Theory Questions

Question 1 : Explain the fundamental differences between DDL, DML, and DQLcommands in SQL. Provide one example for each type of command.

1️⃣DDL – Data Definition Language 🔹 What it does

DDL commands are used to define, create, change, or delete the structure of database objects like tables, schemas, or databases.

👉 Think of DDL as designing the database layout.

🔹 Key features

Works on structure, not data

Changes are usually permanent (auto-commit)

Affects tables, columns, constraints

🔹 Common DDL commands

CREATE, ALTER, DROP, TRUNCATE

🔹 Example CREATE TABLE Students (student_id INT, name VARCHAR(50), age INT);

📌 This creates the structure of the Students table.

2️⃣DML – Data Manipulation Language 🔹 What it does

DML commands are used to insert, update, delete, or modify the data inside tables.

👉 Think of DML as working with the actual records.

🔹 Key features

Works on data

Changes can be rolled back (if transaction control is used)

Does NOT change table structure

🔹 Common DML commands

INSERT, UPDATE, DELETE

🔹 Example INSERT INTO Students VALUES (1, 'Amit', 18);

📌 This inserts data into the table, not the structure.

3️⃣DQL – Data Query Language 🔹 What it does

DQL commands are used to retrieve or query data from the database.

👉 Think of DQL as asking questions to the database.

🔹 Key features

Read-only

Does NOT modify data

Used for reporting and analysis

🔹 Common DQL command

SELECT

🔹 Example SELECT * FROM Students;

📌 This fetches and displays data from the table.

🔁 Quick Comparison Table Type Full Form Purpose Example DDL Data Definition Language Defines structure CREATE TABLE DML Data Manipulation Language Modifies data INSERT INTO DQL Data Query Language Retrieves data SELECT

Question 2 : What is the purpose of SQL constraints? Name and describe three common types of constraints, providing a simple scenario where each would be useful.

🔹 What is the purpose of SQL Constraints?

SQL constraints are rules applied to table columns to ensure data accuracy, consistency, and integrity in a database.

👉 In simple words: Constraints stop wrong or invalid data from entering the database.

🔹 Why do we need constraints?

To maintain data quality

To avoid duplicate or invalid values

To enforce relationships between tables

To protect the database from human mistakes

🔹 Three Common SQL Constraints (with scenarios)1️⃣PRIMARY KEY Constraint

🔸 What it does

Ensures each row has a unique identifier

Cannot be NULL

No duplicate values allowed

🔸 When it is useful

Used when each record must be uniquely identifiable.

🔸 Example scenario

In a student database, each student must have a unique student ID.

🔸 Example CREATE TABLE Students (student_id INT PRIMARY KEY, name VARCHAR(50), age INT);

📌 This ensures no two students can have the same student_id.

2️⃣NOT NULL Constraint 🔸 What it does

Ensures a column cannot have NULL (empty) values

🔸 When it is useful

Used when a field is mandatory.

🔸 Example scenario

In an employee table, every employee must have a name.

🔸 Example CREATE TABLE Employees (emp_id INT, emp_name VARCHAR(50) NOT NULL, salary INT);

📌 This ensures emp_name cannot be left empty.

3️⃣UNIQUE Constraint 🔸 What it does

Ensures all values in a column are different

Allows only one NULL value (in most databases)

🔸 When it is useful

Used when data should be unique but not necessarily a primary key.

🔸 Example scenario

In a user system, each user must have a unique email ID.

🔸 Example CREATE TABLE Users (user_id INT, email VARCHAR(100) UNIQUE);

📌 This ensures no two users can register with the same email.

🔁 Quick Summary Table Constraint Purpose Simple Use Case PRIMARY KEY Unique + Not Null Student ID NOT NULL Mandatory field Employee name UNIQUE No duplicates Email ID

Question 3 : Explain the difference between LIMIT and OFFSET clauses in SQL. How would you use them together to retrieve the third page of results, assuming each page has 10 records?

🔹 What is LIMIT in SQL? 👉 Purpose

LIMIT is used to restrict how many rows are returned by a query.

👉 In simple words

“Show me only this many records.”

🔹 Example SELECT * FROM Students LIMIT 10;

📌 This returns only 10 records, even if the table has 1000 rows.

🔹 What is OFFSET in SQL? 👉 Purpose

OFFSET is used to skip a specific number of rows before starting to return results.

👉 In simple words

“Ignore the first N records, then show the result.”

🔹 Example SELECT * FROM Students OFFSET 5;

📌 This skips the first 5 records and shows the rest.

🔹 Difference between LIMIT and OFFSET Clause What it does LIMIT Controls how many rows to display OFFSET Controls how many rows to skip 🔹 Using LIMIT and OFFSET together (Pagination) 📄 Assumption

Each page has 10 records

We want the 3rd page

🧠 Calculation

Page 1 → skip 0 records

Page 2 → skip 10 records

Page 3 → skip 20 records

👉 Formula:

OFFSET = (page_number - 1) × records_per_page OFFSET = (3 - 1) × 10 = 20

✅ SQL Query for 3rd Page SELECT * FROM Students LIMIT 10 OFFSET 20;

📌 Meaning:

Skip the first 20 records

Return the next 10 records

Question 4 : What is a Common Table Expression (CTE) in SQL, and what are its main benefits? Provide a simple SQL example demonstrating its usage.

🔹 What is a Common Table Expression (CTE)?

A Common Table Expression (CTE) is a temporary named result set that you can use inside a SELECT, INSERT, UPDATE, or DELETE query.

👉 In simple words:

A CTE is like creating a temporary table that exists only for one query.

It is defined using the WITH keyword.

🔹 Why do we use CTEs? (Main Benefits)1️⃣Improves Readability

Makes complex queries easier to understand

Breaks big queries into logical steps

2️⃣Reusability within a Query

You can refer to the CTE multiple times in the same query

Avoids repeating the same subquery again and again

3️⃣Easier Alternative to Subqueries

Cleaner and more readable than nested subqueries

Very useful in interviews and real projects

4️⃣Supports Recursive Queries

Used for hierarchical data (manager–employee, category trees)

🔹 Simple SQL Example Using CTE 📌 Scenario

We want to find students whose marks are greater than the average marks.

✅ SQL Query with CTE WITH AvgMarks AS (SELECT AVG(marks) AS avg_marks FROM Students) SELECT name, marks FROM Students WHERE marks > (SELECT avg_marks FROM AvgMarks);

🔍 Explanation (step-by-step)

The CTE AvgMarks calculates the average marks

The main query uses that result to filter students

The CTE exists only during this query execution

🔁 Same logic without CTE (harder to read) SELECT name, marks FROM Students WHERE marks > (SELECT AVG(marks) FROM Students);

📌 Works, but CTE is clearer for complex logic.

Question 5 : Describe the concept of SQL Normalization and its primary goals. Briefly explain the first three normal forms (1NF, 2NF, 3NF).

🔹 What is SQL Normalization?

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity.

👉 In simple words:

Normalization means storing data in a clean, structured way so that the same data is not repeated again and again.

🔹 Primary Goals of Normalization

Reduce data redundancy (avoid duplicate data)

Improve data consistency

Prevent update, insert, and delete anomalies

Maintain data integrity

Make the database easier to maintain

🔹 Normal Forms Explained (1NF, 2NF, 3NF)1️⃣First Normal Form (1NF) 🔸 Rule

Each column must contain atomic (indivisible) values

No repeating groups or multi-valued columns

❌ Before 1NF (Not normalized) StudentID Name Subjects 1 Ravi Math, Science ✅ After 1NF StudentID Name Subject 1 Ravi Math 1 Ravi Science

📌 Each cell now has only one value.

2️⃣Second Normal Form (2NF) 🔸 Rule

Must be in 1NF

No partial dependency

Non-key columns should depend on the entire primary key

🔍 Usually applies to tables with composite keys ❌ Before 2NF

Table: StudentCourses Primary Key: (StudentID, CourseID)

StudentID CourseID StudentName 1 C1 Ravi

📌 StudentName depends only on StudentID, not on full key.

✅ After 2NF

Students Table

StudentID StudentName 1 Ravi

Courses Table

CourseID CourseName

📌 Now every non-key column depends on the full primary key.

3️⃣ Third Normal Form (3NF) 🔸 Rule

Must be in 2NF

No transitive dependency

Non-key column should not depend on another non-key column

❌ Before 3NF StudentID DeptID DeptName

📌 DeptName depends on DeptID, not directly on StudentID.

✅ After 3NF

Students Table

StudentID DeptID

Departments Table

DeptID DeptName

📌 All non-key attributes depend only on the primary key.

🔁 Quick Summary Table Normal Form Main Rule 1NF Atomic values, no repeating groups 2NF No partial dependency 3NF No transitive dependency

Practical Questions

-- Question no 6-- Step 1: Create the database CREATE DATABASE ECommerceDB;-- Step 2: Use the database USE ECommerceDB;-- Step 3: Create Categories table CREATE TABLE Categories (CategoryID INT PRIMARY KEY, CategoryName VARCHAR(50) NOT NULL UNIQUE);-- Step 4: Create Customers table CREATE TABLE Customers (CustomerID INT PRIMARY KEY, CustomerName VARCHAR(100) NOT NULL, Email VARCHAR(100) UNIQUE, JoinDate DATE);-- Step 5: Create Products table with foreign key reference to Categories CREATE TABLE Products (ProductID INT PRIMARY KEY, ProductName VARCHAR(100) NOT NULL UNIQUE, CategoryID INT, Price DECIMAL(10,2) NOT NULL, StockQuantity INT, FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID));-- Step 6: Create Orders table with foreign key reference to Customers CREATE TABLE Orders (OrderID INT PRIMARY KEY, CustomerID INT, OrderDate DATE NOT NULL, TotalAmount DECIMAL(10,2), FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));-- Step 7: Insert data into Categories table INSERT INTO Categories VALUES (1, 'Electronics'), (2, 'Books'), (3, 'Home Goods'), (4, 'Apparel');-- Step 8: Insert data into Products table INSERT INTO Products VALUES (101, 'Laptop Pro', 1, 1200.00, 50), (102, 'SQL Handbook', 2, 45.50, 200), (103, 'Smart Speaker', 1, 99.99, 150), (104, 'Coffee Maker', 3, 75.00, 80), (105, 'Novel: The Great SQL', 2, 25.00, 120), (106, 'Wireless Earbuds', 1, 150.00, 100), (107, 'Blender X', 3, 120.00, 60), (108, 'T-Shirt Casual', 4, 20.00, 300);-- Step 9: Insert data into Customers table INSERT INTO Customers VALUES (1, 'Alice Wonderland', 'alice@example.com', '2023-01-10'), (2, 'Bob the Builder', 'bob@example.com', '2022-11-25'), (3, 'Charlie Chaplin', 'charlie@example.com', '2023-03-01'), (4, 'Diana Prince', 'diana@example.com', '2021-04-26');-- Step 10: Insert data into Orders table INSERT INTO Orders VALUES (1001, 1, '2023-04-26', 1245.50), (1002, 2, '2023-10-12', 99.99), (1003, 1, '2023-07-01', 145.00), (1004, 3, '2023-01-14', 150.00), (1005, 2, '2023-09-24', 120.00), (1006, 1, '2023-06-19', 20.00);-- Step 11: Display data from all tables for verification SELECT * FROM

Categories; SELECT * FROM Products; SELECT * FROM Customers; SELECT * FROM Orders;

Answer # ✅ Step 1: Create Database CREATE DATABASE ECommerceDB; USE ECommerceDB; # # ✅ 📦 Step 2: Create Tables with Constraints Categories Table CREATE TABLE Categories (CategoryID INT PRIMARY KEY, CategoryName VARCHAR(50) NOT NULL UNIQUE); # 🔲 🔲 Products Table CREATE TABLE Products (ProductID INT PRIMARY KEY, ProductName VARCHAR(100) NOT NULL UNIQUE, CategoryID INT, Price DECIMAL(10,2) NOT NULL, StockQuantity INT, FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)); 👤 # Customers Table CREATE TABLE Customers (CustomerID INT PRIMARY KEY, CustomerName VARCHAR(100) NOT NULL, Email VARCHAR(100) UNIQUE, JoinDate DATE); # 🔲 🔲 Orders Table CREATE TABLE Orders (OrderID INT PRIMARY KEY, CustomerID INT, OrderDate DATE NOT NULL, TotalAmount DECIMAL(10,2), FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)); # # # ✅ ➕ ➕ Step 3: Insert Records Insert into Categories INSERT INTO Categories VALUES (1, 'Electronics'), (2, 'Books'), (3, 'Home Goods'), (4, 'Apparel'); Insert into Products INSERT INTO Products VALUES ➕ (101, 'Laptop Pro', 1, 1200.00, 50), (102, 'SQL Handbook', 2, 45.50, 200), (103, 'Smart Speaker', 1, 99.99, 150), (104, 'Coffee Maker', 3, 75.00, 80), (105, 'Novel: The Great SQL', 2, 25.00, 120), (106, 'Wireless Earbuds', 1, 150.00, 100), (107, 'Blender X', 3, 120.00, 60), (108, 'T-Shirt Casual', 4, 20.00, 300); # Insert into Customers INSERT INTO Customers VALUES ➕ (1, 'Alice Wonderland', 'alice@example.com', '2023-01-10'), (2, 'Bob the Builder', 'bob@example.com', '2022-11-25'), (3, 'Charlie Chaplin', 'charlie@example.com', '2023-03-01'), (4, 'Diana Prince', 'diana@example.com', '2021-04-26'); # Insert into Orders INSERT INTO Orders VALUES (1001, 1, '2023-04-26', 1245.50), (1002, 2, '2023-10-12', 99.99), (1003, 1, '2023-07-01', 145.00), (1004, 3, '2023-01-14', 150.00), (1005, 2, '2023-09-24', 120.00), (1006, 1, '2023-06-19', 20.00); # ✅ Step 4: Verify Data SELECT * FROM Categories; SELECT * FROM Products; SELECT * FROM Customers; SELECT * FROM Orders;

-- Question no 7-- Display total number of orders for each customer-- Include customers who have not placed any orders-- Sort the result by CustomerName SELECT c.CustomerID, c.CustomerName, COUNT(o.OrderID) AS TotalNumberofOrders FROM Customers c LEFT JOIN Orders o ON c.CustomerID = o.CustomerID GROUP BY c.CustomerID, c.CustomerName ORDER BY c.CustomerName;

Answer SELECT c.CustomerID, c.CustomerName, COUNT(o.OrderID) AS TotalNumberofOrders FROM Customers c LEFT JOIN Orders o ON c.CustomerID = o.CustomerID GROUP BY c.CustomerID, c.CustomerName ORDER BY c.CustomerName; #Why this works #LEFT JOIN ensures customers with no orders are included #COUNT(o.OrderID) returns 0 when there are no matching orders #GROUP BY aggregates orders per customer #ORDER BY CustomerName sorts the results alphabetically

-- Question no 8-- Display product name, price, stock quantity, and category name-- Join Products table with Categories table-- Sort results by CategoryName and then ProductName alphabetically SELECT p.ProductName, p.Price, p.StockQuantity, c.CategoryName FROM Products p INNER JOIN Categories c ON p.CategoryID = c.CategoryID ORDER BY c.CategoryName, p.ProductName;

#Answer SELECT p.ProductName, p.Price, p.StockQuantity, c.CategoryName FROM Products p INNER JOIN Categories c ON p.CategoryID = c.CategoryID ORDER BY c.CategoryName, p.ProductName;

-- Question 9 -- Use a Common Table Expression (CTE) to rank products by price within each category-- ROW_NUMBER() assigns a unique rank to products in descending order of price-- Retrieve the top 2 most expensive products from each category WITH RankedProducts AS (SELECT c.CategoryName, p.ProductName, p.Price, ROW_NUMBER() OVER (PARTITION BY c.CategoryName ORDER BY p.Price DESC) AS PriceRank FROM Products p INNER JOIN Categories c ON p.CategoryID = c.CategoryID)-- Select only the top 2 products per category based on price SELECT CategoryName, ProductName, Price FROM RankedProducts WHERE PriceRank <= 2 ORDER BY CategoryName, Price DESC;

#Answer WITH RankedProducts AS (SELECT c.CategoryName, p.ProductName, p.Price, ROW_NUMBER() OVER (PARTITION BY c.CategoryName ORDER BY p.Price DESC) AS PriceRank FROM Products p INNER JOIN Categories c ON p.CategoryID = c.CategoryID) SELECT CategoryName, ProductName, Price FROM RankedProducts WHERE PriceRank <= 2 ORDER BY CategoryName, Price DESC;

-- Question no 10 -- Display top 5 customers based on total payment amount-- Include customer name, email, and total amount spent SELECT CONCAT(c.first_name, ' ', c.last_name) AS CustomerName, c.email, SUM(p.amount) AS TotalAmountSpent FROM customer c INNER JOIN payment p ON c.customer_id = p.customer_id GROUP BY c.customer_id, c.first_name, c.last_name, c.email ORDER BY TotalAmountSpent DESC LIMIT 5;-- Identify top 3 movie categories based on rental count-- Display category name and number of rentals SELECT cat.name AS CategoryName, COUNT(r.rental_id) AS RentalCount FROM category cat INNER JOIN film_category fc ON cat.category_id = fc.category_id INNER JOIN inventory i ON fc.film_id = i.film_id INNER JOIN rental r ON i.inventory_id = r.inventory_id GROUP BY cat.name ORDER BY RentalCount DESC LIMIT 3;-- Display total films available at each store-- Also calculate how many films have never been rented SELECT s.store_id, COUNT(i.inventory_id) AS TotalFilms, SUM(CASE WHEN r.rental_id IS NULL THEN 1 ELSE 0 END) AS NeverRentedFilms FROM store s INNER JOIN inventory i ON s.store_id = i.store_id LEFT JOIN rental r ON i.inventory_id = r.inventory_id GROUP BY s.store_id; -- Calculate total revenue per month for the year 2023-- Used to analyze seasonality in business SELECT YEAR(p.payment_date) AS Year, MONTH(p.payment_date) AS Month, SUM(p.amount) AS TotalRevenue FROM payment p WHERE YEAR(p.payment_date) = 2023 GROUP BY YEAR(p.payment_date), MONTH(p.payment_date) ORDER BY Month; -- Identify customers who rented more than 10 times in the last 6 months -- Uses current date to calculate recent rentals SELECT CONCAT(c.first_name, ' ', c.last_name) AS CustomerName, COUNT(r.rental_id) AS RentalCount FROM customer c INNER JOIN rental r ON c.customer_id = r.customer_id WHERE r.rental_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH) GROUP BY c.customer_id, c.first_name, c.last_name HAVING COUNT(r.rental_id) > 10 ORDER BY RentalCount DESC;

#Answer ✅ # Question 10 – Sakila Video Rentals Analysis #1. Top 5 customers based on total amount spent SELECT CONCAT(c.first_name, ' ', c.last_name) AS CustomerName,

c.email, SUM(p.amount) AS TotalAmountSpent FROM customer c INNER JOIN payment p ON c.customer_id = p.customer_id GROUP BY c.customer_id, c.first_name, c.last_name, c.email ORDER BY TotalAmountSpent DESC LIMIT 5; SELECT cat.name AS CategoryName, COUNT(r.rental_id) AS RentalCount FROM category cat INNER JOIN film_category fc ON cat.category_id = fc.category_id INNER JOIN inventory i ON fc.film_id = i.film_id INNER JOIN rental r ON i.inventory_id = r.inventory_id GROUP BY cat.name ORDER BY RentalCount DESC LIMIT 3; SELECT s.store_id, COUNT(i.inventory_id) AS TotalFilms, SUM(CASE WHEN r.rental_id IS NULL THEN 1 ELSE 0 END) AS NeverRentedFilms FROM store s INNER JOIN inventory i ON s.store_id = i.store_id LEFT JOIN rental r ON i.inventory_id = r.inventory_id GROUP BY s.store_id; SELECT YEAR(p.payment_date) AS Year, MONTH(p.payment_date) AS Month, SUM(p.amount) AS TotalRevenue FROM payment p WHERE YEAR(p.payment_date) = 2023 GROUP BY YEAR(p.payment_date), MONTH(p.payment_date) ORDER BY Month; SELECT CONCAT(c.first_name, ' ', c.last_name) AS CustomerName, COUNT(r.rental_id) AS RentalCount FROM customer c INNER JOIN rental r ON c.customer_id = r.customer_id WHERE r.rental_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH) GROUP BY c.customer_id, c.first_name, c.last_name HAVING COUNT(r.rental_id) > 10 ORDER BY RentalCount DESC;