MySQL Practice Questions

1. Create a database named college_db

➔
```
CREATE DATABASE college_db;
```

2. Create a table students with fields: id, name, age, department.

➔
```
USE college_db;
CREATE TABLE students (
   id INT PRIMARY KEY,
   name VARCHAR(100),
   age INT,
   department VARCHAR(100)
);
```

3. Insert 5 records into the students table.

➔
```
INSERT INTO students (id, name, age, department)
VALUES
(1, 'Namrata', 21, 'Computer Science'),
(2, 'Pari', 19, 'Mechanical'),
(3, 'Aayush', 22, 'Electrical'),
(4, 'Ajay', 20, 'Computer Science'),

(5, 'Rinki', 23, 'Civil');
```

4. Write a query to fetch all records from students.

➔
```
SELECT * FROM students;
```

5. Fetch students whose age is greater than 20.

➔
```
SELECT * FROM students WHERE age > 20;
```

6. Update the department of a student where name is ‘John’.

➔
```
UPDATE students SET department = 'Mechanical'
WHERE name = 'Rinki';
```

7. Delete a student whose ID is 3.

➔ DELETE FROM students WHERE id = 3;

8. Select students ordered by age in descending order.

➔ SELECT * FROM students ORDER BY age DESC;

9. Fetch only distinct departments from the students table.

➔ SELECT DISTINCT department FROM students;

10. Count the number of students in the table.

➔ SELECT COUNT(*) AS total_students FROM students;

11. Rename the students table to student_info.

➔ RENAME TABLE students TO student_info;

12. Add a new column email to the student_info table.

➔ ALTER TABLE student_info ADD email VARCHAR(100);

13. Query to find students whose name starts with ‘A’.

➔ SELECT * FROM student_info WHERE name LIKE 'A%';

14. Display students whose age is between 18 and 25.

➔ SELECT * FROM student_info WHERE age BETWEEN 18 AND 25;

15. Query to find the student with the highest age.

➔ SELECT * FROM student_info ORDER BY age DESC LIMIT 1;

16. Use LIMIT to display the first 3 students.

➔ SELECT * FROM student_info LIMIT 3;

17. Create a table courses with fields: course_id, course_name, credits.

➔ CREATE TABLE courses (
  course_id INT PRIMARY KEY,

course_name VARCHAR(100),

credits INT

);

18. Insert 3 records into the courses table.

➔ INSERT INTO courses (course_id, course_name, credits) VALUES

(101, 'Data Structures', 4),

(102, 'Thermodynamics', 3),
(103, 'Electrical Circuits', 4);

19. Select all students whose department is ‘Computer Science’

➔ SELECT * FROM student_info WHERE department = 'Computer Science';

20. Use IN to fetch students from specific departments.

➔ SELECT * FROM student_info WHERE department IN ('Computer Science', 'Mechanical');

21. Use BETWEEN to find students aged between 20 and 30.

➔ SELECT * FROM student_info WHERE age BETWEEN 20 AND 30;

22. Query to display current system date and time.

➔ SELECT NOW() AS current_datetime;

23. Use AS to rename a column in the SELECT query.

➔ SELECT name AS student_name, age AS student_age FROM student_info;

24. Query to fetch all data except students of a particular department.

➔ SELECT * FROM student_info WHERE department != 'Civil';

25. Delete all records from the student_info table without dropping the table.

➔ DELETE FROM student_info;

26. Create a marks table with fields: student_id, subject, marks.

➔
```
CREATE TABLE marks (

 student_id INT,

  subject VARCHAR(100),
 marks INT
 );
```

27. Insert at least 5 records into the marks table.

➔
```
INSERT INTO marks (student_id, subject, marks)
VALUES
(1, 'Math', 85),
(1, 'Science', 90),
(2, 'Math', 70),
(2, 'Science', 75),
(4, 'Math', 95);
```

28. Use JOIN to combine students and marks data.

➔
```
SELECT si.id, si.name, m.subject, m.marks
FROM student_info si
JOIN marks m ON si.id = m.student_id;
```

29. Query to calculate average marks per student.

➔
```
SELECT student_id, AVG(marks) AS average_marks
FROM marks
GROUP BY student_id;
```

30. Use GROUP BY to find total marks obtained by each student.

➔
```
SELECT student_id, SUM(marks) AS total_marks
FROM marks
GROUP BY student_id;
```

31. Use HAVING to find students who scored more than 200 in total.

➔
```
SELECT student_id, SUM(marks) AS total_marks
FROM marks
GROUP BY student_id
HAVING total_marks > 200;
```

32. Fetch students with the same age using GROUP BY and COUNT()

➔
```
SELECT age, COUNT(*) AS count
FROM student_info
GROUP BY age
HAVING count > 1;
```

33. INNER JOIN, LEFT JOIN, RIGHT JOIN with explanation.

➔
```
-- INNER JOIN: Only matched records
SELECT si.name, m.subject FROM student_info si
INNER JOIN marks m ON si.id = m.student_id;

-- LEFT JOIN: All students, even if no marks
SELECT si.name, m.subject FROM student_info si
LEFT JOIN marks m ON si.id = m.student_id;

-- RIGHT JOIN: All marks entries, even if student
missing
```

```
SELECT si.name, m.subject FROM student_info si
RIGHT JOIN marks m ON si.id = m.student_id;
```

34. Create a new table with a PRIMARY KEY and AUTO_INCREMENT.

➔
```
CREATE TABLE teachers (
    teacher_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100)
);
```

35. Create a table with a FOREIGN KEY referencing another table.

➔
```
CREATE TABLE attendance (
    student_id INT,
    date DATE,
    status VARCHAR(10),
    FOREIGN KEY (student_id) REFERENCES student_info(id)
);
```

36. Subquery to find the maximum marks in marks table.

➔
```
SELECT MAX(marks) AS highest_marks FROM marks;
```

37. Create a VIEW to display student names and their total marks.

➔
```
CREATE VIEW student_totals AS
SELECT si.name, SUM(m.marks) AS total_marks
FROM student_info si
JOIN marks m ON si.id = m.student_id
GROUP BY si.name;
```

38. Subquery to list students who scored more than average mark.

➔
```
SELECT * FROM marks
WHERE marks > (SELECT AVG(marks) FROM marks);
```

39. Stored procedure to insert new student data.

➔
```
DELIMITER $$

CREATE PROCEDURE AddStudent(
   IN s_name VARCHAR(100),
   IN s_age INT,
   IN s_dept VARCHAR(100)
)
BEGIN
   INSERT INTO student_info (name, age, department)
VALUES (s_name, s_age, s_dept);
END$$
DELIMITER ;
```

40. Stored procedure to update student department.

➔
```
DELIMITER $$

CREATE PROCEDURE UpdateDepartment(
   IN s_id INT,
   IN new_dept VARCHAR(100)
)
BEGIN
   UPDATE student_info SET department = new_dept
WHERE id = s_id;
END$$
```

```
DELIMITER ;
```

41. User-defined function to calculate grade from marks.

➔

```
DELIMITER $$

CREATE FUNCTION GetGrade(mark INT)
RETURNS VARCHAR(2)
DETERMINISTIC
BEGIN
   DECLARE grade VARCHAR(2);
   IF mark >= 90 THEN SET grade = 'A';
   ELSEIF mark >= 75 THEN SET grade = 'B';
   ELSEIF mark >= 60 THEN SET grade = 'C';
   ELSE SET grade = 'F';
   END IF;
   RETURN grade;
END$$

DELIMITER ;
```

42. Trigger to log insert operations on student_info.

➔

```
CREATE TABLE student_log (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   student_name VARCHAR(100),
   action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER $$
```

```
CREATE TRIGGER after_student_insert
AFTER INSERT ON student_info
FOR EACH ROW
BEGIN
   INSERT INTO student_log (student_name) VALUES
(NEW.name);
END$$

DELIMITER ;
```

43. Use a transaction to update multiple records atomically.

➔ START TRANSACTION;

```
UPDATE student_info SET department = 'IT' WHERE id
= 1;
UPDATE student_info SET department = 'IT' WHERE id
= 2;

COMMIT;
-- Use ROLLBACK; if needed to cancel changes
```

44. Query to find duplicate records using GROUP BY and HAVING.

➔ SELECT name, COUNT(*) AS count

```
FROM student_info
GROUP BY name
HAVING count > 1;
```

45. Create a backup of a database using mysqldump.

➔ mysqldump -u root -p college_db >

```
college_db_backup.sql
```

46. Restore a MySQL database from a backup file.

➔

```
mysql -u root -p college_db < college_db_backup.sql
```

47. import data from a CSV file into a MySQL table.

➔

```
LOAD DATA INFILE '/path/to/file.csv'
INTO TABLE student_info
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

48. create an index on student name for faster search.

➔

```
CREATE INDEX idx_name ON student_info(name);
```

49. Query to find the second highest mark in a subject.

➔

```
SELECT MAX(marks) AS second_highest
FROM marks
WHERE marks < (SELECT MAX(marks) FROM marks);
```

50. Drop the courses table and explain the effect.

➔

```
DROP TABLE courses;
```