Quiz-1 (11-06-2025)

# Section 1: Managing Databases

Which of the following is NOT a system database in SQL Server?  
a) master  
b) model  
c) tempdb  
d) userdb

Which system database stores all login accounts and configuration settings?  
a) tempdb  
b) model  
c) master  
d) msdb

What is the purpose of the model database in SQL Server?  
a) Backup  
b) Log storage  
c) Template for new databases  
d) System configuration

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

Which SQL command is used to create a new database?  
a) MAKE DATABASE  
b) NEW DATABASE  
c) CREATE DATABASE  
d) INIT DATABASE

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

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

Which data type should be used to store a date of birth?  
a) VARCHAR  
b) DATE  
c) INT  
d) TEXT

What command is used to create a table?  
a) MAKE TABLE  
b) INSERT TABLE  
c) CREATE TABLE  
d) DEFINE TABLE

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

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

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

Which command adds data into a table?  
a) INSERT INTO  
b) ADD ROW  
c) CREATE DATA  
d) APPEND TO

Which clause is used to update data in a table?  
a) MODIFY  
b) UPDATE  
c) CHANGE  
d) SET TABLE

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

Which clause is used to filter rows in a SELECT statement?  
a) HAVING  
b) SELECT  
c) WHERE  
d) ORDER BY

Which keyword ensures no duplicate records are returned?  
a) UNIQUE  
b) NO\_REPEAT  
c) DISTINCT  
d) ONLY

What does the LIKE keyword do in SQL?  
a) Finds exact matches  
b) Finds pattern-based matches  
c) Sorts records  
d) Deletes matches

Which operator is used to combine multiple conditions in a WHERE clause?  
a) TO  
b) WITH  
c) AND / OR  
d) IF / ELSE

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

**12-06-2025**

**1. Insert and Update with Integrity**

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 VALUES

(1, 'Tony Stark', 'tony@avengers.com', 85),

(2, 'Steve Rogers', 'steve@avengers.com', 90),

(3, 'Bruce Banner', 'bruce@avengers.com', 78),

(4, 'Natasha Romanoff', 'natasha@avengers.com', 92),

(5, 'Clint Barton', 'clint@avengers.com', 70);

UPDATE students SET marks = 95 WHERE student\_id = 2;

---------------------------------------------------------------------------------------------------------------------------

**2. String Function Challenge**

CREATE TABLE customers (

customer\_id INT PRIMARY KEY,

full\_name VARCHAR(100) NOT NULL

);

INSERT INTO customers VALUES

(1, 'Tony Stark'),

(2, 'Steve Rogers'),

(3, 'Natasha Romanoff'),

(4, 'Bruce Banner'),

(5, 'Clint Barton');

SELECT

full\_name,

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**

CREATE TABLE sales (

sale\_id INT PRIMARY KEY,

sale\_date DATE

);

INSERT INTO sales VALUES

(1, '2025-06-10'),

(2, '2025-05-01'),

(3, '2025-01-15');

SELECT

sale\_date,

MONTHNAME(sale\_date) AS month\_name,

YEAR(sale\_date) AS sale\_year,

DATEDIFF(CURDATE(), sale\_date) AS days\_ago

FROM sales;

---------------------------------------------------------------------------------------------------------------------------

**4. Mathematical Functions on Salary**

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(100),

salary DECIMAL(10, 2)

);

INSERT INTO employees VALUES

(1, 'Tony Stark', 120000),

(2, 'Steve Rogers', 95000);

SELECT

name,

salary,

salary \* 1.10 AS salary\_with\_hike,

ROUND(salary, -2) AS rounded\_salary

FROM employees;

---------------------------------------------------------------------------------------------------------------------------

**5. System Function Check**

SELECT

NOW() AS current\_datetime,

DATABASE() AS current\_database,

USER() AS logged\_in\_user;

---------------------------------------------------------------------------------------------------------------------------

**6. Demo: Custom Result Set**

CREATE TABLE products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(100),

price DECIMAL(10,2)

);

INSERT INTO products VALUES

(1, 'Arc Reactor', NULL),

(2, 'Shield', 15000),

(3, 'Hammer', 25000);

SELECT

UPPER(product\_name) AS product\_name\_uppercase,

IFNULL(price, 'Not Available') AS price\_status

FROM products;

---------------------------------------------------------------------------------------------------------------------------

**7. Aggregate Functions Practice**

CREATE TABLE transactions (

transaction\_id INT PRIMARY KEY,

sale\_amount DECIMAL(10,2)

);

INSERT INTO transactions VALUES

(1, 1000),

(2, 1500),

(3, 500),

(4, 2000);

SELECT

SUM(sale\_amount) AS total\_sales,

AVG(sale\_amount) AS average\_sale,

MAX(sale\_amount) AS max\_sale,

MIN(sale\_amount) AS min\_sale

FROM transactions;

---------------------------------------------------------------------------------------------------------------------------

**8. Grouping with Aggregation**

CREATE TABLE sales (

sale\_id INT,

category VARCHAR(100),

amount DECIMAL(10,2)

);

INSERT INTO sales VALUES

(1, 'Weapons', 2000),

(2, 'Armor', 1500),

(3, 'Weapons', 3000),

(4, 'Gadgets', 1200);

SELECT

category,

COUNT(\*) AS transaction\_count,

SUM(amount) AS total\_sales

FROM sales

GROUP BY category;

---------------------------------------------------------------------------------------------------------------------------

**9. Inner Join for Orders and Customers**

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_amount DECIMAL(10,2)

);

INSERT INTO orders VALUES

(101, 1, 5000),

(102, 2, 3000),

(103, 4, 4500);

SELECT

c.full\_name AS customer\_name,

o.order\_amount

FROM orders o

INNER JOIN customers c ON o.customer\_id = c.customer\_id;

---------------------------------------------------------------------------------------------------------------------------

**10. Left Join for Products with or without Orders**

CREATE TABLE order\_details (

order\_id INT,

product\_id INT,

quantity INT

);

INSERT INTO order\_details VALUES

(101, 1, 2),

(102, 2, 1);

SELECT

p.product\_name,

od.quantity

FROM products p

LEFT JOIN order\_details od ON p.product\_id = od.product\_id;

---------------------------------------------------------------------------------------------------------------------------

**11. Right Join for Customer Contacts**

CREATE TABLE contacts (

contact\_id INT,

customer\_id INT,

phone VARCHAR(15)

);

INSERT INTO contacts VALUES

(1, 1, '1234567890'),

(2, 3, '9999999999');

SELECT

c.full\_name,

ct.phone

FROM contacts ct

RIGHT JOIN customers c ON ct.customer\_id = c.customer\_id;

---------------------------------------------------------------------------------------------------------------------------

**12. Full Outer Join for Suppliers and Products (use UNION)**

CREATE TABLE suppliers (

supplier\_id INT,

supplier\_name VARCHAR(100)

);

INSERT INTO suppliers VALUES

(1, 'Stark Industries'),

(2, 'Wakanda Tech');

SELECT s.supplier\_name, p.product\_name

FROM suppliers s

LEFT JOIN products p ON s.supplier\_id = p.product\_id

UNION

SELECT s.supplier\_name, p.product\_name

FROM suppliers s

RIGHT JOIN products p ON s.supplier\_id = p.product\_id;

---------------------------------------------------------------------------------------------------------------------------

**13. Cross Join for Offers**

CREATE TABLE offers (

offer\_id INT,

offer\_desc VARCHAR(100)

);

INSERT INTO offers VALUES

(1, '10% Off'),

(2, 'Free Shipping');

SELECT

p.product\_name,

o.offer\_desc

FROM products p

CROSS JOIN offers o;

---------------------------------------------------------------------------------------------------------------------------

**14. Join with Aggregation**

ALTER TABLE products ADD category VARCHAR(50);

UPDATE products SET category = 'Tech' WHERE product\_id = 1;

UPDATE products SET category = 'Defense' WHERE product\_id = 2;

UPDATE products SET category = 'Mythical' WHERE product\_id = 3;

CREATE TABLE orders\_extended (

order\_id INT,

product\_id INT,

quantity INT,

price DECIMAL(10,2)

);

INSERT INTO orders\_extended VALUES

(1, 1, 2, 50000),

(2, 2, 1, 15000),

(3, 3, 3, 75000);

SELECT

p.category,

SUM(o.quantity) AS total\_quantity\_sold,

AVG(o.price) AS avg\_price

FROM orders\_extended o

JOIN products p ON o.product\_id = p.product\_id

GROUP BY p.category;

---------------------------------------------------------------------------------------------------------------------------

**15. Join with Grouping and Filter**

CREATE TABLE marks (

student\_id INT,

subject VARCHAR(50),

mark INT

);

INSERT INTO marks VALUES

(1, 'Math', 80),

(1, 'Science', 85),

(2, 'Math', 95),

(2, 'Science', 90),

(3, 'Math', 70),

(3, 'Science', 75);

SELECT

s.name AS student\_name,

AVG(m.mark) AS average\_marks

FROM students s

JOIN marks m ON s.student\_id = m.student\_id

GROUP BY s.student\_id, s.name

HAVING AVG(m.mark) > 75;

---------------------------------------------------------------------------------------------------------------------------

13-06-2025

1) Querying Data by Using Subqueries

2) Querying Data by Using Subqueries Using the EXISTS,

3) Querying Data by Using Subqueries using ANY,

4) Querying Data by Using Subqueries using ALL Keywords

5) Querying Data by Using Subqueries using Using Nested Subqueries

6) Querying Data by Using Subqueries Using Correlated Subqueries

7) Querying Data by Using Subqueries Using UNION,

8) Querying Data by Using Subqueries using INTERSECT,

9) Querying Data by Using Subqueries using EXCEPT,

10)Querying Data by Using Subqueries using MERGE.

**Create Avengers table**

CREATE TABLE Avengers (

AvengerID INT PRIMARY KEY,

Name VARCHAR(50),

Role VARCHAR(30),

PowerLevel INT

);

CREATE TABLE Missions (

MissionID INT PRIMARY KEY,

MissionName VARCHAR(100),

DifficultyLevel INT,

AssignedTo INT,

FOREIGN KEY (AssignedTo) REFERENCES Avengers(AvengerID)

);

INSERT INTO Avengers VALUES

(1, 'Iron Man', 'Tech Genius', 95),

(2, 'Captain America', 'Leader', 90),

(3, 'Thor', 'God of Thunder', 99),

(4, 'Hulk', 'Strongest Avenger', 97),

(5, 'Black Widow', 'Spy', 88);

INSERT INTO Missions VALUES

(101, 'Battle of New York', 95, 1),

(102, 'Sokovia Incident', 97, 3),

(103, 'Hydra Base Raid', 85, 2),

(104, 'Space Threat', 99, 3),

(105, 'Infiltrate Hydra', 88, 5);

**1. Basic Subquery**

SELECT Name FROM Avengers

WHERE AvengerID IN (

SELECT AssignedTo FROM Missions WHERE DifficultyLevel > 90

);

**2. EXISTS**

SELECT Name FROM Avengers A

WHERE Name = 'Thor'

AND EXISTS (

SELECT 1 FROM Missions M WHERE M.AssignedTo = A.AvengerID

);

**3. ANY**

SELECT Name FROM Avengers

WHERE PowerLevel > ANY (

SELECT DifficultyLevel FROM Missions

);

**4. ALL**

SELECT Name FROM Avengers

WHERE PowerLevel > ALL (

SELECT DifficultyLevel FROM Missions

);

**5. Nested Subqueries**

SELECT MissionName FROM Missions

WHERE AssignedTo IN (

SELECT AvengerID FROM Avengers

WHERE PowerLevel = (

SELECT MAX(PowerLevel) FROM Avengers

)

);

-- 6. Correlated Subquery

SELECT Name FROM Avengers A

WHERE EXISTS (

SELECT 1 FROM Missions M

WHERE M.AssignedTo = A.AvengerID AND M.DifficultyLevel > A.PowerLevel

);

**7. UNION**

SELECT Name FROM Avengers WHERE PowerLevel > 95

UNION

SELECT A.Name FROM Avengers A

JOIN Missions M ON A.AvengerID = M.AssignedTo

WHERE M.DifficultyLevel > 95;

**8. INTERSECT (Note: Not supported in MySQL, use INNER JOIN and WHERE IN instead)**

SELECT Name FROM Avengers WHERE PowerLevel > 90

AND Name IN (

SELECT A.Name FROM Avengers A

JOIN Missions M ON A.AvengerID = M.AssignedTo

WHERE M.DifficultyLevel > 90

);

**9. EXCEPT (MySQL does not support EXCEPT, use LEFT JOIN and WHERE IS NULL)**

SELECT Name FROM Avengers WHERE PowerLevel > 90

AND Name NOT IN (

SELECT A.Name FROM Avengers A

JOIN Missions M ON A.AvengerID = M.AssignedTo

WHERE M.DifficultyLevel > 90

);

**10. MERGE equivalent (for MySQL: use INSERT ... ON DUPLICATE KEY UPDATE)**

CREATE TABLE AvengerStatus (

AvengerID INT PRIMARY KEY,

PowerLevel INT

);

SQL Practical Question – Answer

# Section A: Advanced Concepts & Schema Design

## Q1. NoSQL vs SQL and Types of NoSQL

NoSQL is preferred over SQL when:  
- The data is unstructured or semi-structured (e.g., JSON, XML).  
- High scalability and availability are required.  
- Rapid development and flexibility in schema design are important.  
  
Types of NoSQL Databases and Real-time Applications:  
1. Document Store (e.g., MongoDB): Used in content management systems like blogging platforms.  
2. Key-Value Store (e.g., Redis): Used in caching user sessions in web applications.  
3. Column-Family Store (e.g., Cassandra): Used in logging and analytics platforms like monitoring tools.  
4. Graph Database (e.g., Neo4j): Used in social networks for relationships and recommendations.

## Q2. Normalization to BCNF

Given Unnormalized Table:  
Customer (CustomerID, Name, Orders (OrderID, ProductID, Quantity, ProductName))  
  
1NF:  
Customer (CustomerID, Name)  
Order (OrderID, CustomerID, ProductID, Quantity, ProductName)  
  
2NF:  
Customer (CustomerID, Name)  
Order (OrderID, CustomerID)  
OrderDetails (OrderID, ProductID, Quantity, ProductName)  
  
3NF/BCNF:  
Customer (CustomerID, Name)  
Order (OrderID, CustomerID)  
Product (ProductID, ProductName)  
OrderDetails (OrderID, ProductID, Quantity)

# Section B: Complex DDL and DML

## Q3. Schema and Constraints

a) CREATE DATABASE RetailDB;  
  
CREATE TABLE Customers (  
 CustomerID INT PRIMARY KEY,  
 Name 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),  
 FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),  
 FOREIGN KEY (ProductID) REFERENCES Products(ProductID)  
);  
  
b) Implemented CHECK constraint in Orders table.  
  
c) ALTER TABLE Products ADD Discount DECIMAL(5,2);  
UPDATE Products SET Discount = 5.00 WHERE ProductID = 1;

## Q4. Insert/Update/Delete

a) INSERT INTO Orders VALUES (1, 1, 101, 3), (2, 1, 102, 2), (3, 1, 103, 1);  
  
b) UPDATE Products SET Price = Price \* 1.10  
WHERE ProductID IN (SELECT ProductID FROM Orders GROUP BY ProductID HAVING SUM(Quantity) > 5);  
  
c) DELETE FROM Orders WHERE ProductID NOT IN (SELECT ProductID FROM Orders);

## Q5. Retrieval Queries

a) SELECT CustomerID FROM Orders GROUP BY CustomerID HAVING COUNT(DISTINCT ProductID) > 3;  
  
b) SELECT ProductName FROM Products WHERE ProductID NOT IN (SELECT DISTINCT ProductID FROM Orders);  
  
c) SELECT CustomerID, COUNT(\*) FROM Orders  
WHERE OrderDate >= CURDATE() - INTERVAL 30 DAY GROUP BY CustomerID;

# Section C: Advanced Functions and Aggregations

## Q6. Functions Usage

a) SELECT LOWER(SUBSTRING\_INDEX(email, '@', 1)) AS username FROM Customers;  
  
b) SELECT DATEDIFF(CURDATE(), OrderDate) AS DaysSinceOrder FROM Orders;  
  
c) SELECT USER(), HOST();  
  
d) SELECT CONCAT('Hello ', UPPER(Name), '!') AS Greeting FROM Customers;

## Q7. Aggregation & Grouping

a) SELECT Category, SUM(Price \* Quantity) AS TotalRevenue FROM Products P  
JOIN Orders O ON P.ProductID = O.ProductID GROUP BY Category;  
  
b) SELECT Category, SUM(Price \* Quantity) AS Revenue FROM Products P  
JOIN Orders O ON P.ProductID = O.ProductID GROUP BY Category WITH ROLLUP;  
  
c) 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

## Q8. Various Joins

a) SELECT C1.Name AS Referrer, C2.Name AS Referred FROM Customers C1  
JOIN Customers C2 ON C1.CustomerID = C2.ReferredBy;  
  
b) SELECT O.OrderID, P.ProductName FROM Orders O JOIN Products P ON O.ProductID = P.ProductID;  
  
c) SELECT Name, SUM(Price \* Quantity) OVER (PARTITION BY CustomerID) AS TotalSpend  
FROM Customers C JOIN Orders O ON C.CustomerID = O.CustomerID  
JOIN Products P ON O.ProductID = P.ProductID ORDER BY TotalSpend DESC LIMIT 3;  
  
d) SELECT C.Name FROM Customers C LEFT JOIN Orders O ON C.CustomerID = O.CustomerID WHERE O.OrderID IS NULL;  
  
e) SELECT P1.ProductName AS A, P2.ProductName AS B FROM Products P1 CROSS JOIN Products P2 WHERE P1.ProductID < P2.ProductID;

## Q9. Subqueries

a) SELECT Name FROM Customers C WHERE EXISTS (  
 SELECT 1 FROM Orders O WHERE O.CustomerID = C.CustomerID   
 GROUP BY O.CustomerID HAVING SUM(O.Amount) >   
 (SELECT AVG(Amount) FROM Orders WHERE CustomerID = C.CustomerID)  
);  
  
b) SELECT Name FROM Customers C WHERE EXISTS (  
 SELECT 1 FROM Orders O WHERE O.CustomerID = C.CustomerID GROUP BY ProductID HAVING COUNT(DISTINCT ProductID) >= 2  
);  
  
c) SELECT Name FROM Customers WHERE CustomerID = ALL (  
 SELECT CustomerID FROM Orders GROUP BY CustomerID HAVING COUNT(\*) >= ALL (  
 SELECT COUNT(\*) FROM Orders GROUP BY CustomerID  
 )  
);  
  
d) SELECT ProductName FROM Products WHERE Price > ANY (  
 SELECT Price FROM Products WHERE Category = 'Electronics'  
);  
  
e) SELECT ProductID, SUM(Quantity) AS Sold FROM Orders GROUP BY ProductID ORDER BY Sold DESC LIMIT 3;

## Q10. Set Operations

a) SELECT Name FROM SegmentA INNER JOIN SegmentB USING(CustomerID);  
  
b) SELECT ProductName FROM Products WHERE ProductID NOT IN (SELECT DISTINCT ProductID FROM Orders);  
  
c) INSERT INTO Customers (CustomerID, Name)  
VALUES (101, 'John Doe') ON DUPLICATE KEY UPDATE Name = 'John Doe';  
  
d) SELECT Name FROM Region1 UNION SELECT Name FROM Region2;  
  
e) WITH RankedCustomers AS (  
 SELECT CustomerID, SUM(Amount) AS TotalSpend,  
 RANK() OVER (ORDER BY SUM(Amount) DESC) AS Rank FROM Orders GROUP BY CustomerID  
)  
SELECT \* FROM RankedCustomers WHERE Rank <= 5;

SQL Practical Question - Answer

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

## Q1. Differentiate between SQL and NoSQL

SQL (Structured Query Language):  
- SQL databases are relational, table-based.  
- Examples: MySQL, PostgreSQL.  
  
Advantages:  
1. Structured and predictable schema – ideal for complex queries.  
2. ACID compliance ensures data integrity.  
  
Disadvantages:  
1. Scaling is difficult – usually vertical.  
2. Poor performance with unstructured data.

**Real-World Example:**

**Online Banking System**

* **Use Case**: Banks use **MySQL** to store customer details, transactions, and account balances.
* **Why SQL?**: Data is highly structured and relational. It requires strict integrity and consistency (ACID compliance).

**Example Tables**:

* Customers(CustomerID, Name, Address)
* Accounts(AccountID, CustomerID, Balance)
* Transactions(TransactionID, AccountID, Date, Amount)

**Sample Query**:

SELECT Name, Balance

FROM Customers

JOIN Accounts ON Customers.CustomerID = Accounts.CustomerID

WHERE Balance > 10000;

NoSQL (Not only SQL):  
- Non-relational, document/key-value/graph-based.  
- Examples: MongoDB, Cassandra.  
  
Advantages:  
1. Easily scalable – especially horizontally.  
2. Flexible schema – ideal for big data and real-time applications.  
  
Disadvantages:  
1. Lack of standardization across databases.  
2. No support for complex joins or ACID properties.

**Real-World Example:**

**E-commerce Website (e.g., Amazon)**

* **Use Case**: Product catalogs, customer preferences, and real-time activity logs are stored in MongoDB.
* **Why NoSQL?**: Schema-less, easily handles diverse product structures, and scales horizontally for high traffic.
* **Query Example (MongoDB)**:

db.products.find({ "Price": { "$lt": 30000 } })

## Q2. Normalize the data to 1NF, 2NF, 3NF

Unnormalized: Student(StudentID, Name, CourseID, CourseName, InstructorName, InstructorPhone)  
  
1NF: Remove repeating groups:  
Student(StudentID, Name, CourseID, CourseName, InstructorName, InstructorPhone)  
  
2NF: Remove partial dependencies:  
Student(StudentID, Name)  
Course(CourseID, CourseName, InstructorName, InstructorPhone)  
  
3NF: Remove transitive dependencies:  
Student(StudentID, Name)  
Course(CourseID, CourseName, InstructorID)  
Instructor(InstructorID, InstructorName, InstructorPhone)

## Q3. SQL Commands

a) CREATE DATABASE StudentDB;  
b) CREATE TABLE Students (StudentID INT, Name VARCHAR(50), DOB DATE, Email VARCHAR(100));  
c) ALTER TABLE Students RENAME TO Student\_Info;  
d) ALTER TABLE Student\_Info ADD PhoneNumber VARCHAR(15);  
e) DROP TABLE Student\_Info;

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

## Q4. Student\_Info DML Operations

a) INSERT INTO Student\_Info (StudentID, Name, DOB, Email, PhoneNumber) VALUES   
(1, 'Arjun', '2001-06-21', 'arjun@gmail.com', '9876543210'),  
(2, 'Bala', '1999-02-14', 'bala@yahoo.com', '9123456780'),  
(3, 'Charan', '2002-10-30', 'charan@gmail.com', '9988776655');  
  
b) UPDATE Student\_Info SET PhoneNumber = '9000000000' WHERE StudentID = 2;  
  
c) DELETE FROM Student\_Info WHERE Email LIKE '%@gmail.com';  
  
d) SELECT Name, Email FROM Student\_Info WHERE YEAR(DOB) > 2000;  
  
e) SELECT DISTINCT SUBSTRING\_INDEX(Email, '@', -1) AS Domain FROM Student\_Info;

## Q5. Filtering Techniques

a) SELECT \* FROM Student\_Info WHERE Name LIKE 'A%';  
  
b) SELECT \* FROM Student\_Info WHERE PhoneNumber BETWEEN '9000000000' AND '9999999999';  
  
c) SELECT \* FROM Student\_Info WHERE City IN ('Chennai', 'Delhi', 'Mumbai');  
  
d) SELECT \* FROM Student\_Info WHERE (YEAR(CURDATE()) - YEAR(DOB)) > 22 AND Email LIKE '%@yahoo.com';  
  
e) SELECT SI.Name AS StudentName, SI.DOB AS DateOfBirth FROM Student\_Info AS SI;

## Q6. Marks Table Creation and Queries

CREATE TABLE Marks (StudentID INT, Subject VARCHAR(50), Marks INT);  
  
INSERT INTO Marks VALUES   
(1, 'Maths', 85),  
(2, 'Science', 78),  
(3, 'English', 65);  
  
a) SELECT StudentID, Subject FROM Marks WHERE Marks > 70;  
  
b) SELECT Subject, AVG(Marks) AS AvgMarks FROM Marks GROUP BY Subject;  
  
c) SELECT Subject FROM Marks GROUP BY Subject HAVING AVG(Marks) BETWEEN 60 AND 90;

# Section C: Functions & Grouping (10 Marks)

## Q7. SQL Functions

a) SELECT DATE\_FORMAT(CURDATE(), '%Y-%m-%d') AS TodayDate;  
  
b) SELECT MONTH(DOB) AS BirthMonth, YEAR(DOB) AS BirthYear FROM Student\_Info;  
  
c) SELECT UPPER(Name) AS UpperName FROM Student\_Info;  
  
d) SELECT ROUND(Marks, 2) AS RoundedMarks FROM Marks;  
  
e) SELECT USER() AS CurrentUser, DATABASE() AS CurrentDatabase;

## Q8. Aggregate Functions and Grouping

a) SELECT StudentID, SUM(Marks) AS TotalMarks FROM Marks GROUP BY StudentID;  
  
b) SELECT Subject, MAX(Marks) AS HighestMark FROM Marks GROUP BY Subject;  
  
c) SELECT Subject, AVG(Marks) AS AvgMarks FROM Marks GROUP BY Subject HAVING AVG(Marks) > 75;

# Section D: Joins and Subqueries (25 Marks)

## Q9. Joins

Assuming tables: Students(StudentID, Name), Courses(CourseID, CourseName), Enrollment(StudentID, CourseID)  
  
a) SELECT S.StudentID, S.Name, C.CourseName  
 FROM Students S  
 INNER JOIN Enrollment E ON S.StudentID = E.StudentID  
 INNER JOIN Courses C ON E.CourseID = C.CourseID;  
  
b) SELECT S.StudentID, S.Name, C.CourseName  
 FROM Students S  
 LEFT JOIN Enrollment E ON S.StudentID = E.StudentID  
 LEFT JOIN Courses C ON E.CourseID = C.CourseID;  
  
c) SELECT S.StudentID, S.Name, C.CourseName  
 FROM Courses C  
 RIGHT JOIN Enrollment E ON C.CourseID = E.CourseID  
 RIGHT JOIN Students S ON E.StudentID = S.StudentID;  
  
d) -- FULL OUTER JOIN using UNION  
 SELECT S.StudentID, S.Name, C.CourseName  
 FROM Students S  
 LEFT JOIN Enrollment E ON S.StudentID = E.StudentID  
 LEFT JOIN Courses C ON E.CourseID = C.CourseID  
 UNION  
 SELECT S.StudentID, S.Name, C.CourseName  
 FROM Courses C  
 LEFT JOIN Enrollment E ON C.CourseID = E.CourseID  
 LEFT JOIN Students S ON E.StudentID = S.StudentID;  
  
e) SELECT S.StudentID, C.CourseID  
 FROM Students S  
 CROSS JOIN Courses C;

## Q10. Subqueries

a) SELECT \* FROM Marks WHERE Subject = 'Maths' AND Marks > (SELECT AVG(Marks) FROM Marks WHERE Subject = 'Maths');  
  
b) SELECT \* FROM Students WHERE StudentID NOT IN (SELECT DISTINCT StudentID FROM Marks);  
  
c) SELECT \* FROM Students S WHERE EXISTS (SELECT 1 FROM Marks M WHERE M.StudentID = S.StudentID);  
  
d) SELECT \* FROM Marks WHERE Subject = 'Science' AND Marks > ALL (SELECT Marks FROM Marks WHERE Subject = 'Science');  
  
e) SELECT \* FROM Marks WHERE Subject = 'English' AND Marks > ANY (SELECT Marks FROM Marks WHERE Subject = 'English');

## Q11. Set Operators and Correlated Subquery

a) SELECT Name FROM Students  
 UNION  
 SELECT Name FROM Alumni;  
  
b) SELECT Name FROM Students  
 INTERSECT  
 SELECT Name FROM Marks;  
  
c) SELECT Name FROM Students  
 EXCEPT  
 SELECT Name FROM Marks;  
  
d) -- Simulating MERGE with UPDATE and INSERT  
 UPDATE Marks SET Marks = 90 WHERE StudentID = 2 AND Subject = 'Science';  
 INSERT INTO Marks (StudentID, Subject, Marks)  
 SELECT 4, 'Science', 88 FROM DUAL WHERE NOT EXISTS (  
 SELECT \* FROM Marks WHERE StudentID = 4 AND Subject = 'Science');  
  
e) SELECT M1.StudentID, M1.Subject, M1.Marks  
 FROM Marks M1  
 WHERE Marks > (SELECT AVG(Marks) FROM Marks M2 WHERE M2.Subject = M1.Subject);