Q.1 Create a Sales Table and Use Aggregate Functions a) Create a Sales table with columns: SaleID, ProductID, Quantity, SaleAmount, and SaleDate. b) Insert at least 10 sales records with different products and quantities. c) Write a query to calculate the total revenue generated using the SUM function. d) Find the product with the highest sale amount using the MAX function. e) Retrieve the average sale amount per transaction using the AVG function.

Creating the Sales Table:

CREATE TABLE Sales (

SaleID INT PRIMARY KEY,

ProductID INT,

Quantity INT,

SaleAmount DECIMAL(10, 2),

SaleDate DATE

);

2 Inserting Records:

INSERT INTO Sales (SaleID, ProductID, Quantity, SaleAmount, SaleDate)

VALUES

(1, 101, 3, 29.99, '2025-04-01'),

(2, 102, 2, 49.99, '2025-04-02'),

(3, 103, 1, 15.00, '2025-04-03'),

(4, 104, 5, 99.99, '2025-04-04'),

(5, 105, 10, 199.99, '2025-04-05'),

(6, 106, 4, 39.99, '2025-04-06'),

(7, 107, 6, 79.99, '2025-04-07'),

(8, 108, 8, 159.99, '2025-04-08'),

(9, 109, 2, 24.99, '2025-04-09'),

(10, 110, 3, 34.99, '2025-04-10');

3 Total Revenue Calculation:

```
SELECT SUM(SaleAmount) AS TotalRevenue
FROM Sales;
```

4 Product with Highest Sale Amount:

```
SELECT ProductID, MAX(SaleAmount) AS HighestSaleAmount
FROM Sales
GROUP BY ProductID
ORDER BY HighestSaleAmount DESC
LIMIT 1;
```

5 Average Sale Amount Per Transaction:

```
SELECT AVG(SaleAmount) AS AverageSaleAmount
FROM Sales;
```

Q.2 Use DDL and DML Commands a) Create a Products table with columns for ProductID, ProductName, Price, and StockQuantity using DDL commands. b) Insert five product records and display all products using a SELECT query. c) Update the price of a product with ProductID = 3 and check the changes using a SELECT statement. d) Delete a product from the table and verify whether the changes are reflected. e) Alter the table to add a new column Discount and set a default value of 5%.

1 **Create the Products Table**:

```
CREATE TABLE Products (
ProductID INT PRIMARY KEY,
ProductName VARCHAR(100),
Price DECIMAL(10, 2),
StockQuantity INT
);
```

2 **Insert Product Records**:

```
INSERT INTO Products (ProductID, ProductName, Price, StockQuantity)
VALUES
(1, 'Laptop', 999.99, 50),
(2, 'Smartphone', 599.99, 150),
(3, 'Headphones', 199.99, 200),
(4, 'Keyboard', 49.99, 300),
(5, 'Mouse', 29.99, 400);
```

3 Update the Price of a Product:

```
UPDATE Products
SET Price = 179.99
WHERE ProductID = 3;
```

4 Delete a Product:

```
DELETE FROM Products
WHERE ProductID = 4;
```

5 **Alter the Table to Add a Discount Column**:

```
ALTER TABLE Products
ADD Discount DECIMAL(5, 2) DEFAULT 5;
```

Q .3 Create a Customer Table with Integrity Constraints a) Create a Customers table with constraints: CustomerID (PRIMARY KEY), Email (UNIQUE), Age (CHECK Age > 18). b) Insert a valid customer record and verify that the default country is assigned if not explicitly provided. c) Attempt to insert a customer with an age of 16 and observe the CHECK constraint violation. d) Try inserting two customers with the same email ID and observe the UNIQUE constraint violation. e) Retrieve all customers who are older than 25 and belong to a country other than 'India'.

1 **Create the Customers Table**:

```
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY,
Name VARCHAR(100),
```

```
Email VARCHAR(100) UNIQUE,
Age INT CHECK (Age > 18),
Country VARCHAR(100) DEFAULT 'India'
);
```

2 Insert a Valid Customer Record:

```
INSERT INTO Customers (CustomerID, Name, Email, Age)
VALUES (1, 'John Doe', 'john.doe@example.com', 30);
```

3 Attempt to Insert a Customer with Age 16 (CHECK constraint violation)

```
INSERT INTO Customers (CustomerID, Name, Email, Age, Country)
VALUES (2, 'Jane Smith', 'jane.smith@example.com', 16, 'USA');
```

4 Attempt to Insert Two Customers with the Same Email ID (UNIQUE constraint violation):

```
INSERT INTO Customers (CustomerID, Name, Email, Age, Country)
VALUES (3, 'Alice Brown', 'john.doe@example.com', 28, 'UK');
```

5 Retrieve Customers Older Than 25 and Not from 'India':

```
SELECT * FROM Customers
WHERE Age > 25 AND Country <> 'India';
```

Q.4 Create a Table with Constraints a) Create an EmployeeDetails table with EmployeeID as the PRIMARY KEY and DepartmentID as a FOREIGN KEY referencing a Department table. b) Insert a valid employee record with an existing DepartmentID, then attempt to insert an employee with a non-existent DepartmentID and observe the constraint violation. c) Insert an employee with a duplicate EmployeeID and check how the primary key constraint prevents duplicate entries. d) Modify the Salary column to have a UNIQUE constraint and attempt to insert two employees with the same salary to test the constraint. e) Write a query to delete an employee from EmployeeDetails and ensure that the deletion does not violate any referential integrity constraints.

1 **Create the Department and EmployeeDetails Tables**:

```
CREATE TABLE Department (
DepartmentID INT PRIMARY KEY,
DepartmentName VARCHAR(100)
);

CREATE TABLE EmployeeDetails (
EmployeeID INT PRIMARY KEY,
EmployeeName VARCHAR(100),
Salary DECIMAL(10, 2),
DepartmentID INT,
FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)
);
```

2 **Insert a Valid Employee Record** and **Attempt to Insert with Non-Existent**

```
INSERT INTO Department (DepartmentID, DepartmentName)
VALUES (1, 'HR');

INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)
VALUES (1001, 'John Doe', 50000.00, 1);

INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)
VALUES (1002, 'Jane Smith', 60000.00, 999);  -- Foreign key violation
```

3 **Insert Employee with Duplicate EmployeeID** (PRIMARY KEY constraint violation):

```
INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)
VALUES (1001, 'Alice Brown', 55000.00, 1);  -- Duplicate EmployeeID
```

4 **Alter Salary Column to Have a UNIQUE Constraint** and **Test Unique Constraint**:

```
ALTER TABLE EmployeeDetails
```

ADD CONSTRAINT unique_salary UNIQUE (Salary);

INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)

VALUES (1003, 'Bob Green', 50000.00, 1);

INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, Salary, DepartmentID)

VALUES (1004, 'Eve Black', 50000.00, 1); -- Salary already exists

5 **Delete an Employee and Verify**:

DELETE FROM EmployeeDetails

WHERE EmployeeID = 1001;

SELECT * FROM EmployeeDetails WHERE EmployeeID = 1001;

Q.5 Create an Employee Table with Various Columns a) Create a table Employee with attributes: EmployeeID (INT, PRIMARY KEY), Name (VARCHAR), Salary (DECIMAL), JoiningDate (DATE), and ActiveStatus (BOOLEAN). b) Insert five sample employee records and ensure each employee has a unique EmployeeID. c) Write a query to find all employees who joined before January 1, 2023. d) Update the salary of an employee named 'Amit Sharma' by 10% and display the updated record. e) Retrieve all employees who are currently active (ActiveStatus = TRUE).

1 **Create the Employee Table**:

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100),

Salary DECIMAL(10, 2),

JoiningDate DATE,

ActiveStatus BOOLEAN

);

2 **Insert Five Sample Employee Records**:

```
INSERT INTO Employee (EmployeeID, Name, Salary, JoiningDate, ActiveStatus)
VALUES
(1, 'Amit Sharma', 50000.00, '2020-03-15', TRUE),
(2, 'Priya Verma', 60000.00, '2022-01-10', TRUE),
(3, 'Ravi Kumar', 55000.00, '2021-06-20', FALSE),
(4, 'Neha Patel', 70000.00, '2019-07-25', TRUE),
(5, 'Sanjay Singh', 48000.00, '2020-09-30', TRUE);
```

3 **Find All Employees Who Joined Before January 1, 2023**:

```
SELECT * FROM Employee
WHERE JoiningDate < '2023-01-01';
```

4 **Update the Salary of Amit Sharma by 10% and Display the Updated Record**:

```
UPDATE Employee
SET Salary = Salary * 1.10
WHERE Name = 'Amit Sharma';

SELECT * FROM Employee
WHERE Name = 'Amit Sharma';
```

5 **Retrieve All Active Employees**:

```
SELECT * FROM Employee
WHERE ActiveStatus = TRUE;
```

Q.6 Aggregate Functions (on a single table: Create a Sales table with columns: SaleID, ProductID, ProductName, Quantity, Discount, SaleAmount, and SaleDate.) a) From the Sales table, calculate the total sales amount (SUM) generated in the month of February 2025. b) Find the average (AVG) billing amount from the Sales table to assess customer spending behavior. c) Identify the minimum (MIN) quantity of products sold in any transaction using the Sales table. d) Determine the highest (MAX) discount applied on any sale using the Sales table. e) Use the COUNT function to find how many transactions were recorded in the Sales table for the product “Laptop”.

1 **Create the Sales Table**:

```
CREATE TABLE Sales (

SaleID INT PRIMARY KEY,

ProductID INT,

ProductName VARCHAR(100),

Quantity INT,

Discount DECIMAL(5, 2),

SaleAmount DECIMAL(10, 2),

SaleDate DATE

);
```

2 **Insert Sample Data**:

```
INSERT INTO Sales (SaleID, ProductID, ProductName, Quantity, Discount, SaleAmount, SaleDate)

VALUES

(1, 101, 'Laptop', 2, 5.00, 2000.00, '2025-02-10'),

(2, 102, 'Smartphone', 1, 10.00, 600.00, '2025-02-15'),

(3, 103, 'Headphones', 5, 15.00, 500.00, '2025-03-01'),

(4, 104, 'Keyboard', 3, 5.00, 150.00, '2025-02-18'),

(5, 101, 'Laptop', 1, 5.00, 1000.00, '2025-02-20'),

(6, 105, 'Mouse', 4, 0.00, 80.00, '2025-02-10'),

(7, 102, 'Smartphone', 2, 10.00, 1200.00, '2025-03-05');
```

3 **Calculate the Total Sales Amount for February 2025**:

```
SELECT SUM(SaleAmount) AS TotalSalesAmount

FROM Sales

WHERE SaleDate BETWEEN '2025-02-01' AND '2025-02-28';
```

4 **Find the Average Billing Amount**:

```
SELECT AVG(SaleAmount) AS AverageBillingAmount
```

FROM Sales;

5 **Find the Minimum Quantity of Products Sold in Any Transaction**:

SELECT MIN(Quantity) AS MinimumQuantitySold

FROM Sales;

6 **Determine the Highest Discount Applied on Any Sale**:

SELECT MAX(Discount) AS MaximumDiscount

FROM Sales;

7 **Count How Many Transactions Were Recorded for the Product "Laptop"**:

SELECT COUNT(*) AS LaptopTransactionCount

FROM Sales

WHERE ProductName = 'Laptop';

Q.7 Constraints (on a single table: Employees) a) Create the Employees table with EmployeeID as PRIMARY KEY, Email as UNIQUE, and Salary with a CHECK (Salary > 10000) constraint. b) Add a NOT NULL constraint on the Name column in the Employees table and try inserting a record without the name. c) Add a DEFAULT value 'Active' to the Status column in Employees, and insert a record without specifying the status to verify the default. d) Insert a record into Employees where Salary is less than 10000 to test the CHECK constraint. e) Try inserting two employees with the same Email ID to verify the enforcement of the UNIQUE constraint.

1 **Create the Employees Table with Constraints**:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100),

Email VARCHAR(100) UNIQUE,

Salary DECIMAL(10, 2) CHECK (Salary > 10000),

Status VARCHAR(20)

);

2 **Add NOT NULL Constraint to the Name Column and Insert a Record Without Name**:

```
ALTER TABLE Employees
MODIFY COLUMN Name VARCHAR(100) NOT NULL;

-- This insert should fail due to the NOT NULL constraint
INSERT INTO Employees (EmployeeID, Name, Email, Salary, Status)
VALUES (1, NULL, 'john.doe@example.com', 12000.00, 'Active');
```

3 **Add DEFAULT Value 'Active' to the Status Column and Insert a Record Without Specifying the Status**:

```
ALTER TABLE Employees
ADD CONSTRAINT DF_Status DEFAULT 'Active' FOR Status;

-- Insert a record without specifying the Status
INSERT INTO Employees (EmployeeID, Name, Email, Salary)
VALUES (2, 'Jane Smith', 'jane.smith@example.com', 15000.00);
```

4 **Try Inserting a Record with Salary Less Than 10,000 to Test the CHECK Constraint**:

```
-- This insert will fail due to the CHECK constraint on Salary
INSERT INTO Employees (EmployeeID, Name, Email, Salary, Status)
VALUES (3, 'Michael Lee', 'michael.lee@example.com', 9000.00, 'Active');
```

5 **Insert Two Employees with the Same Email to Verify the UNIQUE Constraint**:

```
-- First insert
INSERT INTO Employees (EmployeeID, Name, Email, Salary, Status)
VALUES (4, 'Alice Turner', 'alice.turner@example.com', 15000.00, 'Active');

-- Second insert will fail due to UNIQUE constraint on Email
INSERT INTO Employees (EmployeeID, Name, Email, Salary, Status)
VALUES (5, 'Bob Harris', 'alice.turner@example.com', 16000.00, 'Active');
```

Q.8 DDL and DML Commands a) Use DDL commands to create a Library database and define a Books table with fields: BookID, Title, Author, Genre, and Price. b) Insert at least five sample records into the Books table using INSERT (DML) and verify them using a SELECT query. c) A new column PublicationYear needs to be added. Use ALTER TABLE to modify the existing table structure. d) Update the price of all books published before 2020 by increasing 10% using the UPDATE statement. e) Use DELETE to remove all books where the genre is 'Outdated Technology' and validate the change with a SELECT query.

1 **Create the Library Database and Books Table**:

```
CREATE DATABASE Library;

USE Library;


CREATE TABLE Books (

BookID INT PRIMARY KEY,

Title VARCHAR(255),

Author VARCHAR(255),

Genre VARCHAR(100),

Price DECIMAL(10, 2)

);
```

2 **Insert Sample Data into Books Table**:

```
INSERT INTO Books (BookID, Title, Author, Genre, Price)

VALUES

(1, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', 15.99),

(2, '1984', 'George Orwell', 'Dystopian', 12.99),

(3, 'The Catcher in the Rye', 'J.D. Salinger', 'Classic', 10.50),

(4, 'Introduction to Algorithms', 'Thomas H. Cormen', 'Computer Science', 60.00),

(5, 'The Pragmatic Programmer', 'Andrew Hunt', 'Programming', 45.00);
```

3 **Add a New Column PublicationYear to Books Table**:

```
ALTER TABLE Books

ADD PublicationYear INT;
```

4 **Update Price for Books Published Before 2020 by Increasing 10%**:

```
UPDATE Books

SET Price = Price * 1.10

WHERE PublicationYear < 2020;
```

5 **Delete All Books with Genre Outdated Technology**:

```
DELETE FROM Books

WHERE Genre = 'Outdated Technology';
```

Q.9 DDL and DML Commands (on a single table: Books) a) Create a table Books using DDL with fields: BookID, Title, Author, Price, and StockAvailable. b) Insert 5 book records into the Books table using the INSERT command. c) Modify the structure of Books table by adding a new column Genre using the ALTER TABLE command. d) Use the UPDATE command to increase the price of all books by RS 50 in the Books table. e) Delete all records from the Books table where StockAvailable is 0 using the DELETE command.

1 **Create the Books Table**:

```
CREATE TABLE Books (

BookID INT PRIMARY KEY,

Title VARCHAR(255),

Author VARCHAR(255),

Price DECIMAL(10, 2),

StockAvailable INT

);
```

2 **Insert 5 Records into the Books Table**:

```
INSERT INTO Books (BookID, Title, Author, Price, StockAvailable)

VALUES

(1, 'The Great Gatsby', 'F. Scott Fitzgerald', 15.99, 10),
```

(2, '1984', 'George Orwell', 12.99, 5),

(3, 'To Kill a Mockingbird', 'Harper Lee', 10.50, 0),

(4, 'Pride and Prejudice', 'Jane Austen', 18.00, 3),

(5, 'The Catcher in the Rye', 'J.D. Salinger', 14.00, 7);

3 **Add the Genre Column to the Books Table**:

```
ALTER TABLE Books

ADD Genre VARCHAR(100);
```

4 **Increase the Price of All Books by Rs 50**:

```
UPDATE Books

SET Price = Price + 50;
```

5 **Delete All Books Where StockAvailable is 0**:

```
DELETE FROM Books

WHERE StockAvailable = 0;
```

Q.10 Analyze Sales Performance Using Aggregate Functions Create a Sales table with columns: SaleID, ProductID, ProductName, Quantity, Discount, SaleAmount, and SalesPerson.) a) Calculate the total quantity of products sold across all transactions in the Sales table. b) Find the average sale amount for transactions made in March 2025. c) Identify the product with the minimum sale quantity from the Sales table. d) Determine the maximum discount offered in February 2025. e) Count how many sales were made by each salesperson using GROUP BY SalesPerson.

1 **Create the Sales Table**

```
-- Create the Sales table with necessary columns

CREATE TABLE Sales (

SaleID INT PRIMARY KEY,

ProductID INT,

ProductName VARCHAR(255),

Quantity INT,

Discount DECIMAL(5, 2),
```

SaleAmount DECIMAL(10, 2),

SalesPerson VARCHAR(100),

SaleDate DATE

);

2. Insert Sample Data into the Sales Table

-- Insert sample records into the Sales table

INSERT INTO Sales (SaleID, ProductID, ProductName, Quantity, Discount, SaleAmount, SalesPerson, SaleDate)

VALUES

(1, 101, 'Laptop', 5, 10, 50000, 'Alice', '2025-03-05'),

(2, 102, 'Smartphone', 3, 5, 30000, 'Bob', '2025-02-12'),

(3, 103, 'Tablet', 2, 20, 20000, 'Alice', '2025-03-20'),

(4, 104, 'Smartwatch', 4, 15, 40000, 'Charlie', '2025-03-15'),

(5, 101, 'Laptop', 6, 10, 60000, 'Bob', '2025-02-25'),

(6, 105, 'Headphones', 10, 0, 10000, 'Alice', '2025-01-30'),

(7, 102, 'Smartphone', 1, 5, 10000, 'Charlie', '2025-02-20'),

(8, 103, 'Tablet', 3, 10, 30000, 'Alice', '2025-02-28');

3 **Calculate the Total Quantity of Products Sold Across All Transactions**:

SELECT SUM(Quantity) AS TotalQuantitySold

FROM Sales;

4 **Find the Average Sale Amount for Transactions Made in March 2025**:

SELECT AVG(SaleAmount) AS AvgSaleAmountMarch2025

FROM Sales

WHERE SaleDate BETWEEN '2025-03-01' AND '2025-03-31';

5 **Identify the Product with the Minimum Sale Quantity**:

SELECT ProductName, MIN(Quantity) AS MinQuantitySold

```
FROM Sales
GROUP BY ProductName
ORDER BY MinQuantitySold ASC
LIMIT 1;
```

6 **Determine the Maximum Discount Offered in February 2025**:

```
SELECT MAX(Discount) AS MaxDiscountFeb2025
FROM Sales
WHERE SaleDate BETWEEN '2025-02-01' AND '2025-02-28';
```

7 **Count How Many Sales Were Made by Each Salesperson**:

```
SELECT SalesPerson, COUNT(*) AS SalesCount
FROM Sales
GROUP BY SalesPerson;
```