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.

```
-- Create Sales Table
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY,
    ProductID INT,
    Quantity INT,
    SaleAmount DECIMAL(10, 2),
    SaleDate DATE
);

-- Insert 10 sales records
INSERT INTO Sales (SaleID, ProductID, Quantity, SaleAmount, SaleDate) VALUES
(1, 101, 2, 200.00, '2025-04-01'),
(2, 102, 1, 150.00, '2025-04-02'),
(3, 103, 5, 500.00, '2025-04-03'),
(4, 101, 3, 300.00, '2025-04-04'),
(5, 104, 4, 400.00, '2025-04-05'),
(6, 105, 2, 120.00, '2025-04-06'),
(7, 106, 7, 700.00, '2025-04-07'),
(8, 107, 1, 80.00, '2025-04-08'),
(9, 103, 6, 600.00, '2025-04-09'),
(10, 105, 3, 180.00, '2025-04-10');

-- Calculate total revenue using SUM function
SELECT SUM(SaleAmount) AS TotalRevenue
FROM Sales;

-- Find product with the highest sale amount using MAX function
SELECT ProductID, MAX(SaleAmount) AS HighestSaleAmount
FROM Sales;

-- Retrieve the average sale amount per transaction using AVG function
SELECT AVG(SaleAmount) AS AverageSaleAmount
FROM Sales;
```

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%.

```
-- a) Create the Products table
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(255),
    Price DECIMAL(10, 2),
    StockQuantity INT
);
-- b) Insert five product records
INSERT INTO Products (ProductID, ProductName, Price, StockQuantity) VALUES
(1, 'Laptop', 1000.00, 50),
(2, 'Smartphone', 500.00, 150),
(3, 'Headphones', 100.00, 200),
(4, 'Monitor', 300.00, 30),
(5, 'Keyboard', 50.00, 100);
-- Display all products
SELECT * FROM Products;
-- c) Update the price of a product with ProductID = 3
UPDATE Products
SET Price = 120.00
WHERE ProductID = 3;
-- Verify the changes
SELECT * FROM Products WHERE ProductID = 3;
-- d) Delete a product from the table
DELETE FROM Products WHERE ProductID = 5;
-- Verify the deletion
SELECT * FROM Products;
-- e) Alter the table to add a new column Discount with a default value of 5%
ALTER TABLE Products
ADD Discount DECIMAL(5, 2) DEFAULT 5.00;
-- Display the table to check the new column
SELECT * FROM Products;
```

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'.

```
-- a) Create the Customers table with constraints
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255) UNIQUE,
    Age INT CHECK (Age > 18),
    Country VARCHAR(100) DEFAULT 'India'
);

-- b) Insert a valid customer and verify that the default country is assigned
INSERT INTO Customers (CustomerID, Name, Email, Age)
VALUES (1, 'John Doe', 'john.doe@example.com', 30);

-- Verify the inserted customer
SELECT * FROM Customers WHERE CustomerID = 1;

-- c) Attempt to insert a customer with an age of 16 (violating the CHECK constraint)
-- This will fail due to the CHECK constraint on Age
-- Uncomment the following line to test the violation
-- INSERT INTO Customers (CustomerID, Name, Email, Age)
-- VALUES (2, 'Jane Smith', 'jane.smith@example.com', 16);

-- d) Try inserting two customers with the same email (violating the UNIQUE constraint)
-- Insert the first customer
INSERT INTO Customers (CustomerID, Name, Email, Age)
VALUES (3, 'Alice Brown', 'alice.brown@example.com', 28);

-- Attempt to insert a second customer with the same email
-- Uncomment the following line to test the violation
-- INSERT INTO Customers (CustomerID, Name, Email, Age)
-- VALUES (4, 'Bob White', 'alice.brown@example.com', 30);

-- e) Retrieve all customers older than 25 and from a country other than 'India'
SELECT *
FROM Customers
WHERE Age > 25 AND Country != 'India';
```

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.

```
-- a) Create Department and EmployeeDetails tables with constraints
CREATE TABLE Department (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(255) NOT NULL
);

CREATE TABLE EmployeeDetails (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(255) NOT NULL,
    DepartmentID INT,
    Salary DECIMAL(10, 2),
    FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)
);

-- b) Insert valid employee and test foreign key violation
INSERT INTO Department (DepartmentID, DepartmentName)
VALUES (1, 'HR'), (2, 'Finance');

-- Insert a valid employee record
INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, DepartmentID, Salary)
VALUES (101, 'John Doe', 1, 50000);

-- Attempt to insert an employee with a non-existent DepartmentID
-- Uncomment the line below to test foreign key violation
-- INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, DepartmentID, Salary)
-- VALUES (102, 'Jane Smith', 999, 60000);

-- c) Insert duplicate EmployeeID and test primary key violation
INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, DepartmentID, Salary)
VALUES (103, 'Alice Brown', 1, 55000);

-- Attempt to insert a duplicate EmployeeID
-- Uncomment the line below to test primary key violation
-- INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, DepartmentID, Salary)
-- VALUES (103, 'Bob White', 2, 60000);

-- d) Modify Salary column to have UNIQUE constraint and test
ALTER TABLE EmployeeDetails
ADD CONSTRAINT UNIQUE_Salary UNIQUE (Salary);
```

```
-- Insert a valid salary
INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, DepartmentID, Salary)
VALUES (104, 'Charlie Green', 1, 60000);

-- Attempt to insert an employee with the same salary (violates UNIQUE constraint)
-- Uncomment the line below to test UNIQUE constraint violation
-- INSERT INTO EmployeeDetails (EmployeeID, EmployeeName, DepartmentID, Salary)
-- VALUES (105, 'Diana Blue', 2, 60000);

-- e) Delete an employee and ensure referential integrity is maintained
DELETE FROM EmployeeDetails WHERE EmployeeID = 101;

-- Verify the deletion
SELECT * FROM EmployeeDetails;
```

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).

```
-- a) Create the Employee table
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(255),
    Salary DECIMAL(10, 2),
    JoiningDate DATE,
    ActiveStatus BOOLEAN
);

-- b) Insert five sample employee records
INSERT INTO Employee (EmployeeID, Name, Salary, JoiningDate, ActiveStatus)
VALUES
(1, 'Amit Sharma', 50000.00, '2022-08-15', TRUE),
(2, 'Priya Gupta', 60000.00, '2021-06-10', TRUE),
(3, 'Rahul Verma', 55000.00, '2020-12-01', FALSE),
(4, 'Suman Reddy', 70000.00, '2023-02-01', TRUE),
(5, 'Neha Kumar', 65000.00, '2019-04-20', TRUE);

-- c) Find all employees who joined before January 1, 2023
SELECT *
FROM Employee
WHERE JoiningDate < '2023-01-01';

-- d) Update the salary of 'Amit Sharma' by 10% and display the updated record
UPDATE Employee
```

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

-- Display the updated record of 'Amit Sharma'
SELECT * FROM Employee WHERE Name = 'Amit Sharma';

-- e) Retrieve all employees who are currently active (ActiveStatus = TRUE)
SELECT *
FROM Employee
WHERE ActiveStatus = TRUE;
```

6) Aggregate Functions (on a single table: Sales) 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”.

```
-- a) 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';

-- b) Find the average billing amount
SELECT AVG(SaleAmount) AS AverageBillingAmount
FROM Sales;

-- c) Find the minimum quantity of products sold in any transaction
SELECT MIN(Quantity) AS MinQuantitySold
FROM Sales;

-- d) Find the highest discount applied on any sale
SELECT MAX(Discount) AS MaxDiscount
FROM Sales;

-- e) Find how many transactions were recorded for the product "Laptop"
SELECT COUNT(*) AS LaptopTransactionCount
FROM Sales
WHERE ProductName = 'Laptop';
```

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.

```
-- a) Create the Employees table with PRIMARY KEY, UNIQUE, and CHECK constraints
CREATE TABLE Employees (
```

```
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255) UNIQUE,
    Salary DECIMAL(10, 2),
    Status VARCHAR(50),
    CHECK (Salary > 10000)
);

-- b) Add a NOT NULL constraint on the Name column and try inserting a record without the
Name
ALTER TABLE Employees
MODIFY Name VARCHAR(255) NOT NULL;

-- Attempt to insert a record without specifying the Name (This should fail)
-- Uncomment the line below to see the violation
-- INSERT INTO Employees (EmployeeID, Email, Salary, Status)
-- VALUES (1, 'john.doe@example.com', 15000, 'Active');  -- Missing Name

-- c) Add a DEFAULT value 'Active' to the Status column and insert a record without specifying
the status
ALTER TABLE Employees
ALTER COLUMN Status SET DEFAULT 'Active';

-- Insert a record without specifying the Status (This should use 'Active' as the default)
INSERT INTO Employees (EmployeeID, Name, Email, Salary)
VALUES (2, 'Jane Doe', 'jane.doe@example.com', 20000);

-- Verify the inserted record
SELECT * FROM Employees WHERE EmployeeID = 2;

-- d) Insert a record with a Salary less than 10,000 to test the CHECK constraint
-- Attempt to insert a record with Salary less than 10,000 (This should fail)
-- Uncomment the line below to see the violation
-- INSERT INTO Employees (EmployeeID, Name, Email, Salary, Status)
-- VALUES (3, 'Mark Smith', 'mark.smith@example.com', 8000, 'Active');  -- Salary < 10000

-- e) Try inserting two employees with the same Email ID to verify the enforcement of the
UNIQUE constraint
-- First insert should work
INSERT INTO Employees (EmployeeID, Name, Email, Salary, Status)
VALUES (4, 'Alice Brown', 'alice.brown@example.com', 25000, 'Active');

-- Second insert with the same Email should fail due to the UNIQUE constraint
-- Uncomment the line below to see the violation
-- INSERT INTO Employees (EmployeeID, Name, Email, Salary, Status)
-- VALUES (5, 'Bob White', 'alice.brown@example.com', 30000, 'Active');  -- Duplicate Email
```

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.

```
-- a) Create the Library database
CREATE DATABASE Library;

-- Switch to the Library database
USE Library;

-- Create the Books table with BookID as PRIMARY KEY
CREATE TABLE Books (
   BookID INT PRIMARY KEY,
   Title VARCHAR(255),
   Author VARCHAR(255),
   Genre VARCHAR(100),
   Price DECIMAL(10, 2)
);

-- b) Insert sample records into the Books table
INSERT INTO Books (BookID, Title, Author, Genre, Price)
VALUES
(1, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction', 10.99),
(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction', 7.99),
(3, '1984', 'George Orwell', 'Dystopian', 8.99),
(4, 'Introduction to Algorithms', 'Thomas H. Cormen', 'Technology', 60.00),
(5, 'The Pragmatic Programmer', 'David Thomas', 'Technology', 45.00);
-- Verify the inserted records
SELECT * FROM Books;
-- c) Add a new column PublicationYear to the Books table
ALTER TABLE Books
ADD PublicationYear INT;
-- d) Update the price of all books published before 2020 by increasing 10%
UPDATE Books
SET Price = Price * 1.10
WHERE PublicationYear < 2020;
-- e) Delete all books where the genre is 'Outdated Technology'
DELETE FROM Books
WHERE Genre = 'Outdated Technology';
-- Verify the change by selecting all remaining records
SELECT * FROM Books;
```

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.

```
-- a) Create the Books table with BookID as PRIMARY KEY
CREATE TABLE Books (
   BookID INT PRIMARY KEY,
   Title VARCHAR(255),
   Author VARCHAR(255),
   Price DECIMAL(10, 2),
   StockAvailable INT
);

-- b) Insert 5 book records into the Books table
INSERT INTO Books (BookID, Title, Author, Price, StockAvailable)
VALUES
(1, 'The Catcher in the Rye', 'J.D. Salinger', 350.50, 20),
(2, 'To Kill a Mockingbird', 'Harper Lee', 400.75, 15),
(3, '1984', 'George Orwell', 500.00, 10),
(4, 'Pride and Prejudice', 'Jane Austen', 250.25, 0),
(5, 'The Great Gatsby', 'F. Scott Fitzgerald', 300.00, 5);

-- c) Add a new column Genre to the Books table
ALTER TABLE Books
ADD Genre VARCHAR(100);

-- d) Increase the price of all books by RS 50
UPDATE Books
SET Price = Price + 50;

-- e) Delete all records where StockAvailable is 0
DELETE FROM Books
WHERE StockAvailable = 0;
```

10) Analyze Sales Performance Using Aggregate Functions 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.

```
-- a) Calculate the total quantity of products sold across all transactions
SELECT SUM(Quantity) AS TotalQuantitySold
FROM Sales;

-- b) Find the average sale amount for transactions made in March 2025
SELECT AVG(SaleAmount) AS AverageSaleAmount
FROM Sales
```

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

-- c) Identify the product with the minimum sale quantity from the Sales table
SELECT ProductID, MIN(Quantity) AS MinimumSaleQuantity
FROM Sales
GROUP BY ProductID
ORDER BY MinimumSaleQuantity ASC
LIMIT 1;

-- d) Determine the maximum discount offered in February 2025
SELECT MAX(Discount) AS MaxDiscount
FROM Sales
WHERE SaleDate BETWEEN '2025-02-01' AND '2025-02-28';

-- e) Count how many sales were made by each salesperson
SELECT SalesPerson, COUNT(*) AS SalesCount
FROM Sales
GROUP BY SalesPerson;
```