SQL Queries for All Sets

Set 1

Q1 (15 marks): Create `Orders` and `Customers` tables. Retrieve customer order history using different types of JOINS.

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

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    Amount DECIMAL(10, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

-- Retrieve customer order history using INNER JOIN
SELECT Customers.Name, Orders.OrderDate, Orders.Amount
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

-- Using LEFT JOIN to include customers without orders
SELECT Customers.Name, Orders.OrderDate, Orders.Amount
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Q2 (15 marks): Build a `SalesRecord` table and apply aggregate functions using different types of JOINS.

```
CREATE TABLE SalesRecord (
    SaleID INT PRIMARY KEY,
    ProductID INT,
    Quantity INT,
    SaleAmount DECIMAL(10, 2),
    SaleDate DATE
);

-- Aggregate functions with JOIN
SELECT Products.ProductName, SUM(SalesRecord.SaleAmount) AS TotalSales
FROM SalesRecord
INNER JOIN Products ON SalesRecord.ProductID = Products.ProductID
GROUP BY Products.ProductName;
```

Q3 (10 marks): Perform `UPDATE` on a sales table to increase all sales values by 10%. Summarize sales by region or category.

```
UPDATE SalesRecord
SET SaleAmount = SaleAmount * 1.10;

-- Summarize sales by region
SELECT Region, SUM(SaleAmount) AS TotalSales
FROM SalesRecord
GROUP BY Region;
```

Set 2

Q1: Create a table `Employee` with constraints: Primary Key, Unique, and Check.

```
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100) UNIQUE,
    Salary DECIMAL(10, 2) CHECK (Salary > 0)
);
```

Q2: Use `INSERT`, `UPDATE`, and `DELETE` commands on the `Employee` table.

```
INSERT INTO Employee (EmployeeID, Name, Email, Salary)
VALUES (1, 'John Doe', 'john@example.com', 5000);

UPDATE Employee
SET Salary = 5500
WHERE EmployeeID = 1;

DELETE FROM Employee
WHERE EmployeeID = 1;
```

Q3: Use `SELECT` queries to fetch employee records based on certain conditions (e.g., salary ¿ 5000).

```
SELECT * FROM Employee
WHERE Salary > 5000;
```

Set 3

Q1: Create a table with a `DATE` column and use functions like `NOW()`, `DATE_ADD()`, `DATEDIFF()`.

```
CREATE TABLE Events (
    EventID INT PRIMARY KEY,
    EventName VARCHAR(100),
    EventDate DATE
);

-- Insert a record with current date
INSERT INTO Events (EventID, EventName, EventDate)
VALUES (1, 'Meeting', NOW());

-- Add 7 days to a date
SELECT EventID, EventName, DATE_ADD(EventDate, INTERVAL 7 DAY) AS NewDate
FROM Events;

-- Calculate difference between two dates
SELECT EventID, EventName, DATEDIFF(NOW(), EventDate) AS DaysDifference
FROM Events;
```

Q2: Design a table with constraints including `DEFAULT` and `NOT NULL`.

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL,
    Email VARCHAR(100) DEFAULT 'unknown@example.com'
);
```

Q3: Use `ORDER BY`, `LIMIT`, and `BETWEEN` in your queries.

```
SELECT * FROM Users
ORDER BY Username ASC
LIMIT 5;

SELECT * FROM Users
WHERE UserID BETWEEN 1 AND 10;
```

Set 4

Q1: Use GROUP BY and HAVING clauses on a Sales table.

```
SELECT Region, SUM(SalesAmount) AS TotalSales
FROM Sales
GROUP BY Region
HAVING SUM(SalesAmount) > 1000;
```

Q2: Use CASE statements in a SELECT query to classify data (e.g., grades based on marks).

```
SELECT StudentID, Name, Marks,
    CASE
        WHEN Marks >= 90 THEN 'A'
        WHEN Marks >= 80 THEN 'B'
        WHEN Marks >= 70 THEN 'C'
        ELSE 'D'
    END AS Grade
FROM Students;
```

Q3: Apply mathematical functions like ROUND(), FLOOR(), and CEIL() in queries.

```
SELECT SaleAmount,
    ROUND(SaleAmount, 2) AS RoundedAmount,
    FLOOR(SaleAmount) AS FloorAmount,
    CEIL(SaleAmount) AS CeilAmount
FROM Sales;
```

Set 5

Q1: Create a table StudentRecord with StudentID, Name, Marks, Grade. Use SELECT WHERE Marks > 50.

```
CREATE TABLE StudentRecord (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100),
    Marks INT,
    Grade CHAR(1)
);

SELECT * FROM StudentRecord
WHERE Marks > 50;
```

Q2: Create an Enrollment table with a composite primary key (e.g., StudentID + CourseID).

```
CREATE TABLE Enrollment (
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (StudentID, CourseID)
);
```

Q3: Create Departments and Employees tables, and use LEFT JOIN to show all departments, even without employees.

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

CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    DepartmentID INT,
    Name VARCHAR(100),
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);

SELECT Departments.DepartmentName, Employees.Name
FROM Departments
LEFT JOIN Employees ON Departments.DepartmentID = Employees.DepartmentID;
```

Set 6

Q1: Create `Customers` and `Orders` tables with `AUTO_INCREMENT` and `FOREIGN KEY`.

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

CREATE TABLE Orders (
    OrderID INT AUTO_INCREMENT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    Amount DECIMAL(10, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Q2: Use `TRUNCATE`, `DELETE`, and `DROP` commands.

```
TRUNCATE TABLE Orders;

DELETE FROM Orders
WHERE OrderID = 1;

DROP TABLE Orders;
```

Q3: Demonstrate usage of subqueries in `SELECT` and `WHERE` clauses.

```
SELECT CustomerID, Name
FROM Customers
WHERE CustomerID IN (SELECT CustomerID FROM Orders WHERE Amount > 1000);
```

Set 7

Q1: Create a table with a `CHECK` constraint for a numeric column (e.g., `Age >= 18`).

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Name VARCHAR(100),
    Age INT CHECK (Age >= 18)
);
```

Q2: Add `NOT NULL` and `DEFAULT` constraints and verify them using `INSERT` statements.

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100) NOT NULL,
    Price DECIMAL(10, 2) DEFAULT 0.00
);

INSERT INTO Products (ProductID, ProductName)
VALUES (1, 'Laptop');
```

Q3: Retrieve the top 3 employees with the highest salaries using `ORDER BY` and `LIMIT`.

```
SELECT EmployeeID, Name, Salary
FROM Employees
ORDER BY Salary DESC
LIMIT 3;
```

Set 8

Q1: Use `ALTER TABLE` to add and remove columns, and modify column data types.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100)
);

ALTER TABLE Employees
ADD COLUMN Salary DECIMAL(10, 2);

ALTER TABLE Employees
DROP COLUMN Salary;

ALTER TABLE Employees
MODIFY COLUMN Name VARCHAR(200);
```

Q2: Demonstrate use of `INNER JOIN`, `LEFT JOIN`, and `RIGHT JOIN` in one query block.

```
SELECT Customers.Name AS CustomerName, Orders.OrderDate, Products.ProductName
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
LEFT JOIN Products ON Orders.ProductID = Products.ProductID
RIGHT JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID;
```

Q3: Create and use views in SQL.

```
CREATE VIEW TopCustomers AS
SELECT CustomerID, Name, SUM(Amount) AS TotalSpent
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
GROUP BY CustomerID, Name
ORDER BY TotalSpent DESC
LIMIT 5;

SELECT * FROM TopCustomers;
```

Set 9

Q1: Use `GROUP BY` and `HAVING` with multiple conditions on a `Product` sales table.

```
SELECT ProductID, SUM(Quantity) AS TotalQuantity
FROM Sales
GROUP BY ProductID
HAVING SUM(Quantity) > 100 AND ProductID IN (1, 2, 3);
```

Q2: Use nested subqueries in `SELECT`, `FROM`, and `WHERE`.

```
SELECT EmployeeID, Name, Salary
FROM Employees
WHERE Salary > (
    SELECT AVG(Salary)
    FROM Employees
    WHERE DepartmentID = (
        SELECT DepartmentID
        FROM Departments
        WHERE DepartmentName = 'Sales'
    )
);
```

Q3: Use string functions like `CONCAT()`, `SUBSTRING()`, `LENGTH()`.

```
SELECT CONCAT(FirstName, ' ', LastName) AS FullName,
    SUBSTRING(Email, 1, INSTR(Email, '@') - 1) AS Username,
    LENGTH(Email) AS EmailLength
FROM Employees;
```

Set 10

Q1: Create a `Library` table. Use `ALTER TABLE` to add an `Author` column, then `UPDATE` values.

```
CREATE TABLE Library (
    BookID INT PRIMARY KEY,
    Title VARCHAR(100)
);

ALTER TABLE Library
ADD COLUMN Author VARCHAR(100);

UPDATE Library
SET Author = 'J.K. Rowling'
WHERE Title = 'Harry Potter';
```

Q2: Use built-in date functions like `CURDATE()`, `YEAR()`, `MONTH()`.

```
SELECT CURDATE() AS CurrentDate,
    YEAR(CURDATE()) AS CurrentYear,
    MONTH(CURDATE()) AS CurrentMonth;
```

Q3: Create a stored procedure to display details from a specific table.

```
DELIMITER //

CREATE PROCEDURE GetEmployeeDetails()
BEGIN
    SELECT EmployeeID, Name, Salary
    FROM Employees;
END //

DELIMITER ;

-- Call the stored procedure
CALL GetEmployeeDetails();
```