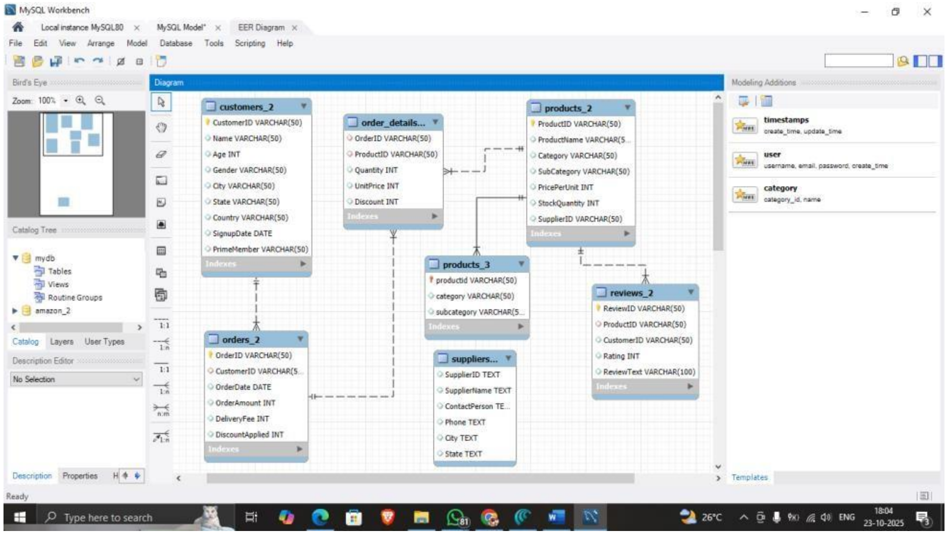
Amazon Fresh SQL Project

Task 1: ER diagram



customers_2
- CustomerID VARCHAR(50)
- Name VARCHAR(50)
- Age INT
- Gender VARCHAR(50)
- City VARCHAR(50)
- State VARCHAR(50)
- Country VARCHAR(50)
- SignupDate DATE
- PrimeMember VARCHAR(50)

order_details...
- OrderID VARCHAR(50)
- ProductID VARCHAR(50)
- Quantity INT
- UnitPrice INT
- Discount INT

products_2
- ProductID VARCHAR(50)
- ProductName VARCHAR(5...
- Category VARCHAR(50)
- SubCategory VARCHAR(50)
- PricePerUnit INT
- StockQuantity INT
- SupplierID VARCHAR(50)

products_3
- productid VARCHAR(50)
- category VARCHAR(50)
- subcategory VARCHAR(5...

reviews_2
- ReviewID VARCHAR(50)
- ProductID VARCHAR(50)
- CustomerID VARCHAR(50)
- Rating INT
- ReviewText VARCHAR(100)

orders_2
- OrderID VARCHAR(50)
- CustomerID VARCHAR(5...
- OrderDate DATE
- OrderAmount INT
- DeliveryFee INT
- DiscountApplied INT

suppliers...
- SupplierID TEXT
- SupplierName TEXT
- ContactPerson TE...
- Phone TEXT
- City TEXT
- State TEXT

Step 1 – create database

```
CREATE DATABASE amazon_2;
USE amazon_2;
```

Explanation

Creates a new database called amazon_2

The USE command sets it as the active database where all further operations will be executed.

Step 2 – import csv

Right click on database in schema > click table import wizard > select path of csv > import

Step 3 – alter table

Right click table to be alter in schema > click alter table > alter table > apply

Apply these steps for each table or alter table by query method

```
ALTER TABLE customer_2

ADD CONSTRAINT pk_custmer PRIMARY KEY (customerID);

ALTER TABLE order_details_2

ADD CONSTRAINT fk_order

FOREIGN KEY (productID)   REFERENCES products_2 (productID);
```

Task 2: Primary and foreign key

Primary key – uniquely identify each row in table and has no null

Foreign key - A column that creates a link between two tables, it references the primary key in another table.

Relationships in the Amazon_2 Database:

Customers → Orders: One customer can place many orders (One-to-Many relationship).

Orders → Order_Details: Each order can have multiple items (One-to-Many).

Products → Order_Details: A product can appear in multiple order details (One-to-Many).

Products → Reviews: Each product can have multiple reviews (One-to-Many).

Suppliers → Products: A supplier can supply many products (One-to-Many).

Customers → Reviews: A customer can write multiple reviews (One-to-Many).

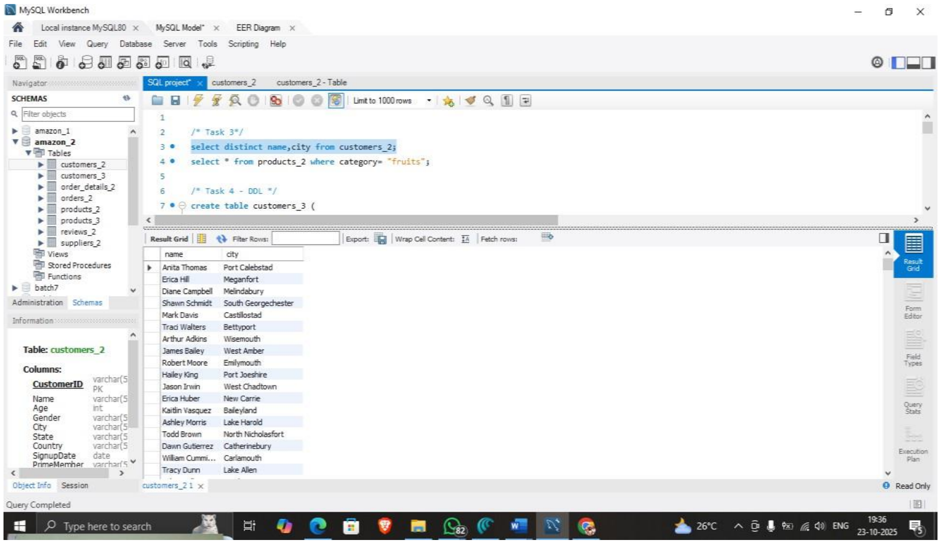
Task 3: Basic SELECT Queries

Retrieve all customers from a specific city.

SELECT DISTINCT name, city FROM customers_2;

Explanation

This query shows **unique name–city pairs** from the customers_2 table.
If the same name and city appear more than once, **only one copy** will be displayed in the result.



```
/* Task 3*/
select distinct name,city from customers_2;
select * from products_2 where category= "fruits";

/* Task 4 - DDL */
create table customers_3 (
```

name	city
Anita Thomas	Port Calebstad
Erica Hill	Meganfort
Diane Campbell	Melindabury
Shawn Schmidt	South Georgechester
Mark Davis	Castillostad
Traci Walters	Bettyport
Arthur Adkins	Wisemouth
James Bailey	West Amber
Robert Moore	Emilymouth
Hailey King	Port Joeshire
Jason Irwin	West Chadtown
Erica Huber	New Carrie
Kaitlin Vasquez	Baileyland
Ashley Morris	Lake Harold
Todd Brown	North Nicholasfort
Dawn Gutierrez	Catherinebury
William Cummi...	Carlamouth
Tracy Dunn	Lake Allen

Table: customers_2

Columns:

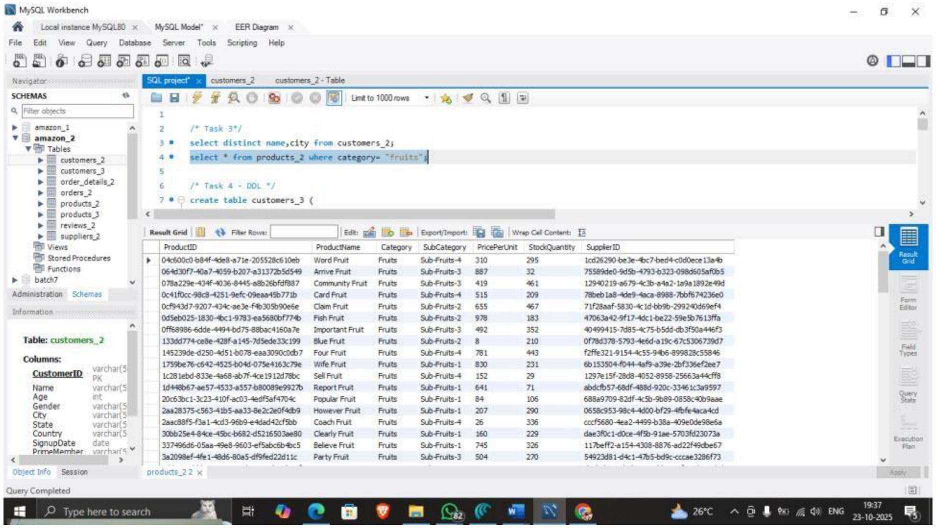
CustomerID	varchar(5 PK
Name	varchar(5
Age	int
Gender	varchar(5
City	varchar(5
State	varchar(5
Country	varchar(5
SignupDate	date
PrimeMember	varchar(5

Fetch all products under the "Fruits" category.

SELECT * FROM products_2 WHERE category = 'fruits';

Explanation

This query selects all product details from the products_2 table where the category is “fruits.”
It filters only those records belonging to the fruit category.
The * symbol means all columns will be shown.
It helps to analyze fruit-related products specifically.



```
/* Task 3*/
select distinct name,city from customers_2;
select * from products_2 where category= "fruits";

/* Task 4 - DDL */
create table customers_3 (
```

ProductID	ProductName	Category	SubCategory	PricePerUnit	StockQuantity	SupplierID
04c600c0-b84f-4de8-a71e-205528c610eb	Word Fruit	Fruits	Sub-Fruits-4	310	295	1cd26290-be3e-4bc7-bed4-c0d0ece13a4b
064d30f7-40a7-4059-b207-a31372b5d549	Arrive Fruit	Fruits	Sub-Fruits-3	887	32	75589de0-9d5b-4793-b323-098d605af0b5
078a229e-434f-4036-8445-a8b26bfdf887	Community Fruit	Fruits	Sub-Fruits-3	419	461	12940219-a679-4c3b-a4a2-1a9a1892e49d
0c41f0cc-98c8-4251-9efc-09eaa45b771b	Card Fruit	Fruits	Sub-Fruits-4	515	209	78beb1a8-4de9-4aca-8988-7bbf674236e0
0cf943d7-9207-434c-ae3e-f4b305b90e6e	Claim Fruit	Fruits	Sub-Fruits-2	655	467	71f28aaf-5830-4c1d-bb9b-299240d69ef4
0d5eb025-1830-4bc1-9783-ea5680bf774b	Fish Fruit	Fruits	Sub-Fruits-2	978	183	47063a42-9f17-4dc1-be22-59e5b7613ffa
0ff68986-6dde-4494-bd75-88bac4160a7e	Important Fruit	Fruits	Sub-Fruits-3	492	352	40499415-7d85-4c75-b5dd-db3f50a446f3
133dd774-ce8e-428f-a145-7d5ede33c199	Blue Fruit	Fruits	Sub-Fruits-2	8	210	0f78d378-5793-4e6d-a19c-67c5306739d7
145239de-d250-4d51-b078-eaa3090c0db7	Four Fruit	Fruits	Sub-Fruits-4	781	443	f2ffe321-9154-4c55-94b6-899828c55846
1759be76-c642-4525-b04d-075e4163c79e	Wife Fruit	Fruits	Sub-Fruits-1	830	231	6b153504-f044-4af9-a39e-2bf336ef2ee7
1c281ebd-833e-4a68-ab7f-4ce1912d78bc	Sell Fruit	Fruits	Sub-Fruits-4	152	29	1297e15f-28d8-4052-8958-25663a44cff8
1d448b67-ae57-4533-a557-b80089e9927b	Report Fruit	Fruits	Sub-Fruits-1	641	71	abdcfb57-68df-488d-920c-33461c3a9597
20c63bc1-3c23-410f-ac03-4edf5af4704c	Popular Fruit	Fruits	Sub-Fruits-1	84	106	688a9709-82df-4c5b-9b89-0858c40b9aae
2aa28375-c563-41b5-aa33-8e2c2e0f4db9	However Fruit	Fruits	Sub-Fruits-1	207	290	0658c953-98c4-4d00-bf29-4fbfe4aca4cd
2aac88f5-f3a1-4cd3-96b9-e4dad42cf5bb	Coach Fruit	Fruits	Sub-Fruits-4	26	336	cccf5680-4ea2-4499-b38a-409e0de98e6a
30bb25e4-84ce-45bc-b682-d5216503ae80	Clearly Fruit	Fruits	Sub-Fruits-1	160	229	dae3f0c1-d0ce-4f5b-91ae-5703fd23073a
337496d6-05aa-49e8-9603-ef5abc6b4bc5	Believe Fruit	Fruits	Sub-Fruits-1	745	326	117beff2-a154-4308-8876-ad22f49dbe67
3a2098ef-4fe1-48d6-80a5-df9fed22d11c	Party Fruit	Fruits	Sub-Fruits-3	504	270	54923d81-d4c1-47b5-bd9c-cccae3286f73

Task 4: Creating Table with Constraints

Write DDL statements to recreate the Customers table with the following constraints:

```
CREATE TABLE customers_3 (
 customerID VARCHAR(50) PRIMARY KEY,
 Name VARCHAR(50) UNIQUE,
 Age INT NOT NULL CHECK (Age > 18)
);
```

Explanation

This query creates a new table named customers_3 with defined constraints.
The PRIMARY KEY ensures each customer ID is unique.
The UNIQUE constraint prevents duplicate names.
The CHECK (Age > 18) ensures only adult customers are added.

Task 5: DML Operations

Insert 3 new rows into the Products table using INSERT statements.

```
insert into products_2 (productID, productname, category, subcategory, priceperunit, stockquantity, supplierID)

values ("DF00000001"," Dry fruit","fruit","sub-fruit-1",250,300,"DFS0000001"),

("PS00000001","packed snack","snack","sub-snack-3",20,400,"PSS0000001"),

("GF00000001","ground vegetable","vegetable","sub-vegetable-1",50,300,"GFS0000001");
```

Explanation

This query adds three new product records to the products_2 table.
Each row includes details such as product ID, name, category, and stock quantity.
The INSERT command is used to add new data into an existing table.
It helps expand the product database for further analysis.

Task 6: DML Operations

Update the stock quantity of a product where ProductID matches a specific ID

```
UPDATE products_2 SET stockquantity = 300 WHERE productid = "0006853b-74cb-44a2-91ed-699aa31c5b5b";
```

Explanation

This query updates the stock quantity of a specific product to 300 units.
The WHERE clause ensures that only the matching product ID is updated.
It is used to correct or modify inventory levels.
This helps maintain accurate stock data in the system.

Task 7: DML Operations

Delete a supplier from the Suppliers table where their city matches a specific value.

DELETE FROM suppliers_2 WHERE city= "south ana";

Explanation

This query removes supplier records located in "South Ana."
The WHERE clause ensures that only suppliers from that city are deleted.
It's used for cleaning up outdated or irrelevant supplier data.
Once deleted, the record cannot be recovered.

Task 8: Adding Constraints and Defaults

Add a CHECK constraint to ensure that ratings in the Reviews table are between 1 and 5.

ALTER TABLE reviews_2 ADD CONSTRAINT ck_review CHECK (rating BETWEEN 1 AND 5);
Add a DEFAULT constraint for the PrimeMember column in the Customers table (default value: "No").
ALTER TABLE customers_2 ALTER COLUMN primemember SET DEFAULT 'No';

Explanation

This adds a check constraint that limits ratings to values between 1 and 5.
It prevents invalid ratings like 0 or 6 from being inserted.
This maintains data accuracy in the reviews table.
It helps ensure only valid feedback scores are stored.

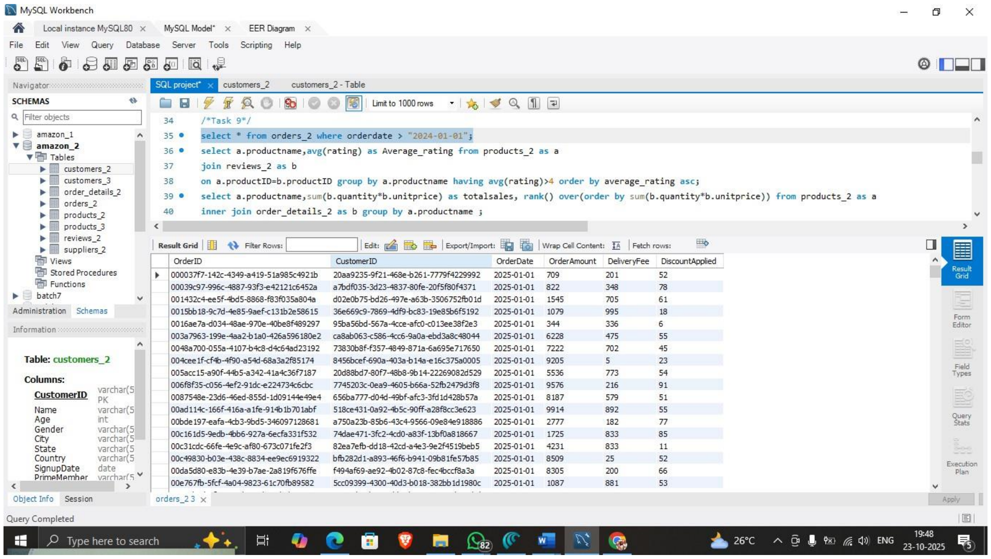
Task 9: Filtering and Aggregation

WHERE clause to find orders placed after 2024-01-01.

SELECT * FROM orders_2 WHERE orderdate > '2024-01-01';

Explanation

This query retrieves all orders placed after January 1, 2024.
The WHERE clause filters records based on the order date.
It helps in analyzing recent sales or post-2024 performance.
Useful for time-based order tracking and reports.



```
/*Task 9*/
select * from orders_2 where orderdate > "2024-01-01";
select a.productname,avg(rating) as Average_rating from products_2 as a
join reviews_2 as b
on a.productID=b.productID group by a.productname having avg(rating)>4 order by average_rating asc;
select a.productname,sum(b.quantity*b.unitprice) as totalsales, rank() over(order by sum(b.quantity*b.unitprice)) from products_2 as a
inner join order_details_2 as b group by a.productname ;
```

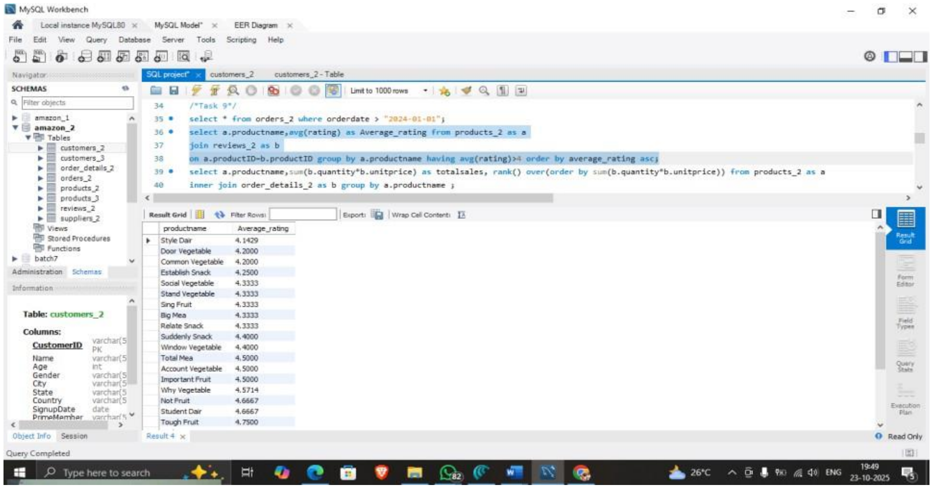
OrderID	CustomerID	OrderDate	OrderAmount	DeliveryFee	DiscountApplied
000037f7-142c-4349-a419-51a985c4921b	20aa9235-9f21-468e-b261-7779f4229992	2025-01-01	709	201	52
00039c97-996c-4887-93f3-e42121c6452a	a7bdf035-3d23-4837-80fe-20f5f80f4371	2025-01-01	822	348	78
001432c4-ee5f-4bd5-8868-f83f035a804a	d02e0b75-bd26-497e-a63b-3506752fb01d	2025-01-01	1545	705	61
0015bb18-9c7d-4e85-9aef-c131b2e58615	36e669c9-7869-4df9-bc83-19e85b6f5192	2025-01-01	1079	995	18
0016ae7a-d034-48ae-970e-40be8f489297	95ba56bd-567a-4cce-afc0-c013ee38f2e3	2025-01-01	344	336	6
003a7963-199e-4aa2-b1a0-426a596180e2	ca8ab063-c586-4cc6-9a0a-ebd3a8c48044	2025-01-01	6228	475	55
0048a700-055a-4107-b4c8-d4c64ad23192	73830b8f-f357-4849-871a-6a695e717650	2025-01-01	7222	702	45
004cee1f-cf4b-4f90-a54d-68a3a2f85174	8456bcef-690a-403a-b14a-e16c375a0005	2025-01-01	9205	5	23
005acc15-a90f-44b5-a342-41a4c36f7187	20d88bd7-80f7-48b8-9b14-22269082d529	2025-01-01	5536	773	54
006f8f35-c056-4ef2-91dc-e224734c6cbc	7745203c-0ea9-4605-b66a-52fb2479d3f8	2025-01-01	9576	216	91
0087548e-23d6-46ed-855d-1d09144e49e4	656ba777-d04d-49bf-afc3-3fd1d428b57a	2025-01-01	8187	579	51
00ad114c-166f-416a-a1fe-914b1b701abf	518ce431-0a92-4b5c-90ff-a28f8cc3e623	2025-01-01	9914	892	55
00bde197-eafa-4cb3-9bd5-346097128681	a750a23b-85b6-43c4-9566-09e84e918886	2025-01-01	2777	182	77
00c161d5-9edb-4bb6-927a-6ecfa331f532	74dae471-3fc2-4cd0-a83f-13bf0a818667	2025-01-01	1725	833	85
00c31cdc-66fe-4e9c-af80-673c071fe2f3	82ea7efb-dd18-42cd-a4e3-9e2f4519beb5	2025-01-01	4231	833	11
00c49830-b03e-438c-8834-ee9ec6919322	bfb282d1-a893-46f6-b941-09b81fe57b85	2025-01-01	8509	25	52
00da5d80-e83b-4e39-b7ae-2a819f676ffe	f494af69-ae92-4b02-87c8-fec4bccf8a3a	2025-01-01	8305	200	66
00e767fb-5fcf-4a04-9823-61c70fb89582	5cc09399-4300-40d3-b018-382bb1d1980c	2025-01-01	1087	881	53

HAVING clause to list products with average ratings greater than 4 , GROUP BY and ORDER BY clauses to rank products by total sales.

```
SELECT a.productname, AVG(rating) AS Average_rating
FROM products_2 a JOIN reviews_2 b ON a.productID=b.productID
GROUP BY a.productname
HAVING AVG(rating)>4;
```

Explanation

This query finds products with an average rating above 4.
It joins products_2 and reviews_2 to calculate ratings per product.
GROUP BY groups products, while HAVING filters high-rated ones.
It identifies top-performing products based on customer feedback



```
/*Task 9*/
select * from orders_2 where orderdate > "2024-01-01";
select a.productname,avg(rating) as Average_rating from products_2 as a
join reviews_2 as b
on a.productID=b.productID group by a.productname having avg(rating)>4 order by average_rating asc;
select a.productname,sum(b.quantity*b.unitprice) as totalsales, rank() over(order by sum(b.quantity*b.unitprice)) from products_2 as a
inner join order_details_2 as b group by a.productname ;
```

productname	Average_rating
Style Dair	4.1429
Door Vegetable	4.2000
Common Vegetable	4.2000
Establish Snack	4.2500
Social Vegetable	4.3333
Stand Vegetable	4.3333
Sing Fruit	4.3333
Big Mea	4.3333
Relate Snack	4.3333
Suddenly Snack	4.4000
Window Vegetable	4.4000
Total Mea	4.5000
Account Vegetable	4.5000
Important Fruit	4.5000
Why Vegetable	4.5714
Not Fruit	4.6667
Student Dair	4.6667
Tough Fruit	4.7500

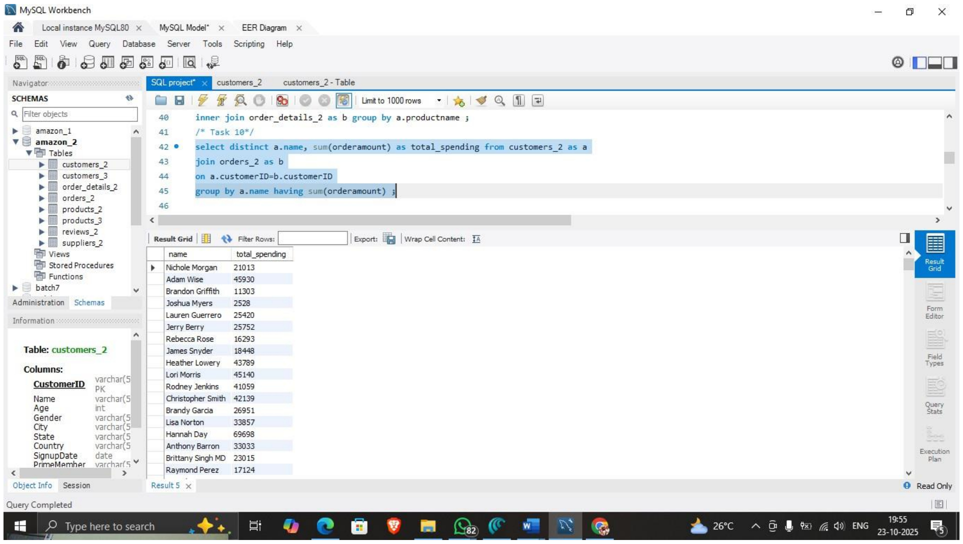
Task 10: High-Value Customer Identification

Calculate each customer's total spending.

```
SELECT a.name, SUM(b.orderamount) AS total_spending
FROM customers_2 a JOIN orders_2 b ON a.customerID=b.customerID
GROUP BY a.name
HAVING SUM(orderamount)>5000;
```

Explanation

This query calculates how much each customer has spent in total.
It sums up all order amounts for each customer.
Only those spending above ₹5,000 are displayed.
This helps identify high-value customers.



```
inner join order_details_2 as b group by a.productname ;
/* Task 10*/
select distinct a.name, sum(orderamount) as total_spending from customers_2 as a
join orders_2 as b
on a.customerID=b.customerID
group by a.name having sum(orderamount) ;
```

name	total_spending
Nichole Morgan	21013
Adam Wise	45930
Brandon Griffith	11303
Joshua Myers	2528
Lauren Guerrero	25420
Jerry Berry	25752
Rebecca Rose	16293
James Snyder	18448
Heather Lowery	43789
Lori Morris	45140
Rodney Jenkins	41059
Christopher Smith	42139
Brandy Garcia	26951
Lisa Norton	33857
Hannah Day	69698
Anthony Barron	33033
Brittany Singh MD	23015
Raymond Perez	17124

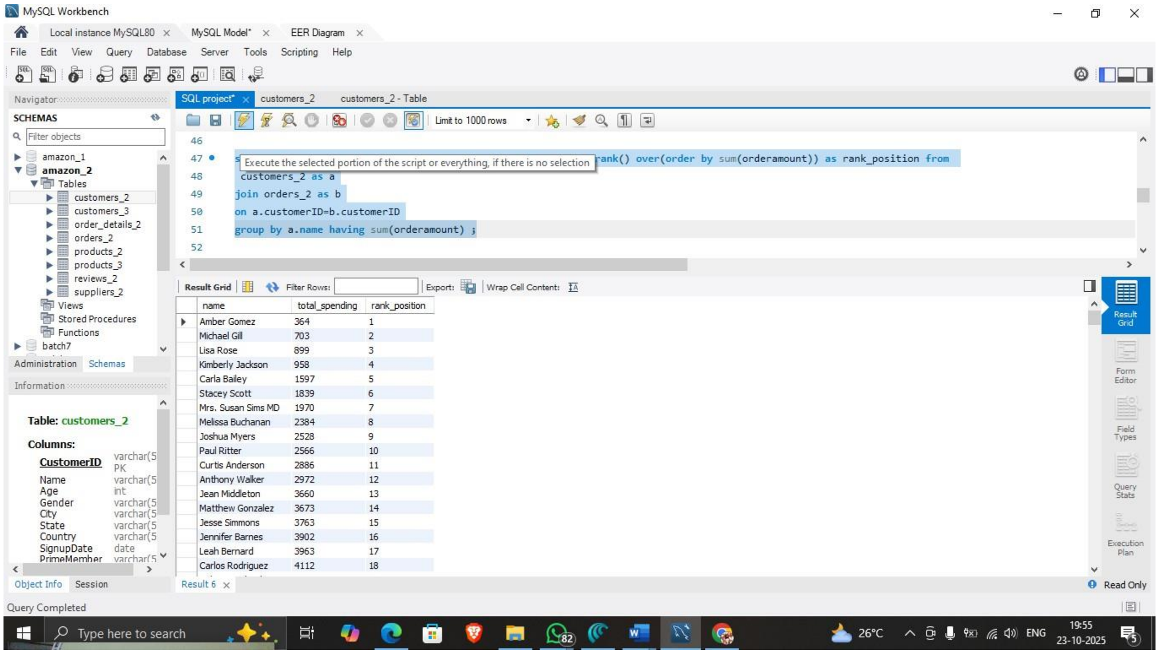
Query Completed

Rank customers based on their spending.

SELECT distinct a.name, SUM(orderamount) AS total_spending , RANK()
OVER(ORDER BY SUM(orderamount)) AS rank_position FROM
customers_2 AS a
JOIN orders_2 AS b
ON a.customerID=b.customerID
GROUP BY a.name HAVING SUM(orderamount) ;

Explanation

This query ranks customers based on total spending.
RANK() assigns position numbers according to spending amounts.
GROUP BY ensures totals are calculated per customer.
It helps find the top buyers in order of expenditure.



Execute the selected portion of the script or everything, if there is no selection

```
s                                                    rank() over(order by sum(orderamount)) as rank_position from
 customers_2 as a
join orders_2 as b
on a.customerID=b.customerID
group by a.name having sum(orderamount) ;
```

Table: customers_2

Columns:

CustomerID	varchar(5 PK
Name	varchar(5
Age	int
Gender	varchar(5
City	varchar(5
State	varchar(5
Country	varchar(5
SignupDate	date
PrimeMember	varchar(5

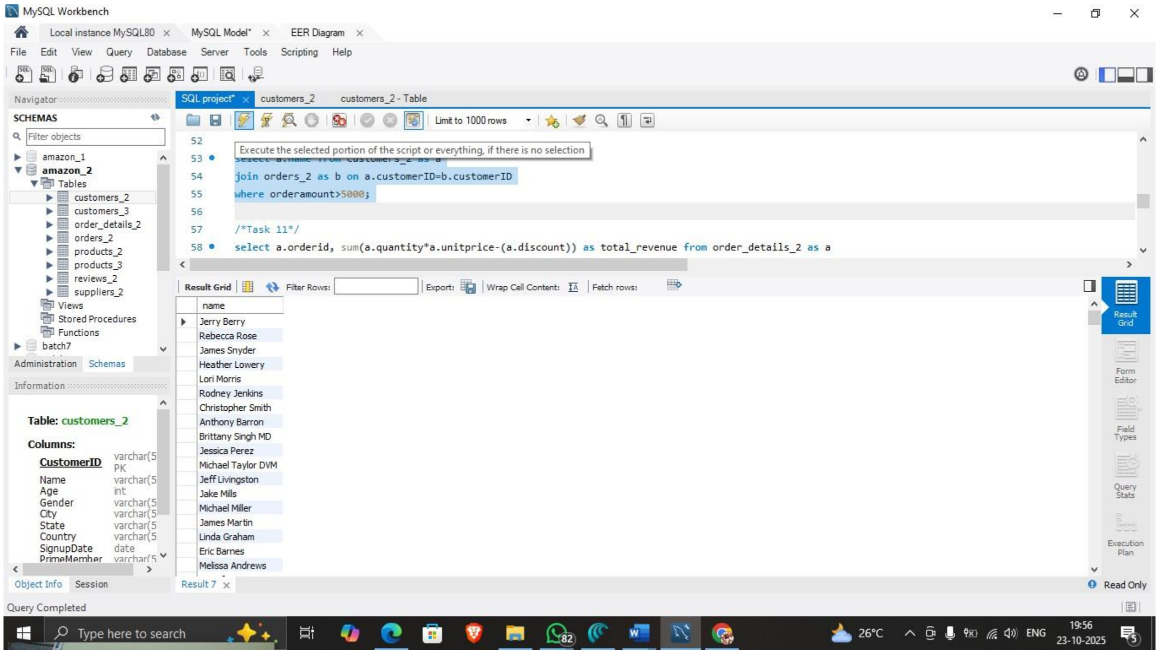
name	total_spending	rank_position
Amber Gomez	364	1
Michael Gill	703	2
Lisa Rose	899	3
Kimberly Jackson	958	4
Carla Bailey	1597	5
Stacey Scott	1839	6
Mrs. Susan Sims MD	1970	7
Melissa Buchanan	2384	8
Joshua Myers	2528	9
Paul Ritter	2566	10
Curtis Anderson	2886	11
Anthony Walker	2972	12
Jean Middleton	3660	13
Matthew Gonzalez	3673	14
Jesse Simmons	3763	15
Jennifer Barnes	3902	16
Leah Bernard	3963	17
Carlos Rodriguez	4112	18

Identify customers who have spent more than ₹5,000.

SELECT a.name FROM customers_2 AS a
JOIN orders_2 AS b ON a.customerID=b.customerID
WHERE orderamount>5000;

Explanation

This query lists customer names who placed orders over ₹5,000.
It connects the customers and orders tables using JOIN.
The WHERE clause filters only large purchases.
It helps identify premium or frequent buyers.



```
select a.name from customers_2 as a
join orders_2 as b on a.customerID=b.customerID
where orderamount>5000;

/*Task 11*/
select a.orderid, sum(a.quantity*a.unitprice-(a.discount)) as total_revenue from order_details_2 as a
```

name
Jerry Berry
Rebecca Rose
James Snyder
Heather Lowery
Lori Morris
Rodney Jenkins
Christopher Smith
Anthony Barron
Brittany Singh MD
Jessica Perez
Michael Taylor DVM
Jeff Livingston
Jake Mills
Michael Miller
James Martin
Linda Graham
Eric Barnes
Melissa Andrews

Table: customers_2

Columns:

CustomerID	varchar(5 PK
Name	varchar(5
Age	int
Gender	varchar(5
City	varchar(5
State	varchar(5
Country	varchar(5
SignupDate	date
PrimeMember	varchar(5

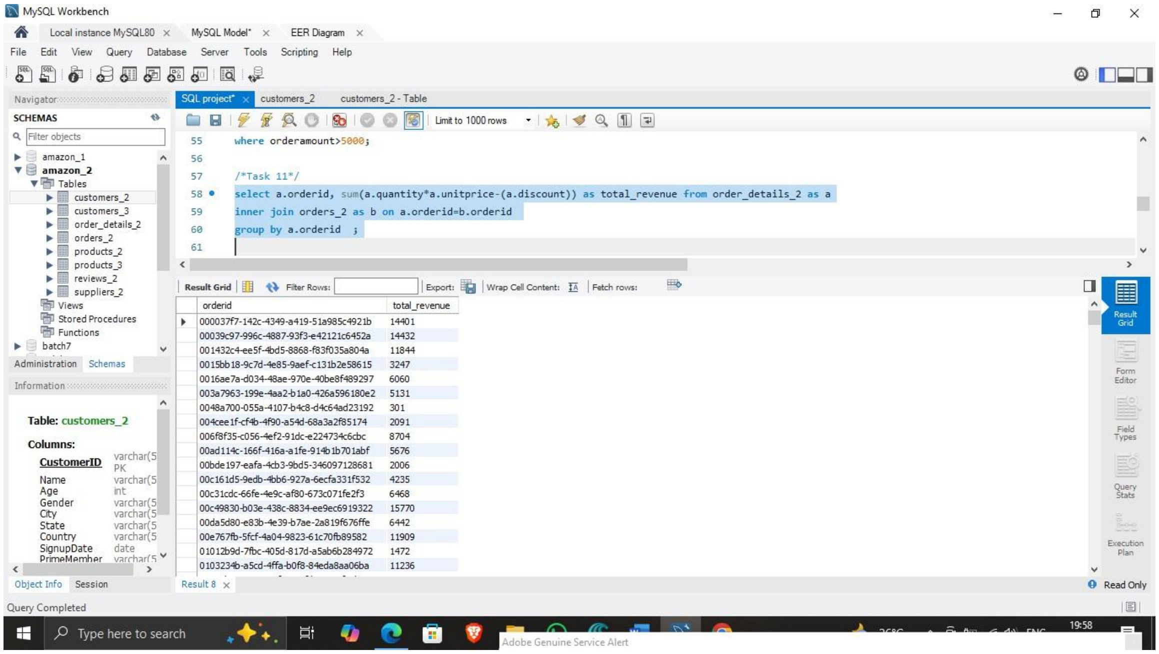
Task 11: Revenue and Supplier Stock Analysis

Join the Orders and OrderDetails tables to calculate total revenue per order.

SELECT a.orderid, SUM(a.quantity*a.unitprice - a.discount) AS total_revenue
FROM order_details_2 a GROUP BY a.orderid;

Explanation

This query calculates the total revenue for each order.
It multiplies quantity by unit price and subtracts discounts.
GROUP BY ensures the result is per order ID.
Useful for understanding sales performance per transaction.



```
where orderamount>5000;

/*Task 11*/
select a.orderid, sum(a.quantity*a.unitprice-(a.discount)) as total_revenue from order_details_2 as a
inner join orders_2 as b on a.orderid=b.orderid
group by a.orderid  ;
```

Table: customers_2

Columns:

CustomerID	varchar(5 PK
Name	varchar(5
Age	int
Gender	varchar(5
City	varchar(5
State	varchar(5
Country	varchar(5
SignupDate	date
PrimeMember	varchar(5

orderid	total_revenue
000037f7-142c-4349-a419-51a985c4921b	14401
00039c97-996c-4887-93f3-e42121c6452a	14432
001432c4-ee5f-4bd5-8868-f83f035a804a	11844
0015bb18-9c7d-4e85-9aef-c131b2e58615	3247
0016ae7a-d034-48ae-970e-40be8f489297	6060
003a7963-199e-4aa2-b1a0-426a596180e2	5131
0048a700-055a-4107-b4c8-d4c64ad23192	301
004cee1f-cf4b-4f90-a54d-68a3a2f85174	2091
006f8f35-c056-4ef2-91dc-e224734c6cbc	8704
00ad114c-166f-416a-a1fe-914b1b701abf	5676
00bde197-eafa-4cb3-9bd5-346097128681	2006
00c161d5-9edb-4bb6-927a-6ecfa331f532	4235
00c31cdc-66fe-4e9c-af80-673c071fe2f3	6468
00c49830-b03e-438c-8834-ee9ec6919322	15770
00da5d80-e83b-4e39-b7ae-2a819f676ffe	6442
00e767fb-5fcf-4a04-9823-61c70fb89582	11909
01012b9d-7fbc-405d-817d-a5ab6b284972	1472
0103234b-a5cd-4ffa-b0f8-84eda8aa06ba	11236

Identify customers who placed the most orders in a specific time period.

SELECT a.name,b.orderid FROM Customers_2 AS a
INNER JOIN orders_2 AS b ON a.customerid=b.customerid
WHERE orderdate="2025-01-01" ORDER BY a.name ASC LIMIT 5;

Explanation

This query finds customers who placed orders on January 1, 2025.
It lists up to five records ordered alphabetically by name.
The INNER JOIN links customers with their orders.
It’s used to track order activity on a specific date.

Find the supplier with the most products in stock.

```
SELECT b.suppliername, SUM(a.stockquantity) AS total_stock
FROM products_2 a JOIN suppliers_2 b ON a.supplierid=b.supplierid
GROUP BY b.suppliername ORDER BY total_stock DESC;
```

Explanation

This query identifies which supplier has the highest stock quantity.
It totals the stock quantity for each supplier using SUM().
ORDER BY DESC lists them from highest to lowest.
Useful for supply chain and inventory planning.

Task 12: Normalization

Separate product categories and subcategories into a new table.
Step 1 – separate table

```
SELECT productid ,category, subcategory FROM products_2;
```

column separated and export as csv new table (products_3)

Explanation

This query extracts only the productid, category, and subcategory columns from the products_2 table.
It separates them into a new table to remove redundancy and organize data more efficiently.
This step follows normalization rules by dividing data into smaller, related tables.
The exported data (as CSV) is then used to create a new table named products_3.

Create foreign keys to maintain relationships.
step 2 – alter table

```
ALTER TABLE products_3
ADD CONSTRAINT fk_pro3
FOREIGN KEY (productid) REFERENCES products_2 (productid);
```

Explanation

This query links the new products_3 table with the original products_2 table using a foreign key.
It ensures that every productid in products_3 must already exist in products_2.
This maintains referential integrity between both tables.
It helps prevent orphan records and ensures consistency in the database.

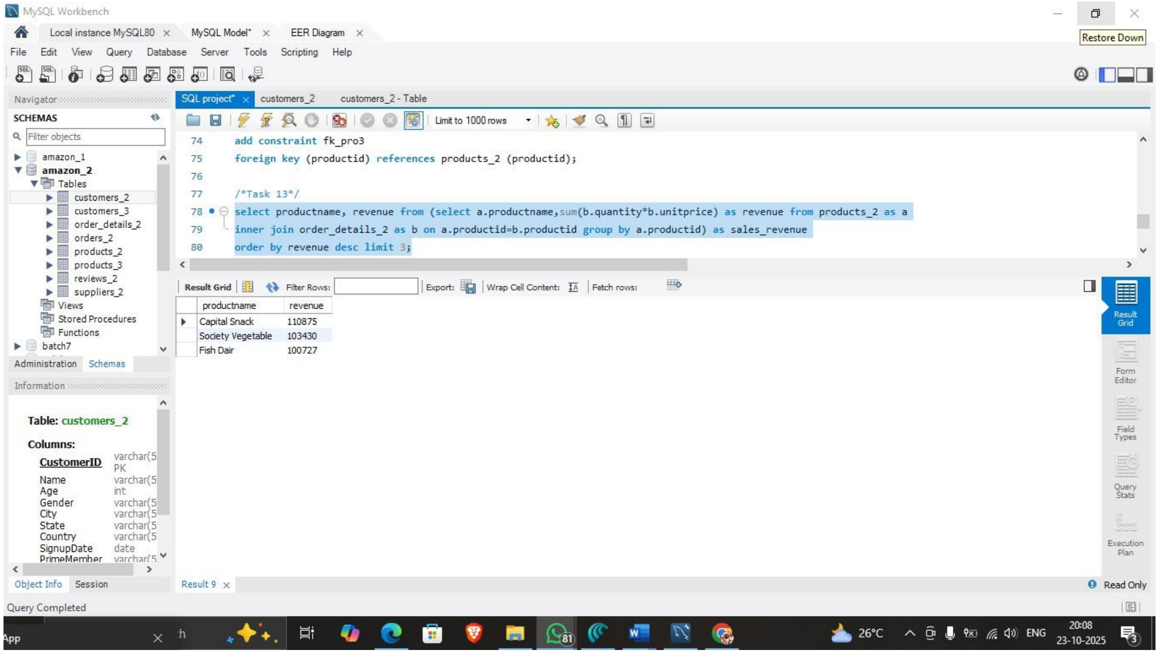
Task 13: Subquery for Top 3 Products

Identify the top 3 products based on sales revenue.

```
SELECT productname, revenue FROM (
 SELECT a.productname, SUM(b.quantity*b.unitprice) AS revenue
 FROM products_2 a JOIN order_details_2 b ON a.productid=b.productid
 GROUP BY a.productid
) AS sales_revenue ORDER BY revenue DESC LIMIT 3;
```

Explanation

This query finds the top 3 products generating the most revenue.
It uses a subquery to calculate revenue for each product.
Then it orders them in descending order of sales.
Helps identify the best-selling products overall.



```
add constraint fk_pro3
foreign key (productid) references products_2 (productid);

/*Task 13*/
select productname, revenue from (select a.productname,sum(b.quantity*b.unitprice) as revenue from products_2 as a
inner join order_details_2 as b on a.productid=b.productid group by a.productid) as sales_revenue
order by revenue desc limit 3;
```

productname	revenue
Capital Snack	110875
Society Vegetable	103430
Fish Dair	100727

Table: customers_2

Columns:

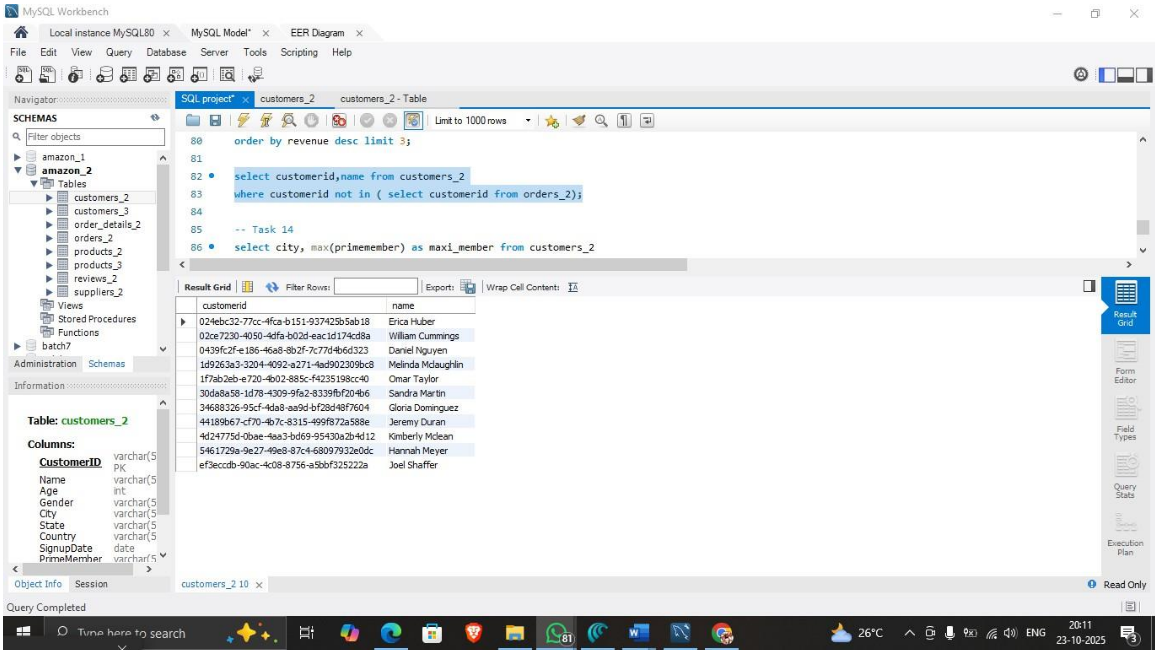
CustomerID	varchar(5 PK
Name	varchar(5
Age	int
Gender	varchar(5
City	varchar(5
State	varchar(5
Country	varchar(5
SignupDate	date
PrimeMember	varchar(5

Find customers who haven’t placed any orders yet.

SELECT customerid,name FROM customers_2
WHERE customerid NOT IN (SELECT customerid FROM orders_2);

Explanation

This query lists customers who haven’t placed any orders yet.
It compares IDs from the customers_2 table with orders_2.
Those missing in orders_2 are displayed.
It’s helpful for identifying inactive customers.



```
order by revenue desc limit 3;

select customerid,name from customers_2
where customerid not in ( select customerid from orders_2);

-- Task 14
select city, max(primemember) as maxi_member from customers_2
```

customerid	name
024ebc32-77cc-4fca-b151-937425b5ab18	Erica Huber
02ce7230-4050-4dfa-b02d-eac1d174cd8a	William Cummings
0439fc2f-e186-46a8-8b2f-7c77d4b6d323	Daniel Nguyen
1d9263a3-3204-4092-a271-4ad902309bc8	Melinda Mclaughlin
1f7ab2eb-e720-4b02-885c-f4235198cc40	Omar Taylor
30da8a58-1d78-4309-9fa2-8339fbf204b6	Sandra Martin
34688326-95cf-4da8-aa9d-bf28d48f7604	Gloria Dominguez
44189b67-cf70-4b7c-8315-499f872a588e	Jeremy Duran
4d24775d-0bae-4aa3-bd69-95430a2b4d12	Kimberly Mclean
5461729a-9e27-49e8-87c4-68097932e0dc	Hannah Meyer
ef3eccdb-90ac-4c08-8756-a5bbf325222a	Joel Shaffer

Query Completed

Task 14: Prime Member and Category Insights

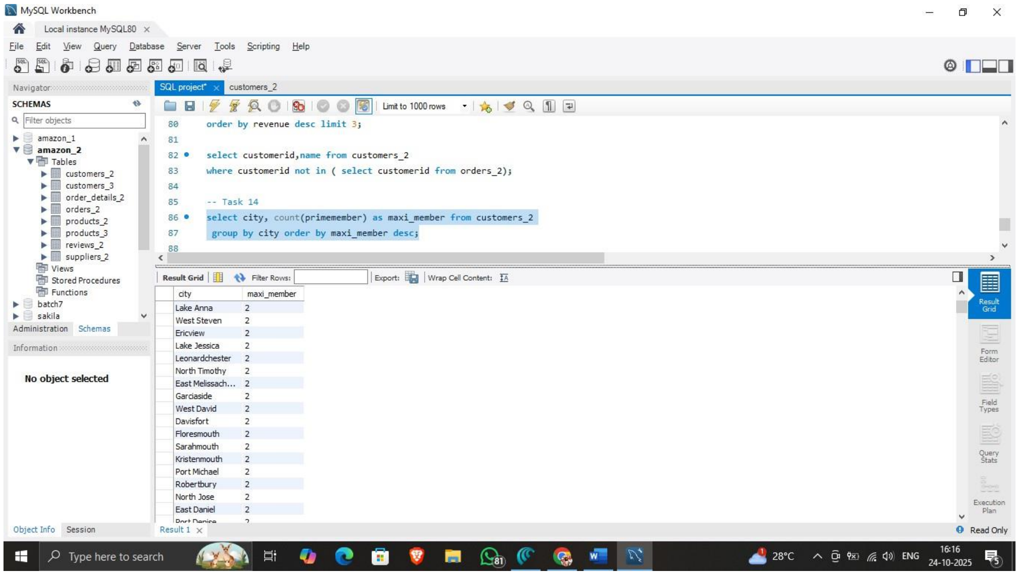
Which cities have the highest concentration of Prime members?

SELECT city, COUNT(primemember) AS maxi_member FROM customers_2

GROUP BY city ORDER BY maxi_member DESC;

Explanation

This query finds which cities have the highest number of Prime members.
It counts how many customers (using COUNT(primemember)) are in each city.
The GROUP BY groups customers by city, and ORDER BY DESC arranges results from highest to lowest.
This helps identify cities with the most Prime subscribers for business targeting.



```
order by revenue desc limit 3;

select customerid,name from customers_2
where customerid not in ( select customerid from orders_2);

-- Task 14
select city, count(primemember) as maxi_member from customers_2
 group by city order by maxi_member desc;
```

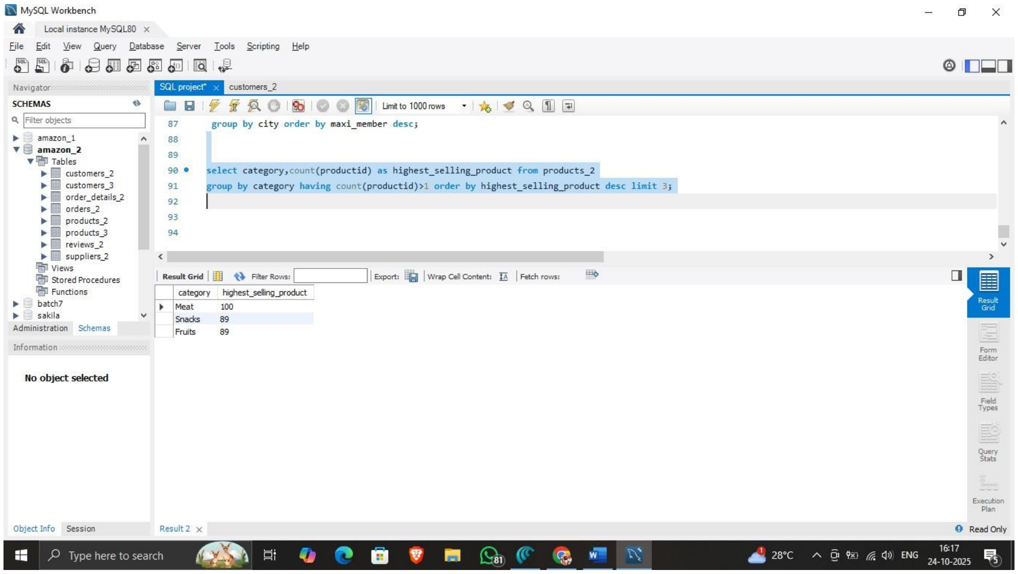
city	maxi_member
Lake Anna	2
West Steven	2
Ericview	2
Lake Jessica	2
Leonardchester	2
North Timothy	2
East Melissach...	2
Garciaside	2
West David	2
Davisfort	2
Floresmouth	2
Sarahmouth	2
Kristenmouth	2
Port Michael	2
Robertbury	2
North Jose	2
East Daniel	2
Port Denise	2

What are the top 3 most frequently ordered categories?

SELECT category,COUNT(productid) AS highest_selling_product FROM products_2 GROUP BY category HAVING COUNT(productid)>1 ORDER BY highest_selling_product DESC LIMIT 3;

Explanation

This query lists the top 3 most frequently ordered product categories.
It counts how many products belong to each category using COUNT(productid).
The HAVING clause filters only categories with more than one product.
ORDER BY DESC ranks them by total count, and LIMIT 3 shows the top three



```
 group by city order by maxi_member desc;



select category,count(productid) as highest_selling_product from products_2
group by category having count(productid)>1 order by highest_selling_product desc limit 3;
```

category	highest_selling_product
Meat	100
Snacks	89
Fruits	89