
| 2 | Khilan   |  25 | Delhi     |  1500.00 |
| 3 | kaushik  |  23 | Kota      |  2000.00 |
| 5 | Hardik   |  27 | Bhopal    |  8500.00 |
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
+----+----------+-----+-----------+----------+
```

5(a).IMPLEMENT DCL COMMANDS TO SET AND REVOKE PRIVILEGES

COMMANDS:

SQL> Grant Create session to student;

SQL> Grant create table to student;

SQL> Connect student/young;

SQL> Connect system/managers;

SQL> Create user staff identified by guru;

SQL> Grant resource to staff;

SQL> Connect staff/guru;

SQL> Select * from staff;

Staff master in a table in the user staff we first log on the staff [SQL> Connect staff/guru;]

SQL> Grant select insert on staff master to student;

Now log on to student and try the select command

SQL> Connect student/young;

SQL> Select * from staff;

SQL> Grant select, update, delete on student-master to staff;.

SQL> REVOKE SELECT, INSERT ON STUDENT_MASTER from staff;

SQL> REVOKE SELECT ON STUDENT_MASTER from rajan;

5(b).IMPLEMENTATION OF TCL COMMANDS SAVE-POINT, ROLL BACK AND ROLL BACK TO COMMANDS

COMMANDS:

Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL> DELETE FROM CUSTOMERS WHERE AGE = 25;

SQL>ROLLBACK;

SQL> select * from customers

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL> DELETE FROM CUSTOMERS WHERE AGE = 25;

SQL>COMMIT;

SQL> select * from customers

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL> SAVEPOINT SP1;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=1;

1 row deleted.

SQL> SAVEPOINT SP2;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=2;

1 row deleted.

SQL> SAVEPOINT SP3;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=3;

1 row deleted.

SQL> ROLLBACK TO SP2;

Rollback complete.

SQL> SELECT * FROM CUSTOMERS;

```
+----+----------+-----+-----------+----+
|ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
6 rows selected.
```

6.IMPLEMENTATION OF PL/SQL USING CONDITIONAL STATEMENTS

PROGRAM:

Program to find whether a given number by user is even or odd:

```
setserveroutputon;

DECLARE
        xint;
BEGIN
        x := 15;
        if mod(x,2) = 0 then
                dbms_output.put_line('Even Number');
        else
                dbms_output.put_line('Odd Number');
        end if;
END;
/
```

Output:

Odd Number

Program to find whether the two given numbers are equal and if they are not equal then which one is greater:

```
setserveroutputon;

DECLARE
        aint;
        bint;
BEGIN
        a := 10;
        b := 20;
        if(a>b) then
                dbms_output.put_line('a is greater than b');
        elsif(b>a) then
                dbms_output.put_line('b is greater than a');
        else
                dbms_output.put_line('Both a and b are equal');
        end if;
END;
/
```

Output:

b is greater than a

Program to demonstrate the use of a simple case statement.

```
setserveroutputon;

DECLARE
grade CHAR(1);
BEGIN
grade := 'B';


  CASE grade
    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
    WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
    WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
    WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
    ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
  END CASE;
END;
/
```

Output:

Very Good

7. IMPLEMENTATION OF IMPLICIT AND EXPLICIT CURSOR TO MANIPULATE A TABLE IN PL/SQL

PROGRAM:

1. Program to update the table and increase the salary of each customer by 500

Using implicit Cursor

```
SQL>Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+

SQL>DECLARE
2 total_rowsnumber(2);
  3 BEGIN
4 UPDATE customers
5 SET salary = salary + 500;
6 IF sql%notfound THEN
7 dbms_output.put_line('no customers selected');
8 ELSIF sql%foundTHEN
9 total_rows := sql%rowcount;
10 dbms_output.put_line(total_rows || ' customers selected ');
11END IF;
 12 END;
```

Output:

```
6 customers selected
PL/SQL procedure successfully completed.
SQL>Select * from customers;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2500.00 |
|  2 | Khilan   |  25 | Delhi     |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2500.00 |
|  4 | Chaitali |  25 | Mumbai    |  7000.00 |
```

```
|  5 | Hardik   |  27 | Bhopal    |  9000.00 |
|  6 | Komal    |  22 | MP        |  5000.00 |
+----+----------+-----+-----------+----------+
```

2. Program to illustrate the concepts of explicit cursors &minua;

```
SQL>DECLARE
2 c_idcustomers.id%type;
 3c_namecustomers.name%type;
4c_addrcustomers.address%type;
5 CURSOR c_customersis
6 SELECT id, name, address FROM customers;
   7 BEGIN
8 OPEN c_customers;
9 LOOP
10 FETCH c_customers into c_id, c_name, c_addr;
11 EXIT WHEN c_customers%notfound;
12cdbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
13  END LOOP;
14 CLOSE c_customers;
  15 END;
/
```

Output:

```
1 Ramesh Ahmedabad
2 Khilan Delhi
3 kaushik Kota
4 Chaitali Mumbai
5 Hardik Bhopal
6 Komal MP

PL/SQL procedure successfully completed.
```

8.IMPLEMENTATION OF CREATING AND DROPING A TRIGGER IN PL/SQL

COMMANDS:

```
SQL> create table itstudent4(name varchar2(15),username varchar2(15));

Table created.

SQL> create or replace trigger student4 before insert on student4 for each row
 2 declare
 3 name varchar2(20);
 4 begin
 5 select user into name from dual;
6 :new.username:=name;
 7 end;
8 /

Trigger created.
```

Output:

```
SQL> insert into student4 values('&name','&username');

Enter value for name: akbar

Enter value for username: ranjani

old 1: insert into student4 values('&name','&username')

new 1: insert into student4 values('akbar','ranjani')

1 row created.

SQL> /

Enter value for name: suji

Enter value for username: priya

old 1: insert into student4 values('&name','&username')

new 1: insert into student4 values('suji','priya')

1 row created.

SQL> select * from itudent4;
```

NAME USERNAME

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

akbar SCOTT

suji SCOTT

Develop a query to Drop the Created Trigger

SQL> drop trigger ittrigg;

Trigger dropped.

9.IMPLEMENTATION OF PROCEDURE AND FUNCTION MANIPULATE A DATABASE USING PL/SQL

Tables used:

SQL> select * from ititems;

ITEMID ACTUALPRICE ORDID PRODID

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

101 2000 500 201

102 3000 1600 202

103 4000 600 202

Program For General Procedure – Selected Record'S Price Is Incremented By 500 , Executing The Procedure Created And Displaying The Updated Table

```
SQL> create procedure itsum(identity number, total number) is price number;
 2 null_price exception;
 3 begin
 4 select actualprice into price from ititems where itemid=identity;
 5 if price is null then
 6 raise null_price;
 7 else
 8 update ititems set actualprice=actualprice+total where itemid=identity;
 9 end if;
10 exception
11 when null_price then
12 dbms_output.put_line('price is null');
13 end;
14 /
Procedure created.
SQL> exec itsum(101, 500);
```

```
PL/SQL procedure successfully completed.

SQL> select * from ititems;

ITEMID ACTUALPRICE ORDID PRODID

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

 101 2500 500 201

 102 3000 1600 202

 103 4000 600 202
```

Procedure For IN Parameter – Creation, Execution

```
SQL> set serveroutputon;

SQL> create procedure yyy (a IN number) is price number;

 2 begin

 3 select actualprice into price from ititems where itemid=a;

4 dbms_output.put_line('Actual price is ' || price);

 5 if price is null then

 6 dbms_output.put_line('price is null');

 7 end if;

 8 end;

 9 /

Procedure created.

SQL> exec yyy(103);

Actual price is 4000

PL/SQL procedure successfully completed.
```

Procedure For OUT Parameter – Creation, Execution

```
SQL> set serveroutputon;

SQL> create procedure zzz (a in number, b out number) is identity number;

 2 begin

 3 select ordid into identity from ititems where itemid=a;

 4 if identity<1000 then
```

```
 5 b:=100;
 6 end if;
 7 end;
 8 /
Procedure created.
SQL> declare
 2 a number;
 3 b number;
 4 begin
 5 zzz(101,b);
 6 dbms_output.put_line('The value of b is '|| b);
 7 end;
 8 /
The value of b is 100
PL/SQL procedure successfully completed.
```

Procedure For INOUT Parameter – Creation, Execution

```
SQL> create procedure itit( ainout number) is
 2 begin
 3 a:=a+1;
 4 end;
 5 /
 Procedure created.
SQL> declare
 2 a number:=7;
 3 begin
 4 itit(a);
 5 dbms_output.put_line(„The updated value is „||a);
 6 end;
```

```
7 /
```

The updated value is 8

PL/SQL procedure successfully completed.

Tables used:

```
SQL>select * from ittrain;

 TNO TFARE
--------- ------------
 1001 550
 1002 600
```

Program For Function And It‘s Execution

```
SQL> create function trainfn (trainnumber number) return number is
 2 trainfunctionittrain.tfare % type;
 3 begin
 4 select tfare into trainfunction from ittrain where tno=trainnumber;
 5 return(trainfunction);
 6 end;
 7 /
```

Function created.

```
SQL> declare
 2 total number;
 3 begin
 4 total:=trainfn (1001);
 5 dbms_output.put_line('Train fare is Rs. '||total);
 6 end;
 7 /
```

Train fare is Rs.550

PL/SQL procedure successfully completed.

Factorial Of A Number Using Function — Program And Execution

```
SQL> create function itfact (a number) return number is
 2 fact number:=1;
 3 b number;
 4 begin
 5 b:=a;
 6 while b>0
 7 loop
 8 fact:=fact*b;
 9 b:=b-1;
10 end loop;
11 return(fact);
12 end;
13 /
Function created.
SQL> declare
 2 a number:=7;
 3 f number(10);
 4 begin
 5 f:=itfact(a);
 6 dbms_output.put_line(„The factorial of the given number is"||f);
 7 end;
 8 /
The factorial of the given number is 5040
```

Procedure to calculate total for the all the students and pass regno, mark1, & mark2 as arguments.

```
SQL> create table student2(regno number(3),name varchar(9),mark1 number(3),mark2
number(3));
Table created.
```

```
SQL> insert into student2
 2 values(&a,'&b',&c,&d);
Enter value for a: 110
Enter value for b: arun
Enter value for c: 99 Enter value for d: 100
old 2: values(&a,'&b',&c,&d)
new 2: values(110,'arun',99,100)
1 row created.
SQL> /
Enter value for a: 112 Enter value for b: siva Enter value for c: 99 Enter value
for d: 90
old 2: values(&a,'&b',&c,&d)
new 2: values(112,'siva',99,90)
1 row created.
SQL> select * from student2;
 REGNO NAME MARK1 MARK2
 110 arun 99 100
 112 siva 99 90
SQL> alter table student2 add(total number(5)); Table altered.
SQL> select * from student2;
REGNO NAME MARK1 MARK2 TOTAL
 110 arun 99 100
 112 siva 99 90
SQL> create or replace procedure p1(sno number,mark1 number,mark2 number) is
 2 tot number(5);
 3 begin
 4 tot:=mark1+mark2;
 5 update itstudent2 set total=tot where regno=sno;
```

```
 6 end;
 7 /
Procedure created.
SQL> declare
 2 cursor c1 is select * from student2;
 3 rec itstudent2 % rowtype;
 4 begin
 5 open c1;
 6 loop
 7 fetch c1 into rec;
8 exit when c1%notfound;
 9 p1(rec.regno,rec.mark1,rec.mark2);
10 end loop;
11 close c1;
12 end;
13 /
PL/SQL procedure successfully completed.
```

Output:

```
SQL> select * from student2;
REGNO NAME MARK1 MARK2 TOTAL
--------- --------- ---------- ---------- ----------
 110 arun 99 100 199
112 va 99 90 189
```

PL/SQL procedure that takes two numbers as parameter and displays the multiplication of the first parameter till the second parameter.

//p2.sql

```
create or replace procedure multi_table (a number, b number) as
mulnumber;
begin
```

```
fori in 1. .b
loop
mul : = a * i;
dbms_output.put_line (a || „*“ || i || „=“ || mul);
end loop;
end;
```

//pq2.sql

```
declare
a number; b number;
begin
a:=&a; b:=&b; multi_table(a,b);
end;
```

Output:

```
SQL> @p2.sql;
 Procedure created.
SQL> @pq2.sql;
Enter value for a: 4
old 5: a:=&a; new 5: a:=4;
Enter value for b: 3
old 6: b:=&b; new 6: b:=3;
4*1=4
4*2=8
4*3=12
```

Consider the EMPLOYEE (EMPNO, SALARY, ENAME) Table.

Write a procedure raise_sal which increases the salary of an employee. It accepts an employee number and salary increase amount. It uses the employee number to find the current salary from the EMPLOYEE table and update the salary.

//p3.sql

```
create or replace procedure raise_sal( mempno employee . empno % type, msal_percent
```

```
number ) as
begin
update employee set salary = salary + salary*msal_percent /100 where empno = mempno;
end;
/
```

//pq3.sql

```
declare
cursor c1 is select * from emp;
recemp % rowtype;
begin
open c1;
loop
fetch c1 into rec;
exit when c1%notfound;
raisal(rec.empno,10);
end loop;
close c1;
end;
 /
```

Output:

```
SQL> @p3.sql;
 Procedure created.
SQL> select * from emp;
 EMPNO ENAME JOB DEPTNO SAL
---------- -------------------- ------------ ---------- ----------
 1 Mathi AP 1 10000
 2 Arjun ASP 2 15000
 3 Gugan ASP 1 15000
```

4 Karthik Prof 2 30000

5 Akalya AP 1 10000

SQL> @pq3.sql;

PL/SQL procedure successfully completed.

SQL> select * from emp;

EMPNO ENAME JOB DEPTNO SAL

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

1 Mathi AP 1 11000

2 Arjun ASP 2 16500

3 Gugan ASP 1 16500

4 Karthik Prof 2 33000

5 Akalya AP 1 11000

Write a PL/SQL function CheckDiv that takes two numbers as arguments and returns thevalues 1 if the first argument passed to it is divisible by the second argument, else will return thevalue 0;

//p4.sql

```
create or replace function checkdiv (n1 number, n2 number) return number as res
number;
begin
if mod (n1, n2) = 0 then
res := 1;
else
res:= 0;
end if;
return res;
end;
/
```

//pq4.sql

```
declare
```

```
a number;
b number;
begin
a:=&a; b:=&b;
dbms_output.put_line(„result=“||checkdiv(a,b));
end;
/
```

Output:

```
SQL> @p4.sql;
Function created.
SQL> @pq4.sql;
Enter value for a: 4
old 5: a:=&a; new 5: a:=4;
Enter value for b: 2
old 6: b:=&b; new 6: b:=2;
result=1
```

Write a PL/SQL function called POW that takes two numbers as argument and return thevalue of the first number raised to the power of the second .

//p5.sql

```
create or replace function pow (n1 number, n2 number) return number as
res number;
begin
select power ( n1, n2) into res from dual; return res;
end;
or
create or replace function pow (n1 number, n2 number) return number as
res number : =1;
begin
for res in 1..n2
```

```
loop
res : = n1 * res;
end loop;
return res;
end;
```

//pq5.sql

```
declare
a number;
b number;
begin
a:=&a; b:=&b;
dbms_output.put_line('power(n1,n2)='||pow(a,b));
end;
/
```

Output:

```
SQL> @p5.sql;
Function created.
SQL> @ pq5.sql;
Enter value for a: 2
old 5: a:=&a;
new 5: a:=2;
Enter value for b: 3
old 6: b:=&b;
new 6: b:=3;
power(n1,n2)=8
```

Write a PL/SQL function ODDEVEN to return value TRUE if the number passed to it isEVEN else will return FALSE.

//p6.sql

```
create or replace function oddeven (n number) return boolean as
```

```
begin
if mod (n, 2) = 0 then return true;
else
return false;
end if;
end;
/
```

//pq6.sql

```
declare
a number; b boolean;
begin
a:=&a; b:=oddeven(a);
if b then
dbms_output.put_line('The given number is Even');
else
dbms_output.put_line('The given number is Odd');
end if;
end;
/
```

Output:

```
SQL> @p6.sql;
Function created.
SQL> @pq6.sql;
Enter value for a: 5
old 5: a:=&a; new 5: a:=5;
The given number is Odd
```

10. IMPLEMENTATION OF HANDLING EXCEPTION IN QUERY

COMMANDS:

```
SET SERVEROUTPUT ON;
DECLARE
c_idcustomers.id%type := 8;
c_namecustomerS.Name%type;
c_addrcustomers.address%type;
BEGIN
SELECT  name, address INTO  c_name, c_addr
  FROM customers
  WHERE id = c_id;
  DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
  DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
  WHEN no_data_found THEN
dbms_output.put_line('No such customer!');
  WHEN others THEN
dbms_output.put_line('Error!');
END;
/
```

OUTPUT:

```
No such customer!
PL/SQL procedure successfully completed.
```

User-defined Exceptions

```
DECLARE
c_idcustomers.id%type :=&cc_id;
c_namecustomerS.Name%type;
```

```
c_addrcustomers.address%type;
  -- user defined exception
ex_invalid_idEXCEPTION;
BEGIN
  IF c_id<= 0 THEN
    RAISE ex_invalid_id;
  ELSE
SELECT  name, address INTO  c_name, c_addr
    FROM customers
    WHERE id = c_id;
    DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
    DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
  END IF;
EXCEPTION
  WHEN ex_invalid_id THEN
dbms_output.put_line('ID must be greater than zero!');
  WHEN no_data_found THEN
dbms_output.put_line('No such customer!');
  WHEN others THEN
dbms_output.put_line('Error!');
END;
/
```

OUTPUT:

```
Enter value for cc_id: -6 (let's enter a value -6)
old  2: c_idcustomers.id%type := &cc_id;
new  2: c_idcustomers.id%type := -6;
ID must be greater than zero!
PL/SQL procedure successfully completed.
```

11. DESIGNING A DATABASE USING ER MODELLING AND NORMALIZATION

First Normal Form

1. Create a property table with the following fields: property id, country name, padd, area, price, tax rate and having property id as the primary key.

SQL> create table prop(propid number(2) primary key, cnamevarchar(20), paddvarchar(50), area int,

price number(9,2),tax_rate number(2));

SQL>desc prop;

Name Null? Type

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

PROPID NOT NULL NUMBER(2)

CNAME VARCHAR2(20)

PADD VARCHAR2(50)

AREA NUMBER(38)

PRICE NUMBER(9,2)

TAX_RATE NUMBER(2)

2. Insert values in the property table.

SQL> insert into prop values('34','india','ganthi nagar,Coimbatore, india','500','500000','2');

1 row created.

SQL> insert into prop values('45','united states','first street southeast, Washington, United states','400','2550000','5');

1 row created.

SQL> insert into prop values('39','scotland','capelrig road, Glasgow, scotland','600','2500000','4');

1 row created.

Before Normalization

prop

PropidCnamePadd Area Price Tax_rate

Normalization to first normal form

1. Creating the prop11 tabale with propid, cname, area,price, tax_rate from prop.

SQL> create table prop11 as select ,cname, area,price, tax_rate from prop;

2. Creating the table prop12 with propid, sname,city,country from prop

SQL> create table prop12 as select propid,padd from emp;

3. Altering the table prop11 with primary key on prop.

SQL> alter table prop12 add constraint c1 foreign key(propid) references prop11(propid);

4. Altering the table prop12 with foreign key on propid with reference from prop11.

SQL> alter table prop12 add constraint c1 foreign key(propid) references prop11(propid);

After Normalization

Prop11

PropidCname Area Price Tax_rate

Prop12

Propidsname City country

SECOND NORMAL FORM

Normalization to Second Normal Form

1. Create the table prop21 with propid, cname, area, price from the table prop.

SQL> create table prop21 as select propid,cname,area, price from prop;

2. Create the table prop22 with cname, tax_rate from the table prop.

SQL> create table prop22 as select cname,tax_rate from prop;

3. Alter table prop21 with a primary key constraint on propid.

SQL> alter table prop21 add constraint prop21 primary key(propid);

4. Alter table prop22 with a primary key constraint on cname.

SQL> alter table prop22 add constraint prop22 primary key(cname);

5. Alter table prop21 with foreign key on cname with references on cname from prop22.

SQL> alter table prop21 add constraint prop212 foreign key(cname) references prop22(cname);

After normalization

Prop21 prop22

PropidCname Area Price

THIRD NORMAL FORM

The 2NF table is given as input here and convert it to 3NF.

Input: prop21, prop22 tables.

For converting to 3NF it is enough making changes in prop21 table.

Before Normalization

Prop21

PropidCname Area Price

1. Create table prop31 with propid, cname, area from prop21.

SQL> create table prop31 as select propid,cname,area from prop21;

2. Create table prop32 with area, price from prop21.

SQL> create table prop32 as select area, price from prop21;

3. Alter table prop31 with the constraint primary key on propid.

SQL> alter table prop31 add constraint prop31 primary key(propid);

4. Alter table prop32 with the constraint primary key on area.

SQL> alter table prop32 add constraint prop32 primary key(area);

5. Alter table prop31 with the constraint foreign key on area with refernce from area in prop32.

SQL> alter table prop31 add constraint prop311 foreign key(area) references prop32(area);

After Normalization

Prop31 prop32

PropidCname Area

12. DEVELOPING AN ENTERPRISE APPLICATION USING USER INTERFACE AND DATABASE

VB SCRIPT:

ADD:

Private Sub Add_Click() Adodc1.Recordset.AddNew Textl.SetFocus End Sub

DELETE:

Private Sub Delete_Click()

If MsgBox ("DELETE IT?",vbOKCancel)= vbOK Then

Adodc1.Recordset.Delete End If MsgBox "ONE ROW DELETED" Textl.Text- " "

Text2.Text - " "

Text3.Text - " "

Text4.Text - " "

Text5.Text - " "

Text6.Text - " "

Text7.Text - " "

Text8.Text - " "

Text9.Text - " " Textl0.Text - " " End Sub SAVE:

Private Sub Save_Click()

If MsgBox ("SAVE IT?",vbOKCancel) = vbOK Then Adodc1.Recordset.Update Else Adodc1.Recordset.CancelUpdate

End If End Sub FIND:

Private Sub Find_Click() Dim N as string

N = InputBox ("Enter the accno") Adodc1.Recordset.Find "accno=" & N

If Adodcl.Recordset.BOF or Adodc1.Recordset.EOF Then MsgBox "Record not found" End If End Sub

UPDATE:

Private Sub Update_Click() Adodc1.Recordset.EditMode Adodc1.Recordset.Update End Sub FIRST:

Private Sub First_Click() Adodc1.Recordset.MoveFirst End Sub LAST:

Private Sub Last_Click() Adodc1.Recordset.MoveLast End Sub NEXT:

Private Sub Next_Click() Adodc1.Recordset.MoveNext End Sub PREVIOUS:

Private Sub Previous_Click() Adodc1.Recordset.MovePrevious End Sub DEPOSIT:

Private Sub Deposit_Click0 Dim N1 as string

N = InputBox ("Enter the accno") Adodcl.Recordset.Find "accno=" & N Nl = InputBox ("Enter the amount") Text4.Text= val (Text4.Text) + Nl Adodc1.Recordset.Update

End Sub

WITHDRAW:

Private Sub Withdraw_Click() Dim Nl as string

N = InputBox ("Enter the accno") Adodcl.Recordset.Find "accno=" & N Nl = InputBox ("Enter the amount") Text4.Text= val (Text4.Text) - NlAdodcl.Recordset.Update

End Sub EXIT:

Private Sub Add_Click() Unload Me End Sub FUNCTION:

Function Calculate()

Text8.Text=val(Text4.Text) + val (Text5.Text) + val (Text6.Text) + val (Text7.Text) Text9.Text=val(Text5.Text) + val (Text6.Text) + val (Text7.Text)

Text 10.Text=val(Text8.Text) + val (Text9.Text) End Function BASICPAY,HRA,DA,MA,GROSSPAY,DEDUCTION,NETPAY:

Private Sub Basicpay_Change() Call Calculate End Sub

Private Sub HRA_Change() Call Calculate End Sub

Private Sub DA_Change() Call Calculate End Sub

Private Sub MA_Change() Call Calculate

End Sub

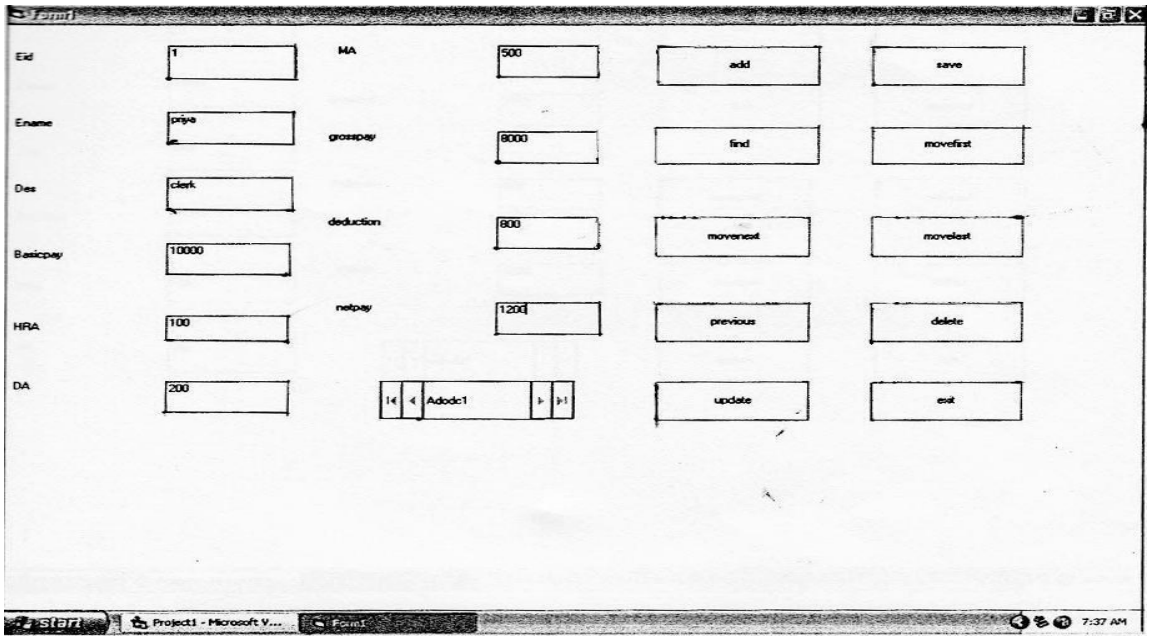
Private Sub Grosspay_Change() Call Calculate End Sub

Private Sub Deduction_Change() Call Calculate End Sub

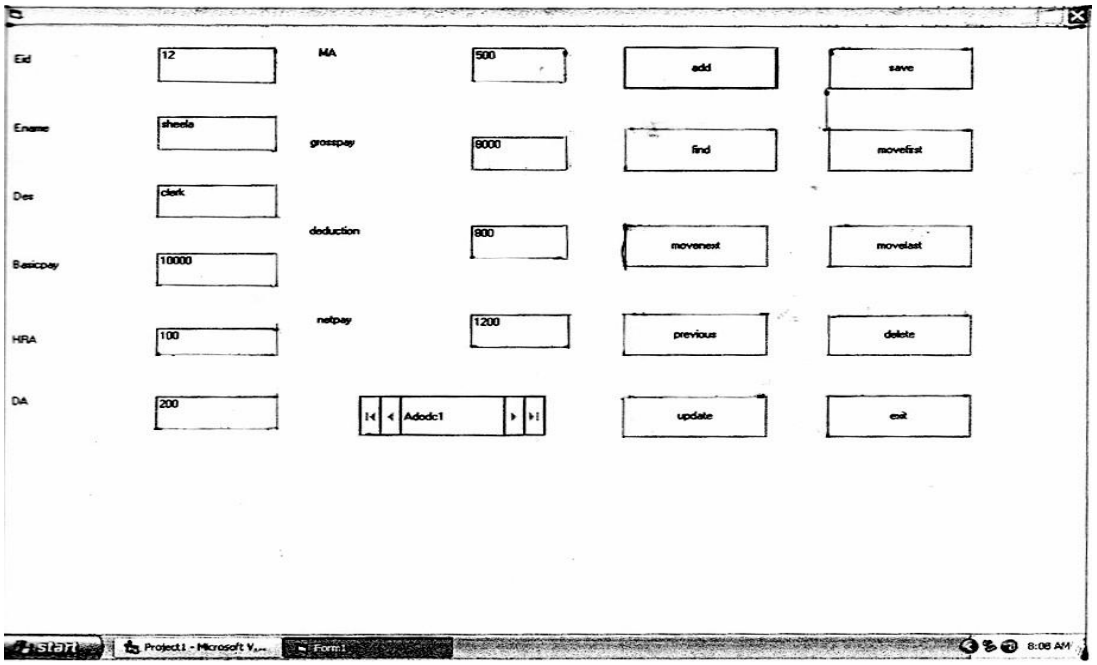
Private Sub Netpay_Change() Call Calculate End Sub

OUTPUT:

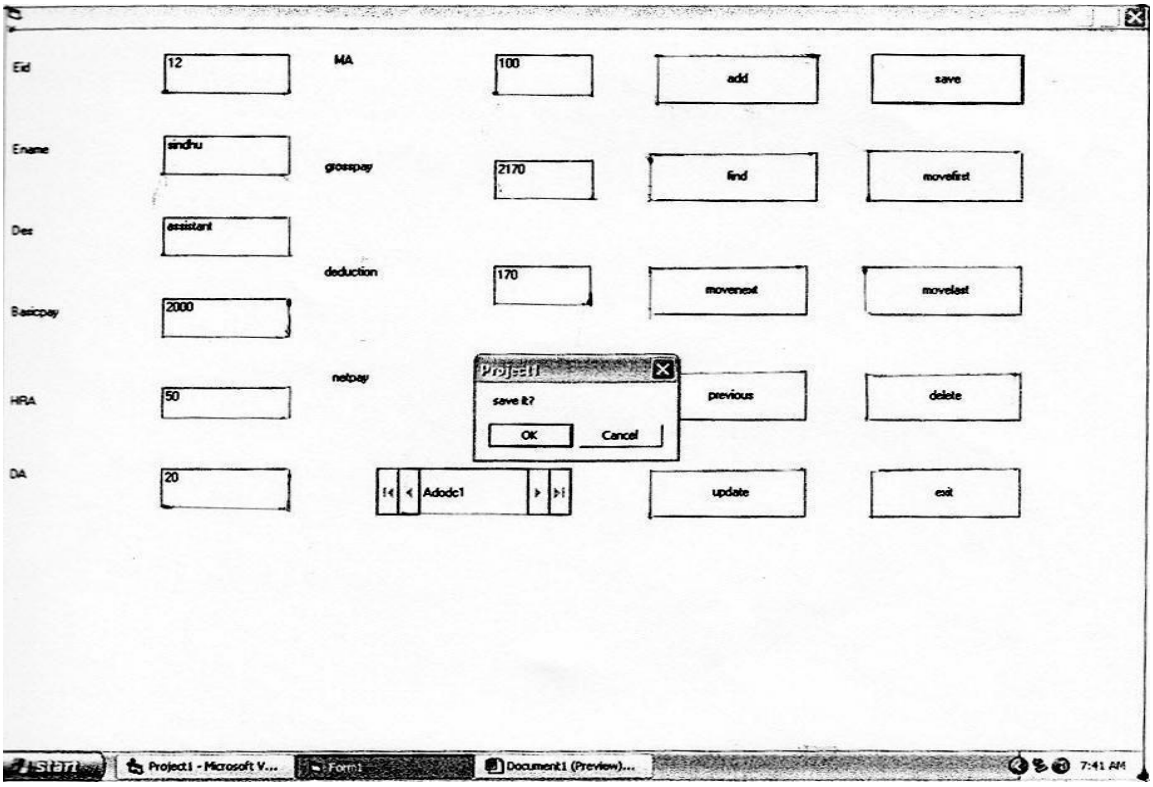
STARTUP FORM



ADDFORM



SAVEFORM


Eid
12
MA
100
add
save
Ename
sindhu
grosspay
2170
find
movefirst
Des
assistant
deduction
170
movenext
movelast
Basicpay
2000
netpay
Project1
save it?
OK
Cancel
HRA
50
previous
delete
DA
20
Adodc1
update
exit
Project1 - Microsoft V...
Form1
Document1 (Preview)...
7:41 AM

EX.No.:13

IMPLEMENTATION OF CREATING INDEX

AIM:

To create and drop index inatable

PROCEDURE:

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. An index is a pointer to data in a table. An index in a database is very similar to an index in the back of abook. An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data. Index in sql is created on existing tables to retrieve the rows quickly. When there are thousands of records in a table, retrieving information will take a long time. When an index is created, it first sorts the data and then it assigns a ROWID for each row.

An index can be created in a table to find data more quickly and efficiently.

The users cannot see the indexes, they are just used to speed up searches/queries

1.Syntax to create Index

CREATE INDEX index_name ON table_name (column_name1,column_name2...);

2.Syntax to create SQL unique index

CREATE UNIQUE INDEX index_name ON table_name (column_name1, column_name2...);

•index_name is the name of the INDEX.

•table_name is the name of the table to which the indexed column belongs.

•column_name1, column_name2..is the list of columns which make up the INDEX.

3.The Drop Index Command

An index can be dropped using SQL DROP command. Care should be taken when dropping an index because performance may be slowed or improved.

DROP INDEX index_name;

13.IMPLEMENTATION OF CREATING INDEX

COMMANDS:

SQL> create table persons (first name varchar (20), last name varchar(10));

Table created;

Create an index for the above relation based on last name

SQL> create index plndex on persons (last name);

Index created.

SQL> select * from persons; No rows selected.

SQL> drop plndex on persons;

Drop index plndex on persons

* ERROR at line1:

ORA_00950: Invalid DROP option

RESULT:

Thus the index has been created and dropped in a table has been implemented and executed successfully.

EX.No.:14 INTRODUCTION TO NOSQL DATABASES USING MONGODB

Aim:

The objective is to introduce some features of non-relational or NoSQL databases using MongoDB. MongoDB stores data in JSON objects which it calls documents and uses a custom language for queries.

Installation

Option 1:

1. Set up a free cluster on Mongo Atlas by following the instructions here: https://docs.atlas.mongodb.com/tutorial/deploy-free-tier-cluster/
2. Install the mongo shell on your machine and connect to your cluster following the instructions on the dashboard.

Option 2:

1. Download and setup MongoDB for your OS following these instructions https://www.mongodb.com/download-center/community

2. Start the server. Then use mongo shell to connect your server: https://docs.mongodb.com/manual/mongo/ Preparation The instructions below assume you have MongoDB installed and started on your local machine. For Atlas, create a database and a user by following the instructions on the dashboard.

1. Once you have installed and started the Mogodb you can log into your server as root. mongo -u root

2. To create a new database do use mongolab

3. Now create a new user to access the database.

db.createUser({user:"e14xxx", pwd:"abc123", roles:[{role: "dbOwner" , db:"mongolab"}]})

1. Log out of the mongo shell and log back in using the user you created. mongo localhost/mongolab -u e14xxx

Data Validation:

Document databases are a flexible alternative to the predefined schemas of relational databases. Each document in a collection can have a unique set of fields, and those fields can be added or removed from documents once they are inserted. Since the data fields can be changed for each document in a collection, data validation is extremely important to ensure queries run predictability.

To create the customer collection with a custom data validation function, enter the following code.

```
db.createCollection("customers", {
validator: {
$and: [
{
"firstName": {$type: "string", $exists: true}
 },
{
"lastName": { $type: "string", $exists: true}
},
{
"phoneNumber":
{
$type: "string",
$exists: true,
$regex: /^[0-9]{3}-[0-9]{3}-[0-9]{4}$/
}
},
{
"email": {
$type: "string",
$exists: true
}
}    ]
} })
```

RESULT:

Thus the Data validation in NoSQL databases using MongoDB has been implemented and executed successfully.