## QUESTION PAPER 2

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

Q1. (4 marks) Explain with examples the scenarios where NoSQL is preferred over SQL. Discuss types of NoSQL databases and suggest a real-time application for each.

**Scenarios where NoSQL is preferred over SQL:**

**High scalability & performance:** NoSQL databases like MongoDB are used in systems handling massive real-time data (e.g., sensor networks, social media). **Unstructured or semi-structured data:** When data formats are dynamic, such as JSON logs or user-generated content. **Flexible schema requirements:** Useful in fast-paced development where schemas evolve rapidly (e.g., content platforms). **High availability over consistency (BASE model):** In globally distributed apps like e-commerce and messaging apps.

Example: Content Management System (CMS) for blogs storing articles as documents and Session management in web applications.

Q2. (6 marks) A retail store keeps the following unnormalized record: Customer (CustomerID, Name, Orders (OrderID, ProductID, Quantity, ProductName)) Normalize the data up to BCNF with appropriate table structures.

**Unnormalized Data:**

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

**1NF (Atomic values, remove repeating groups):**

Break nested data:

Customer\_Order (

CustomerID, Name, OrderID, ProductID, Quantity, ProductName

)

**2NF (No partial dependency):**

Remove dependency of Product info on part of primary key

Tables:

Customer (CustomerID, Name)

OrderDetails (OrderID, CustomerID)

Product (ProductID, ProductName)

OrderItems (OrderID, ProductID, Quantity)

**3NF (No transitive dependency):**

Already achieved since all non-key columns depend only on keys.

**BCNF (All determinants are candidate keys):**

Already satisfied. All functional dependencies have determinant as candidate key.

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

Q3. (5 marks) a) Create a database RetailDB and design a schema for Customers, Orders, and Products with primary and foreign keys. b) Implement a check constraint on Quantity (>0) in Orders. c) Alter the Products table to add 'Discount' column and update some values.

a) CREATE DATABASE RetailDB; USE RetailDB;

b) CREATE TABLE Customers (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100)

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY AUTO\_INCREMENT,

ProductName VARCHAR(100),

Price DECIMAL(10, 2)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT,

ProductID INT,

Quantity INT CHECK (Quantity > 0),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

c) ALTER TABLE Products ADD COLUMN Discount DECIMAL(5,2);

Q4. (5 marks) Using the above schema:

a) Insert 3 sample orders per customer.

INSERT INTO Customers (Name) VALUES ('Alice'), ('Bob');

INSERT INTO Products (ProductName, Price) VALUES ('Laptop', 50000), ('Mouse', 500), ('Keyboard', 1200);

INSERT INTO Orders (CustomerID, ProductID, Quantity) VALUES (1, 1, 2), (1, 2, 6), (1, 3, 1);

INSERT INTO Orders (CustomerID, ProductID, Quantity) VALUES (2, 2, 3), (2, 1, 1), (2, 3, 10);

b) Update prices with 10% increase where quantity sold > 5.

UPDATE Products

SET Price = Price \* 1.10

WHERE ProductID IN (

SELECT ProductID FROM Orders GROUP BY ProductID HAVING SUM(Quantity) > 5);

c) Delete orders where the product has never been sold.

DELETE FROM Products

WHERE ProductID NOT IN ( SELECT DISTINCT ProductID FROM Orders);

Q5. (5 marks) Retrieve the following:

a) Customers who ordered more than 3 different products.

SELECT CustomerID

FROM Orders

GROUP BY CustomerID

HAVING COUNT(DISTINCT ProductID) > 3;

b) Products not ordered by any customer.

SELECT \* FROM Products

WHERE ProductID NOT IN ( SELECT DISTINCT ProductID FROM Orders);

c) Count of orders placed by each customer in the last 30 days

ALTER TABLE Orders ADD COLUMN OrderDate DATE DEFAULT CURDATE();

SELECT CustomerID, COUNT(\*) AS RecentOrders FROM Orders WHERE OrderDate >= CURDATE() - INTERVAL 30 DAY GROUP BY CustomerID;

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

Q6. (5 marks)

a) Use string functions to standardize and extract parts from customer email IDs.

SELECT

LOWER(Email) AS StandardizedEmail,

SUBSTRING\_INDEX(Email, '@', 1) AS Username,

SUBSTRING\_INDEX(Email, '@', -1) AS Domain

FROM Customers;

b) Use date functions to compute days between order date and today

SELECT OrderID,

DATEDIFF(CURDATE(), OrderDate) AS DaysSinceOrder

FROM Orders;

c) Use system functions to return current user and host.

SELECT CURRENT\_USER() AS User,

HOST\_NAME() AS Host;

SELECT SYSTEM\_USER();

d) Use nested functions to format a customer greeting string.

SELECT

CONCAT('Hello ', UPPER(Name), '! Your email is ', Email) AS Greeting

FROM Customers;

Q7. (5 marks)

a) Aggregate total revenue by product category.

SELECT p.Category,

SUM(o.Quantity \* p.Price) AS TotalRevenue

FROM Orders o JOIN Products p ON o.ProductID = p.ProductID

GROUP BY p.Category;

b) Use GROUP BY with ROLLUP to compute subtotal and grand total sales.

SELECT product\_category,region,

SUM(sale\_amount) AS total\_sales

FROM sales GROUP BY product\_category, region WITH ROLLUP;

c) Use HAVING clause to filter categories with revenue > 100000.

SELECT p.Category,

SUM(o.Quantity \* p.Price) AS Revenue

FROM Orders o JOIN Products p ON o.ProductID = p.ProductID

GROUP BY p.Category

HAVING Revenue > 100000;

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

Q8. (5 marks)

1. Self join to list customers referred by other customers.

SELECT c1.Name AS Customer, c2.Name AS ReferredBy

FROM Customers c1

JOIN Customers c2 ON c1.ReferredByID = c2.CustomerID;

1. Equi join across Orders and Products.

SELECT o.OrderID, p.ProductName, o.Quantity

FROM Orders o

JOIN Products p ON o.ProductID = p.ProductID;

1. Join Customers and Orders to display top 3 spenders using window function.

SELECT \* FROM (

SELECT

c.CustomerID, c.Name,

SUM(o.Quantity \* p.Price) AS TotalSpent,

RANK() OVER (ORDER BY SUM(o.Quantity \* p.Price) DESC) AS Rank

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

JOIN Products p ON o.ProductID = p.ProductID

GROUP BY c.CustomerID

) AS Ranked

WHERE Rank <= 3;

1. LEFT OUTER JOIN with WHERE NULL to identify inactive customers

SELECT c.CustomerID, c.Name

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

WHERE o.OrderID IS NULL;

1. Cross join for all product combinations in a bundle offer.

SELECT

p1.ProductName AS ProductA,

p2.ProductName AS ProductB

FROM Products p1

CROSS JOIN Products p2

WHERE p1.ProductID < p2.ProductID;

Q9. (5 marks)

1. Correlated subquery to get customers whose order amount exceeds their average.

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);

1. Subquery using EXISTS to find customers with at least 2 different products.

SELECT \* FROM Customers c

WHERE EXISTS (

SELECT 1

FROM Orders o

WHERE o.CustomerID = c.CustomerID

GROUP BY o.ProductID

HAVING COUNT(DISTINCT o.ProductID) >= 2);

1. Use ALL to find customers who ordered more than every other customer.

SELECT c.CustomerID, c.Name

FROM Customers c

WHERE (

SELECT COUNT(\*) FROM Orders o WHERE o.CustomerID = c.CustomerID

) > ALL (

SELECT COUNT(\*) FROM Orders o2 WHERE o2.CustomerID != c.CustomerID GROUP BY o2.CustomerID

);

1. Use ANY to find products costlier than some in category 'Electronics'.

SELECT \* FROM Products

WHERE Price > ANY (

SELECT Price

FROM Products

WHERE Category = 'Electronics');

1. Nested subquery to list top 3 best-selling products.

SELECT \* FROM (

SELECT p.ProductID, p.ProductName, SUM(o.Quantity) AS TotalSold,

RANK() OVER (ORDER BY SUM(o.Quantity) DESC) AS rnk

FROM Products p

JOIN Orders o ON p.ProductID = o.ProductID

GROUP BY p.ProductID

) AS RankedProducts

WHERE rnk <= 3;

Q10. (5 marks)

1. Simulate INTERSECT using INNER JOIN on two customer segments.

SELECT pc.CustomerID, pc.Name

FROM PremiumCustomers pc

INNER JOIN LoyalCustomers lc ON pc.CustomerID = lc.CustomerID;

1. Use EXCEPT to find products in inventory not yet ordered.

SELECT p.ProductID, p.ProductName

FROM Products p

LEFT JOIN Orders o ON p.ProductID = o.ProductID

WHERE o.ProductID IS NULL;

1. Simulate MERGE: If customer exists, update; else insert.

INSERT INTO Customers (CustomerID, Name, Email)

VALUES (101, 'John Doe', '[john@example.com](mailto:john@example.com)')

ON DUPLICATE KEY UPDATE

Name = VALUES(Name),

Email = VALUES(Email);

1. Use UNION to combine two regional customer tables.

SELECT CustomerID, Name, Email FROM EastRegionCustomers

UNION

SELECT CustomerID, Name, Email FROM WestRegionCustomers;

1. Write a WITH CTE that ranks customers by total spend and filters top 5.

WITH CustomerSpending AS (

SELECT c.CustomerID, c.Name,

SUM(o.Quantity \* p.Price) AS TotalSpent,

RANK() OVER (ORDER BY SUM(o.Quantity \* p.Price) DESC) AS rnk

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

JOIN Products p ON o.ProductID = p.ProductID

GROUP BY c.CustomerID

)

SELECT CustomerID, Name, TotalSpent

FROM CustomerSpending

WHERE rnk <= 5;