Retail Sales Database Documentation

BY MUHAMMAD TALHA

13 Feb 2025

1. Introduction

The Retail Sales Database is designed to manage and track sales transactions, customers, employees, products, suppliers, and sales details. This database structure ensures efficient organization of data, enabling businesses to maintain accurate records of sales and inventory. It facilitates smooth business operations by streamlining order processing, managing customer information, tracking employee performance, and maintaining supplier relationships. The well-structured schema also allows for insightful sales analysis, helping businesses make informed decisions based on sales trends, product demand, and revenue generation.

2. Database Schema

The table schema representation:

i. customers (customer_id, name, email, phone, city)

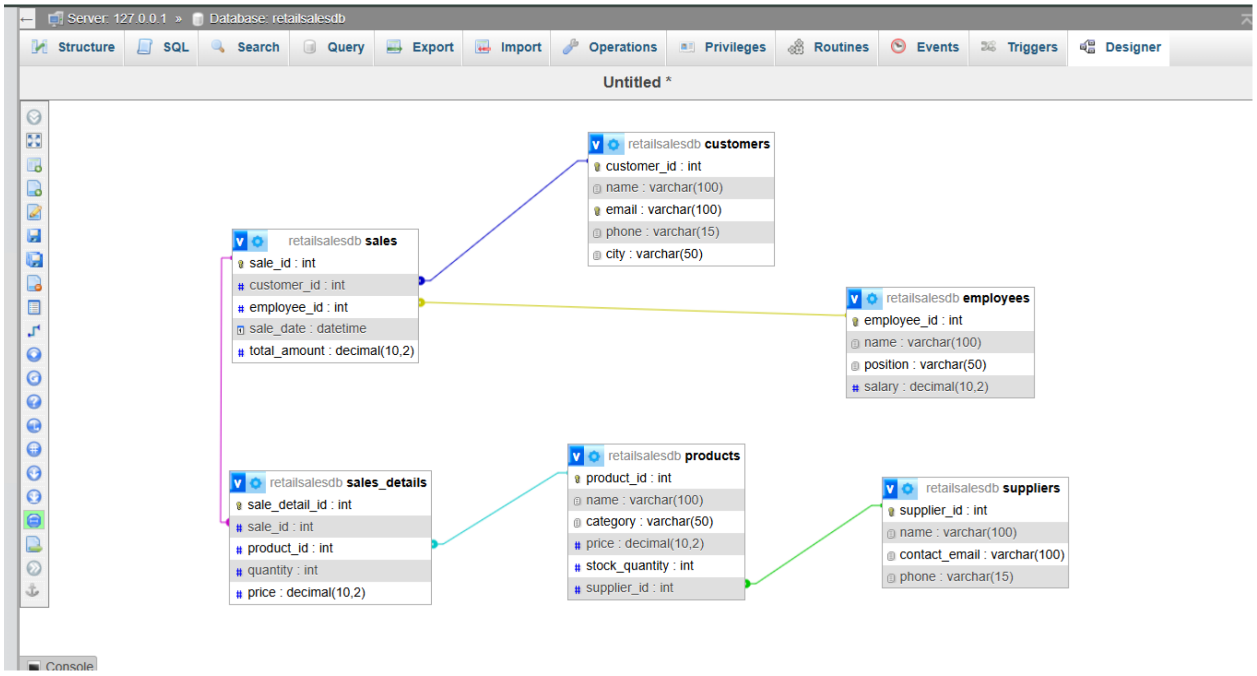
ii. employees (employee_id, name, position, salary)

iii. products (product_id, name, category, price, stock_quantity)

iv. suppliers (supplier_id, name, contact_email, phone)

v. sales (sale_id, customer_id, employee_id, sale_date, total_amount)

vi. sales_details (sale_detail_id, sale_id, product_id, quantity, price



---

3. Entities and Attributes

The database consists of six key entities, each representing a fundamental aspect of the retail sales process.

1. Customers

- **customer_id** (INT, Primary Key) - Unique identifier for each customer.
- **name** (VARCHAR(100)) - Full name of the customer.
- **email** (VARCHAR(100)) - Email address of the customer.
- **phone** (VARCHAR(15)) - Contact number.
- **city** (VARCHAR(50)) - City of residence.

2. Employees

- **employee_id** (INT, Primary Key) - Unique identifier for each employee.
- **name** (VARCHAR(100)) - Employee's full name.

- **position** (VARCHAR(50)) - Job role or designation.
- **salary** (DECIMAL(10,2)) - Salary of the employee.

3. Products

- **product_id** (INT, Primary Key) - Unique identifier for each product.
- **name** (VARCHAR(100)) - Name of the product.
- **category** (VARCHAR(50)) - Product category.
- **price** (DECIMAL(10,2)) - Price of the product.
- **stock_quantity** (INT) - Available stock count.

4. Suppliers

- **supplier_id** (INT, Primary Key) - Unique identifier for each supplier.
- **name** (VARCHAR(100)) - Supplier's name.
- **contact_email** (VARCHAR(100)) - Supplier’s email.
- **phone** (VARCHAR(15)) - Contact number.

5. Sales

- **sale_id** (INT, Primary Key) - Unique identifier for each sale.
- **customer_id** (INT, Foreign Key) - References customers.
- **employee_id** (INT, Foreign Key) - References employees.
- **sale_date** (DATETIME) - Date and time of sale.
- **total_amount** (DECIMAL(10,2)) - Total amount of the sale.

6. Sales Details

- **sale_detail_id** (INT, Primary Key) - Unique identifier for each sale detail entry.
- **sale_id** (INT, Foreign Key) - References sales.
- **product_id** (INT, Foreign Key) - References products.
- **quantity** (INT) - Number of units sold.
- **price** (DECIMAL(10,2)) - Price per unit.

4. Relationships Between Entities

The entities in the database are linked to ensure proper data flow and integrity.

i. **Customers & Sales (One-to-Many Relationship)**

   - A customer can have multiple sales transactions, but each sale is associated with only one customer.
   - **Foreign Key:** customer_id in **Sales** table references customer_id in **Customers** table.

ii. **Employees & Sales (One-to-Many Relationship)**

   - An employee (sales representative) processes multiple sales, but each sale is handled by a single employee.
   - **Foreign Key:** employee_id in **Sales** table references employee_id in **Employees** table.

iii. **Sales & Sales Details (One-to-Many Relationship)**

   - A single sale can have multiple items (products), but each sale detail entry belongs to a single sale.
   - **Foreign Key:** sale_id in **Sales Details** table references sale_id in **Sales** table.

iv. **Products & Sales Details (One-to-Many Relationship)**

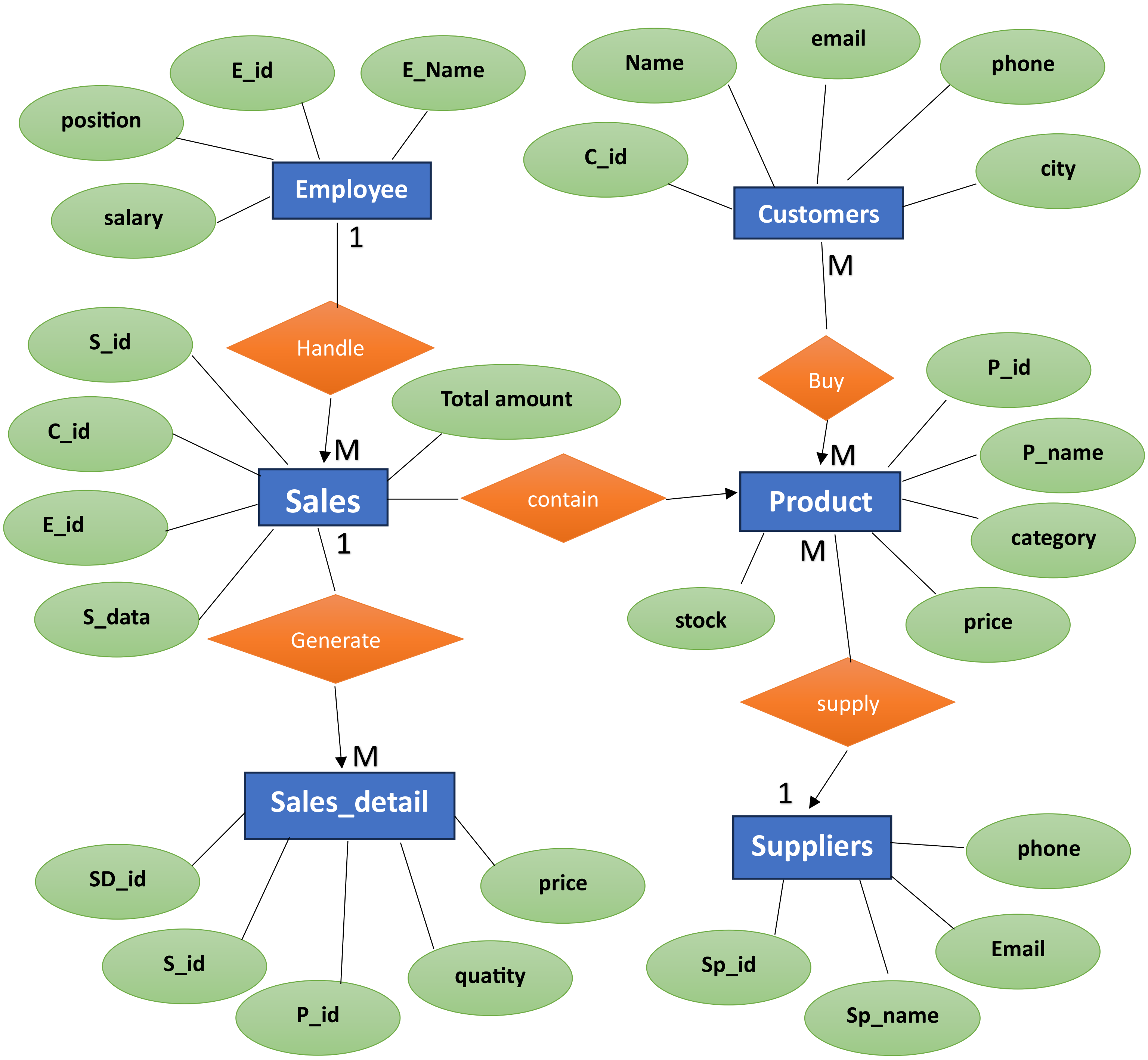
   - A product can be sold multiple times, but each sale detail entry corresponds to a single product.
   - **Foreign Key:** product_id in **Sales Details** table references product_id in **Products** table.

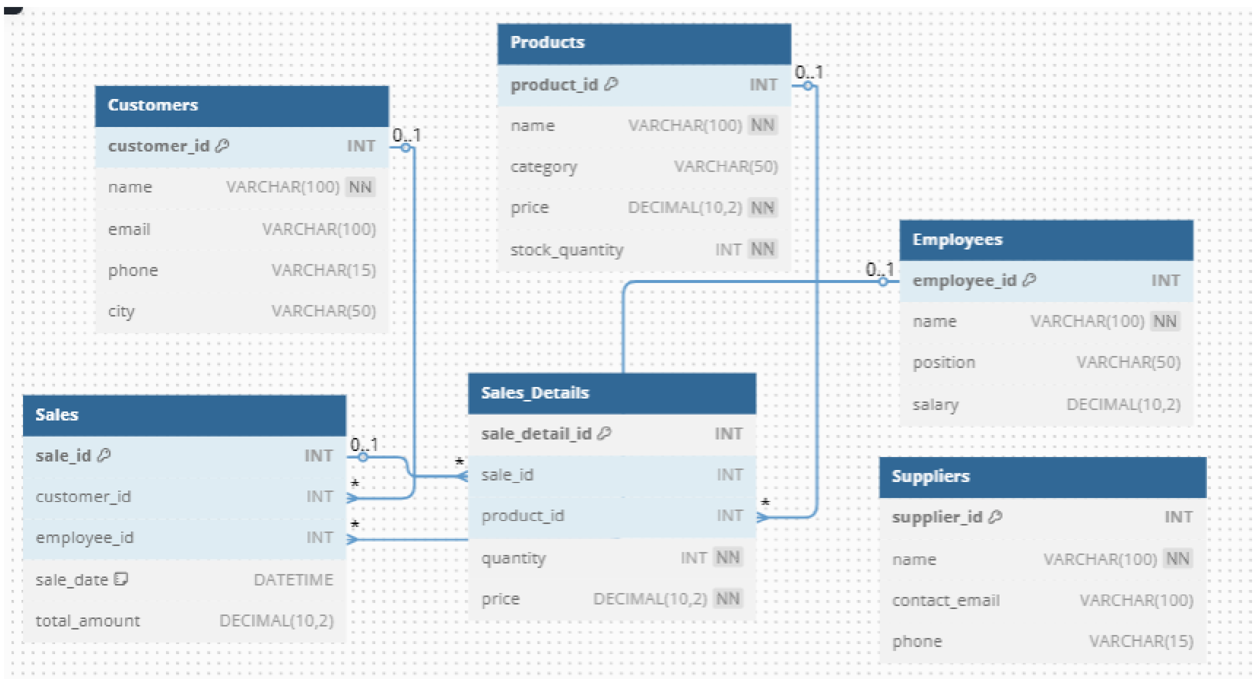
v. **Suppliers & Products (One-to-Many Relationship - Missing in Schema)**

   - Each supplier provides multiple products, but each product is supplied by only one supplier.

- **Foreign Key (Not Present in Schema):** Ideally, a supplier_id should be added in the **Products** table to track suppliers.

5. ER Diagram

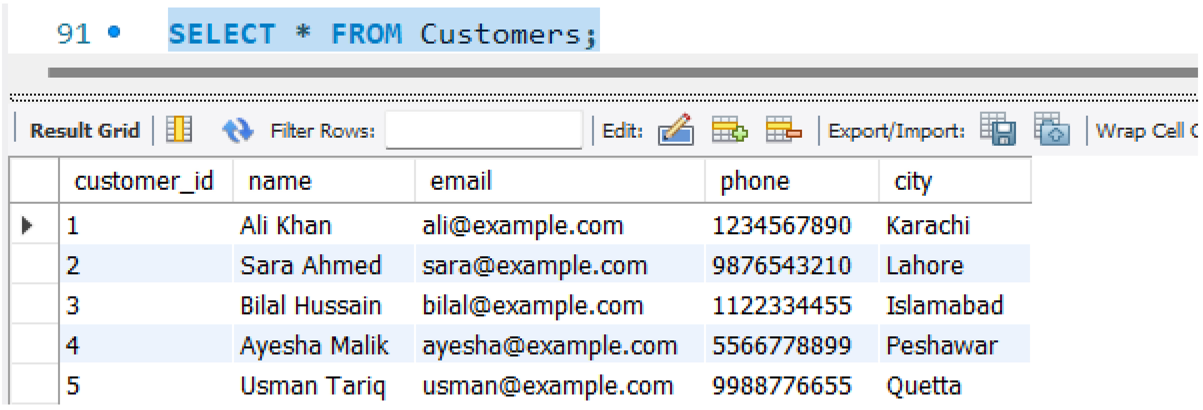




6. SQL Queries Implementation Screenshots

Step 1: CRUD Operations

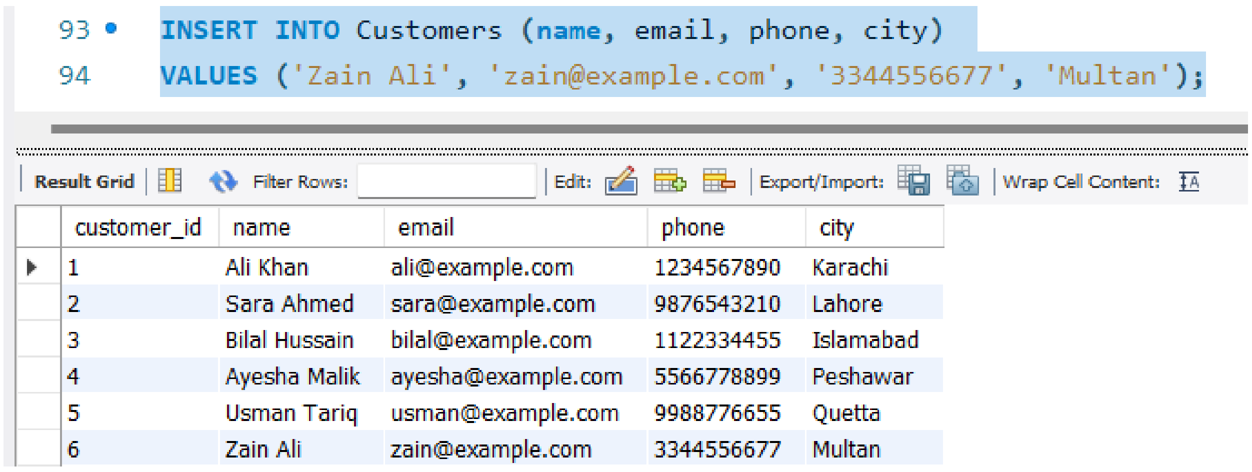
Read Customer table.



```
SELECT * FROM Customers;
```

customer_id	name	email	phone	city
1	Ali Khan	ali@example.com	1234567890	Karachi
2	Sara Ahmed	sara@example.com	9876543210	Lahore
3	Bilal Hussain	bilal@example.com	1122334455	Islamabad
4	Ayesha Malik	ayesha@example.com	5566778899	Peshawar
5	Usman Tariq	usman@example.com	9988776655	Quetta

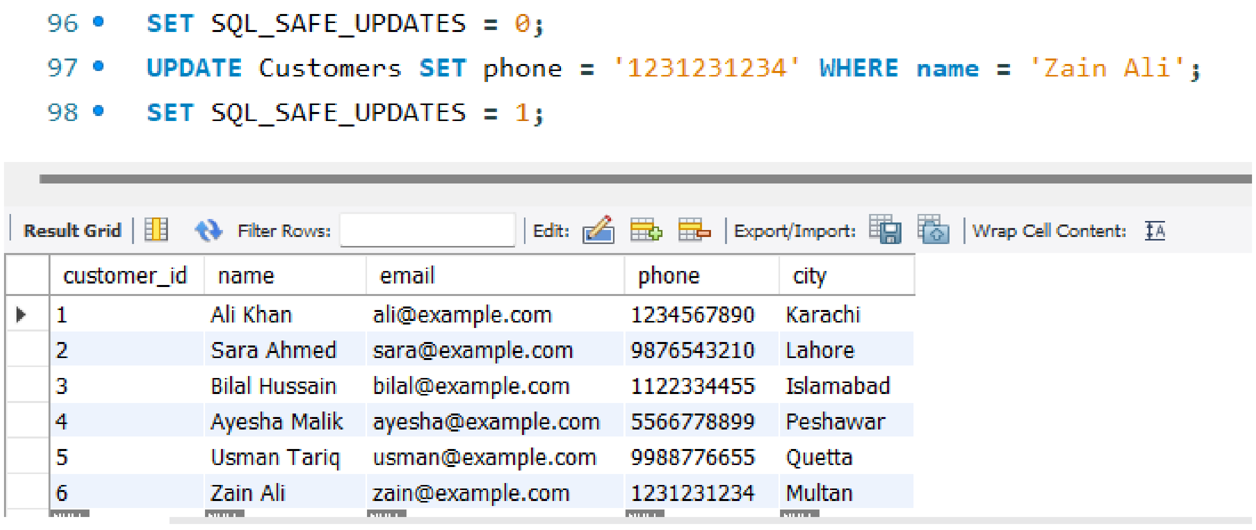
Customers Table Create (Insert)



```sql
INSERT INTO Customers (name, email, phone, city)
VALUES ('Zain Ali', 'zain@example.com', '3344556677', 'Multan');
```

customer_id	name	email	phone	city
1	Ali Khan	ali@example.com	1234567890	Karachi
2	Sara Ahmed	sara@example.com	9876543210	Lahore
3	Bilal Hussain	bilal@example.com	1122334455	Islamabad
4	Ayesha Malik	ayesha@example.com	5566778899	Peshawar
5	Usman Tariq	usman@example.com	9988776655	Quetta
6	Zain Ali	zain@example.com	3344556677	Multan

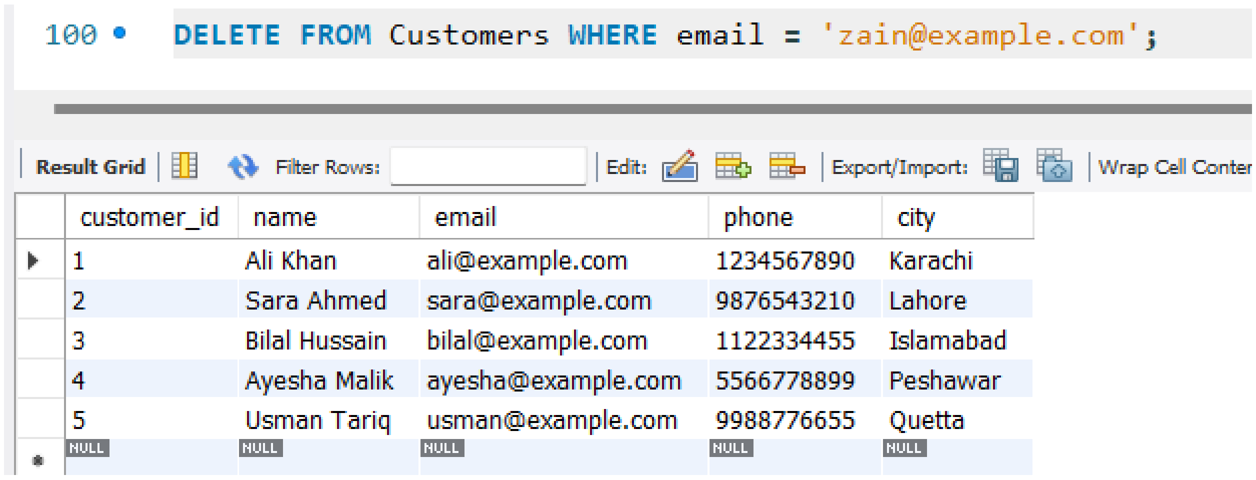
Update



```sql
SET SQL_SAFE_UPDATES = 0;
UPDATE Customers SET phone = '1231231234' WHERE name = 'Zain Ali';
SET SQL_SAFE_UPDATES = 1;
```

customer_id	name	email	phone	city
1	Ali Khan	ali@example.com	1234567890	Karachi
2	Sara Ahmed	sara@example.com	9876543210	Lahore
3	Bilal Hussain	bilal@example.com	1122334455	Islamabad
4	Ayesha Malik	ayesha@example.com	5566778899	Peshawar
5	Usman Tariq	usman@example.com	9988776655	Quetta
6	Zain Ali	zain@example.com	1231231234	Multan

Delete



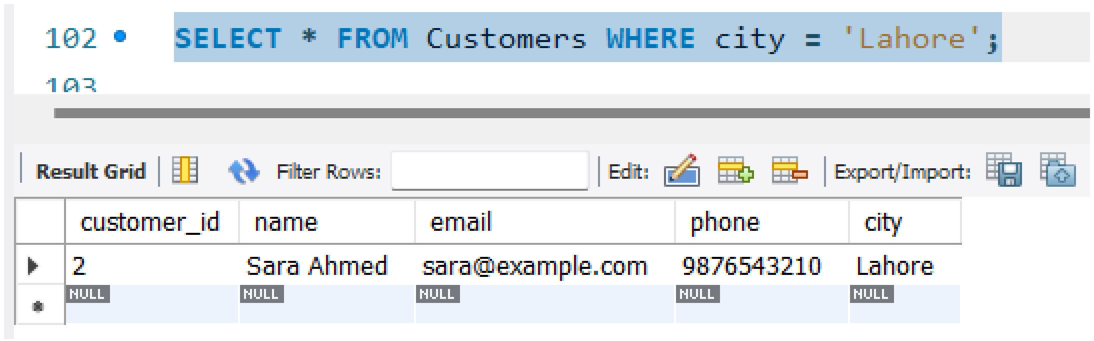
```sql
DELETE FROM Customers WHERE email = 'zain@example.com';
```

customer_id	name	email	phone	city
1	Ali Khan	ali@example.com	1234567890	Karachi
2	Sara Ahmed	sara@example.com	9876543210	Lahore
3	Bilal Hussain	bilal@example.com	1122334455	Islamabad
4	Ayesha Malik	ayesha@example.com	5566778899	Peshawar
5	Usman Tariq	usman@example.com	9988776655	Quetta
NULL	NULL	NULL	NULL	NULL

Step 2: Advanced SQL Operators

- **WHERE, LIKE, BETWEEN, IN, ORDER BY, GROUP BY, HAVING**

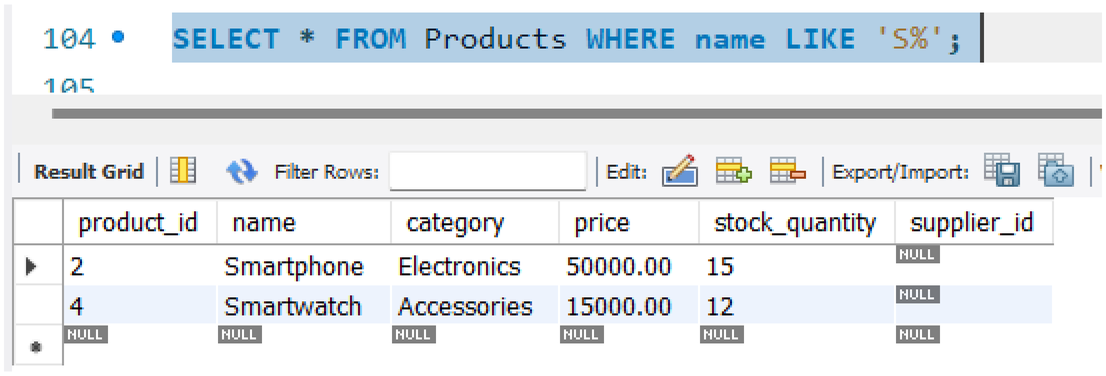
WHERE Clause



```sql
SELECT * FROM Customers WHERE city = 'Lahore';
```

customer_id	name	email	phone	city
2	Sara Ahmed	sara@example.com	9876543210	Lahore
NULL	NULL	NULL	NULL	NULL

LIKE (Names starting with S)

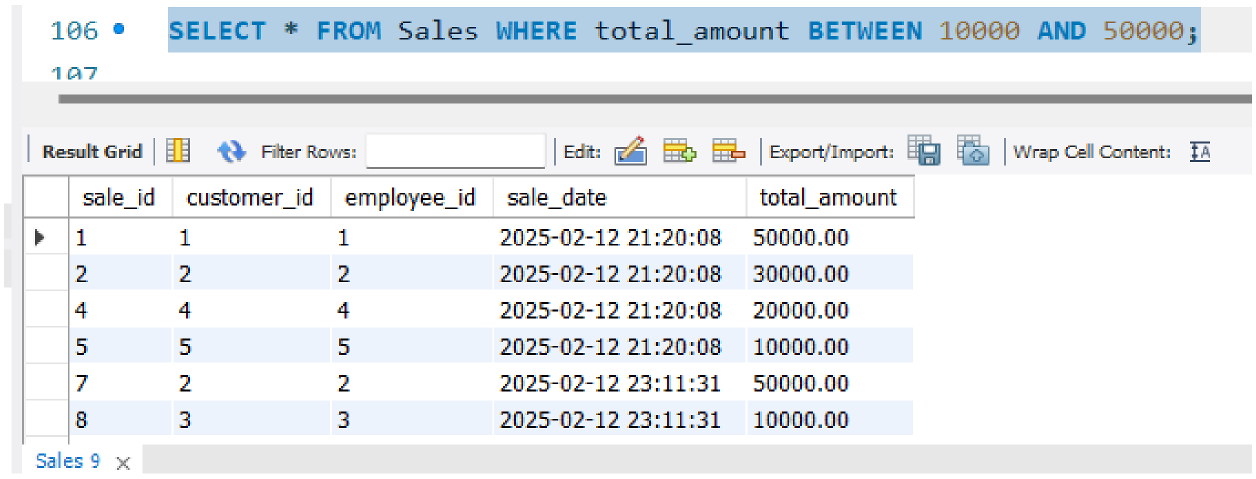


```
SELECT * FROM Products WHERE name LIKE 'S%';
```

Result Grid | Filter Rows: | Edit: | Export/Import:

	product_id	name	category	price	stock_quantity	supplier_id
▶	2	Smartphone	Electronics	50000.00	15	NULL
	4	Smartwatch	Accessories	15000.00	12	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL

BETWEEN



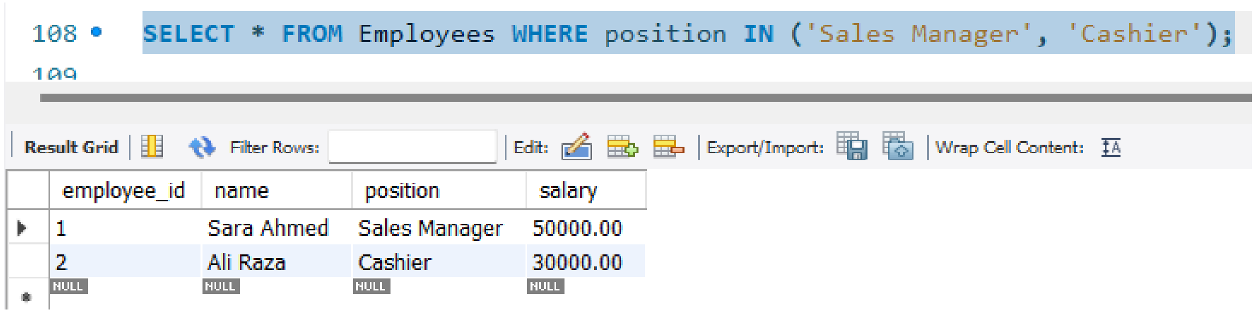
```
SELECT * FROM Sales WHERE total_amount BETWEEN 10000 AND 50000;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

	sale_id	customer_id	employee_id	sale_date	total_amount
▶	1	1	1	2025-02-12 21:20:08	50000.00
	2	2	2	2025-02-12 21:20:08	30000.00
	4	4	4	2025-02-12 21:20:08	20000.00
	5	5	5	2025-02-12 21:20:08	10000.00
	7	2	2	2025-02-12 23:11:31	50000.00
	8	3	3	2025-02-12 23:11:31	10000.00

Sales 9 ×

IN Clause

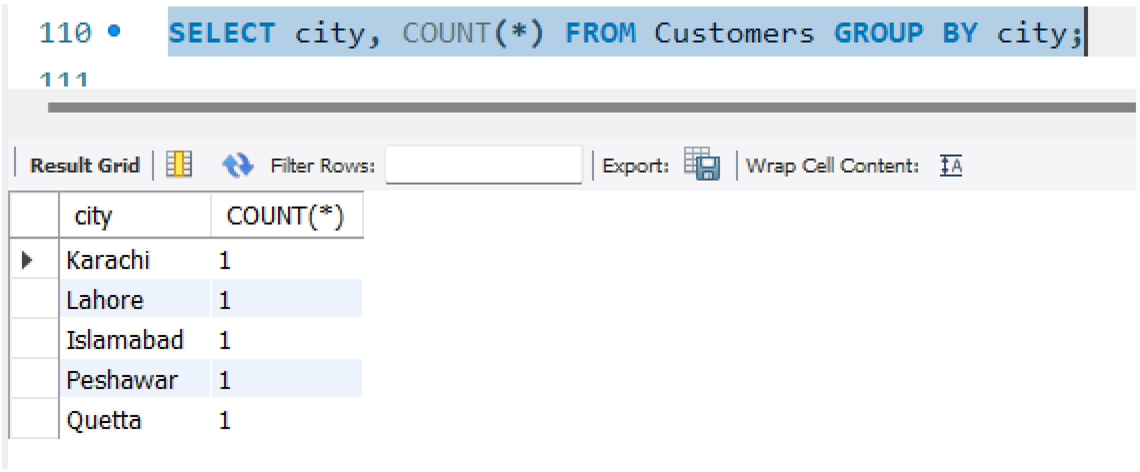


```
SELECT * FROM Employees WHERE position IN ('Sales Manager', 'Cashier');
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

	employee_id	name	position	salary
▶	1	Sara Ahmed	Sales Manager	50000.00
	2	Ali Raza	Cashier	30000.00
*	NULL	NULL	NULL	NULL

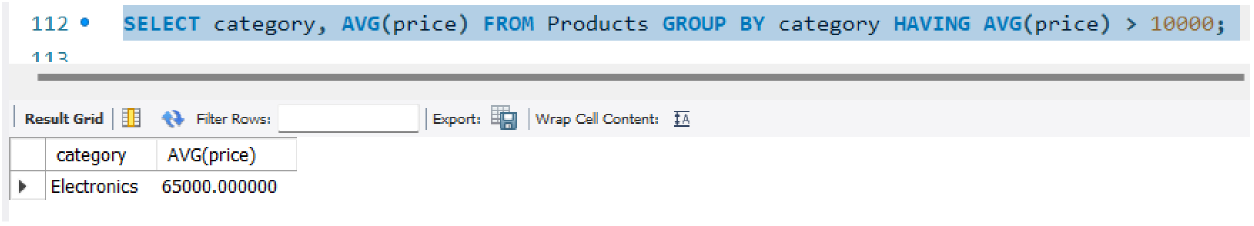
Group BY



```sql
SELECT city, COUNT(*) FROM Customers GROUP BY city;
```

city	COUNT(*)
Karachi	1
Lahore	1
Islamabad	1
Peshawar	1
Quetta	1

HAVING

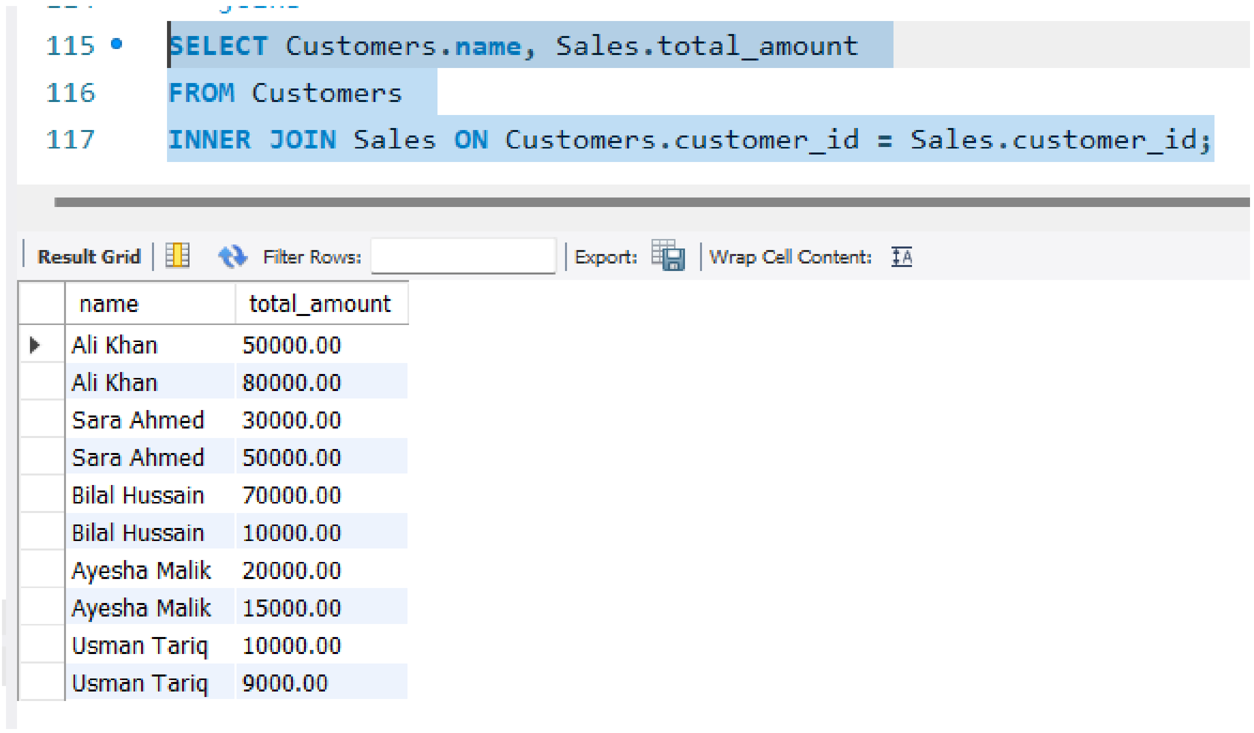


```sql
SELECT category, AVG(price) FROM Products GROUP BY category HAVING AVG(price) > 10000;
```

category	AVG(price)
Electronics	65000.000000

Step 3: Joins

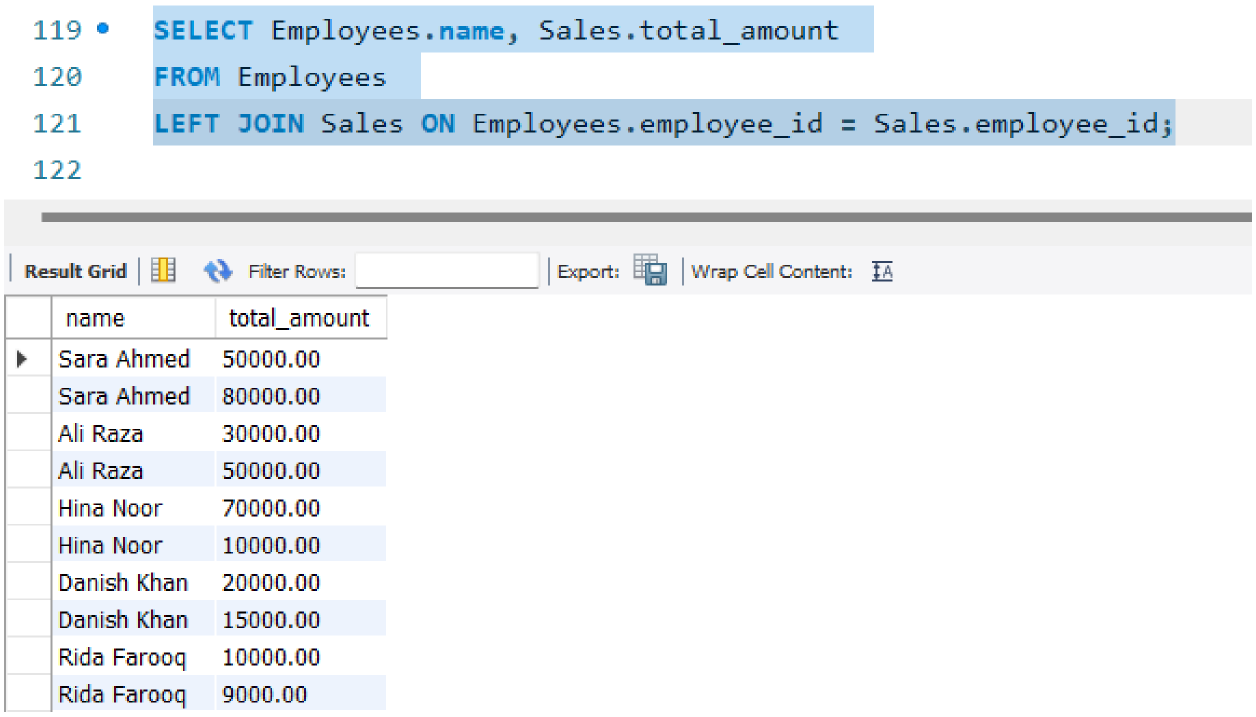
Inner Join



```sql
SELECT Customers.name, Sales.total_amount
FROM Customers
INNER JOIN Sales ON Customers.customer_id = Sales.customer_id;
```

name	total_amount
Ali Khan	50000.00
Ali Khan	80000.00
Sara Ahmed	30000.00
Sara Ahmed	50000.00
Bilal Hussain	70000.00
Bilal Hussain	10000.00
Ayesha Malik	20000.00
Ayesha Malik	15000.00
Usman Tariq	10000.00
Usman Tariq	9000.00

Left join

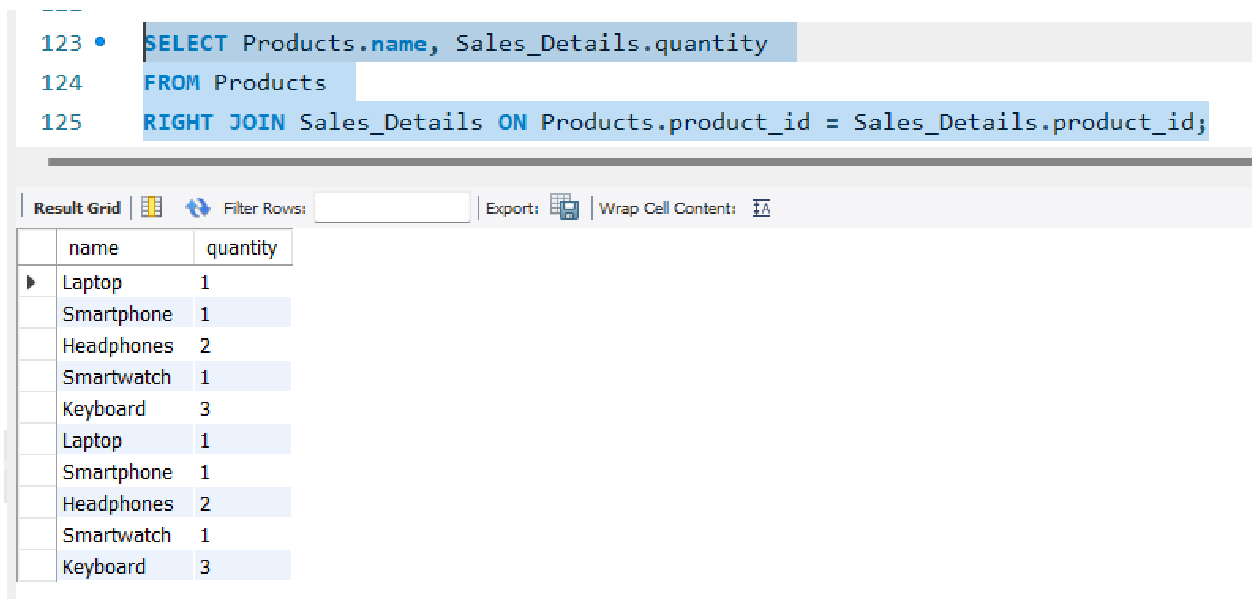


```
SELECT Employees.name, Sales.total_amount
FROM Employees
LEFT JOIN Sales ON Employees.employee_id = Sales.employee_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	total_amount
Sara Ahmed	50000.00
Sara Ahmed	80000.00
Ali Raza	30000.00
Ali Raza	50000.00
Hina Noor	70000.00
Hina Noor	10000.00
Danish Khan	20000.00
Danish Khan	15000.00
Rida Farooq	10000.00
Rida Farooq	9000.00

Right join



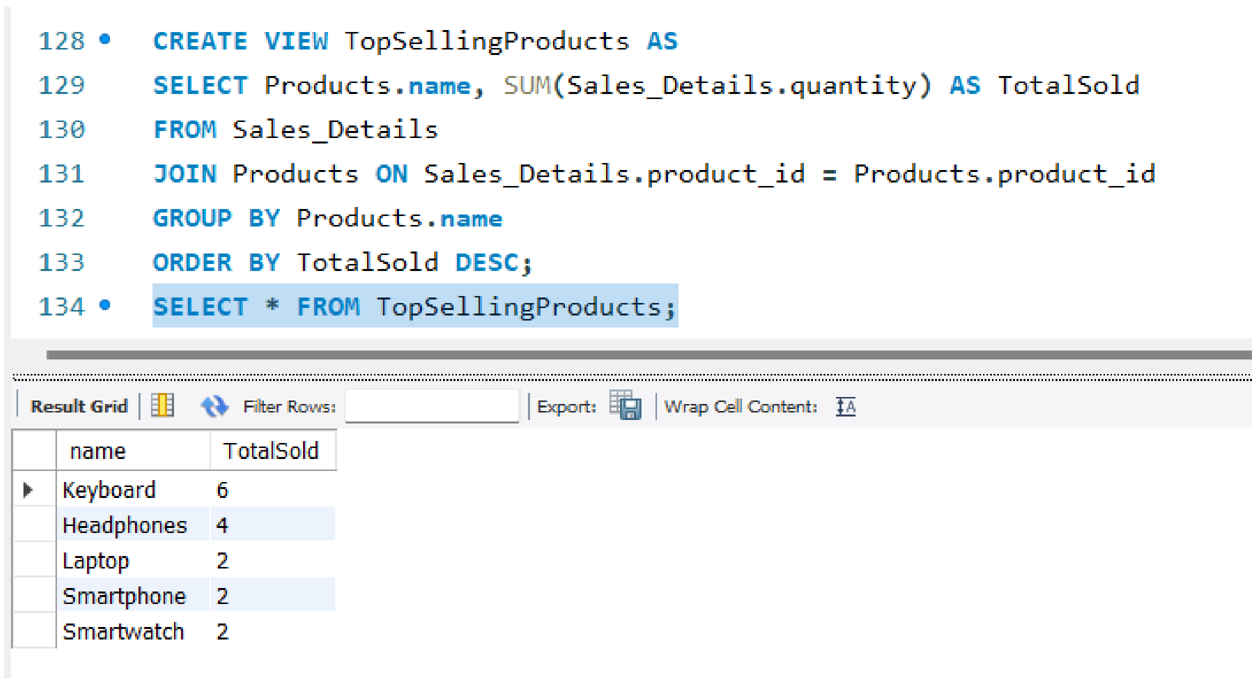
```
SELECT Products.name, Sales_Details.quantity
FROM Products
RIGHT JOIN Sales_Details ON Products.product_id = Sales_Details.product_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	quantity
Laptop	1
Smartphone	1
Headphones	2
Smartwatch	1
Keyboard	3
Laptop	1
Smartphone	1
Headphones	2
Smartwatch	1
Keyboard	3

Step 4: Views

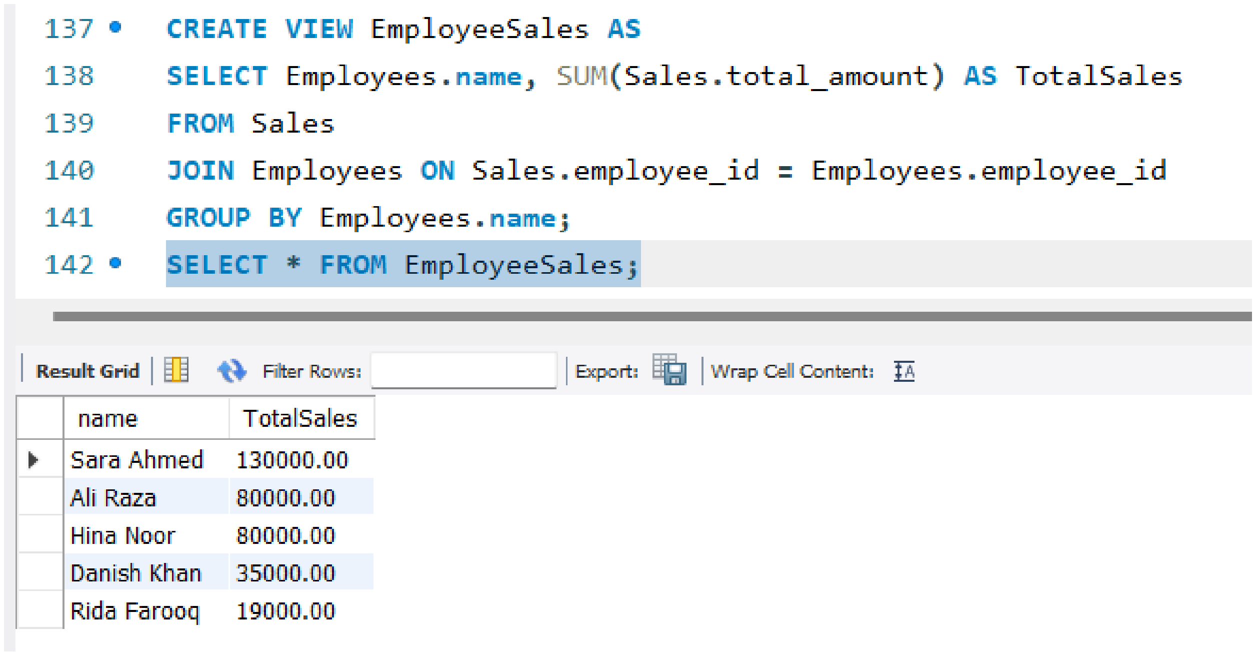
- **View 1: Top Selling Products**



```sql
CREATE VIEW TopSellingProducts AS
SELECT Products.name, SUM(Sales_Details.quantity) AS TotalSold
FROM Sales_Details
JOIN Products ON Sales_Details.product_id = Products.product_id
GROUP BY Products.name
ORDER BY TotalSold DESC;
SELECT * FROM TopSellingProducts;
```

name	TotalSold
Keyboard	6
Headphones	4
Laptop	2
Smartphone	2
Smartwatch	2

View 2: Employee Sales Report

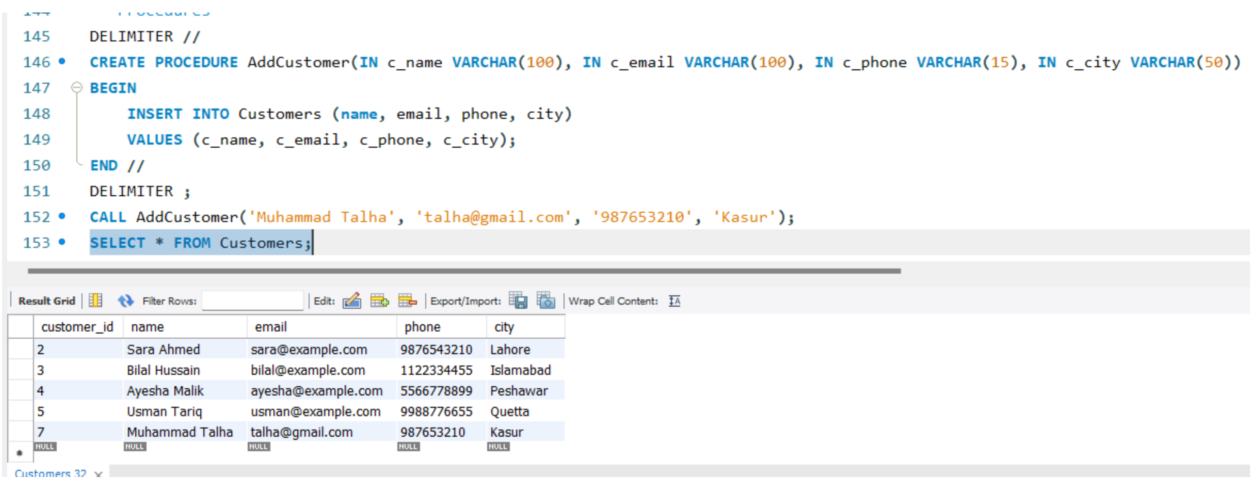


```sql
CREATE VIEW EmployeeSales AS
SELECT Employees.name, SUM(Sales.total_amount) AS TotalSales
FROM Sales
JOIN Employees ON Sales.employee_id = Employees.employee_id
GROUP BY Employees.name;
SELECT * FROM EmployeeSales;
```

name	TotalSales
Sara Ahmed	130000.00
Ali Raza	80000.00
Hina Noor	80000.00
Danish Khan	35000.00
Rida Farooq	19000.00

Step 5: Stored Procedures

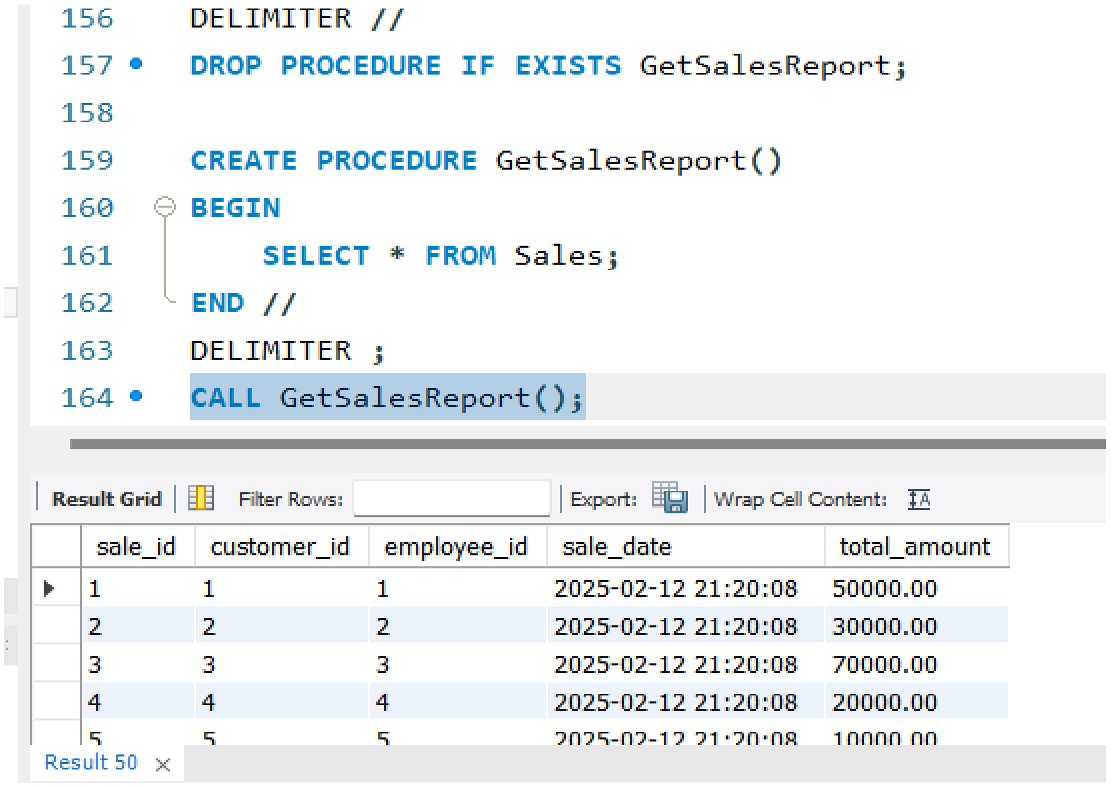
- **Procedure 1: Add New Customer**



```sql
DELIMITER //
CREATE PROCEDURE AddCustomer(IN c_name VARCHAR(100), IN c_email VARCHAR(100), IN c_phone VARCHAR(15), IN c_city VARCHAR(50))
BEGIN
    INSERT INTO Customers (name, email, phone, city)
    VALUES (c_name, c_email, c_phone, c_city);
END //
DELIMITER ;
CALL AddCustomer('Muhammad Talha', 'talha@gmail.com', '987653210', 'Kasur');
SELECT * FROM Customers;
```

customer_id	name	email	phone	city
2	Sara Ahmed	sara@example.com	9876543210	Lahore
3	Bilal Hussain	bilal@example.com	1122334455	Islamabad
4	Ayesha Malik	ayesha@example.com	5566778899	Peshawar
5	Usman Tariq	usman@example.com	9988776655	Quetta
7	Muhammad Talha	talha@gmail.com	987653210	Kasur
NULL	NULL	NULL	NULL	NULL

- **Procedure 2: Get Sales Report**

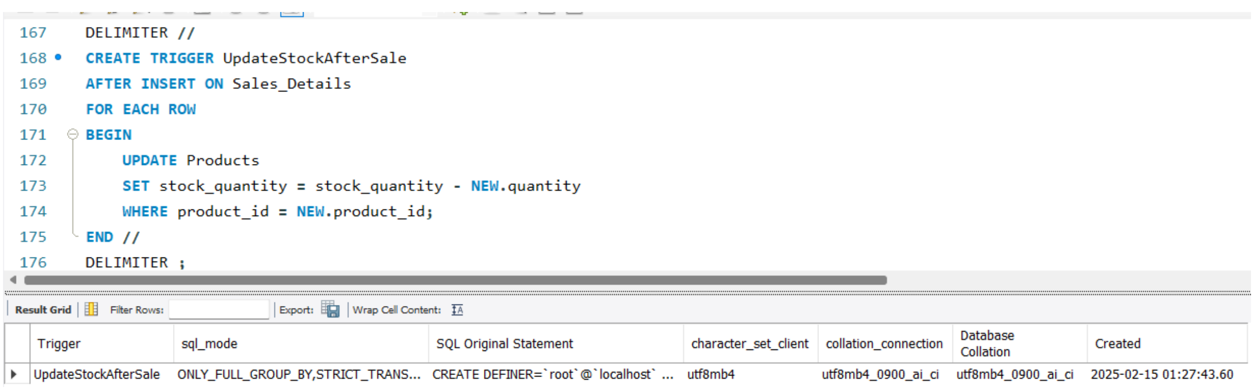


```sql
DELIMITER //
DROP PROCEDURE IF EXISTS GetSalesReport;

CREATE PROCEDURE GetSalesReport()
BEGIN
    SELECT * FROM Sales;
END //
DELIMITER ;
CALL GetSalesReport();
```

sale_id	customer_id	employee_id	sale_date	total_amount
1	1	1	2025-02-12 21:20:08	50000.00
2	2	2	2025-02-12 21:20:08	30000.00
3	3	3	2025-02-12 21:20:08	70000.00
4	4	4	2025-02-12 21:20:08	20000.00
5	5	5	2025-02-12 21:20:08	10000.00

Step 6: Triggers

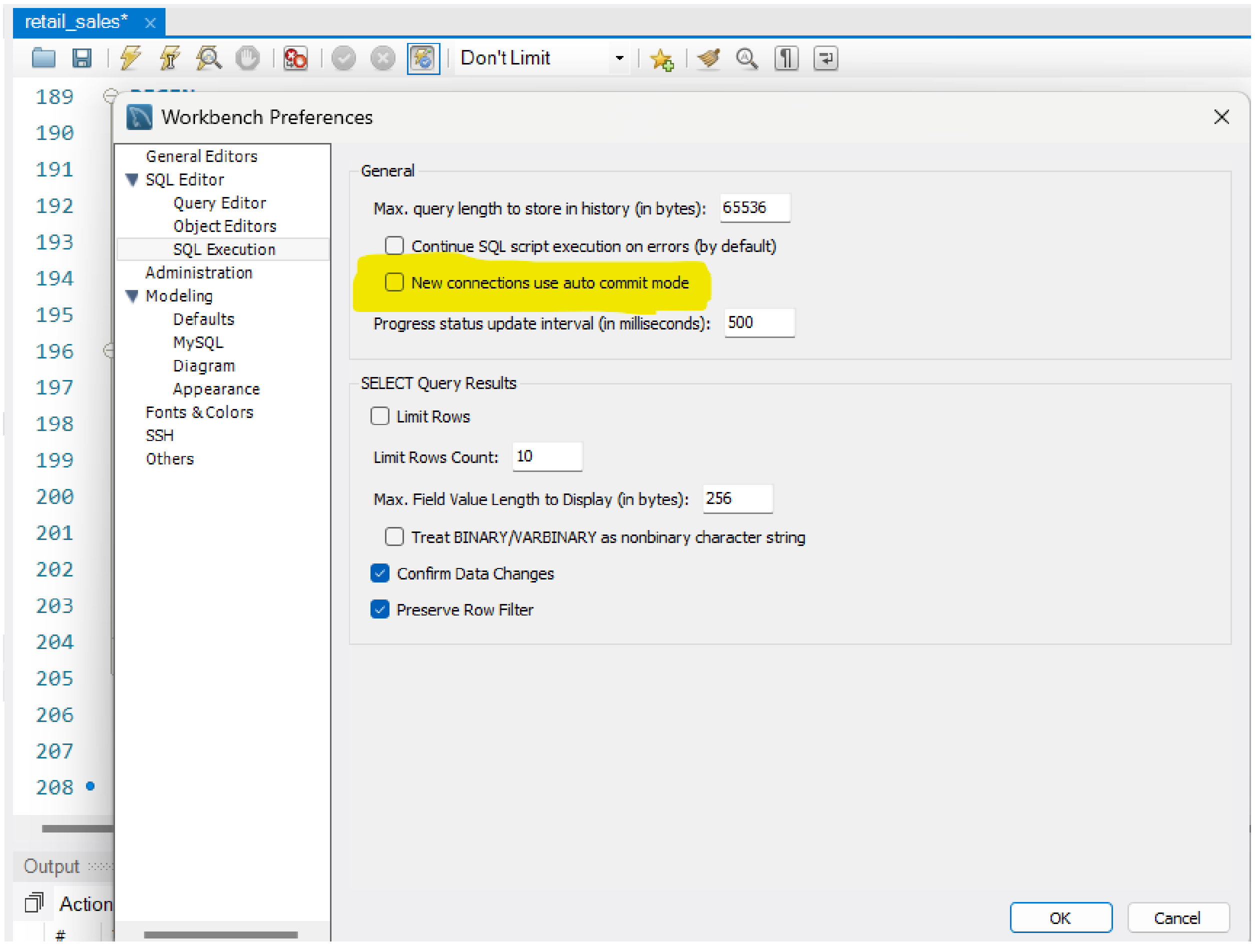
- **Trigger: Update Stock After Sale**



```
DELIMITER //
CREATE TRIGGER UpdateStockAfterSale
AFTER INSERT ON Sales_Details
FOR EACH ROW
BEGIN
    UPDATE Products
    SET stock_quantity = stock_quantity - NEW.quantity
    WHERE product_id = NEW.product_id;
END //
DELIMITER ;
```

Trigger	sql_mode	SQL Original Statement	character_set_client	collation_connection	Database Collation	Created
UpdateStockAfterSale	ONLY_FULL_GROUP_BY,STRICT_TRANS...	CREATE DEFINER=`root`@`localhost` ...	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci	2025-02-15 01:27:43.60

Step 7: COMMIT & Rollback



Workbench Preferences

General

Max. query length to store in history (in bytes): 65536

☐ Continue SQL script execution on errors (by default)

☐ New connections use auto commit mode

Progress status update interval (in milliseconds): 500

SELECT Query Results

☐ Limit Rows

Limit Rows Count: 10

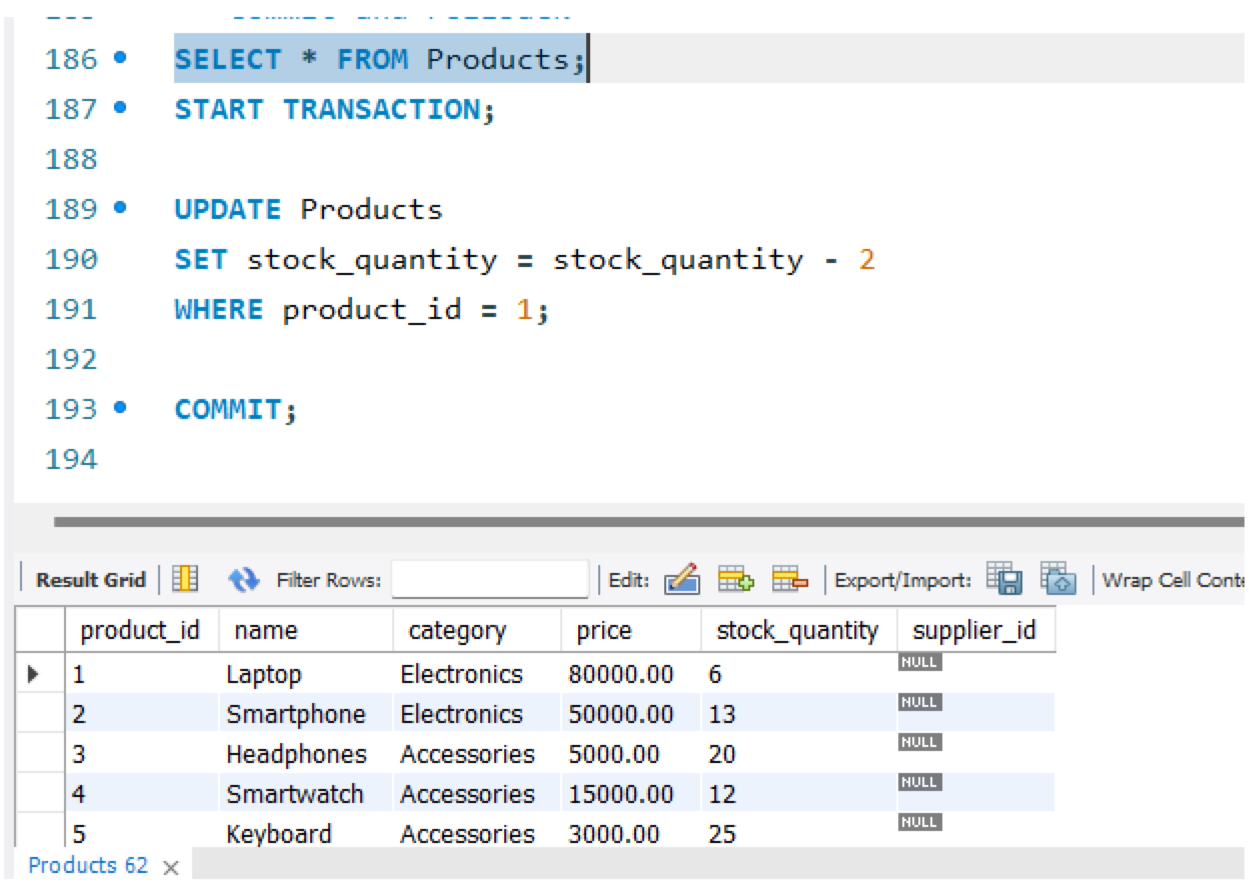
Max. Field Value Length to Display (in bytes): 256

☐ Treat BINARY/VARBINARY as nonbinary character string

☑ Confirm Data Changes

☑ Preserve Row Filter

First uncheck this for using commit and rollback command in Mysql.



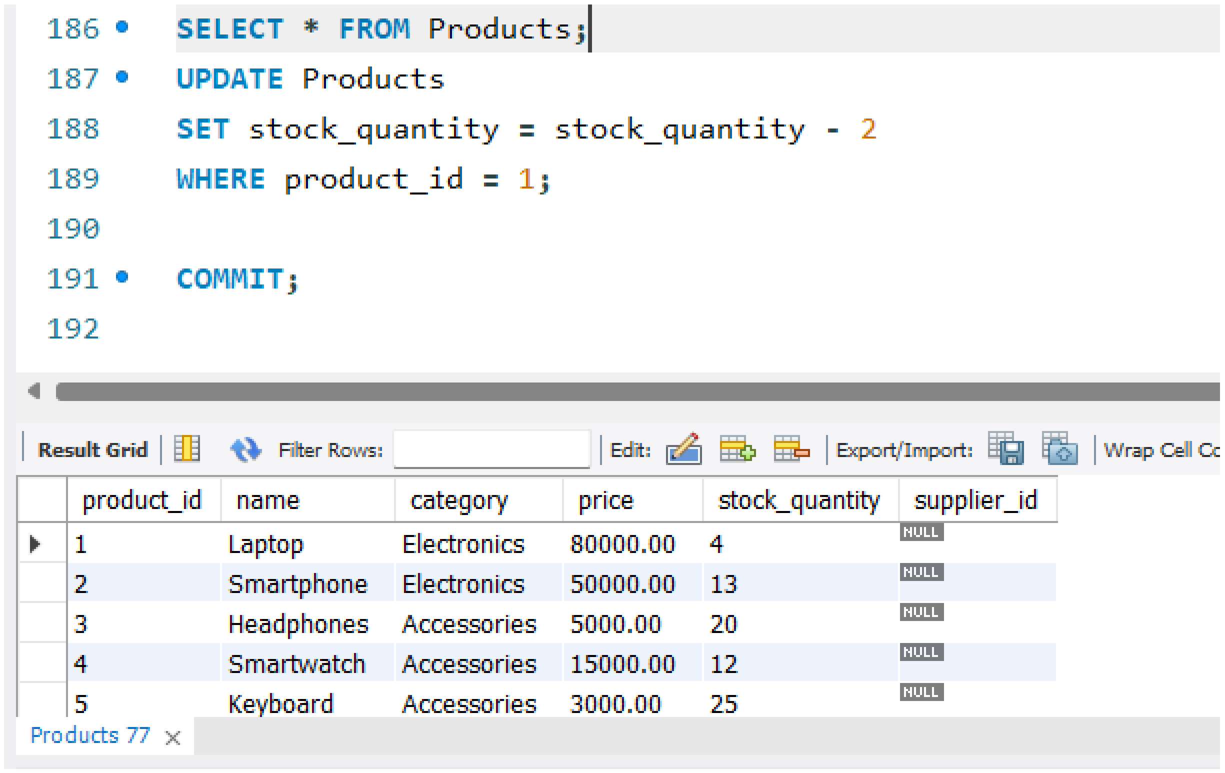
```
SELECT * FROM Products;
START TRANSACTION;

UPDATE Products
SET stock_quantity = stock_quantity - 2
WHERE product_id = 1;

COMMIT;

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conte

product_id	name	category	price	stock_quantity	supplier_id
1	Laptop	Electronics	80000.00	6	NULL
2	Smartphone	Electronics	50000.00	13	NULL
3	Headphones	Accessories	5000.00	20	NULL
4	Smartwatch	Accessories	15000.00	12	NULL
5	Keyboard	Accessories	3000.00	25	NULL

Products 62 ×

Showing all data of Products table



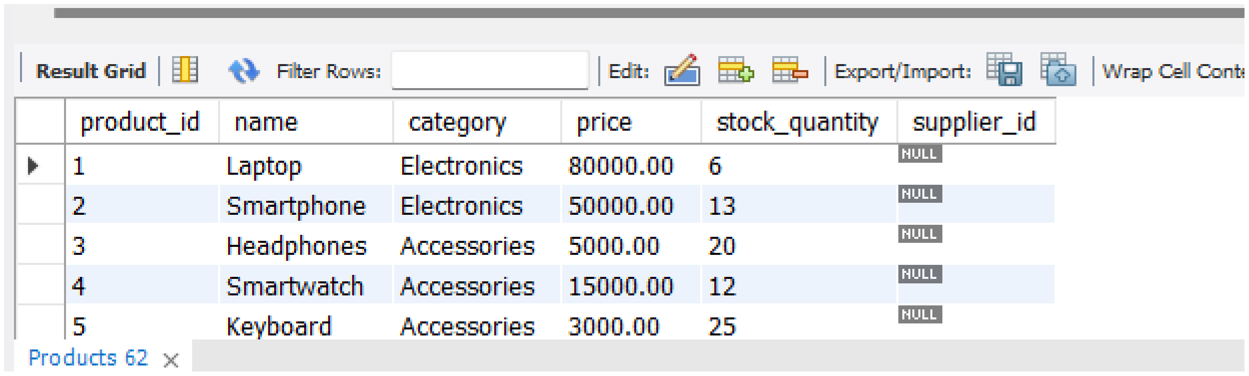
```
SELECT * FROM Products;
UPDATE Products
SET stock_quantity = stock_quantity - 2
WHERE product_id = 1;

COMMIT;

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Cc

product_id	name	category	price	stock_quantity	supplier_id
1	Laptop	Electronics	80000.00	4	NULL
2	Smartphone	Electronics	50000.00	13	NULL
3	Headphones	Accessories	5000.00	20	NULL
4	Smartwatch	Accessories	15000.00	12	NULL
5	Keyboard	Accessories	3000.00	25	NULL

Products 77 ×

After commit showing all data of product table



product_id	name	category	price	stock_quantity	supplier_id
1	Laptop	Electronics	80000.00	6	NULL
2	Smartphone	Electronics	50000.00	13	NULL
3	Headphones	Accessories	5000.00	20	NULL
4	Smartwatch	Accessories	15000.00	12	NULL
5	Keyboard	Accessories	3000.00	25	NULL

Table after Rollback (Undo table).

---

7. Challenges & Learnings

During the development of the Retail Sales Analysis database project, I faced multiple challenges and learned valuable concepts. Here are some key points:

Challenges Faced:

1. **Understanding Database Design:**
   - Initially, designing tables with proper relationships was confusing.
   - Understanding Primary Keys, Foreign Keys, and Normalization took some time.
2. **Normalization:**
   - At first, the data had redundancy, making queries slow.
   - Applying 1NF, 2NF, and 3NF helped remove duplicate data and improve efficiency.
3. **Writing Complex SQL Queries:**
   - Joining multiple tables required learning JOIN operations.

   - Aggregate functions like SUM, COUNT, and GROUP BY were tricky to use correctly.

4. **Stored Procedures & Triggers:**
   - Creating stored procedures to update stock automatically was a challenge.
   - Triggers were used to handle automatic updates, but debugging errors was difficult.

5. **Transaction Handling (Commit & Rollback):**
   - Ensuring data consistency using COMMIT and ROLLBACK in case of failures.

6. **Performance Optimization:**
   - Queries took time to execute due to large datasets.
   - Creating Indexes and using optimized queries improved performance.

Learnings & Improvements:

1. **Database Normalization Improves Data Quality:**
   - Breaking tables into smaller ones reduced redundancy and ensured data integrity.

2. **Joins & Indexing are Important for Speed:**
   - Using INNER JOIN, LEFT JOIN, and indexing made queries faster and more efficient.

3. **Stored Procedures Help in Automation:**
   - Writing reusable stored procedures saved time in executing repetitive queries.

4. **Error Handling in SQL is Necessary:**

    - Using ROLLBACK to undo unwanted transactions prevented data loss.

5. **Practical Experience is Key:**
    - Theoretical knowledge is useful, but hands-on practice helped in understanding real-world scenarios.

---

8. Conclusion

This project helped me understand **database design, normalization, and query optimization**. Initially, structuring tables and managing relationships was challenging, but **applying 1NF, 2NF, and 3NF** improved data integrity. Learning **joins, stored procedures, triggers, and transactions (COMMIT & ROLLBACK)** made the system more efficient and automated key tasks. Query optimization using **indexes and aggregate functions** improved performance. Overall, this project provided valuable **hands-on experience** in real-world database management.