**QUIZ – 11/06/2025**

Section 1: Managing Databases

1. Which of the following is NOT a system database in SQL Server?  
   d) userdb
2. Which system database stores all login accounts and configuration settings?  
   c) master
3. What is the purpose of the model database in SQL Server?  
   c) Template for new databases
4. What are the two main types of database files in SQL Server?  
   b) LDF and MDF
5. Which SQL command is used to create a new database?  
   c) CREATE DATABASE
6. What happens when you execute DROP DATABASE SalesDB?  
   c) SalesDB is deleted permanently
7. Which command renames a database in SQL Server?  
   b) ALTER DATABASE old\_name MODIFY NAME = new\_name

Section 2: Managing Tables

1. Which data type should be used to store a date of birth?  
   b) DATE
2. What command is used to create a table?  
   c) CREATE TABLE
3. How do you add a new column to an existing table?  
   a) ALTER TABLE table\_name ADD column\_name datatype
4. Which command is used to rename a table?  
   c) EXEC sp\_rename 'old\_name', 'new\_name'
5. What is the command to delete a table permanently?  
   c) DROP TABLE table\_name

Section 3: DML - Manipulating Data

1. Which command adds data into a table?  
    a) INSERT INTO
2. Which clause is used to update data in a table?  
   b) UPDATE
3. What does the DELETE statement do?  
   c) Removes specific rows
4. Which clause is used to filter rows in a SELECT statement?  
    c) WHERE
5. Which keyword ensures no duplicate records are returned?  
    c) DISTINCT
6. What does the LIKE keyword do in SQL?  
    b) Finds pattern-based matches
7. Which operator is used to combine multiple conditions in a WHERE clause?  
    c) AND / OR
8. What does the BETWEEN operator do?  
    c) Filters values within a range

**ASSIGNMENT – 11/06/2025**

Section A: Managing Databases

1. List all system databases in SQL Server

SELECT name FROM sys.databases WHERE database\_id < 5;

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;

Section B: Managing Tables

1. Create a table Employees

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;

Section C: DML Operations

1. Insert three rows into Employees

INSERT INTO Employees (EmpID, Name, Department, JoiningDate, IsActive, Salary) 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');

**ASSIGNMENT DAY 4 – 12/06/2004**

1. Insert and Update with Integrity

Create students table

CREATE TABLE students (

student\_id INT PRIMARY KEY AUTO\_INCREMENT,

student\_name VARCHAR(100) NOT NULL,

marks INT NOT NULL,

email VARCHAR(100) UNIQUE

);

Insert 5 records

INSERT INTO students (student\_name, marks, email) VALUES

('Arun Kumar', 85, 'arun@example.com'),

('Divya Ramesh', 90, 'divya@example.com'),

('Rajiv Menon', 78, 'rajiv@example.com'),

('Sneha Iyer', 92, 'sneha@example.com'),

('Vinay Rao', 80, 'vinay@example.com');

Update marks for a student

UPDATE students

SET marks = 95

WHERE student\_name = 'Rajiv Menon';

2. String Function Challenge

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

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

SELECT

employee\_name,

salary,

salary \* 1.10 AS salary\_after\_hike,

ROUND(salary, -2) AS rounded\_salary

FROM employees;

5. System Function Check

SELECT

NOW() AS current\_datetime,

DATABASE() AS database\_name,

USER() AS logged\_in\_user;

6. Demo: Custom Result Set

SELECT

UPPER(product\_name) AS product\_name,

IFNULL(price, 'Not Available') AS display\_price

FROM products;

7. Aggregate Functions Practice

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

SELECT

product\_category,

SUM(sale\_amount) AS total\_sales,

COUNT(\*) AS transaction\_count

FROM sales

GROUP BY product\_category;

9. Inner Join for Orders and Customers

SELECT

c.customer\_name,

o.order\_amount

FROM orders o

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

10. Left Join for Products with or without Orders

SELECT

p.product\_name,

o.order\_id,

o.order\_amount

FROM products p

LEFT JOIN orders o ON p.product\_id = o.product\_id;

11. Right Join for Customer Contacts

SELECT

c.customer\_name,

ct.phone\_number

FROM contacts ct

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

12. Full Outer Join for Suppliers and Products

SELECT

s.supplier\_name,

p.product\_name

FROM suppliers s

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

UNION

SELECT

s.supplier\_name,

p.product\_name

FROM suppliers s

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

13. Cross Join for Offers

SELECT

p.product\_name,

o.offer\_name

FROM products p

CROSS JOIN offers o;

14. Join with Aggregation

SELECT

pr.product\_category,

SUM(o.quantity) AS total\_quantity,

AVG(pr.price) AS average\_price

FROM orders o

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

GROUP BY pr.product\_category;

15. Join with Grouping and Filter

SELECT

s.student\_name,

AVG(m.marks) AS average\_marks

FROM students s

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

GROUP BY s.student\_name

HAVING AVG(m.marks) > 75;

**13/06/2025**

**SQL Practical Question Paper - 1**

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

**Q1. (3 marks)**

-- Differentiate between SQL and NoSQL with pros/cons and examples

-- SQL (Structured Query Language) databases:

-- Pros: 1. Strong ACID compliance (e.g., MySQL, PostgreSQL)

-- 2. Relational data integrity

-- Cons: 1. Poor scalability for large distributed systems

-- 2. Rigid schema

-- Example: MySQL used in banking systems

-- NoSQL (Not Only SQL) databases:

-- Pros: 1. Highly scalable (e.g., MongoDB, Cassandra)

-- 2. Flexible schema

-- Cons: 1. Weaker consistency model

-- 2. Lack of standard query language

-- Example: MongoDB used in real-time analytics

**Q2. (2 marks)**

-- Unnormalized: Student(StudentID, Name, CourseID, CourseName, InstructorName, InstructorPhone)

-- 1NF: Split repeating groups:

-- Student(StudentID, Name)

-- Enrollment(StudentID, CourseID, CourseName, InstructorName, InstructorPhone)

-- 2NF: Remove partial dependencies:

-- Student(StudentID, Name)

-- Course(CourseID, CourseName, InstructorName, InstructorPhone)

-- Enrollment(StudentID, CourseID)

-- 3NF: Remove transitive dependencies:

-- Instructor(InstructorName, InstructorPhone)

-- Course(CourseID, CourseName, InstructorName)

-- Student(StudentID, Name)

-- Enrollment(StudentID, CourseID)

**Q3. (5 marks)**

CREATE DATABASE StudentDB;

USE StudentDB;

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

Name VARCHAR(50),

DOB DATE,

Email VARCHAR(100)

);

EXEC sp\_rename 'Students', 'Student\_Info';

ALTER TABLE Student\_Info ADD PhoneNumber VARCHAR(15);

DROP TABLE Student\_Info;

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

**Q4. (5 marks)**

INSERT INTO Student\_Info VALUES (1, 'Alice', '2001-03-10', 'alice@example.com', '9876543210');

INSERT INTO Student\_Info VALUES (2, 'Bob', '2002-07-15', 'bob@gmail.com', '9000000001');

INSERT INTO Student\_Info VALUES (3, 'Charlie', '1999-05-21', 'charlie@yahoo.com', '9000000002');

UPDATE Student\_Info SET PhoneNumber = '9999999999' WHERE StudentID = 1;

DELETE FROM Student\_Info WHERE Email LIKE '%@gmail.com';

SELECT Name, Email FROM Student\_Info WHERE YEAR(DOB) > 2000;

SELECT DISTINCT SUBSTRING(Email, CHARINDEX('@', Email) + 1, LEN(Email)) AS Domain FROM Student\_Info;

**Q5. (5 marks)**

SELECT \* FROM Student\_Info WHERE Name LIKE 'A%';

SELECT \* FROM Student\_Info WHERE PhoneNumber BETWEEN '9000000000' AND '9999999999';

SELECT \* FROM Student\_Info WHERE City IN ('Chennai', 'Mumbai', 'Delhi');

SELECT \* FROM Student\_Info WHERE Age > 20 AND Email LIKE '%@gmail.com';

SELECT SI.Name AS StudentName, SI.DOB AS BirthDate FROM Student\_Info SI;

**Q6. (5 marks)**

CREATE TABLE Marks (

StudentID INT,

Subject VARCHAR(50),

Marks FLOAT

);

INSERT INTO Marks VALUES (1, 'Maths', 85);

INSERT INTO Marks VALUES (2, 'Science', 78);

INSERT INTO Marks VALUES (1, 'English', 92);

SELECT StudentID, Subject FROM Marks WHERE Marks > 70;

SELECT Subject, AVG(Marks) AS AvgMarks FROM Marks GROUP BY Subject;

SELECT Subject FROM Marks GROUP BY Subject HAVING AVG(Marks) BETWEEN 60 AND 90;

**Section C: Functions & Grouping (10 Marks)**

**Q7. (5 marks)**

SELECT FORMAT(GETDATE(), 'yyyy-MM-dd') AS CurrentDate;

SELECT MONTH(DOB) AS BirthMonth, YEAR(DOB) AS BirthYear FROM Student\_Info;

SELECT UPPER(Name) AS UpperCaseName FROM Student\_Info;

SELECT ROUND(Marks, 2) FROM Marks;

SELECT SYSTEM\_USER AS CurrentUser, DB\_NAME() AS CurrentDB;

**Q8. (5 marks)**

SELECT StudentID, SUM(Marks) AS TotalMarks FROM Marks GROUP BY StudentID;

SELECT Subject, MAX(Marks) AS HighestMark FROM Marks GROUP BY Subject;

SELECT Subject, AVG(Marks) AS AvgMarks FROM Marks GROUP BY Subject HAVING AVG(Marks) > 75;

**Section D: Joins and Subqueries (25 Marks)**

**Q9. (5 marks)**

SELECT SI.Name, C.CourseName FROM Student\_Info SI

INNER JOIN Enrollment E ON SI.StudentID = E.StudentID

INNER JOIN Course C ON E.CourseID = C.CourseID;

SELECT SI.Name, C.CourseName FROM Student\_Info SI

LEFT JOIN Enrollment E ON SI.StudentID = E.StudentID

LEFT JOIN Course C ON E.CourseID = C.CourseID;

SELECT SI.Name, C.CourseName FROM Course C

RIGHT JOIN Enrollment E ON C.CourseID = E.CourseID

RIGHT JOIN Student\_Info SI ON E.StudentID = SI.StudentID;

SELECT SI.Name, C.CourseName FROM Student\_Info SI

LEFT JOIN Enrollment E ON SI.StudentID = E.StudentID

LEFT JOIN Course C ON E.CourseID = C.CourseID

UNION

SELECT SI.Name, C.CourseName FROM Course C

LEFT JOIN Enrollment E ON C.CourseID = E.CourseID

LEFT JOIN Student\_Info SI ON E.StudentID = SI.StudentID;

SELECT \* FROM Student\_Info CROSS JOIN Course;

**Q10. (5 marks)**

SELECT StudentID FROM Marks WHERE Subject = 'Maths' AND Marks > (SELECT AVG(Marks) FROM Marks WHERE Subject = 'Maths');

SELECT \* FROM Student\_Info WHERE StudentID NOT IN (SELECT DISTINCT StudentID FROM Marks);

SELECT \* FROM Student\_Info WHERE EXISTS (SELECT 1 FROM Marks WHERE Student\_Info.StudentID = Marks.StudentID);

SELECT \* FROM Marks WHERE Marks > ALL (SELECT Marks FROM Marks WHERE Subject = 'Science');

SELECT \* FROM Marks WHERE Marks > ANY (SELECT Marks FROM Marks WHERE Subject = 'English');

**Q11. (5 marks)**

SELECT Name FROM Student\_Info

UNION

SELECT Name FROM Alumni;

SELECT Name FROM Student\_Info

INTERSECT

SELECT Name FROM Marks;

SELECT Name FROM Student\_Info

EXCEPT

SELECT Name FROM Marks;

-- Simulate MERGE

IF EXISTS (SELECT \* FROM Student\_Info WHERE StudentID = 1)

UPDATE Student\_Info SET PhoneNumber = '1234567890' WHERE StudentID = 1;

ELSE

INSERT INTO Student\_Info VALUES (1, 'New Student', '2002-01-01', 'new@example.com', '1234567890');

Correlated subquery

SELECT StudentID, Subject, Marks FROM Marks M1

WHERE Marks > (SELECT AVG(Marks) FROM Marks M2 WHERE M1.Subject = M2.Subject);

**13/06/2025**

**SQL Practical Question Paper -2**

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

**Q1. (4 marks)**

-- NoSQL is preferred over SQL when:

-- 1. The data is unstructured or semi-structured (e.g., JSON, XML).

-- 2. There is a need for high scalability and availability.

-- 3. Schema flexibility is required.

-- 4. Real-time big data processing is involved.

-- Types of NoSQL Databases and Real-Time Applications:

-- a) Document Store (e.g., MongoDB) – Used in CMS systems like WordPress

-- b) Key-Value Store (e.g., Redis) – Used for caching in e-commerce apps

-- c) Column Store (e.g., Cassandra) – Used in time-series data like IoT sensors

-- d) Graph Store (e.g., Neo4j) – Used in social networks and recommendation engines

**Q2. (6 marks)**

-- Unnormalized: Customer(CustomerID, Name, Orders(OrderID, ProductID, Quantity, ProductName))

-- 1NF: Flatten nested structure

-- Customer(CustomerID, Name)

-- Order(OrderID, CustomerID, ProductID, Quantity, ProductName)

-- 2NF: Remove partial dependencies

-- Product(ProductID, ProductName)

-- Order(OrderID, CustomerID, ProductID, Quantity)

-- 3NF/BCNF: Remove transitive dependencies

-- Final tables:

-- Customer(CustomerID, Name)

-- Product(ProductID, ProductName)

-- Orders(OrderID, CustomerID, ProductID, Quantity)

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

**Q3. (5 marks)**

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),

Category VARCHAR(50)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

ProductID INT,

Quantity INT CHECK (Quantity > 0),

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

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

UPDATE Products SET Discount = 10.00 WHERE ProductID = 1;

**Q4. (5 marks)**

INSERT INTO Orders VALUES (1, 1, 1, 3, '2025-06-10');

INSERT INTO Orders VALUES (2, 1, 2, 6, '2025-06-11');

INSERT INTO Orders VALUES (3, 1, 3, 2, '2025-06-12');

INSERT INTO Orders VALUES (4, 2, 1, 1, '2025-06-13');

INSERT INTO Orders VALUES (5, 2, 2, 4, '2025-06-14');

INSERT INTO Orders VALUES (6, 2, 3, 7, '2025-06-15');

UPDATE Products SET Price = Price \* 1.10 WHERE ProductID IN (

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

);

DELETE FROM Products WHERE ProductID NOT IN (

SELECT DISTINCT ProductID FROM Orders

);

**Q5. (5 marks)**

SELECT CustomerID FROM Orders GROUP BY CustomerID HAVING COUNT(DISTINCT ProductID) > 3;

SELECT \* FROM Products WHERE ProductID NOT IN (SELECT ProductID FROM Orders);

SELECT CustomerID, COUNT(\*) AS OrderCount FROM Orders

WHERE OrderDate >= DATEADD(DAY, -30, GETDATE())

GROUP BY CustomerID;

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

**Q6. (5 marks)**

SELECT LOWER(Email), SUBSTRING(Email, 1, CHARINDEX('@', Email)-1) AS Username FROM Customers;

SELECT DATEDIFF(DAY, OrderDate, GETDATE()) AS DaysPassed FROM Orders;

SELECT SYSTEM\_USER, HOST\_NAME();

SELECT CONCAT('Hello ', UPPER(Name), ', welcome back!') AS Greeting FROM Customers;

**Q7. (5 marks)**

SELECT Category, SUM(Price \* Quantity) AS Revenue

FROM Orders O

JOIN Products P ON O.ProductID = P.ProductID

GROUP BY Category;

SELECT Category, SUM(Price \* Quantity) AS Revenue

FROM Orders O

JOIN Products P ON O.ProductID = P.ProductID

GROUP BY ROLLUP(Category);

SELECT Category, SUM(Price \* Quantity) AS Revenue

FROM Orders O

JOIN Products P ON O.ProductID = P.ProductID

GROUP BY Category

HAVING SUM(Price \* Quantity) > 100000;

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

**Q8. (5 marks)**

SELECT A.Name AS Referrer, B.Name AS Referred

FROM Customers A JOIN Customers B ON A.CustomerID = B.ReferrerID;

SELECT \* FROM Orders O JOIN Products P ON O.ProductID = P.ProductID;

SELECT CustomerID, SUM(Price \* Quantity) AS TotalSpend FROM Orders

JOIN Products ON Orders.ProductID = Products.ProductID

GROUP BY CustomerID

ORDER BY TotalSpend DESC

OFFSET 0 ROWS FETCH NEXT 3 ROWS ONLY;

SELECT \* FROM Customers C

LEFT JOIN Orders O ON C.CustomerID = O.CustomerID

WHERE O.CustomerID IS NULL;

SELECT \* FROM Products CROSS JOIN Products AS P2;

**Q9. (5 marks)**

SELECT \* FROM Orders O1

WHERE Quantity \* (SELECT Price FROM Products WHERE ProductID = O1.ProductID) > (

SELECT AVG(Quantity \* Price) FROM Orders O2

JOIN Products P2 ON O2.ProductID = P2.ProductID

WHERE O2.CustomerID = O1.CustomerID

);

SELECT \* FROM Customers C

WHERE EXISTS (

SELECT 1 FROM Orders O WHERE C.CustomerID = O.CustomerID

GROUP BY ProductID HAVING COUNT(DISTINCT ProductID) >= 2

);

SELECT \* FROM Customers C

WHERE (SELECT COUNT(\*) FROM Orders O WHERE O.CustomerID = C.CustomerID) > ALL (

SELECT COUNT(\*) FROM Orders O2 GROUP BY O2.CustomerID

);

SELECT \* FROM Products

WHERE Price > ANY (

SELECT Price FROM Products WHERE Category = 'Electronics'

);

SELECT TOP 3 ProductID, SUM(Quantity) AS TotalSold FROM Orders

GROUP BY ProductID ORDER BY TotalSold DESC;

**Q10. (5 marks)**

SELECT C1.Name FROM SegmentA C1

INNER JOIN SegmentB C2 ON C1.CustomerID = C2.CustomerID;

SELECT \* FROM Products

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

MERGE Customers AS Target

USING (SELECT 1 AS DummyID) AS Source

ON Target.CustomerID = 1

WHEN MATCHED THEN UPDATE SET Email = 'updated@example.com'

WHEN NOT MATCHED THEN INSERT (CustomerID, Name, Email) VALUES (1, 'New Customer', 'new@example.com');

SELECT \* FROM Region1\_Customers

UNION

SELECT \* FROM Region2\_Customers;

WITH RankedCustomers AS (

SELECT CustomerID, SUM(Price \* Quantity) AS TotalSpend,

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

FROM Orders O

JOIN Products P ON O.ProductID = P.ProductID

GROUP BY CustomerID

)

SELECT \* FROM RankedCustomers WHERE Rank <= 5;