**SQL Practical Answer Document**

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

**Q1: SQL vs NoSQL**

| Feature | SQL | NoSQL |
| --- | --- | --- |
| Type | Relational DB (Structured) | Non-relational (Document, Key-Value etc) |
| Schema | Fixed schema | Dynamic schema |
| Example | MySQL, PostgreSQL | MongoDB, Cassandra |
| Use case | Banking systems | Real-time analytics, IoT, social apps |
| Advantage 1 | Strong ACID compliance | High scalability |
| Advantage 2 | Structured query language for complex joins | Flexible data models |
| Disadvantage 1 | Not good with unstructured data | Lacks mature query languages |
| Disadvantage 2 | Limited horizontal scaling | Weaker ACID guarantees |

**Q2: Normalization**

* **UNF:** Student (StudentID, Name, CourseID, CourseName, InstructorName, InstructorPhone)
* **1NF:**
  + Ensure atomicity:
    - Separate repeating groups
* **2NF:**
  + Remove partial dependency:
    - Student (StudentID, Name)
    - Course (CourseID, CourseName, InstructorName, InstructorPhone)
    - Enrollment (StudentID, CourseID)
* **3NF:**
  + Remove transitive dependency:
    - Instructor (InstructorName, InstructorPhone)
    - Course (CourseID, CourseName, InstructorName)

**Q3:**

CREATE DATABASE StudentDB;  
  
USE StudentDB;  
  
CREATE TABLE Students (  
 StudentID INT PRIMARY KEY,  
 Name VARCHAR(50),  
 DOB DATE,  
 Email VARCHAR(100)  
);  
  
ALTER TABLE Students RENAME TO Student\_Info;  
  
ALTER TABLE Student\_Info ADD PhoneNumber BIGINT;  
  
DROP TABLE Student\_Info;

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

**Q4:**

-- a)  
INSERT INTO Student\_Info VALUES (1, 'Alice', '2002-01-01', 'alice@gmail.com', 9876543210);  
INSERT INTO Student\_Info VALUES (2, 'Bob', '1999-05-05', 'bob@yahoo.com', 9765432109);  
INSERT INTO Student\_Info VALUES (3, 'Cathy', '2001-03-03', 'cathy@gmail.com', 9654321098);  
  
-- b)  
UPDATE Student\_Info SET PhoneNumber = 9123456789 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:**

-- 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 ('Mumbai', 'Chennai', 'Delhi');  
  
-- d)  
SELECT \* FROM Student\_Info WHERE YEAR(CURDATE()) - YEAR(DOB) > 20 AND Email LIKE '%@gmail.com';  
  
-- e)  
SELECT s.Name AS Student\_Name, s.DOB AS Date\_of\_Birth FROM Student\_Info s;

**Q6:**

CREATE TABLE Marks (  
 StudentID INT,  
 Subject VARCHAR(50),  
 Marks FLOAT  
);  
  
INSERT INTO Marks VALUES (1, 'Maths', 80);  
INSERT INTO Marks VALUES (2, 'Science', 90);  
INSERT INTO Marks VALUES (3, 'English', 75);  
  
-- 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:**

-- a)  
SELECT DATE\_FORMAT(CURDATE(), '%Y-%m-%d') AS Current\_Date;  
  
-- b)  
SELECT MONTH(DOB) AS BirthMonth, YEAR(DOB) AS BirthYear FROM Student\_Info;  
  
-- c)  
SELECT UPPER(Name) AS Name\_Upper FROM Student\_Info;  
  
-- d)  
SELECT ROUND(Marks, 2) AS RoundedMarks FROM Marks;  
  
-- e)  
SELECT CURRENT\_USER() AS UserName, DATABASE() AS CurrentDB;

**Q8:**

-- 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:**

-- a)  
SELECT s.StudentID, s.Name, c.CourseName FROM Student\_Info s  
JOIN Enrollment e ON s.StudentID = e.StudentID  
JOIN Course c ON e.CourseID = c.CourseID;  
  
-- b)  
SELECT s.Name FROM Student\_Info s  
LEFT JOIN Enrollment e ON s.StudentID = e.StudentID;  
  
-- c)  
SELECT c.CourseName FROM Course c  
RIGHT JOIN Enrollment e ON c.CourseID = e.CourseID;  
  
-- d)  
SELECT s.Name, c.CourseName FROM Student\_Info s  
LEFT JOIN Enrollment e ON s.StudentID = e.StudentID  
LEFT JOIN Course c ON e.CourseID = c.CourseID  
UNION  
SELECT s.Name, c.CourseName FROM Course c  
LEFT JOIN Enrollment e ON c.CourseID = e.CourseID  
LEFT JOIN Student\_Info s ON e.StudentID = s.StudentID;  
  
-- e)  
SELECT \* FROM Student\_Info CROSS JOIN Course;

**Q10:**

-- a)  
SELECT \* FROM Marks m WHERE Subject = 'Maths' AND Marks > (SELECT AVG(Marks) FROM Marks WHERE Subject = 'Maths');  
  
-- b)  
SELECT \* FROM Student\_Info WHERE StudentID NOT IN (SELECT DISTINCT StudentID FROM Marks);  
  
-- c)  
SELECT \* FROM Student\_Info s WHERE EXISTS (SELECT 1 FROM Marks m WHERE s.StudentID = m.StudentID);  
  
-- d)  
SELECT \* FROM Marks m1 WHERE Subject = 'Science' AND Marks > ALL (SELECT Marks FROM Marks WHERE Subject = 'Science');  
  
-- e)  
SELECT \* FROM Marks m1 WHERE Subject = 'English' AND Marks > ANY (SELECT Marks FROM Marks WHERE Subject = 'English');

**Q11:**

-- a)  
SELECT Name FROM Student\_Info  
UNION  
SELECT Name FROM Alumni;  
  
-- b)  
SELECT Name FROM Student\_Info  
INTERSECT  
SELECT Name FROM Alumni;  
  
-- c)  
SELECT Name FROM Student\_Info  
EXCEPT  
SELECT Name FROM Marks;  
  
-- d)  
-- Simulating MERGE  
UPDATE Student\_Info SET Email = 'newemail@test.com' WHERE StudentID = 1;  
INSERT INTO Student\_Info (StudentID, Name, DOB, Email, PhoneNumber)  
SELECT 4, 'New Student', '2003-04-04', 'new@test.com', 9090909090  
WHERE NOT EXISTS (SELECT 1 FROM Student\_Info WHERE StudentID = 4);  
  
-- e)  
SELECT \* FROM Marks m1  
WHERE Marks > (  
 SELECT AVG(Marks) FROM Marks m2 WHERE m1.Subject = m2.Subject  
);