# SQL Challenges and Solutions

## 1. Insert and Update with Integrity

Create the 'students' table with constraints:

CREATE TABLE students (  
 student\_id INT PRIMARY KEY,  
 name VARCHAR(50) NOT NULL,  
 email VARCHAR(100) UNIQUE NOT NULL,  
 marks INT CHECK (marks >= 0 AND marks <= 100)  
);

Insert 5 records:

INSERT INTO students VALUES  
(1, 'John', 'john@example.com', 85),  
(2, 'alice', 'alice@example.com', 92),  
(3, 'rose', 'rose@example.com', 76),  
(4, 'priya', 'priya@example.com', 65),  
(5, 'ram', 'ram@example.com', 88);

Update a student's marks:

UPDATE students  
SET marks = 90  
WHERE student\_id = 4;

## 2. String Function Challenge

Split full name and display name lengths:

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 firstname\_length,  
 LENGTH(SUBSTRING\_INDEX(full\_name, ' ', -1)) AS lastname\_length  
FROM customers;

## 3. Date Function Usage

Extract month, year and days ago from sale\_date:

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

Calculate salary hike and round salary:

SELECT   
 salary,  
 salary \* 1.10 AS salary\_after\_hike,  
 ROUND(salary, -2) AS rounded\_salary  
FROM employees;

## 5. System Function Check

Retrieve system information:

SELECT   
 NOW() AS current\_datetime,  
 DATABASE() AS current\_database,  
 USER() AS login\_user;

## 6. Demo: Custom Result Set

Display product name in uppercase and handle NULL prices:

SELECT   
 UPPER(product\_name) AS product\_name\_upper,  
FROM products;

UPDATE products

SET price = 'Not Available'

WHERE price IS NULL;

# SQL Joins and Aggregates Practice

## 7. Aggregate Functions Practice

From a 'transactions' table:

SELECT   
 SUM(amount) AS total\_sales,  
 AVG(amount) AS average\_sale,  
 MAX(amount) AS max\_sale,  
 MIN(amount) AS min\_sale  
FROM transactions;

## 8. Grouping with Aggregation

Group by product category:

SELECT   
 category,  
 SUM(sale\_amount) AS total\_sales,  
 COUNT(\*) AS transaction\_count  
FROM sales  
GROUP BY category;

## 9. Inner Join for Orders and Customers

Join 'orders' and 'customers' to show only customers with orders:

SELECT   
 c.name AS customer\_name,  
 o.amount AS order\_amount  
FROM orders o  
INNER JOIN customers c ON o.customer\_id = c.id;

## 10. Left Join for Products with or without Orders

Show all products and their order details (if available):

SELECT   
 p.product\_name,  
 o.order\_id,  
 o.amount  
FROM products p  
LEFT JOIN orders o ON p.product\_id = o.product\_id;

## 11. Right Join for Customer Contacts

Show all customers, even if they don't have contact info:

SELECT   
 c.customer\_id,  
 c.name,  
 ct.phone\_number  
FROM contacts ct  
RIGHT JOIN customers c ON ct.customer\_id = c.customer\_id;

## 12. Full Outer Join for Suppliers and Products

List all suppliers and products with matching or NULL values:

SELECT   
 s.supplier\_name,  
 p.product\_name  
FROM suppliers s  
FULL OUTER JOIN products p ON s.supplier\_id = p.supplier\_id;

**13. Cross Join for Offers**

Show all possible combinations of products and offers.

SELECT

p.product\_name,

o.offer\_name

FROM products p

CROSS JOIN offers o;

**14. Join with Aggregation**

Join orders and products, group by product category, and show total quantity sold and average price.

SELECT

p.category,

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

AVG(p.price) AS average\_price

FROM orders o

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

GROUP BY p.category;

**15. Join with Grouping and Filter**

Join students and marks, display student name and average marks, and filter to show only students with average marks > 75

SELECT

s.name AS student\_name,

AVG(m.marks) AS average\_marks

FROM students s

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

GROUP BY s.name

HAVING AVG(m.marks) > 75;