## QUESTION PAPER 1.

## 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**

SQL is a relational database which means that the representation is in Rows and Columns. SQL stands for Structured Query Language and it has a fixed schema. Examples of SQL are MySQL and Postgre.

ADVANTAGES AND DISADVANTAGES – ACID Compliance (adv), Less Scalable (disadv).

REAL TIME EXAMPLE – Employee database.

**NoSQL**

It is a non-relational database hence making it more flexible and dynamic than SQL. NoSQL has key- value pair as its data structure. Examples of NoSQL are MongoDB and Firebase.

ADVANTAGES AND DISADVANTAGES – Handles large volume easily (adv), No standard query language (disadv)

REAL TIME EXAMPLE – Netflix.

Q2. (2 marks) Given the below unnormalized data, convert it to 1NF, 2NF, and 3NF: Student (StudentID, Name, CourseID, CourseName, InstructorName, InstructorPhone).

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

1NF data – Split into repeating groups

2NF data – Split into 2 tables

Student(StudentID, Name)

Enrollment(StudentID, CourseID)

Course(CourseID, CourseName, InstructorName, InstructorPhone)

3NF data - Split Instructor into new table

Instructor(InstructorName, InstructorPhone)

Course(CourseID, CourseName, InstructorName)

Q3. (5 marks) a) Create a database named StudentDB. b) Create a table Students with fields: StudentID, Name, DOB, Email. c) Rename the table to Student\_Info. d) Add a column PhoneNumber. e) Drop the table.

1. CREATE DATABASE StudentDB;

USE StudentDB;

1. CREATE TABLE Students (

StudentID INT PRIMARY KEY,

Name VARCHAR(50),

DOB DATE,

Email VARCHAR(100));

1. RENAME TABLE Students TO Student\_Info;
2. ALTER TABLE Student\_Info ADD PhoneNumber Varchar(15);
3. DROP TABLE STUDENT INFO;

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

Q4. (5 marks) a) Insert 3 student records into Student\_Info. b) Update one student's phone number. c) Delete one student whose email ends with @gmail.com. d) Retrieve only names and emails of students born after the year 2000. e) Retrieve distinct domain names from the email column.

1. INSERT INTO Student\_Info VALUES

(1, 'Anas', '2002-05-10', 'anas@gmail.com', 9876543210),

(2, 'Priya', '1999-12-01', 'priya@yahoo.com', 8765432109),

(3, 'Ravi', '2003-08-20', 'ravi@gmail.com', 9988776655);

1. UPDATE Student\_Info SET PhoneNumber = 9090909090 WHERE StudentID = 2;
2. DELETE FROM Student\_Info WHERE Email LIKE '%@gmail.com';
3. SELECT Name, Email FROM Student\_Info WHERE YEAR(DOB) > 2000;
4. SELECT DISTINCT SUBSTRING\_INDEX(Email, '@', -1) AS domain FROM Student\_Info;

Q5. (5 marks) a) Retrieve students with names starting with 'A'. b) Retrieve students with phone number between 9000000000 and 9999999999. c) Retrieve students using IN operator on city names. d) Use AND, OR to filter students based on age and email provider. e) Use table and column aliasing in a query to get all student names and DOBs.

1. 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) < 25 AND Email LIKE '%@gmail.com';

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

Q6. (5 marks) Create a new table Marks(StudentID, Subject, Marks). Insert at least 3 rows. a) Display student IDs and their subjects where marks > 70. b) Display subjects with average marks. c) Filter subjects with average marks between 60 and 90.

CREATE TABLE Marks (

StudentID INT,

Subject VARCHAR(50),

Marks INT

);

INSERT INTO Marks VALUES

(1, 'Maths', 85),

(1, 'English', 78),

(2, 'Maths', 92);

a. SELECT StudentID, Subject FROM Marks WHERE Marks > 70;

b. SELECT Subject, AVG(Marks) AS avg\_marks 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. (5 marks) a) Get the current date and format it as "YYYY-MM-DD". b) Extract month and year from a DOB column. c) Convert a student's name to uppercase. d) Round off marks to 2 decimal places. e) Use system function to return user name or current database.

a. SELECT DATE\_FORMAT(CURDATE(), '%Y-%m-%d') AS today;

b. SELECT MONTH(DOB) AS birth\_month, YEAR(DOB) AS birth\_year FROM Student\_Info;

c. SELECT UPPER(Name) FROM Student\_Info;

d. SELECT ROUND(Marks, 2) FROM Marks;

e. SELECT DATABASE() AS db\_name, USER() AS user;

Q8. (5 marks) a) Display total marks of each student. b) Display subject-wise highest mark. c) Use GROUP BY and HAVING to display subjects with average marks > 75.

a. SELECT StudentID, SUM(Marks) AS total\_marks FROM Marks GROUP BY StudentID;

b. SELECT Subject, MAX(Marks) AS highest FROM Marks GROUP BY Subject;

c. SELECT Subject, AVG(Marks) AS avg\_marks

FROM Marks

GROUP BY Subject

HAVING AVG(Marks) > 75;

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

Q9. (5 marks) a) Inner Join to retrieve students and their courses. b) Left Join to get all students even if not enrolled. c) Right Join to get all courses even if no students. d) Full Outer Join equivalent using UNION. e) Cross Join to show all combinations.

Assume tables: Students, Courses, Enrollments

a. Inner Join

SELECT s.Name, c.CourseName

FROM Students s

JOIN Enrollments e ON s.StudentID = e.StudentID

JOIN Courses c ON e.CourseID = c.CourseID;

b. Left Join

SELECT s.Name, c.CourseName

FROM Students s

LEFT JOIN Enrollments e ON s.StudentID = e.StudentID

LEFT JOIN Courses c ON e.CourseID = c.CourseID;

c.Right Join

SELECT s.Name, c.CourseName

FROM Courses c

RIGHT JOIN Enrollments e ON c.CourseID = e.CourseID

RIGHT JOIN Students s ON e.StudentID = s.StudentID;

d.Full Outer Join (using UNION)

SELECT s.Name, c.CourseName

FROM Students s

LEFT JOIN Enrollments e ON s.StudentID = e.StudentID

LEFT JOIN Courses c ON e.CourseID = c.CourseID

UNION

SELECT s.Name, c.CourseName

FROM Courses c

LEFT JOIN Enrollments e ON c.CourseID = e.CourseID

LEFT JOIN Students s ON e.StudentID = s.StudentID;

e.Cross Join

SELECT s.Name, c.CourseName

FROM Students s

CROSS JOIN Courses c;

Q10. (5 marks) a) Students who scored more than average in 'Maths'. b) Students not in the Marks table. c) Use EXISTS to get students with at least one subject. d) Use ALL to find those scoring more than all in 'Science'. e) Use ANY for students scoring better than some in 'English'.

a. SELECT StudentID 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 StudentID FROM Marks

WHERE Subject = 'Science' AND Marks > ALL (

SELECT Marks FROM Marks WHERE Subject = 'Science'

);

e.SELECT StudentID FROM Marks

WHERE Subject = 'English' AND Marks > ANY (

SELECT Marks FROM Marks WHERE Subject = 'English'

);

Q11. (5 marks) a) UNION of student names from two tables. b) INTERSECT to find common students. c) EXCEPT to list students in Students but not in Marks. d) MERGE concept or simulate with UPDATE and INSERT. e) Correlated subquery to list students with above average per subject

a. SELECT Name FROM Students

UNION

SELECT Name FROM Student\_Info;

b. SELECT s.Name

FROM Students s

JOIN Student\_Info si ON s.Name = si.Name;

c. SELECT Name FROM Students

WHERE Name NOT IN (SELECT Name FROM Student\_Info);

d. INSERT INTO Students (StudentID, Name)

VALUES (4, 'Riya')

ON DUPLICATE KEY UPDATE Name = 'Riya';

e. SELECT s.Name

FROM Students s

WHERE EXISTS (

SELECT 1 FROM Marks m

WHERE m.StudentID = s.StudentID

GROUP BY m.Subject

HAVING AVG(m.Marks) > (

SELECT AVG(Marks) FROM Marks WHERE Subject = m.Subject

)

);