**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;