CS 411 – Project Task 1 (PT1) Task 3

Title	*Description*
Team Name	SelectStar
Project Title	Data-Driven Forecasting and Visualization of Fresh Food Demand in a Retail Store
Team Members	Mihir Shah (mihirss3@illinois.edu) Mohit Badve (mbadve2@illinois.edu) Prajakta Pikale (ppikale2@illinois.edu) Riya Tendulkar (rtend@illinois.edu)

Contents

1. DDL Commands for Creation of Tables

Below are the DDL Commands that have been used to create the tables in GCP:

1.1. Table 1: Users

```
-- User table
CREATE TABLE User (
   UserId INT PRIMARY KEY,
   EmailId VARCHAR(255),
   Password VARCHAR(255),
   Type VARCHAR(10),
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   PhoneNumber VARCHAR(20)
);
```

1.2. Table 2: Supplier

```
-- Supplier table
CREATE TABLE Supplier (
   SupplierId INT PRIMARY KEY,
   Name VARCHAR(100),
   Address VARCHAR(255),
   Contact VARCHAR(100)
);
```

1.3. Table 3: Category

```
-- Category table
CREATE TABLE Category (
   CategoryId INT PRIMARY KEY,
   Name VARCHAR(50),
   LeadTime VARCHAR(50),
   StorageRequirements VARCHAR(255)
);
```

1.4. Table 4: Product

```
-- Product table
CREATE TABLE Product (
   ProductId INT PRIMARY KEY,
   Name VARCHAR(100),
   PackagingType VARCHAR(50),
   Weight FLOAT,
   CategoryId INT,
   SupplierId INT,
   FOREIGN KEY (CategoryId) REFERENCES Category(CategoryId),
```

```
    FOREIGN KEY (SupplierId) REFERENCES Supplier(SupplierId)
  );
```

1.5. Table 5: Inventory

```
-- Inventory table
CREATE TABLE Inventory (
  InventoryId INT PRIMARY KEY,
  StockDate DATE,
  ProductId INT,
  UnitPrice FLOAT,
  ManufactureDate DATE,
  ExpiryDate DATE,
  Quantity INT,
  FOREIGN KEY (ProductId) REFERENCES Product(ProductId)
);
```

1.6. Table 6: Promotional Offer

```
-- PromotionalOffer table
CREATE TABLE PromotionalOffer (
  PromotionalOfferId INT PRIMARY KEY,
  StartDate DATE,
  EndDate DATE,
  DiscountRate FLOAT
);
```

1.7. Table 7: Order

```
-- Order table
CREATE TABLE `Order` (
  OrderId INT PRIMARY KEY,
  OrderDate DATE,
  TotalPrice FLOAT
);
```

1.8. Table 8: Forecast

```
-- Forecast table
CREATE TABLE Forecast (
  ForecastId INT PRIMARY KEY,
  ProductId INT,
  ForecastDate DATE,
  ForecastQuantity INT,
  FOREIGN KEY (ProductId) REFERENCES Product(ProductId)
);
```

1.9. Table 9: Order_Contains_Inventories

```
-- Order_Contains_Inventories table (link table between Order and Inventory)
CREATE TABLE Order_Contains_Inventories (
   OrderId INT,
   InventoryId INT,
   Quantity INT,
   PRIMARY KEY (OrderId, InventoryId),
   FOREIGN KEY (OrderId) REFERENCES `Order`(OrderId),
   FOREIGN KEY (InventoryId) REFERENCES Inventory(InventoryId)
);
```

1.10. Table 10: PromotionalOffers_AppliedOn_Order

```
-- PromotionalOffers_AppliedOn_Order table (link table between Order and
PromotionalOffer)
CREATE TABLE PromotionalOffers_AppliedOn_Order (
   OrderId INT,
   PromotionalOfferId INT,
   PRIMARY KEY (OrderId, PromotionalOfferId),
   FOREIGN KEY (OrderId) REFERENCES `Order`(OrderId),
   FOREIGN KEY (PromotionalOfferId) REFERENCES PromotionalOffer(PromotionalOfferId)
);
```

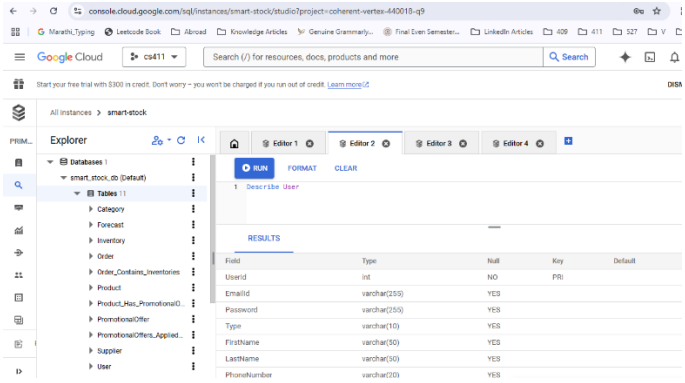
1.11. Table 11: Product_Has_PromotionalOffers

```
-- Product_Has_PromotionalOffers table (link table between Product and
PromotionalOffer)
CREATE TABLE Product_Has_PromotionalOffers (
   ProductId INT,
   PromotionalOfferId INT,
   PRIMARY KEY (ProductId, PromotionalOfferId),
   FOREIGN KEY (ProductId) REFERENCES Product(ProductId),
   FOREIGN KEY (PromotionalOfferId) REFERENCES PromotionalOffer(PromotionalOfferId)
)
```

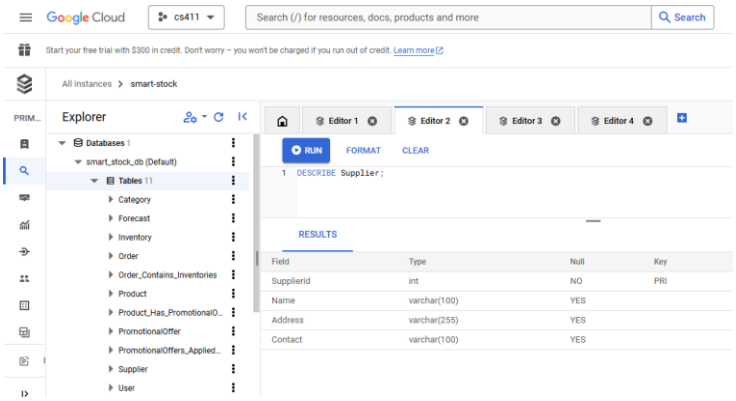
2. Created Tables from Relational Schema

We have implemented the schema on GCP and below are the screenshots for the created tables:

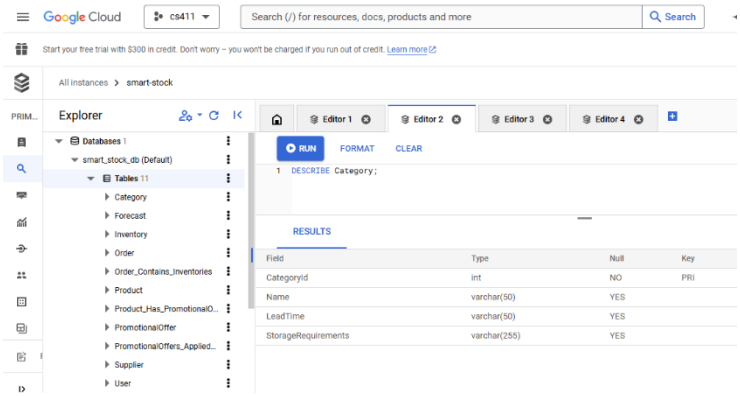
2.1. Table 1: Users



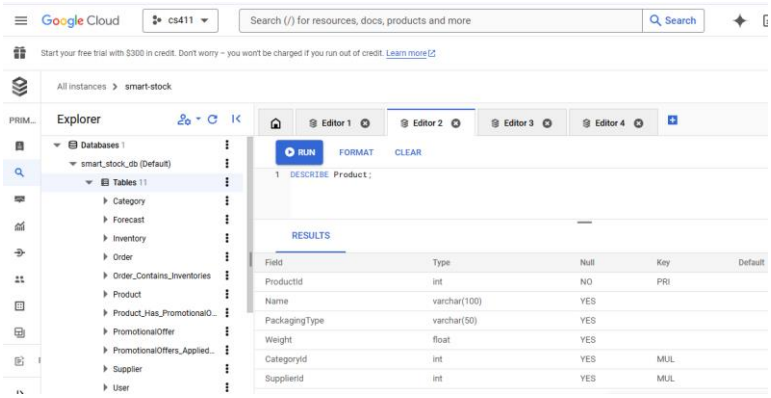
2.2. Table 2: Supplier



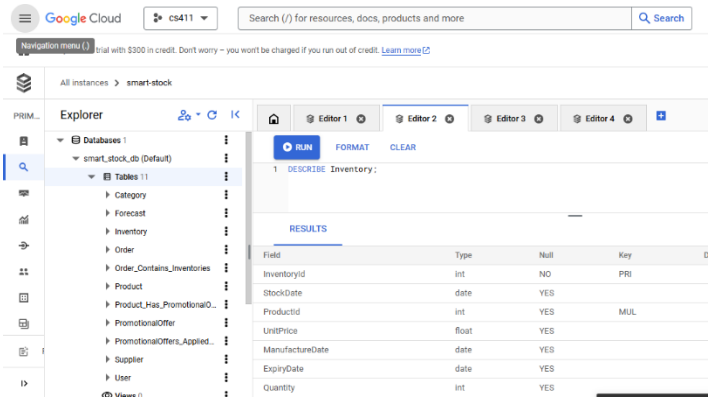
2.3. Table 3: Category



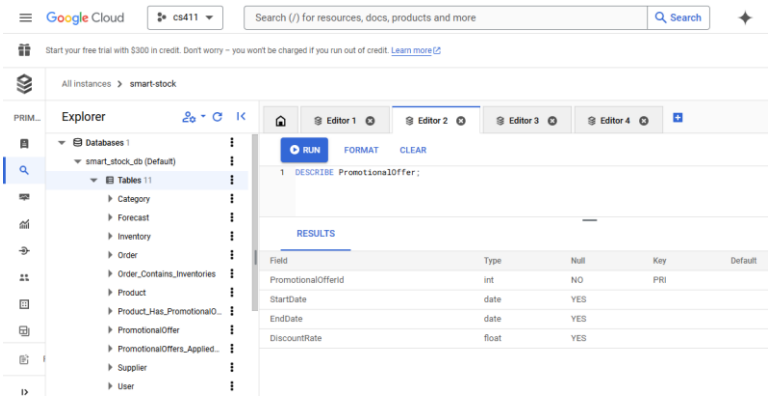
2.4. Table 4: Product



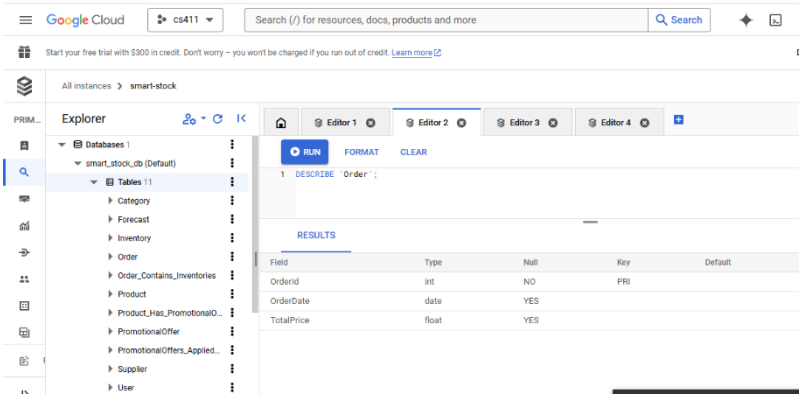
2.5. Table 5: Inventory



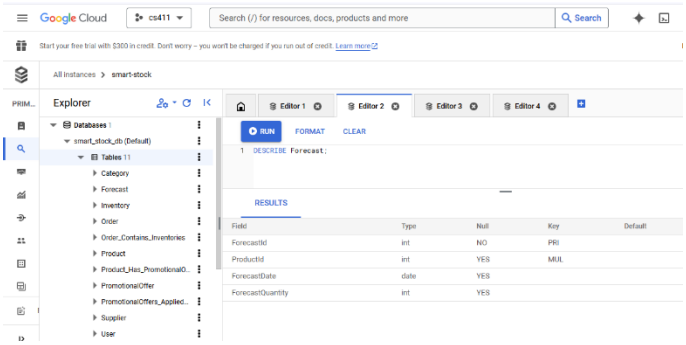
2.6. Table 6: Promotional Offer



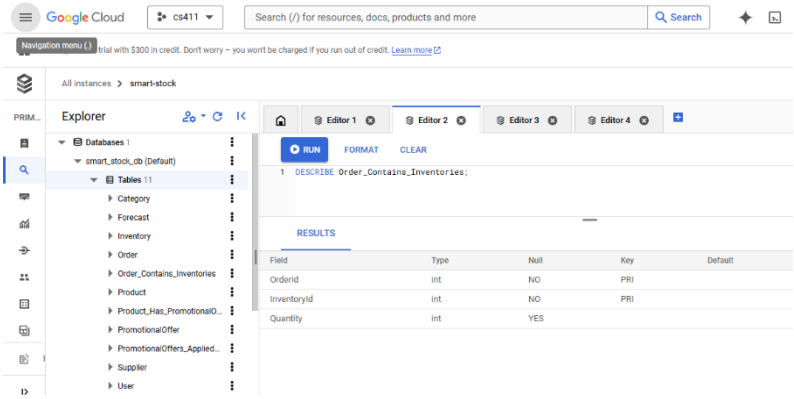
2.7. Table 7: Order



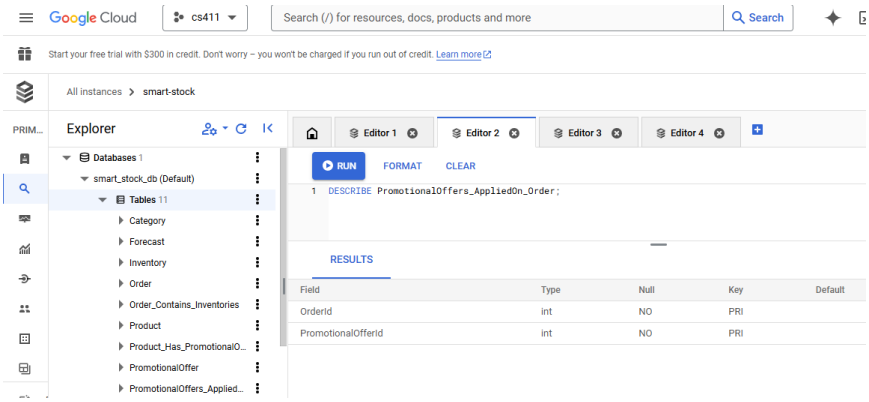
2.8. Table 8: Forecast



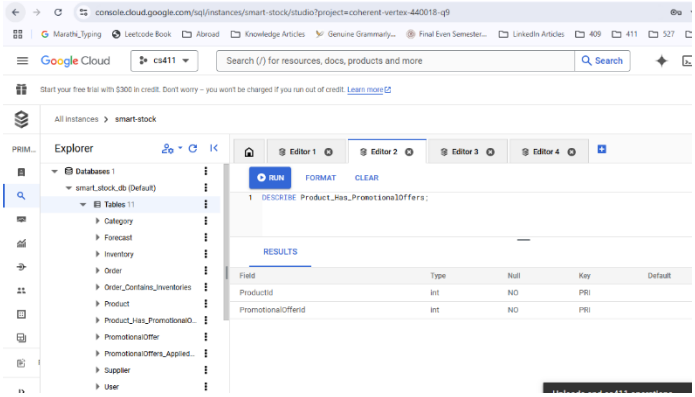
2.9. Table 9: Order_Contains_Inventories



2.10. Table 10: PromotionalOffers_AppliedOn_Order



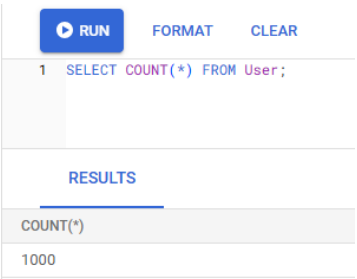
2.11. Table 11: Product_Has_PromotionalOffers

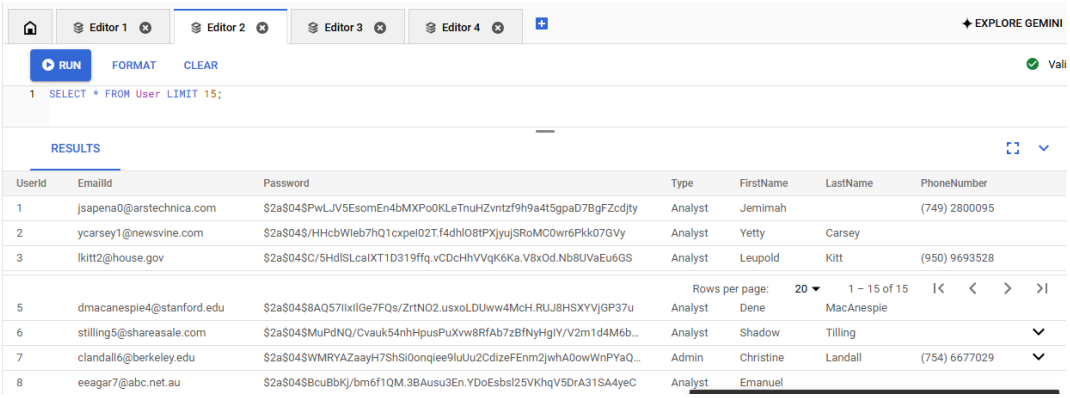


3. Data in Tables

Below are screenshots of the count(*) and top 15 rows of the data.

3.1. Table 1: Users





UserId	EmailId	Password	Type	FirstName	LastName	PhoneNumber
1	jsapena0@arstechnica.com	$2a$04$PwLJV5EsomEn4bMXPo0KLeTnuHZvntzf9h9a4t5gpaD7BgFZcdjty	Analyst	Jemimah		(749) 2800095
2	ycarsey1@newsvine.com	$2a$04$/HHcbWleb7hQ1cxpel02T.f4dhlO8tPXjyujSRoMC0wr6Pkk07GVy	Analyst	Yetty	Carsey	
3	lkitt2@house.gov	$2a$04$C/5HdlSLcaIXT1D319ffq.vCDcHhVVqK6Ka.V8xOd.Nb8UVaEu6GS	Analyst	Leupold	Kitt	(950) 9693528
5	dmacanespie4@stanford.edu	$2a$04$8AQ57IlxIlGe7FQs/ZrtNO2.usxoLDUww4McH.RUJ8HSXYVjGP37u	Analyst	Dene	MacAnespie	
6	stilling5@shareasale.com	$2a$04$MuPdNQ/Cvauk54nhHpusPuXvw8RfAb7zBfNyHgIY/V2m1d4M6b...	Analyst	Shadow	Tilling	
7	clandall6@berkeley.edu	$2a$04$WMRYAZaayH7ShSi0onqiee9luUu2CdizeFEnm2jwhA0owWnPYaQ...	Admin	Christine	Landall	(754) 6677029
8	eeagar7@abc.net.au	$2a$04$BcuBbKj/bm6f1QM.3BAusu3En.YDoEsbsl25VKhqV5DrA31SA4yeC	Analyst	Emanuel		

3.2. Table 2: Supplier

Editor 1 | Editor 2

RUN FORMAT CLEAR

```
1 SELECT COUNT(*) FROM Supplier;
```

RESULTS

COUNT(*)
1000

Editor 1 | Editor 2 | Editor 3 | Editor 4 | Editor 5 | EXPLORE GEM

RUN FORMAT CLEAR

```
1 SELECT * FROM Supplier LIMIT 15;
```

RESULTS

SupplierId	Name	Address	Contact
1	Gusikowski and Sons	PO Box 76081	(164) 4380733
2	Satterfield-Oberbrunner	PO Box 39346	(422) 3717614
3	Kassulke LLC	Apt 484	(451) 6292990
4	Paucek-Emmerich	Room 122	(925) 1107136
5	Nikolaus LLC	PO Box 44332	(285) 2009829
6	Hand LLC	Room 91	(414) 4542184
7	West Group	Suite 61	(551) 3400128

3.3. Table 3: Category

Editor 1 | Editor 2

RUN FORMAT CLEAR

```
1 SELECT COUNT(*) FROM Category;
```

RESULTS

COUNT(*)
42

Editor 1 | Editor 2 | Editor 3 | Editor 4 | Editor 5 | EXPLORE GE

RUN FORMAT CLEAR

```
1 SELECT * FROM Category LIMIT 15;
```

RESULTS

CategoryId	Name	LeadTime	StorageRequirements
1	Fruits	5d	Store in a cool, dry place or refrigerate depending on type
2	Vegetables	3d	Refrigerate in a crisper drawer or store in a cool, dry place
3	Fresh Herbs	3h	Refrigerate with stems in water or wrapped in damp paper towels
4	Leafy Greens	1d	Refrigerate in a crisper drawer with high humidity
5	Dairy Products	7d	Refrigerate at or below 4°C (39°F)
6	Meat	2d	Store in the coldest part of the refrigerator, wrapped to avoid contamination
7	Poultry	2d	Refrigerate at or below 4°C (39°F) in a covered container

Rows per page: 20 1 – 15 of 15

3.4. Table 4: Product

Editor 1 | Editor 2

RUN FORMAT CLEAR

```
1 SELECT COUNT(*) FROM Product;
```

RESULTS

COUNT(*)
450

Editor 1 | Editor 2 | Editor 3 | Editor 4 | Editor 5 | EXPL

RUN FORMAT CLEAR

```
1 SELECT * FROM Product LIMIT 15;
```

RESULTS

ProductId	Name	PackagingType	Weight	CategoryId	SupplierId
1	Corn Kernels - Frozen	Bag	1	2	493
2	Wine - Tribal Sauvignon	Bottle	0.75	12	999
3	Chicken - Diced, Cooked	Pack	2	7	517
4	Piping - Bags Quizna	Bag	0.5	36	586
5	Eggplant - Regular	Each	0.5	2	377
6	Shrimp - 16/20, Peeled Deviened	Pack	1.5	8	127
7	Whmis Spray Bottle Graduated	Bottle	0.3	37	504

3.5. Table 5: Inventory

Editor 1 | Editor 2

RUN FORMAT CLEAR

```
1 SELECT COUNT(*) FROM Inventory;
```

RESULTS

COUNT(*)
1000

Editor 1 | Editor 2 | Editor 3 | Editor 4 | Editor 5 | EXPLO

RUN FORMAT CLEAR

```
1 SELECT * FROM Inventory LIMIT 15;
```

RESULTS

InventoryId	StockDate	ProductId	UnitPrice	ManufactureDate	ExpiryDate	Quantity
1	2024-07-15	445	89.27	2024-09-16	2024-01-23	17
2	2024-10-13	359	1.45	2024-06-04	2023-10-03	8
3	2024-09-12	90	45.84	2024-01-14	2024-06-24	12
4	2024-09-11	379	12.53	2023-09-14	2024-03-25	27
5	2024-03-21	284	11.72	2024-01-06	2023-09-07	1
6	2024-06-15	26	82.42	2024-10-14	2024-09-15	27
7	2024-02-06	302	17.13	2024-05-15	2023-10-29	22

3.6. Table 6: Order

Editor 1 | Editor 2

RUN FORMAT CLEAR

```
1 SELECT COUNT(*) FROM `Order`;
```

RESULTS

COUNT(*)
1000

Editor 1 | Editor 2 | Editor 3 | Editor 4 | Editor 5

RUN FORMAT CLEAR

```
1 SELECT * FROM `Order` LIMIT 15;
```

RESULTS

OrderId	OrderDate	TotalPrice
1	2023-09-05	12.92
2	2024-10-29	48.17
3	2024-06-16	55.97
4	2024-03-28	7.08
5	2024-10-07	37.05
6	2024-09-12	35.55
7	2024-03-23	12.65

3.7. Table 7: Order_Contains_Inventories

Editor 1 | Editor 2 | Editor 3

RUN FORMAT CLEAR

```
1 SELECT COUNT(*) FROM Order_Contains_Inventories;
```

RESULTS

COUNT(*)
1000

All instances > smart-stock

Editor 1 | Editor 2 | Editor 3 | Editor 4 | Editor 5

RUN FORMAT CLEAR

```
1 SELECT * FROM Order_Contains_Inventories LIMIT 15;
```

RESULTS

OrderId	InventoryId	Quantity
2	362	19
3	299	29
4	196	9
5	228	19
6	922	4
9	581	6
10	965	25

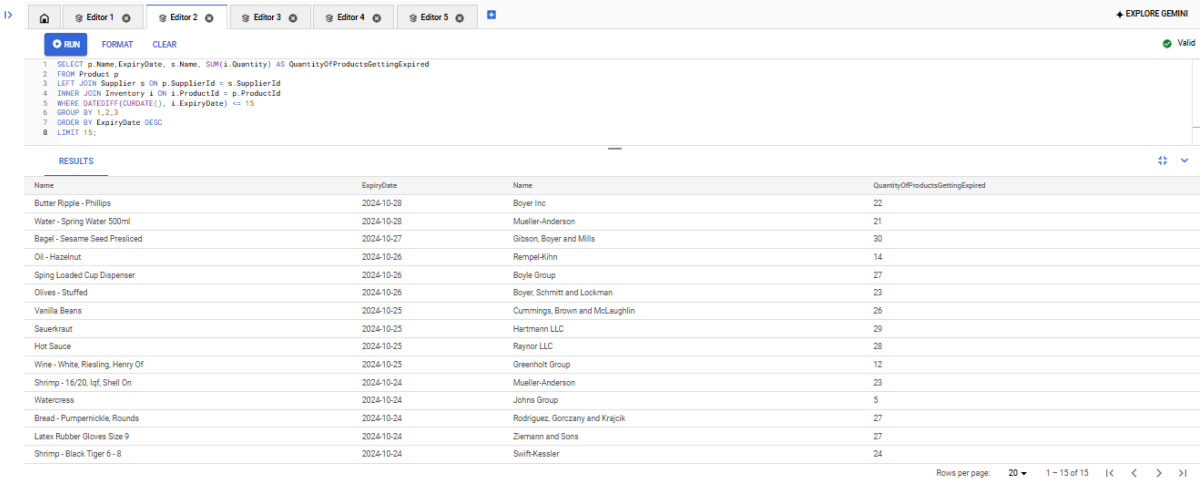
4. Advanced SQL Queries

4.1. Objective: To find the products which are expiring soon (threshold of 15 days from current date) and their required quantity which is to be replaced.

Code:

```
SELECT p.Name,ExpiryDate, s.Name, SUM(i.Quantity) AS QuantityOfProductsGettingExpired
FROM Product p
LEFT JOIN Supplier s ON p.SupplierId = s.SupplierId
INNER JOIN Inventory i ON i.ProductId = p.ProductId
WHERE DATEDIFF(CURDATE(), i.ExpiryDate) <= 15
GROUP BY 1,2,3
ORDER BY ExpiryDate DESC
LIMIT 15;
```

Result:



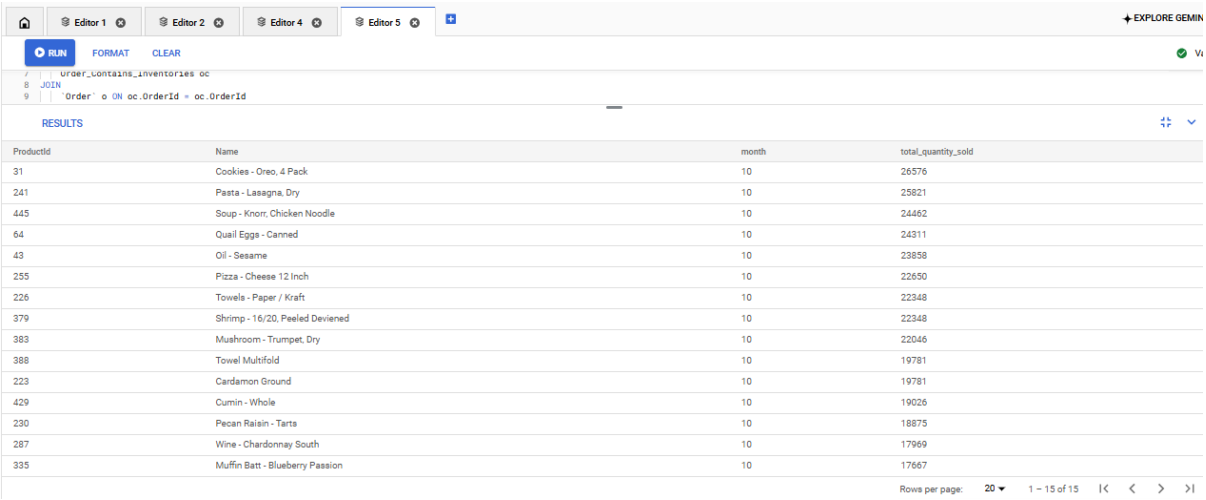
Name	ExpiryDate	Name	QuantityOfProductsGettingExpired
Butter Ripple - Phillips	2024-10-28	Boyer Inc	22
Water - Spring Water 500ml	2024-10-28	Mueller-Anderson	21
Bagel - Sesame Seed Presliced	2024-10-27	Gibson, Boyer and Mills	30
Oil - Hazelnut	2024-10-26	Rempel-Kihn	14
Sping Loaded Cup Dispenser	2024-10-26	Boyle Group	27
Olives - Stuffed	2024-10-26	Boyer, Schmitt and Lockman	23
Vanilla Beans	2024-10-25	Cummings, Brown and McLaughlin	26
Sauerkraut	2024-10-25	Hartmann LLC	29
Hot Sauce	2024-10-25	Raynor LLC	28
Wine - White, Riesling, Henry Of	2024-10-25	Greenholt Group	12
Shrimp - 16/20, Iqf, Shell On	2024-10-24	Mueller-Anderson	23
Watercress	2024-10-24	Johns Group	5
Bread - Pumpernickle, Rounds	2024-10-24	Rodriguez, Gorczany and Krajcik	27
Latex Rubber Gloves Size 9	2024-10-24	Ziemann and Sons	27
Shrimp - Black Tiger 6 - 8	2024-10-24	Swift-Kessler	24

4.2. Objective: To find the TOP 15 most ordered products in a month.

Code:

```
SELECT
   p.ProductId,
   p.Name,
   MONTH(o.OrderDate) AS month,
   SUM(oc.quantity) AS total_quantity_sold
FROM
   Order_Contains_Inventories oc
JOIN
   `Order` o ON oc.OrderId = oc.OrderId
JOIN
   Inventory i ON i.InventoryId = oc.InventoryId
JOIN
   Product p ON i.ProductId = p.ProductId
WHERE MONTH(o.OrderDate) = MONTH(CURDATE())
GROUP BY
   1, 2, 3
ORDER BY
   month, total_quantity_sold DESC
LIMIT 15;
```

Result:



Editor 1 | Editor 2 | Editor 4 | Editor 5 | EXPLORE GEMIN

RUN FORMAT CLEAR

```
7     Order_Contains_Inventories oc
8 JOIN
9     `Order` o ON oc.OrderId = oc.OrderId
```

RESULTS

ProductId	Name	month	total_quantity_sold
31	Cookies - Oreo, 4 Pack	10	26576
241	Pasta - Lasagna, Dry	10	25821
445	Soup - Knorr, Chicken Noodle	10	24462
64	Quail Eggs - Canned	10	24311
43	Oil - Sesame	10	23858
255	Pizza - Cheese 12 Inch	10	22650
226	Towels - Paper / Kraft	10	22348
379	Shrimp - 16/20, Peeled Deviened	10	22348
383	Mushroom - Trumpet, Dry	10	22046
388	Towel Multifold	10	19781
223	Cardamon Ground	10	19781
429	Cumin - Whole	10	19026
230	Pecan Raisin - Tarts	10	18875
287	Wine - Chardonnay South	10	17969
335	Muffin Batt - Blueberry Passion	10	17667

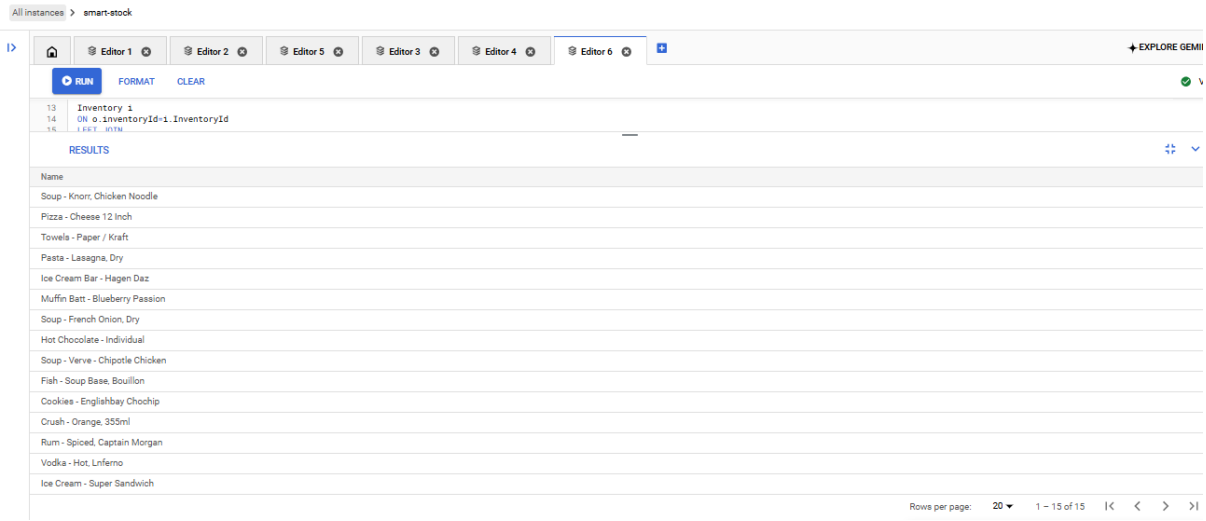
Rows per page: 20 1 – 15 of 15

4.3. Objective: Finding the top 15 Products (by sales) of the Category having the most sales.

Code:

```
SELECT p.Name FROM
Order_Contains_Inventories o
LEFT JOIN
Inventory i
ON o.inventoryId=i.InventoryId
LEFT JOIN
Product p
ON p.ProductId=i.ProductId WHERE p.CategoryId =
(
  SELECT c.CategoryID FROM
  Order_Contains_Inventories o
  LEFT JOIN
  Inventory i
  ON o.inventoryId=i.InventoryId
  LEFT JOIN
  Product p
  ON p.ProductId=i.ProductId
  LEFT JOIN
Category c
  ON p.CategoryId=c.CategoryId
  GROUP BY c.CategoryId
  ORDER BY SUM(o.Quantity) DESC
  LIMIT 1
)
GROUP BY 1
ORDER BY SUM(o.Quantity*i.UnitPrice) DESC
LIMIT 15;
```

Result:

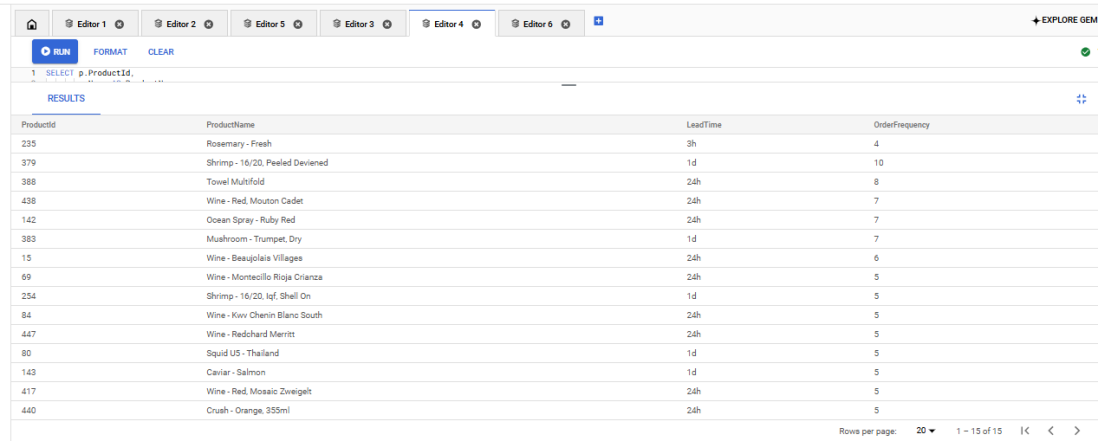


4.4. Objective: To find the top 15 Products based on order frequency, ordered by the LeadTime of their Category in descending order.

Code:

```
SELECT p.ProductId,  p.Name AS ProductName, c.LeadTime, COUNT(oci.OrderId) AS OrderFrequency
FROM Product p
JOIN Category c ON p.CategoryId = c.CategoryId
JOIN Inventory i ON p.ProductId = i.ProductId
JOIN Order_Contains_Inventories oci ON i.InventoryId = oci.InventoryId
JOIN `Order` o ON oci.OrderId = o.OrderId
GROUP BY p.ProductId, p.Name, c.LeadTime
ORDER BY
CASE
    WHEN c.LeadTime LIKE '%d' THEN
CAST(SUBSTRING(c.LeadTime, 1, LENGTH(c.LeadTime) - 1) AS UNSIGNED) * 24
    WHEN c.LeadTime LIKE '%h' THEN
CAST(SUBSTRING(c.LeadTime, 1, LENGTH(c.LeadTime) - 1) AS UNSIGNED)
    ELSE 0
END,
   OrderFrequency DESC
LIMIT 15;
```

Result:



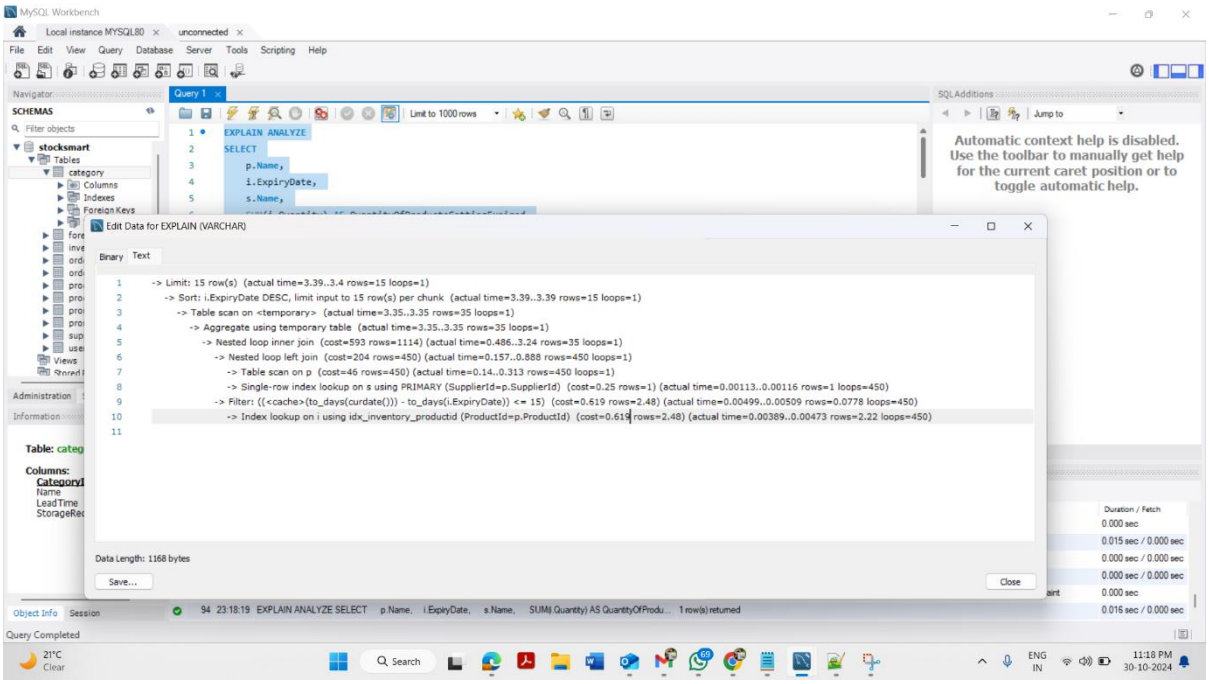
ProductId	ProductName	LeadTime	OrderFrequency
235	Rosemary - Fresh	3h	4
379	Shrimp - 16/20, Peeled Deviened	1d	10
388	Towel Multifold	24h	8
438	Wine - Red, Mouton Cadet	24h	7
142	Ocean Spray - Ruby Red	24h	7
383	Mushroom - Trumpet, Dry	1d	7
15	Wine - Beaujolais Villages	24h	6
69	Wine - Montecillo Rioja Crianza	24h	5
254	Shrimp - 16/20, Iqf, Shell On	1d	5
84	Wine - Kwv Chenin Blanc South	24h	5
447	Wine - Redchard Merritt	24h	5
80	Squid U5 - Thailand	1d	5
143	Caviar - Salmon	1d	5
417	Wine - Red, Mosaic Zweigelt	24h	5
440	Crush - Orange, 355ml	24h	5

5. Indexing Analysis

5.1. Complex Query 1

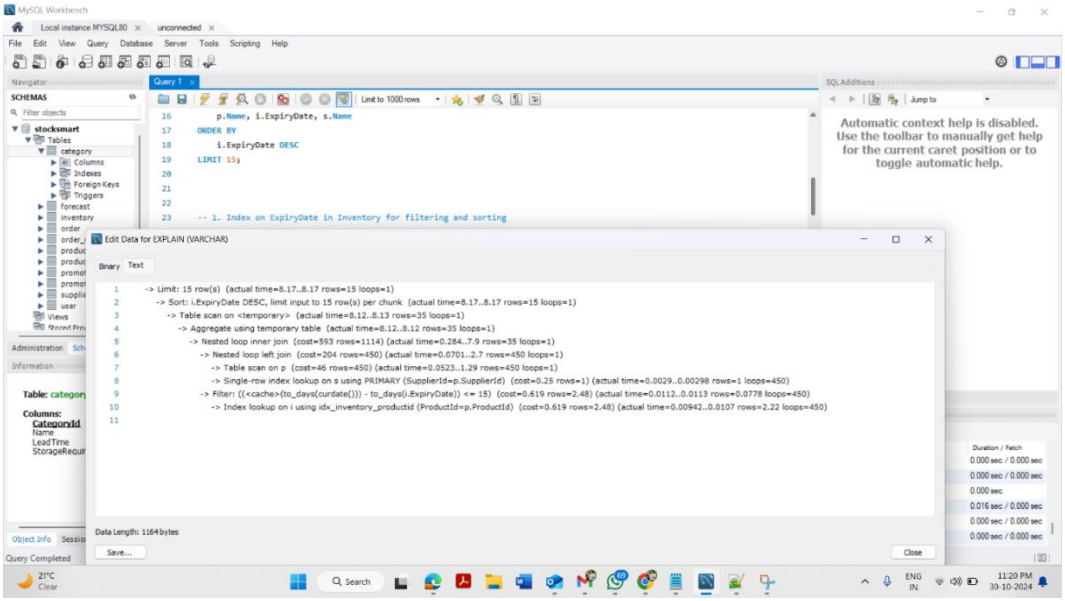
- Baseline:

```
EXPLAIN ANALYZE
SELECT
   p.Name,
   i.ExpiryDate,
   s.Name,
   SUM(i.Quantity) AS QuantityOfProductsGettingExpired
FROM
   Product p
LEFT JOIN
   Supplier s ON p.SupplierId = s.SupplierId
INNER JOIN
   Inventory i ON i.ProductId = p.ProductId
WHERE
   DATEDIFF(CURDATE(), i.ExpiryDate) <= 15
GROUP BY
   p.Name, i.ExpiryDate, s.Name
ORDER BY
   i.ExpiryDate DESC
LIMIT 15;
```



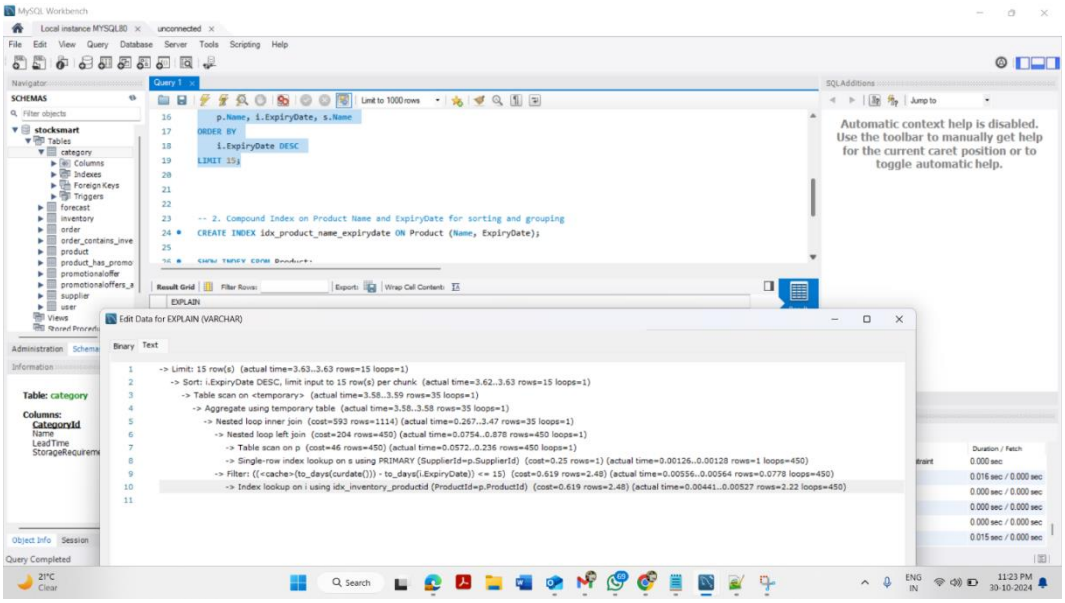
- Index Design 1: Index on ExpiryDate in Inventory for filtering and sorting

```
CREATE INDEX idx_inventory_expirydate ON Inventory (ExpiryDate);
```



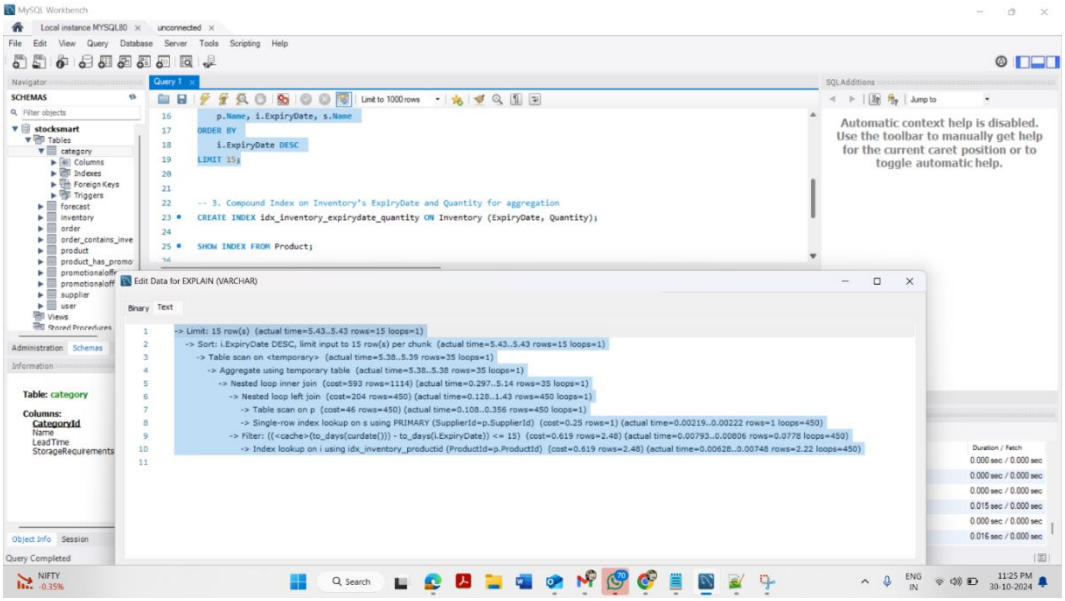
- Index Design 2: Compound Index on Product Name and ExpiryDate for sorting and grouping

  CREATE INDEX idx_product_name_expirydate ON Product (Name, ExpiryDate);



- Index Design 3: Compound Index on Inventory's ExpiryDate and Quantity for aggregation

  CREATE INDEX idx_inventory_expirydate_quantity ON Inventory (ExpiryDate, Quantity);

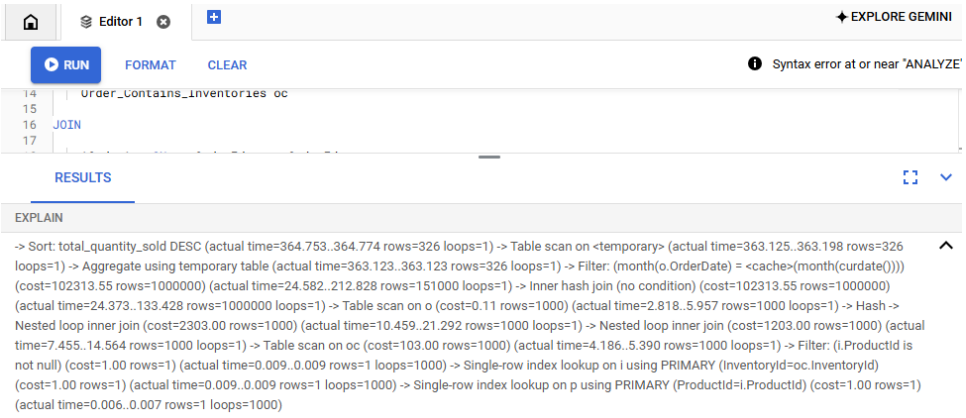


- Conclusion for Query 1: Since the indexes did not improve query cost, they are not recommended for Query 1.

5.2. Complex Query 2

- **Baseline:**

```
EXPLAIN ANALYZE
SELECT
   p.ProductId,
   p.Name,
   MONTH(o.OrderDate) AS month,
   SUM(oc.quantity) AS total_quantity_sold
FROM
   Order_Contains_Inventories oc
JOIN
   `Order` o ON oc.OrderId = oc.OrderId
JOIN
   Inventory i ON i.InventoryId = oc.InventoryId
JOIN
   Product p ON i.ProductId = p.ProductId
WHERE MONTH(o.OrderDate) = MONTH(CURDATE())
GROUP BY
   1, 2, 3
ORDER BY
   month, total_quantity_sold DESC;
```

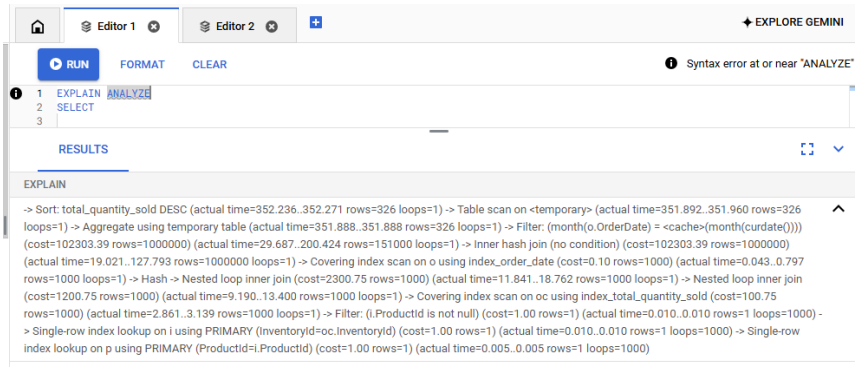


- **Index Design 1:** OrderDate and Quantity Sold
  - We have selected these two columns for the index because they are used in both where condition and to order by the results.

```
CREATE INDEX index_order_date
ON `Order` (OrderDate);

CREATE INDEX index_total_quantity_sold
ON Order_Contains_Inventories (Quantity);
```

Result:



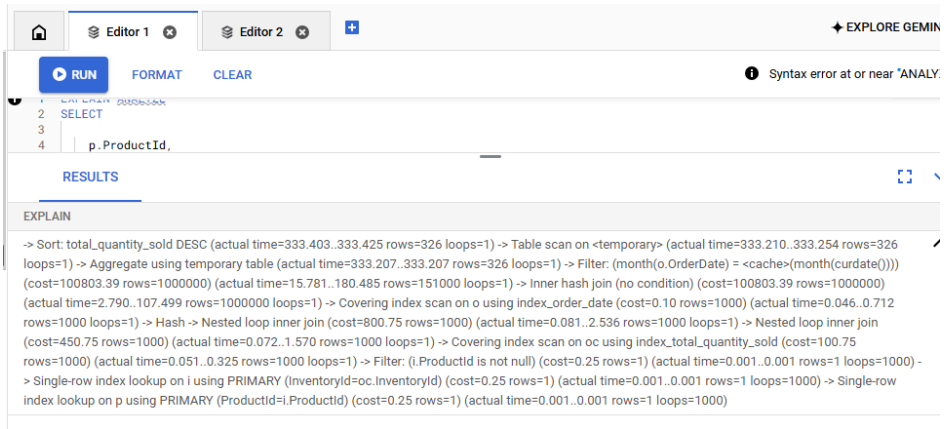
Observations: There is a decrease in cost but the cost hasn't decreased significantly.

- **Index Design 2:** OrderDate, Quantity and Name

```
CREATE INDEX index_order_date
ON `Order` (OrderDate);

CREATE INDEX index_total_quantity_sold
ON Order_Contains_Inventories (Quantity);

CREATE INDEX index_product_name
ON Product (Name);
```



Index Design 3: (OrderDate,OrderId), (OrderId, InventoryId) and (InventoryId, ProductId)

```
CREATE INDEX idx_order_orderdate ON `Order` (OrderDate, OrderId);
```

CREATE INDEX idx_order_contains_inventories ON Date, Order_Contains_Inventories (OrderId, InventoryId);

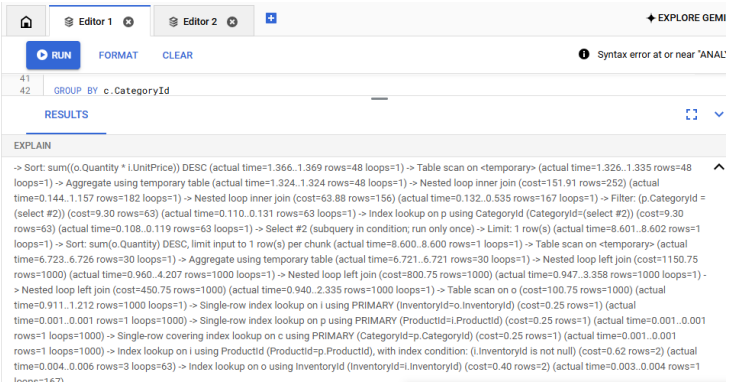
CREATE INDEX idx_inventory_product ON Id, Inventory (InventoryId, ProductId);

- Conclusion for Query 2: We are selecting Design 3 as the indexing strategy as we can see that the cost is decreasing from the original cost and also, we are using single-column index on frequently queried columns and are using composite index with filtering columns

5.3. Complex Query 3

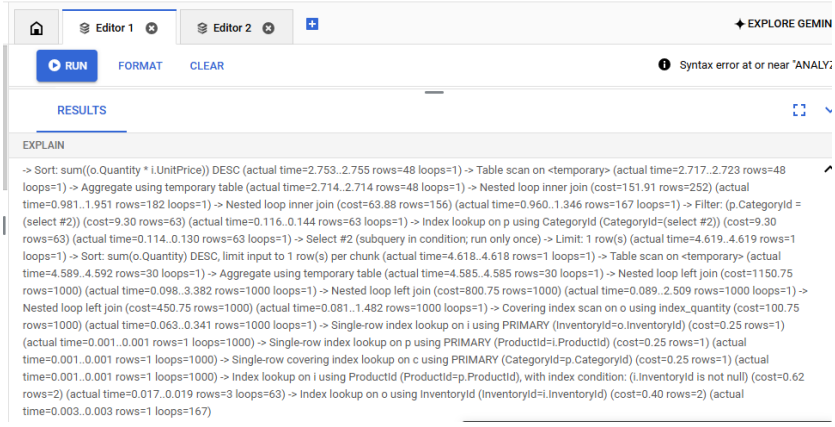
- Baseline:

```
EXPLAIN ANALYZE
SELECT p.Name FROM
Order_Contains_Inventories o
LEFT JOIN
Inventory i
ON o.inventoryId=i.InventoryId
LEFT JOIN
Product p
ON p.ProductId=i.ProductId WHERE p.CategoryId =
(
 SELECT c.CategoryID FROM
 Order_Contains_Inventories o
 LEFT JOIN
 Inventory i
 ON o.inventoryId=i.InventoryId
 LEFT JOIN
 Product p
 ON p.ProductId=i.ProductId
 LEFT JOIN
Category c
 ON p.CategoryId=c.CategoryId
 GROUP BY c.CategoryId
 ORDER BY SUM(o.Quantity) DESC
 LIMIT 1
)
GROUP BY 1
ORDER BY SUM(o.Quantity*i.UnitPrice) DESC ;
```



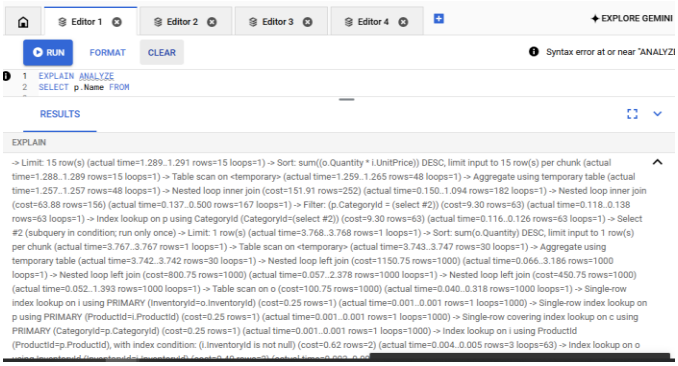
- **Index Design 1:** Quantity

CREATE INDEX index_quantity ON Order_Contains_Inventories(Quantity);



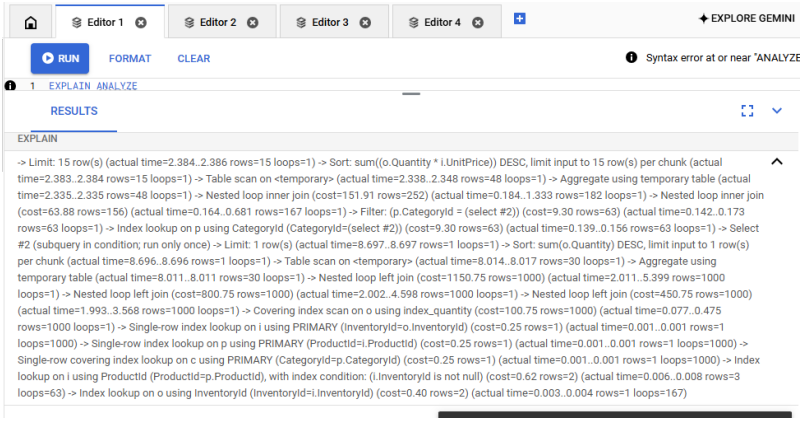
- **Index Design 2:** Product Name

CREATE INDEX index_price ON Inventory(UnitPrice);



- **Index Design 3:**

CREATE INDEX index_product ON Product(Name);
CREATE INDEX index_quantity ON Order_Contains_Inventories(Quantity);
CREATE INDEX index_price ON Inventory(UnitPrice);



- **Conclusion for Query 3:** All designs are giving the same cost results with no significant improvements and hence we do not need any indexing techniques for this query.

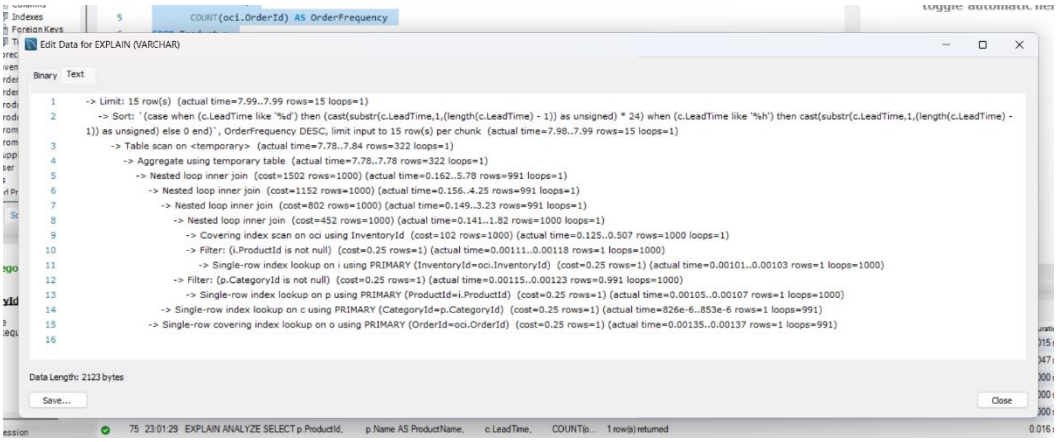
5.4. Complex Query 4:

- Baseline:

```
EXPLAIN ANALYZE
    SELECT p.ProductId,
    p.Name AS ProductName,
    c.LeadTime,
    COUNT(oci.OrderId) AS OrderFrequency
FROM Product p
JOIN Category c ON p.CategoryId = c.CategoryId
JOIN Inventory i ON p.ProductId = i.ProductId
JOIN Order_Contains_Inventories oci ON i.InventoryId = oci.InventoryId
JOIN Order o ON oci.OrderId = o.OrderId
GROUP BY p.ProductId, p.Name, c.LeadTime
ORDER BY
  CASE
    WHEN c.LeadTime LIKE '%d' THEN
      CAST(SUBSTRING(c.LeadTime, 1, LENGTH(c.LeadTime) - 1) AS UNSIGNED) * 24
    WHEN c.LeadTime LIKE '%h' THEN
      CAST(SUBSTRING(c.LeadTime, 1, LENGTH(c.LeadTime) - 1) AS UNSIGNED)
    ELSE 0
  END,
```

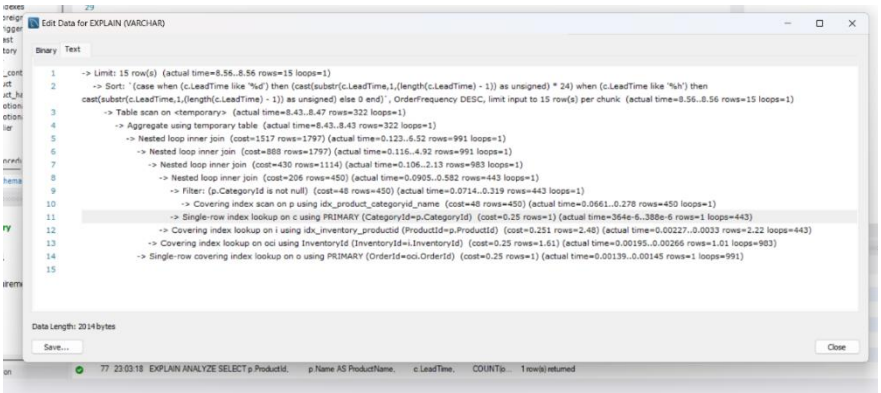
OrderFrequency DESC

LIMIT 15;



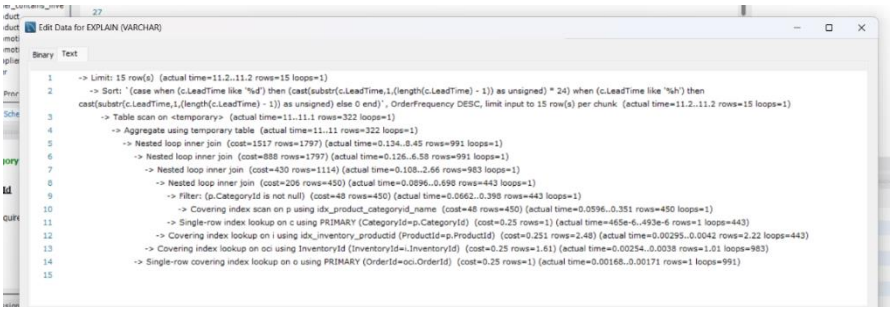
- Index Design 1: Index on Name to speed up grouping and ordering by product name

  CREATE INDEX idx_product_name ON Product (Name);



- Index Design 2: -- Index on LeadTime to support sorting by lead time in ascending order

  CREATE INDEX idx_category_leadtime ON Category (LeadTime);



- **Index Design 3:** Index on UnitPrice to aid in calculating total sales by category

  CREATE INDEX idx_inventory_unitprice ON Inventory (UnitPrice);

Edit Data for EXPLAIN (VARCHAR)

Binary Text

```
-> Limit: 15 row(s)  (actual time=10.8..10.8 rows=15 loops=1)
    -> Sort: `(case when (c.LeadTime like '%d') then (cast(substr(c.LeadTime,1,(length(c.LeadTime) - 1)) as unsigned) * 24) when (c.LeadTime like '%h') then
cast(substr(c.LeadTime,1,(length(c.LeadTime) - 1)) as unsigned) else 0 end)`, OrderFrequency DESC, limit input to 15 row(s) per chunk  (actual time=10.7..10.8 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=10.6..10.7 rows=322 loops=1)
            -> Aggregate using temporary table  (actual time=10.6..10.6 rows=322 loops=1)
                -> Nested loop inner join  (cost=1517 rows=1797) (actual time=0.118..7.42 rows=991 loops=1)
                    -> Nested loop inner join  (cost=888 rows=1797) (actual time=0.113..5.79 rows=991 loops=1)
                        -> Nested loop inner join  (cost=430 rows=1114) (actual time=0.104..2.57 rows=983 loops=1)
                            -> Nested loop inner join  (cost=206 rows=450) (actual time=0.0904..0.735 rows=443 loops=1)
                                -> Filter: (p.CategoryId is not null)  (cost=48 rows=450) (actual time=0.0757..0.479 rows=443 loops=1)
                                    -> Covering index scan on p using idx_product_categoryid_name  (cost=48 rows=450) (actual time=0.0713..0.394 rows=450 loops=1)
                                -> Single-row index lookup on c using PRIMARY (CategoryId=p.CategoryId)  (cost=0.25 rows=1) (actual time=360e-6..382e-6 rows=1 loops=443)
                            -> Covering index lookup on i using idx_inventory_productid (ProductId=p.ProductId)  (cost=0.251 rows=2.48) (actual time=0.00271..0.0039 rows=2.22 loops=443)
                        -> Covering index lookup on oci using InventoryId (InventoryId=i.InventoryId)  (cost=0.25 rows=1.61) (actual time=0.0023..0.00311 rows=1.01 loops=983)
                    -> Single-row covering index lookup on o using PRIMARY (OrderId=oci.OrderId)  (cost=0.25 rows=1) (actual time=0.00144..0.00147 rows=1 loops=991)
```

- Conclusion for Query 4:
  No additional indexes are recommended for Query 4, as they did not yield performance improvements. This analysis highlights the importance of assessing both positive and negative impacts of indexing when working with complex sorting and grouping logic.