**The University Of Azad Jammu & Kashmir,**

**Muzaffarabad**

**Department of Software Engineering**

**LAB TASK 07**

**Database Systems**

**Course Code**: **CS-2204**

**Submitted To:**

Engr. Tahir Jahangir

**Submitted By:**

Ahmed Ali

**Roll No:**

2023-SE-41

**Session**: 2023-2027

Contents

[Lab Task – 7: SQL Server – Working with JOINS 5](#_Toc206755502)

[🎯 Objective: 5](#_Toc206755503)

[📘 Task 01: Table Creation and Relationship Design 5](#_Toc206755504)

[Instructions: 5](#_Toc206755505)

[📝 Task 02: Data Insertion for All Tables 8](#_Toc206755506)

[Instructions: 8](#_Toc206755507)

[🔍 Task 03: Understanding JOINs through Scenario-Based Queries 13](#_Toc206755508)

[Instructions: 13](#_Toc206755509)

[📗 Task 04: Complex Queries with Multiple JOINs 16](#_Toc206755510)

[Instructions: 16](#_Toc206755511)

[**Why We Need Another Table** 17](#_Toc206755512)

[**Purpose of the Link Table (ORDER\_DETAILS)** 17](#_Toc206755513)

[**Reflection** 20](#_Toc206755514)

[**Challenges Faced** 20](#_Toc206755515)

# Lab Task – 7: SQL Server – Working with JOINS

# 🎯 Objective:

In this lab, students will work with SQL Server to explore **JOIN operations** between multiple tables. Students will begin by creating and populating sample tables and then apply different types of JOINs to solve real-life query scenarios. This task enhances understanding of data relationships and retrieval using JOINs in SQL Server.

# 📘 Task 01: Table Creation and Relationship Design

**Objective:** Set up normalized tables with primary and foreign key relationships.

# Instructions:

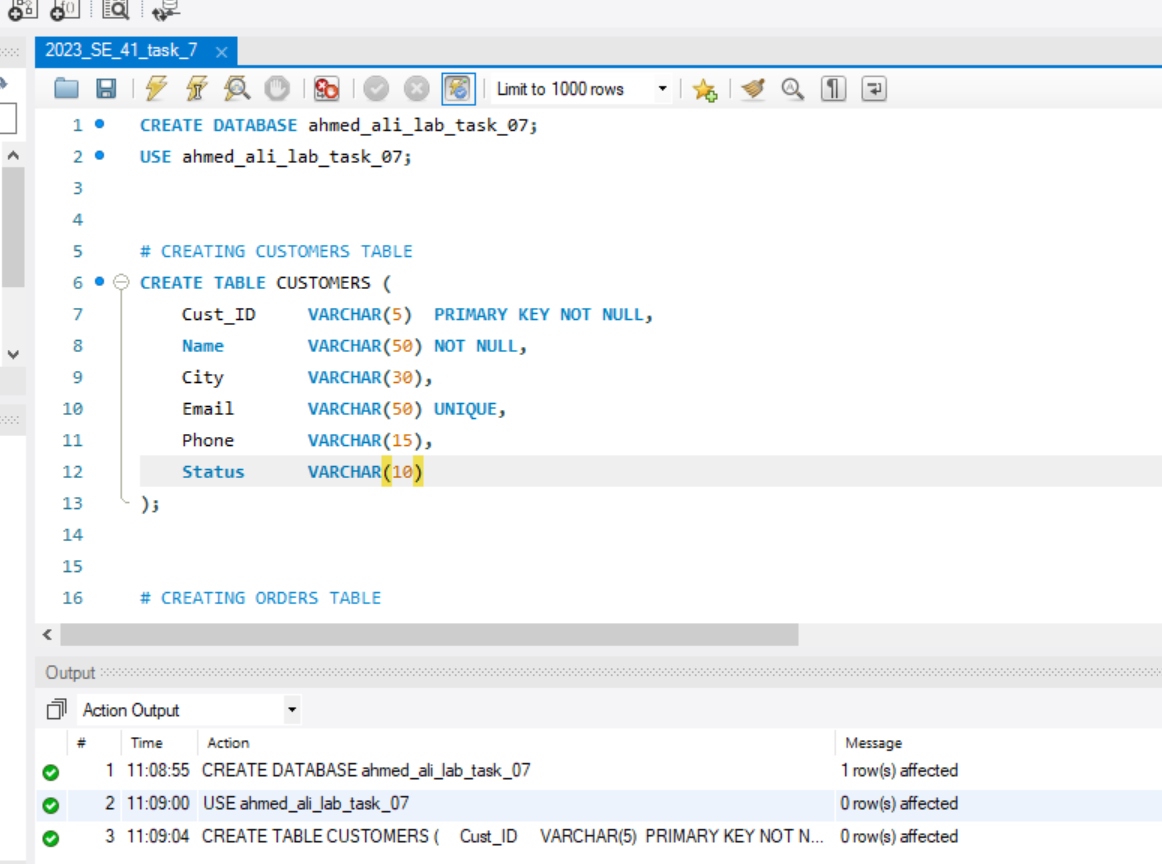
1. Create a new database named in the following format:

**your\_full\_name\_lab\_task\_07** *Example: zain\_ali\_lab\_task\_07*

1. Create the following **three related tables** with **appropriate constraints**:

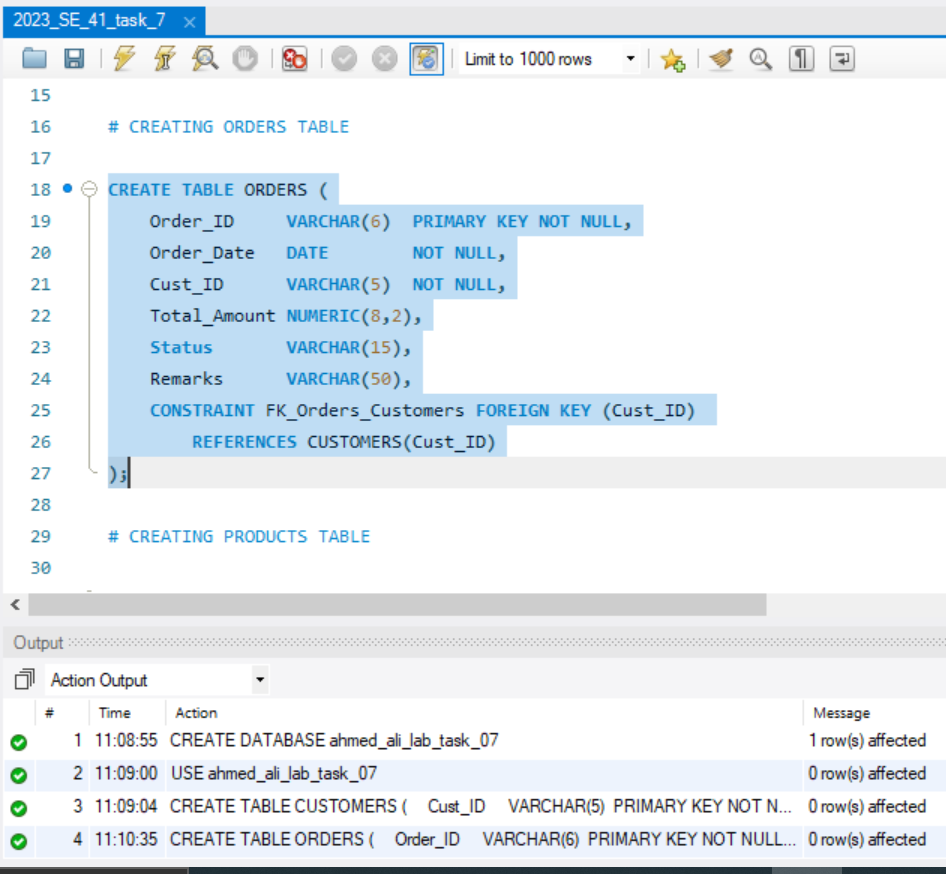
**CUSTOMERS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Size** | **Constraints** |
| Cust\_ID | VARCHAR | 5 | PRIMARY KEY, NOT NULL |
| Name | VARCHAR | 50 | NOT NULL |
| City | VARCHAR | 30 | — |
| Email | VARCHAR | 50 | UNIQUE |
| Phone | VARCHAR | 15 | — |
| Status | VARCHAR | 10 | — |



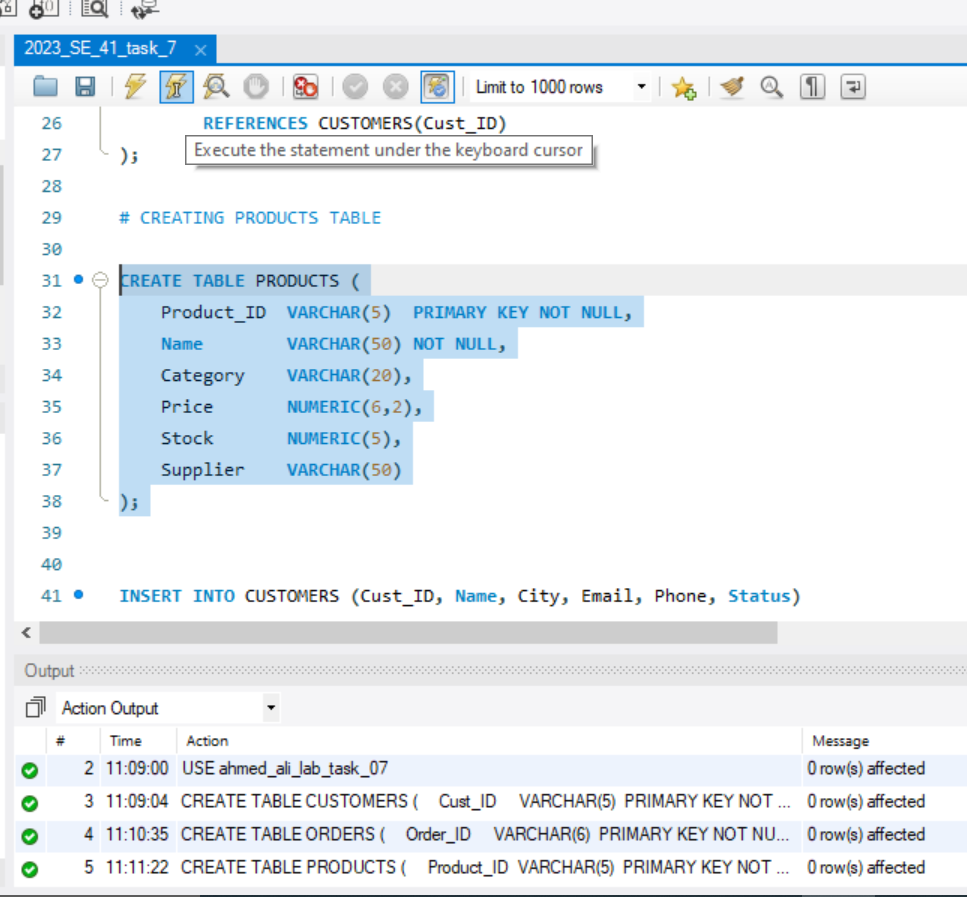
**ORDERS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Size** | **Constraints** |
| Order\_ID | VARCHAR | 6 | PRIMARY KEY, NOT NULL |
| Order\_Date | DATE | — | NOT NULL |
| Cust\_ID | VARCHAR | 5 | FOREIGN KEY REFERENCES CUSTOMERS(Cust\_ID) |
| Total\_Amount | NUMERIC | 8,2 | — |
| Status | VARCHAR | 15 | — |
| Remarks | VARCHAR | 50 | — |

****

**PRODUCTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Size** | **Constraints** |
| Product\_ID | VARCHAR | 5 | PRIMARY KEY, NOT NULL |
| Name | VARCHAR | 50 | NOT NULL |
| Category | VARCHAR | 20 | — |
| Price | NUMERIC | 6,2 | — |
| Stock | NUMERIC | 5 | — |
| Supplier | VARCHAR | 50 | — |



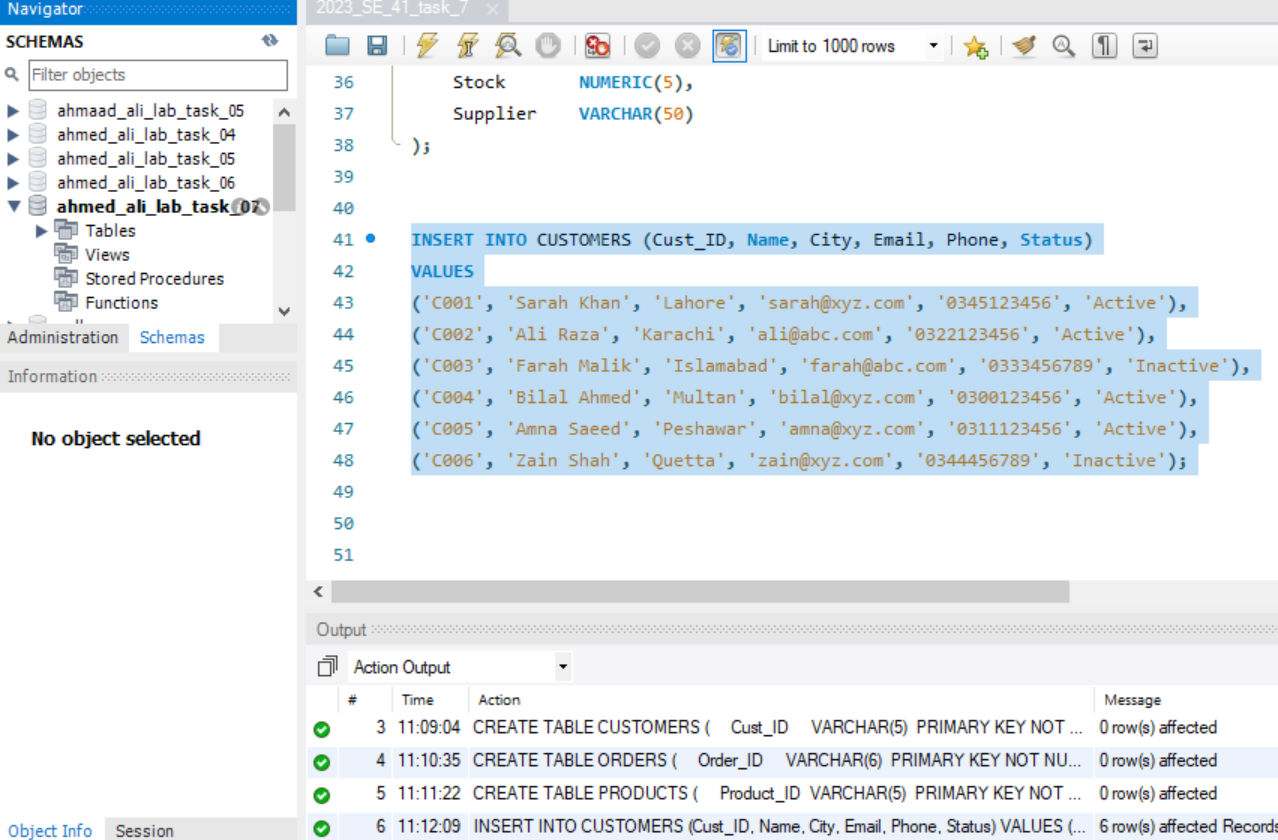
# 📝 Task 02: Data Insertion for All Tables

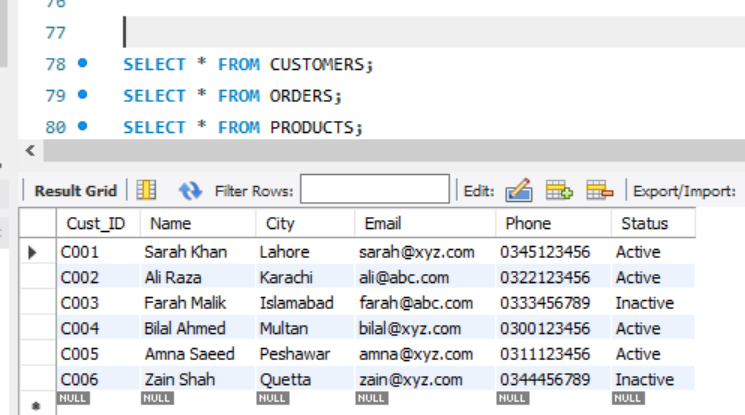
**Objective:** Insert a minimum of 6 sample records into each table.

# Instructions:

**Insert into CUSTOMERS**

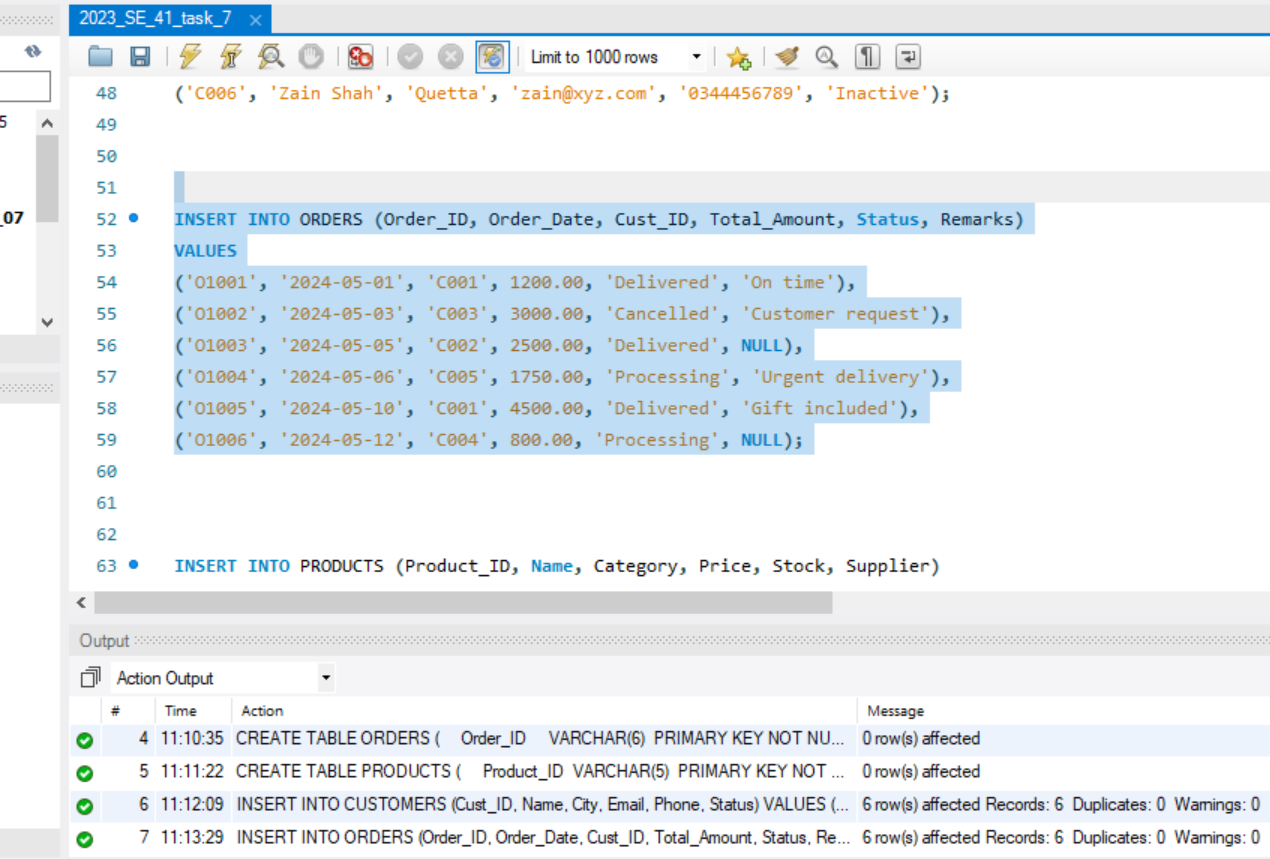
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Cust\_ID** | **Name** | **City** | **Email** | **Phone** | **Status** |
| C001 | Sarah Khan | Lahore | sarah@xyz.com | 0345123456 | Active |
| C002 | Ali Raza | Karachi | ali@abc.com | 0322123456 | Active |
| C003 | Farah Malik | Islamabad | farah@abc.com | 0333456789 | Inactive |
| C004 | Bilal Ahmed | Multan | bilal@xyz.com | 0300123456 | Active |
| C005 | Amna Saeed | Peshawar | amna@xyz.com | 0311123456 | Active |
| C006 | Zain Shah | Quetta | zain@xyz.com | 0344456789 | Inactive |

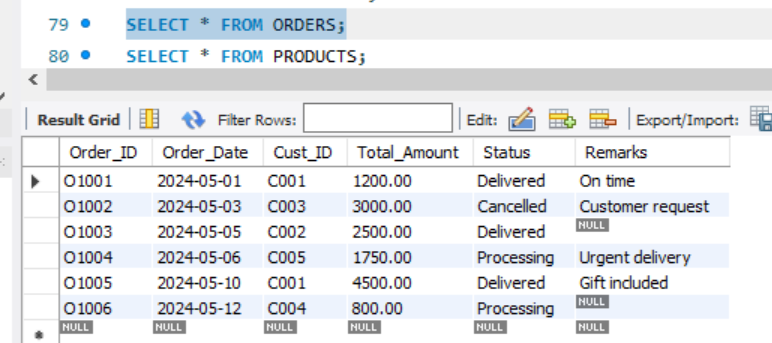
****

****

**Insert into ORDERS**

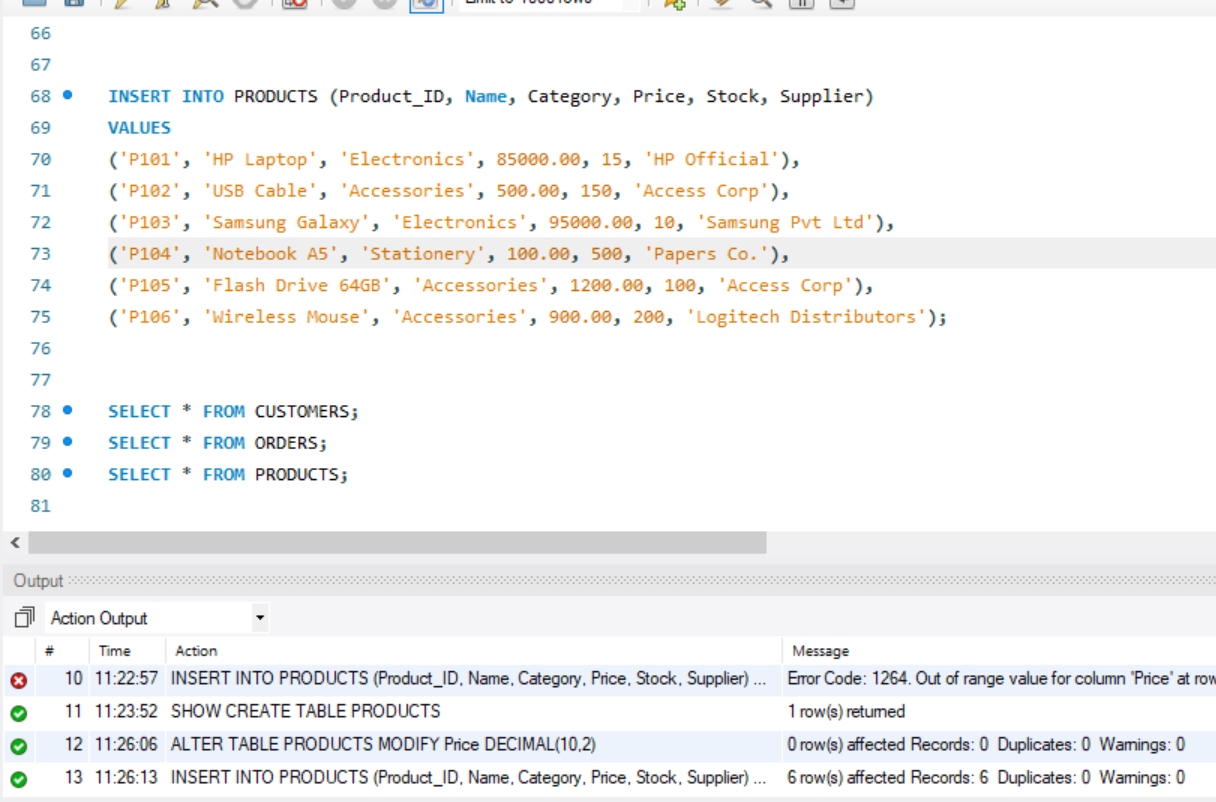
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Order\_ID** | **Order\_Date** | **Cust\_ID** | **Total\_Amount** | **Status** | **Remarks** |
| O1001 | 2024-05-01 | C001 | 1200.00 | Delivered | On time |
| O1002 | 2024-05-03 | C003 | 3000.00 | Cancelled | Customer request |
| O1003 | 2024-05-05 | C002 | 2500.00 | Delivered | — |
| O1004 | 2024-05-06 | C005 | 1750.00 | Processing | Urgent delivery |
| O1005 | 2024-05-10 | C001 | 4500.00 | Delivered | Gift included |
| O1006 | 2024-05-12 | C004 | 800.00 | Processing | — |

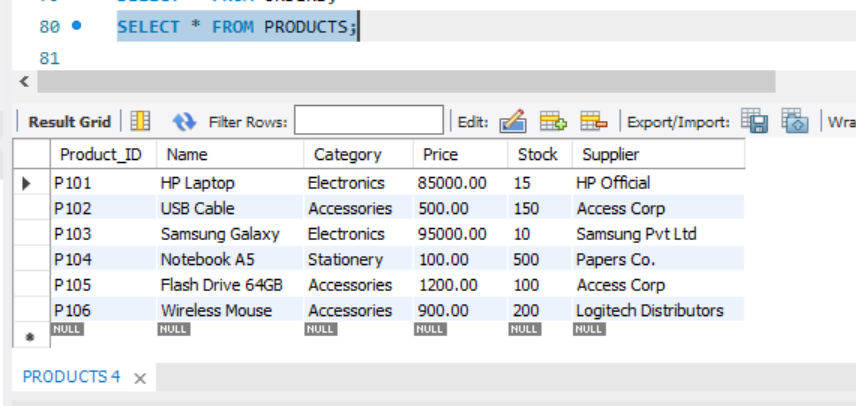
****

****

**Insert into PRODUCTS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Product\_ID** | **Name** | **Category** | **Price** | **Stock** | **Supplier** |
| P101 | HP Laptop | Electronics | 85000.00 | 15 | HP Official |
| P102 | USB Cable | Accessories | 500.00 | 150 | Access Corp |
| P103 | Samsung Galaxy | Electronics | 95000.00 | 10 | Samsung Pvt Ltd |
| P104 | Notebook A5 | Stationery | 100.00 | 500 | Papers Co. |
| P105 | Flash Drive 64GB | Accessories | 1200.00 | 100 | Access Corp |
| P106 | Wireless Mouse | Accessories | 900.00 | 200 | Logitech Distributors |





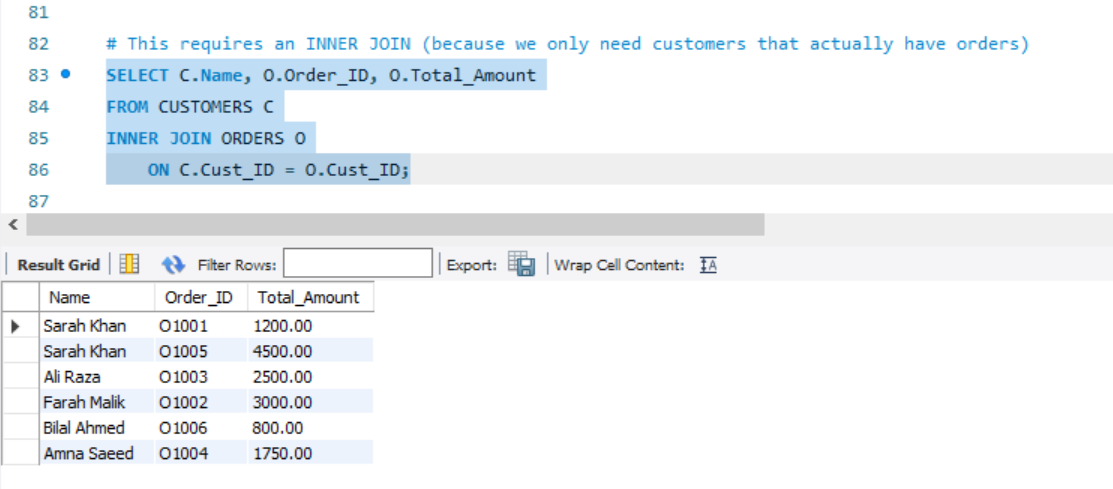
# 🔍 Task 03: Understanding JOINs through Scenario-Based Queries

**Objective:** Analyze the queries and identify the correct type of JOIN required.

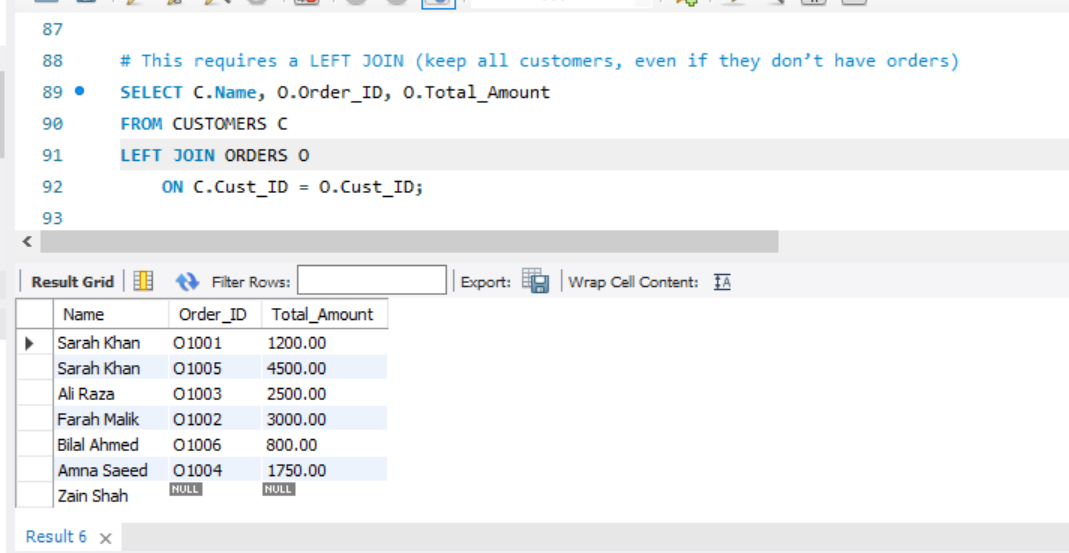
# Instructions:

Read each query and determine which type of JOIN (INNER, LEFT, RIGHT, FULL) best fits the scenario.

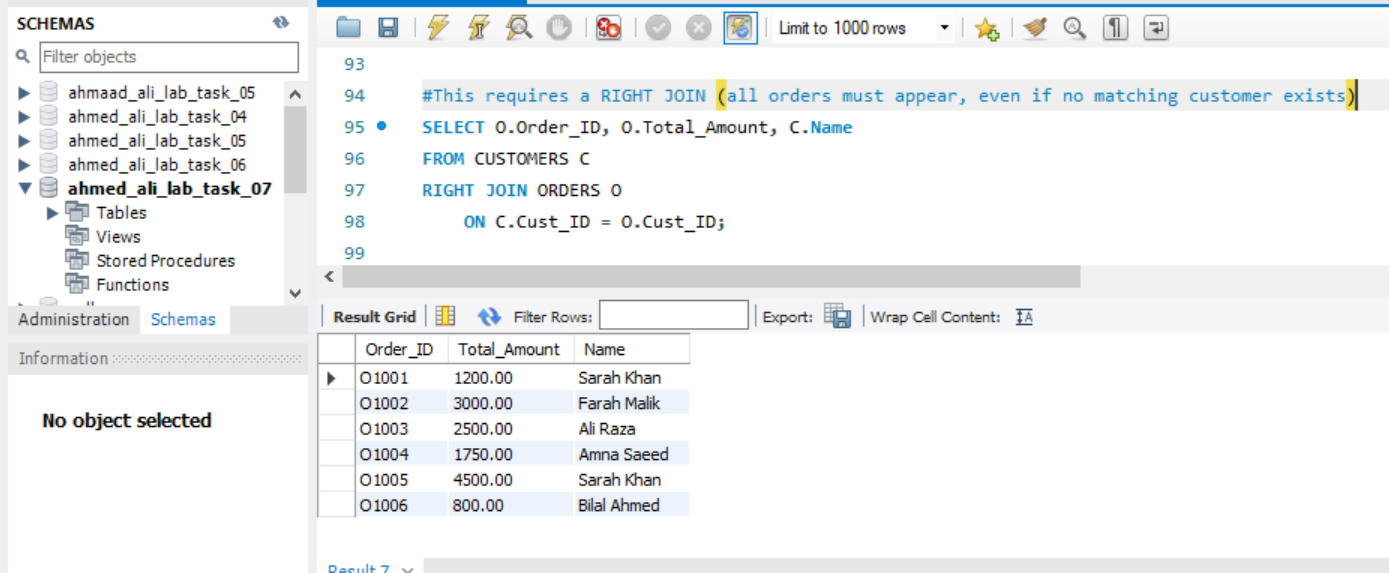
1. Display customer names and their order IDs and total amount. (Only include customers who placed orders.)



1. Show all customers and their orders, including those who haven’t placed any orders yet.

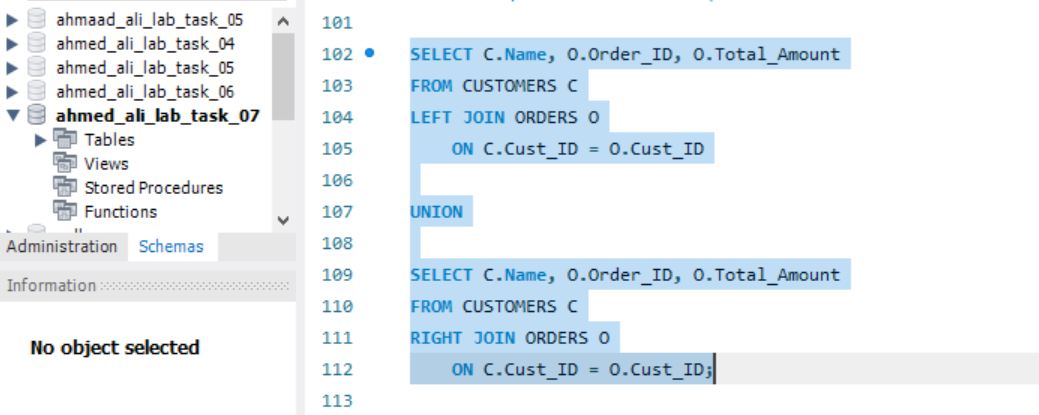


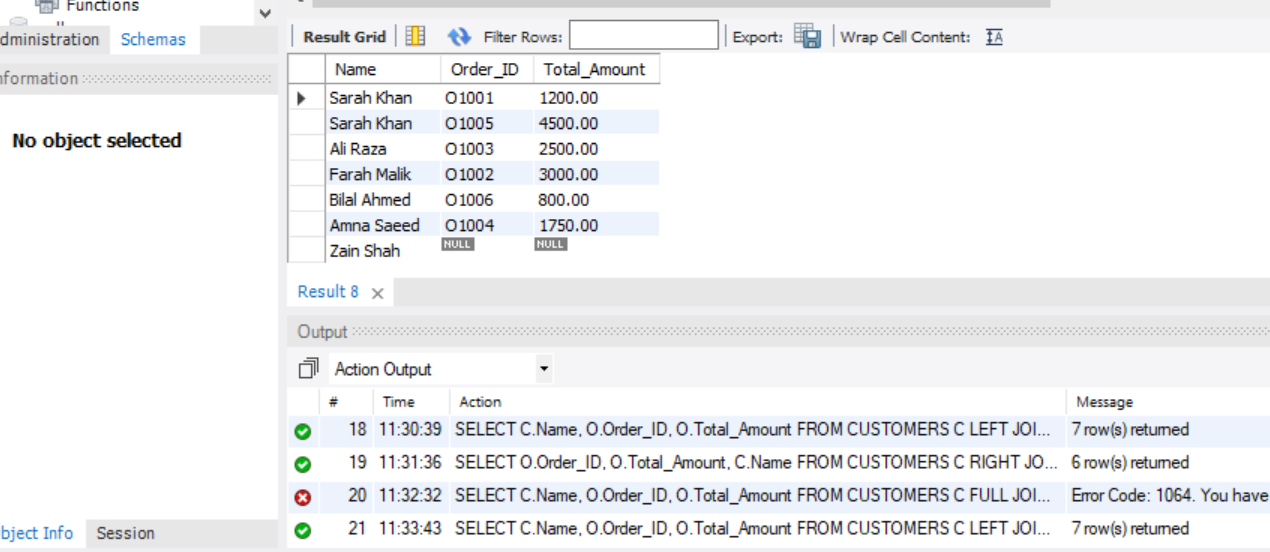
1. List all orders and corresponding customer names. If an order is linked to a non-existent customer, still show the order.



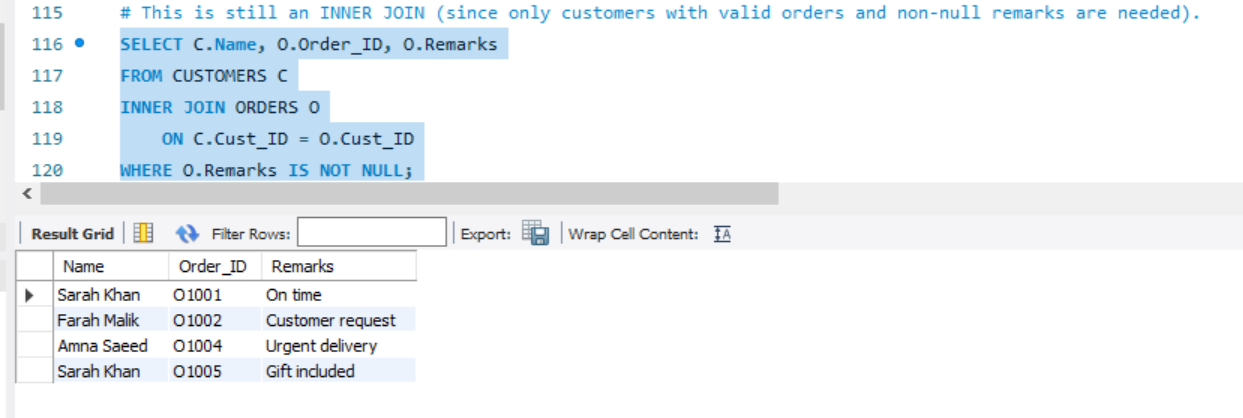
1. Display a list of all customers and orders, whether or not the order has a matching customer or the customer has placed an order.

**SQL Workbench does not support Full Joins so it is union of Left and Right join which is Actually the full Join**

****



1. Display customer name, order ID, and remarks for only those whose remarks column is not null.

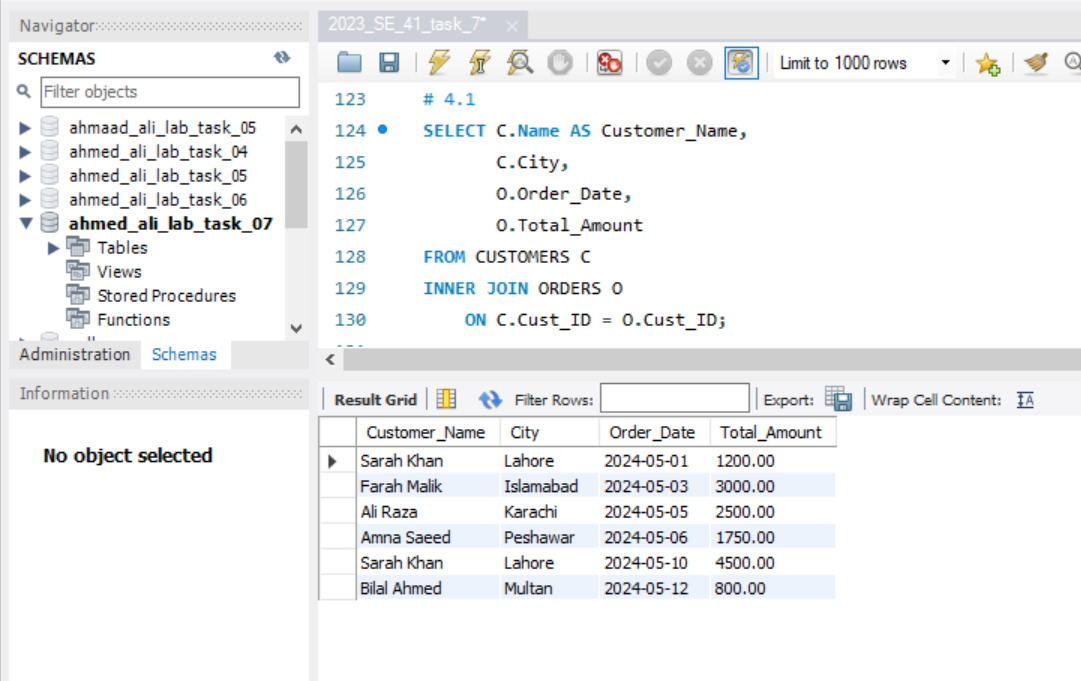


# 📗 Task 04: Complex Queries with Multiple JOINs

**Objective:** Apply multi-table JOINs to retrieve integrated data.

# Instructions:

1. Retrieve customer name, city, order date, and total amount for all orders placed.



1. Show product name, supplier, and all customers who ordered it. (Assume a separate link table is created if needed – for simplicity, you can explain this part theoretically.)

**Query Statement:**  
Show product name, supplier, and all customers who ordered it.

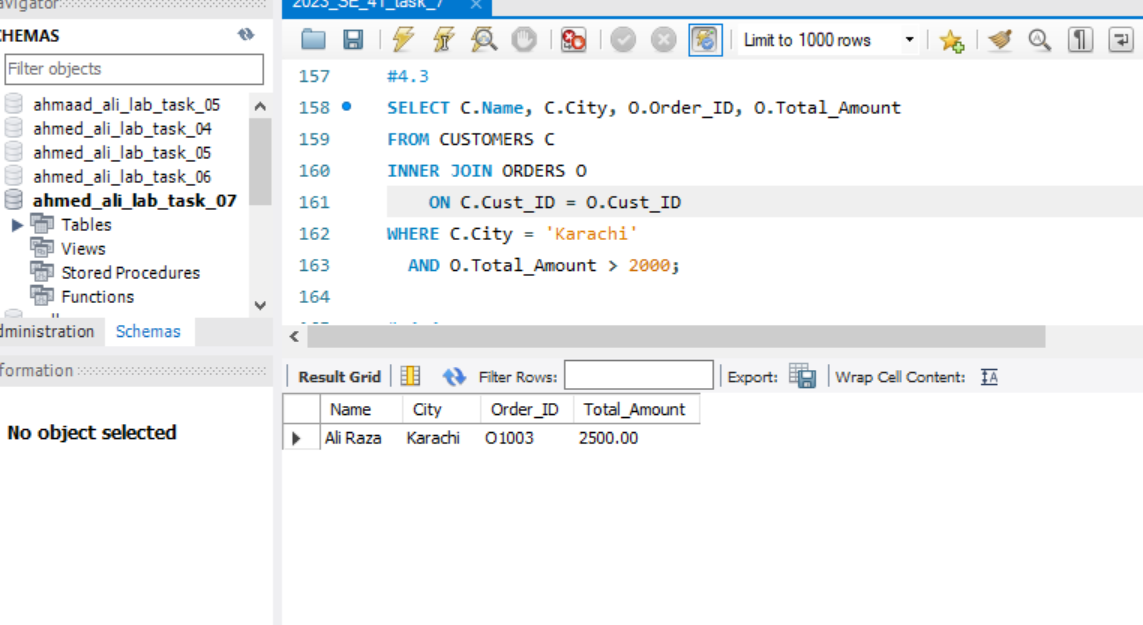
### **Why We Need Another Table**

* In our database design, **ORDERS** table stores information about customer orders but does not directly store which **products** were included in that order.
* Similarly, the **PRODUCTS** table lists all products but has no direct connection to which customer bought them.
* To connect **ORDERS** and **PRODUCTS**, we need a **link (junction) table**, usually called ORDER\_DETAILS or ORDER\_ITEMS.

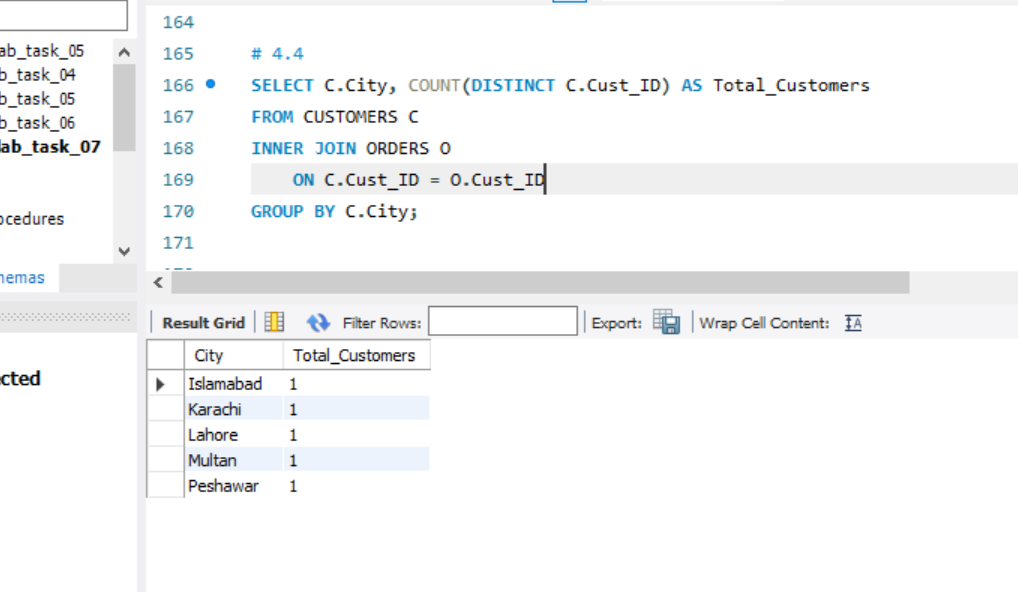
### **Purpose of the Link Table (ORDER\_DETAILS)**

* It represents the **many-to-many relationship** between ORDERS and PRODUCTS.
* One **order** can have multiple products.
* One **product** can appear in many different orders.
* Therefore, a separate table is required to manage this relationship.

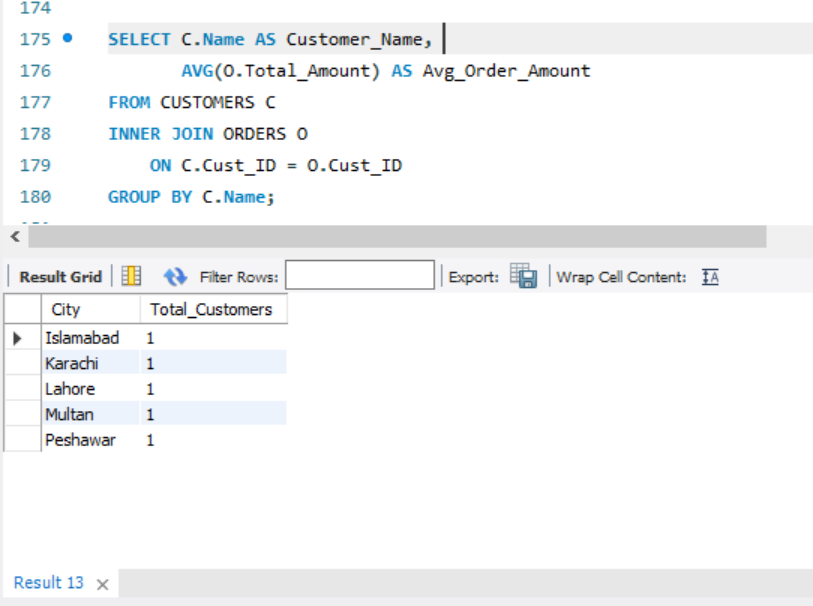
1. Retrieve all customers from "Karachi" who placed orders worth more than 2000.



1. List all cities and the number of customers from each city who have placed an order.



1. Find the average order amount for each customer who has placed at least one order.



📘 **Task 05: Theoretical Understanding of JOINs**

1. **What is a JOIN in SQL? Why are JOINs important in relational databases?**

A **JOIN** in SQL is used to combine rows from two or more tables based on a related column between them.

* **Importance:**
  + Relational databases store data in multiple normalized tables.
  + JOINs help retrieve meaningful combined information (e.g., customer details with their orders, books with their publishers).
  + They prevent data duplication and maintain consistency.

1. **Differentiate between the following types of JOINs with an example each:**

**INNER JOIN**

Returns only the rows that have matching values in both tables.

**LEFT OUTER JOIN**

Returns all rows from the **left table** and matching rows from the right table. Unmatched rows from right are shown as NULL

.

**RIGHT OUTER JOIN**

Returns all rows from the **right table** and matching rows from the left table. Unmatched left rows are NULL.

**FULL OUTER JOIN**

Returns all rows from both tables, with NULLs for missing matches.

1. **Explain a real-world scenario where using a LEFT OUTER JOIN would be more appropriate than an INNER JOIN.**

➤ Use your own PUBLISHERS and BOOKS table structure to illustrate your explanation.

Suppose I have two tables:

* **PUBLISHERS (Publisher\_ID, Publisher\_Name)**
* **BOOKS (Book\_ID, Title, Publisher\_ID**)

If I want to list **all books including those that are not yet assigned to any publisher**, a **LEFT JOIN** is more appropriate than INNER JOIN.

**SELECT B.Title, P.Publisher\_Name**

**FROM BOOKS B**

**LEFT JOIN PUBLISHERS P**

**ON B.Publisher\_ID = P.Publisher\_ID**;

INNER JOIN would exclude books without a publisher. LEFT JOIN ensures they are still shown (Publisher\_Name will be NULL.

1. **What is the significance of ON clause in a JOIN operation? What will happen if we use WHERE instead of ON?**

 **ON clause**: Defines the relationship (condition) between tables when joining. Without it, SQL would create a **Cartesian product** (every row of one table combined with every row of another).

 **WHERE clause**: Can also filter results, but when used instead of ON, it applies after joining, which might lead to wrong results or performance issues.

1. **How does using JOINs enhance data retrieval compared to querying a single table?**

 Single tables usually store **partial information**.

 JOINs allow combining **related data from multiple tables** ( more meaningful insights).

 Example: Instead of just knowing a book’s title, JOINs let you also fetch its publisher, author, price, and sales data in a single query.

## **Reflection**

During this lab, I learned the **importance of JOINs in relational databases** and how they allow us to combine data from multiple normalized tables. I understood the differences between **INNER, LEFT, RIGHT, and FULL JOINs**, and how each type is used in different real-world scenarios. For example, an INNER JOIN is best when we only want matching records, while a LEFT JOIN ensures that all records from one table appear even if there are no matches in the other.

I also practiced writing queries that involve **multi-table JOINs**, grouping, and aggregation. This helped me see how JOINs can provide more meaningful results than querying a single table alone.

## **Challenges Faced**

* At first, it was a bit confusing to differentiate when to use **LEFT JOIN vs INNER JOIN** since both can return similar results in some cases.
* Another challenge was understanding the need for a **link table (ORDER\_DETAILS)** in many-to-many relationships like **orders and products**.
* Remembering the correct SQL syntax (especially with ON vs WHERE) also required practice.