## 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.

Nosql is preferred when:  
- data is unstructured or semi-structured (e.g., json, xml).  
- schema flexibility is required.  
- horizontal scalability and high-speed access are essential.  
  
Types of nosql databases and examples:  
1. Document store – mongodb: used in content management systems.  
2. Key-value store – redis: used in session management.  
3. Column-family store – cassandra: used in time-series data like logs.  
4. Graph store – neo4j: used in social networks.

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 table: customer(customerid, name, orders(orderid, productid, quantity, productname))  
  
1nf:  
customer(customerid, name)  
orders(orderid, customerid, productid, quantity, productname)  
  
2nf:  
products(productid, productname)  
orders(orderid, customerid, productid, quantity)  
  
3nf and bcnf:  
no transitive dependencies; already in bcnf.

## 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.

create database retaildb;  
create table customers (customerid int primary key, name varchar(50));  
create table products (productid int primary key, name varchar(50), price decimal(8,2));  
create table orders (orderid int primary key, customerid int, productid int, quantity int,  
foreign key (customerid) references customers(customerid),  
foreign key (productid) references products(productid),  
check (quantity > 0));

b) ) Implement a check constraint on Quantity (>0) in Orders.

Handled above with check constraint.  
C) Alter the Products table to add 'Discount' column and update some values.

Alter table products add column discount decimal(5,2);  
update products set discount = 5.00 where productid = 1;

Q4. (5 marks)  
a) Insert 3 sample orders per customer.

Insert into orders values (1, 1, 1, 2), (2, 1, 2, 3), (3, 1, 3, 1);  
b) Update prices with 10% increase where quantity sold > 5.

Update products p join orders o on p.productid = o.productid  
set p.price = p.price \* 1.10  
where o.quantity > 5;  
c) Delete orders where the product has never been sold

Delete from orders where productid not in (select productid from orders);

Q5. (5 marks)  
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.

Select customerid, count(\*) 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), substring\_index(email, '@', -1) as domain from customers;  
b) Use date functions to compute days between order date and today.

Select datediff(curdate(), orderdate) as days\_passed from orders;  
c) Use system functions to return current user and host

Select user(), system\_user();  
d) Use nested functions to format a customer greeting string.

Select concat('hello ', upper(name), '!') as greeting from customers;

Q7. (5 marks)  
a) Aggregate total revenue by product category

Select category, sum(price \* quantity) as revenue from products p join orders o on p.productid = o.productid group by category;  
b) Use GROUP BY with ROLLUP to compute subtotal and grand total sales.

Select category, sum(price \* quantity) from products p join orders o on p.productid = o.productid group by category with rollup;  
c) Use HAVING clause to filter categories with revenue > 100000  
Select category, sum(price \* quantity) as revenue from products p join orders o on p.productid = o.productid group by category having revenue > 100000;

## Section d: complex joins, subqueries, and set ops (25 marks)

Q8. (5 marks)  
a) Self join to list customers referred by other customers.

Select a.name as customer, b.name as referred\_by from customers a join customers b on a.referredby = b.customerid;  
b) Equi join across Orders and Products

Select \* from orders o join products p on o.productid = p.productid;  
c) Join Customers and Orders to display top 3 spenders using window function.

Select customerid, name, total\_spent from (  
Select o.customerid, c.name, sum(p.price \* o.quantity) as total\_spent,  
rank() over (order by sum(p.price \* o.quantity) desc) as rnk  
from orders o join products p on o.productid = p.productid join customers c on c.customerid = o.customerid  
group by o.customerid) where rnk <= 3;  
d) 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;  
e) Cross join for all product combinations in a bundle offer.

Select \* from products p1 cross join products p2 where p1.productid < p2.productid;

Q9. (5 marks)  
a) Correlated subquery to get customers whose order amount exceeds their average.

Select \* from customers c where exists (  
select 1 from orders o where o.customerid = c.customerid  
having sum(price \* quantity) > (select avg(price \* quantity) from orders where customerid = c.customerid));  
b) Subquery using EXISTS to find customers with at least 2 different products.

Select \* from customers c where exists (  
select productid from orders o where o.customerid = c.customerid group by productid having count(distinct productid) >= 2);  
c) Use ALL to find customers who ordered more than every other customer

Select \* from customers where customerid > all (select customerid from customers where customerid <> customers.customerid);  
d) Use ANY to find products costlier than some in category 'Electronics'.

Select \* from products where price > any (select price from products where category = 'electronics');  
e) Nested subquery to list top 3 best-selling products.

Select productid, sum(quantity) as total from orders group by productid order by total desc limit 3;

Q10. (5 marks)  
a) Simulate INTERSECT using INNER JOIN on two customer segments.

Select \* from segment1 inner join segment2 on segment1.customerid = segment2.customerid;  
b) Use EXCEPT to find products in inventory not yet ordered.

Select \* from products where productid not in (select productid from orders);  
c) Simulate MERGE: If customer exists, update; else insert.

Insert into customers (customerid, name) values (1, 'ram') on duplicate key update name = 'ram';  
d) Use UNION to combine two regional customer tables.

Select \* from region1 union select \* from region2;  
e) Write a WITH CTE that ranks customers by total spend and filters top 5.

With cte as (  
select customerid, sum(price \* quantity) as total\_spent from orders o join products p on o.productid = p.productid group by customerid)  
select \* from cte order by total\_spent desc limit 5;