# ASSIGNMENT 2

1. Inset and Update with Integrity: Create a 'students' table with constraints (NOT NULL, UNIQUE). Insert 5 records. Then, update a student's marks ensuring data integrity is maintained.

CREATE DATABASE ASSIGNMENT2;

USE ASSIGNMENT2;

CREATE TABLE STUDENTS(

studentID INT PRIMARY KEY AUTO\_INCREMENT,

Name varchar(30) NOT NULL,

Marks INT NOT NULL,

email VARCHAR(100) UNIQUE

);

INSERT INTO STUDENTS VALUES

(1,'Anas',90, 'anas@gmail.com'),

(2,'Jothi',80,'jothi@gmail.com'),

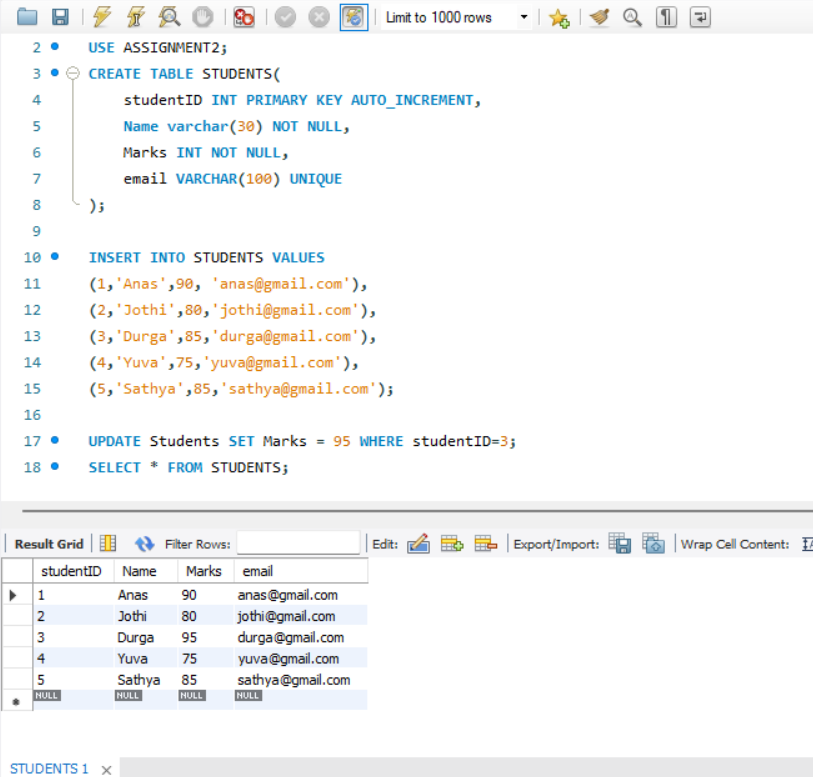
(3,'Durga',85,'durga@gmail.com'),

(4,'Yuva',75,'yuva@gmail.com'),

(5,'Sathya',85,'sathya@gmail.com');

UPDATE Students SET Marks = 95 WHERE studentID=3;

SELECT \* FROM STUDENTS;



2. String Function Challenge: Given a 'customers' table with a 'full\_name' column, write a query to display: - First name - Last name - Length of each name

CREATE TABLE CUSTOMERS(

fullname varchar(50));

INSERT INTO CUSTOMERS VALUES

('Anas Muhammad'),

('Abay Ragav'),

('Afeef Abdulla'),

('Deepak t'),

('Prem Kumar'),

('Loganathan');

SELECT

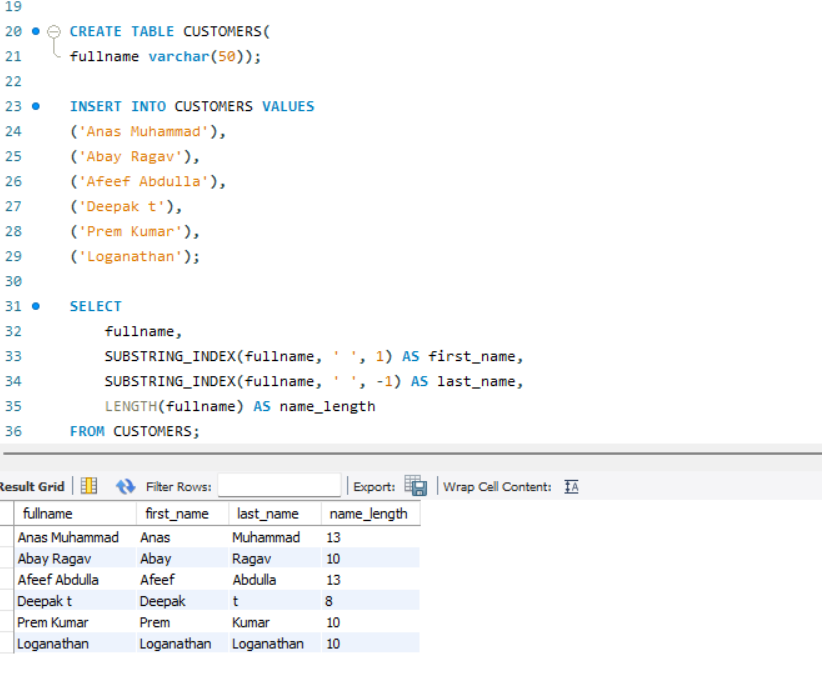
fullname,

SUBSTRING\_INDEX(fullname, ' ', 1) AS first\_name,

SUBSTRING\_INDEX(fullname, ' ', -1) AS last\_name,

LENGTH(fullname) AS name\_length

FROM CUSTOMERS;



3. Date Function Usage: From a 'sales' table with a 'sale\_date' column, write a query to: - Extract the month name and year - Display how many days ago the sale happened

CREATE TABLE sales (

sale\_id INT PRIMARY KEY,

sale\_date DATE

);

INSERT INTO sales VALUES

(1, '2024-05-10'),

(2, '2023-11-15'),

(3, '2025-01-01'),

(4, CURDATE());

SELECT

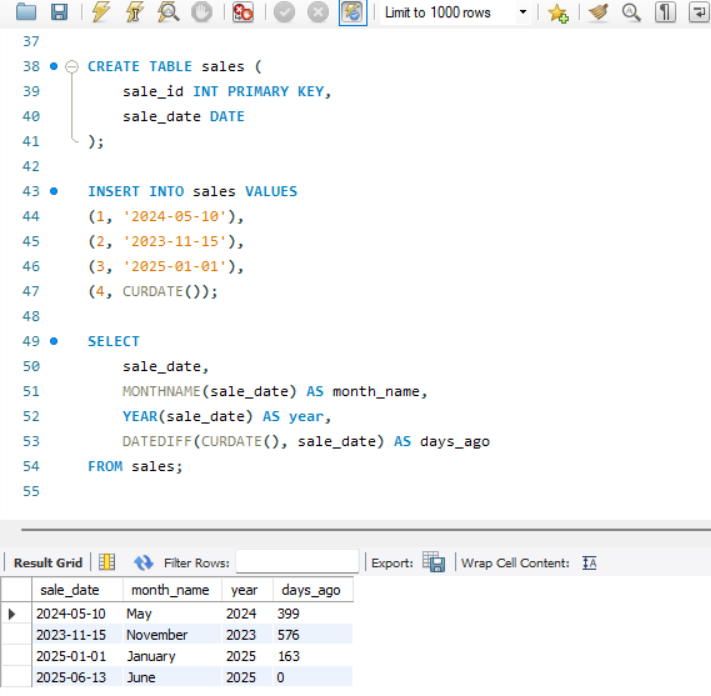
sale\_date,

MONTHNAME(sale\_date) AS month\_name,

YEAR(sale\_date) AS year,

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

FROM sales;



4. Mathematical Functions on Salary: In an 'employees' table, calculate: - Salary after a 10% hike - Round the salary to the nearest hundred

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(50),

salary DECIMAL(10,2)

);

INSERT INTO employees VALUES

(1, 'Anas', 52000),

(2, 'Sneha', 65000),

(3, 'Ravi', 45000);

SELECT \* FROM EMPLOYEES;

SELECT

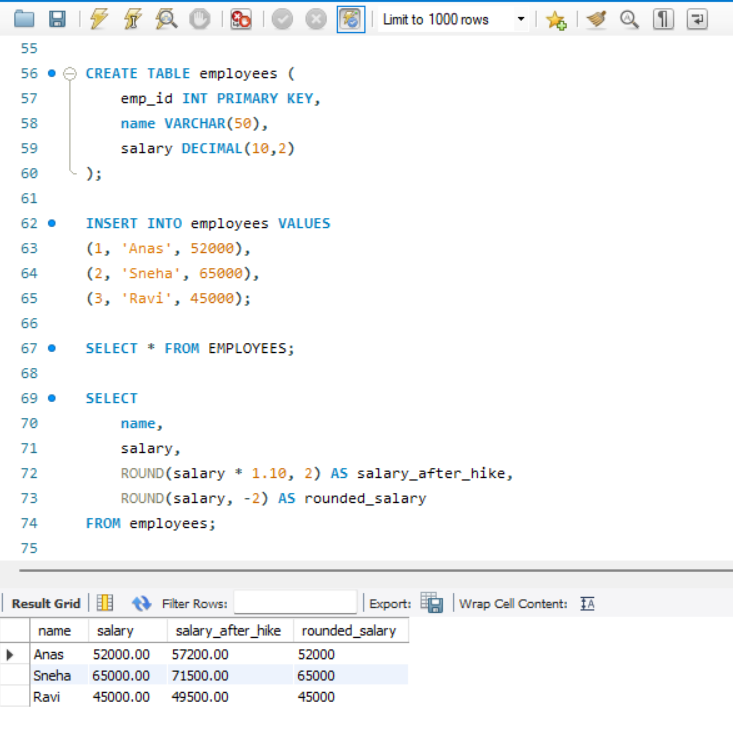
name,

salary,

ROUND(salary \* 1.10, 2) AS salary\_after\_hike,

ROUND(salary, -2) AS rounded\_salary

FROM employees;



5. System Function Check: Retrieve: - Current date and time - Database name and logged-in user

SELECT

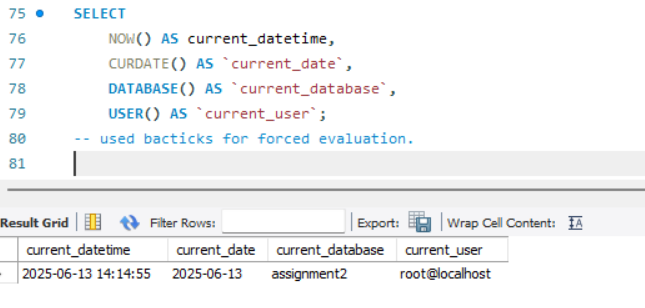
NOW() AS current\_datetime,

CURDATE() AS `current\_date`,

DATABASE() AS `current\_database`,

USER() AS `current\_user`;

-- used bacticks for forced evaluation.



6. Demo: Custom Result Set: From the 'products' table, write a query that: - Returns product name in uppercase - Replaces any NULL prices with 'Not Available'

CREATE TABLE products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(100),

price DECIMAL(10,2)

);

-- inserting values without null

INSERT INTO products VALUES

(1, 'Laptop', 50000),

(2, 'Tablet', 18000);

-- inserting value with null

INSERT INTO products (product\_id,product\_name) values (3,'Phone');

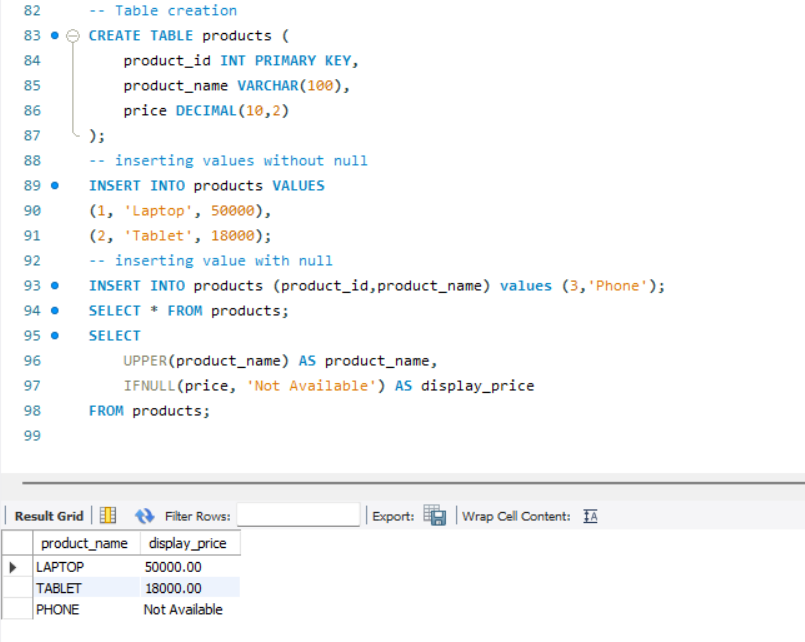
SELECT \* FROM products;

SELECT

UPPER(product\_name) AS product\_name,

IFNULL(price, 'Not Available') AS display\_price

FROM products;



7. Aggregate Functions Practice: From a 'transactions' table, get: - Total sales - Average sale value - Maximum and minimum sale on a single transaction

CREATE TABLE transactions (

trans\_id INT PRIMARY KEY,

sale\_amount DECIMAL(10,2)

);

INSERT INTO transactions VALUES

(1, 200),

(2, 450),

(3, 150),

(4, 700);

SELECT

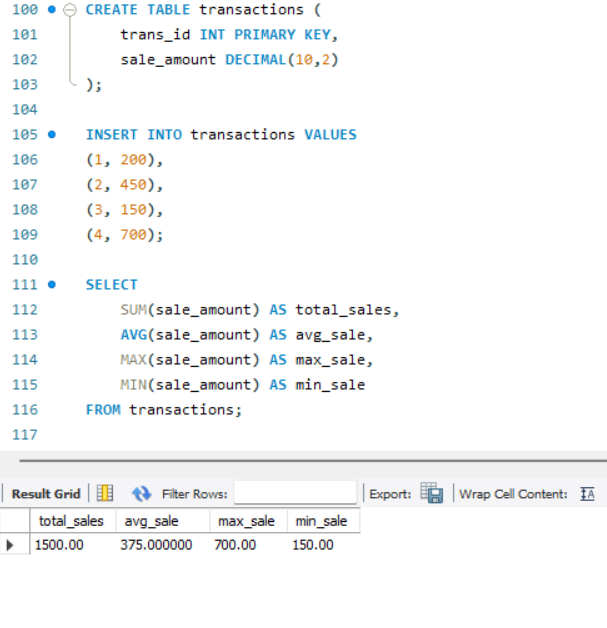
SUM(sale\_amount) AS total\_sales,

AVG(sale\_amount) AS avg\_sale,

MAX(sale\_amount) AS max\_sale,

MIN(sale\_amount) AS min\_sale

FROM transactions;



8. Grouping with Aggregation: From a 'sales' table: - Group by product category - Show total sales and number of transactions in each category

CREATE TABLE sales2 (

sale2\_id INT PRIMARY KEY,

product\_category VARCHAR(50),

sale\_amount DECIMAL(10,2)

);

INSERT INTO sales2 VALUES

(1, 'Electronics', 1000),

(2, 'Electronics', 2000),

(3, 'Clothing', 800),

(4, 'Clothing', 1200);

SELECT

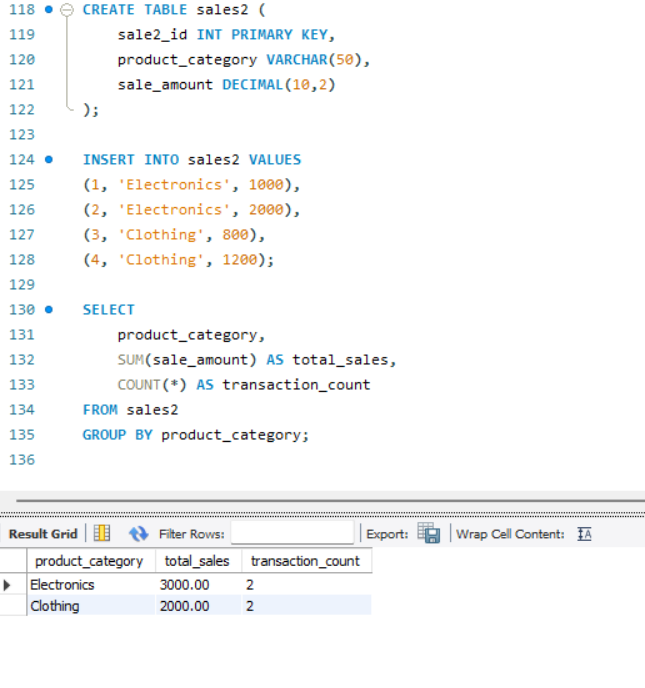
product\_category,

SUM(sale\_amount) AS total\_sales,

COUNT(\*) AS transaction\_count

FROM sales2

GROUP BY product\_category;



9. Inner Join for Orders and Customers: Join 'orders' and 'customers' to show: - Customer name - Order amount - Only for customers who made orders

ALTER TABLE CUSTOMERS

ADD COLUMN CustomerId INT PRIMARY KEY AUTO\_INCREMENT;

SELECT \* FROM CUSTOMERS;

ALTER TABLE CUSTOMERS

MODIFY CustomerId INT FIRST;

SELECT\*FROM CUSTOMERS;

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

customerId INT,

order\_amount DECIMAL(10,2),

FOREIGN KEY (customerId) REFERENCES CUSTOMERS(CustomerId)

);

INSERT INTO orders VALUES

(101,1,1200),

(102,2,800),

(103,3,600),

(104,4,1200),

(105,5,1800),

(106,6,1200)

;

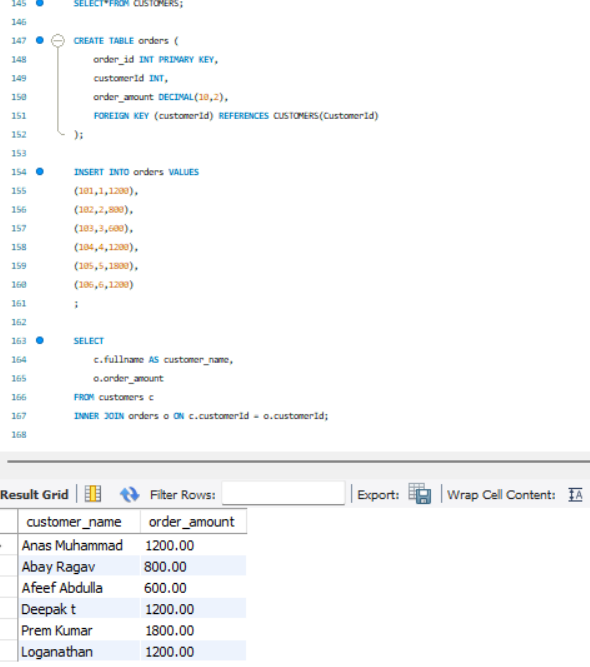
SELECT

c.fullname AS customer\_name,

o.order\_amount

FROM customers c

INNER JOIN orders o ON c.customerId = o.customerId;



10. Left Join for Products with or without Orders: Show all products with: - Their order details (if available) - Use LEFT JOIN

SELECT \* FROM PRODUCTS;

ALTER TABLE ORDERS

ADD COLUMN Product\_id INT;

ALTER TABLE ORDERS

ADD FOREIGN KEY (product\_id) REFERENCES PRODUCTS(product\_id);

SELECT

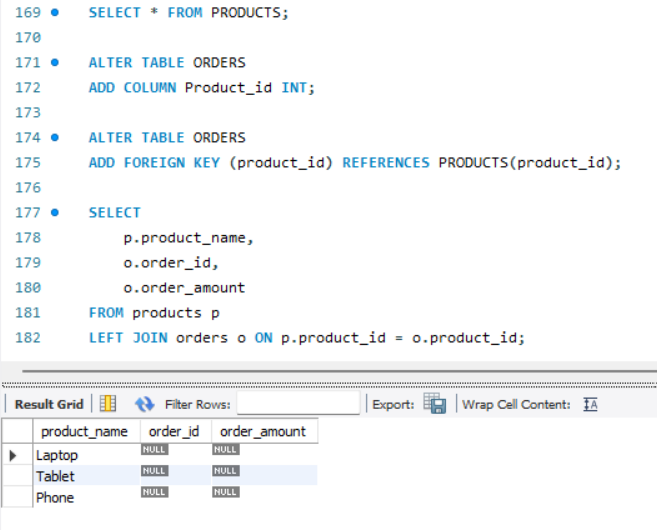
p.product\_name,

o.order\_id,

o.order\_amount

FROM products p

LEFT JOIN orders o ON p.product\_id = o.product\_id;



11. Right Join for Customer Contacts: Use a RIGHT JOIN between 'contacts' and 'customers' to display: - All customers, even if they don't have contact info

SELECT \* FROM CUSTOMERS;

CREATE TABLE contacts (

contact\_id INT PRIMARY KEY,

customerId INT,

phone VARCHAR(15),

email VARCHAR(50),

FOREIGN KEY (customerId) REFERENCES customers(customerId)

);

INSERT INTO contacts VALUES

(1, 1, '9999999999', 'anas@gmail.com'),

(2,2,'2020202020','abay@gmail.com'),

(3,3,'303030303030','afeef@gmail.com'),

(4,4,'4040404040','deepak@gmail.com');

SELECT

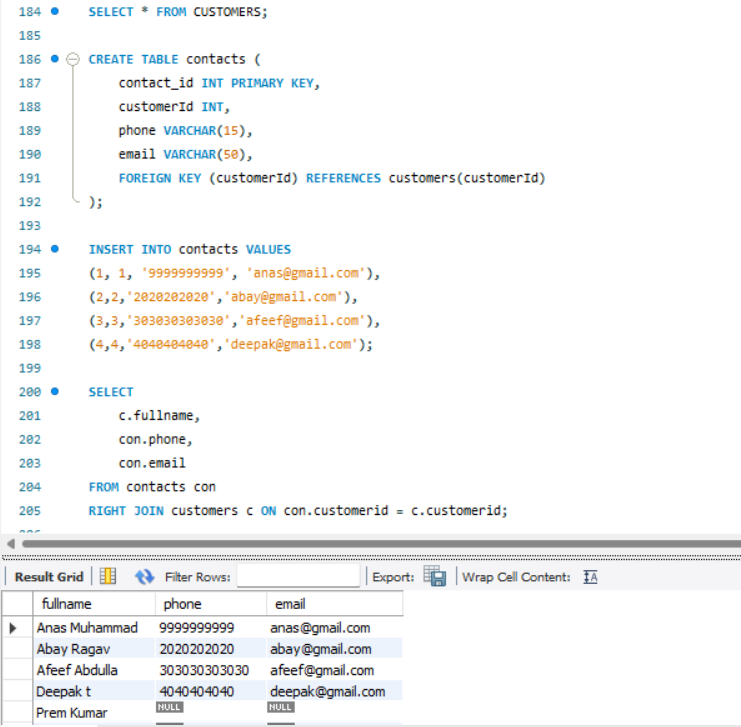
c.fullname,

con.phone,

con.email

FROM contacts con

RIGHT JOIN customers c ON con.customerid = c.customerid;



12. Full Outer Join for Suppliers and Products: Use a FULL OUTER JOIN to list: - All suppliers and products - Match supplier to product, or show NULLs where not available

CREATE TABLE suppliers (

supplier\_id INT PRIMARY KEY,

supplier\_name VARCHAR(50)

);

INSERT INTO suppliers VALUES

(1, 'Dell'),

(2, 'HP');

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

FROM suppliers s

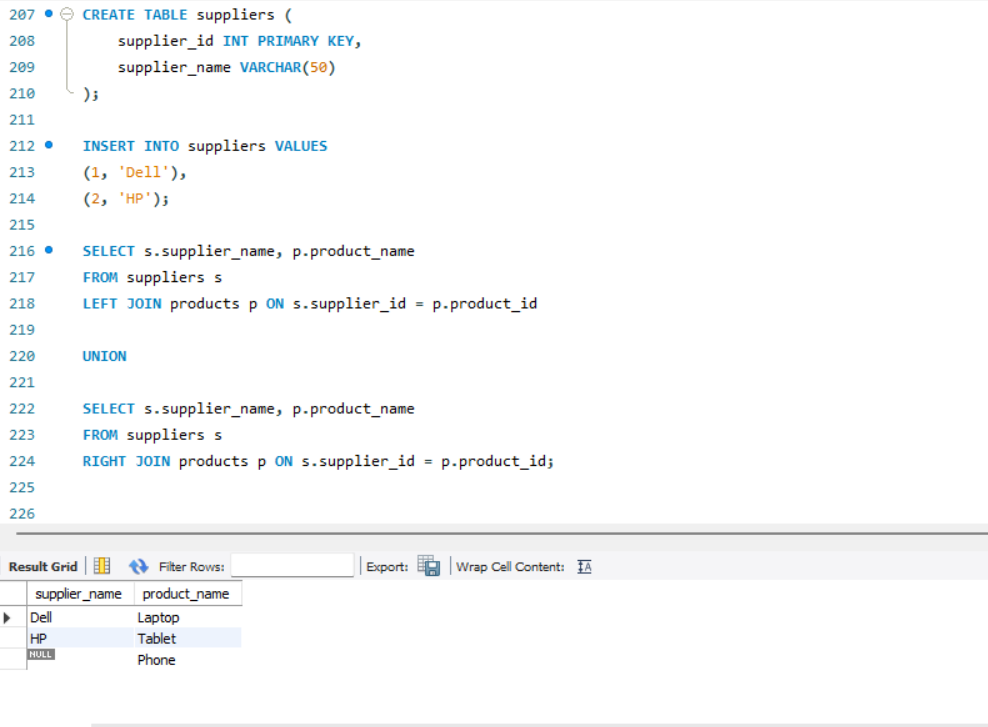
LEFT JOIN products p ON s.supplier\_id = p.product\_id

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: Suppose you have tables 'products' and 'offers'. Write a CROSS JOIN to show: - All possible combinations of products and offers

CREATE TABLE offers (

offer\_id INT PRIMARY KEY,

offer\_name VARCHAR(50)

);

INSERT INTO offers VALUES

(1, 'Summer Sale'),

(2, 'New Year Offer');

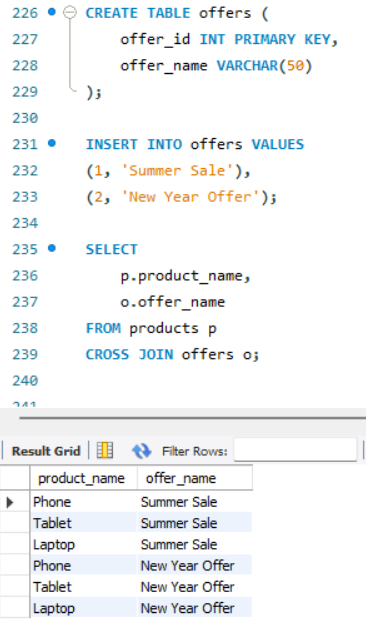
SELECT

p.product\_name,

o.offer\_name

FROM products p

CROSS JOIN offers o;



14. Join with Aggregation: Join 'orders' and 'products', then group by product category and: - Show total quantity sold and average price per category

ALTER TABLE products ADD COLUMN category VARCHAR(50);

UPDATE products SET category = 'Electronics' WHERE product\_id IN (1,3);

UPDATE products SET category = 'Gadgets' WHERE product\_id = 2;

ALTER TABLE orders ADD COLUMN quantity INT;

UPDATE orders SET quantity = 2 WHERE order\_id = 1;

UPDATE orders SET quantity = 1 WHERE order\_id = 2;

SELECT

p.category,

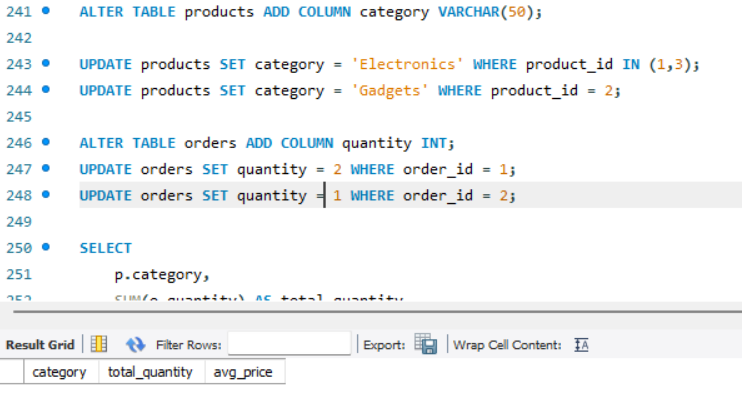
SUM(o.quantity) AS total\_quantity,

AVG(p.price) AS avg\_price

FROM orders o

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

GROUP BY p.category;



15. Demo: Join with Grouping and Filter: Join 'students' and 'marks' tables. Display: - Student name - Average marks - Filter to show only students with average marks > 75

CREATE TABLE marks (

mark\_id INT PRIMARY KEY,

student\_id INT,

marks INT

);

INSERT INTO marks VALUES (1, 1, 85), (2, 2, 75), (3, 3, 95), (4, 3, 90), (5, 4, 60);

SELECT \* FROM STUDENTS;

SELECT \* FROM MARKS;

ALTER TABLE MARKS

CHANGE student\_id studentid int;

SELECT

s.name,

AVG(m.marks) AS avg\_marks

FROM students s

JOIN marks m ON s.studentid = m.studentid

GROUP BY s.name

HAVING AVG(m.marks) > 75;

