**11/06/2025**

**Section 1: Managing Databases**

1. **Which of the following is NOT a system database in SQL Server?**  
   a) master  
   b) model  
   c) tempdb  
   d) userdb
2. **Which system database stores all login accounts and configuration settings?**  
   a) tempdb  
   b) model  
   c) master  
   d) msdb
3. **What is the purpose of the model database in SQL Server?**  
   a) Backup  
   b) Log storage  
   c) Template for new databases  
   d) System configuration
4. **What are the two main types of database files in SQL Server?**  
   a) MDF and NDF  
   b) LDF and MDF  
   c) NDF and BAK  
   d) BAK and TRN
5. **Which SQL command is used to create a new database?**  
   a) MAKE DATABASE  
   b) NEW DATABASE  
   c) CREATE DATABASE  
   d) INIT DATABASE
6. **What happens when you execute DROP DATABASE SalesDB?**  
   a) SalesDB is backed up  
   b) SalesDB is renamed  
   c) SalesDB is deleted permanently  
   d) SalesDB is restored
7. **Which command renames a database in SQL Server?**  
   a) RENAME DATABASE old\_name TO new\_name  
   b) ALTER DATABASE old\_name MODIFY NAME = new\_name   
   c) UPDATE DATABASE NAME  
   d) SET DATABASE NAME

**Section 2: Managing Tables**

1. **Which data type should be used to store a date of birth?**  
   a) VARCHAR  
   b) DATE  
   c) INT  
   d) TEXT
2. **What command is used to create a table?**  
   a) MAKE TABLE  
   b) INSERT TABLE  
   c) CREATE TABLE  
   d) DEFINE TABLE
3. **How do you add a new column to an existing table?**  
   a) ALTER TABLE table\_name ADD column\_name datatype  
   b) MODIFY TABLE table\_name ADD column\_name  
   c) UPDATE TABLE table\_name ADD column\_name  
   d) APPEND column\_name TO table\_name
4. **Which command is used to rename a table?**  
   a) RENAME TABLE old\_name TO new\_name  
   b) ALTER TABLE old\_name RENAME TO new\_name  
   c) EXEC sp\_rename 'old\_name', 'new\_name'  
   d) MODIFY TABLE RENAME
5. **What is the command to delete a table permanently?**  
   a) DELETE TABLE table\_name  
   b) ERASE TABLE table\_name  
   c) DROP TABLE table\_name  
   d) REMOVE TABLE table\_name

**Section 3: DML - Manipulating Data**

1. **Which command adds data into a table?**  
   a) INSERT INTO  
   b) ADD ROW  
   c) CREATE DATA  
   d) APPEND TO
2. **Which clause is used to update data in a table?**  
   a) MODIFY  
   b) UPDATE  
   c) CHANGE  
   d) SET TABLE
3. **What does the DELETE statement do?**  
   a) Removes a column  
   b) Removes all data from a table  
   c) Removes specific rows  
   d) Deletes the table schema
4. **Which clause is used to filter rows in a SELECT statement?**  
   a) HAVING  
   b) SELECT  
   c) WHERE  
   d) ORDER BY
5. **Which keyword ensures no duplicate records are returned?**  
   a) UNIQUE  
   b) NO\_REPEAT  
   c) DISTINCT  
   d) ONLY
6. **What does the LIKE keyword do in SQL?**  
   a) Finds exact matches  
   b) Finds pattern-based matches  
   c) Sorts records  
   d) Deletes matches
7. **Which operator is used to combine multiple conditions in a WHERE clause?**  
   a) TO  
   b) WITH  
   c) AND / OR  
   d) IF / ELSE
8. **What does the BETWEEN operator do?**  
   a) Compares text fields  
   b) Finds rows outside a range  
   c) Filters values within a range  
   d) Joins tables

**12/06/2025**

1. Insert and Update with Integrity

CREATE TABLE students (

student\_id INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

marks INT CHECK (marks >= 0 AND marks <= 100));

2. String Function Challenge

CREATE TABLE customers (

customer\_id INT PRIMARY KEY,

full\_name VARCHAR(100) NOT NULL);

INSERT INTO customers VALUES

(1, 'Virat Kohli'),

(2, 'Rohit Sharma'),

(3, 'MS Dhoni'),

(4, 'KL Rahul'),

(5, 'Jasprit Bumrah');

SELECT

full\_name,

SUBSTRING\_INDEX(full\_name, ' ', 1) AS first\_name,

SUBSTRING\_INDEX(full\_name, ' ', -1) AS last\_name,

LENGTH(SUBSTRING\_INDEX(full\_name, ' ', 1)) AS first\_name\_length,

LENGTH(SUBSTRING\_INDEX(full\_name, ' ', -1)) AS last\_name\_length

FROM customers;

3. Date Function Usage

CREATE TABLE sales (

sale\_id INT PRIMARY KEY,

sale\_date DATE,

category VARCHAR(100),

amount DECIMAL(10,2)

);

INSERT INTO sales VALUES

(1, '2025-06-10', 'Bats', 2000),

(2, '2025-05-01', 'Balls', 1500),

(3, '2025-01-15', 'Gloves', 3000),

(4, '2025-06-01', 'Pads', 1200);

SELECT

sale\_date,

MONTHNAME(sale\_date) AS month\_name,

YEAR(sale\_date) AS sale\_year,

DATEDIFF(CURDATE(), sale\_date) AS days\_ago

FROM sales;

4. Mathematical Functions on Salary

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(100),

salary DECIMAL(10, 2)

);

INSERT INTO employees VALUES

(1, 'Virat Kohli', 120000),

(2, 'Rohit Sharma', 95000);

SELECT

name,

salary,

salary \* 1.10 AS salary\_with\_hike,

ROUND(salary, -2) AS rounded\_salary

FROM employees;

5. System Function Check

SELECT

NOW() AS current\_datetime,

DATABASE() AS current\_database,

USER() AS logged\_in\_user;

-- 6. Demo: Custom Result Set

CREATE TABLE products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(100),

price DECIMAL(10,2),

category VARCHAR(50)

);

INSERT INTO products VALUES

(1, 'Bat', NULL, 'Gear'),

(2, 'Ball', 1500, 'Gear'),

(3, 'Helmet', 2500, 'Protection');

SELECT

UPPER(product\_name) AS product\_name\_uppercase,

IFNULL(price, 'Not Available') AS price\_status

FROM products;

7. Aggregate Functions Practice

CREATE TABLE transactions (

transaction\_id INT PRIMARY KEY,

sale\_amount DECIMAL(10,2)

);

INSERT INTO transactions VALUES

(1, 1000),

(2, 1500),

(3, 500),

(4, 2000);

SELECT

SUM(sale\_amount) AS total\_sales,

AVG(sale\_amount) AS average\_sale,

MAX(sale\_amount) AS max\_sale,

MIN(sale\_amount) AS min\_sale

FROM transactions;

8. Grouping with Aggregation

SELECT

category,

COUNT(\*) AS transaction\_count,

SUM(amount) AS total\_sales

FROM sales

GROUP BY category;

9. Inner Join for Orders and Customers

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_amount DECIMAL(10,2)

);

INSERT INTO orders VALUES

(101, 1, 5000),

(102, 2, 3000),

(103, 4, 4500);

SELECT

c.full\_name AS customer\_name,

o.order\_amount

FROM orders o

INNER JOIN customers c ON o.customer\_id = c.customer\_id;

10. Left Join for Products with or without Orders

CREATE TABLE order\_details (

order\_id INT,

product\_id INT,

quantity INT

);

INSERT INTO order\_details VALUES

(101, 1, 2),

(102, 2, 1);

SELECT

p.product\_name,

od.quantity

FROM products p

LEFT JOIN order\_details od ON p.product\_id = od.product\_id;

11. Right Join for Customer Contacts

CREATE TABLE contacts (

contact\_id INT,

customer\_id INT,

phone VARCHAR(15)

);

INSERT INTO contacts VALUES

(1, 1, '1234567890'),

(2, 3, '9999999999');

SELECT

c.full\_name,

ct.phone

FROM contacts ct

RIGHT JOIN customers c ON ct.customer\_id = c.customer\_id;

12. Full Outer Join for Suppliers and Products (simulated)

CREATE TABLE suppliers (

supplier\_id INT,

supplier\_name VARCHAR(100)

);

INSERT INTO suppliers VALUES

(1, 'SG'),

(2, 'MRF');

SELECT s.supplier\_name, p.product\_name

FROM suppliers s

LEFT JOIN products p ON s.supplier\_id = p.product\_id

UNION

SELECT s.supplier\_name, p.product\_name

FROM suppliers s

RIGHT JOIN products p ON s.supplier\_id = p.product\_id;

13. Cross Join for Offers

CREATE TABLE offers (

offer\_id INT,

offer\_desc VARCHAR(100)

);

INSERT INTO offers VALUES

(1, '10% Off'),

(2, 'Free Shipping');

SELECT

p.product\_name,

o.offer\_desc

FROM products p

CROSS JOIN offers o;

14. Join with Aggregation

CREATE TABLE orders\_extended (

order\_id INT,

product\_id INT,

quantity INT,

price DECIMAL(10,2)

);

INSERT INTO orders\_extended VALUES

(1, 1, 2, 5000),

(2, 2, 1, 1500),

(3, 3, 3, 7500);

SELECT

p.category,

SUM(o.quantity) AS total\_quantity\_sold,

AVG(o.price) AS avg\_price

FROM orders\_extended o

JOIN products p ON o.product\_id = p.product\_id

GROUP BY p.category;

15. Join with Grouping and Filter

CREATE TABLE marks (

student\_id INT,

subject VARCHAR(50),

mark INT

);

INSERT INTO marks VALUES

(1, 'Math', 80),

(1, 'Science', 85),

(2, 'Math', 95),

(2, 'Science', 90),

(3, 'Math', 70),

(3, 'Science', 75);

SELECT

s.name AS student\_name,

AVG(m.mark) AS average\_marks

FROM students s

JOIN marks m ON s.student\_id = m.student\_id

GROUP BY s.student\_id, s.name

HAVING AVG(m.mark) > 75;

**13/06/2025**

**SQL Practical Answer Documents**

**Advanced Subqueries and Set Operations**

**1) Querying Data by Using Subqueries**

SELECT Name FROM Customers  
WHERE CustomerID IN (SELECT CustomerID FROM Orders WHERE Quantity > 5);

*This retrieves customer names who have placed orders with quantity greater than 5.*

**2) Using the EXISTS Keyword**

SELECT Name FROM Customers c  
WHERE EXISTS (  
 SELECT 1 FROM Orders o WHERE o.CustomerID = c.CustomerID  
);

*Checks for the existence of at least one order per customer.*

**3) Using the ANY Keyword**

SELECT Name FROM Products  
WHERE Price > ANY (  
 SELECT Price FROM Products WHERE Category = 'Electronics'  
);

*Returns products more expensive than at least one in the Electronics category.*

**4) Using the ALL Keyword**

SELECT Name FROM Products  
WHERE Price > ALL (  
 SELECT Price FROM Products WHERE Category = 'Accessories'  
);

*Lists products priced higher than all products in the Accessories category.*

**5) Using Nested Subqueries**

SELECT Name FROM Customers  
WHERE CustomerID IN (  
 SELECT CustomerID FROM Orders  
 WHERE ProductID IN (  
 SELECT ProductID FROM Products WHERE Price > 1000  
 )  
);

*Selects customers who ordered products with price > 1000.*

**6) Using Correlated Subqueries**

SELECT \* FROM Orders o1  
WHERE Quantity > (  
 SELECT AVG(Quantity) FROM Orders o2 WHERE o1.CustomerID = o2.CustomerID  
);

*Returns orders having quantity greater than average quantity by that customer.*

**7) Using UNION**

SELECT Name FROM SegmentA  
UNION  
SELECT Name FROM SegmentB;

*Combines unique names from both segments.*

**8) Using INTERSECT (simulated)**

SELECT Name FROM SegmentA  
WHERE Name IN (SELECT Name FROM SegmentB);

*Finds common names between two segments.*

**9) Using EXCEPT (simulated)**

SELECT Name FROM SegmentA  
WHERE Name NOT IN (SELECT Name FROM SegmentB);

*Returns names from SegmentA not found in SegmentB.*

**10) Using MERGE (Simulated with UPDATE + INSERT)**

-- Update if exists  
UPDATE Customers SET Email = 'updated@domain.com'  
WHERE CustomerID = 1001;  
  
-- Insert if not exists  
INSERT INTO Customers (CustomerID, Name, Email)  
SELECT 1001, 'New Customer', 'updated@domain.com'  
WHERE NOT EXISTS (  
 SELECT 1 FROM Customers WHERE CustomerID = 1001  
);

*Implements upsert (update or insert) behavior similar to MERGE.*

**13/06/2025**

**SQL Practical Answer Document**

**Section A: Basics & Data Definition (10 Marks)**

**Q1: SQL vs NoSQL**

| Feature | SQL | NoSQL |
| --- | --- | --- |
| Type | Relational DB (Structured) | Non-relational (Document, Key-Value etc) |
| Schema | Fixed schema | Dynamic schema |
| Example | MySQL, PostgreSQL | MongoDB, Cassandra |
| Use case | Banking systems | Real-time analytics, IoT, social apps |
| Advantage 1 | Strong ACID compliance | High scalability |
| Advantage 2 | Structured query language for complex joins | Flexible data models |
| Disadvantage 1 | Not good with unstructured data | Lacks mature query languages |
| Disadvantage 2 | Limited horizontal scaling | Weaker ACID guarantees |

**Q2: Normalization**

* **UNF:** Student (StudentID, Name, CourseID, CourseName, InstructorName, InstructorPhone)
* **1NF:**
  + Ensure atomicity:
    - Separate repeating groups
* **2NF:**
  + Remove partial dependency:
    - Student (StudentID, Name)
    - Course (CourseID, CourseName, InstructorName, InstructorPhone)
    - Enrollment (StudentID, CourseID)
* **3NF:**
  + Remove transitive dependency:
    - Instructor (InstructorName, InstructorPhone)
    - Course (CourseID, CourseName, InstructorName)

**Q3:**

CREATE DATABASE StudentDB;  
  
USE StudentDB;  
  
CREATE TABLE Students (  
 StudentID INT PRIMARY KEY,  
 Name VARCHAR(50),  
 DOB DATE,  
 Email VARCHAR(100)  
);  
  
ALTER TABLE Students RENAME TO Student\_Info;  
  
ALTER TABLE Student\_Info ADD PhoneNumber BIGINT;  
  
DROP TABLE Student\_Info;

**Section B: DML & Filtering Data (15 Marks)**

**Q4:**

-- a)  
INSERT INTO Student\_Info VALUES (1, 'Alice', '2002-01-01', 'alice@gmail.com', 9876543210);  
INSERT INTO Student\_Info VALUES (2, 'Bob', '1999-05-05', 'bob@yahoo.com', 9765432109);  
INSERT INTO Student\_Info VALUES (3, 'Cathy', '2001-03-03', 'cathy@gmail.com', 9654321098);  
  
-- b)  
UPDATE Student\_Info SET PhoneNumber = 9123456789 WHERE StudentID = 2;  
  
-- c)  
DELETE FROM Student\_Info WHERE Email LIKE '%@gmail.com';  
  
-- d)  
SELECT Name, Email FROM Student\_Info WHERE YEAR(DOB) > 2000;  
  
-- e)  
SELECT DISTINCT SUBSTRING\_INDEX(Email, '@', -1) AS Domain FROM Student\_Info;

**Q5:**

-- a)  
SELECT \* FROM Student\_Info WHERE Name LIKE 'A%';  
  
-- b)  
SELECT \* FROM Student\_Info WHERE PhoneNumber BETWEEN 9000000000 AND 9999999999;  
  
-- c)  
SELECT \* FROM Student\_Info WHERE City IN ('Mumbai', 'Chennai', 'Delhi');  
  
-- d)  
SELECT \* FROM Student\_Info WHERE YEAR(CURDATE()) - YEAR(DOB) > 20 AND Email LIKE '%@gmail.com';  
  
-- e)  
SELECT s.Name AS Student\_Name, s.DOB AS Date\_of\_Birth FROM Student\_Info s;

**Q6:**

CREATE TABLE Marks (  
 StudentID INT,  
 Subject VARCHAR(50),  
 Marks FLOAT  
);  
  
INSERT INTO Marks VALUES (1, 'Maths', 80);  
INSERT INTO Marks VALUES (2, 'Science', 90);  
INSERT INTO Marks VALUES (3, 'English', 75);  
  
-- a)  
SELECT StudentID, Subject FROM Marks WHERE Marks > 70;  
  
-- b)  
SELECT Subject, AVG(Marks) AS Avg\_Marks FROM Marks GROUP BY Subject;  
  
-- c)  
SELECT Subject FROM Marks GROUP BY Subject HAVING AVG(Marks) BETWEEN 60 AND 90;

**Section C: Functions & Grouping (10 Marks)**

**Q7:**

-- a)  
SELECT DATE\_FORMAT(CURDATE(), '%Y-%m-%d') AS Current\_Date;  
  
-- b)  
SELECT MONTH(DOB) AS BirthMonth, YEAR(DOB) AS BirthYear FROM Student\_Info;  
  
-- c)  
SELECT UPPER(Name) AS Name\_Upper FROM Student\_Info;  
  
-- d)  
SELECT ROUND(Marks, 2) AS RoundedMarks FROM Marks;  
  
-- e)  
SELECT CURRENT\_USER() AS UserName, DATABASE() AS CurrentDB;

**Q8:**

-- a)  
SELECT StudentID, SUM(Marks) AS TotalMarks FROM Marks GROUP BY StudentID;  
  
-- b)  
SELECT Subject, MAX(Marks) AS HighestMark FROM Marks GROUP BY Subject;  
  
-- c)  
SELECT Subject, AVG(Marks) AS AvgMarks FROM Marks GROUP BY Subject HAVING AVG(Marks) > 75;

**Section D: Joins and Subqueries (25 Marks)**

**Q9:**

-- a)  
SELECT s.StudentID, s.Name, c.CourseName FROM Student\_Info s  
JOIN Enrollment e ON s.StudentID = e.StudentID  
JOIN Course c ON e.CourseID = c.CourseID;  
  
-- b)  
SELECT s.Name FROM Student\_Info s  
LEFT JOIN Enrollment e ON s.StudentID = e.StudentID;  
  
-- c)  
SELECT c.CourseName FROM Course c  
RIGHT JOIN Enrollment e ON c.CourseID = e.CourseID;  
  
-- d)  
SELECT s.Name, c.CourseName FROM Student\_Info s  
LEFT JOIN Enrollment e ON s.StudentID = e.StudentID  
LEFT JOIN Course c ON e.CourseID = c.CourseID  
UNION  
SELECT s.Name, c.CourseName FROM Course c  
LEFT JOIN Enrollment e ON c.CourseID = e.CourseID  
LEFT JOIN Student\_Info s ON e.StudentID = s.StudentID;  
  
-- e)  
SELECT \* FROM Student\_Info CROSS JOIN Course;

**Q10:**

-- a)  
SELECT \* FROM Marks m WHERE Subject = 'Maths' AND Marks > (SELECT AVG(Marks) FROM Marks WHERE Subject = 'Maths');  
  
-- b)  
SELECT \* FROM Student\_Info WHERE StudentID NOT IN (SELECT DISTINCT StudentID FROM Marks);  
  
-- c)  
SELECT \* FROM Student\_Info s WHERE EXISTS (SELECT 1 FROM Marks m WHERE s.StudentID = m.StudentID);  
  
-- d)  
SELECT \* FROM Marks m1 WHERE Subject = 'Science' AND Marks > ALL (SELECT Marks FROM Marks WHERE Subject = 'Science');  
  
-- e)  
SELECT \* FROM Marks m1 WHERE Subject = 'English' AND Marks > ANY (SELECT Marks FROM Marks WHERE Subject = 'English');

**Q11:**

-- a)  
SELECT Name FROM Student\_Info  
UNION  
SELECT Name FROM Alumni;  
  
-- b)  
SELECT Name FROM Student\_Info  
INTERSECT  
SELECT Name FROM Alumni;  
  
-- c)  
SELECT Name FROM Student\_Info  
EXCEPT  
SELECT Name FROM Marks;  
  
-- d)  
-- Simulating MERGE  
UPDATE Student\_Info SET Email = 'newemail@test.com' WHERE StudentID = 1;  
INSERT INTO Student\_Info (StudentID, Name, DOB, Email, PhoneNumber)  
SELECT 4, 'New Student', '2003-04-04', 'new@test.com', 9090909090  
WHERE NOT EXISTS (SELECT 1 FROM Student\_Info WHERE StudentID = 4);  
  
-- e)  
SELECT \* FROM Marks m1  
WHERE Marks > (  
 SELECT AVG(Marks) FROM Marks m2 WHERE m1.Subject = m2.Subject  
);

**SQL Practical Answer Document**

**Section A: Advanced Concepts & Schema Design (10 Marks)**

**Q1:**

**NoSQL Preference Scenarios:**

* Real-time analytics (e.g., weather monitoring, social media feeds)
* Flexible schema needs (e.g., product catalog with variable fields)
* High-volume data with low latency (e.g., IoT sensors, messaging apps)

**Types of NoSQL Databases:**

| Type | Description | Example | Real-time Application |
| --- | --- | --- | --- |
| Document-based | Stores data as JSON-like documents | MongoDB | Product catalogs |
| Key-Value | Simple key-value pairs | Redis | Caching user sessions |
| Column-family | Data stored in columns | Cassandra | Time-series data like logs |
| Graph | Relationships-focused | Neo4j | Social networks, fraud detection |

**Q2:**

**UNF:** Customer (CustomerID, Name, Orders(OrderID, ProductID, Quantity, ProductName))

**1NF:** Flatten repeating group:

* Customer(CustomerID, Name)
* Orders(OrderID, CustomerID, ProductID, Quantity, ProductName)

**2NF:** Remove partial dependencies:

* Product(ProductID, ProductName)
* OrderDetails(OrderID, ProductID, Quantity)

**3NF/BCNF:** Remove transitive dependencies:

* Orders(OrderID, CustomerID)

Final tables:

* Customer(CustomerID, Name)
* Product(ProductID, ProductName)
* Orders(OrderID, CustomerID)
* OrderDetails(OrderID, ProductID, Quantity)

**Section B: Complex DDL and DML (15 Marks)**

**Q3:**

CREATE DATABASE RetailDB;  
USE RetailDB;  
  
CREATE TABLE Customers (  
 CustomerID INT PRIMARY KEY,  
 Name VARCHAR(100),  
 Email VARCHAR(100)  
);  
  
CREATE TABLE Products (  
 ProductID INT PRIMARY KEY,  
 ProductName VARCHAR(100),  
 Price DECIMAL(10,2)  
);  
  
CREATE TABLE Orders (  
 OrderID INT PRIMARY KEY,  
 CustomerID INT,  
 ProductID INT,  
 Quantity INT CHECK (Quantity > 0),  
 OrderDate DATE,  
 FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),  
 FOREIGN KEY (ProductID) REFERENCES Products(ProductID)  
);  
  
ALTER TABLE Products ADD Discount DECIMAL(5,2);  
UPDATE Products SET Discount = 5.00 WHERE ProductID = 1;

**Q4:**

-- a)  
INSERT INTO Orders VALUES (101, 1, 1, 2, CURDATE());  
INSERT INTO Orders VALUES (102, 1, 2, 1, CURDATE());  
INSERT INTO Orders VALUES (103, 1, 3, 6, CURDATE());  
  
-- b)  
UPDATE Products SET Price = Price \* 1.1  
WHERE ProductID IN (  
 SELECT ProductID FROM Orders GROUP BY ProductID HAVING SUM(Quantity) > 5  
);  
  
-- c)  
DELETE FROM Products WHERE ProductID NOT IN (SELECT DISTINCT ProductID FROM Orders);

**Q5:**

-- a)  
SELECT c.CustomerID, c.Name FROM Customers c  
JOIN Orders o ON c.CustomerID = o.CustomerID  
GROUP BY c.CustomerID, c.Name  
HAVING COUNT(DISTINCT o.ProductID) > 3;  
  
-- b)  
SELECT \* FROM Products  
WHERE ProductID NOT IN (SELECT DISTINCT ProductID FROM Orders);  
  
-- c)  
SELECT CustomerID, COUNT(\*) AS OrderCount FROM Orders  
WHERE OrderDate >= CURDATE() - INTERVAL 30 DAY  
GROUP BY CustomerID;

**Section C: Advanced Functions and Aggregations (10 Marks)**

**Q6:**

-- a)  
SELECT Email,   
 LOWER(Email) AS lower\_email,  
 SUBSTRING\_INDEX(Email, '@', 1) AS username,  
 SUBSTRING\_INDEX(Email, '@', -1) AS domain  
FROM Customers;  
  
-- b)  
SELECT OrderID, DATEDIFF(CURDATE(), OrderDate) AS DaysSinceOrder FROM Orders;  
  
-- c)  
SELECT CURRENT\_USER() AS user, @@hostname AS host;  
  
-- d)  
SELECT CONCAT('Hello ', Name, '!') AS Greeting FROM Customers;

**Q7:**

-- a)  
SELECT p.ProductName, SUM(o.Quantity \* p.Price) AS Revenue  
FROM Orders o  
JOIN Products p ON o.ProductID = p.ProductID  
GROUP BY p.ProductName;  
  
-- b)  
SELECT p.ProductName, SUM(o.Quantity \* p.Price) AS Revenue  
FROM Orders o  
JOIN Products p ON o.ProductID = p.ProductID  
GROUP BY p.ProductName WITH ROLLUP;  
  
-- c)  
SELECT p.ProductName, SUM(o.Quantity \* p.Price) AS Revenue  
FROM Orders o  
JOIN Products p ON o.ProductID = p.ProductID  
GROUP BY p.ProductName  
HAVING Revenue > 100000;

**Section D: Complex Joins, Subqueries, and Set Ops (25 Marks)**

**Q8:**

-- a)  
SELECT A.CustomerID AS ReferrerID, B.CustomerID AS ReferredID  
FROM Customers A  
JOIN Customers B ON A.CustomerID = B.ReferredBy;  
  
-- b)  
SELECT \* FROM Orders o JOIN Products p ON o.ProductID = p.ProductID;  
  
-- c)  
SELECT CustomerID, SUM(p.Price \* o.Quantity) AS TotalSpend,  
 RANK() OVER (ORDER BY SUM(p.Price \* o.Quantity) DESC) AS Rank  
FROM Orders o  
JOIN Products p ON o.ProductID = p.ProductID  
GROUP BY CustomerID  
LIMIT 3;  
  
-- d)  
SELECT c.\* FROM Customers c  
LEFT JOIN Orders o ON c.CustomerID = o.CustomerID  
WHERE o.CustomerID IS NULL;  
  
-- e)  
SELECT \* FROM Products p1 CROSS JOIN Products p2 WHERE p1.ProductID < p2.ProductID;

**Q9:**

-- a)  
SELECT \* FROM Orders o1  
WHERE o1.Quantity \* (SELECT Price FROM Products WHERE ProductID = o1.ProductID)  
 > (SELECT AVG(o2.Quantity \* p2.Price)  
 FROM Orders o2 JOIN Products p2 ON o2.ProductID = p2.ProductID  
 WHERE o2.CustomerID = o1.CustomerID);  
  
-- b)  
SELECT \* FROM Customers c  
WHERE EXISTS (  
 SELECT 1 FROM Orders o WHERE o.CustomerID = c.CustomerID  
 GROUP BY o.CustomerID HAVING COUNT(DISTINCT o.ProductID) >= 2  
);  
  
-- c)  
SELECT \* FROM Customers c  
WHERE (SELECT COUNT(\*) FROM Orders o WHERE o.CustomerID = c.CustomerID)  
 > ALL (SELECT COUNT(\*) FROM Orders GROUP BY CustomerID);  
  
-- d)  
SELECT \* FROM Products  
WHERE Price > ANY (  
 SELECT Price FROM Products WHERE Category = 'Electronics'  
);  
  
-- e)  
SELECT ProductID, SUM(Quantity) AS TotalSold  
FROM Orders  
GROUP BY ProductID  
ORDER BY TotalSold DESC  
LIMIT 3;

**Q10:**

-- a)  
SELECT c1.CustomerID FROM SegmentA c1  
INNER JOIN SegmentB c2 ON c1.CustomerID = c2.CustomerID;  
  
-- b)  
SELECT \* FROM Inventory  
WHERE ProductID NOT IN (SELECT DISTINCT ProductID FROM Orders);  
  
-- c)  
-- Simulating MERGE  
UPDATE Customers SET Email = 'update@x.com' WHERE CustomerID = 5;  
INSERT INTO Customers (CustomerID, Name, Email)  
SELECT 5, 'John Doe', 'update@x.com'  
WHERE NOT EXISTS (SELECT 1 FROM Customers WHERE CustomerID = 5);  
  
-- d)  
SELECT \* FROM RegionalCustomers1  
UNION  
SELECT \* FROM RegionalCustomers2;  
  
-- e)  
WITH RankedCustomers AS (  
 SELECT CustomerID, SUM(p.Price \* o.Quantity) AS TotalSpent,  
 RANK() OVER (ORDER BY SUM(p.Price \* o.Quantity) DESC) AS rnk  
 FROM Orders o  
 JOIN Products p ON o.ProductID = p.ProductID  
 GROUP BY CustomerID  
)  
SELECT \* FROM RankedCustomers WHERE rnk <= 5;