MySQL Assignment Questions and Answers (1–50)

1. Create a database named college_db

```
CREATE DATABASE college_db;
```

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

```
CREATE TABLE students (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  age INT,
  department VARCHAR(50)
);
```

3. Insert 5 records into the students table

```
INSERT INTO students VALUES
(1, 'Alice', 21, 'Computer Science'),
(2, 'Bob', 19, 'Electronics'),
(3, 'Charlie', 22, 'Mechanical'),
(4, 'David', 20, 'Computer Science'),
(5, 'John', 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 =
'John';
```

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(*) 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. Write a 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. Write a 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 VALUES
(1, 'DBMS', 4),
(2, 'OOP', 3),
(3, 'DSA', 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', 'Electronics');
```

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

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

22. Write a query to display current system date and time

```
SELECT NOW();
```

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

```
SELECT name AS student_name FROM student_info;
```

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

```
SELECT * FROM student_info WHERE department != 'Mechanical';
```

25. Delete all records from the students 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(50),
  marks INT
);
```

27. Insert at least 5 records into the marks table

```
INSERT INTO marks VALUES
(1, 'DBMS', 85),
(2, 'DBMS', 78),
(1, 'OOP', 90),
(3, 'OOP', 88),
(2, 'DSA', 95);
```

28. Use JOIN to combine students and marks data

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

29. Write a query to calculate average marks per student

```
SELECT student_id, AVG(marks) AS avg_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. Use INNER JOIN, LEFT JOIN, RIGHT JOIN and explain the difference

```
-- INNER JOIN: only matching rows from both tables
SELECT * FROM student_info INNER JOIN marks ON student_info.id
= marks.student_id;

-- LEFT JOIN: all from student_info, matching from marks
SELECT * FROM student_info LEFT JOIN marks ON student_info.id =
marks.student_id;

-- RIGHT JOIN: all from marks, matching from student_info
SELECT * FROM student_info RIGHT JOIN marks ON student_info.id
= marks.student_id;
```

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

```
CREATE TABLE departments (
  dept_id INT AUTO_INCREMENT PRIMARY KEY,
  dept_name VARCHAR(100)
);
```

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

```
CREATE TABLE enrollments (
  id INT AUTO_INCREMENT PRIMARY KEY,
  student_id INT,
  course_id INT,
  FOREIGN KEY (student_id) REFERENCES student_info(id),
   FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

36. Write a subquery to find the maximum marks in the marks table

```
SELECT * FROM marks
WHERE marks = (SELECT MAX(marks) FROM marks);
```

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

```
CREATE VIEW student_total_marks AS
SELECT s.name, SUM(m.marks) AS total
FROM student_info s
JOIN marks m ON s.id = m.student_id
GROUP BY s.name;
```

38. Use a subquery to list students who scored more than the average mark

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

39. Create a stored procedure to insert new student data

```
DELIMITER //
CREATE PROCEDURE insert_student(IN s_name VARCHAR(50), IN
s_age INT, IN s_dept VARCHAR(50))
BEGIN
   INSERT INTO student_info(name, age, department) VALUES
(s_name, s_age, s_dept);
END //
DELIMITER ;
```

40. Create a stored procedure to update student department

```
DELIMITER //
CREATE PROCEDURE update_department(IN s_id INT, IN new_dept
VARCHAR(50))
BEGIN
   UPDATE student_info SET department = new_dept WHERE id =
s_id;
END //
DELIMITER ;
```

41. Create a user-defined function to calculate grade from marks

```
DELIMITER //
CREATE FUNCTION get_grade(m INT) RETURNS VARCHAR(10)
BEGIN
  RETURN CASE
    WHEN m >= 90 THEN 'A'
    WHEN m >= 75 THEN 'B'
    WHEN m >= 60 THEN 'C'
    ELSE 'D'
  END;
END //
DELIMITER ;
```

42. Create a trigger that logs insert operations on students

```
CREATE TABLE student_log (
  log_id INT AUTO_INCREMENT PRIMARY KEY,
  student_name VARCHAR(50),
  action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER //
CREATE TRIGGER log_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 age = age + 1 WHERE id = 1;
UPDATE student_info SET age = age + 1 WHERE id = 2;
COMMIT;
```

44. Write a 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 ','
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. Write a query to find the second highest mark in a subject

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

50. Drop the courses table and explain the effect

```
DROP TABLE courses;
-- This will permanently delete the table and all its data.
```