PSGR
Krishnammal College for Women

Affiliated to Bharathiar University \ Autonomous \ College of Excellence \ Accredited with A++ Grade \ Ranked 9th in NIRF

DEPARTMENT OF COMPUTER SCIENCE (PG)

ADBMS LAB - (MCS24P3)

2025 - 2027

PSGR
Krishnammal College for Women

Affiliated to Bharathiar University \ Autonomous \ College of Excellence \ Accredited with A++ Grade \ Ranked 9th in NIRF

DEPARTMENT OF COMPUTER SCIENCE (PG)

ADBMS LAB (MCS24P3)

REGISTER NUMBER: _____

Certified that this is a bonafide record work done by _____ of I MSc (Computer Science) during the year 2025-2027.

FACULTY INCHARGE **HEAD OF THE DEPARTMENT**

Submitted for the practical examination held on by _____ at PSGR Krishnammal College for Women, Coimbatore.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

S No	Date	Topics	Page No	Sign
SQL				
1		Implementation constraints		
2		Join operation		
3		Partition query		
4		Parallel query		
5		Object queries		
MongoDB				
6		Student database in MongoDB		
7		CRUD operations		
8		Querying and filtering		
9		MongoDB to perform aggregation		
10		MongoDB Backup and Restore		
Neo4j				
11		Neo4j for creating Node and Relationship		
12		Neo4j for Retrieving Data		
Applications				
1		Banking Management system		
2		Inventory system for Supermarket		
3		Student information system		
4		Doctor’s appointment system		
5		Employee payroll system		

Ex No: 1 Date:	IMPLEMENTATION CONSTRAINTS

AIM

To Create an employee table and perform Insertion, updation, deletion and set the following Constraints for Primary, Foreign Key, Check, Unique, Null

ALGORITHM

STEP 1: Start the process.

STEP 2: Create a table Emp30 with required attributes emp_id, name, age, email_id, phone.

STEP 3: Set **Primary Key** constraint for emp_id.

STEP 4: Apply **Check** constraint for age (age < 18 is violated).

STEP 5: Apply **Unique** constraint for email_id and Not-Null constraint for phone.

STEP 6: Insert records into Emp30 table and display the result.

STEP 7: Create another table empaddress with required fields such as emp_id, address and apply **Foreign Key** constraint referencing Emp30(emp_id).

STEP 8: Insert records into empaddress table and display the result.

STEP 9: Display both tables using SELECT command.

STEP 10: Perform Updation operation and display the table.

STEP 11: Perform Deletion operation and verify integrity constraint.

STEP 12: Stop the process.

PROGRAM

1) IMPLEMENTATION OF PRIMARY KEY, CHECK, UNIQUE AND NULL CONSTRAINTS

```
SQL> CREATE TABLE Emp30 (
  2     emp_id INT NOT NULL PRIMARY KEY,
  3     Name VARCHAR(10) NOT NULL,
  4     Age INT CHECK (Age >= 18),
  5     Email_id VARCHAR(50) UNIQUE,
  6     Phone INT NULL
  7  );
```

Table created.

2) IMPLEMENTATION OF FOREIGN KEY CONSTRAINT

```
SQL> CREATE TABLE empaddress (
  2     emp_id INT,
  3     city VARCHAR(30),
  4     FOREIGN KEY (emp_id) REFERENCES Emp30(emp_id)
  5  );
```

Table created.

3) INSERTION OPERATION

```
SQL> INSERT INTO Emp30 VALUES (101, 'Divya', 26, 'divya3123@gmail.com',
9723463728);
```

1 row created.

```
SQL> INSERT INTO Emp30 VALUES (102, 'Revathi', 23, 'revathi54@gmail.com',
8436737078);
```

1 row created.

```
SQL> INSERT INTO Emp30 VALUES (103, 'Malini', 24, 'malini@gmail.com',
6795327095);
```

1 row created.

```
SQL> INSERT INTO Emp30 VALUES (104, 'Varshini', 22, 'varshini@gmail.com',
9998887776);
```

1 row created.

SQL>
SQL> INSERT INTO empaddress VALUES (101, 'Dindigul');

1 row created.

SQL> INSERT INTO empaddress VALUES (102, 'Trichy');

1 row created.

SQL> INSERT INTO empaddress VALUES (103, 'Karur');

1 row created.

SQL> INSERT INTO empaddress VALUES (104, 'Namakkal');

1 row created.

4) DISPLAYING THE TABLES

SQL> SELECT * FROM Emp30;

```
   EMP_ID NAME            AGE
---------- ---------- ----------
EMAIL_ID                                   PHONE
---------------------------------------------------------- ----------
      101 Divya           26
divya3123@gmail.com                         9723463728

      102 Revathi          23
revathi54@gmail.com                         8436737078

      103 Malini           24
malini@gmail.com                           6795327095


   EMP_ID NAME            AGE
---------- ---------- ----------
EMAIL_ID                                   PHONE
---------------------------------------------------------- ----------
      104 Varshini         22
varshini@gmail.com                         9998887776
```

```
SQL> SELECT * FROM empaddress;

    EMP_ID CITY
---------- --------------------------------
       101 Dindigul
       102 Trichy
       103 Karur
       104 Namakkal
```

5) UPDATION OPERATION

```
SQL> UPDATE Emp30
  2  SET Age = 27
  3  WHERE emp_id = 101;

1 row updated.

SQL> SELECT * FROM Emp30;

    EMP_ID NAME            AGE
---------- ----------- ----------
EMAIL_ID                                                  PHONE
---------------------------------------------------------- ----------
       101 Divya           27
divya3123@gmail.com                        9723463728

       102 Revathi         23
revathi54@gmail.com                        8436737078

       103 Malini          24
malini@gmail.com                          6795327095


    EMP_ID NAME            AGE
---------- ----------- ----------
EMAIL_ID                                                  PHONE
---------------------------------------------------------- ----------
       104 Varshini        22
varshini@gmail.com                         9998887776
```

6) DELETION OPERATION

```
SQL> DELETE FROM Emp30
  2 WHERE emp_id = 104;
DELETE FROM Emp30
*
ERROR at line 1:
ORA-02292: integrity constraint (SYSTEM.SYS_C008355) violated - child record
found
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 2 Date:	JOIN OPERATION

AIM

To Create an employee table and perform following join operation for Self join , Inner join , Outer join , Left join , Right join

ALGORITHM

STEP 1: Start the process.

STEP 2: Create a table named employee_info with attributes emp_id, emp_name, joining_date using the CREATE command.

STEP 3: Set emp_id as the Primary Key to make it unique.

STEP 4: Insert records into the employee_info table using the INSERT command.

STEP 5: Display the employee_info table using the SELECT command to verify the inserted data.

STEP 6: Create another table named department_info with attributes emp_id, department, location and insert records into it using the INSERT command.

STEP 7: Display the department_info table using the SELECT command.

STEP 8: Perform a SELF JOIN on employee_info to compare records within the same table and display the result.

STEP 9: Perform an INNER JOIN between employee_info and department_info to display only the matching records.

STEP 10: Perform a LEFT JOIN to display all records from employee_info along with matching records from department_info, if any.

STEP 11: Perform a RIGHT JOIN to display all records from department_info along with

matching records from employee_info, if any.

STEP 12: Perform a FULL OUTER JOIN to display all records from both tables with matches where available.

STEP 13: Stop the process.

PROGRAM

1) CREATE TABLE

```
CREATE TABLE department_info (
 EMP_ID NUMBER(5),
 DEPARTMENT VARCHAR2(20),
 LOCATION VARCHAR2(20)
);
INSERT INTO department_info VALUES (101, 'HR', 'Chennai');
INSERT INTO department_info VALUES (102, 'Finance', 'Bangalore');
INSERT INTO department_info VALUES (105, 'IT', 'Hyderabad');

SQL> SELECT * FROM department_info;

   EMP_ID DEPARTMENT           LOCATION
---------- ---------------------- ----------------------
       101 HR                 Chennai
       102 Finance             Bangalore
       105 IT                 Hyderabad
```

2) IMPLEMENTATION OF SELF JOIN OPERATION

```
SQL> SELECT e1.emp_id, e1.emp_name, e1.joining_date
  2  FROM employee_info e1
  3  JOIN employee_info e2
  4  ON e1.emp_id = e2.emp_id;

   EMP_ID EMP_NAME            JOINING_D
---------- ---------------------- ----------
       101 Anitha              10-JAN-23
       102 Bharathi            15-MAR-23
       103 Chandru             20-MAY-23
```

104 Deepak 25-JUL-23

3) IMPLEMENTATION OF INNER JOIN OPERATION

```
SQL> SELECT e.emp_id, e.emp_name, e.joining_date, d.department, d.location
  2  FROM employee_info e
  3  INNER JOIN department_info d
  4  ON e.emp_id = d.emp_id;

   EMP_ID EMP_NAME            JOINING_D DEPARTMENT
---------- ---------------------- ---------- ----------------------
LOCATION
----------
      101 Anitha              10-JAN-23 HR
Chennai

      102 Bharathi            15-MAR-23 Finance
Bangalore
```

4) IMPLEMENTATION OF LEFT OPERATION

```
SQL> SELECT e.emp_id, e.emp_name, e.joining_date, d.department, d.location
  2  FROM employee_info e
  3  LEFT JOIN department_info d
  4  ON e.emp_id = d.emp_id;

   EMP_ID EMP_NAME            JOINING_D DEPARTMENT
---------- ---------------------- ---------- ----------------------
LOCATION
----------
      101 Anitha              10-JAN-23 HR
Chennai

      102 Bharathi            15-MAR-23 Finance
```

Bangalore

104 Deepak 25-JUL-23

EMP_ID EMP_NAME JOINING_D DEPARTMENT

LOCATION

---

103 Chandru 20-MAY-23

5) IMPLEMENTATION OF RIGHT JOIN OPERATION

```
SQL> SELECT e.emp_id, e.emp_name, e.joining_date, d.department, d.location
  2  FROM employee_info e
  3  RIGHT JOIN department_info d
  4  ON e.emp_id = d.emp_id;
```

EMP_ID EMP_NAME JOINING_D DEPARTMENT

LOCATION

---

101 Anitha 10-JAN-23 HR

Chennai

102 Bharathi 15-MAR-23 Finance

Bangalore

IT

Hyderabad

6) IMPLEMENTATION OF FULL OUTER JOIN OPERATION

```
SQL> SELECT e.emp_id, e.emp_name, e.joining_date, d.department, d.location
  2  FROM employee_info e
  3  FULL OUTER JOIN department_info d
  4  ON e.emp_id = d.emp_id;

    EMP_ID EMP_NAME            JOINING_D DEPARTMENT
---------- ---------- ---------- ----------
LOCATION
----------
       101 Anitha              10-JAN-23 HR
Chennai

       102 Bharathi            15-MAR-23 Finance
Bangalore

       103 Chandru             20-MAY-23



    EMP_ID EMP_NAME            JOINING_D DEPARTMENT
---------- ---------- ---------- ----------
LOCATION
----------
       104 Deepak              25-JUL-23


                                IT
Hyderabad
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 3 Date:	PARTITION QUERY

AIM

To create a table using range partitioning, list partitioning and hash partitioning and display the output.

ALGORITHM

STEP 1: Start the process

STEP 2: RANGE PARTITION

• Create table salesdata with fields (sid, sname, sales_amount, sales_date)

• Include range partition for sales_date for Jan, Feb, Mar

• Insert values into salesdata

• View values from each partition using select query

STEP 3: LIST PARTITION

• Create table sales_records with fields (sid, sname, sales_amount, sales_date)

• Include list partition for sid

• Insert values into sales_records

• View values from each partition using select query

STEP 4: HASH PARTITION

• Create table sales_hash with fields (sid, sname, sales_amount, sales_date)

• Include hash partition for sid with 4 partitions

• Insert values into sales_hash

STEP 5: Stop the process

PROGRAM

1)TO IMPLEMENT RANGE PARTITIONING

```
SQL> CREATE TABLE salesdata (
        sid NUMBER,
        sname VARCHAR2(100),
        sales_amount NUMBER(12,2),
       sales_date DATE
    )
 PARTITION BY RANGE (sales_date)
  (
      PARTITION sales_jan VALUES LESS THAN (DATE '2025-02-01'),
      PARTITION sales_feb VALUES LESS THAN (DATE '2025-03-01'),
      PARTITION sales_mar VALUES LESS THAN (DATE '2025-04-01'),
      PARTITION sales_other VALUES LESS THAN (DATE '2100-01-01')
  );
Table created.

SQL> INSERT INTO salesdata VALUES (1, 'A', 5000, DATE '2025-01-10');
1 row created.

SQL> INSERT INTO salesdata VALUES (2, 'B', 6500, DATE '2025-02-15');
1 row created.

SQL> INSERT INTO salesdata VALUES (3, 'C', 7200, DATE '2025-10-08');
1 row created.

SQL> SELECT * FROM salesdata PARTITION (sales_jan);

       SID
-----------
SNAME
--------------------------------------------------------------------------------
```

SALES_AMOUNT SALES_DAT

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

1

A

5000 10-JAN-25

SQL> SELECT * FROM salesdata PARTITION (sales_feb);

SID

SNAME

SALES_AMOUNT SALES_DAT

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

2

B

6500 15-FEB-25

SQL> SELECT * FROM salesdata PARTITION (sales_mar);

no rows selected

SQL> SELECT * FROM salesdata PARTITION (sales_other);

SID

SNAME

SALES_AMOUNT SALES_DAT

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

3

C

7200 08-OCT-25

2) TO IMPLEMENT LIST PARTITIONING

```
SQL> CREATE TABLE sales_records (
        sid NUMBER,
        sname VARCHAR2(100),
       sales_amount NUMBER(12,2),
        sales_date DATE
      )
   PARTITION BY LIST (sid)
    (
       PARTITION sid_1_to_3 VALUES (1,2,3),
      PARTITION sid_4_to_6 VALUES (4,5,6),
      PARTITION sid_other VALUES (DEFAULT)
    );
Table created.

SQL> INSERT INTO sales_records VALUES (1, 'John', 5000, DATE '2025-01-15');
1 row created.

SQL> INSERT INTO sales_records VALUES (2, 'Mary', 6200, DATE '2025-02-20');
1 row created.

SQL> INSERT INTO sales_records VALUES (4, 'Anu', 8000, DATE '2025-07-10');
1 row created.

SQL> SELECT * FROM sales_records WHERE sid BETWEEN 1 AND 3 ORDER BY sid;

       SID
----------
SNAME
--------------------------------------------------------------------------------
SALES_AMOUNT SALES_DAT
-------------- ----------
         1
```

John

5000 15-JAN-25

2

Mary

6200 20-FEB-25

SID

SNAME

SALES_AMOUNT SALES_DAT

--- ---

SQL> SELECT * FROM sales_records WHERE sid BETWEEN 4 AND 6 ORDER BY sid;

SID

SNAME

SALES_AMOUNT SALES_DAT

--- ---

4

Anu

8000 10-JUL-25

SQL> SELECT * FROM sales_records WHERE sid NOT BETWEEN 1 AND 6 ORDER BY sid;

no rows selected

3) TO IMPLEMENT HASH PARTITIONING

```
SQL> CREATE TABLE sales_hash (
  2      sid NUMBER,
  3      sname VARCHAR2(100),
  4      sales_amount NUMBER(12,2),
  5      sales_date DATE
  6  )
  7  PARTITION BY HASH (sid)
  8  PARTITIONS 4;

Table created.

SQL> INSERT INTO sales_hash VALUES (1, 'John', 5000, DATE '2025-01-15');
1 row created.

SQL> INSERT INTO sales_hash VALUES (2, 'Mary', 6200, DATE '2025-02-20');
1 row created.

SQL> INSERT INTO sales_hash VALUES (3, 'Sam', 7100, DATE '2025-03-25');
1 row created.

SQL> SELECT * FROM sales_hash ORDER BY sid;

       SID
-----------
SNAME
--------------------------------------------------------------------------------
SALES_AMOUNT SALES_DAT
------------ ----------
         1
John
      5000 15-JAN-25

         2
Mary
```

```
      6200 20-FEB-25

      SID
----------
SNAME
--------------------------------------------------------------------------------
SALES_AMOUNT SALES_DAT
-------------- ----------

       3
Sam
      7100 25-MAR-25


SQL> SELECT * FROM sales_hash WHERE sid = 3;

      SID
----------
SNAME
--------------------------------------------------------------------------------
SALES_AMOUNT SALES_DAT
-------------- ----------
       3
Sam
      7100 25-MAR-25
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 4 Date:	PARALLEL QUERY

AIM

To Create table employee (empno, dept, salary) and another table emp2 using the parallel query.

ALGORITHM

STEP 1: Start the process

STEP 2: Create table emp1(empno, dept, salary)

STEP 3: Insert values into emp1

STEP 4: Create table emp2 using parallel query concept

CREATE TABLE emp2 PARALLEL AS SELECT * FROM emp1;

STEP 5: Insert values into emp2

STEP 6: Display values of both tables

STEP 7: Alter table emp1 to add age column

STEP 8: Insert new record in emp1 and display final output

STEP 9: Stop the process

PROGRAM

1)CREATE TABLE AS EMP1 TABLE

```
SQL> CREATE TABLE emp1 (
  2    empno NUMBER,
  3    dept VARCHAR2(20),
  4    salary NUMBER
  5  );
```

Table created.

2) INSERT VALUES INTO EMP1

```
SQL> INSERT INTO emp1 VALUES (1,'HR',40000);
```

1 row created.

```
SQL> INSERT INTO emp1 VALUES (2,'IT',50000);
```

1 row created.

```
SQL> INSERT INTO emp1 VALUES (3,'Manager',55000);
```

1 row created.

3) DISPLAY EMP1 TABLE

```
SQL> SELECT * FROM emp1;

    EMPNO DEPT                 SALARY
---------- -------------------- ----------
         1 HR                   40000
```

2 IT 50000

3 Manager 55000

4) Create EMP2 using PARALLEL

SQL> CREATE TABLE emp2 PARALLEL AS

2 SELECT * FROM emp1;

Table created.

5) INSERT VALUE INTO EMP2

SQL> INSERT INTO emp2 VALUES (4,'CS',43643);

1 row created.

6) DISPLAY EMP2

SQL> SELECT * FROM emp2;

EMPNO DEPT SALARY

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

1 HR 40000

2 IT 50000

3 Manager 55000

4 CS 43643

7) ALTER EMP1 TABLE

SQL> ALTER TABLE emp1 ADD age NUMBER;

Table altered.

8) INSERT VALUE INTO EMP1 AFTER ALTER

SQL> INSERT INTO emp1 VALUES (5,'Finance',38000,24);

1 row created.

9) TO DISPLAY EMP1

SQL> SELECT * FROM emp1;

EMPNO	DEPT	SALARY	AGE
1	HR	40000	
2	IT	50000	
3	Manager	55000	
5	Finance	38000	24

10) TO DISPLAY EMP2 (UNCHANGED)

SQL> SELECT * FROM emp2;

EMPNO	DEPT	SALARY
1	HR	40000
2	IT	50000
3	Manager	55000
4	CS	43643

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 5 Date:	OBJECT QUERIES

AIM

To create an object type address, create an employee table using the object type, insert values, and update the address of an employee.

ALGORITHM

STEP 1: Start the process.

STEP 2: Create type for data as object with fields street, city, state.

STEP 3: Create table employee with fields emp_id, emp_name, address.

STEP 4: Insert values into employee table using the object type.

STEP 5: Display the contents of the employee table.

STEP 6: Update the address of employee with emp_id = 1 and display the result.

STEP 7: Stop the process.

PROGRAM

1) CREATE ADDRESS OBJECT TYPE

```
SQL> CREATE TYPE address_type AS OBJECT (
  2     street VARCHAR2(30),
  3     city   VARCHAR2(20),
  4     state  VARCHAR2(20)
  5  );
  6  /
```

Type created.

2) CREATE EMPLOYEE TABLE USING OBJECT TYPE

```
SQL> CREATE TABLE employee (
  2     emp_id   NUMBER,
  3     emp_name VARCHAR2(20),
  4     addr     address_type
  5  );
```

Table created.

3) INSERT VALUES IN EMPLOYEE TABLE

```
SQL> INSERT INTO employee VALUES (
  2     1,
  3     'Ravi',
  4     address_type('MG Road','Chennai','Tamil Nadu')
```

```
  5  );
```

1 row created.

```
SQL> INSERT INTO employee VALUES (
  2      2,
  3      'Anu',
  4      address_type('Park Street','Bangalore','Karnataka')
  5  );
```

1 row created.

4) DISPLAY EMPLOYEE TABLE

```
SQL> SELECT e.emp_id, e.emp_name,
  2         e.addr.street,
  3         e.addr.city,
  4         e.addr.state
  5  FROM employee e;

    EMP_ID EMP_NAME            ADDR.STREET
---------- ---------- ---------- ----------
ADDR.CITY            ADDR.STATE
---------- ----------

         1 Ravi                MG Road
Chennai              Tamil Nadu

         2 Anu                 Park Street
```

Bangalore Karnataka

5) UPDATE ADDRESS

```
SQL> UPDATE employee
  2  SET addr = address_type('Anna Nagar','Chennai','Tamil Nadu')
  3  WHERE emp_id = 1;
1 row updated.
```

6) DISPLAY AFTER UPDATE

```
SQL> SELECT e.emp_id, e.emp_name,
  2         e.addr.street,
  3         e.addr.city,
  4         e.addr.state
  5  FROM employee e;

    EMP_ID EMP_NAME            ADDR.STREET
---------- ---------- ---------- ---------- ----------
ADDR.CITY           ADDR.STATE
---------- ---------- ---------- ---------- ----------
         1 Ravi                Anna Nagar
Chennai             Tamil Nadu

         2 Anu                 Park Street
Bangalore           Karnataka
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 6 Date:	STUDENT DATABASE IN MONGODB

AIM

To create MongoDB collections, insert student records, and perform queries to display students based on percentage, address, and course with sorting.

ALGORITHM

STEP 1: Start the MongoDB server and open the MongoDB shell.

STEP 2: Select the required database using the use command.

STEP 3: Create a capped collection named students.

STEP 4: Insert student records into the students collection with fields such as id, name, course, and percentage.

STEP 5: Create another capped collection named students2.

STEP 6: Insert records into the students2 collection with fields such as id, age, gender, and address.

STEP 7: Retrieve all students whose percentage is greater than 50.

STEP 8: Retrieve students from students2 collection whose address is erode and gender is female.

STEP 9: Retrieve BSC students from the students collection and sort them in descending order based on percentage.

PROGRAM

1) CREATE COLLECTION AS STUDENTS

```
test> use test
already on db test
test> db.createCollection("students", { capped: true, size: 1024, max: 50 })
{ ok: 1 }
```

2) INSERT VALUES FOR STUDENTS

```
test> db.students.insertOne({ id:1, name:"nivi", course:"cs", percentage:85 })
... db.students.insertOne({ id:2, name:"neena", course:"cs", percentage:80 })
... db.students.insertOne({ id:3, name:"jayasree", course:"it", percentage:92 })
... db.students.insertOne({ id:4, name:"harsha", course:"it", percentage:88 })
... db.students.insertOne({ id:5, name:"meena", course:"bsc", percentage:85 })
... db.students.insertOne({ id:6, name:"pooja", course:"bsc", percentage:82 })
{
  acknowledged: true,
  insertedId: ObjectId('6978332629cca47864628ca5')
}
```

4) CREATE COLLECTION AS STUDENTS2

```
test> db.createCollection("students2", { capped: true, size: 1024, max: 50 })
{ ok: 1 }
test> db.students2.insertOne({ id:1, age:"23", gender:"female", address:"coimbatore" })
... db.students2.insertOne({ id:2, age:"20", gender:"female", address:"coimbatore" })
... db.students2.insertOne({ id:3, age:"25", gender:"female", address:"erode" })
... db.students2.insertOne({ id:4, age:"24", gender:"female", address:"karur" })
... db.students2.insertOne({ id:5, age:"23", gender:"female", address:"salem" })
```

```
... db.students2.insertOne({ id:6, age:"22", gender:"female", address:"tiruppur" })
{
  acknowledged: true,
  insertedId: ObjectId('6978333c29cca47864628cab')
}
```

5) DISPLAY ALL STUDENTS COMING FROM COIMBATORE

```
test> db.students2.find({ address:"coimbatore" })
[
  {
    _id: ObjectId('6978288d7f626d21b2628ca6'),
    id: 1,
    age: '23',
    gender: 'female',
    address: 'coimbatore'
  },
  {
    _id: ObjectId('6978288d7f626d21b2628ca7'),
    id: 2,
    age: '20',
    gender: 'female',
    address: 'coimbatore'
  },
  {
    _id: ObjectId('6978333b29cca47864628ca6'),
    id: 1,
```

```
    age: '23',
    gender: 'female',
    address: 'coimbatore'
  },
  {
    _id: ObjectId('6978333b29cca47864628ca7'),
    id: 2,
    age: '20',
    gender: 'female',
    address: 'coimbatore'
  }
]
```

6) DISPLAY ALL STUDENTS GETTING ABOVE 50 PERCENT

```
test> db.students.find({ percentage: { $gt: 50 } })
[
  {
    _id: ObjectId('697828697f626d21b2628ca1'),
    id: 2,
    name: 'neena',
    course: 'cs',
    percentage: 80
  },
  {
    _id: ObjectId('697828697f626d21b2628ca2'),
    id: 3,
```

```
  name: 'kavya',
  course: 'cs',
  percentage: 90
},
{
  _id: ObjectId('697828697f626d21b2628ca3'),
  id: 4,
  name: 'suhi',
  course: 'cs',
  percentage: 89
},
{
  _id: ObjectId('697828697f626d21b2628ca4'),
  id: 5,
  name: 'jayasree',
  course: 'it',
  percentage: 92
},
{
  _id: ObjectId('697828697f626d21b2628ca5'),
  id: 6,
  name: 'harsha',
  course: 'it',
  percentage: 88
},
```

```
{
  _id: ObjectId('69782c380addd89353628ca0'),
  id: 1,
  name: 'nivi',
  course: 'cs',
  percentage: 85
},
{
  _id: ObjectId('69782c380addd89353628ca1'),
  id: 2,
  name: 'neena',
  course: 'cs',
  percentage: 80
},
{
  _id: ObjectId('6978332629cca47864628ca0'),
  id: 1,
  name: 'nivi',
  course: 'cs',
  percentage: 85
},
{
  _id: ObjectId('6978332629cca47864628ca1'),
  id: 2,
  name: 'neena',
```

```
    course: 'cs',
    percentage: 80
  },
  {
    _id: ObjectId('6978332629cca47864628ca2'),
    id: 3,
    name: 'jayasree',
    course: 'it',
    percentage: 92
  },
  {
    _id: ObjectId('6978332629cca47864628ca3'),
    id: 4,
    name: 'harsha',
    course: 'it',
    percentage: 88
  },
{
    _id: ObjectId('6978332629cca47864628ca4'),
    id: 5,
    name: 'meena',
    course: 'bsc',
    percentage: 85
  },
  {
```

```
    _id: ObjectId('6978332629cca47864628ca5'),
    id: 6,
    name: 'pooja',
    course: 'bsc',
    percentage: 82
  }
]
```

2) DISPLAY ALL FEMALE STUDENTS FROM ERODE

```
test> db.students2.find({ address:"erode", gender:"female" })
[
  {
    _id: ObjectId('6978288d7f626d21b2628ca8'),
    id: 3,
    age: '25',
    gender: 'female',
    address: 'erode'
  },
  {
    _id: ObjectId('6978333b29cca47864628ca8'),
    id: 3,
    age: '25',
    gender: 'female',
    address: 'erode'
  }
]
```

3) DISPLAY ALL BSC STUDENTS IN DESCENDING ORDER OF PERCENTAGE

```
test> db.students.find({ course:"bsc" }).sort({ percentage:-1 })
[
  {
    _id: ObjectId('6978332629cca47864628ca4'),
    id: 5,
    name: 'meena',
    course: 'bsc',
    percentage: 85
  },
  {
    _id: ObjectId('6978332629cca47864628ca5'),
    id: 6,
    name: 'pooja',
    course: 'bsc',
    percentage: 82
  }
]
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 7 Date:	CRUD OPERATIONS

AIM

To write Mongodb queries to perform crud operations by inserting user documents, retrieving users whose usernames start with the letter **'a'**, updating the email address of a specific user, and deleting users who have not logged in for more than six months.

ALGORITHM

STEP 1: Start the MongoDB server and open the MongoDB shell using mongosh.

STEP 2: Create or switch to the required database using the use command.

STEP 3: Insert a sample of five user documents into a collection using the insertMany() command.

STEP 4: Retrieve the users whose usernames start with the letter **'a'** using B command with a regular expression.

STEP 5: Update the email address of a specific user using the updateOne() command.

STEP 6: Verify the updated user details using the find() command.

STEP 7: Calculate the date representing six months before the current date.

STEP 8: Identify and delete users who have not logged in for more than six months using the deleteMany() command.

STEP 9: Display the remaining user documents to verify the delete operation.

PROGRAM

```
test> use userDB
switched to db userDB
userDB> db.users.insertMany([
...   {
...     username: "Anitha",
...     email: "anitha@gmail.com",
...     age: 22,
...     lastLogin: new Date("2025-08-10")
...   },
...   {
...     username: "Arun",
...     email: "arun@gmail.com",
...     age: 25,
...     lastLogin: new Date("2025-07-15")
...   },
...   {
...     username: "Bala",
...     email: "bala@gmail.com",
...     age: 24,
...     lastLogin: new Date("2024-10-01")
...   },
...   {
...     username: "Akash",
...     email: "akash@gmail.com",
...     age: 26,
...     lastLogin: new Date("2025-01-05")
...   },
...   {
...     username: "Divya",
...     email: "divya@gmail.com",
...     age: 23,
...     lastLogin: new Date("2024-09-20")
...   }
```

```
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('697f88625658328c2e628ca0'),
    '1': ObjectId('697f88625658328c2e628ca1'),
    '2': ObjectId('697f88625658328c2e628ca2'),
    '3': ObjectId('697f88625658328c2e628ca3'),
    '4': ObjectId('697f88625658328c2e628ca4')
  }
}
userDB> db.users.find({ username: { $regex: "^A" } })
[
  {
    _id: ObjectId('69788ad26d6464a196628ca0'),
    username: 'Anitha',
    email: 'anitha@gmail.com',
    age: 22,
    lastLogin: ISODate('2025-08-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('69788ad26d6464a196628ca4'),
    username: 'Ajay',
    email: 'ajay@gmail.com',
    age: 26,
    lastLogin: ISODate('2025-09-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('6979d36e40e34a1491628ca0'),
    username: 'Anitha',
    email: 'anitha@gmail.com',
    age: 22,
    lastLogin: ISODate('2025-08-10T00:00:00.000Z')
```

```
  },
  {
    _id: ObjectId('697f88625658328c2e628ca0'),
    username: 'Anitha',
    email: 'anitha@gmail.com',
    age: 22,
    lastLogin: ISODate('2025-08-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('697f88625658328c2e628ca1'),
    username: 'Arun',
    email: 'arun@gmail.com',
    age: 25,
    lastLogin: ISODate('2025-07-15T00:00:00.000Z')
  },
  {
    _id: ObjectId('697f88625658328c2e628ca3'),
    username: 'Akash',
    email: 'akash@gmail.com',
    age: 26,
    lastLogin: ISODate('2025-01-05T00:00:00.000Z')
  }
]
userDB> db.users.updateOne(
...   { username: "Arun" },
...   { $set: { email: "arun_new@gmail.com" } }
... )
...
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
```

```
}
userDB> db.users.find({ username: "Arun" })
[
  {
    _id: ObjectId('697f88625658328c2e628ca1'),
    username: 'Arun',
    email: 'arun_new@gmail.com',
    age: 25,
    lastLogin: ISODate('2025-07-15T00:00:00.000Z')
  }
]
userDB> var sixMonthsAgo = new Date()
... sixMonthsAgo.setMonth(sixMonthsAgo.getMonth() - 6)
...
1754068102302
userDB> db.users.deleteMany({
...   lastLogin: { $lt: sixMonthsAgo }
... })
...
{ acknowledged: true, deletedCount: 4 }
userDB> db.users.find()
[
  {
    _id: ObjectId('69788ad26d6464a196628ca0'),
    username: 'Anitha',
    email: 'anitha@gmail.com',
    age: 22,
    lastLogin: ISODate('2025-08-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('69788ad26d6464a196628ca4'),
    username: 'Ajay',
    email: 'ajay@gmail.com',
    age: 26,
```

```
    lastLogin: ISODate('2025-09-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('6979d36e40e34a1491628ca0'),
    username: 'Anitha',
    email: 'anitha@gmail.com',
    age: 22,
    lastLogin: ISODate('2025-08-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('697f88625658328c2e628ca0'),
    username: 'Anitha',
    email: 'anitha@gmail.com',
    age: 22,
    lastLogin: ISODate('2025-08-10T00:00:00.000Z')
  }
]
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 8 Date:	QUERYING AND FILTERING

AIM

To perform querying and filtering operations in MongoDB to retrieve specific documents based on range, category, date, and sorting conditions.

ALGORITHM

STEP 1: Start the MongoDB server and open the MongoDB shell.

STEP 2: Create and switch to the required database using the use storeDB command.

STEP 3: Create a collection named products in the database.

STEP 4: Insert multiple product documents into the products collection with fields such as name, category, price, and creationDate.

STEP 5: Retrieve products belonging to the Electronics category whose price ranges between 50 and 100 using comparison operators.

STEP 6: Retrieve products created within the specified date range and sort the documents by the creationDate field in ascending order.

STEP 7: Display the output and verify the retrieved results.

PROGRAM

```
test> use storeDB
switched to db storeDB
storeDB> db.createCollection("products")
{ ok: 1 }
storeDB> db.products.insertMany([
...  { name: "Laptop", category: "Electronics", price: 800, creationDatcreationDate:
ISODate("2024-01-10") },
...  { name: "Smartphone", category: "Electronics", price: 500, creatiocreationDate:
ISODate("2024-01-12") },
...  { name: "HeadPhones", category: "Electronics", price: 100, creatiocreationDate:
ISODate("2024-03-15") },
...  { name: "T-Shirt", category: "Clothing", price: 20, creationDate: ISODate("2024-03-20")
},
...  { name: "Monitor", category: "Electronics", price: 200, creationDacreationDate:
ISODate("2024-03-25") },
...  { name: "Mouse", category: "Electronics", price: 30, creationDate: ISODate("2024-03-
28") },
...  { name: "Keyboard", category: "Electronics", price: 450, creationDcreationDate:
ISODate("2024-03-30") }
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('698019102f3d026bcf628ca0'),
    '1': ObjectId('698019102f3d026bcf628ca1'),
    '2': ObjectId('698019102f3d026bcf628ca2'),
    '3': ObjectId('698019102f3d026bcf628ca3'),
    '4': ObjectId('698019102f3d026bcf628ca4'),
    '5': ObjectId('698019102f3d026bcf628ca5'),
    '6': ObjectId('698019102f3d026bcf628ca6')
  }
}
```

```
storeDB> db.products.find({
...   category: "Electronics",
...   price: { $gte: 50, $lte: 100 }
... })
[
  {
    _id: ObjectId('698019102f3d026bcf628ca2'),
    name: 'HeadPhones',
    category: 'Electronics',
    price: 100,
    creationDate: ISODate('2024-03-15T00:00:00.000Z')
  }
]
storeDB> db.products.find({
...   creationDate: {
...     $gte: ISODate("2024-03-01"),
...     $lte: ISODate("2024-03-31")
...   }
... }).sort({ creationDate: 1 })
[
  {
    _id: ObjectId('698019102f3d026bcf628ca2'),
    name: 'HeadPhones',
    category: 'Electronics',
    price: 100,
    creationDate: ISODate('2024-03-15T00:00:00.000Z')
  },
  {
    _id: ObjectId('698019102f3d026bcf628ca3'),
    name: 'T-Shirt',
    category: 'Clothing',
    price: 20,
    creationDate: ISODate('2024-03-20T00:00:00.000Z')
  },
```

```
  {
    _id: ObjectId('698019102f3d026bcf628ca4'),
    name: 'Monitor',
    category: 'Electronics',
    price: 200,
    creationDate: ISODate('2024-03-25T00:00:00.000Z')
  },
  {
    _id: ObjectId('698019102f3d026bcf628ca5'),
    name: 'Mouse',
    category: 'Electronics',
    price: 30,
    creationDate: ISODate('2024-03-28T00:00:00.000Z')
  },
  {
    _id: ObjectId('698019102f3d026bcf628ca6'),
    name: 'Keyboard',
    category: 'Electronics',
    price: 450,
    creationDate: ISODate('2024-03-30T00:00:00.000Z')
  }
]
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 9 Date:	MONGODB TO PERFORM AGGREGATION

AIM

To write queries in MongoDB to perform aggregation.

ALGORITHM

STEP 1: Create products collection.

STEP 2: Insert product documents with name and customerReviews array.

STEP 3: Unwind customerReviews array.

STEP 4: Group by product _id and calculate averageRating.

STEP 5: Project product name and count of customerReviews as numComments.

STEP 6: Sort products by numComments in descending order.

STEP 7: Limit the output to top 5 products.

STEP 8: End the process.

PROGRAM

```
test> db.createCollection("products");
{ ok: 1 }
test> db.products.insertMany([
...   {
...     name: "Product A",
...     customerReviews: [
...       { rating: 4, comment: "Great product!" },
...       { rating: 5, comment: "Excellent quality." },
...       { rating: 3, comment: "Could be better." }
...     ]
...   },
...   {
...     name: "Product B",
...     customerReviews: [
...       { rating: 2, comment: "Disappointing." },
...       { rating: 4, comment: "Good value for money." },
...       { rating: 5, comment: "Love it!" },
...       { rating: 3, comment: "Average product." }
...     ]
...   },
...   {
...     name: "Product C",
...     customerReviews: [
...       { rating: 5, comment: "Amazing product!" },
...       { rating: 5, comment: "Highly recommended." },
...       { rating: 4, comment: "Satisfied with the purchase." },
...       { rating: 5, comment: "Beyond expectations!" },
...       { rating: 4, comment: "Good features." },
...       { rating: 4, comment: "Nice design." }
...     ]
...   }
... ]);
{
```

```
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('69802826e867a42b6e628ca0'),
    '1': ObjectId('69802826e867a42b6e628ca1'),
    '2': ObjectId('69802826e867a42b6e628ca2')
  }
}
test> db.products.aggregate([
...   { $unwind: "$customerReviews" }, // Flatten the customerReviews array
...   {
...     $group: {
...       _id: "$_id",
...       name: { $first: "$name" },              // Keep product name
...       averageRating: { $avg: "$customerReviews.rating" } // Calculate average rating
...     }
...   }
... ]);
[
  {
    _id: ObjectId('69802826e867a42b6e628ca1'),
    name: 'Product B',
    averageRating: 3.5
  },
  {
    _id: ObjectId('69802826e867a42b6e628ca0'),
    name: 'Product A',
    averageRating: 4
  },
  {
    _id: ObjectId('69802826e867a42b6e628ca2'),
    name: 'Product C',
    averageRating: 4.5
  }
]
```

```
test> db.products.aggregate([
...   {
...     $project: {
...       _id: 1,
...       name: 1,
...       numComments: { $size: { $ifNull: ["$customerReviews", []] } } // Count comments
safely
...     }
...   },
...   { $sort: { numComments: -1 } },
...   { $limit: 5 }
... ]);
...
[
  {
    _id: ObjectId('69802826e867a42b6e628ca2'),
    name: 'Product C',
    numComments: 6
  },
  {
    _id: ObjectId('69802826e867a42b6e628ca1'),
    name: 'Product B',
    numComments: 4
  },
  {
    _id: ObjectId('69802826e867a42b6e628ca0'),
    name: 'Product A',
    numComments: 3
  }
]
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 10 Date:	MONGODB BACKUP AND RESTORE

AIM

To create database backup and restore using mangodb

ALGORITHM

STEP 1: Open MongoDB shell and create a new database named studentDB.

STEP 2: Create a students collection and insert sample student records.

STEP 3: Verify that the data is correctly inserted in the collection.

STEP 4: Install MongoTools, Open terminal/command prompt and use mongodump to create a backup of studentDB in a desired folder.

STEP 5: Drop the students collection or the entire studentDB database to simulate deletion.

STEP 6: Verify that the collection or database has been removed.

STEP 7: Use mongorestore from terminal/command prompt to restore the studentDB from the backup folder.

STEP 8: Open MongoDB shell and verify that the students collection and data have been restored successfully.

STEP 9: End. Database is now restored and ready for use.

PROGRAM

1) OPEN MONGOSH

```
test> use studentDB
switched to db studentDB
studentDB> db.students.insertMany([
...   { studentId: 1, name: "Alice", age: 20, course: "BSc CS", email: "alice@gmail.com" },
...   { studentId: 2, name: "Bob", age: 22, course: "BCom", email: "bob@gmail.com" },
...   { studentId: 3, name: "Charlie", age: 21, course: "BA English", email:
"charlie@gmail.com" },
...   { studentId: 4, name: "David", age: 23, course: "BSc Maths", email: "david@gmail.com"
}
... ]);
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6980298b1bf0d399bf628ca0'),
    '1': ObjectId('6980298b1bf0d399bf628ca1'),
    '2': ObjectId('6980298b1bf0d399bf628ca2'),
    '3': ObjectId('6980298b1bf0d399bf628ca3')
  }
}
studentDB> db.students.find().pretty()
[
  {
    _id: ObjectId('695d305030c4fd6f761e2621'),
    rollNo: 1,
    name: 'Arun',
    age: 20,
    course: 'BCA'
  },
  {
    _id: ObjectId('695d305030c4fd6f761e2622'),
    rollNo: 2,
```

```
  name: 'Divya',
  age: 21,
  course: 'BSc'
},
{
  _id: ObjectId('695d305030c4fd6f761e2623'),
  rollNo: 3,
  name: 'Kumar',
  age: 22,
  course: 'BCom'
},
{
  _id: ObjectId('6980298b1bf0d399bf628ca0'),
  studentId: 1,
  name: 'Alice',
  age: 20,
  course: 'BSc CS',
  email: 'alice@gmail.com'
},
{
  _id: ObjectId('6980298b1bf0d399bf628ca1'),
  studentId: 2,
  name: 'Bob',
  age: 22,
  course: 'BCom',
  email: 'bob@gmail.com'
},
{
  _id: ObjectId('6980298b1bf0d399bf628ca2'),
  studentId: 3,
  name: 'Charlie',
  age: 21,
  course: 'BA English',
  email: 'charlie@gmail.com'
```

```
  },
  {
    _id: ObjectId('6980298b1bf0d399bf628ca3'),
    studentId: 4,
    name: 'David',
    age: 23,
    course: 'BSc Maths',
    email: 'david@gmail.com'
  }
]
```

2) OPEN MONGODUMP IN MONGO TOOLS

```
mongodump --db studentDB --out C:\backup\student_backup
```

3) DROP TABLE

```
studentDB>  db.dropDatabase()
{ ok: 1, dropped: 'studentDB' }
```

4) OPEN RESTORE IN MONGO TOOLS

```
mongorestore --db studentDB C:\backup\student_backup\studentDB
```

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 11 Date:	CREATING NODE AND RELATIONSHIP IN NEO4J

AIM

To create a node and relationship in Neo4j

ALGORITHM

STEP 1: Start the process

STEP 2: Create two nodes representing users with properties like name and email

STEP 3: Establish a friendship relationship by FRIEND_OF function

STEP 4: Create nodes representing different cities with properties name and population

STEP 5: Connect the user to the cities with a relationship type indicate the current

STEP 6: Stop the process

PROGRAM

CREATE (:User {name: 'Alice', email: 'alice@example.com'});

CREATE (:User {name: 'Bob', email: 'bob@example.com'});

MATCH (n) RETURN n;

MATCH (a:User {name: 'Alice'}), (b:User {name: 'Bob'}

CREATE (a)-[:FRIENDS_OF]->(b);

MATCH (u:User {name: 'Bob'}), (c:City {name: 'San Francisco'})

CREATE (u)-[:LIVES_IN]->(c);

CREATE (:City {name: 'New York', population: 8419000});

CREATE (:City {name: 'San Francisco', population: 883305});

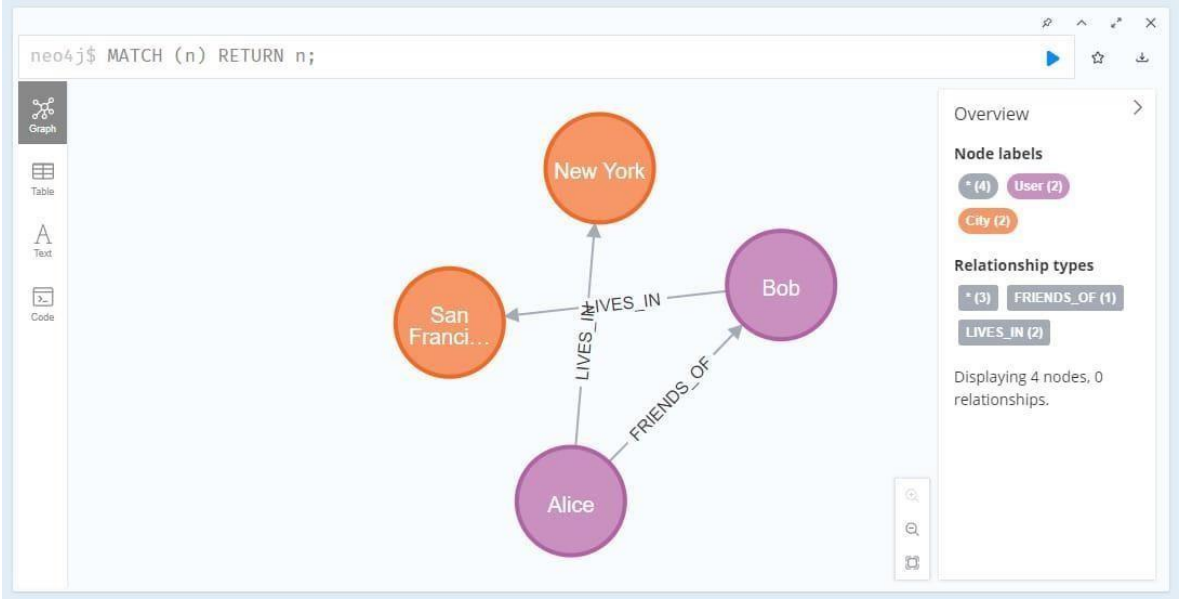
MATCH (u:User {name: 'Alice'}), (c:City {name: 'New York'})

CREATE (u)-[:LIVES_IN]->(c);

MATCH (u:User {name: 'Bob'}), (c:City {name: 'San Francisco'})

CREATE (u)-[:LIVES_IN]->(c);

MATCH (n) RETURN n;



RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 12 Date:	NEO4J FOR RETRIEVING DATA

AIM

To perform basic cipher queries in Neo4j

ALGORITHM

STEP 1: Start the process

STEP 2: Create the nodes

STEP 3: Match the nodes with desired label to retrieve all users

STEP 4: Return specific properties (name and email address) of users

STEP 5: Match the friendship relationships (FRIEND_OF) between users to identify friend

STEP 6: Return the name of the users who are friends

STEP 7: Match the users who live in a specific city by using MATCH clause and specify city name in WHERE clause

STEP 8: Return name and email address of users who live in specific city

STEP 9: Stop the process

PROGRAM

CREATE (:User {name: "Alice", email: "alice@gmail.com", city: "New York"});

CREATE (:User {name: "Bob", email: "bob@gmail.com", city: "London"});

MATCH (a:User {name: "Alice"}), (b:User {name: "Bob"})

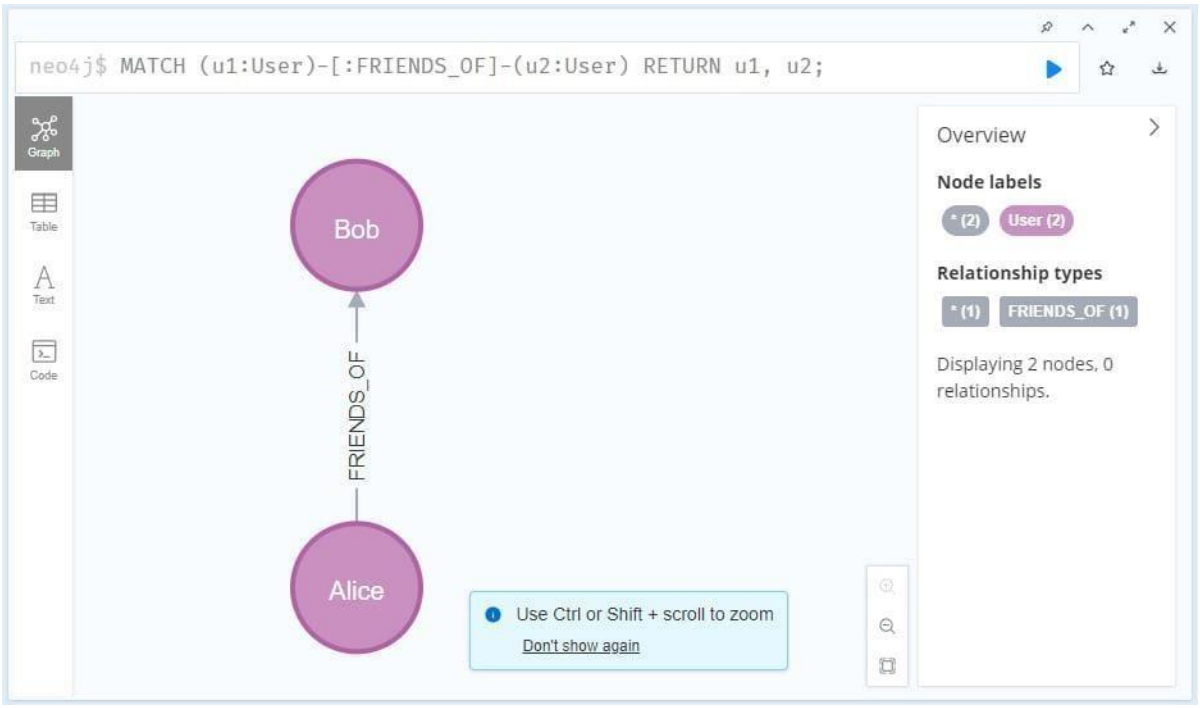
CREATE (a)-[:FRIENDS_OF]->(b);

MATCH (u:User)

RETURN u.name, u.email;

MATCH (u1:User)-[:FRIENDS_OF]-(u2:User)

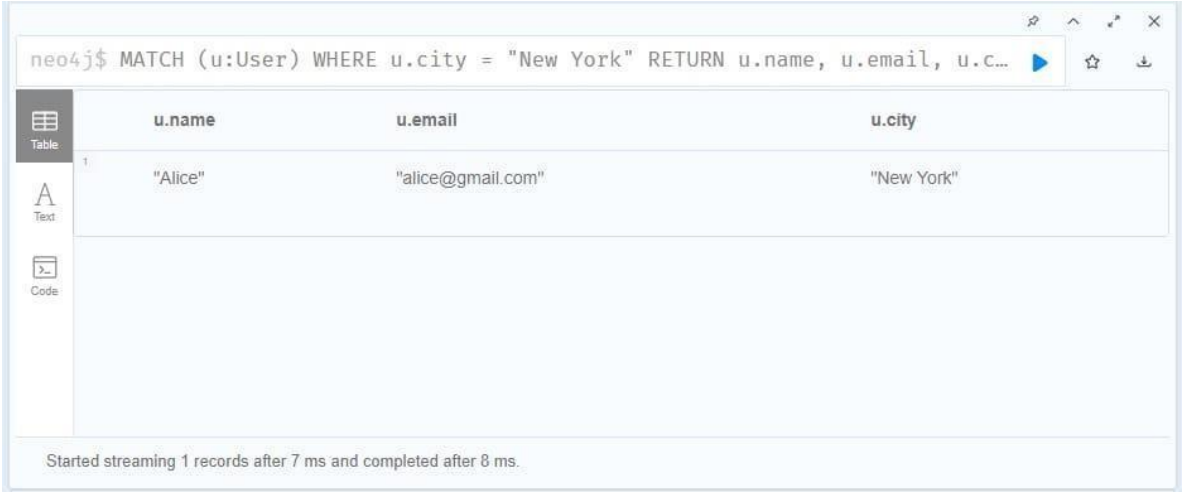
RETURN u1.name AS User1, u2.name AS User2;



MATCH (u:User)

WHERE u.city = "New York"

RETURN u.name, u.email, u.city;



RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 1 Date:	BANKING MANAGEMENT

AIM

To Develop an application for Banking Management system.

ALGORITHM

STEP 1: Start the program.

STEP 2: Initialize an empty data structure to store account details such as account number, account holder name, and balance.

STEP 3: Create the GUI window using Tkinter, react js and add labels, text fields, and buttons for user input.

STEP 4: Accept account details from the user and create a new account with an initial balance when the create account button is clicked.

STEP 5: Accept deposit amount and add it to the corresponding account balance when the deposit button is clicked.

STEP 6: Accept withdrawal amount and subtract it from the account balance after checking for sufficient balance when the withdraw button is clicked.

STEP 7: Display the current balance of the account when the check balance button is clicked.

STEP 8: Repeat the process until the user closes the application, then stop the program.

PROGRAM

```
import tkinter as tk
from tkinter import messagebox
import json
import os

# ------------------ File Setup ------------------ #

FILE_NAME = &quot;accounts.json&quot;

if not os.path.exists(FILE_NAME):
with open(FILE_NAME, &quot;w&quot;) as file:
json.dump([], file)

# ------------------ Helper Functions ------------------ #

def load_accounts():
with open(FILE_NAME, &quot;r&quot;) as file:
return json.load(file)

def save_accounts(accounts):
with open(FILE_NAME, &quot;w&quot;) as file:
json.dump(accounts, file, indent=4)

# ------------------ Banking Functions ------------------ #

def create_account():
name = entry_name.get()
acc_no = entry_acc.get()
balance = entry_balance.get()

if name == &quot;&quot; or acc_no == &quot;&quot; or balance == &quot;&quot;:
messagebox.showerror(&quot;Error&quot;, &quot;All fields are required&quot;)
return

if not balance.isdigit():
messagebox.showerror(&quot;Error&quot;, &quot;Balance must be a number&quot;)
return

accounts = load_accounts()
```

```
for acc in accounts:
if acc[&quot;account_no&quot;] == acc_no:
messagebox.showerror(&quot;Error&quot;, &quot;Account number already exists&quot;)
return

account = {
&quot;name&quot;: name,
&quot;account_no&quot;: acc_no,
&quot;balance&quot;: int(balance)
}

accounts.append(account)
save_accounts(accounts)

messagebox.showinfo(&quot;Success&quot;, &quot;Account created successfully&quot;)
clear_fields()

def deposit_money():
acc_no = entry_acc.get()
amount = entry_amount.get()

if acc_no == &quot;&quot; or amount == &quot;&quot;:
messagebox.showerror(&quot;Error&quot;, &quot;Account number and amount required&quot;)
return

if not amount.isdigit():
messagebox.showerror(&quot;Error&quot;, &quot;Amount must be a number&quot;)
return

accounts = load_accounts()

for acc in accounts:
if acc[&quot;account_no&quot;] == acc_no:
acc[&quot;balance&quot;] += int(amount)
save_accounts(accounts)
messagebox.showinfo(&quot;Success&quot;, &quot;Amount deposited successfully&quot;)
return

messagebox.showerror(&quot;Error&quot;, &quot;Account not found&quot;)

def withdraw_money():
acc_no = entry_acc.get()
```

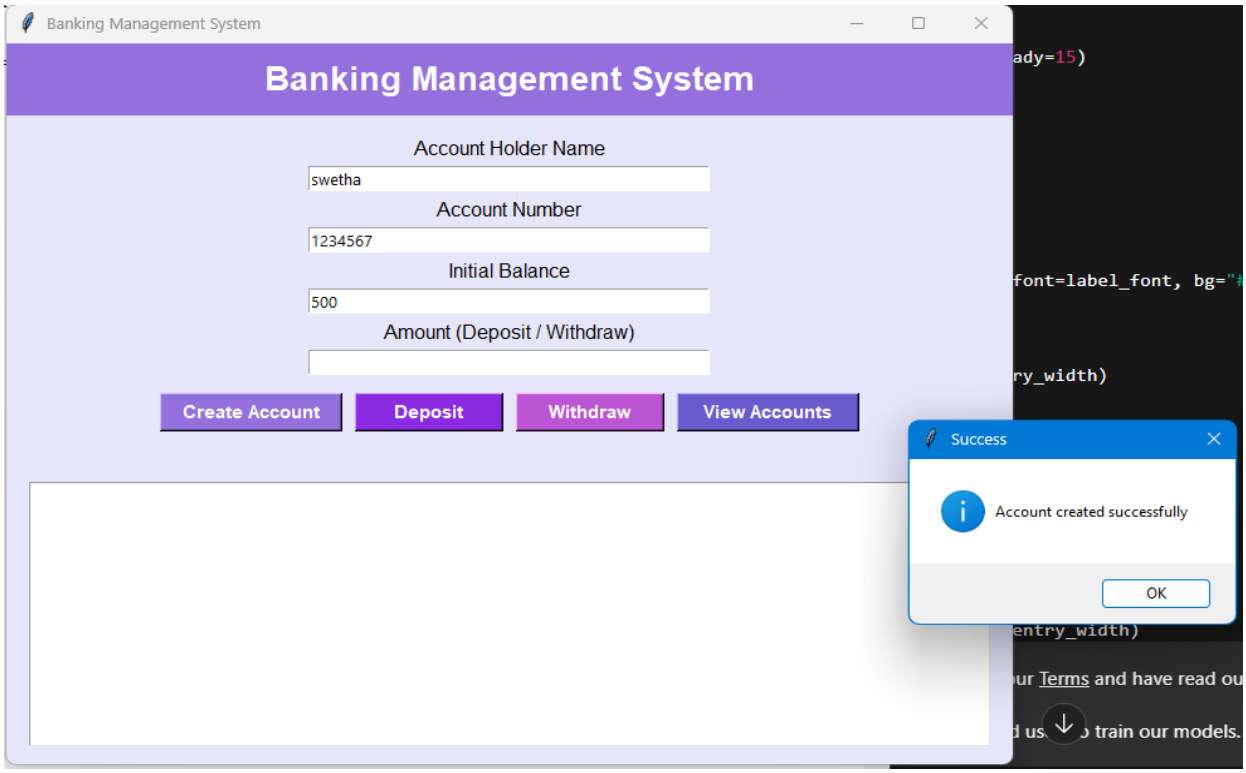
```
amount = entry_amount.get()

if acc_no == &quot;&quot; or amount == &quot;&quot;:

messagebox.showerror(&quot;Error&quot;, &quot;Account number and amount required&quot;)
return

if not amount.isdigit():
messagebox.showerror(&quot;Error&quot;, &quot;Amount must be a number&quot;)
return

accounts = load_accounts()

for acc in accounts:
if acc[&quot;account_no&quot;] == acc_no:
if acc[&quot;balance&quot;] &lt; int(amount):
messagebox.showerror(&quot;Error&quot;, &quot;Insufficient balance&quot;)
return
acc[&quot;balance&quot;] -= int(amount)
save_accounts(accounts)
messagebox.showinfo(&quot;Success&quot;, &quot;Amount withdrawn successfully&quot;)
return

messagebox.showerror(&quot;Error&quot;, &quot;Account not found&quot;)

def view_accounts():
text_display.delete(&quot;1.0&quot;,  tk.END)
accounts = load_accounts()

if not accounts:
text_display.insert(tk.END, &quot;No accounts available.\n&quot;)
return

for acc in accounts:
text_display.insert(
tk.END,
f&quot;Name: {acc[&#39;name&#39;]} | Account No: {acc[&#39;account_no&#39;]} | &quot;
f&quot;Balance: â‚¹{acc[&#39;balance&#39;]}\n&quot;
&quot;------------------------------------------------------------\n&quot;
)

def clear_fields():
```

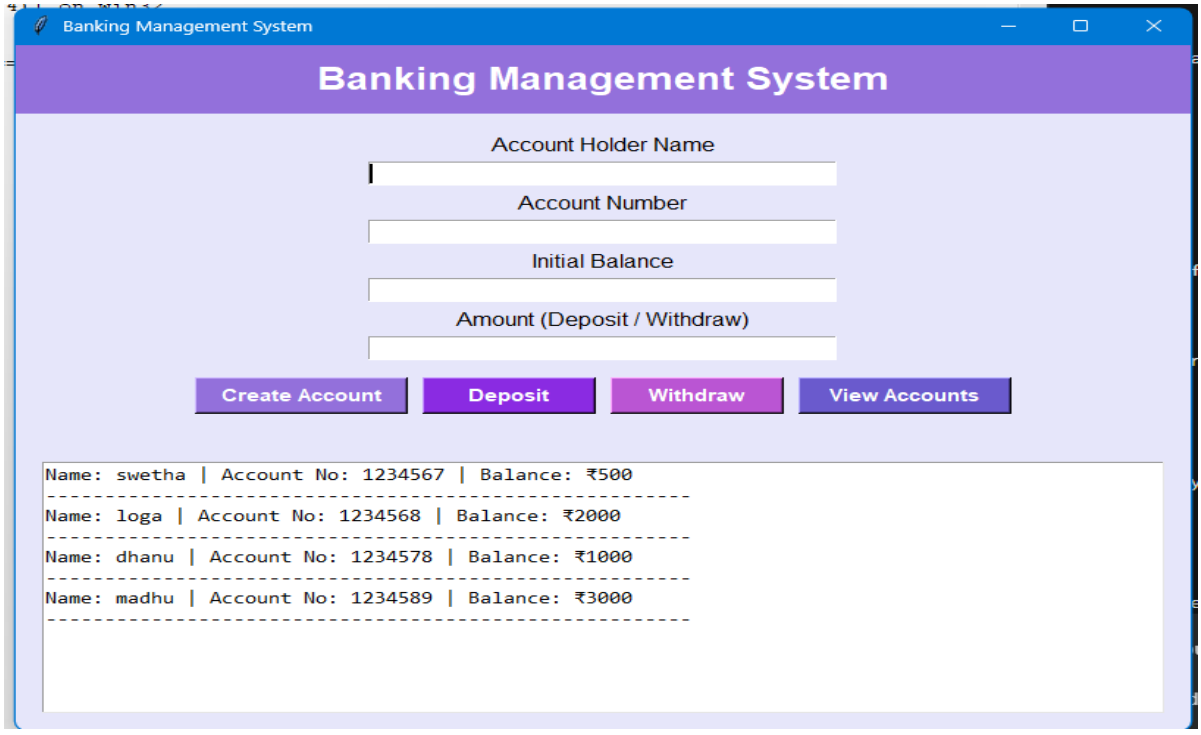
```
entry_name.delete(0, tk.END)
entry_acc.delete(0, tk.END)
entry_balance.delete(0, tk.END)
entry_amount.delete(0, tk.END)

# ---------- UI ---------- #

root = tk.Tk()
root.title(&quot;Banking Management System&quot;)
root.geometry(&quot;800x600&quot;)
root.configure(bg=&quot;#E6E6FA&quot;) # Lavender background

# Header
header = tk.Label(
root,
text=&quot;Banking Management System&quot;,
font=(&quot;Arial&quot;, 18, &quot;bold&quot;),
bg=&quot;#9370DB&quot;, # Dark lavender

fg=&quot;white&quot;,
pady=12
)
header.pack(fill=&quot;x&quot;)

# Main Frame
main_frame = tk.Frame(root, bg=&quot;#E6E6FA&quot;, pady=15)
main_frame.pack()

label_font = (&quot;Arial&quot;, 11)
entry_width = 45

def label(text):
return tk.Label(main_frame, text=text, font=label_font, bg=&quot;#E6E6FA&quot;)

label(&quot;Account Holder Name&quot;).pack()
entry_name = tk.Entry(main_frame, width=entry_width)
entry_name.pack(pady=3)

label(&quot;Account Number&quot;).pack()
entry_acc = tk.Entry(main_frame, width=entry_width)
entry_acc.pack(pady=3)
```

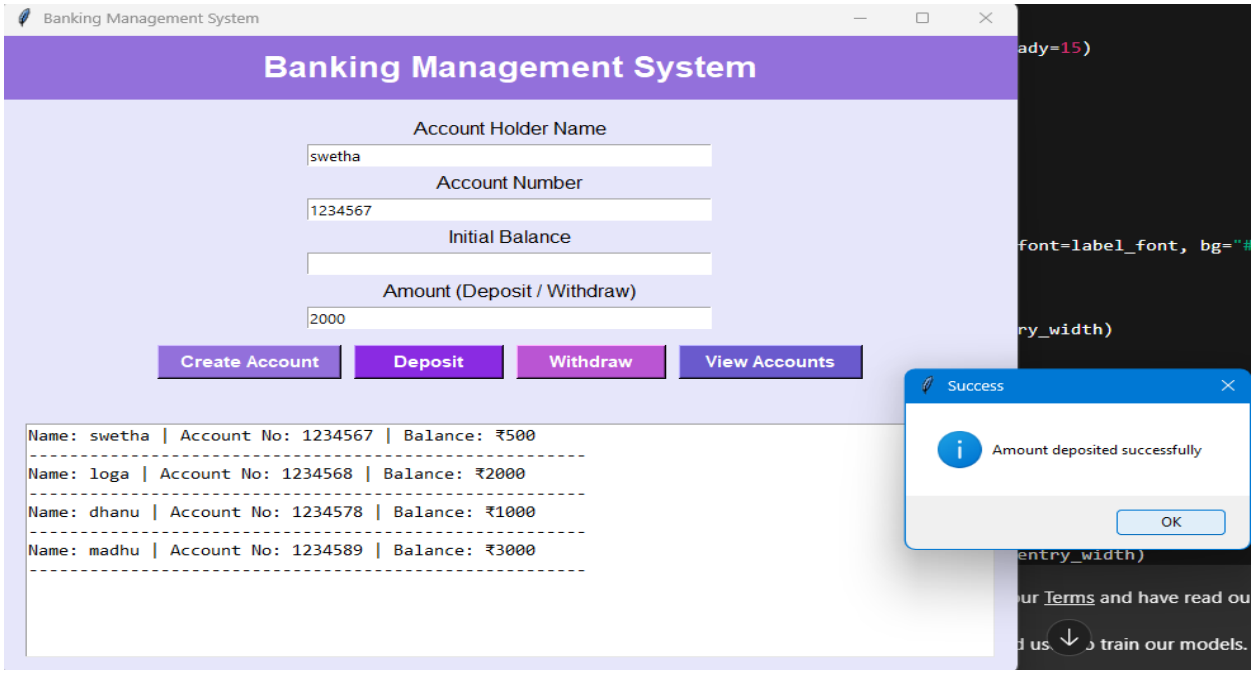
```
label(&amp;quot;Initial Balance&amp;quot;).pack()
entry_balance = tk.Entry(main_frame, width=entry_width)
entry_balance.pack(pady=3)

label(&amp;quot;Amount (Deposit / Withdraw)&amp;quot;).pack()

entry_amount = tk.Entry(main_frame, width=entry_width)
entry_amount.pack(pady=3)

# Buttons
btn_frame = tk.Frame(main_frame, bg=&amp;quot;#E6E6FA&amp;quot;)
btn_frame.pack(pady=12)

tk.Button(
btn_frame,
text=&amp;quot;Create Account&amp;quot;,
command=create_account,
bg=&amp;quot;#9370DB&amp;quot;,
fg=&amp;quot;white&amp;quot;,
font=(&amp;quot;Arial&amp;quot;, 10, &amp;quot;bold&amp;quot;),
width=15
).grid(row=0, column=0, padx=5)

tk.Button(
btn_frame,
text=&amp;quot;Deposit&amp;quot;,
command=deposit_money,
bg=&amp;quot;#8A2BE2&amp;quot;,
fg=&amp;quot;white&amp;quot;,
font=(&amp;quot;Arial&amp;quot;, 10, &amp;quot;bold&amp;quot;),
width=12
).grid(row=0, column=1, padx=5)

tk.Button(

btn_frame,
text=&amp;quot;Withdraw&amp;quot;,
command=withdraw_money,
bg=&amp;quot;#BA55D3&amp;quot;,
fg=&amp;quot;white&amp;quot;,
font=(&amp;quot;Arial&amp;quot;, 10, &amp;quot;bold&amp;quot;),
width=12
```

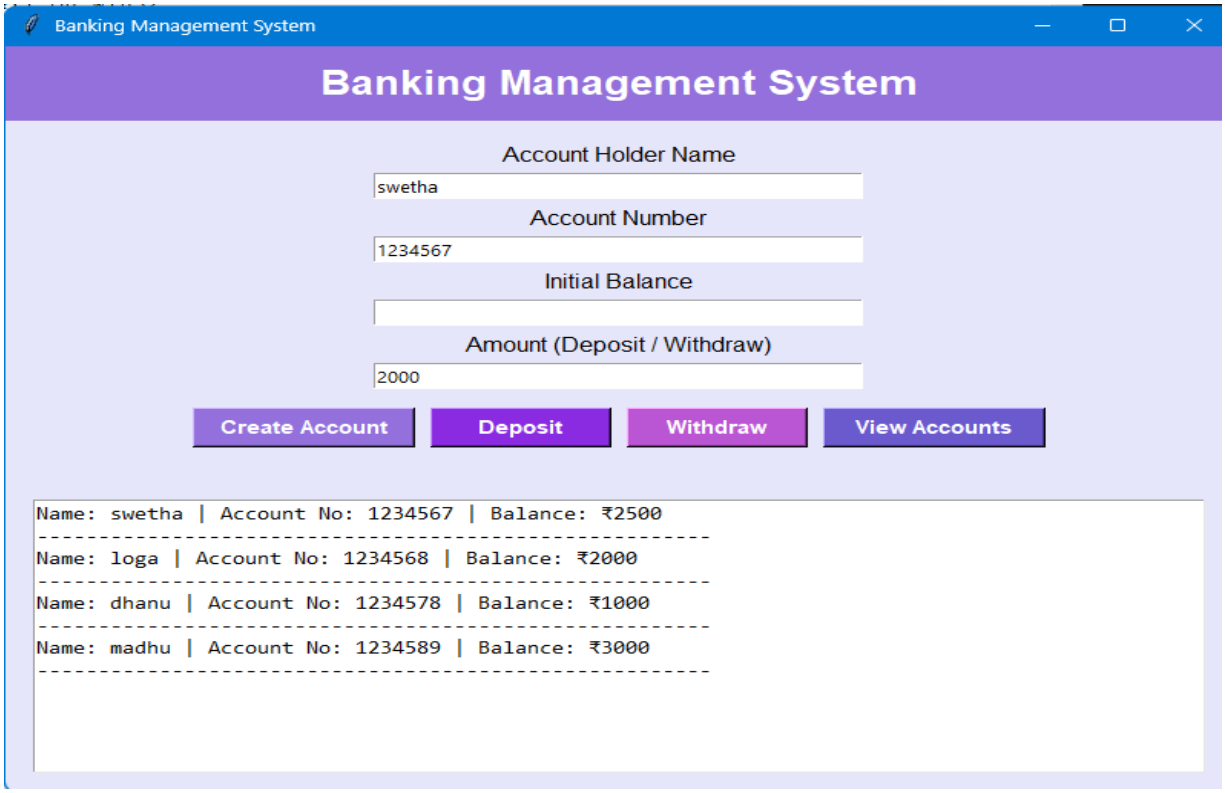
```
).grid(row=0, column=2, padx=5)

tk.Button(
btn_frame,
text=&quot;View Accounts&quot;,
command=view_accounts,
bg=&quot;#6A5ACD&quot;,
fg=&quot;white&quot;,
font=(&quot;Arial&quot;, 10, &quot;bold&quot;),
width=15
).grid(row=0, column=3, padx=5)

# Display Area
text_display = tk.Text(
root,
height=14,
width=95,
font=(&quot;Consolas&quot;, 10),
bg=&quot;white&quot;
)
text_display.pack(pady=15)

root.mainloop()
```

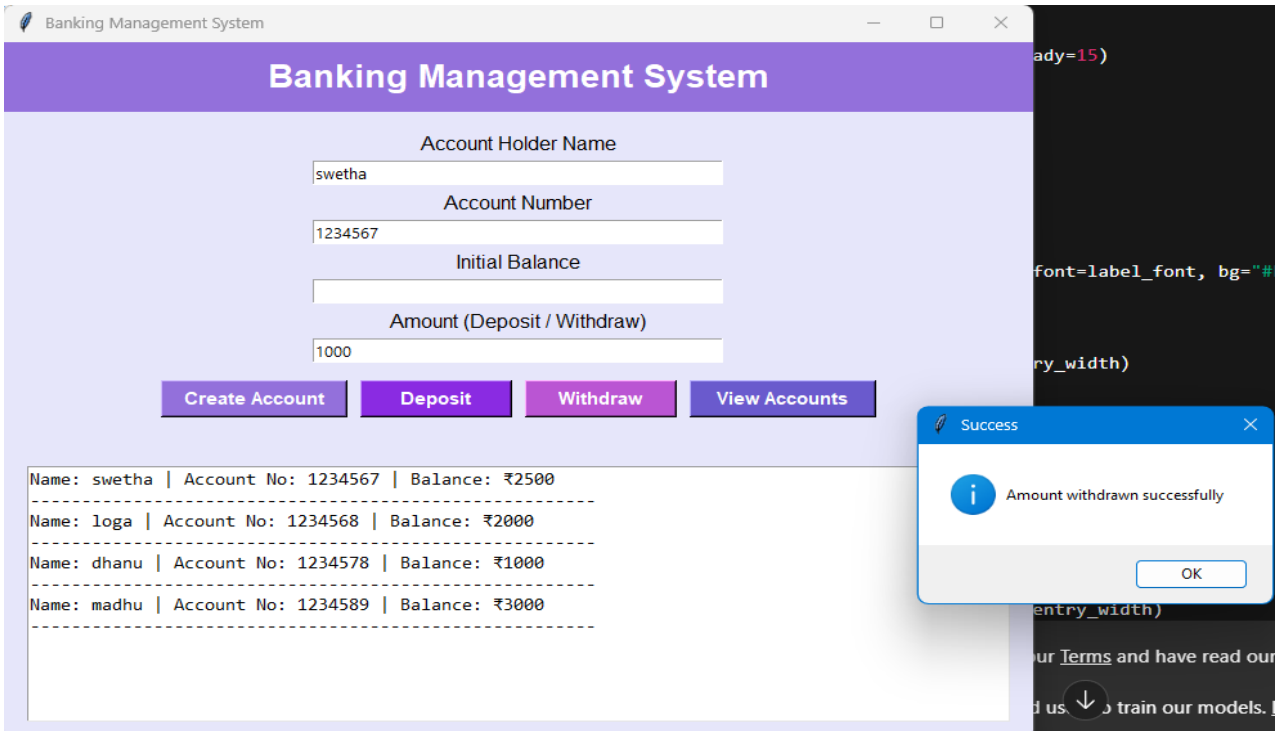
OUTPUT

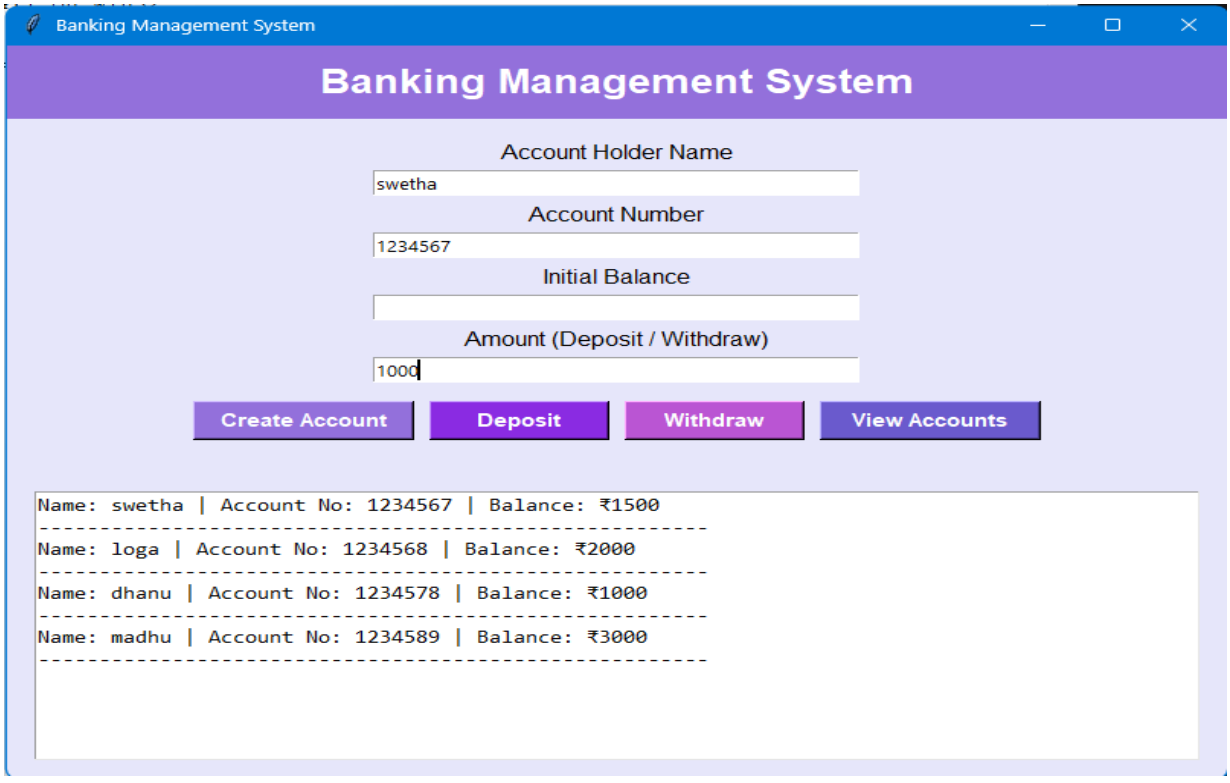

Banking Management System
Banking Management System
Account Holder Name
swetha
Account Number
1234567
Initial Balance
500
Amount (Deposit / Withdraw)
Create Account
Deposit
Withdraw
View Accounts
Success
Account created successfully
OK


Banking Management System
Banking Management System
Account Holder Name
Account Number
Initial Balance
Amount (Deposit / Withdraw)
Create Account
Deposit
Withdraw
View Accounts
Name: swetha | Account No: 1234567 | Balance: ₹500
Name: loga | Account No: 1234568 | Balance: ₹2000
Name: dhanu | Account No: 1234578 | Balance: ₹1000
Name: madhu | Account No: 1234589 | Balance: ₹3000


Banking Management System
Banking Management System
Account Holder Name
swetha
Account Number
1234567
Initial Balance
Amount (Deposit / Withdraw)
2000
Create Account
Deposit
Withdraw
View Accounts
Name: swetha | Account No: 1234567 | Balance: ₹500
Name: loga | Account No: 1234568 | Balance: ₹2000
Name: dhanu | Account No: 1234578 | Balance: ₹1000
Name: madhu | Account No: 1234589 | Balance: ₹3000
Success
Amount deposited successfully
OK
ady=15)
font=label_font, bg="#
ry_width)
entry_width)
ur Terms and have read ou
train our models.


Banking Management System
Banking Management System
Account Holder Name
swetha
Account Number
1234567
Initial Balance
Amount (Deposit / Withdraw)
2000
Create Account
Deposit
Withdraw
View Accounts
Name: swetha | Account No: 1234567 | Balance: ₹2500
Name: loga | Account No: 1234568 | Balance: ₹2000
Name: dhanu | Account No: 1234578 | Balance: ₹1000
Name: madhu | Account No: 1234589 | Balance: ₹3000


Banking Management System
Banking Management System
Account Holder Name
swetha
Account Number
1234567
Initial Balance
Amount (Deposit / Withdraw)
1000
Create Account
Deposit
Withdraw
View Accounts
Name: swetha | Account No: 1234567 | Balance: ₹2500
Name: loga | Account No: 1234568 | Balance: ₹2000
Name: dhanu | Account No: 1234578 | Balance: ₹1000
Name: madhu | Account No: 1234589 | Balance: ₹3000
Success
Amount withdrawn successfully
OK


Banking Management System
Banking Management System
Account Holder Name
swetha
Account Number
1234567
Initial Balance
Amount (Deposit / Withdraw)
1000
Create Account
Deposit
Withdraw
View Accounts
Name: swetha | Account No: 1234567 | Balance: ₹1500
Name: loga | Account No: 1234568 | Balance: ₹2000
Name: dhanu | Account No: 1234578 | Balance: ₹1000
Name: madhu | Account No: 1234589 | Balance: ₹3000

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 2 Date:	INVENTORY SYSTEM FOR SUPERMARKET

AIM

To design and develop a Supermarket Application using React JS as frontend and Oracle Database (SQL Plus) as backend, and to insert records into the CART table by implementing the Add to Cart functionality.

ALGORITHM

STEP 1: Start the Oracle Database and open **SQL Plus**.

STEP 2: Login using valid username and password.

STEP 3: Create the required database tables namely **PRODUCTS** and **CART**.

STEP 4: Insert sample product records into the **PRODUCTS** table.

STEP 5: Verify the inserted records using the SELECT command.

STEP 6: Create a backend folder and initialize a Node.js project.

STEP 7: Create and configure server.js to establish connection with the Oracle database.

STEP 8: Run the backend server using Node.js.

STEP 9: Create a React frontend application.

STEP 10: Design components to display product records on the screen using Visual Studio Code.

STEP 11: Provide **Add to Cart** button for each product and implement the Add to Cart functionality to display the added product.

STEP 12: Stop the execution.

PROGRAM

```
CREATE TABLE products (
  id NUMBER PRIMARY KEY,
  name VARCHAR2(100),
  price NUMBER(10,2),
  image VARCHAR2(255),
  category VARCHAR2(50)
);
INSERT INTO products (id, name, price, image, category)
VALUES (1, &amp;#39;Apple&amp;#39;, 120, &amp;#39;apple.jpg&amp;#39;,
    &amp;#39;Fruits&amp;#39;);

INSERT INTO products (id, name, price, image, category)
VALUES (2, &amp;#39;Orange&amp;#39;, 80, &amp;#39;orange.jpg&amp;#39;,
    &amp;#39;Fruits&amp;#39;);

INSERT INTO products (id, name, price, image, category)
VALUES (3, &amp;#39;Carrot&amp;#39;, 50, &amp;#39;carrot.jpg&amp;#39;,
    &amp;#39;Vegetables&amp;#39;);

INSERT INTO products (id, name, price, image, category)
VALUES (4, &amp;#39;Potato&amp;#39;, 40, &amp;#39;potato.jpg&amp;#39;,
    &amp;#39;Vegetables&amp;#39;);

INSERT INTO products (id, name, price, image, category)
VALUES (5, &amp;#39;Onion&amp;#39;, 45, &amp;#39;onion.jpg&amp;#39;,
    &amp;#39;Vegetables&amp;#39;);

INSERT INTO products (id, name, price, image, category)
VALUES (6, &amp;#39;Milk&amp;#39;, 30, &amp;#39;milk.jpg&amp;#39;,
    &amp;#39;Dairy&amp;#39;);

INSERT INTO products (id, name, price, image, category)
VALUES (7, &amp;#39;Bread&amp;#39;, 35, &amp;#39;bread.jpg&amp;#39;,
    &amp;#39;Bakery&amp;#39;);

INSERT INTO products (id, name, price, image, category)
VALUES (8, &amp;#39;Eggs&amp;#39;, 60, &amp;#39;eggs.jpg&amp;#39;,
    &amp;#39;Poultry&amp;#39;);

INSERT INTO products (id, name, price, image, category)
VALUES (9, &amp;#39;Rice&amp;#39;, 65, &amp;#39;rice.jpg&amp;#39;,
    &amp;#39;Grains&amp;#39;);
```

```
INSERT INTO products (id, name, price, image, category)
VALUES (10, &amp;#39;Sugar&amp;#39;, 50, &amp;#39;sugar.jpg&amp;#39;,
    &amp;#39;Grocery&amp;#39;);
SET LINESIZE 3000
SET PAGESIZE 50

COLUMN id FORMAT 99
COLUMN name FORMAT A15
COLUMN price FORMAT 999
COLUMN image FORMAT A20
COLUMN category FORMAT A15

SELECT * FROM products;
CART TABLE
CREATE TABLE cart (
  id NUMBER PRIMARY KEY,
  product_id NUMBER,
  quantity NUMBER,
  CONSTRAINT fk_product
  FOREIGN KEY (product_id)
  REFERENCES products(id)
);
npm init –y
Server.js
const express = require(&amp;#39;express&amp;#39;);
const bodyParser = require(&amp;#39;body-parser&amp;#39;);
const oracledb = require(&amp;#39;oracledb&amp;#39;);
const app = express();

app.use(bodyParser.json());
// Oracle DB connection config
const dbConfig = {
  user: &amp;#39;SYSTEM&amp;#39;,
  password: &amp;#39;psgr&amp;#39;,
  connectString: &amp;#39;localhost:1521/XEPDB1&amp;#39;
};
// Get products
app.get(&amp;#39;/products&amp;#39;, async (req, res) =&amp;gt; {
  let connection;
  try {
    connection = await oracledb.getConnection(dbConfig);
    const result = await connection.execute(&amp;#39;SELECT * FROM products&amp;#39;);
    res.json(result.rows);
  } catch (err) {
```

```
  console.error(err);
    res.status(500).send(&amp;#39;DB  Error&amp;#39;);
  } finally {
    if (connection) await connection.close();
  }
});
// Add to cart
app.post(&amp;#39;/cart&amp;#39;, async (req, res) =&amp;gt; {

  const { product_id, quantity } = req.body;
  let connection;
  try {
    connection = await oracledb.getConnection(dbConfig);
    await connection.execute(
      &amp;#39;INSERT INTO cart (product_id, quantity) VALUES (:id, :qty)&amp;#39;,
      [product_id, quantity],

      { autoCommit: true }
    );
    res.send(&amp;#39;Added to cart&amp;#39;);
  } catch (err) {
    console.error(err);
    res.status(500).send(&amp;#39;DB  Error&amp;#39;);
  } finally {
    if (connection) await connection.close();
  }
});
app.get(&amp;quot;/&amp;quot;, (req, res) =&amp;gt; {
  res.send(&amp;quot;Server is running ��&amp;quot;);
});
app.listen(3000, () =&amp;gt; {
  console.log(&amp;quot;Server running on port 3000&amp;quot;);
});
node server.js
cd oracle-backend
cd New folder
npx create-react-app frontend
cd frontend
npm install axios
npm install react-router-dom
npm start
cart.js
import React, { useContext } from &amp;quot;react&amp;quot;;
import { CartContext } from &amp;quot;../context/cartcontext&amp;quot;;
const Cart = () =&amp;gt; {
```

```
 const { cart, removeFromCart } = useContext(CartContext);
 const total = cart.reduce(
  (sum, item) =&amp;gt; sum + item.price * item.qty,

  0
 );
 if (cart.length === 0) {
  return &amp;lt;h2 style={{ padding: &amp;quot;30px&amp;quot; }}&amp;gt;Cart is empty
  ��&amp;lt;/h2&amp;gt;;
 }
 return (
  &amp;lt;div className=&amp;quot;cart-container&amp;quot;&amp;gt;
   &amp;lt;h2&amp;gt;Your Cart&amp;lt;/h2&amp;gt;
   {cart.map((item) =&amp;gt; (
    &amp;lt;div className=&amp;quot;cart-item&amp;quot; key={item.id}&amp;gt;
     &amp;lt;img src={item.image} alt={item.name} /&amp;gt;
     &amp;lt;div&amp;gt;
      &amp;lt;h4&amp;gt;{item.name}&amp;lt;/h4&amp;gt;
      &amp;lt;p&amp;gt;₹{item.price}&amp;lt;/p&amp;gt;
      &amp;lt;p&amp;gt;Quantity: {item.qty}&amp;lt;/p&amp;gt;
      &amp;lt;p&amp;gt;Subtotal: ₹{item.price * item.qty}&amp;lt;/p&amp;gt;
      &amp;lt;button onClick={() =&amp;gt;
   removeFromCart(item.id)}&amp;gt;Remove&amp;lt;/button&amp;gt;
     &amp;lt;/div&amp;gt;
    &amp;lt;/div&amp;gt;
   ))}
   &amp;lt;h3&amp;gt;Total Amount: ₹{total}&amp;lt;/h3&amp;gt;
  &amp;lt;/div&amp;gt;
 );
};
export default Cart;
navbar.js
import React, { useContext } from &amp;quot;react&amp;quot;;
import { Link } from &amp;quot;react-router-dom&amp;quot;;
import { CartContext } from &amp;quot;../context/cartcontext&amp;quot;;
const Navbar = () =&amp;gt; {
 const { cart } = useContext(CartContext);

 return (
  &amp;lt;nav className=&amp;quot;navbar&amp;quot;&amp;gt;
   &amp;lt;h2&amp;gt;��FreshMart&amp;lt;/h2&amp;gt;
   &amp;lt;ul&amp;gt;
    &amp;lt;li&amp;gt;&amp;lt;Link
  to=&amp;quot;/&amp;quot;&amp;gt;Products&amp;lt;/Link&amp;gt;&amp;lt;/li&amp;gt;
    &amp;lt;li&amp;gt;&amp;lt;Link  to=&amp;quot;/cart&amp;quot;&amp;gt;Cart
```

```
      ({cart.length})&amp;lt;/Link&amp;gt;&amp;lt;/li&amp;gt;
    &amp;lt;/ul&amp;gt;
  &amp;lt;/nav&amp;gt;
 );
};
export default Navbar;
```

productcard.js

```
import React, { useContext } from &amp;quot;react&amp;quot;;
import { CartContext } from &amp;quot;../context/cartcontext&amp;quot;;
const ProductCard = ({ product }) =&amp;gt; {
 const { addToCart } = useContext(CartContext);
 return (
  &amp;lt;div className=&amp;quot;card&amp;quot;&amp;gt;
   &amp;lt;img src={product.image} alt={product.name} /&amp;gt;
   &amp;lt;h3&amp;gt;{product.name}&amp;lt;/h3&amp;gt;
   &amp;lt;p&amp;gt;₹{product.price}&amp;lt;/p&amp;gt;
   &amp;lt;button onClick={() =&amp;gt; addToCart(product)}&amp;gt;
    Add to Cart
  &amp;lt;/button&amp;gt;
  &amp;lt;/div&amp;gt;
 );
};
export default ProductCard;
```

productlist.js

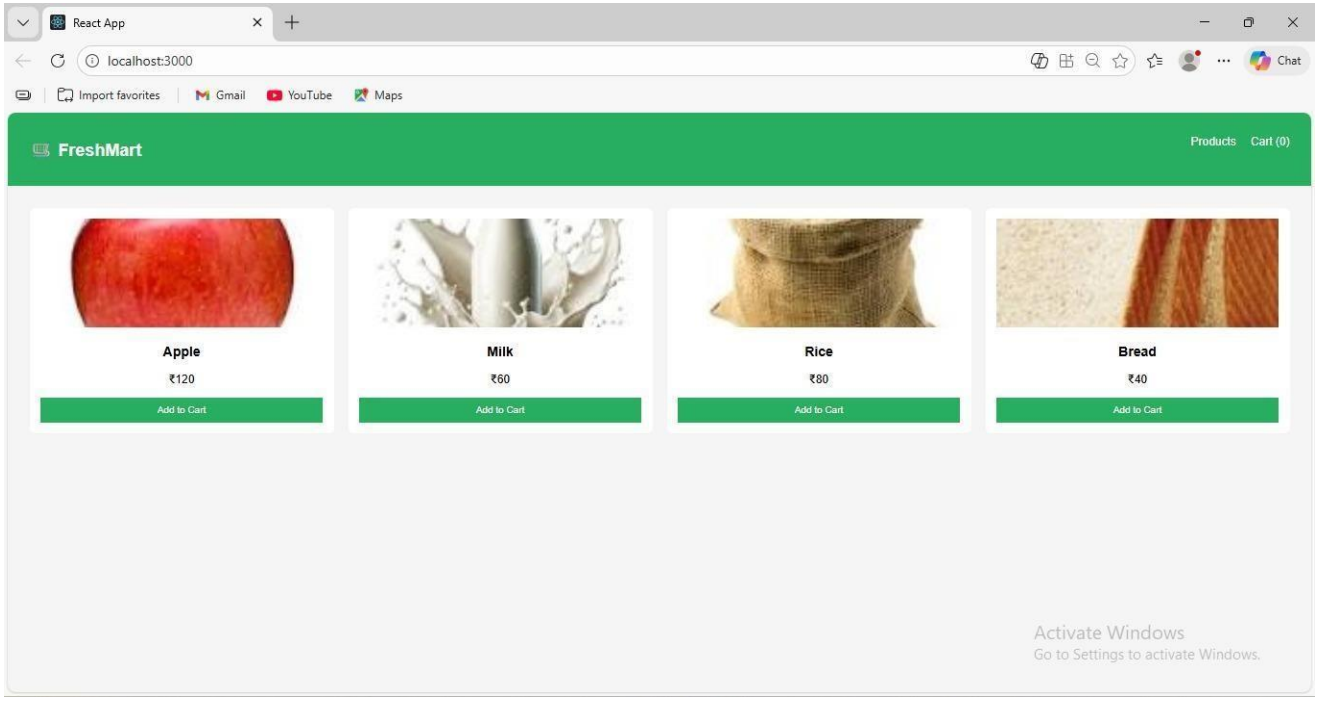
```
import React from &amp;quot;react&amp;quot;;
import ProductCard from &amp;quot;./productcard&amp;quot;;

const products = [
 { id: 1, name: &amp;quot;Apple&amp;quot;, price: 120, image:
  &amp;quot;/images/apple.jpg&amp;quot; },
 { id: 2, name: &amp;quot;Milk&amp;quot;, price: 60, image:
  &amp;quot;/images/milk.jpg&amp;quot; },
 { id: 3, name: &amp;quot;Rice&amp;quot;, price: 80, image:
  &amp;quot;/images/rice.jpg&amp;quot; },
 { id: 4, name: &amp;quot;Bread&amp;quot;, price: 40, image:
  &amp;quot;/images/bread.jpg&amp;quot; }
];

const ProductList = () =&amp;gt; {
 return (
  &amp;lt;div  className=&amp;quot;product-grid&amp;quot;&amp;gt;
   {products.map((p) =&amp;gt; (
    &amp;lt;ProductCard key={p.id} product={p} /&amp;gt;
   ))}
```

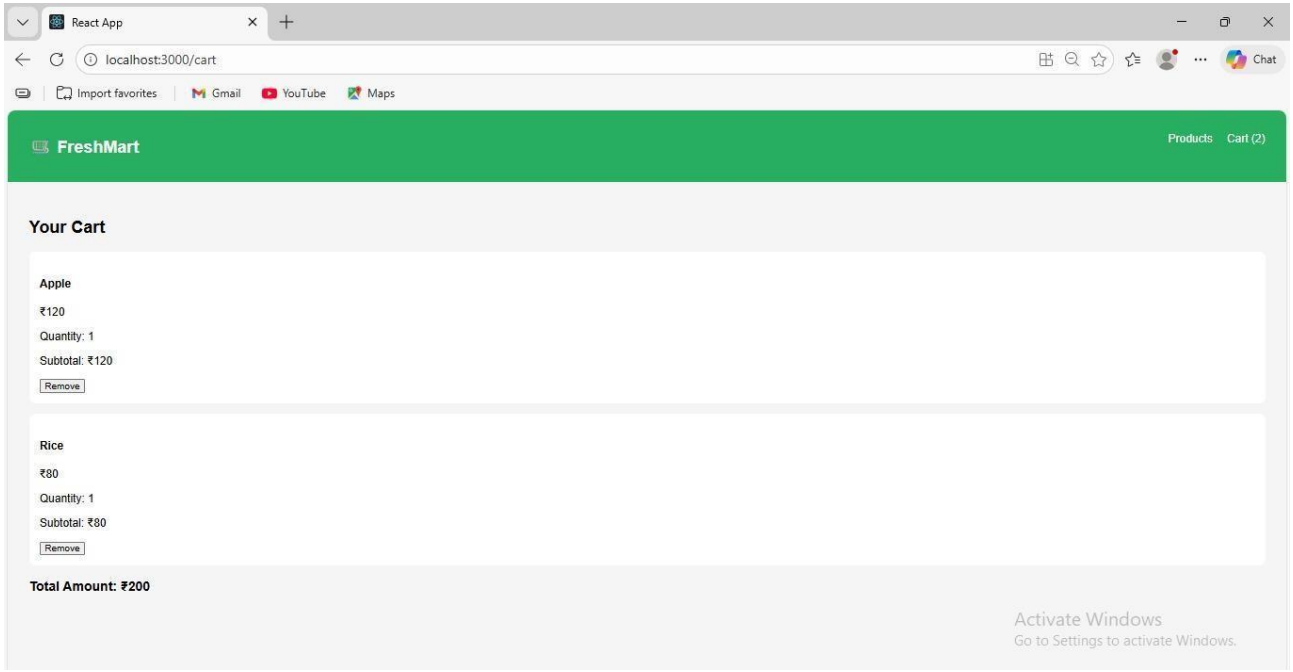
```
    &amp;lt;/div&amp;gt;
  );
};
export default ProductList;
cartcontext.js
import { createContext, useState } from &amp;quot;react&amp;quot;;
export const CartContext = createContext();
export const CartProvider = ({ children }) =&amp;gt; {
const [cart, setCart] = useState([]);
  const addToCart = (product) =&amp;gt; {
    setCart((prev) =&amp;gt; {
      const item = prev.find((p) =&amp;gt; p.id === product.id);
      if (item) {
        return prev.map((p) =&amp;gt;
          p.id === product.id ? { ...p, qty: p.qty + 1 } : p
        );
      }

      return [...prev, { ...product, qty: 1 }];
    });
  };
  const removeFromCart = (id) =&amp;gt; {
    setCart(cart.filter((item) =&amp;gt; item.id !== id));
  };
  return (
    &amp;lt;CartContext.Provider value={{ cart, addToCart, removeFromCart }}&amp;gt;

      {children}
    &amp;lt;/CartContext.Provider&amp;gt;
  );
};
App.css
body {
margin: 0;
  font-family: Arial, sans-serif;
  background: #f5f5f5;
}
/* Navbar */
.navbar {
  background: #27ae60;
  padding: 15px 30px;
  display: flex;
  justify-content: space-between;
  color: white;
```

```
}
.navbar ul {
  list-style: none;
  display: flex;
  gap: 20px;

}
.navbar a {
  color: white;
  text-decoration: none;
}
/* Products */
.product-grid {
  padding: 30px;
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 20px;
}
.card {
  background: white;
  padding: 15px;
  border-radius: 8px;
  text-align: center;
}
.card img {
  width: 100%;
  height: 150px;
  object-fit: cover;

}
.card button {
  background: #27ae60;
  color: white;
  border: none;
  padding: 10px;
  width: 100%;
  cursor: pointer;

}
/* Cart */
.cart-container {
  padding: 30px;
}
.cart-item {
  background: white;
```

```
display: flex;
  gap: 20px;
  padding: 15px;
  margin-bottom: 15px;
  border-radius: 8px;
}
.cart-item img {
  width:  100px;
  height: 100px;
}
APP.JS
import React from &amp;quot;react&amp;quot;;
import { BrowserRouter, Routes, Route } from &amp;quot;react-router-dom&amp;quot;;
import Navbar from &amp;quot;./components/Navbar&amp;quot;;
import ProductList from &amp;quot;./components/ProductList&amp;quot;;
import Cart from &amp;quot;./components/cart&amp;quot;;
import &amp;quot;./App.css&amp;quot;;
function App() {
  return (
    &amp;lt;BrowserRouter&amp;gt;
    &amp;lt;Navbar /&amp;gt;
    &amp;lt;Routes&amp;gt;
       &amp;lt;Route path=&amp;quot;/&amp;quot; element={&amp;lt;ProductList  /&amp;gt;}
    /&amp;gt;
       &amp;lt;Route path=&amp;quot;/cart&amp;quot; element={&amp;lt;Cart  /&amp;gt;}
    /&amp;gt;

      &amp;lt;/Routes&amp;gt;
    &amp;lt;/BrowserRouter&amp;gt;
  );
}
export default App;
index.js
import React from &amp;quot;react&amp;quot;;
import ReactDOM from &amp;quot;react-dom/client&amp;quot;;
import App from &amp;quot;./App&amp;quot;;
import { CartProvider } from &amp;quot;./context/cartcontext&amp;quot;;
const root = ReactDOM.createRoot(document.getElementById(&amp;quot;root&amp;quot;));
root.render(
  &amp;lt;CartProvider&amp;gt;
  &amp;lt;App /&amp;gt;
  &amp;lt;/CartProvider&amp;gt
```

OUTPUT


React App
localhost:3000
Import favorites
Gmail
YouTube
Maps
Chat
FreshMart
Products
Cart (0)
Apple
₹120
Add to Cart
Milk
₹60
Add to Cart
Rice
₹80
Add to Cart
Bread
₹40
Add to Cart
Activate Windows
Go to Settings to activate Windows.


React App
localhost:3000/cart
Import favorites
Gmail
YouTube
Maps
Chat
FreshMart
Products
Cart (2)
Your Cart
Apple
₹120
Quantity: 1
Subtotal: ₹120
Remove
Rice
₹80
Quantity: 1
Subtotal: ₹80
Remove
Total Amount: ₹200
Activate Windows
Go to Settings to activate Windows.

RESULT

Thus the program is executed successfully and the output was verified.

Ex No: 3 Date:	STUDENT INFORMATION SYSTEM

AIM

To develop a Student Information System application.

ALGORITHM

STEP 1: Start the program.

STEP 2: Import required modules (**tkinter, ttk, messagebox, sqlite3**).

STEP 3: Connect to the SQLite database and create the students table if it does not exist.

STEP 4: Create the main GUI window with labels and entry fields for student details.

STEP 5: Create buttons for Add, Update, Delete, and Clear operations.

STEP 6: Create a Treeview to display student records.

STEP 7: When the Add button is clicked, validate inputs and insert student data into the database.

STEP 8: When a record is selected, display its data in the entry fields and allow update or delete.

STEP 9: Refresh the Treeview after add, update, or delete operations.

STEP 10: Run the Tkinter main loop and close the database connection on exit.

PROGRAM

```
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3

# ------------------ Database ------------------
conn = sqlite3.connect("students.db")
cursor = conn.cursor()

cursor.execute("""
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER,
    course TEXT,
    email TEXT,
    marks INTEGER
)
""")
conn.commit()

# ------------------ Functions ------------------
def add_student():
    if id_entry.get() == "" or name_entry.get() == "":
        messagebox.showerror("Error", "ID and Name are required")
        return
    try:
        cursor.execute(
            "INSERT INTO students VALUES (?, ?, ?, ?, ?, ?)",
            (
                int(id_entry.get()),
```

```
        name_entry.get(),
        age_entry.get(),course_entry.get(
        ), email_entry.get(),
        marks_entry.get()
              )
          )
          conn.commit()
          messagebox.showinfo("Success", "Student added successfully")
          clear_fields()
          fetch_students()
      except sqlite3.IntegrityError:
          messagebox.showerror("Error", "Student ID already exists")

def fetch_students():
      for row in tree.get_children():
          tree.delete(row)
      cursor.execute("SELECT * FROM students")
      for row in cursor.fetchall():
          tree.insert("", tk.END, values=row)

def select_student(event):
      selected = tree.focus()
      if not selected:
          return
      values = tree.item(selected, "values")
      clear_fields()
      id_entry.insert(0, values[0])
      name_entry.insert(0, values[1])
      age_entry.insert(0, values[2])
      course_entry.insert(0, values[3])
      email_entry.insert(0, values[4])
```

```
        if id_entry.get() == "":
            messagebox.showerror("Error", "Select a student first")
            return
        cursor.execute("""
            UPDATE students
            SET name=?, age=?, course=?, email=?, marks=?
            WHERE id=?
        """, (
            name_entry.get(),
            age_entry.get(),
            course_entry.get(),
            email_entry.get(),
            marks_entry.get(),
            int(id_entry.get())
        ))
        conn.commit()
        messagebox.showinfo("Updated", "Student updated successfully")
        clear_fields()
        fetch_students()

    def delete_student():
        if id_entry.get() == "":
            messagebox.showerror("Error", "Select a student first")
            return
        cursor.execute("DELETE FROM students WHERE id=?", (int(id_entry.get()),))
        conn.commit()
        messagebox.showinfo("Deleted", "Student deleted successfully")
        clear_fields()
        fetch_students()

    def clear_fields():
```

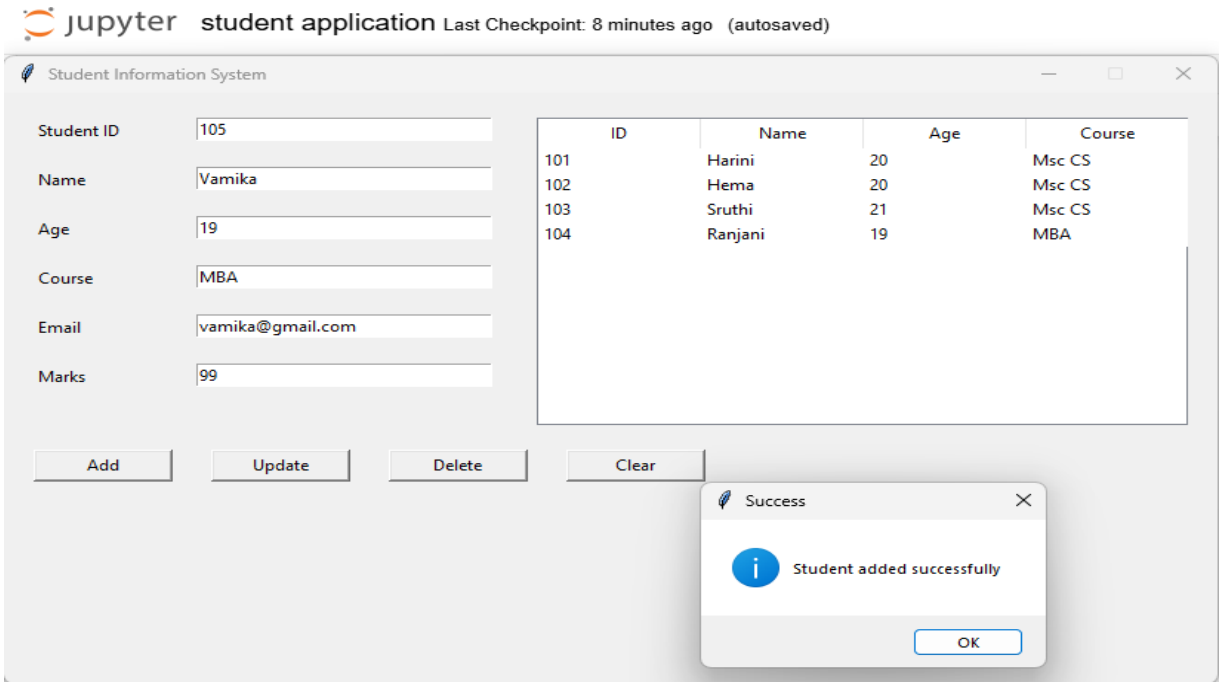
```
    id_entry.delete(0, tk.END)
    name_entry.delete(0, tk.END)
    age_entry.delete(0, tk.END)
    course_entry.delete(0, tk.END)
    email_entry.delete(0, tk.END)
    marks_entry.delete(0, tk.END)

# ---------- GUI ----------
root = tk.Tk()
root.title("Student Information System")
root.geometry("820x480")
root.resizable(False, False)

labels = ["Student ID", "Name", "Age", "Course", "Email", "Marks"]
for i, label in enumerate(labels):
    tk.Label(root, text=label).place(x=20, y=20 + i * 40)

id_entry = tk.Entry(root)
name_entry = tk.Entry(root)
age_entry = tk.Entry(root)
course_entry = tk.Entry(root)
email_entry = tk.Entry(root)
marks_entry = tk.Entry(root)

entries = [id_entry, name_entry, age_entry, course_entry, email_entry, marks_entry]
for i, entry in enumerate(entries):
    entry.place(x=130, y=20 + i * 40, width=200)

tk.Button(root, text="Add", width=12, command=add_student).place(x=20, y=290)
tk.Button(root, text="Update", width=12, command=update_student).place(x=140, y=290)
tk.Button(root, text="Delete", width=12, command=delete_student).place(x=260, y=290)
```

```
tk.Button(root, text="Clear", width=12, command=clear_fields).place(x=380, y=290)

columns = ("ID", "Name", "Age", "Course", "Email", "Marks")
tree = ttk.Treeview(root, columns=columns, show="headings")

for col in columns:
    tree.heading(col, text=col)
    tree.column(col, width=110)

tree.place(x=360, y=20, width=440, height=250)
tree.bind("<ButtonRelease-1>", select_student)

fetch_students()
root.mainloop()
conn.close()

    age_entry.delete(0, tk.END)
    course_entry.delete(0, tk.END)
    email_entry.delete(0, tk.END)
    marks_entry.delete(0, tk.END)

# ------------------ GUI ------------------
root = tk.Tk()
root.title("Student Information System")
root.geometry("820x480")
root.resizable(False, False)

labels = ["Student ID", "Name", "Age", "Course", "Email", "Marks"]
for i, label in enumerate(labels):
    tk.Label(root, text=label).place(x=20, y=20 + i * 40)
```

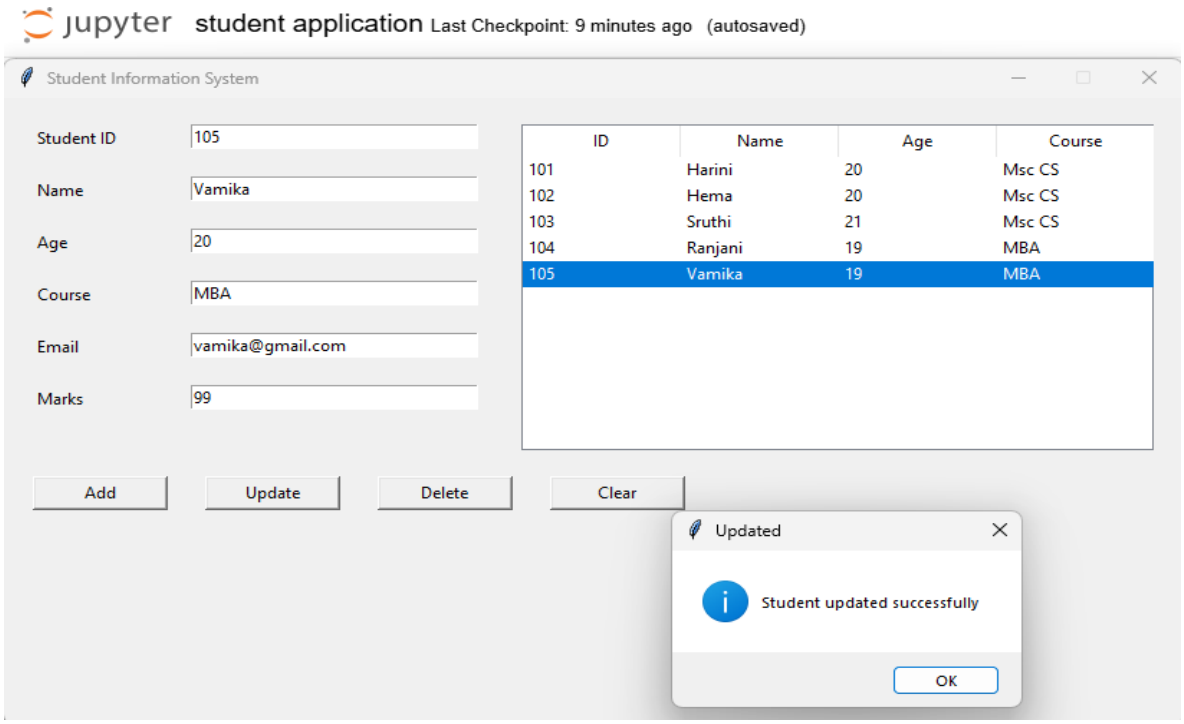
```
id_entry = tk.Entry(root)
name_entry = tk.Entry(root)
age_entry = tk.Entry(root)
course_entry = tk.Entry(root)
email_entry = tk.Entry(root)
marks_entry = tk.Entry(root)

entries = [id_entry, name_entry, age_entry, course_entry, email_entry, marks_entry]
for i, entry in enumerate(entries):
    entry.place(x=130, y=20 + i * 40, width=200)

tk.Button(root, text="Add", width=12, command=add_student).place(x=20, y=290)
tk.Button(root, text="Update", width=12, command=update_student).place(x=140, y=290)
tk.Button(root, text="Delete", width=12, command=delete_student).place(x=260, y=290)
tk.Button(root, text="Clear", width=12, command=clear_fields).place(x=380, y=290)

columns = ("ID", "Name", "Age", "Course", "Email", "Marks")
tree = ttk.Treeview(root, columns=columns, show="headings")

for col in columns:
    tree.heading(col, text=col)
    tree.column(col, width=110)

tree.place(x=360, y=20, width=440, height=250)
tree.bind("<ButtonRelease-1>", select_student)

fetch_students() root.mainloop() conn.close
```

OUTPUT

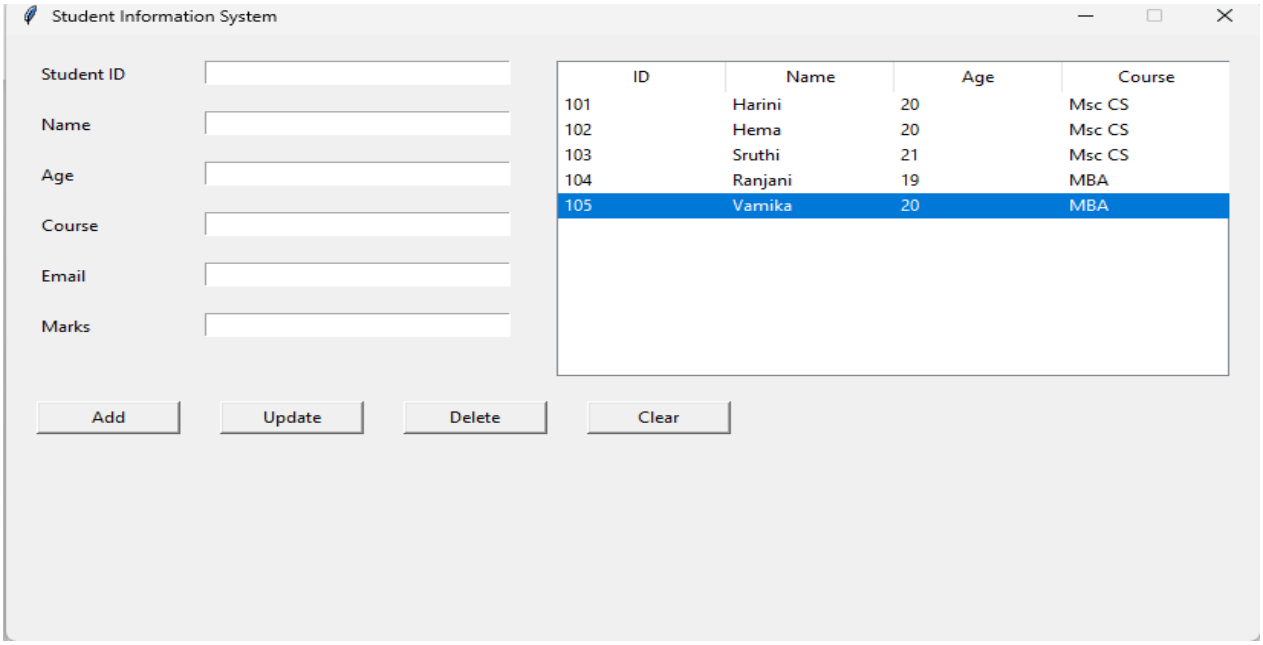
Adding student record:



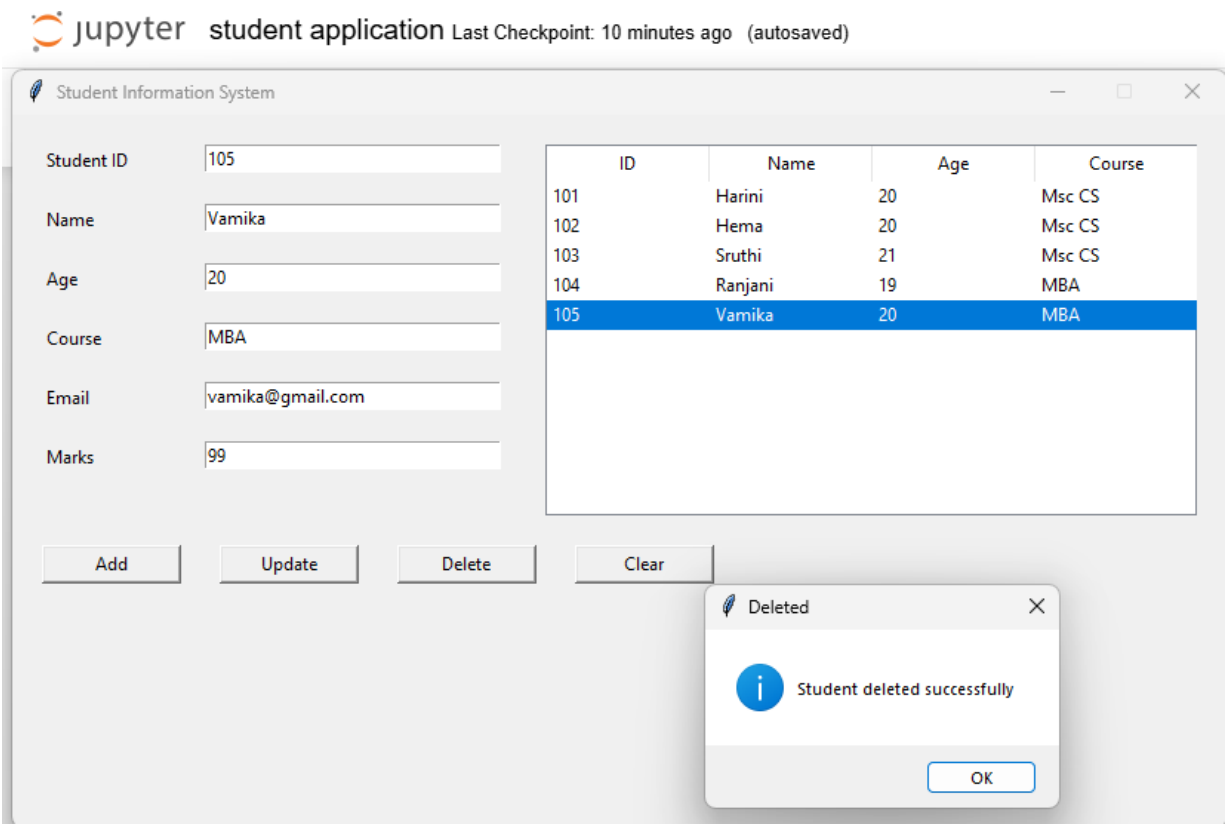
Updating student record

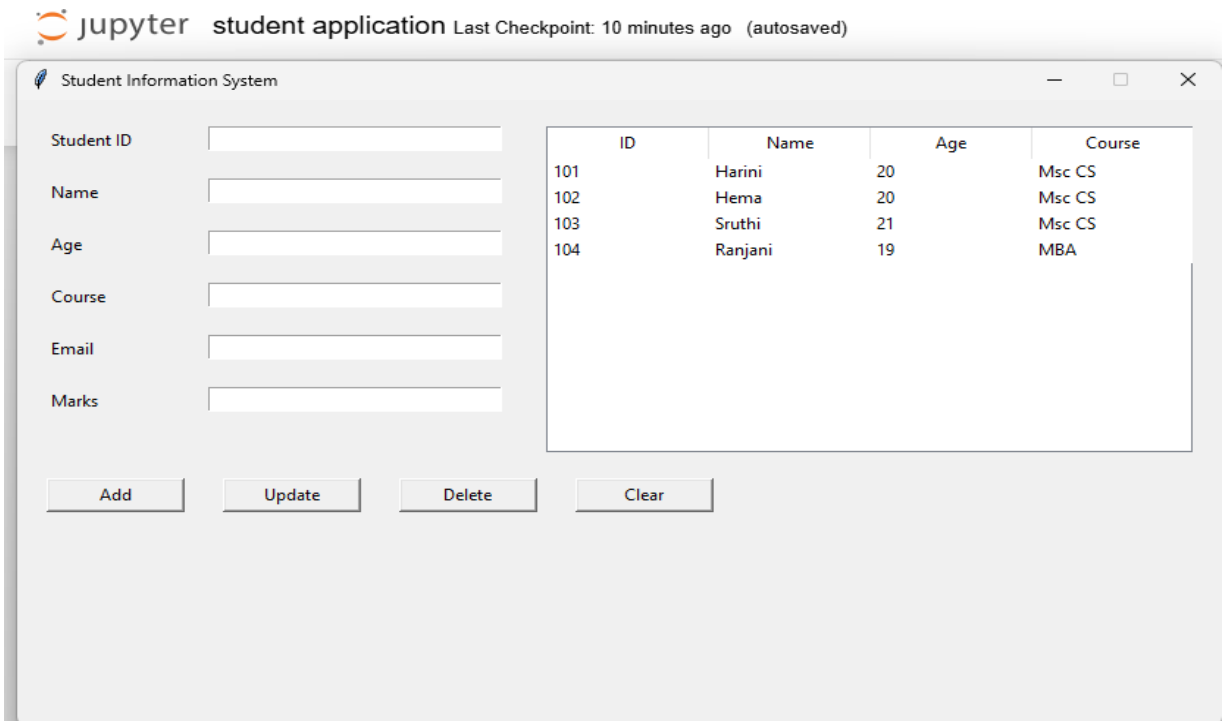


Clearing information in student record:



Deleting student record:




jupyter student application Last Checkpoint: 10 minutes ago (autosaved)
Student Information System
Student ID
Name
Age
Course
Email
Marks
ID
Name
Age
Course
101
Harini
20
Msc CS
102
Hema
20
Msc CS
103
Sruthi
21
Msc CS
104
Ranjani
19
MBA
Add
Update
Delete
Clear

RESULT

Thus the program is executed successfully and output is verified

Ex No: 4 Date:	DOCTOR APPOINTMENT

AIM

To design and develop a application doctor appointment.

ALGORITHM

STEP 1: Start the process

STEP 2: Create Doctors table to store doctor details (ID, Name, Specialization, Time).

STEP 3: Create Appointments table to store appointment records (ID, Patient Name, Doctor Name, Appointment Date).

STEP 4: Create sequence or auto-increment ID for appointment records.

STEP 5: Insert sample doctor data into Doctors table.

STEP 6: Configure database connection in backend server node server.js.

STEP 7: Create components for doctor list and appointment booking.

STEP 8: Create Angular service to call backend APIs.

STEP 9: Display doctors list fetched from database. Collect patient details using Angular form.

STEP 10: Connect Angular frontend with backend APIs.

STEP 11: Display confirmation message after successful appointment booking.

STEP 12: Stop the process.

PROGRAM

SQL> CREATE TABLE doctors id NUMBER PRIMARY KEY,name VARCHAR2(50), specialization VARCHAR2(50),time VARCHAR2(20));

Table created.

SQL> INSERT INTO doctors VALUES (1, 'Dr. Arun', 'Cardiologist', '10AM-2PM');

INSERT INTO doctors VALUES (2, 'Dr. Meena', 'Dentist', '3PM-6PM');

COMMIT;

SQL> SELECT * FROM doctors;

SQL> CREATE TABLE appointments (id NUMBER PRIMARY KEY, patient_name VARCHAR2(50),doctor_name VARCHAR2(50), appointment_date DATE);

SQL> CREATE SEQUENCE appointments_seq START WITH 1 INCREMENT BY 1 NOCACHE;

SQL> INSERT INTO appointments(id, patient_name, doctor_name, appointment_date) VALUES(appointments_seq.NEXTVAL, 'Ravi','Dr. Arun',TO_DATE('2026-01-30','YYYY-MM-DD'));

COMMIT;

SQL> SELECT * FROM appointments;

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent implements OnInit {
 patientName = '';
  doctorName = '';
  appointmentDate = '';
  doctors: any[] = [];
  appointments: any[] = [];
 constructor(private http: HttpClient) {}
  ngOnInit() {
    this.loadDoctors();
    this.loadAppointments();
  }
   loadDoctors() {
    this.http.get<any[]>('http://localhost:3000/doctors')
      .subscribe(data => this.doctors = data);
       }
     loadAppointments() {
    this.http.get<any[]>('http://localhost:3000/appointments')
      .subscribe(data => this.appointments = data);
  }
 bookAppointment() {
    const body = {
      patientName: this.patientName,
      doctorName: this.doctorName,
      appointmentDate: this.appointmentDate
    };
      this.http.post('http://localhost:3000/appointments',  body)
      .subscribe(() => {
```

```
      alert('Appointment Booked');
      this.loadAppointments();
    });
  }
}
```

app.component.html

```
<h2>Book Doctor Appointment</h2>
<label>Patient Name:</label>
<input type="text" [(ngModel)]="patientName"><br><br>
<label>Select Doctor:</label>
<select [(ngModel)]="doctorName">
  <option value="">--Select--</option>
  <option *ngFor="let d of doctors" [value]="d.NAME">
    {{ d.NAME }}
  </option>
</select><br><br>
<label>Appointment Date:</label>
<input type="date" [(ngModel)]="appointmentDate"><br><br>
<button (click)="bookAppointment()">Book</button>
<hr>
<h2>Doctor List</h2>
<table border="1">
  <tr>
    <th>ID</th>
    <th>Name</th>
    <th>Specialization</th>
    <th>Time</th>
  </tr>
  <tr *ngFor="let d of doctors">
    <td>{{d.ID}}</td>
    <td>{{d.NAME}}</td>
    <td>{{d.SPECIALIZATION}}</td>
    <td>{{d.TIME}}</td>
  </tr>
```

```
</table>
<hr>
<h2>Appointments</h2>
<table border="1">
  <tr>
    <th>ID</th>
    <th>Patient</th>
    <th>Doctor</th>
    <th>Date</th>
  </tr>
  <tr *ngFor="let a of appointments">
    <td>{{a.ID}}</td>
    <td>{{a.Patient_name}}</td>
    <td>{{a.Doctor_name}}</td>
    <td>{{a.Appointment_date | date}}</td>
  </tr>
</table>
```

doctor.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Injectable({
  providedIn: 'root'
})
export class DoctorService {
 private apiUrl = 'http://localhost:3000';
constructor(private http: HttpClient) {}
getDoctors() {
    return this.http.get<any[]>(`${this.apiUrl}/doctors`);
  }
  bookAppointment(data: any) {
    return this.http.post(`${this.apiUrl}/book`, data);
```

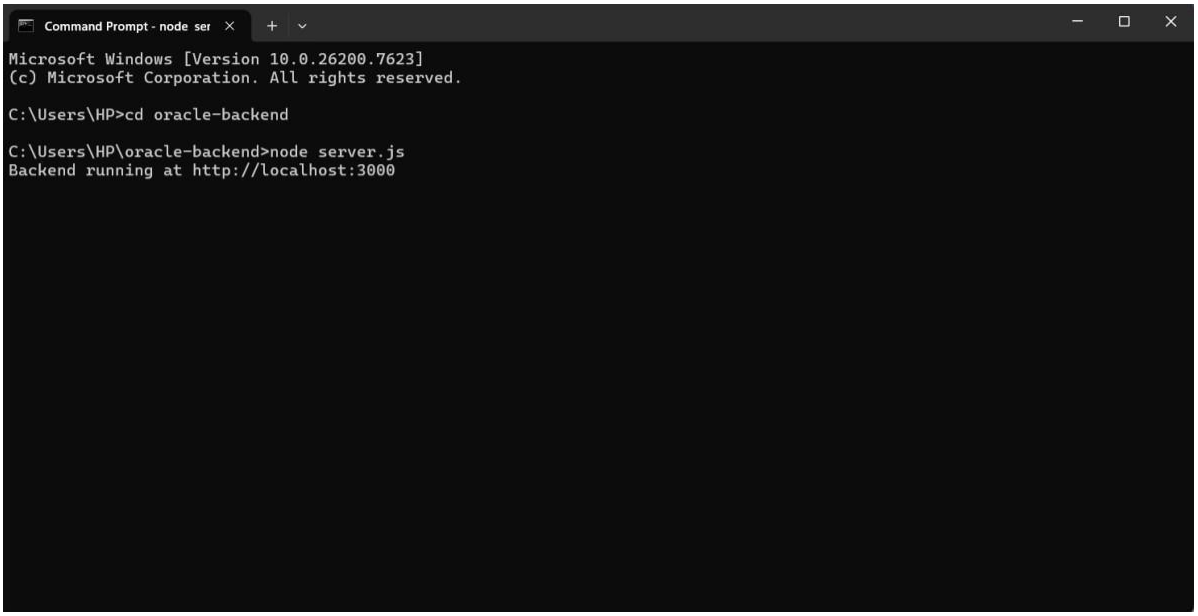
```
  }
}
```

doctor.service.spec.ts

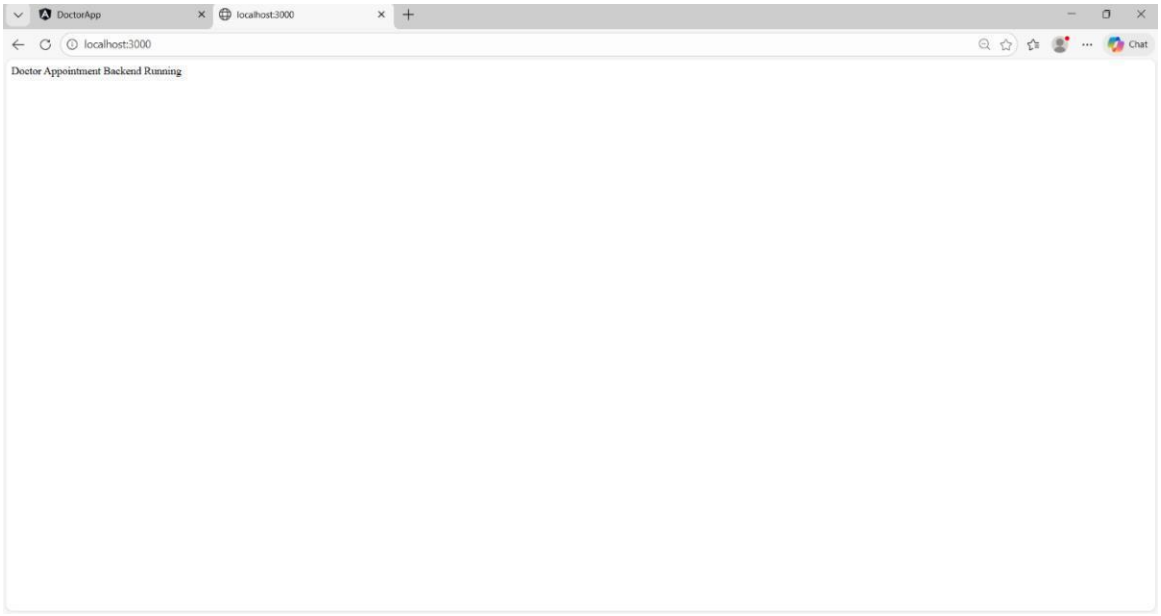
```
import { TestBed } from '@angular/core/testing';
import { DoctorService } from './doctor.service';
describe('DoctorService', () => {
  let service: DoctorService;
 beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(DoctorService);
  });
 it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

OUTPUT

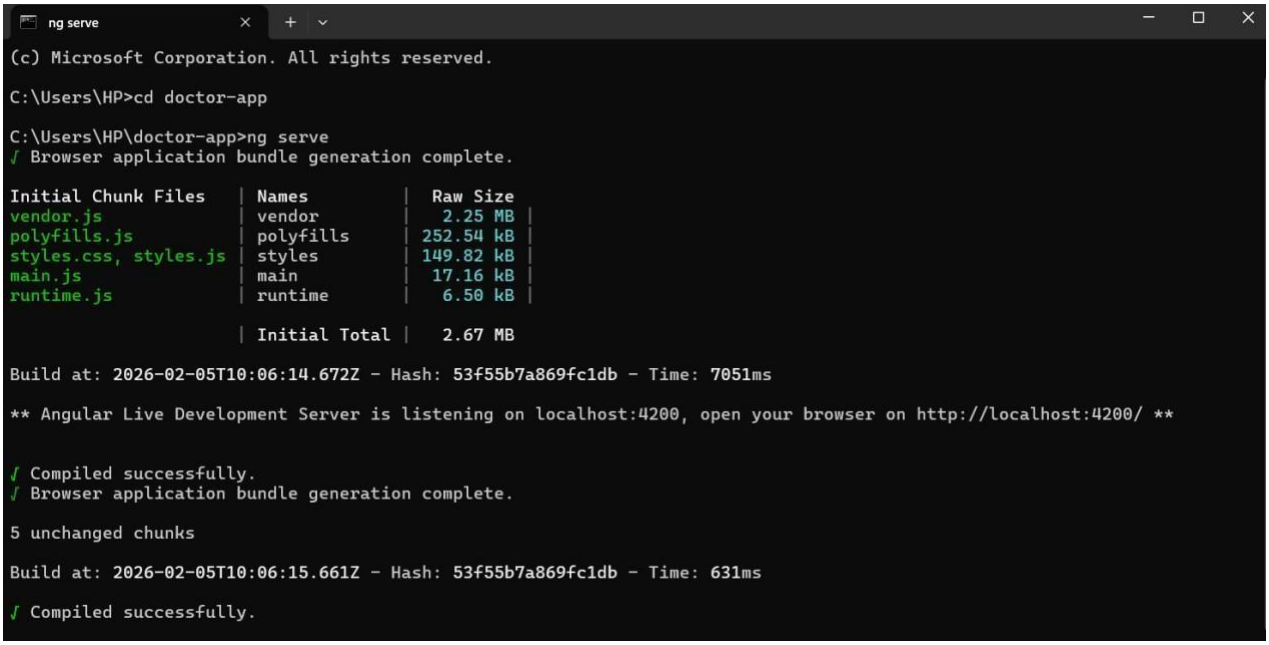
Backend Running node Server.js



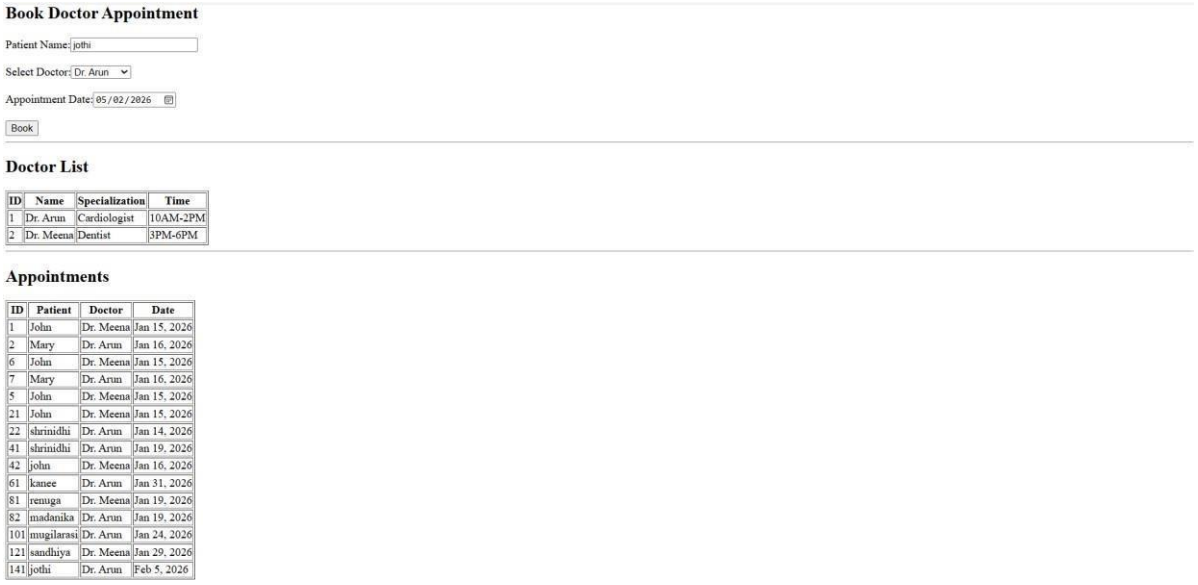
Doctor Appointment Running



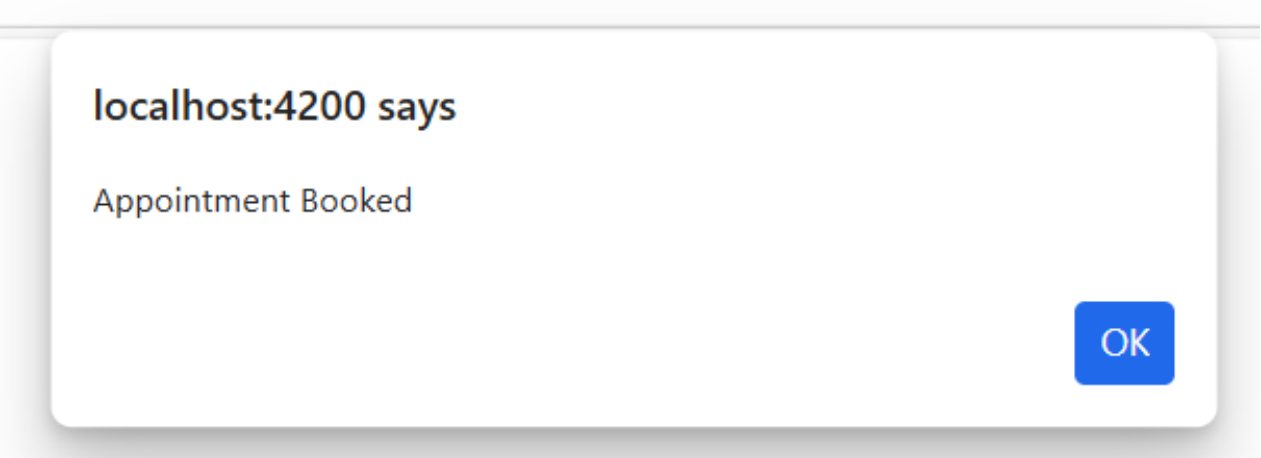
Angular Running in Command Prompt



Book Appointment Running



Local Host Booked Status



To check Backend To Conform Appointment

```
        ID PATIENT_NAME
---------- --------------------------------------------------
DOCTOR_NAME                                        APPOINTME
-------------------------------------------------- ---------
       101 mugilarasi
Dr. Arun                                           24-JAN-26

       121 sandhiya
Dr. Meena                                          29-JAN-26

       141 jothi
Dr. Arun                                           05-FEB-26


15 rows selected.
```

RESULT

Thus, the program was successfully executed and the output was verified.

Ex No: 5 Date:	EMPLOYEE PAYROLL

AIM

To develop an Employee Payroll System for managing employee records and salary calculation.

ALGORITHM

STEP 1: Start the program and import sqlite3 and tkinter.

STEP 2: Connect to the database payroll.db and create the employee table if it does not exist.

STEP 3: Design the GUI with fields for Employee ID, Name, Basic, Allowance, Deduction, and buttons for Add and Calculate Salary.

STEP 4: When Add Employee is clicked, read inputs and store the employee details in the database.

STEP 5: When Calculate Salary is clicked, fetch the employee record and calculate Net Salary = Basic + Allowance − Deduction.

STEP 6: Display success messages or errors using message boxes.import sqlite3.

STEP 7: Close the database and exit when the program ends.

PROGRAM

```
import sqlite3
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk

# ---------- DATABASE ----------
conn = sqlite3.connect("payroll.db")
cur = conn.cursor()

cur.execute("""
CREATE TABLE IF NOT EXISTS employee(
    emp_id INTEGER PRIMARY KEY,
    name TEXT,
    basic REAL,
    allowance REAL,
    deduction REAL
)
""")
conn.commit()

# ---------- FUNCTIONS ----------
table_visible = False


def add_employee():
    try:
        emp_id = int(entry_id.get())
        name = entry_name.get()
        basic = float(entry_basic.get())
        allowance = float(entry_allowance.get())
        deduction = float(entry_deduction.get())

        if name.strip() == "":
            messagebox.showerror("Error", "Employee name cannot be empty")
            return
```

```
            if basic < 0 or allowance < 0 or deduction < 0:
                messagebox.showerror("Error", "Salary values cannot be negative")
                return

            cur.execute(
                "INSERT INTO employee VALUES (?, ?, ?, ?, ?)",
                (emp_id, name, basic, allowance, deduction)
            )
            conn.commit()

            messagebox.showinfo("Success", "Employee Added Successfully")
            clear_entries()

        except sqlite3.IntegrityError:
            messagebox.showerror("Error", "Employee ID already exists")
        except ValueError:
            messagebox.showerror("Error", "Please enter valid values")



    def calculate_salary():
        global table_visible
        try:
            emp_id = int(entry_id.get())

            cur.execute(
                "SELECT emp_id, name, basic, allowance, deduction FROM employee WHERE emp_id=?",
                (emp_id,)
            )
            data = cur.fetchone()
            if not data:
                messagebox.showerror("Error", "Employee Not Found")
                return

            emp_id, name, basic, allowance, deduction = data
            net_salary = basic + allowance - deduction

            # Fill text boxes
```

```
        entry_name.delete(0, tk.END)
        entry_basic.delete(0, tk.END)
        entry_allowance.delete(0, tk.END)
        entry_deduction.delete(0, tk.END)
        entry_net.delete(0, tk.END)

        entry_name.insert(0, name)
        entry_basic.insert(0, f"{basic:.2f}")
        entry_allowance.insert(0, f"{allowance:.2f}")
        entry_deduction.insert(0, f"{deduction:.2f}")
        entry_net.insert(0, f"{net_salary:.2f}")

        # Show only selected employee in table
        load_selected_employee(
            emp_id, name, basic, allowance, deduction, net_salary
        )

        # Auto-show table if hidden
        if not table_visible:
            tree.pack(pady=20)
            view_btn.config(text="Hide Employees")
            table_visible = True

    except ValueError:
        messagebox.showerror("Error", "Enter a valid Employee ID"
def load_selected_employee(emp_id, name, basic, allowance, deduction, net_salary):
    for row in tree.get_children():
        tree.delete(row)

    tree.insert(
        "", tk.END,
        values=(
            emp_id,
            name,
            f"{basic:.2f}",
            f"{allowance:.2f}",
            f"{deduction:.2f}",
```

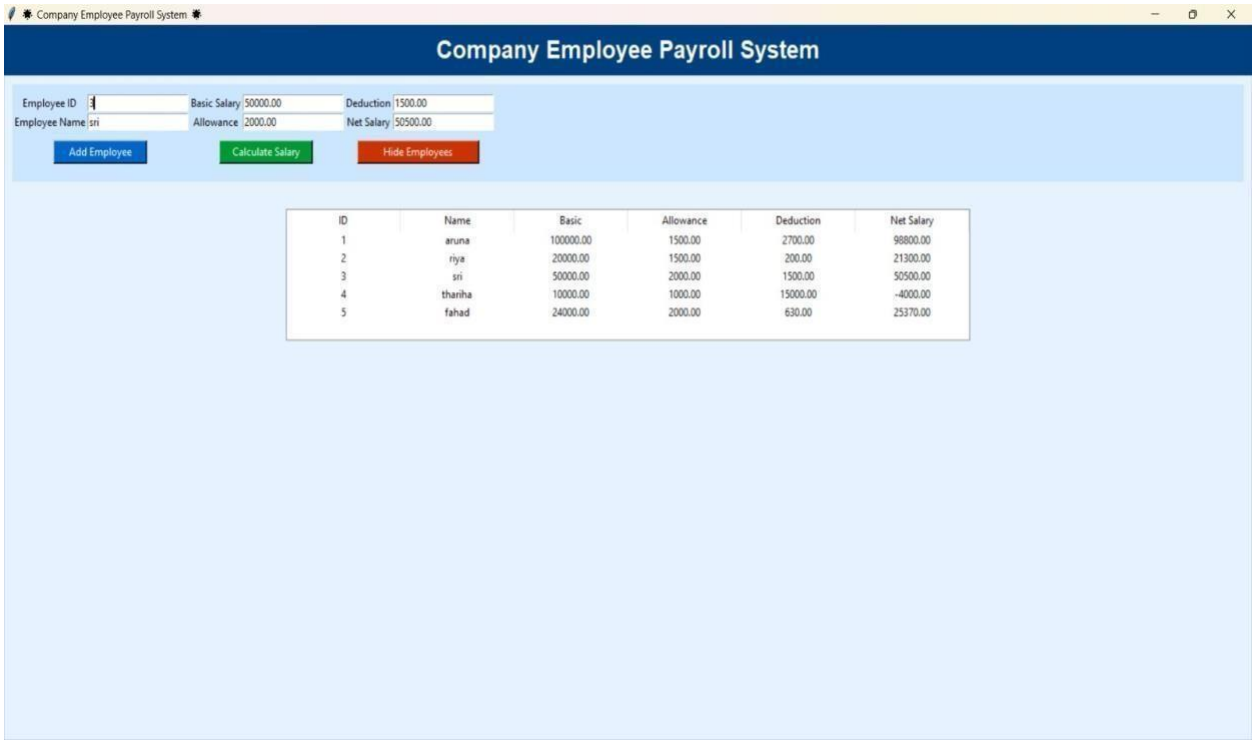
```
            f"{net_salary:.2f}"
        )
    )


def load_all_employees():
    for row in tree.get_children():
        tree.delete(row)

    cur.execute("SELECT * FROM employee")
    for row in cur.fetchall():
        net_salary = row[2] + row[3] - row[4]
        tree.insert(
            "", tk.END,
            values=(
                row[0], row[1],
                f"{row[2]:.2f}", f"{row[3]:.2f}",
                f"{row[4]:.2f}", f"{net_salary:.2f}"
            )

def toggle_employees():
    global table_visible

    if table_visible:
        tree.pack_forget()
        view_btn.config(text="View Employees")
        table_visible = False
    else:
        load_all_employees()
        tree.pack(pady=20)
        view_btn.config(text="Hide Employees")
        table_visible = True


def clear_entries():
    entry_id.delete(0, tk.END)
    entry_name.delete(0, tk.END)
```

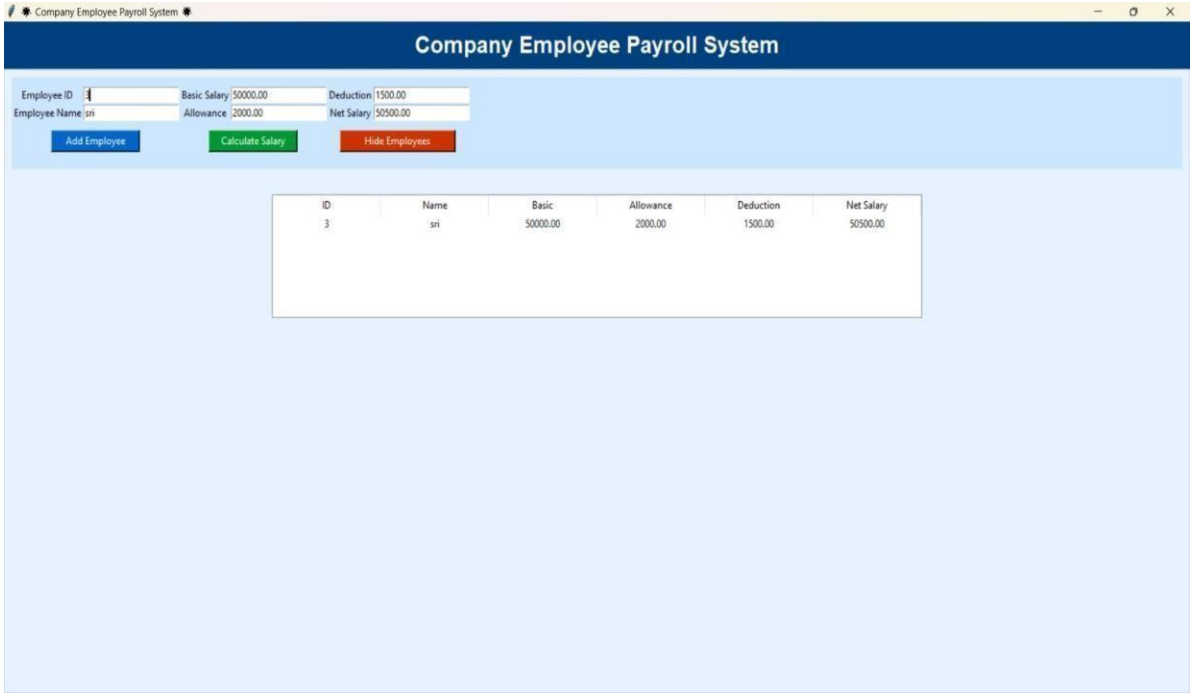
```
    entry_basic.delete(0, tk.END)
    entry_allowance.delete(0, tk.END)
    entry_deduction.delete(0, tk.END)
    entry_net.delete(0, tk.END)


def on_close():
    conn.close()
    root.destroy()

# ------------------ GUI ------------------
root = tk.Tk()
root.title("★ Company Employee Payroll System ★")
root.geometry("950x600")
root.configure(bg="#e6f2ff")
root.protocol("WM_DELETE_WINDOW", on_close)

# Title
tk.Label(
    root,
    text="Company Employee Payroll System",
    font=("Helvetica", 20, "bold"),
    fg="white",
    bg="#004080",
    pady=10
).pack(fill=tk.X)

# Input Frame
frame = tk.Frame(root, bg="#cce6ff", pady=10)
frame.pack(pady=10, padx=10, fill=tk.X)

# Inputs
tk.Label(frame, text="Employee ID", bg="#cce6ff").grid(row=0, column=0)
entry_id = tk.Entry(frame)
entry_id.grid(row=0, column=1)

tk.Label(frame, text="Employee Name", bg="#cce6ff").grid(row=1, column=0)
```

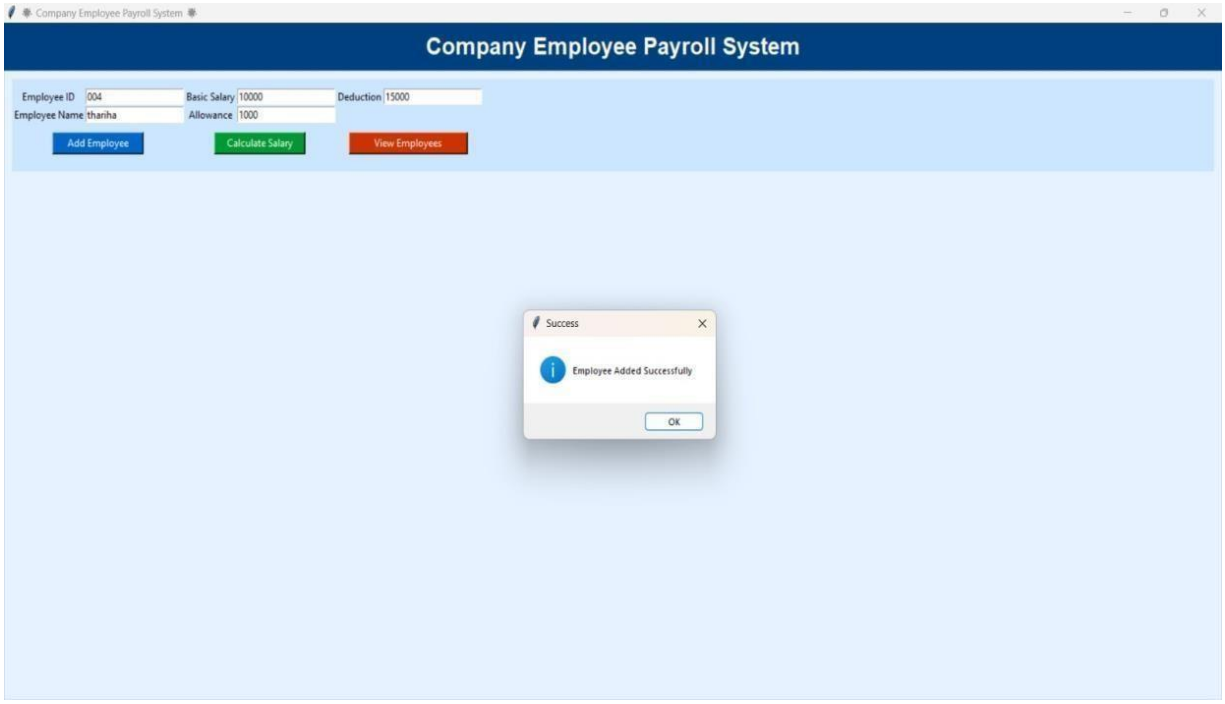
```
entry_name = tk.Entry(frame)
entry_name.grid(row=1, column=1)

tk.Label(frame, text="Basic Salary", bg="#cce6ff").grid(row=0, column=2)
entry_basic = tk.Entry(frame)
entry_basic.grid(row=0, column=3)

tk.Label(frame, text="Allowance", bg="#cce6ff").grid(row=1, column=2)
entry_allowance = tk.Entry(frame)
entry_allowance.grid(row=1, column=3)

tk.Label(frame, text="Deduction", bg="#cce6ff").grid(row=0, column=4)
entry_deduction = tk.Entry(frame)
entry_deduction.grid(row=0, column=5)

tk.Label(frame, text="Net Salary", bg="#cce6ff").grid(row=1, column=4)
entry_net = tk.Entry(frame)
entry_net.grid(row=1,  column=5)

# Buttons
tk.Button(
    frame, text="Add Employee", command=add_employee,
    bg="#0066cc", fg="white", width=15
).grid(row=2, column=0, columnspan=2, pady=10)

tk.Button(
    frame, text="Calculate Salary", command=calculate_salary,
    bg="#009933", fg="white", width=15
).grid(row=2, column=2, columnspan=2, pady=10)

view_btn = tk.Button(
    frame, text="View Employees", command=toggle_employees,
    bg="#cc3300", fg="white", width=20
)
view_btn.grid(row=2, column=4, columnspan=2, pady=10)

# ---------- TABLE ----------
```

```
columns = ("ID", "Name", "Basic", "Allowance", "Deduction", "Net Salary")
tree = ttk.Treeview(root, columns=columns, show="headings", height=6)

for col in columns:
    tree.heading(col, text=col)
    tree.column(col, width=140, anchor="center")

# ---------- RUN ----------
root.mainloop()
```

OUTPUT


Company Employee Payroll System
Company Employee Payroll System
Employee ID 3
Employee Name sri
Basic Salary 50000.00
Allowance 2000.00
Deduction 1500.00
Net Salary 50500.00
Add Employee
Calculate Salary
Hide Employees
ID Name Basic Allowance Deduction Net Salary
1 aruna 100000.00 1500.00 2700.00 98800.00
2 riya 20000.00 1500.00 200.00 21300.00
3 sri 50000.00 2000.00 1500.00 50500.00
4 thariha 10000.00 1000.00 15000.00 -4000.00
5 fahad 24000.00 2000.00 630.00 25370.00


Company Employee Payroll System
Company Employee Payroll System
Employee ID 3
Employee Name sri
Basic Salary 50000.00
Allowance 2000.00
Deduction 1500.00
Net Salary 50500.00
Add Employee
Calculate Salary
Hide Employees
ID Name Basic Allowance Deduction Net Salary
3 sri 50000.00 2000.00 1500.00 50500.00


Company Employee Payroll System
Company Employee Payroll System
Employee ID
004
Basic Salary
10000
Deduction
15000
Employee Name
thariha
Allowance
1000
Add Employee
Calculate Salary
View Employees
Success
Employee Added Successfully
OK

RESULT

Thus the program is executed successfully and the output was verified.