**DAY 3:- TASK 1:-**

1) create database HexaAirlines

create database hexaairlines; use hexaairlines;

2) create tables flight, piolets ,airhostess , foodTeam, customers

create table flight(flight\_id int primary key,flight\_name varchar(50),destination varchar(50),price int);

create table piolets(piolet\_id int primary key,piolet\_name varchar(50),age int,flight\_id int,foreign key(flight\_id) references flight(flight\_id));

create table airhostess(hostess\_id int primary key,hostess\_name varchar(50),age int,experience int,flight\_id int,foreign key(flight\_id) references flight(flight\_id));

create table foodteam(food\_id int primary key,food\_type varchar(20),cost int);

create table customers(cust\_id int primary key,cust\_name varchar(50),destination varchar(50),flight\_id int,food\_id int,foreign key(flight\_id) references flight(flight\_id),foreign key(food\_id) references foodteam(food\_id));

3) fill 6 to 7 rows in these and have primary and foreign keys

insert into flight values(1,'indigo','bangalore',4500),(2,'jetairways','mumbai',5500),(3,'airindia','goa',6000),(4,'vistara','delhi',5000);

insert into piolets values(1,'ravi',45,1),(2,'arun',52,2),(3,'naveen',38,3),(4,'kumar',61,4);

insert into airhostess values(1,'meena',30,4,1),(2,'rekha',40,6,2),(3,'anu',36,7,3),(4,'seema',50,10,4);

insert into foodteam values(1,'veg',200),(2,'non-veg',300);

insert into customers values(1,'ram','bangalore',1,1),(2,'raj','mumbai',2,2),(3,'rani','goa',3,1),(4,'nina','goa',3,2),(5,'john','delhi',4,1),(6,'kavi','mumbai',2,1),(7,'lavanya','bangalore',1,2);

4) select customers going to bangalore in indigo

select cust\_name from customers where destination='bangalore' and flight\_id=1;

5) select customers going to mumbai in jetairways

select cust\_name from customers where destination='mumbai' and flight\_id=2;

6) select customers going to goa in airindia and having veg food

select cust\_name from customers where destination='goa' and flight\_id=3 and food\_id=1;

7)add new column in piolets table, covid vaccination date and fill values in column

alter table piolets add covid\_vaccine\_date date;

update piolets set covid\_vaccine\_date='2022-01-15' where piolet\_id=1;

update piolets set covid\_vaccine\_date='2022-02-10' where piolet\_id=2;

update piolets set covid\_vaccine\_date='2022-03-05' where piolet\_id=3;

update piolets set covid\_vaccine\_date='2022-04-20' where piolet\_id=4;

8) change the age of airhostess who are elder then 35 to 34

update airhostess set age=34 where age>35;

9) update the price for the customer food or flight charge and make changes

update flight set price=6500 where flight\_name='airindia';

update foodteam set cost=250 where food\_type='veg';

10) delete piolets whose age is greater then 60

delete from piolets where age>60;

11) delete airhostess who are working from last 6 years or elder then 50

delete from airhostess where experience>=6 or age>50;

12) list out the name of piolets

select piolet\_name from piolets;

13) list out the name of airhostess

select hostess\_name from airhostess;

14) use where clause to retrieve data from above tables

select \* from customers where destination='mumbai';

select \* from flight where price>5000;

select \* from foodteam where food\_type='veg';

select \* from airhostess where age<40;

**DAY3: TASK 2:-**

**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

**DAY 3:- TASK: 4:-**

**Managing Databases**

1)List all system databases in SQL Server

select name from sys.databases where database\_id <= 4;

2. List physical file paths for all databases

select name,physical\_name from sys.master\_files;

3. Create a new user-defined database named TeamDB

create database TeamDB;

4. Rename the database TeamDB to ProjectDB

alter database TeamDB modify name=ProjectDB;

5. Drop the ProjectDB database

drop database ProjectDB;

**Managing Tables**

1. Create a table Employees with the following columns

create table Employees(EmpID int primary key,Name varchar(50),Department varchar(30),JoiningDate date,IsActive bit,Salary decimal(10,2));

2. Add a column Salary (DECIMAL) to the table

alter table Employees add Salary decimal(10,2);

3. Rename table Employees to TeamMembers

exec sp\_rename 'Employees','TeamMembers';

4. Drop the table TeamMembers.

drop table TeamMembers;

**DML Operations**

1. Insert three rows into Employees

insert into Employees values(1,'Amit','HR','2022-01-01',1,50000),(2,'Sneha','IT','2021-06-15',1,75000),(3,'John','Finance','2020-10-10',0,65000);

2. Update salary of 'Sneha' to 80000

update Employees set Salary=80000 where Name='Sneha';

3. Delete employee with IsActive = 0

delete from Employees where IsActive=0;

4. Retrieve names and departments of all employees.

select Name,Department from Employees;

5. Fetch employees from 'IT' department with salary above 70000

select \* from Employees where Department='IT' and Salary>70000;

6. Apply filtering using LIKE, BETWEEN, and IN

select \* from Employees where Name like 'S%';

select \* from Employees where Salary between 60000 and 80000;

select \* from Employees where Department in('IT','Finance');

**DAY 4:- ASSIGNMENT**

1. Insert and Update with Integrity: Create a 'students' table with constraints (NOT NULL, UNIQUE). Insert 5 records. Then, update a student's marks ensuring data integrity is maintained.

QUERY:-

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));

insert into students(student\_id,name,email,marks) values(1,'john','john@gmail.com',85),(2,'alice','alice@gmail.com',92),(3,'bob','bob@gmail.com',76),(4,'neha','neha@gmail.com',88),(5,'rahul','rahul@gmail.com',67);

update students set marks=95 where student\_id=3;

select \* from students;

2. String Function Challenge: Given a 'customers' table with a 'full\_name' column, write a query to display: - First name - Last name - Length of each name

QUERY:-

create table customers(customer\_id int primary key,full\_name varchar(100));

insert into customers(customer\_id,full\_name) values(1,'kavya dharshni'),(2,'theshna shree');

select 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: From a 'sales' table with a 'sale\_date' column, write a query to: - Extract the month name and year - Display how many days ago the sale happened

QUERY:-

create table sales(sale\_id int primary key,sale\_date date);

insert into sales(sale\_id,sale\_date) values(1,'2024-06-01'),(2,'2025-06-10');

select sale\_date, monthname(sale\_date) as month\_name, year(sale\_date) as year,

datediff(curdate(),sale\_date) as days\_ago from sales;

4. Mathematical Functions on Salary: In an 'employees' table, calculate: - Salary after a 10% hike - Round the salary to the nearest hundred

QUERY:-

create table employees(emp\_id int primary key,name varchar(50),salary decimal(10,2));

insert into employees(emp\_id,name,salary) values(1,'YUVA',43250),(2,'THESHI',51890);

select emp\_id,name,salary,salary\*1.10 as hiked\_salary,round(salary,-2) as rounded\_salary

from employees;

5. System Function Check: Retrieve: - Current date and time - Database name and logged-in user

QUERY:-

Select now() as currentdatetime,current\_date() as currentdate,current\_time() ascurrenttime,

database() as currentdatabase,user() as loggedinuser;

6. Demo: Custom Result Set: From the 'products' table, write a query that: - Returns product name in uppercase - Replaces any NULL prices with 'Not Available'

QUERY:-

create table products(productid int primary key,productname varchar(50),price decimal(10,2));

insert into products(productid,productname,price) values(1,'pen',25.50),(2,'notebook',null);

select upper(productname) as productname, ifnull(cast(price as char),'Not Available') as price from products;

7. Aggregate Functions Practice: From a 'transactions' table, get: - Total sales - Average sale value - Maximum and minimum sale on a single transaction

QUERY:-

Select sum(amount) as totalsales, avg(amount) as averagesale,max(amount) as maxsale,

min(amount) as minsale from transactions;

8. Grouping with Aggregation: From a 'sales' table: - Group by product category - Show total sales and number of transactions in each category

QUERY:-

Select category,sum(amount) as totalsales,count(\*) as transactioncount from salesgroup by category;

9. Inner Join for Orders and Customers: Join 'orders' and 'customers' to show: - Customer name - Order amount - Only for customers who made orders

create table customers(id int primary key,name varchar(50));

create table orders(id int primary key,customerid int,amount decimal(10,2),foreign key(customerid) references customers(id));

insert into customers(id,name) values(1,'kavya'),(2,'theshna'),(3,'aarav');

insert into orders(id,customerid,amount) values(1,1,250.00),(2,2,500.00);

select c.name as customername,o.amount as orderamount from customers c inner join orders o on c.id=o.customerid;

10. Left Join for Products with or without Orders: Show all products with: - Their order details (if available) - Use LEFT JOIN

QUERY:-

Select p.productname, o.orderid,o.quantity,o.orderdate from products pleft join orders o on p.productid=o.productid;

11. Right Join for Customer Contacts: Use a RIGHT JOIN between 'contacts' and 'customers' to display: - All customers, even if they don''t have contact info

QUERY:-

Select c.name as customername,ct.phone as contactphone,ct.email as contactemail

from contacts ct right join customers c on ct.customerid=c.id;

12. Full Outer Join for Suppliers and Products: Use a FULL OUTER JOIN to list: - All suppliers and products - Match supplier to product, or show NULLs where not available

QUERY:-

Select s.suppliername, p.productname from suppliers s left join products p on s.id = p.supplierid union

Select s.suppliername,p.productname from suppliers s right join products p on s.id = p.supplierid;

13. Cross Join for Offers: Suppose you have tables 'products' and 'offers'. Write a CROSS JOIN to show: - All possible combinations of products and offers

QUERY:-

Select p.productname,o.offername from products p cross join offers o;

14. Join with Aggregation: Join 'orders' and 'products', then group by product category and: - Show total quantity sold and average price per

QUERY:-

Select p.category,sum(o.quantity) as totalquantity,avg(p.price) as averageprice

from orders o join products p on o.productid = p.id group by p.category;

15. . Demo: Join with Grouping and Filter: Join 'students' and 'marks' tables. Display: - Student name - Average marks - Filter to show only students with average marks > 75

QUERY:-

Select s.name as studentname,avg(m.mark) as averagemark from students s

join marks m on s.id = m.studentid group by s.name having avg(m.mark) > 75;

**Day:-5**

1) Querying Data by Using Subqueries  
select name from studentt where marks=(select max(marks) from studentt);

2) Querying Data by Using Subqueries Using the EXISTS,

select name from studentt s where exists(select 1 from studentt where grade=s.grade and marks>85);

3) select name from studentt where marks>any(select marks from studentt where grade='b');

select name from studentt where marks>any(select marks from studentt where grade='b');

4) Querying Data by Using Subqueries using ALL Keywords

select name from studentt where marks>all(select marks from studentt where grade='c');

5) Querying Data by Using Subqueries using Using Nested Subqueries

select name from studentt where marks=(select max(marks) from studentt where marks<(select max(marks) from studentt));

6) Querying Data by Using Subqueries Using Correlated Subqueries

select name from studentt s where marks>(select avg(marks) from studentt where grade=s.grade);

7) Querying Data by Using Subqueries Using UNION,

select name from studentt where marks>85 union select name from studentt where grade='c';

8) Querying Data by Using Subqueries using INTERSECT

select name from studentt where marks>70 intersect select name from studentt where grade='b';

9) Querying Data by Using Subqueries using EXCEPT,

select name from studentt where marks>70 except select name from studentt where grade='b';

10) Querying Data by Using Subqueries using MERGE

merge into studentt as t using(select 6 as student\_id,'gokul' as name,78 as marks,'b' as grade) as s on(t.student\_id=s.student\_id) when matched then update set t.marks=s.marks when not matched then insert(student\_id,name,marks,grade) values(s.student\_id,s.name,s.marks,s.grade);

# **DAY 5:- TASK 2:-**

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

Q1. (3 marks)

Differentiate between SQL and NoSQL. Provide two advantages and two disadvantages of each with real-world examples.

|  |  |
| --- | --- |
| SQL | NOSQL |
| It has data in the rows and column format | It doesn’t have any structure to organize the data |
| The schema is in fixed type | The schema is flexible |
| It uses sql languages | It uses MongoDB which is applicable for large data |

Realtime example for SQL: banking system, College management

Realtime example for NoSQL: social media platforms like facebook,instagarm.

Q2. (2 marks)

Given the below unnormalized data, convert it to 1NF, 2NF, and 3NF:

Student (StudentID,Name,CourseID,CourseName,InstructorName,InstructorPhone)

1Nf- all the data should be atomic

Student(StudentID,Name,CourseID,CourseName,InstructorName,InstructorPhone)

2NF- should be one nF but doesn’t have partial dependency

Student(StudentID,Name)) Course(CourseID,CourseName,InstructorName,InstructorPhone) Enrollment(StudentID, CourseID)

3NF- must be 2NF but does not hava transitive dependencies

Student(StudentID,Name) Course(CourseID,CourseName,InstructorName) Instructor(InstructorName,InstructorPhone) Enrollment(StudentID,CourseID)

Q3. (5 marks)

1. Create a database named StudentDB. create database studentdb;

create database studentdb;

1. Create a table Students with fields: StudentID, Name, DOB, Email.

create table students(studentid int,name varchar(50),dob date,email varchar(100));

1. Rename the table to Student\_Info.

rename table students to student\_info;

1. Add a column PhoneNumber.

alter table student\_info add phonenumber varchar(15);

1. Drop the table.

drop table student\_info;

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

Q4. (5 marks)

1. Insert 3 student records into Student\_Info.

insert into student\_info(studentid,name,dob,email,phonenumber) values

(1,'yuva','2003-05-14','yuva@gmail.com','9876543210'),

(2,'theshi','2002-11-22','theshi@gmail.com','8765432109'),

(3,'kavya','2004-03-30','kavya@gmail.com','7654321098');

1. Update one student's phone number.

update student\_info set phonenumber='9123456780' where studentid=1;

1. Delete one student whose email ends with @gmail.com.

delete from student\_info where email like '%@gmail.com' limit 1;

1. Retrieve only names and emails of students born after the year 2000.

select name,email from student\_info where year(dob)>2000;

1. Retrieve distinct domain names from the email column.

select distinct substring\_index(email,'@',-1) as domain from student\_info;

Q5. (5 marks)

1. Retrieve students with names starting with 'A'.

select \* from student\_info where name like 'a%';

1. Retrieve students with phone number between 9000000000 and 9999999999.

select \* from student\_info where phonenumber between “9000000000” and “9999999999”;

1. Retrieve students using IN operator on city names.

select \* from student\_info where city in('chennai','Namakkal','coimbatore');

1. Use AND, OR to filter students based on age and email provider.

select \* from student\_info where (year(dob)>2000 and email like '%@gmail.com') or (year(dob)<2000 and email like '%@yahoo.com');

1. Use table and column aliasing in a query to get all student names and DOBs.

select s.name as student\_name,s.dob as birth\_date from student\_info as s;

Q6. (5 marks)

a)Create a new table Marks(StudentID, Subject, Marks). Insert at least 3 rows.

create table marks(studentid int,subject varchar(50),marks int);

insert into marks(studentid,subject,marks) values

(1,”maths”,85),

(2,”science”,90),

(3,”english”,78);

b)Display student IDs and their subjects where marks > 70.

select studentid,subject from marks where marks>70;

c)Display subjects with average marks.

select subject,avg(marks) as average\_marks from marks group by subject;

d)Filter subjects with average marks between 60 and 90.

select subject,avg(marks) as average\_marks from marks group by subject having avg(marks) between 60 and 90;

# Section C: Functions & Grouping (10 Marks)

Q7. (5 marks)

1. Get the current date and format it as "YYYY-MM-DD".

select date\_format(curdate(),'%Y-%m-%d') as todaydate;

1. Extract month and year from a DOB column.

select month(dob) as birth\_month,year(dob) as birth\_year from student\_info;

1. Convert a student's name to uppercase.

select upper(name) as uppercase\_name from student\_info;

1. Round off marks to 2 decimal places.

select round(marks,2) as rounded\_marks from marks;

1. Use system function to return user name or current database.

select user() as current\_user,database() as currentdatabase;

Q8. (5 marks)

1. Display total marks of each student.

select studentid,sum(marks) as totalmarks from marks group by studentid;

1. Display subject-wise highest mark.

select subject,max(marks) as highest\_mark from marks group by subject;

1. Use GROUP BY and HAVING to display subjects with average marks > 75.

select subject,avg(marks) as average\_marks from marks group by subject having avg(marks)>75;

# Section D: Joins and Subqueries (25 Marks)

Q9. (5 marks)

create table courses(courseid int,course\_name varchar(50));

create table enrollments(studentid int,courseid int);

insert into courses(courseid,course\_name) values(101,'maths'),(102,'science'),(103,'english');

insert into enrollments(studentid,courseid) values(1,101),(2,102);

1. Inner Join to retrieve students and their courses.

select s.studentid,s.name,c.course\_name from student\_info s inner join enrollments e on s.studentid=e.studentid

inner join courses c on e.courseid=c.courseid;

1. Left Join to get all students even if not enrolled.

select s.studentid,s.name,c.course\_name from student\_info s left join enrollments e on s.studentid=e.studentid

left join courses c on e.courseid=c.courseid;

1. Right Join to get all courses even if no students.

select s.studentid,s.name,c.course\_name from student\_info s right join enrollments e on s.studentid=e.studentid

right join courses c on e.courseid=c.courseid;

1. Full Outer Join equivalent using UNION.

select s.studentid,s.name,c.course\_name from student\_info s left join enrollments e on s.studentid=e.studentid

left join courses c on e.courseid=c.courseid union select s.studentid,s.name,c.course\_name

from student\_info s right join enrollments e on s.studentid=e.studentid

right join courses c on e.courseid=c.courseid;

1. Cross Join to show all combinations.

select s.studentid,s.name,c.course\_name from student\_info s cross join courses c;

Q10. (5 marks)

1. Students who scored more than average in 'Maths'.

select studentid,marks from marks where subject='maths' and marks > (select avg(marks) from marks where subject='maths');

1. Students not in the Marks table.

select \* from student\_info where studentid not in (select distinct studentid from marks);

1. Use EXISTS to get students with at least one subject.

select \* from student\_info s where exists ( select 1 from marks m where m.studentid=s.studentid);

1. Use ALL to find those scoring more than all in 'Science'.

select \* from marks where marks > all (select marks from marks where subject='science');

1. Use ANY for students scoring better than some in 'English'.

select \* from marks where marks > any (select marks from marks where subject='english');

Q11. (5 marks)

create table student2(studentid int,name varchar(50));

insert into student2(studentid,name) values (5,”arjun”),(6,”divya”),(3,”kavya”);

1. UNION of student names from two tables.

select name from student\_info union select name from student2;

1. INTERSECT to find common students.

select name from student\_info where name in(select name from student2);

1. EXCEPT to list students in Students but not in Marks.

select \* from student\_info where studentid not in(select studentid from marks);

1. MERGE concept or simulate with UPDATE and INSERT.

update student\_info set email='yuvanewmail@gmail.com' where studentid=1;

insert into student\_info(studentid,name,dob,email,phonenumber)

select 4,'newstudent','2005-01-01','new@gmail.com','9123456789';

where not exists(select 1 from student\_info where studentid=4);

1. Correlated subquery to list students with above average per subject

select \* from marks m1 where marks>(select avg(marks) from marks m2 where m2.subject=m1.subject);

# 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 always stores unstructured data while sql has fixed schema to store the data . NOSQL can handle large amount of data than sql. NOSQL need not to be predefined it can be changed at any time.

Types of Nosql : Document-oriented,keyvalue stores,column-oriented, graph databases

Realtime application:-sql:- ERP(oracle) NOSQL:-social media plateforms

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.

1nf:-

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

2nf:-

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

BCNF:-

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

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

Q3. (5 marks)

1. Create a database RetailDB and design a schema for Customers, Orders, and Products with primary and foreign keys.

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,orderdate date,foreign key(customerid) references customers(customerid),foreign key(productid) references products(productid));

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

alter table orders add constraint chk\_quantity check(quantity > 0);

1. Alter the Products table to add 'Discount' column and update some values.

alter table products add discount decimal(5,2); update products set discount=10.00 where productid=1; update products set discount=5.00 where productid=2;

Q4. (5 marks)

Using the above schema:

1. Insert 3 sample orders per customer.

insert into orders(orderid,customerid,productid,quantity,orderdate) values(1,1,1,2,'2025-06-10'),(2,1,2,4,'2025-06-11'),(3,1,3,1,'2025-06-12'),(4,2,1,6,'2025-06-10'),(5,2,2,3,'2025-06-11'),(6,2,3,8,'2025-06-12');

1. 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);

1. Delete orders where the product has never been sold.

delete from orders where productid not in(select distinct productid from orders);

Q5. (5 marks) Retrieve the following:

1. Customers who ordered more than 3 different products.

select customerid from orders group by customerid having count(distinct productid)>3;

1. Products not ordered by any customer.

select \* from products where productid not in(select distinct productid from orders);

1. Count of orders placed by each customer in the last 30 days.

select customerid,count(\*) as order\_count from orders where orderdate >= curdate() - interval 30 day group by customerid;

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

Q6. (5 marks)

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

select customerid,lower(email) as standardized\_email,substring\_index(email,'@',1) as username,substring\_index(email,'@',-1) as domain from customers;

1. Use date functions to compute days between order date and today.

select orderid,datediff(curdate(),orderdate) as days\_since\_order from orders;

1. Use system functions to return current user and host.

select user(),current\_user(),version();

1. Use nested functions to format a customer greeting string.

select concat('hello ',upper(left(name,1)),lower(substring(name,2))) as greeting from customers;

Q7. (5 marks)

alter table products add category varchar(50);

1. Aggregate total revenue by product category.

select p.category,sum(p.price\*o.quantity) as total\_revenue from products p join orders o on p.productid=o.productid group by p.category;

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

select p.category,sum(p.price\*o.quantity) as total\_revenue from products p join orders o on p.productid=o.productid group by p.category with rollup;

1. Use HAVING clause to filter categories with revenue > 100000.

select p.category,sum(p.price\*o.quantity) as total\_revenue from products p join orders o on p.productid=o.productid group by p.category having total\_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.customerid as referred\_customer,c1.name as customer\_name,c2.name as referrer\_name from customers c1 join customers c2 on c1.referred\_by=c2.customerid;

1. Equi join across Orders and Products.

select o.orderid,o.customerid,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 customerid,name,total\_spent from (select c.customerid,c.name,sum(p.price\*o.quantity) as total\_spent,row\_number() over(order by sum(p.price\*o.quantity) 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,c.name) 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.customerid is null;

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

select p1.productname as product1,p2.productname as product2 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 o.customerid,o.orderid,sum(p.price\*o.quantity) as order\_total from orders o join products p on o.productid=p.productid group by o.orderid,o.customerid having order\_total > (select avg(p2.price\*o2.quantity) from orders o2 join products p2 on o2.productid=p2.productid where o2.customerid=o.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.customerid having count(distinct o.productid)>=2);

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

select customerid from (select customerid,count(\*) as order\_count from orders group by customerid) as t where order\_count > all(select count(\*) from orders group by 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 productid,productname,total\_sold from (select p.productid,p.productname,sum(o.quantity) as total\_sold,row\_number() over(order by sum(o.quantity) desc) as rk from products p join orders o on p.productid=o.productid group by p.productid,p.productname) as ranked where rk<=3;

Q10. (5 marks)

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

select cn.customerid,cn.name from customer\_north cn inner join customer\_south cs on cn.customerid=cs.customerid;

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

select \* from products where productid not in(select distinct productid from orders);

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

update customers set email='newemail@gmail.com' where customerid=4; insert into customers(customerid,name,email) select 4,'newcustomer','new@gmail.com' where not exists(select 1 from customers where customerid=4);

1. Use UNION to combine two regional customer tables.

Write a WITH CTE that ranks customers by total spend and filters

select customerid,name,email from customer\_north union select customerid,name,email from customer\_south;