DBMS and Visual Basic Practical File

Master of Operational Research

Name: -

College: Department of Operational Research

- B

:

Submitted To: Dr. Jagvinder Singh

Q.1

Create an ORGANIZATION DATABASE having two TABLES EMPLOYEE and TRAINING with the following fields:

EMPLOYEE: {EMP_ID, Name, Gender, Age, Qualification, Department ('Sales', 'Marketing', 'Operations')}

TRAINING: {EMP_ID, PERFORMANCE_TRAINING1, PERFORMANCE_TRAINING2, PERFORMANCE_TRAINING3}

Perform the following queries in SQL:

i. Insert at least 10 records in the tables.
ii. Display the details of all employees who work in 'Operations' Department.
iii. Display the details of employee whose performance is poor.

Solution 1:

```
CREATE TABLE EMPLOYEE (
   EMP_ID INT PRIMARY KEY,
   NAME VARCHAR(50),
   GENDER CHAR(1),
   AGE INT,
   QUALIFICATION VARCHAR(50),
   DEPARTMENT VARCHAR(20)
);

CREATE TABLE TRAINING (
   EMP_ID INT,
   PERFORMANCE_TRAINING1 VARCHAR(20),
   PERFORMANCE_TRAINING2 VARCHAR(20),
   PERFORMANCE_TRAINING3 VARCHAR(20),
   FOREIGN KEY (EMP_ID) REFERENCES EMPLOYEE(EMP_ID)
);
```

i. Insert at least 10 records in the tables

```
INSERT INTO EMPLOYEE VALUES
(1, 'John', 'M', 29, 'MBA', 'Sales'),
(2, 'Asha', 'F', 31, 'B.Tech', 'Operations'),
(3, 'Ravi', 'M', 27, 'MBA', 'Marketing'),
(4, 'Sneha', 'F', 34, 'BCA', 'Operations'),
(5, 'Amit', 'M', 30, 'MCA', 'Sales'),
(6, 'Priya', 'F', 26, 'BBA', 'Marketing'),
(7, 'Vikas', 'M', 35, 'M.Tech', 'Operations'),
(8, 'Neha', 'F', 28, 'B.Com', 'Sales'),
(9, 'Raj', 'M', 32, 'MBA', 'Marketing'),
(10, 'Divya', 'F', 33, 'MCA', 'Operations');

INSERT INTO TRAINING VALUES
(1, 'Good', 'Average', 'Good'),
(2, 'Poor', 'Average', 'Good'),
(3, 'Good', 'Good', 'Good'),
(4, 'Poor', 'Poor', 'Average'),
```

```
(5, 'Good', 'Average', 'Good'),
(6, 'Average', 'Average', 'Good'),
(7, 'Poor', 'Average', 'Poor'),
(8, 'Good', 'Good', 'Average'),
(9, 'Average', 'Average', 'Good'),
(10, 'Poor', 'Poor', 'Poor');

-- ii. Employees from 'Operations'
SELECT * FROM EMPLOYEE WHERE DEPARTMENT = 'Operations';
```

Results Messages

	EMP_ID	NAME	GENDER	AGE	QUALIFICATION	DEPARTMENT
1	2	Asha	F	31	B.Tech	Operations
2	4	Sneha	F	34	BCA	Operations
3	7	Vikas	M	35	M.Tech	Operations
4	10	Divya	F	33	MCA	Operations

```
-- iii. Employees with any 'Poor' performance
SELECT E.*
FROM EMPLOYEE E
JOIN TRAINING T ON E.EMP_ID = T.EMP_ID
WHERE 'Poor' IN (PERFORMANCE_TRAINING1, PERFORMANCE_TRAINING2,
PERFORMANCE_TRAINING3);
```

Results Messages

	EMP_ID	NAME	GENDER	AGE	QUALIFICATION	DEPARTMENT
1	2	Asha	F	31	B.Tech	Operations
2	4	Sneha	F	34	BCA	Operations
3	7	Vikas	M	35	M.Tech	Operations
4	10	Divya	F	33	MCA	Operations

Q.2

Create a College Database and a student table as given bellow and perform various operations

Table 1 Student

S_Name	S_Rollno	S_DOB	Gender	Fee	S_enrolment	D_code
Munesh	101	09-01-1983	F	38000	013445534	103
Karan	205	08-12-1984	M	55000	128665542	102
Sashi	305	19-01-1981	F	35000	017654363	101
Priya	306	20-06-1982	F	55000	128665574	103
Rama	405	15-09-1980	F	38000	133445565	104
Komal	104	31-07-1983	M	38000	143445563	103
Kikndra	109	29-03-1984	M	35000	167654367	101
Gaurav	106	10-11-1985	M	38000	102225411	103

Solution 2:

```
CREATE TABLE STUDENT (
   S_Name VARCHAR(50),
   S_Rollno INT PRIMARY KEY,
   S_DOB DATE,
```

```
    Gender CHAR(1),
    Fee DECIMAL(10, 2),
    S_enrolment VARCHAR(20),
    D_code INT
);

INSERT INTO STUDENT VALUES
('Munesh', 101, '1983-01-09', 'F', 38000, '013445534', 103),
('Karan', 205, '1984-12-08', 'M', 55000, '128665542', 102),
('Sashi', 305, '1981-01-19', 'F', 35000, '017654363', 101),
('Priya', 306, '1982-06-20', 'F', 55000, '128665574', 103),
('Rama', 405, '1980-09-15', 'F', 38000, '133445565', 104),
('Komal', 104, '1983-07-31', 'F', 38000, '143445563', 103),
('Kindhra', 109, '1984-03-29', 'M', 35000, '167654367', 101),
('Gaurav', 106, '1985-11-10', 'M', 38000, '102225411', 103);
```

1. Display all student details:

```
SELECT * FROM STUDENT;
```

Results Messages

	S_Name	S_Rollno	S_DOB	Gender	Fee	S_enrolment	D_code
1	Munesh	101	1983-01-09	F	38000.00	013445534	103
2	Komal	104	1983-07-31	F	38000.00	143445563	103
3	Gaurav	106	1985-11-10	M	38000.00	102225411	103
4	Kindhra	109	1984-03-29	M	35000.00	167654367	101
5	Karan	205	1984-12-08	M	55000.00	128665542	102
6	Sashi	305	1981-01-19	F	35000.00	017654363	101
7	Priya	306	1982-06-20	F	55000.00	128665574	103
8	Rama	405	1980-09-15	F	38000.00	133445565	104

2. Display the names of all students who are female:

```
SELECT S_Name FROM STUDENT WHERE Gender = 'F';
```

Results Messages

	S_Name
1	Munesh
2	Komal
3	Sashi
4	Priya
5	Rama

3. Display students who have paid fees more than 40,000:

```
SELECT * FROM STUDENT WHERE Fee > 40000;
```

Results Messages

	S_Name	S_Rollno	S_DOB	Gender	Fee	S_enrolment	D_code
1	Karan	205	1984-12-08	M	55000.00	128665542	102
2	Priya	306	1982-06-20	F	55000.00	128665574	103

4. Display students whose department code is 103:

```
SELECT * FROM STUDENT WHERE D_code = 103;
```

Results Messages

	S_Name	S_Rollno	S_DOB	Gender	Fee	S_enrolment	D_code
1	Munesh	101	1983-01-09	F	38000.00	013445534	103
2	Komal	104	1983-07-31	F	38000.00	143445563	103
3	Gaurav	106	1985-11-10	M	38000.00	102225411	103
4	Priya	306	1982-06-20	F	55000.00	128665574	103

5. Display the student names in ascending order of their names:
SELECT S_Name FROM STUDENT ORDER BY S_Name ASC;

Results Messages

	S_Name
1	Gaurav
2	Karan
3	Kindhra
4	Komal
5	Munesh
6	Priya
7	Rama
8	Sashi

6. Count the number of students in each department:
SELECT D_code, COUNT(*) AS Number_of_Students FROM STUDENT GROUP BY D_code;

Results Messages

	D_code	Number_of_Students
1	101	2
2	102	1
3	103	4
4	104	1

7. Display the total fee collected from students:
SELECT SUM(Fee) AS Total_Fee_Collected FROM STUDENT;

Results Messages

	Total_Fee_Collected
1	332000.00

Q1. Create a table ORDERS and PRODUCTS with the following fields

ORDERS: {ORDER_ID, CUSTOMER_ID, PRODUCT_ID, QUANTITY, ORDER_DATE}
PRODUCTS: {PRODUCT_ID, PRODUCT_NAME, PRICE}

Perform the following queries in SQL:

1. **Insert at least 10 records into the ORDERS and PRODUCTS tables. (2 marks)**
2. **Count the number of orders placed by each customer. (1 mark)**
3. **Calculate the total quantity of each product sold. (1 mark)**
4. **Find the top 3 products that generated the highest revenue. (2 marks)**
5. **Display the details of orders placed on or after '2024-05-05' sorted by ORDER_DATE in descending order. (1 mark)**

6. **Add a data constraint to ensure that the QUANTITY in the ORDERS table is always greater than zero. (1 mark)**
7. **Calculate the total revenue generated by each customer. (2 marks)**

Solution 1:

```
 Creating the PRODUCTS table
CREATE TABLE PRODUCTS (
   PRODUCT_ID INT PRIMARY KEY,
   PRODUCT_NAME VARCHAR(100),
   PRICE DECIMAL(10,2)
);

-- Creating the ORDERS table
CREATE TABLE ORDERS (
   ORDER_ID INT PRIMARY KEY,
   CUSTOMER_ID INT,
   PRODUCT_ID INT,
   QUANTITY INT CHECK (QUANTITY > 0), -- Constraint to ensure quantity is always greater than zero
   ORDER_DATE DATE,
   FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCTS(PRODUCT_ID)
);

-- Inserting at least 10 records into PRODUCTS
INSERT INTO PRODUCTS (PRODUCT_ID, PRODUCT_NAME, PRICE) VALUES
(1, 'Laptop', 800.00),
(2, 'Smartphone', 500.00),
(3, 'Tablet', 300.00),
(4, 'Smartwatch', 200.00),
(5, 'Headphones', 100.00),
(6, 'Keyboard', 50.00),
(7, 'Mouse', 30.00),
(8, 'Monitor', 250.00),
(9, 'Printer', 150.00),
(10, 'Webcam', 80.00);
```

```
-- Inserting at least 10 records into ORDERS
INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, PRODUCT_ID, QUANTITY, ORDER_DATE) VALUES
(1, 101, 1, 2, '2024-05-01'),
(2, 102, 3, 1, '2024-05-02'),
(3, 103, 2, 3, '2024-05-03'),
(4, 101, 5, 4, '2024-05-04'),
(5, 102, 6, 2, '2024-05-06'),
(6, 104, 4, 1, '2024-05-07'),
(7, 105, 7, 3, '2024-05-08'),
(8, 101, 8, 1, '2024-05-09'),
(9, 103, 9, 2, '2024-05-10'),
(10, 106, 10, 1, '2024-05-11');

-- 2. Count the number of orders placed by each customer
SELECT CUSTOMER_ID, COUNT(*) AS TOTAL_ORDERS
FROM ORDERS
GROUP BY CUSTOMER_ID;
```

Results | Messages

	CUSTOMER_ID	TOTAL_ORDERS
1	101	3
2	102	2
3	103	2
4	104	1
5	105	1
6	106	1

```
-- 3. Calculate the total quantity of each product sold
SELECT PRODUCT_ID, SUM(QUANTITY) AS TOTAL_QUANTITY_SOLD
FROM ORDERS
GROUP BY PRODUCT_ID;
```

Results Messages

	PRODUCT_ID	TOTAL_QUANTITY_SOLD
1	1	2
2	2	3
3	3	1
4	4	1
5	5	4
6	6	2
7	7	3
8	8	1
9	9	2
10	10	1

-- 4. Find the top 3 products that generated the highest revenue

```
SELECT P.PRODUCT_ID, P.PRODUCT_NAME, SUM(O.QUANTITY * P.PRICE) AS TOTAL_REVENUE
FROM ORDERS O
JOIN PRODUCTS P ON O.PRODUCT_ID = P.PRODUCT_ID
GROUP BY P.PRODUCT_ID, P.PRODUCT_NAME
ORDER BY TOTAL_REVENUE DESC
LIMIT 3;
```

-- 5. Display the details of orders placed on or after '2024-05-05' sorted by ORDER_DATE in descending order

```
SELECT * FROM ORDERS
WHERE ORDER_DATE >= '2024-05-05'
ORDER BY ORDER_DATE DESC;
```

Results Messages

	ORDER_ID	CUSTOMER_ID	PRODUCT_ID	QUANTITY	ORDER_DATE
1	10	106	10	1	2024-05-11
2	9	103	9	2	2024-05-10
3	8	101	8	1	2024-05-09
4	7	105	7	3	2024-05-08
5	6	104	4	1	2024-05-07
6	5	102	6	2	2024-05-06

-- 6. Add a data constraint to ensure that the QUANTITY in the ORDERS table is always greater than zero (Already included in table creation)

-- 7. Calculate the total revenue generated by each customer

```
SELECT O.CUSTOMER_ID, SUM(O.QUANTITY * P.PRICE) AS TOTAL_REVENUE
FROM ORDERS O
```

```
JOIN PRODUCTS P ON O.PRODUCT_ID = P.PRODUCT_ID
GROUP BY O.CUSTOMER_ID;
```

Results | Messages

	CUSTOMER_ID	TOTAL_REVENUE
1	101	2250.00
2	102	400.00
3	103	1800.00
4	104	200.00
5	105	90.00
6	106	80.00

Q2. Create a table STUDENT and EXAM with the following fields:

STUDENT TABLE: {STUDENT_ID, Name, Gender, Age, Class ('10th Grade', '11th Grade', '12th Grade')}
EXAM: {STUDENT_ID, MATH_SCORE, SCIENCE_SCORE, ENGLISH_SCORE}

Perform the following queries in SQL:
i. Insert at least 10 records in the tables. (2)
ii. Display the details of male students who are in '12th Grade'. (1)
iii. Display the details of the student who secured the highest total score. (2)
iv. Add a new Column HISTORY_SCORE in EXAM table and modify the table by inserting values to HISTORY_SCORE for the records. (1)
v. List Top 3 students of '11th Grade' based on total score. (1)
vi. Display the Average Age of students in '10th Grade'. (1)
vii. Display the list of students who scored more than the average marks in MATH_SCORE. (2)

Solution 2:

- Creating the STUDENT table

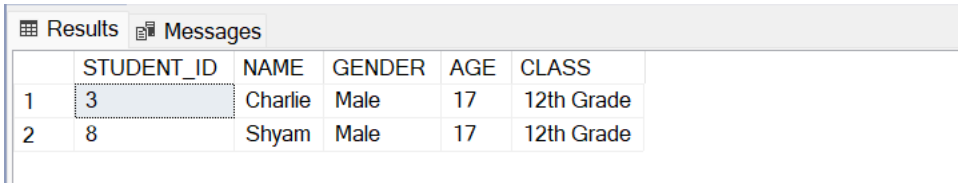
```
CREATE TABLE STUDENT (
    STUDENT_ID INT PRIMARY KEY,
    NAME VARCHAR(100),
    GENDER VARCHAR(10),
    AGE INT,
    CLASS VARCHAR(20)
);
```

```
-- Creating the EXAM table
CREATE TABLE EXAM (
    STUDENT_ID INT,
    MATH_SCORE INT,
```

```
    SCIENCE_SCORE INT,
    ENGLISH_SCORE INT,
    HISTORY_SCORE INT, -- Newly added column
    FOREIGN KEY (STUDENT_ID) REFERENCES STUDENT(STUDENT_ID)
);

-- Inserting at least 10 records into STUDENT
INSERT INTO STUDENT (STUDENT_ID, NAME, GENDER, AGE, CLASS) VALUES
(1, 'Sam', 'Female', 15, '10th Grade'),
(2, 'Ram', 'Male', 16, '11th Grade'),
(3, 'Charlie', 'Male', 17, '12th Grade'),
(4, 'David', 'Male', 16, '11th Grade'),
(5, 'Emma', 'Female', 17, '12th Grade'),
(6, 'Frank', 'Male', 15, '10th Grade'),
(7, 'Sia', 'Female', 16, '11th Grade'),
(8, 'Shyam', 'Male', 17, '12th Grade'),
(9, 'Jenny', 'Female', 15, '10th Grade'),
(10, 'Jack', 'Male', 16, '11th Grade');

-- Inserting at least 10 records into EXAM
INSERT INTO EXAM (STUDENT_ID, MATH_SCORE, SCIENCE_SCORE, ENGLISH_SCORE, HISTORY_SCORE)
VALUES
(1, 85, 78, 90, 88),
(2, 88, 80, 85, 82),
(3, 92, 85, 88, 90),
(4, 80, 75, 78, 79),
(5, 89, 84, 86, 85),
(6, 76, 70, 72, 74),
(7, 91, 88, 90, 92),
(8, 85, 80, 78, 83),
(9, 79, 72, 75, 77),
(10, 87, 83, 86, 85);
```

-- ii. Display the details of male students who are in ‘12th Grade’

```
SELECT * FROM STUDENT WHERE GENDER = 'Male' AND CLASS = '12th Grade';
```



Results | Messages

	STUDENT_ID	NAME	GENDER	AGE	CLASS
1	3	Charlie	Male	17	12th Grade
2	8	Shyam	Male	17	12th Grade

-- iii. Display the details of the student who secured the highest total score

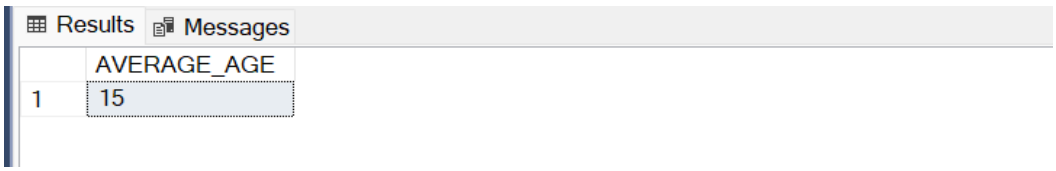
```
SELECT S.*, (E.MATH_SCORE + E.SCIENCE_SCORE + E.ENGLISH_SCORE + E.HISTORY_SCORE) AS TOTAL_SCORE
FROM STUDENT S
JOIN EXAM E ON S.STUDENT_ID = E.STUDENT_ID
ORDER BY TOTAL_SCORE DESC
LIMIT 1;
```

-- v. List Top 3 students of ‘11th Grade’ based on total score

```
SELECT S.*, (E.MATH_SCORE + E.SCIENCE_SCORE + E.ENGLISH_SCORE + E.HISTORY_SCORE) AS TOTAL_SCORE
FROM STUDENT S
JOIN EXAM E ON S.STUDENT_ID = E.STUDENT_ID
WHERE CLASS = '11th Grade'
ORDER BY TOTAL_SCORE DESC
LIMIT 3;
```

-- vi. Display the Average Age of students in ‘10th Grade’

```
SELECT AVG(AGE) AS AVERAGE_AGE FROM STUDENT WHERE CLASS = '10th Grade';
```



Results | Messages

	AVERAGE_AGE
1	15

-- vii. Display the list of students who scored more than the average marks in MATH_SCORE

```
SELECT S.* FROM STUDENT S
JOIN EXAM E ON S.STUDENT_ID = E.STUDENT_ID
WHERE E.MATH_SCORE > (SELECT AVG(MATH_SCORE) FROM EXAM);
```

Results | Messages

	STUDENT_ID	NAME	GENDER	AGE	CLASS
1	2	Ram	Male	16	11th Grade
2	3	Charlie	Male	17	12th Grade
3	5	Emma	Female	17	12th Grade
4	7	Sia	Female	16	11th Grade
5	10	Jack	Male	16	11th Grade

Q3. Create a table named SALES and ITEMS with the following fields:

SALES: {SALE_ID, CUSTOMER_ID, ITEM_ID, UNITS_SOLD, SALE_DATE}
ITEMS: {ITEM_ID, ITEM_NAME, UNIT_PRICE}

Perform the following queries in SQL:

i. **Insert at least 10 records into the SALES and ITEMS tables. (2 marks)**
ii. **Count the number of sales made by each customer. (1 mark)**
iii. **Calculate the total units sold for each item. (1 mark)**
iv. **Find the top 3 items that generated the highest revenue. (2 marks)**
v. **Display the details of sales made on or after '2024-06-01' sorted by SALE_DATE in descending order. (1 mark)**
vi. **Add a data constraint to ensure that the UNITS_SOLD in the SALES table is always greater than zero. (1 mark)**
vii. **Calculate the total revenue generated by each customer. (2 marks)**

Solution 3:

Creating the ITEMS table

```
CREATE TABLE ITEMS (
    ITEM_ID INT PRIMARY KEY,
    ITEM_NAME VARCHAR(100),
    UNIT_PRICE DECIMAL(10,2)
);


-- Creating the SALES table
CREATE TABLE SALES (
    SALE_ID INT PRIMARY KEY,
    CUSTOMER_ID INT,
    ITEM_ID INT,
```

```
    UNITS_SOLD INT CHECK (UNITS_SOLD > 0), -- Constraint to ensure units sold is always greater than zero
    SALE_DATE DATE,
    FOREIGN KEY (ITEM_ID) REFERENCES ITEMS(ITEM_ID)
);

-- Inserting at least 10 records into ITEMS
INSERT INTO ITEMS (ITEM_ID, ITEM_NAME, UNIT_PRICE) VALUES
(1, 'Television', 900.00),
(2, 'Smartphone', 500.00),
(3, 'Tablet', 300.00),
(4, 'Smartwatch', 200.00),
(5, 'Headphones', 100.00),
(6, 'Keyboard', 50.00),
(7, 'Mouse', 30.00),
(8, 'Monitor', 250.00),
(9, 'Printer', 150.00),
(10, 'Webcam', 80.00);

-- Inserting at least 10 records into SALES
INSERT INTO SALES (SALE_ID, CUSTOMER_ID, ITEM_ID, UNITS_SOLD, SALE_DATE) VALUES
(1, 101, 1, 2, '2024-06-01'),
(2, 102, 3, 1, '2024-06-02'),
(3, 103, 2, 3, '2024-06-03'),
(4, 101, 5, 4, '2024-06-04'),
(5, 102, 6, 2, '2024-06-06'),
(6, 104, 4, 1, '2024-06-07'),
(7, 105, 7, 3, '2024-06-08'),
(8, 101, 8, 1, '2024-06-09'),
(9, 103, 9, 2, '2024-06-10'),
(10, 106, 10, 1, '2024-06-11');

-- ii. Count the number of sales made by each customer
SELECT CUSTOMER_ID, COUNT(*) AS TOTAL_SALES
```

```
FROM SALES
GROUP BY CUSTOMER_ID;
```

Results | Messages

	CUSTOMER_ID	TOTAL_SALES
1	101	3
2	102	2
3	103	2
4	104	1
5	105	1
6	106	1

```
-- iii. Calculate the total units sold for each item
SELECT ITEM_ID, SUM(UNITS_SOLD) AS TOTAL_UNITS_SOLD
FROM SALES
GROUP BY ITEM_ID;
```

Results | Messages

	ITEM_ID	TOTAL_UNITS_SOLD
1	1	2
2	2	3
3	3	1
4	4	1
5	5	4
6	6	2
7	7	3
8	8	1
9	9	2
10	10	1

```
-- iv. Find the top 3 items that generated the highest revenue
SELECT I.ITEM_ID, I.ITEM_NAME, SUM(S.UNITS_SOLD * I.UNIT_PRICE) AS TOTAL_REVENUE
FROM SALES S
JOIN ITEMS I ON S.ITEM_ID = I.ITEM_ID
GROUP BY I.ITEM_ID, I.ITEM_NAME
ORDER BY TOTAL_REVENUE DESC
LIMIT 3;

-- v. Display the details of sales made on or after '2024-06-01' sorted by SALE_DATE in descending order
```

SELECT * FROM SALES

WHERE SALE_DATE >= '2024-06-01'

ORDER BY SALE_DATE DESC;

Results Messages

	SALE_ID	CUSTOMER_ID	ITEM_ID	UNITS_SOLD	SALE_DATE
1	10	106	10	1	2024-06-11
2	9	103	9	2	2024-06-10
3	8	101	8	1	2024-06-09
4	7	105	7	3	2024-06-08
5	6	104	4	1	2024-06-07
6	5	102	6	2	2024-06-06
7	4	101	5	4	2024-06-04
8	3	103	2	3	2024-06-03
9	2	102	3	1	2024-06-02
10	1	101	1	2	2024-06-01

-- vi. Add a data constraint to ensure that the UNITS_SOLD in the SALES table is always greater than zero (Already included in table creation)

-- vii. Calculate the total revenue generated by each customer

SELECT S.CUSTOMER_ID, SUM(S.UNITS_SOLD * I.UNIT_PRICE) AS TOTAL_REVENUE

FROM SALES S

JOIN ITEMS I ON S.ITEM_ID = I.ITEM_ID

GROUP BY S.CUSTOMER_ID;

Results Messages

	CUSTOMER_ID	TOTAL_REVENUE
1	101	2450.00
2	102	400.00
3	103	1800.00
4	104	200.00
5	105	90.00
6	106	80.00

Q4. Create a table EMPLOYEES and DEPARTMENTS with the following fields:

EMPLOYEES: {EMPLOYEE_ID, NAME, DEPARTMENT_ID, SALARY, JOIN_DATE}
DEPARTMENTS: {DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID}

Perform the following queries in SQL:

i. **Insert at least 5 records into the EMPLOYEES and DEPARTMENTS tables. (1 mark)**
ii. **Display the names of all employees. (1 mark)**
iii. **Display the department names and their corresponding manager IDs. (1 mark)**
iv. **Find the employee with the highest salary. (1 mark)**
v. **Display the details of employees who joined in the year 2024. (1 mark)**
vi. **Ensure that the SALARY column in the EMPLOYEES table does not accept negative values. (1 mark)**
vii. **Calculate the total number of employees. (1 mark)**
viii. **List the names of employees along with their department names. (2 marks)**
ix. **Display the details of departments that have no employees. (1 mark)**
x. **Update the salary of an employee with EMPLOYEE_ID 1 to 60000. (1 mark)**

Solution 4 :

--1. Table Creation

```
CREATE TABLE DEPARTMENTS (
   DEPARTMENT_ID INT PRIMARY KEY,
   DEPARTMENT_NAME VARCHAR(100),
   MANAGER_ID INT
);

CREATE TABLE EMPLOYEES (
   EMPLOYEE_ID INT PRIMARY KEY,
   NAME VARCHAR(100),
   DEPARTMENT_ID INT,
   SALARY DECIMAL(10, 2) CHECK (SALARY >= 0),
   JOIN_DATE DATE,
   FOREIGN KEY (DEPARTMENT_ID) REFERENCES DEPARTMENTS(DEPARTMENT_ID)
);
```

--2. Insert Records

```
INSERT INTO DEPARTMENTS VALUES (1, 'HR', 101);
INSERT INTO DEPARTMENTS VALUES (2, 'Engineering', 102);
INSERT INTO DEPARTMENTS VALUES (3, 'Sales', 103);
INSERT INTO DEPARTMENTS VALUES (4, 'Marketing', 104);
INSERT INTO DEPARTMENTS VALUES (5, 'Support', 105);

INSERT INTO EMPLOYEES VALUES (1, 'Alice', 1, 50000, '2023-06-01');
INSERT INTO EMPLOYEES VALUES (2, 'Bob', 2, 60000, '2024-01-15');
INSERT INTO EMPLOYEES VALUES (3, 'Charlie', 3, 70000, '2022-09-10');
INSERT INTO EMPLOYEES VALUES (4, 'Diana', 2, 80000, '2024-03-20');
INSERT INTO EMPLOYEES VALUES (5, 'Eve', 4, 55000, '2021-11-05');
```

--3. SQL Queries

```
--ii. Display the names of all employees.
SELECT NAME FROM EMPLOYEES;
```

Results Messages

	NAME
1	Alice
2	Bob
3	Charlie
4	Diana
5	Eve

--iii. Display the department names and their corresponding manager IDs.
SELECT DEPARTMENT_NAME, MANAGER_ID FROM DEPARTMENTS;

Results Messages

	DEPARTMENT_NAME	MANAGER_ID
1	HR	101
2	Engineering	102
3	Sales	103
4	Marketing	104
5	Support	105

--iv. Find the employee with the highest salary.
SELECT * FROM EMPLOYEES WHERE SALARY = (SELECT MAX(SALARY) FROM EMPLOYEES);

Results Messages

	EMPLOYEE_ID	NAME	DEPARTMENT_ID	SALARY	JOIN_DATE
1	4	Diana	2	80000.00	2024-03-20

--v. Display the details of employees who joined in the year 2024.
SELECT * FROM EMPLOYEES WHERE YEAR(JOIN_DATE) = 2024;

Results Messages

	EMPLOYEE_ID	NAME	DEPARTMENT_ID	SALARY	JOIN_DATE
1	2	Bob	2	60000.00	2024-01-15
2	4	Diana	2	80000.00	2024-03-20

--vi. Ensure that the SALARY column in the EMPLOYEES table does not accept negative values.
-- Already included as CHECK (SALARY >= 0) in table definition

--vii. Calculate the total number of employees.
SELECT COUNT(*) AS Total_Employees FROM EMPLOYEES;

--viii. List the names of employees along with their department names.
SELECT E.NAME, D.DEPARTMENT_NAME
FROM EMPLOYEES E
JOIN DEPARTMENTS D ON E.DEPARTMENT_ID = D.DEPARTMENT_ID;

Results Messages

	NAME	DEPARTMENT_NAME
1	Alice	HR
2	Bob	Engineering
3	Charlie	Sales
4	Diana	Engineering
5	Eve	Marketing

--ix. Display the details of departments that have no employees.

```
SELECT * FROM DEPARTMENTS D
WHERE NOT EXISTS (
   SELECT 1 FROM EMPLOYEES E
   WHERE E.DEPARTMENT_ID = D.DEPARTMENT_ID
);
```

Results Messages

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID
1	5	Support	105

--x. Update the salary of an employee with EMPLOYEE_ID 1 to 60000.

```
UPDATE EMPLOYEES SET SALARY = 60000 WHERE EMPLOYEE_ID = 1;
```

Q5. Create a table EMPLOYEES and DEPARTMENTS with the following fields:

EMPLOYEES: {EMPLOYEE_ID, NAME, DEPARTMENT_ID, SALARY, JOIN_DATE}
DEPARTMENTS: {DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID}

Perform the following queries in SQL:

i. **Insert at least 10 records into the EMPLOYEES and DEPARTMENTS tables. (2 marks)**
ii. **Count the number of employees in each department. (1 mark)**
iii. **Calculate the total salary paid to employees in each department. (1 mark)**
iv. **Find the top 3 departments with the highest total salary expenditure. (2 marks)**
v. **Display the details of employees who joined on or after '2024-01-01' sorted by JOIN_DATE in descending order. (1 mark)**
vi. **Add a data constraint to ensure that the SALARY in the EMPLOYEES table is always greater than zero. (1 mark)**
vii. **Calculate the average salary of employees managed by each manager. (2 marks)**

Solution 5:

1. Table Creation

```
CREATE TABLE DEPARTMENTS (
   DEPARTMENT_ID INT PRIMARY KEY,
   DEPARTMENT_NAME VARCHAR(100),
   MANAGER_ID INT
```

```
);

CREATE TABLE EMPLOYEES (
   EMPLOYEE_ID INT PRIMARY KEY,
   NAME VARCHAR(100),
   DEPARTMENT_ID INT,
   SALARY DECIMAL(10, 2) CHECK (SALARY > 0),
   JOIN_DATE DATE,
   FOREIGN KEY (DEPARTMENT_ID) REFERENCES DEPARTMENTS(DEPARTMENT_ID)
);
```

2. Insert Records

```
INSERT INTO DEPARTMENTS VALUES (1, 'HR', 201);
INSERT INTO DEPARTMENTS VALUES (2, 'Engineering', 202);
INSERT INTO DEPARTMENTS VALUES (3, 'Sales', 203);
INSERT INTO DEPARTMENTS VALUES (4, 'Marketing', 204);
INSERT INTO DEPARTMENTS VALUES (5, 'Support', 205);

INSERT INTO EMPLOYEES VALUES (1, 'Alice', 1, 50000, '2023-06-01');
INSERT INTO EMPLOYEES VALUES (2, 'Bob', 2, 60000, '2024-01-15');
INSERT INTO EMPLOYEES VALUES (3, 'Charlie', 3, 70000, '2022-09-10');
INSERT INTO EMPLOYEES VALUES (4, 'Diana', 2, 80000, '2024-03-20');
INSERT INTO EMPLOYEES VALUES (5, 'Eve', 4, 55000, '2021-11-05');
INSERT INTO EMPLOYEES VALUES (6, 'Frank', 5, 62000, '2023-12-01');
INSERT INTO EMPLOYEES VALUES (7, 'Grace', 3, 58000, '2024-02-01');
INSERT INTO EMPLOYEES VALUES (8, 'Heidi', 1, 47000, '2023-05-10');
INSERT INTO EMPLOYEES VALUES (9, 'Ivan', 5, 51000, '2024-04-18');
INSERT INTO EMPLOYEES VALUES (10, 'Judy', 4, 49000, '2022-08-21');
```

3. SQL Queries

ii. Count the number of employees in each department.

```
SELECT DEPARTMENT_ID, COUNT(*) AS Num_Employees
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID;
```

Results | Messages

	DEPARTMENT_ID	Num_Employees
1	1	2
2	2	2
3	3	2
4	4	2
5	5	2

ii. Calculate the total salary paid to employees in each department.

```
SELECT DEPARTMENT_ID, SUM(SALARY) AS Total_Salary
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID;
```

Results Messages

	DEPARTMENT_ID	Total_Salary
1	1	97000.00
2	2	140000.00
3	3	128000.00
4	4	104000.00
5	5	113000.00

iv. Find the top 3 departments with the highest total salary expenditure.

```
SELECT DEPARTMENT_ID, SUM(SALARY) AS Total_Salary
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
ORDER BY Total_Salary DESC
LIMIT 3;
```

v. Display the details of employees who joined on or after '2024-01-01' sorted by JOIN_DATE in descending order.

```
SELECT * FROM EMPLOYEES
WHERE JOIN_DATE >= '2024-01-01'
ORDER BY JOIN_DATE DESC;
```

vi. Add a data constraint to ensure that the SALARY in the EMPLOYEES table is always greater than zero.

```
-- Already included as CHECK (SALARY > 0) in the table definition.
```

vii. Calculate the average salary of employees managed by each manager.

```
SELECT D.MANAGER_ID, AVG(E.SALARY) AS Avg_Salary
FROM EMPLOYEES E
JOIN DEPARTMENTS D ON E.DEPARTMENT_ID = D.DEPARTMENT_ID
GROUP BY D.MANAGER_ID;
```

);

Results Messages

	MANAGER_ID	Avg_Salary
1	201	48500.000000
2	202	70000.000000
3	203	64000.000000
4	204	52000.000000
5	205	56500.000000

Q6. Create a table PATIENTS and APPOINTMENTS with the following fields:

PATIENTS: {PATIENT_ID, NAME, AGE, GENDER}
APPOINTMENTS: {APPOINTMENT_ID, PATIENT_ID, DOCTOR, APPOINTMENT_DATE, FEES}

Perform the following queries in SQL:

i. **Insert at least 5 records into the PATIENTS and APPOINTMENTS tables. (1 mark)**
ii. **Display the names of all patients. (1 mark)**

iii. **Display the details of appointments for a particular doctor (e.g., 'Dr. Smith'). (1 mark)**
iv. **Find the patient with the highest number of appointments. (1 mark)**
v. **Display the details of appointments that occurred in the last month. (1 mark)**
vi. **Ensure that the FEES column in the APPOINTMENTS table does not accept negative values. (1 mark)**
vii. **Calculate the total number of appointments. (1 mark)**
viii. **List the names of patients along with their appointment details. (2 marks)**
ix. **Display the details of patients who have not had any appointments. (1 mark)**
x. **Update the appointment date for a specific appointment (e.g., APPOINTMENT_ID 1) to a new date. (1 mark)**

Solution 6:

```
--Creating the PATIENTS table
CREATE TABLE PATIENTS (
   PATIENT_ID INT PRIMARY KEY,
   NAME VARCHAR(100),
   AGE INT,
   GENDER VARCHAR(10)
);
-- Creating the APPOINTMENTS table
CREATE TABLE APPOINTMENTS (
   APPOINTMENT_ID INT PRIMARY KEY,
   PATIENT_ID INT,
   DOCTOR VARCHAR(100),
   APPOINTMENT_DATE DATE,
   FEES DECIMAL(10,2) CHECK (FEES >= 0), -- Constraint to ensure fees are not negative
   FOREIGN KEY (PATIENT_ID) REFERENCES PATIENTS(PATIENT_ID)
);
-- i. Inserting at least 5 records into PATIENTS and APPOINTMENTS
INSERT INTO PATIENTS (PATIENT_ID, NAME, AGE, GENDER) VALUES
(1, 'Alice Johnson', 30, 'Female'),
(2, 'Bob Smith', 45, 'Male'),
(3, 'Charlie Brown', 29, 'Male'),
(4, 'Diana Green', 50, 'Female'),
(5, 'Ethan White', 35, 'Male');
```

```
INSERT INTO APPOINTMENTS (APPOINTMENT_ID, PATIENT_ID, DOCTOR, APPOINTMENT_DATE, FEES)
VALUES
(1, 1, 'Dr. Smith', '2024-05-10', 200.00),
(2, 2, 'Dr. Lee', '2024-05-15', 150.00),
(3, 3, 'Dr. Smith', '2024-05-20', 250.00),
(4, 4, 'Dr. Adams', '2024-05-22', 180.00),
(5, 5, 'Dr. Lee', '2024-05-25', 220.00);

-- ii. Display the names of all patients
SELECT NAME FROM PATIENTS;
```

Results Messages

	NAME
1	Alice Johnson
2	Bob Smith
3	Charlie Brown
4	Diana Green
5	Ethan White

```
-- iii. Display the details of appointments for a particular doctor (e.g., 'Dr. Smith')
SELECT * FROM APPOINTMENTS WHERE DOCTOR = 'Dr. Smith';
```

Results Messages

	APPOINTMENT_ID	PATIENT_ID	DOCTOR	APPOINTMENT_DATE	FEES
1	1	1	Dr. Smith	2024-05-10	200.00
2	3	3	Dr. Smith	2024-05-20	250.00

```
-- iv. Find the patient with the highest number of appointments
SELECT P.PATIENT_ID, P.NAME, COUNT(A.APPOINTMENT_ID) AS TOTAL_APPOINTMENTS
FROM PATIENTS P
JOIN APPOINTMENTS A ON P.PATIENT_ID = A.PATIENT_ID
GROUP BY P.PATIENT_ID, P.NAME
ORDER BY TOTAL_APPOINTMENTS DESC
LIMIT 1;

-- v. Display the details of appointments that occurred in the last month
```

```
SELECT * FROM APPOINTMENTS

WHERE APPOINTMENT_DATE >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH);


-- vi. Ensure that the FEES column in the APPOINTMENTS table does not accept negative values (Already included in table creation)


-- vii. Calculate the total number of appointments

SELECT COUNT(*) AS TOTAL_APPOINTMENTS FROM APPOINTMENTS;
```

Results Messages

	TOTAL_APPOINTMENTS
1	5

```
-- viii. List the names of patients along with their appointment details

SELECT P.NAME, A.*

FROM PATIENTS P

LEFT JOIN APPOINTMENTS A ON P.PATIENT_ID = A.PATIENT_ID;
```

Results Messages

	NAME	APPOINTMENT_ID	PATIENT_ID	DOCTOR	APPOINTMENT_DATE	FEES
1	Alice Johnson	1	1	Dr. Smith	2024-05-10	200.00
2	Bob Smith	2	2	Dr. Lee	2024-05-15	150.00
3	Charlie Brown	3	3	Dr. Smith	2024-05-20	250.00
4	Diana Green	4	4	Dr. Adams	2024-05-22	180.00
5	Ethan White	5	5	Dr. Lee	2024-05-25	220.00

```
-- ix. Display the details of patients who have not had any appointments

SELECT * FROM PATIENTS

WHERE PATIENT_ID NOT IN (SELECT DISTINCT PATIENT_ID FROM APPOINTMENTS);


-- x. Update the appointment date for a specific appointment (e.g., APPOINTMENT_ID 1) to a new date

UPDATE APPOINTMENTS

SET APPOINTMENT_DATE = '2024-06-01'

WHERE APPOINTMENT_ID = 1;
```

Q7. Create a table EMPLOYEES and SALARIES with the following fields:

EMPLOYEES TABLE: {EMPLOYEE_ID, NAME, DEPARTMENT, GENDER, AGE}

SALARIES TABLE: {EMPLOYEE_ID, BASE_SALARY, BONUS, DEDUCTIONS}

Perform the following queries in SQL:

i. Insert at least 10 records in the tables. (2 marks)
ii. Display the details of employees in the 'Finance' department. (1 mark)
iii. Display the details of the employee with the highest net salary. (2 marks)
iv. Add a new column ALLOWANCES in the SALARIES table and modify the table by inserting values to ALLOWANCES for the records. (1 mark)
v. List the top 3 employees with the highest total compensation (BASE_SALARY + BONUS + ALLOWANCES - DEDUCTIONS). (1 mark)
vi. Display the average age of employees in the 'HR' department. (1 mark)

i. **Display the list of employees who have a net salary (BASE_SALARY + BONUS - DEDUCTIONS) greater than the average net salary of all employees. (2 marks)**

Solution 7:

```
----- Creating the EMPLOYEES table
CREATE TABLE EMPLOYEES (
   EMPLOYEE_ID INT PRIMARY KEY,
   NAME VARCHAR(100),
   DEPARTMENT VARCHAR(50),
   GENDER VARCHAR(10),
   AGE INT
);

-- Creating the SALARIES table
CREATE TABLE SALARIES (
   EMPLOYEE_ID INT PRIMARY KEY,
   BASE_SALARY DECIMAL(10,2),
   BONUS DECIMAL(10,2),
   DEDUCTIONS DECIMAL(10,2),
   ALLOWANCES DECIMAL(10,2),
   FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID)
);

-- i. Inserting at least 10 records in EMPLOYEES and SALARIES
```

```
INSERT INTO EMPLOYEES (EMPLOYEE_ID, NAME, DEPARTMENT, GENDER, AGE) VALUES
(1, 'John Doe', 'Finance', 'Male', 35),
(2, 'Jane Smith', 'HR', 'Female', 40),
(3, 'Robert Brown', 'IT', 'Male', 28),
(4, 'Emily Davis', 'Finance', 'Female', 30),
(5, 'Michael Johnson', 'HR', 'Male', 45),
(6, 'Sarah Wilson', 'IT', 'Female', 32),
(7, 'David Lee', 'Marketing', 'Male', 38),
(8, 'Laura Adams', 'Finance', 'Female', 29),
(9, 'James White', 'IT', 'Male', 41),
(10, 'Sophia Green', 'Marketing', 'Female', 36);

INSERT INTO SALARIES (EMPLOYEE_ID, BASE_SALARY, BONUS, DEDUCTIONS, ALLOWANCES) VALUES
(1, 60000, 5000, 2000, 3000),
(2, 55000, 4000, 1500, 2500),
(3, 70000, 6000, 2500, 4000),
(4, 62000, 5200, 2200, 3100),
(5, 58000, 4500, 1800, 2700),
(6, 73000, 7000, 2800, 4200),
(7, 56000, 4300, 1600, 2600),
(8, 61000, 5100, 2100, 3200),
(9, 75000, 7500, 3000, 4500),
(10, 57000, 4400, 1700, 2800);

-- ii. Display the details of employees in the 'Finance' department
SELECT * FROM EMPLOYEES WHERE DEPARTMENT = 'Finance';
```

Results Messages

	EMPLOYEE_ID	NAME	DEPARTMENT	GENDER	AGE
1	1	John Doe	Finance	Male	35
2	4	Emily Davis	Finance	Female	30
3	8	Laura Adams	Finance	Female	29

```
-- iii. Display the details of the employee with the highest net salary
```

```
SELECT E.*, S.*
FROM EMPLOYEES E
JOIN SALARIES S ON E.EMPLOYEE_ID = S.EMPLOYEE_ID
ORDER BY (S.BASE_SALARY + S.BONUS - S.DEDUCTIONS) DESC
LIMIT 1;

-- iv. List the top 3 employees with the highest total compensation
SELECT E.*, (S.BASE_SALARY + S.BONUS + S.ALLOWANCES - S.DEDUCTIONS) AS TOTAL_COMPENSATION
FROM EMPLOYEES E
JOIN SALARIES S ON E.EMPLOYEE_ID = S.EMPLOYEE_ID
ORDER BY TOTAL_COMPENSATION DESC
LIMIT 3;

-- v. Display the average age of employees in the 'HR' department
SELECT AVG(AGE) AS AVERAGE_AGE FROM EMPLOYEES WHERE DEPARTMENT = 'HR';
```

Results Messages

	AVERAGE_AGE
1	42

```
-- vi. Display employees whose net salary is greater than the average net salary
SELECT E.*, S.*
FROM EMPLOYEES E
JOIN SALARIES S ON E.EMPLOYEE_ID = S.EMPLOYEE_ID
WHERE (S.BASE_SALARY + S.BONUS - S.DEDUCTIONS) >
   (SELECT AVG(BASE_SALARY + BONUS - DEDUCTIONS) FROM SALARIES);
```

Results Messages

	EMPLOYEE_ID	NAME	DEPARTMENT	GENDER	AGE	EMPLOYEE_ID	BASE_SALARY	BONUS	DEDUCTIONS	ALLOWANCES
1	3	Robert Brown	IT	Male	28	3	70000.00	6000.00	2500.00	4000.00
2	6	Sarah Wilson	IT	Female	32	6	73000.00	7000.00	2800.00	4200.00
3	9	James White	IT	Male	41	9	75000.00	7500.00	3000.00	4500.00

Q8. Create a table PROFESSOR and PUBLICATION with the following fields:

PROFESSOR TABLE: {PROFESSOR_ID, NAME, DEPARTMENT, AGE, SALARY}
PUBLICATION TABLE: {PUBLICATION_ID, PROFESSOR_ID, TITLE, JOURNAL, YEAR}

Perform the following queries in SQL:

i. Insert at least 10 records into the PROFESSOR and PUBLICATION tables. (2 marks)

ii. Display the details of professors from the 'OPERATIONS RESEARCH' department. (1 mark)

iii. Display the details of the professor with the highest number of publications. (2 marks)

iv. Add a new column EMAIL in the PROFESSOR table and update the table by inserting values for EMAIL for the records. (1 mark)

v. List the top 3 journals with the highest number of publications. (1 mark)

vi. Display the average salary of professors in the 'MATHEMATICS' department. (1 mark)

viii. Display the list of professors who have published in more than 2 journals. (2 marks)

Solution 8:

```
---- Creating the PROFESSOR table
CREATE TABLE PROFESSOR (
    PROFESSOR_ID INT PRIMARY KEY,
    NAME VARCHAR(100),
    DEPARTMENT VARCHAR(100),
    AGE INT,
    SALARY DECIMAL(10,2),
    EMAIL VARCHAR(100)
);

-- Creating the PUBLICATION table
CREATE TABLE PUBLICATION (
    PUBLICATION_ID INT PRIMARY KEY,
    PROFESSOR_ID INT,
    TITLE VARCHAR(255),
    JOURNAL VARCHAR(100),
    YEAR INT,
    FOREIGN KEY (PROFESSOR_ID) REFERENCES PROFESSOR(PROFESSOR_ID)
);
```

-- i. Inserting at least 10 records into PROFESSOR and PUBLICATION

```
INSERT INTO PROFESSOR (PROFESSOR_ID, NAME, DEPARTMENT, AGE, SALARY, EMAIL) VALUES
(1, 'Dr. Alice Brown', 'Mathematics', 45, 90000, 'alice.brown@university.edu'),
(2, 'Dr. Bob Smith', 'Physics', 50, 95000, 'bob.smith@university.edu'),
(3, 'Dr. Charlie Johnson', 'Operations Research', 40, 88000, 'charlie.johnson@university.edu'),
(4, 'Dr. Diana Green', 'Mathematics', 55, 98000, 'diana.green@university.edu'),
(5, 'Dr. Ethan White', 'Computer Science', 42, 87000, 'ethan.white@university.edu'),
(6, 'Dr. Fiona Adams', 'Operations Research', 47, 93000, 'fiona.adams@university.edu'),
(7, 'Dr. George Wilson', 'Mathematics', 60, 102000, 'george.wilson@university.edu'),
(8, 'Dr. Hannah Lee', 'Physics', 53, 91000, 'hannah.lee@university.edu'),
(9, 'Dr. Ian Brown', 'Computer Science', 39, 85000, 'ian.brown@university.edu'),
(10, 'Dr. Julia Scott', 'Operations Research', 44, 92000, 'julia.scott@university.edu');

INSERT INTO PUBLICATION (PUBLICATION_ID, PROFESSOR_ID, TITLE, JOURNAL, YEAR) VALUES
(1, 1, 'Mathematical Theories in AI', 'Journal of Mathematics', 2023),
(2, 2, 'Quantum Mechanics and Computation', 'Physics World', 2022),
(3, 3, 'Optimisation Techniques', 'OR Journal', 2024),
(4, 4, 'Statistical Models', 'Journal of Mathematics', 2021),
(5, 5, 'AI in Computer Vision', 'CS Journal', 2023),
(6, 6, 'Operations Research in Logistics', 'OR Journal', 2022),
(7, 7, 'Advanced Calculus', 'Mathematical Analysis', 2024),
(8, 8, 'Astrophysics Discoveries', 'Physics World', 2023),
(9, 9, 'Machine Learning Innovations', 'CS Journal', 2024),
(10, 10, 'Game Theory Applications', 'OR Journal', 2021);
```

-- ii. Display the details of professors from the 'OPERATIONS RESEARCH' department

```
SELECT * FROM PROFESSOR WHERE DEPARTMENT = 'Operations Research';
```

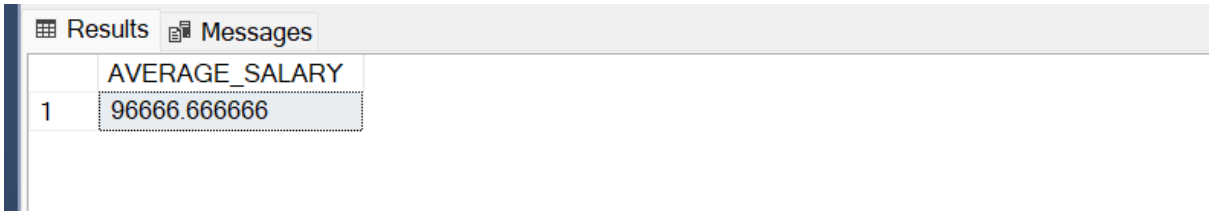
Results | Messages

	PROFESSOR_ID	NAME	DEPARTMENT	AGE	SALARY	EMAIL
1	3	Dr. Charlie Johnson	Operations Research	40	88000.00	charlie.johnson@university.edu
2	6	Dr. Fiona Adams	Operations Research	47	93000.00	fiona.adams@university.edu
3	10	Dr. Julia Scott	Operations Research	44	92000.00	julia.scott@university.edu

-- iii. Display the details of the professor with the highest number of publications

```
SELECT P.PROFESSOR_ID, P.NAME, COUNT(PB.PUBLICATION_ID) AS TOTAL_PUBLICATIONS
```

```
FROM PROFESSOR P
JOIN PUBLICATION PB ON P.PROFESSOR_ID = PB.PROFESSOR_ID
GROUP BY P.PROFESSOR_ID, P.NAME
ORDER BY TOTAL_PUBLICATIONS DESC
LIMIT 1;

-- iv. List the top 3 journals with the highest number of publications
SELECT JOURNAL, COUNT(*) AS PUBLICATION_COUNT
FROM PUBLICATION
GROUP BY JOURNAL
ORDER BY PUBLICATION_COUNT DESC
LIMIT 3;

-- v. Display the average salary of professors in the 'MATHEMATICS' department
SELECT AVG(SALARY) AS AVERAGE_SALARY FROM PROFESSOR WHERE DEPARTMENT = 'Mathematics';
```




```
-- vi. Display the list of professors who have published in more than 2 journals
SELECT P.PROFESSOR_ID, P.NAME, COUNT(DISTINCT PB.JOURNAL) AS JOURNAL_COUNT
FROM PROFESSOR P
JOIN PUBLICATION PB ON P.PROFESSOR_ID = PB.PROFESSOR_ID
GROUP BY P.PROFESSOR_ID, P.NAME
HAVING JOURNAL_COUNT > 2;
```

9. Create a table CUSTOMER and TRANSACTION with the following fields:

CUSTOMER TABLE: {CUSTOMER_ID, NAME, AGE, CITY, PHONE_NUMBER}
TRANSACTION TABLE: {TRANSACTION_ID, CUSTOMER_ID, AMOUNT, TRANSACTION_DATE, TRANSACTION_TYPE}

Perform the following queries in SQL:

i. Insert at least 10 records into the CUSTOMER and TRANSACTION tables. (2 marks)

ii. Display the details of customers living in 'New York'. (1 mark)

iii. Display the details of the customer who made the highest total transaction amount. (2 marks)

iv. Add a new column EMAIL in the CUSTOMER table and update the table by inserting values for EMAIL for the records. (1 mark)

v. List the top 3 customers based on the number of transactions they have made. (1 mark)

vi. Display the average transaction amount of 'Deposit' transactions. (1 mark)

viii. Display the list of customers who have made transactions totalling more than $5000. (2 marks)

Solution 9:

```
CREATE TABLE CUSTOMER (
   CUSTOMER_ID INT PRIMARY KEY,
   NAME VARCHAR(100),
   AGE INT,
   CITY VARCHAR(50),
   PHONE_NUMBER VARCHAR(15)
);

CREATE TABLE TRANSACTION (
   TRANSACTION_ID INT PRIMARY KEY,
   CUSTOMER_ID INT,
   AMOUNT DECIMAL(10, 2),
   TRANSACTION_DATE DATE,
   TRANSACTION_TYPE VARCHAR(20),
   FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER(CUSTOMER_ID)
);
```

i. Insert 10 Records

```
-- CUSTOMER table inserts
INSERT INTO CUSTOMER VALUES
(1, 'Alice', 30, 'New York', '1234567890'),
(2, 'Bob', 25, 'Chicago', '2345678901'),
(3, 'Carol', 40, 'New York', '3456789012'),
(4, 'David', 28, 'Boston', '4567890123'),
(5, 'Eve', 35, 'New York', '5678901234'),
(6, 'Frank', 45, 'Miami', '6789012345'),
(7, 'Grace', 33, 'Dallas', '7890123456'),
(8, 'Heidi', 29, 'San Francisco', '8901234567'),
(9, 'Ivan', 38, 'Seattle', '9012345678'),
(10, 'Judy', 31, 'Los Angeles', '0123456789');

-- TRANSACTION table inserts
```

```
INSERT INTO TRANSACTION VALUES
(101, 1, 1500.00, '2025-04-01', 'Deposit'),
(102, 2, 200.00, '2025-04-02', 'Withdrawal'),
(103, 3, 3000.00, '2025-04-03', 'Deposit'),
(104, 4, 500.00, '2025-04-04', 'Deposit'),
(105, 5, 800.00, '2025-04-05', 'Withdrawal'),
(106, 1, 700.00, '2025-04-06', 'Deposit'),
(107, 3, 2500.00, '2025-04-07', 'Deposit'),
(108, 6, 1000.00, '2025-04-08', 'Deposit'),
(109, 7, 900.00, '2025-04-09', 'Deposit'),
(110, 1, 600.00, '2025-04-10', 'Withdrawal');
```

ii. Customers living in 'New York'

```
SELECT * FROM CUSTOMER
WHERE CITY = 'New York';
```

Results | Messages

	CUSTOMER_ID	NAME	AGE	CITY	PHONE_NUMBER
1	1	Alice	30	New York	1234567890
2	3	Carol	40	New York	3456789012
3	5	Eve	35	New York	5678901234

iii. Customer with highest total transaction amount

```
SELECT C.*
FROM CUSTOMER C
JOIN (
   SELECT CUSTOMER_ID, SUM(AMOUNT) AS TOTAL_AMOUNT
   FROM TRANSACTION
   GROUP BY CUSTOMER_ID
   ORDER BY TOTAL_AMOUNT DESC
   LIMIT 1
) T ON C.CUSTOMER_ID = T.CUSTOMER_ID;
```

iv. Add EMAIL column and update records

```
ALTER TABLE CUSTOMER ADD EMAIL VARCHAR(100);

UPDATE CUSTOMER
SET EMAIL = CASE CUSTOMER_ID
   WHEN 1 THEN 'alice@email.com'
   WHEN 2 THEN 'bob@email.com'
   WHEN 3 THEN 'carol@email.com'
   WHEN 4 THEN 'david@email.com'
   WHEN 5 THEN 'eve@email.com'
   WHEN 6 THEN 'frank@email.com'
   WHEN 7 THEN 'grace@email.com'
   WHEN 8 THEN 'heidi@email.com'
   WHEN 9 THEN 'ivan@email.com'
```

```
    WHEN 10 THEN 'judy@email.com'
END;
```

v. Top 3 customers by transaction count

```
SELECT C.*, COUNT(T.TRANSACTION_ID) AS TRANSACTION_COUNT
FROM CUSTOMER C
JOIN TRANSACTION T ON C.CUSTOMER_ID = T.CUSTOMER_ID
GROUP BY C.CUSTOMER_ID
ORDER BY TRANSACTION_COUNT DESC
LIMIT 3;
```

vi. Average 'Deposit' transaction amount

```
SELECT AVG(AMOUNT) AS AVERAGE_DEPOSIT
FROM TRANSACTION
WHERE TRANSACTION_TYPE = 'Deposit';
```

Results | Messages

	AVERAGE_DEPOSIT
1	1442.857142

vii. Customers with total transactions > $5000

```
SELECT C.*
FROM CUSTOMER C
JOIN (
    SELECT CUSTOMER_ID, SUM(AMOUNT) AS TOTAL_AMOUNT
    FROM TRANSACTION
    GROUP BY CUSTOMER_ID
    HAVING SUM(AMOUNT) > 5000
) T ON C.CUSTOMER_ID = T.CUSTOMER_ID;
```

Results | Messages

	CUSTOMER_ID	NAME	AGE	CITY	PHONE_NUMBER	EMAIL
1	3	Carol	40	New York	3456789012	carol@email.com

10. Create a table BOOK and BORROW with the following fields:

BOOK TABLE: {BOOK_ID, TITLE, AUTHOR, GENRE, PUBLISHED_YEAR}
BORROW TABLE: {BORROW_ID, BOOK_ID, MEMBER_ID, BORROW_DATE, RETURN_DATE}

Perform the following queries in SQL:

i. Insert at least 10 records into the BOOK and BORROW tables. (2 marks)

ii. Display the details of books authored by 'J.K. Rowling'. (1 mark)

iii. Display the details of the book that has been borrowed the most times. (2 marks)

iv. Add a new column ISBN in the BOOK table and update the table by inserting values for ISBN for the records. (1 mark)

v. List the top 3 members based on the number of books they have borrowed. (1 mark)

vi. Display the average number of days books are borrowed for. (1 mark)

vii. Display the list of books borrowed by members who have borrowed more than 5 books. (2 marks

Solution 10:

```
CREATE TABLE BOOK (
   BOOK_ID INT PRIMARY KEY,
   TITLE VARCHAR(100),
   AUTHOR VARCHAR(100),
   GENRE VARCHAR(50),
   PUBLISHED_YEAR INT
);

CREATE TABLE BORROW (
   BORROW_ID INT PRIMARY KEY,
   BOOK_ID INT,
   MEMBER_ID INT,
   BORROW_DATE DATE,
   RETURN_DATE DATE,
   FOREIGN KEY (BOOK_ID) REFERENCES BOOK(BOOK_ID)
);
```

i. Insert 10 Records

```
-- BOOK table inserts
INSERT INTO BOOK VALUES
(1, 'Harry Potter and the Sorcerer''s Stone', 'J.K. Rowling', 'Fantasy', 1997),
(2, 'Harry Potter and the Chamber of Secrets', 'J.K. Rowling', 'Fantasy', 1998),
(3, 'The Hobbit', 'J.R.R. Tolkien', 'Fantasy', 1937),
(4, '1984', 'George Orwell', 'Dystopian', 1949),
(5, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction', 1960),
(6, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction', 1925),
(7, 'The Catcher in the Rye', 'J.D. Salinger', 'Fiction', 1951),
(8, 'The Da Vinci Code', 'Dan Brown', 'Thriller', 2003),
(9, 'The Alchemist', 'Paulo Coelho', 'Fiction', 1988),
(10, 'Harry Potter and the Prisoner of Azkaban', 'J.K. Rowling', 'Fantasy', 1999);

-- BORROW table inserts
INSERT INTO BORROW VALUES
(101, 1, 1001, '2025-04-01', '2025-04-10'),
(102, 2, 1002, '2025-04-03', '2025-04-12'),
```

```
(103, 1, 1003, '2025-04-05', '2025-04-14'),
(104, 3, 1004, '2025-04-06', '2025-04-10'),
(105, 4, 1002, '2025-04-07', '2025-04-15'),
(106, 1, 1001, '2025-04-08', '2025-04-16'),
(107, 5, 1005, '2025-04-09', '2025-04-18'),
(108, 6, 1006, '2025-04-10', '2025-04-13'),
(109, 1, 1007, '2025-04-11', '2025-04-20'),
(110, 7, 1008, '2025-04-12', '2025-04-22');
```

ii. Books authored by 'J.K. Rowling'

```
SELECT * FROM BOOK
WHERE AUTHOR = 'J.K. Rowling';
```

iii. Book borrowed the most times

```
SELECT B.*
FROM BOOK B
JOIN (
   SELECT BOOK_ID, COUNT(*) AS BORROW_COUNT
   FROM BORROW
   GROUP BY BOOK_ID
   ORDER BY BORROW_COUNT DESC
   LIMIT 1
) T ON B.BOOK_ID = T.BOOK_ID;
```

iv. Add ISBN column and update values

```
ALTER TABLE BOOK ADD ISBN VARCHAR(20);

UPDATE BOOK
SET ISBN = CASE BOOK_ID
   WHEN 1 THEN '9780439554930'
   WHEN 2 THEN '9780439064873'
   WHEN 3 THEN '9780547928227'
   WHEN 4 THEN '9780451524935'
   WHEN 5 THEN '9780060935467'
   WHEN 6 THEN '9780743273565'
   WHEN 7 THEN '9780316769488'
   WHEN 8 THEN '9780307474278'
   WHEN 9 THEN '9780061122415'
   WHEN 10 THEN '9780439136365'
END;
```

v. Top 3 members by number of borrowed books

```
SELECT MEMBER_ID, COUNT(*) AS BORROW_COUNT
FROM BORROW
GROUP BY MEMBER_ID
ORDER BY BORROW_COUNT DESC
LIMIT 3;
```

vi. Average number of days books are borrowed

```
SELECT AVG(DATEDIFF(RETURN_DATE, BORROW_DATE)) AS AVERAGE_BORROW_DAYS
FROM BORROW;
```

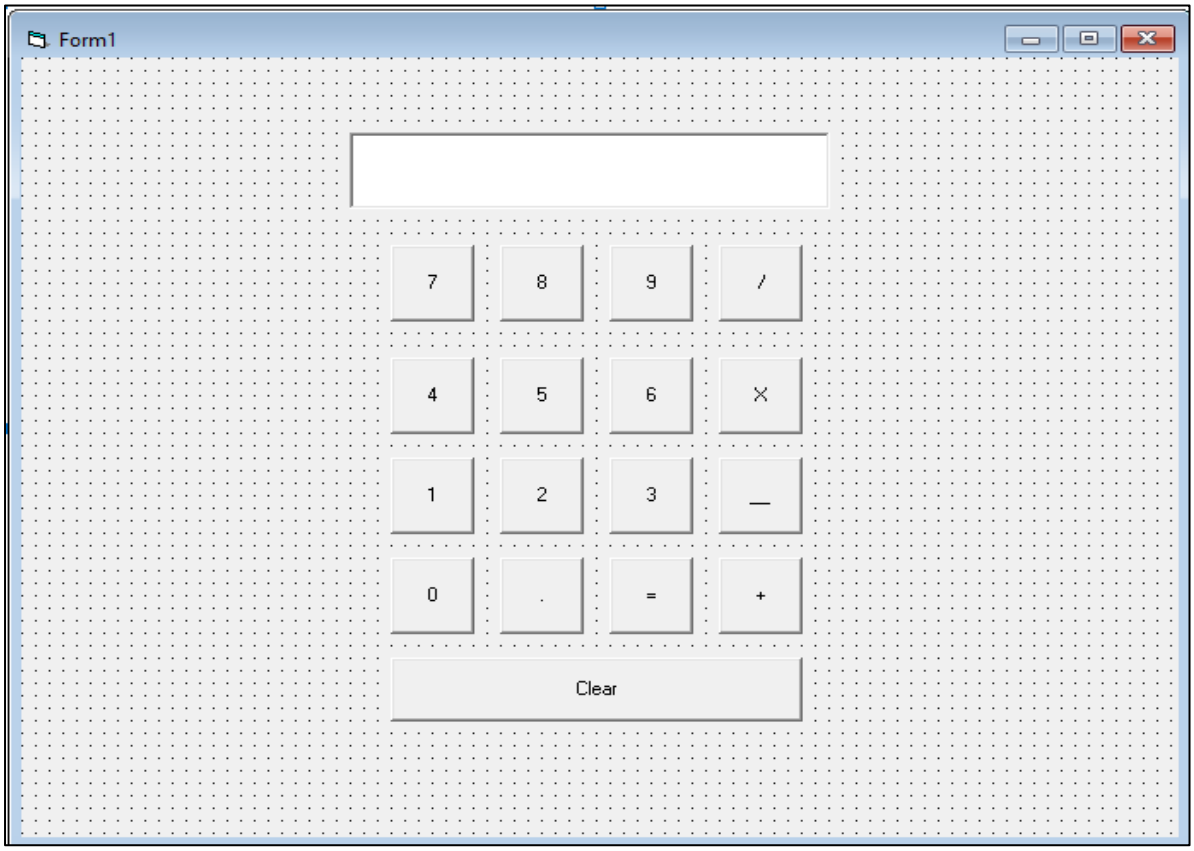
vii. Books borrowed by members who borrowed more than 5 books

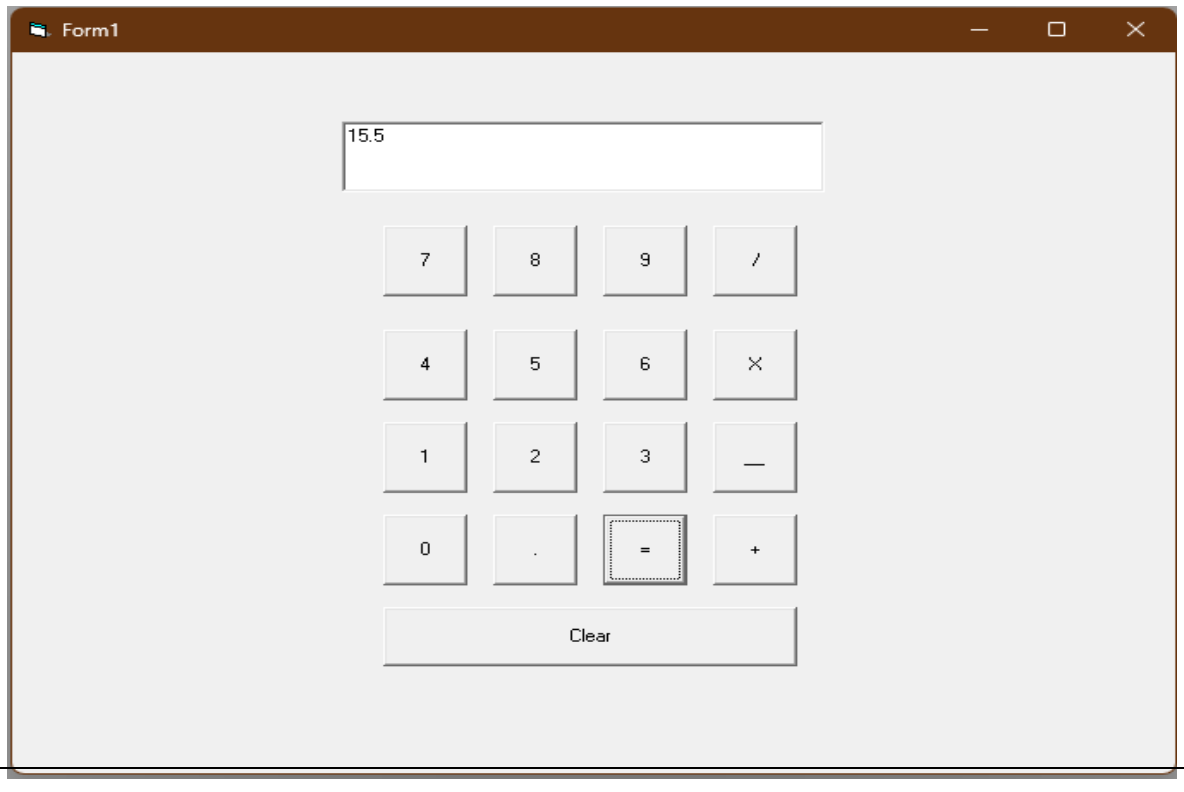
```
SELECT B.*
FROM BOOK B
JOIN BORROW BR ON B.BOOK_ID = BR.BOOK_ID
WHERE BR.MEMBER_ID IN (
   SELECT MEMBER_ID
   FROM BORROW
   GROUP BY MEMBER_ID
   HAVING COUNT(*) > 5
);
```

Visual Basic:

1. **Simple Calculator:**

Create a Simple Calculator in VB.

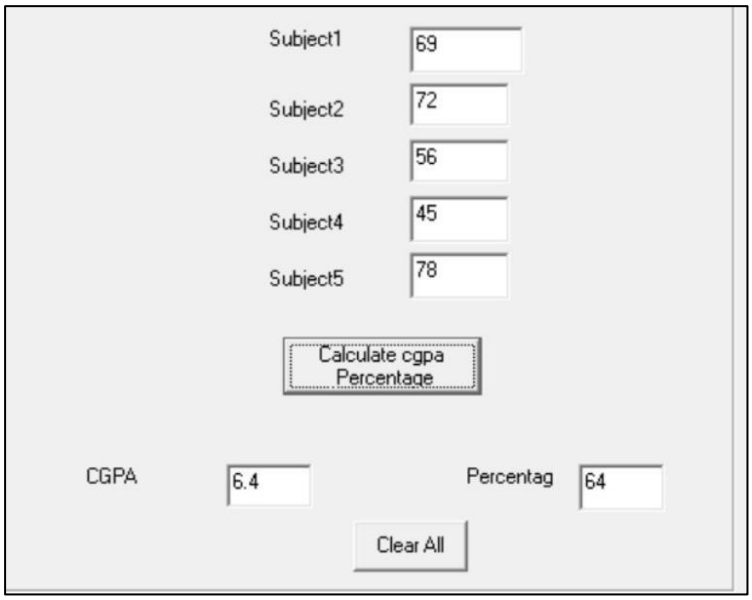




```
Dim n1, n2, res As Double Dim op As String

Private Sub Button0_Click() Text1.Text = Text1.Text & "0"
End Sub

Private Sub Button1_Click() Text1.Text = Text1.Text & "1"
End Sub

Private Sub Button2_Click() Text1.Text = Text1.Text & "2"
End Sub

Private Sub Button3_Click() Text1.Text = Text1.Text & "3"
End Sub

Private Sub Button4_Click() Text1.Text = Text1.Text & "4"
End Sub

Private Sub Button5_Click() Text1.Text = Text1.Text & "5"
End Sub

Private Sub Button6_Click() Text1.Text = Text1.Text & "6"
End Sub

Private Sub Button7_Click() Text1.Text = Text1.Text & "7"
End Sub

Private Sub Button8_Click() Text1.Text = Text1.Text & "8"
End Sub

Private Sub Button9_Click() Text1.Text = Text1.Text & "9"
End Sub

Private Sub ButtonAdd_Click() op = "+"
n1 = Val(Text1.Text) Text1.Text = ""

End Sub

Private Sub ButtonSubtract_Click() op = "-"
n1 = Val(Text1.Text) Text1.Text = ""
End Sub
Private Sub ButtonMultiply_Click()
op = "*"
n1 = Val(Text1.Text) Text1.Text = ""
End Sub

Private Sub ButtonDivide_Click() op = "/"
n1 = Val(Text1.Text) Text1.Text = ""

End Sub

Private Sub ButtonDecimal_Click() Text1.Text = Text1.Text + "."
End Sub

Private Sub ButtonEqual_Click() n2 = Val(Text1.Text)
```

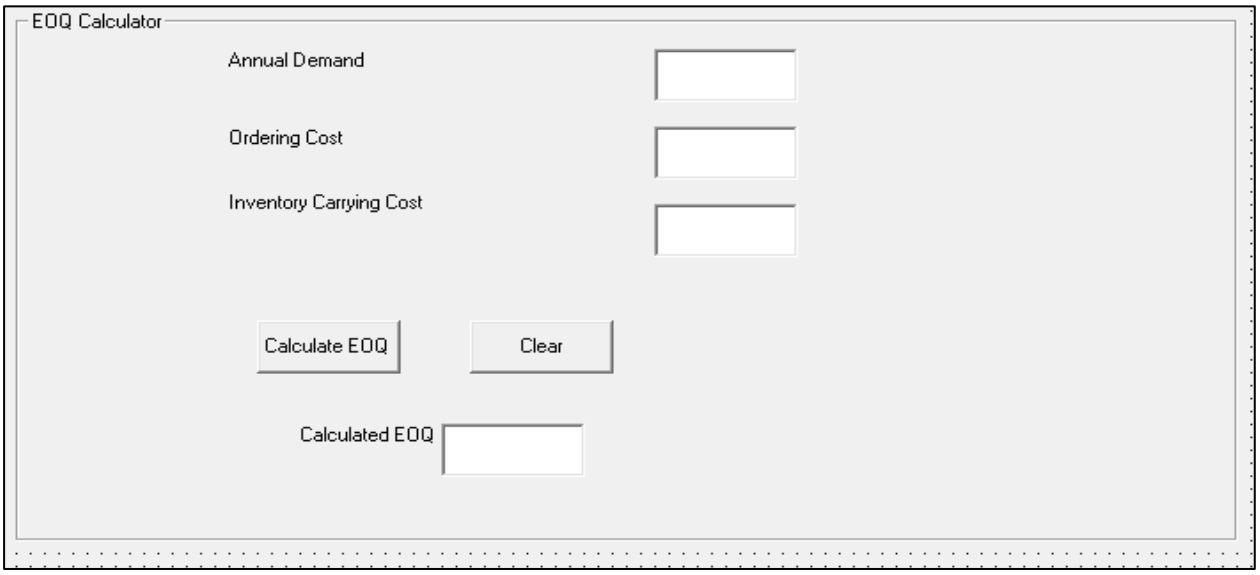
```
Select Case op Case "+"
res = n1 + n2 Case "-"
res = n1 - n2 Case "*"
res = n1 * n2 Case "/"
res = n1 / n2 End Select Text1.Text = res

End Sub

Private Sub ClearButton_Click() Text1.Text = ""

End Sub

Private Sub TextBox_Change()

End Sub
```

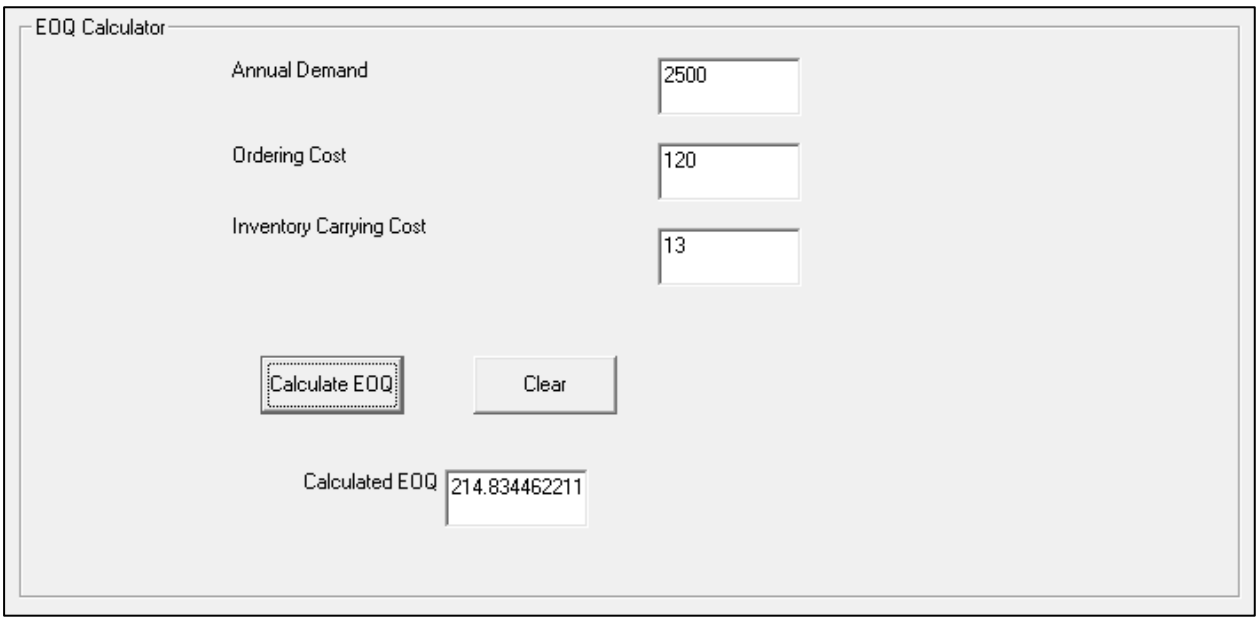
2. Create Form in VB for Percentage and CGPA.



```
Dim s1, s2, s3, s4, s5, cgpa, per As Double
Private Sub Command1_Click()

s1 = Val(Text1.Text)
s2 = Val(Text2.Text)
s3 = Val(Text3.Text)
s4 = Val(Text4.Text)
s5 = Val(Text5.Text)

    per = ((s1 + s2 + s3 + s4 + s5) / 5)
    cgpa = (per / 10)

    Text6.Text = cgpa
    Text7.Text = per

End Sub
```

3. Create Form in VB for EOQ Model.



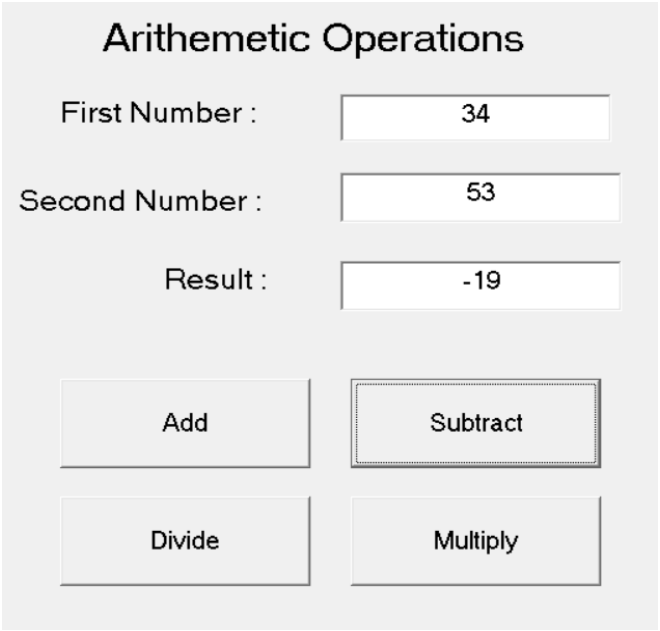



```
Dim lambda, a, ic, eoq As Double
Private Sub Command1_Click()
lambda = Val(Text1.Text)

a = Val(Text2.Text)
ic = Val(Text3.Text)

eoq = Sqr((2 * a * lambda) / ic)
Text4.Text = eoq

End Sub


Private Sub Command2_Click()
Text1.Text = ""

Text2.Text = ""
Text3.Text = ""
Text4.Text = ""


End Sub

 Private Sub Text1_Change() Ad = Val(Text1.Text)

End Sub
```

4. Arithmetic operations on two numbers:

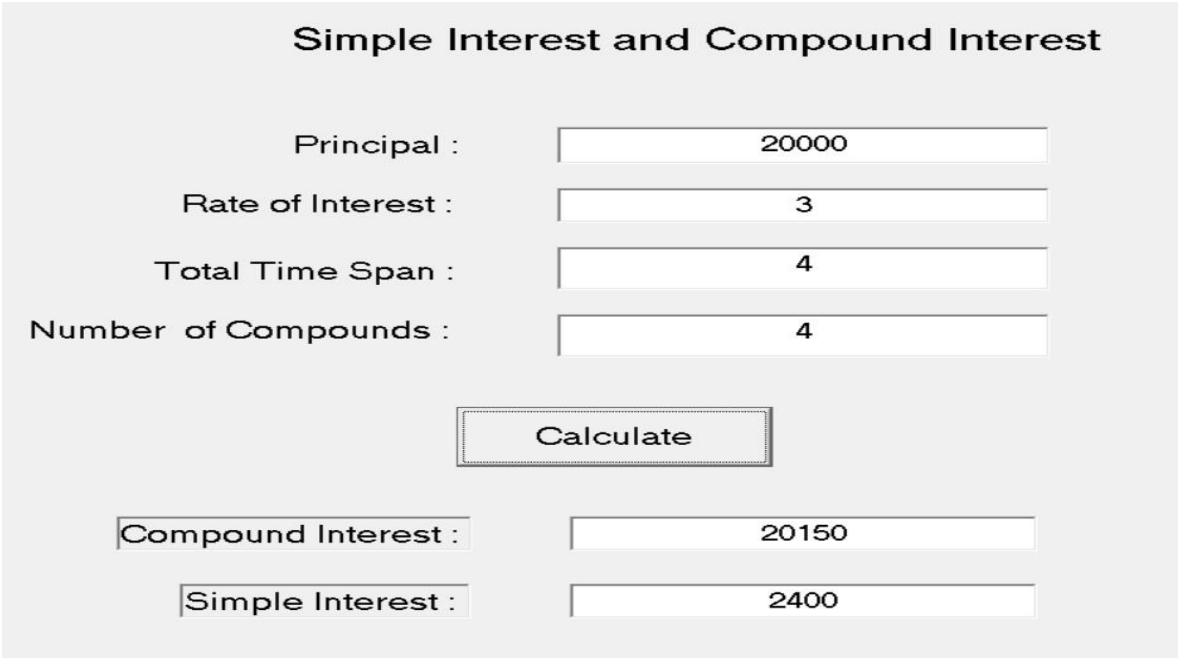
```
Dim a, b, r As Double

Private Sub Command1_Click()

a = Val(Text1.Text)
```

```
b = Val(Text2.Text)
r = a + b
Text3.Text = r
End Sub
Private Sub Command2_Click()
a = Val(Text1.Text)
b = Val(Text2.Text)r = a - b

Text3.Text = r
End Sub
Private Sub Command3_Click()
a = Val(Text1.Text)
b = Val(Text2.Text)
r = a / b
Text3.Text = r
End Sub

Private Sub Command4_Click()
a = Val(Text1.Text)
b = Val(Text2.Text)
r = a * b
Text3.Text = r
End Sub
```



5. Simple and compound interest

```
Dim P, R, T, n, SI, CI As Double
Private Sub Command1_Click()
P = Val(Text1.Text)
R = Val(Text2.Text)
T = Val(Text3.Text)
n = Val(Text4.Text)
CI = P * ((1 + R / (100 * n)))
SI = P * R * T / 100
Text5.Text = CI
Text6.Text = SI
End Sub
```



6. VB and MS-Access Connectivity

```
Private Sub Command1_Click()
Data1.Recordset.AddNew
End Sub
Private Sub Command2_Click()
Data1.Recordset.Update
End Sub

Private Sub Command3_Click()
Data1.Recordset.Delete
End Sub
Private Sub Command4_Click()
Data1.Recordset.MovePrevious
End Sub
Private Sub Command5_Click()
Data1.Recordset.MoveNext
```

```
End Sub
Private Sub Command6_Click()
End
End Sub
```