1. Insert and Update with Integrity

CREATE TABLE students (

student\_id INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

marks INT CHECK (marks >= 0 AND marks <= 100));

2. String Function Challenge

CREATE TABLE customers (

customer\_id INT PRIMARY KEY,

full\_name VARCHAR(100) NOT NULL);

INSERT INTO customers VALUES

(1, 'Virat Kohli'),

(2, 'Rohit Sharma'),

(3, 'MS Dhoni'),

(4, 'KL Rahul'),

(5, 'Jasprit Bumrah');

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

CREATE TABLE sales (

sale\_id INT PRIMARY KEY,

sale\_date DATE,

category VARCHAR(100),

amount DECIMAL(10,2));

INSERT INTO sales VALUES

(1, '2025-06-10', 'Bats', 2000),

(2, '2025-05-01', 'Balls', 1500),

(3, '2025-01-15', 'Gloves', 3000),

(4, '2025-06-01', 'Pads', 1200);

SELECT

sale\_date,

MONTHNAME(sale\_date) AS month\_name,

YEAR(sale\_date) AS sale\_year,

DATEDIFF(CURDATE(), sale\_date) AS days\_ago

FROM sales;

4. Mathematical Functions on Salary

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(100),

salary DECIMAL(10, 2)

);

INSERT INTO employees VALUES

(1, 'Virat Kohli', 120000),

(2, 'Rohit Sharma', 95000);

SELECT

name,

salary,

salary \* 1.10 AS salary\_with\_hike,

ROUND(salary, -2) AS rounded\_salary

FROM employees;

5. System Function Check

SELECT

NOW() AS current\_datetime,

DATABASE() AS current\_database,

USER() AS logged\_in\_user;

-- 6. Demo: Custom Result Set

CREATE TABLE products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(100),

price DECIMAL(10,2),

category VARCHAR(50)

);

INSERT INTO products VALUES

(1, 'Bat', NULL, 'Gear'),

(2, 'Ball', 1500, 'Gear'),

(3, 'Helmet', 2500, 'Protection');

SELECT

UPPER(product\_name) AS product\_name\_uppercase,

IFNULL(price, 'Not Available') AS price\_status

FROM products;

7. Aggregate Functions Practice

CREATE TABLE transactions (

transaction\_id INT PRIMARY KEY,

sale\_amount DECIMAL(10,2)

);

INSERT INTO transactions VALUES

(1, 1000),

(2, 1500),

(3, 500),

(4, 2000);

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

category,

COUNT(\*) AS transaction\_count,

SUM(amount) AS total\_sales

FROM sales

GROUP BY category;

9. Inner Join for Orders and Customers

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_amount DECIMAL(10,2)

);

INSERT INTO orders VALUES

(101, 1, 5000),

(102, 2, 3000),

(103, 4, 4500);

SELECT

c.full\_name AS customer\_name,

o.order\_amount

FROM orders o

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

10. Left Join for Products with or without Orders

CREATE TABLE order\_details (

order\_id INT,

product\_id INT,

quantity INT

);

INSERT INTO order\_details VALUES

(101, 1, 2),

(102, 2, 1);

SELECT

p.product\_name,

od.quantity

FROM products p

LEFT JOIN order\_details od ON p.product\_id = od.product\_id;

11. Right Join for Customer Contacts

CREATE TABLE contacts (

contact\_id INT,

customer\_id INT,

phone VARCHAR(15)

);

INSERT INTO contacts VALUES

(1, 1, '1234567890'),

(2, 3, '9999999999');

SELECT

c.full\_name,

ct.phone

FROM contacts ct

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

12. Full Outer Join for Suppliers and Products (simulated)

CREATE TABLE suppliers (

supplier\_id INT,

supplier\_name VARCHAR(100)

);

INSERT INTO suppliers VALUES

(1, 'SG'),

(2, 'MRF');

SELECT s.supplier\_name, p.product\_name

FROM suppliers s

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

UNION

SELECT s.supplier\_name, p.product\_name

FROM suppliers s

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

13. Cross Join for Offers

CREATE TABLE offers (

offer\_id INT,

offer\_desc VARCHAR(100));

INSERT INTO offers VALUES

(1, '10% Off'),

(2, 'Free Shipping');

SELECT

p.product\_name,

o.offer\_desc

FROM products p

CROSS JOIN offers o;

14. Join with Aggregation

CREATE TABLE orders\_extended (

order\_id INT,

product\_id INT,

quantity INT,

price DECIMAL(10,2));

INSERT INTO orders\_extended VALUES

(1, 1, 2, 5000),

(2, 2, 1, 1500),

(3, 3, 3, 7500);

SELECT

p.category,

SUM(o.quantity) AS total\_quantity\_sold,

AVG(o.price) AS avg\_price

FROM orders\_extended o

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

GROUP BY p.category;

15. Join with Grouping and Filter

CREATE TABLE marks (

student\_id INT,

subject VARCHAR(50),

mark INT);

INSERT INTO marks VALUES

(1, 'Math', 80),

(1, 'Science', 85),

(2, 'Math', 95),

(2, 'Science', 90),

(3, 'Math', 70),

(3, 'Science', 75);

SELECT

s.name AS student\_name,

AVG(m.mark) AS average\_marks

FROM students s

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

GROUP BY s.student\_id, s.name

HAVING AVG(m.mark) > 75;