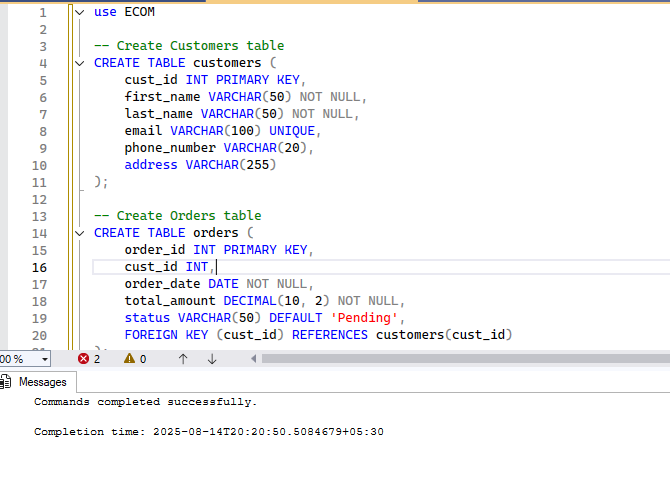
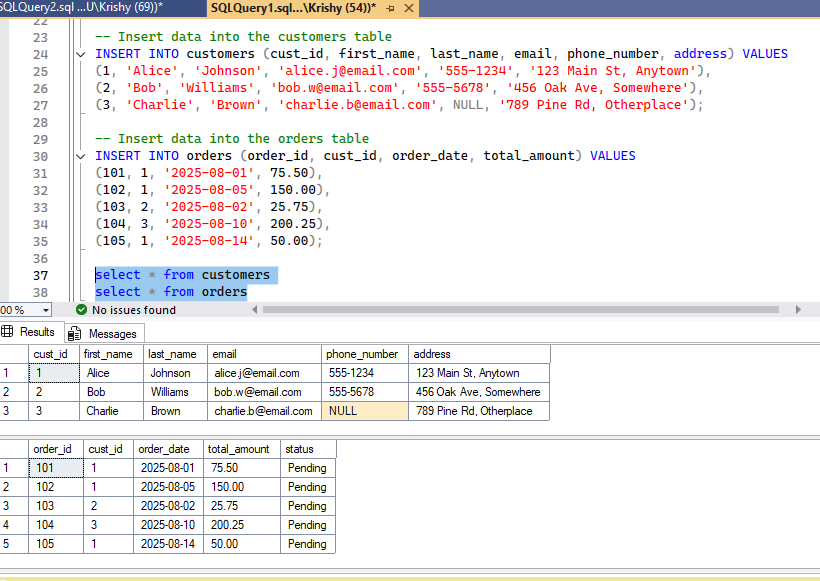
**VIEWS AND ITS USAGE**

**1. Create table and insert values**

Firstly, Created the table Customers and Orders as we are going to do View operations.





**Views**

A view is a stored query that acts like a table. It's a way to encapsulate a complex SELECT statement and give it a name. When you query the view, the database executes the underlying SELECT statement and returns the result.

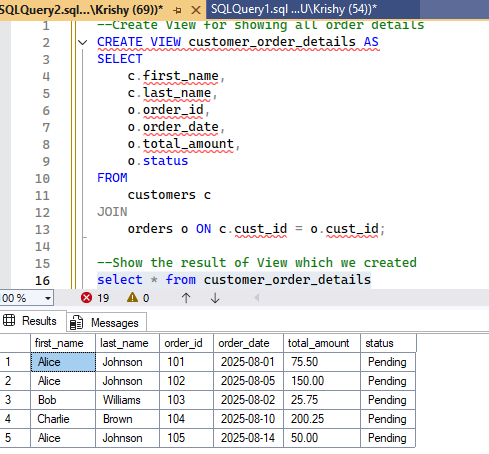
Now we will deep dive into views…

**Usage**

Firstly we create the view customer\_order\_details to show all orders placed using Select

Now, you can query this view as if it were a regular table:

SELECT \* FROM customer\_order\_details;

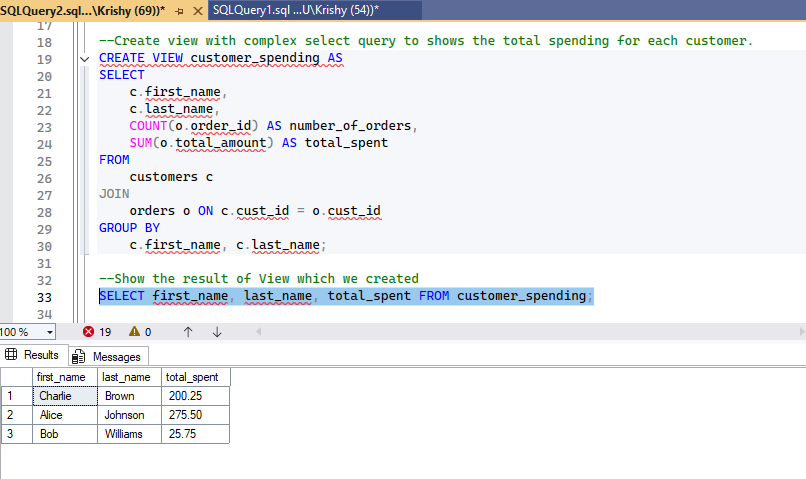


**View with Complex SELECT**

In order to view the total spending of the each customer, we created this view customer\_spending with complex select

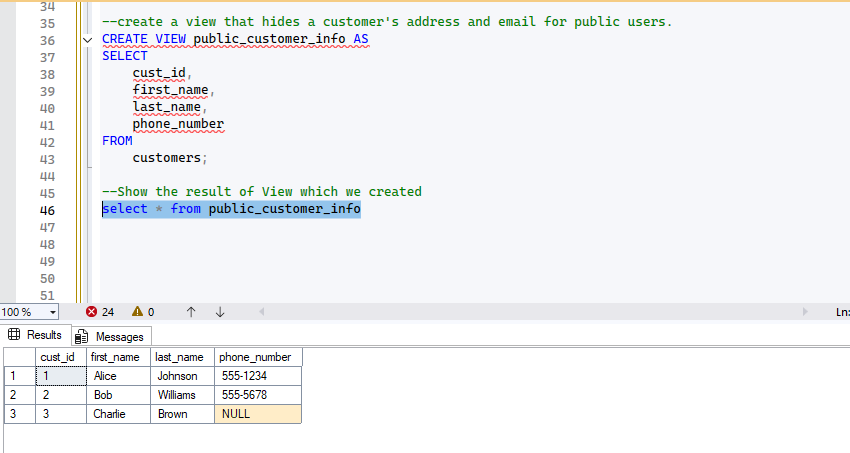
Now you can easily get this summarized data:

SELECT first\_name, last\_name, total\_spent FROM customer\_spending;



**Views for Abstraction and Security**

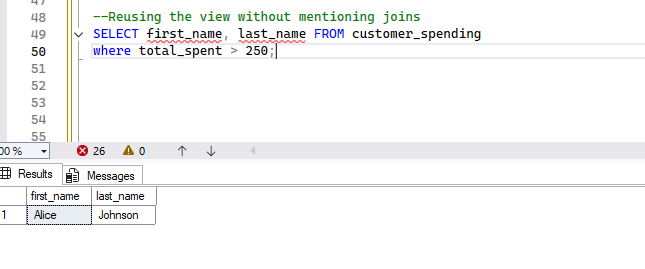
* Views can be used to restrict access to sensitive data. You can grant a user access to a view that shows only non-sensitive columns, while denying them direct access to the underlying table.
* Here we hides the customer address and email from the pubclic users
* You can grant SELECT permission on public\_customer\_info to a user, but not on the customers table itself.



## **Reusable SQL Logic**

Instead of repeating a join in multiple queries, you can use the view itself .

Here I just directly used the view which I have created for Customer\_spending earlier without using joins.



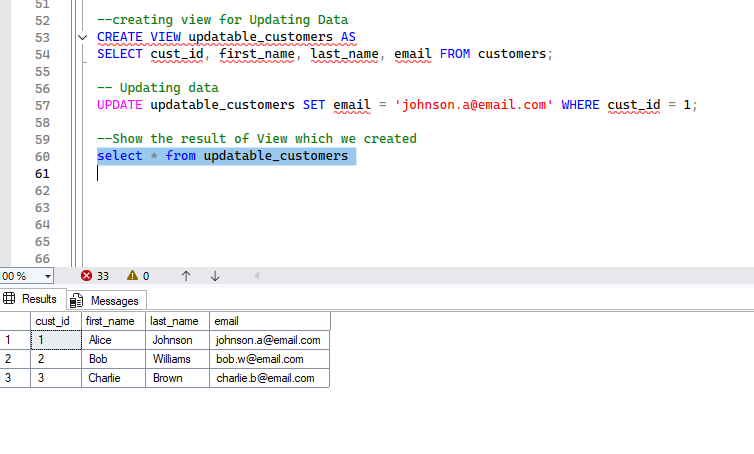
### Updating Data through a View

### In many cases, We can updating data through a view . However, there are certain strict limitations.

### If a view is based on a single table and does not contain aggregations, GROUP BY, DISTINCT, or other complex clauses, you can often INSERT, UPDATE, and DELETE through it.

**Example: Updatable View**

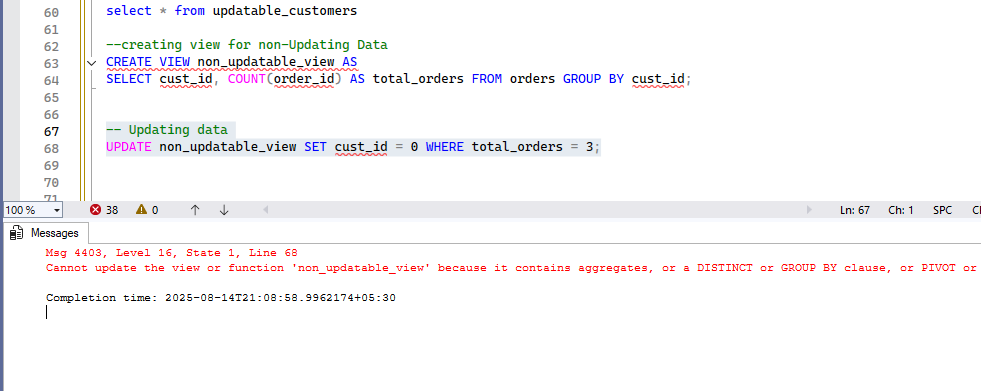
This view is updatable because it's a simple selection from a single table.



**Example: Non-Updatable View**

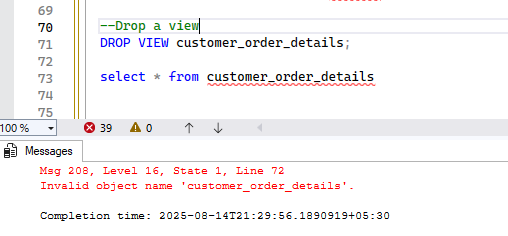
This view is not updatable because it contains an aggregation and a GROUP BY clause.

Here we can see the error below…



**Drop a view**

You can drop a view using drop view statement

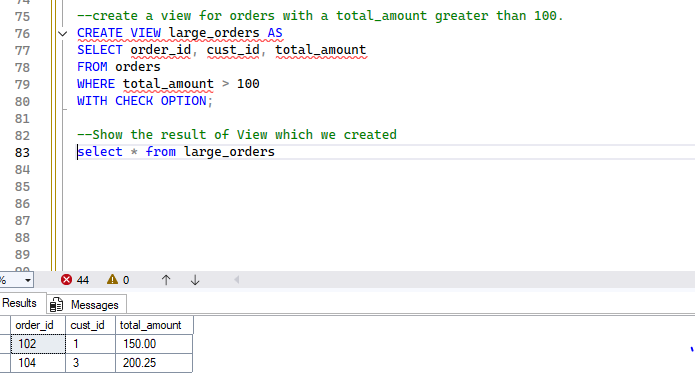


### With Check Option

The WITH CHECK OPTION clause prevents an UPDATE or INSERT statement from creating a row that the view cannot see. It ensures that all data modified through the view adheres to the view's WHERE clause.

**Example**

Let's create a view for orders with a total\_amount greater than 100.

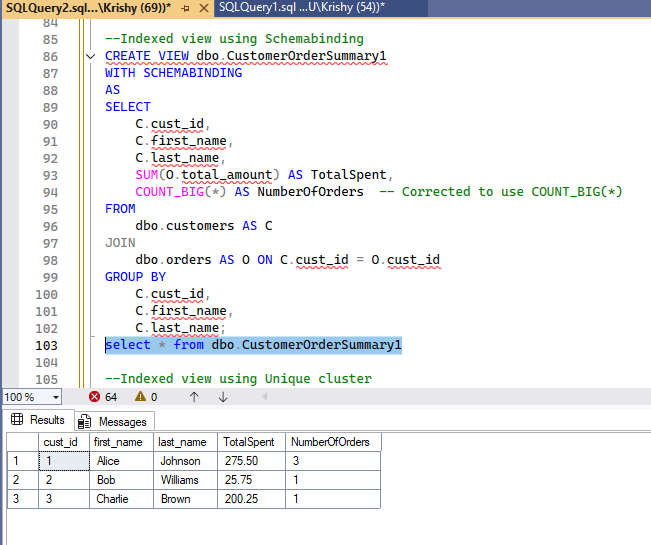


### Creating an Indexed View

Let's create an indexed view that summarizes customer spending.

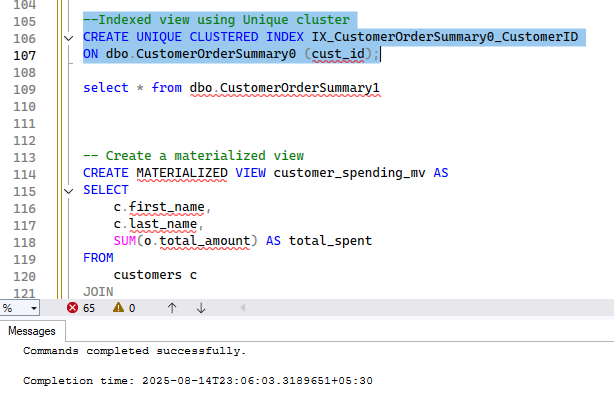
**Create the view with SCHEMABINDING**

We'll create a view that joins the customers and orders tables and aggregates the total spending. Note the use of COUNT\_BIG(\*) to count the number of orders, as required for indexed views.



#### **Create a UNIQUE CLUSTERED INDEX on the view**

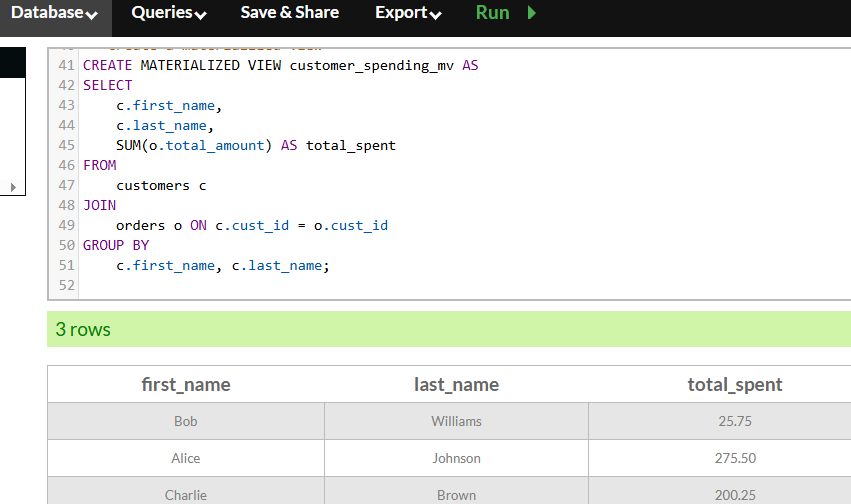
This step is what materializes the view. The database engine will now compute the results of the SELECT query and store them on disk. Any subsequent changes to the customers or orders tables will cause the indexed view's data to be automatically updated.



**Materialized view**

A materialized view is a physical table that stores the pre-computed result of a query. Unlike a regular view, a materialized view's data is stored on disk and can be refreshed manually or automatically at specific intervals.

**Example:** Materialized View (PostgreSQL syntax)



**Thank you!**