

## **CS 411 – Project Task 1 (PT1) Task 3**

<b><i>Title</i></b>	<b><i>Description</i></b>
<i>Team Name</i>	SelectStar
<i>Project Title</i>	Data-Driven Forecasting and Visualization of Fresh Food Demand in a Retail Store
<i>Team Members</i>	Mihir Shah ( <a href="mailto:mihirss3@illinois.edu">mihirss3@illinois.edu</a> ) Mohit Badve ( <a href="mailto:mbadve2@illinois.edu">mbadve2@illinois.edu</a> ) Prajakta Pikale ( <a href="mailto:ppikale2@illinois.edu">ppikale2@illinois.edu</a> ) Riya Tendulkar ( <a href="mailto:rtend@illinois.edu">rtend@illinois.edu</a> )

### **Contents**

1. DDL Commands for Creation of Tables.....	2
2. Created Tables from Relational Schema.....	4
3. Data in Tables.....	8
4. Advanced SQL Queries .....	11
5. Indexing Analysis .....	15

## 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

1 Describe User;

Field	Type	Null	Key	Default
UserId	int	NO	PRI	
Emailid	varchar(255)	YES		
Password	varchar(255)	YES		
Type	varchar(10)	YES		
Firstname	varchar(30)	YES		
Lastname	varchar(30)	YES		
PhoneNumber	varchar(20)	YES		

2.2. Table 2: Supplier

1 DESCRIBE Supplier;

Field	Type	Null	Key
Supplierid	int	NO	PRI
Name	varchar(100)	YES	
Address	varchar(255)	YES	
Contact	varchar(100)	YES	

2.3. Table 3: Category

1 DESCRIBE Category;

Field	Type	Null	Key
Categoryid	int	NO	PRI
Name	varchar(50)	YES	
LeadTime	varchar(50)	YES	
StorageRequirements	varchar(255)	YES	

2.4. Table 4: Product

Google Cloud cs411 Search (/) for resources, docs, products and more

Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

All instances > smart-stock

PRIM... Explorer

Databases 1

smart\_stock\_db (Default)

Tables 11

Category

Forecast

Inventory

Order

Order\_Contains\_Inventories

Product

Product\_Has\_PromotionalO...

PromotionalOffer

PromotionalOffers\_Applied...

Supplier

User

RUN FORMAT CLEAR

1 DESCRIBE Product;

RESULTS

Field	Type	Null	Key	Default
Productid	int	NO	PRI	
Name	varchar(100)	YES		
PackagingType	varchar(50)	YES		
Weight	float	YES		
Categoryid	int	YES	MUL	
Supplierid	int	YES	MUL	

## 2.5. Table 5: Inventory

Google Cloud cs411 Search (/) for resources, docs, products and more

Navigation menu (1) trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

All instances > smart-stock

PRIM... Explorer

Databases 1

smart\_stock\_db (Default)

Tables 11

Category

Forecast

Inventory

Order

Order\_Contains\_Inventories

Product

Product\_Has\_PromotionalO...

PromotionalOffer

PromotionalOffers\_Applied...

Supplier

User

RUN FORMAT CLEAR

1 DESCRIBE Inventory;

RESULTS

Field	Type	Null	Key	Default
Inventoryid	int	NO	PRI	
StockDate	date	YES		
Productid	int	YES	MUL	
UnitPrice	float	YES		
ManufactureDate	date	YES		
ExpiryDate	date	YES		
Quantity	int	YES		

## 2.6. Table 6: Promotional Offer

Google Cloud cs411 Search (/) for resources, docs, products and more

Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

All instances > smart-stock

PRIM... Explorer

Databases 1

smart\_stock\_db (Default)

Tables 11

Category

Forecast

Inventory

Order

Order\_Contains\_Inventories

Product

Product\_Has\_PromotionalO...

PromotionalOffer

PromotionalOffers\_Applied...

Supplier

User

RUN FORMAT CLEAR

1 DESCRIBE PromotionalOffer;

RESULTS

Field	Type	Null	Key	Default
PromotionalOfferid	int	NO	PRI	
StartDate	date	YES		
EndDate	date	YES		
DiscountRate	float	YES		

## 2.7. Table 7: Order

Google Cloud cs411 Search (/) for resources, docs, products and more

Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

All instances > smart-stock

PRIM... Explorer

Database 1

smart\_stock\_db (Default)

Tables 11

Category

Forecast

Inventory

Order

Order\_Contains\_Inventories

Product

Product\_Has\_PromotionalO...

PromotionalOffer

PromotionalOffers\_Applied...

Supplier

User

1 DESCRIBE Order;

RUN FORMAT CLEAR

RESULTS

Field	Type	Null	Key	Default
OrderId	int	NO	PRI	
OrderDate	date	YES		
TotalPrice	float	YES		

## 2.8. Table 8: Forecast

Google Cloud cs411 Search (/) for resources, docs, products and more

Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

All instances > smart-stock

PRIM... Explorer

Database 1

smart\_stock\_db (Default)

Tables 11

Category

Forecast

Inventory

Order

Order\_Contains\_Inventories

Product

Product\_Has\_PromotionalO...

PromotionalOffer

PromotionalOffers\_Applied...

Supplier

User

1 DESCRIBE Forecast;

RUN FORMAT CLEAR

RESULTS

Field	Type	Null	Key	Default
ForecastId	int	NO	PRI	
ProductId	int	YES	MUL	
ForecastDate	date	YES		
ForecastQuantity	int	YES		

## 2.9. Table 9: Order Contains Inventories

Google Cloud cs411 Search (/) for resources, docs, products and more

Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

All instances > smart-stock

PRIM... Explorer

Database 1

smart\_stock\_db (Default)

Tables 11

Category

Forecast

Inventory

Order

Order\_Contains\_Inventories

Product

Product\_Has\_PromotionalO...

PromotionalOffer

PromotionalOffers\_Applied...

Supplier

User

1 DESCRIBE Order\_Contains\_Inventories;

RUN FORMAT CLEAR

RESULTS

Field	Type	Null	Key	Default
OrderId	int	NO	PRI	
InventoryId	int	NO	PRI	
Quantity	int	YES		

## 2.10. Table 10: PromotionalOffers AppliedOn Order

Google Cloud cs411 Search (/) for resources, docs, products and more

Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

All instances > smart-stock

PRIM... Explorer

Database 1

smart\_stock\_db (Default)

Tables 11

Category

Forecast

Inventory

Order

Order\_Contains\_Inventories

Product

Product\_Has\_PromotionalO...

PromotionalOffer

PromotionalOffers\_Applied...

Supplier

User

1 DESCRIBE PromotionalOffers\_AppliedOn\_Order;

RUN FORMAT CLEAR

RESULTS

Field	Type	Null	Key	Default
OrderId	int	NO	PRI	
PromotionalOfferId	int	NO	PRI	

2.11. Table 11: Product Has PromotionalOffers

The screenshot shows the Google Cloud SQL console interface. In the left sidebar, the 'Tables' list is expanded, showing 'Product\_Has\_PromotionalOffers'. The main panel displays the table's structure with the following fields:

Field	Type	Null	Key	Default
Productid	int	NO	PK	
PromotionalOfferid	int	NO	PK	

3. Data in Tables

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

3.1. Table 1: Users

The screenshot shows the Google Cloud SQL console interface. The top panel displays the SQL query: `SELECT COUNT(*) FROM User;` and the result: `COUNT(*)` with the value `1000`. The bottom panel displays the SQL query: `SELECT * FROM User LIMIT 15;` and the results for the first 15 rows of the 'Users' table.

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.f4dhiO8tFXjyujSRoMC0wr6Fkk07GVy	Analyst	Yetty	Carsey	
3	lkitt2@house.gov	\$2a\$04\$/C/5HdlSLcalXT1D319ffq.vCDcHnVVqK6Ka.V8xOd.Nb8UVaEu6GS	Analyst	Leupold	Kitt	(950) 9693528
Rows per page: 20 1 - 15 of 15  < < > >						
5	dmacanespie4@stanford.edu	\$2a\$04\$8AQ57lxlG7FQs/Zr1NO2.usxoLDUww4McH.RLUJ8HSXYVjGP37u	Analyst	Dene	MacAnespie	
6	stiling5@shareasale.com	\$2a\$04\$MuPdNQ/Cvauk54nhHpusPuxvw8RfAb7zBfNyHgY/V2m1d4M6b...	Analyst	Shadow	Tilling	
7	clandall6@berkeley.edu	\$2a\$04\$WMRYAZasyH7ShSi0onqjee9luUu2CdizeFEnm2jwhA0owWnPyaQ...	Admin	Christine	Landall	(754) 6677029
8	eeagar7@abc.net.au	\$2a\$04\$BcuBbKj/bm6f1QM.3BAusu3En.YDoEbsl25VKhqV5DrA31SA4yeC	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	Pauczek-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 Kemels - 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

+

EXPL

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

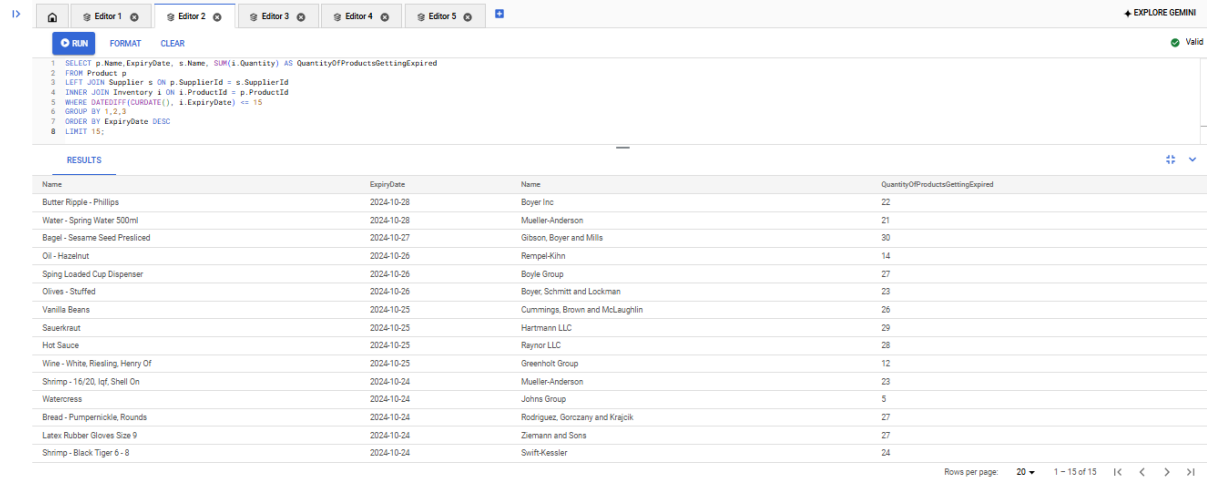
## 4. Advanced SQL Queries

- 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:



The screenshot shows a SQL query editor with a query and its results. The query is the same as the one provided in the previous block. The results table has 4 columns: Name, ExpiryDate, Name, and QuantityOfProductsGettingExpired. The results are sorted by ExpiryDate in descending order.

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
Spring 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 - Pumpkinickie, 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:

<div> <div>Editor 1</div> <div>Editor 2</div> <div>Editor 4</div> <div>Editor 5</div> </div> <div>EXPLORE GEMIN</div>			
<div> <div>RUN</div> <div>FORMAT</div> <div>CLEAR</div> </div> <div> <div>1</div> <div>ORDER_Contains_Inventories o</div> </div> <div> <div>8</div> <div>JOIN</div> </div> <div> <div>9</div> <div>Order o ON oc.OrderId = oc.OrderId</div> </div>			
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

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:

All instances > smart-stock

1x Editor 1 Editor 2 Editor 3 Editor 4 Editor 5 Editor 6 EXPLORE GEM

RUN FORMAT CLEAR

```

13 Inventory 1
14 ON o.inventoryId=1.InventoryId
15 LEFT JOIN

```

RESULTS

Name
Soup - Knorr, Chicken Noodle
Pizza - Cheese 12 Inch
Towels - Paper / Kraft
Pasta - Lasagna, Dry
Ice Cream Bar - Hagen Daz
Muffin Batt - Blueberry Passion
Soup - French Onion, Dry
Hot Chocolate - Individual
Soup - Verve - Chipotle Chicken
Fish - Soup Base, Bouillon
Cookies - Englishbay Chochip
Crush - Orange, 355ml
Rum - Spiced, Captain Morgan
Vodka - Hot, Inferno
Ice Cream - Super Sandwich

Rows per page: 20 1 - 15 of 15 < > >|

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:

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

RUN FORMAT CLEAR

```

1 SELECT p.ProductId,

```

RESULTS

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, lgf, Shell On	1d	5
84	Wine - Kvv Chenin Blanc South	24h	5
447	Wine - Redchard Merritt	24h	5
80	Squid US - Thailand	1d	5
143	Caviar - Salmon	1d	5
417	Wine - Red, Mosale Zweigelt	24h	5
440	Crush - Orange, 355ml	24h	5

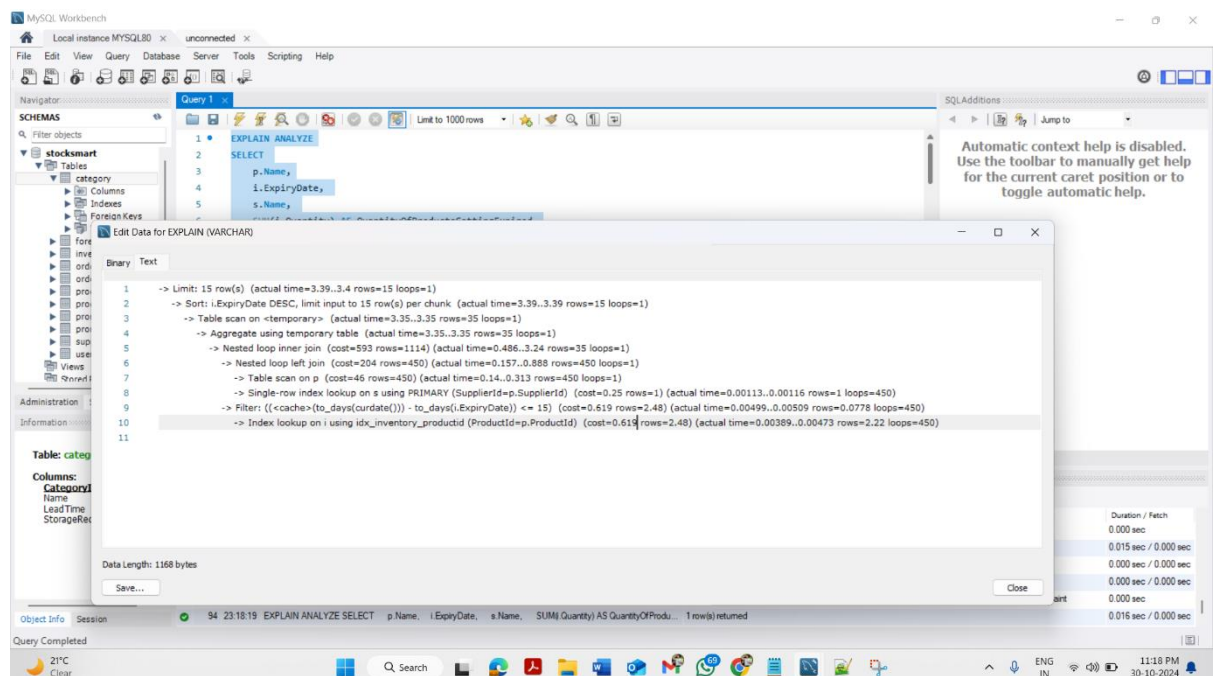
Rows per page: 20 1 - 15 of 15 < > >|

## 5. Indexing Analysis

### 5.1. Complex Query 1

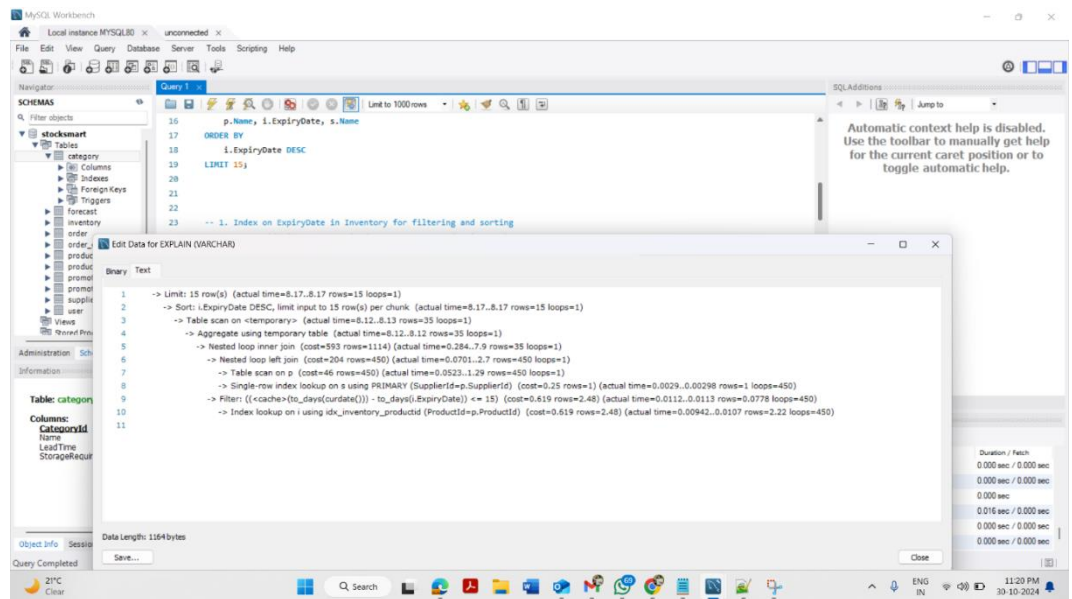
- Baseline:

```
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;
```



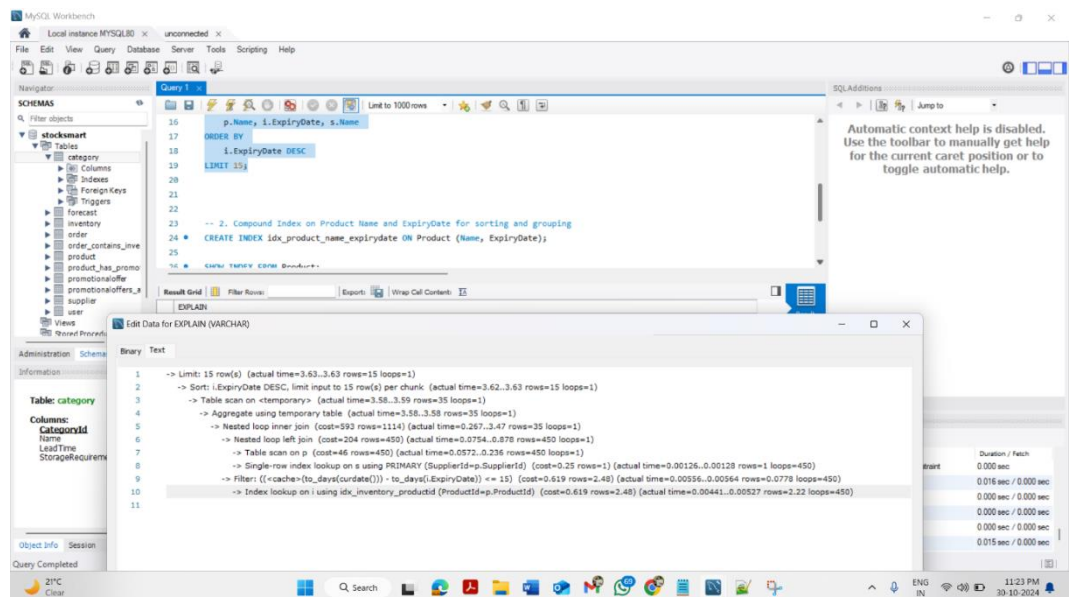
- Index Design 1: Index on ExpiryDate in Inventory for filtering and sorting

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



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

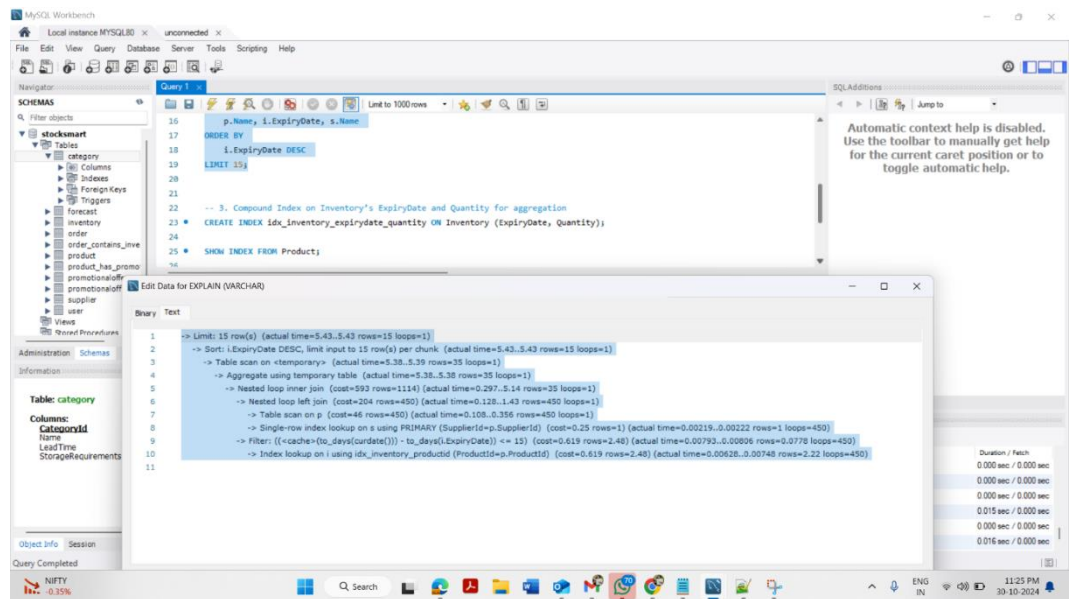
CREATE INDEX idx\_product\_name\_expirydate ON Product (Name, ExpiryDate);



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

CREATE INDEX idx\_inventory\_expirydate\_quantity ON Inventory (ExpiryDate, Quantity);



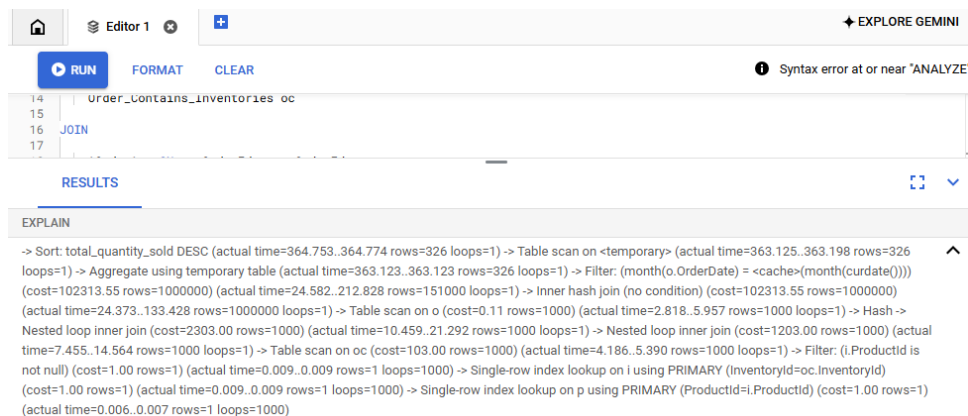


- Conclusion for Query 1: Since the indexes did not improve query cost, they are not recommended for Query 1.

## 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:

EXPLAIN

-> Sort: total\_quantity\_sold DESC (actual time=352.236..352.271 rows=326 loops=1) -> Table scan on <temporary> (actual time=351.892..351.960 rows=326 loops=1) -> Aggregate using temporary table (actual time=351.888..351.888 rows=326 loops=1) -> Filter: (month(o.OrderDate) = <cache>(month(curdate())))) (cost=102303.39 rows=1000000) (actual time=29.687..200.424 rows=151000 loops=1) -> Inner hash join (no condition) (cost=102303.39 rows=1000000) (actual time=19.021..127.793 rows=1000000 loops=1) -> Covering index scan on o using index\_order\_date (cost=0.10 rows=1000) (actual time=0.043..0.797 rows=1000 loops=1) -> Hash -> Nested loop inner join (cost=2300.75 rows=1000) (actual time=11.841..18.762 rows=1000 loops=1) -> Nested loop inner join (cost=1200.75 rows=1000) (actual time=9.190..13.400 rows=1000 loops=1) -> Covering index scan on oc using index\_total\_quantity\_sold (cost=100.75 rows=1000) (actual time=2.861..3.139 rows=1000 loops=1) -> Filter: (i.ProductId is not null) (cost=1.00 rows=1) (actual time=0.010..0.010 rows=1 loops=1000) -> Single-row index lookup on i using PRIMARY (InventoryId=oc.InventoryId) (cost=1.00 rows=1) (actual time=0.010..0.010 rows=1 loops=1000) -> Single-row index lookup on p using PRIMARY (ProductId=i.ProductId) (cost=1.00 rows=1) (actual time=0.005..0.005 rows=1 loops=1000)

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);
```

EXPLAIN

-> Sort: total\_quantity\_sold DESC (actual time=333.403..333.425 rows=326 loops=1) -> Table scan on <temporary> (actual time=333.210..333.254 rows=326 loops=1) -> Aggregate using temporary table (actual time=333.207..333.207 rows=326 loops=1) -> Filter: (month(o.OrderDate) = <cache>(month(curdate())))) (cost=100803.39 rows=1000000) (actual time=15.781..180.485 rows=151000 loops=1) -> Inner hash join (no condition) (cost=100803.39 rows=1000000) (actual time=2.790..107.499 rows=1000000 loops=1) -> Covering index scan on o using index\_order\_date (cost=0.10 rows=1000) (actual time=0.046..0.712 rows=1000 loops=1) -> Hash -> Nested loop inner join (cost=800.75 rows=1000) (actual time=0.081..2.536 rows=1000 loops=1) -> Nested loop inner join (cost=450.75 rows=1000) (actual time=0.072..1.570 rows=1000 loops=1) -> Covering index scan on oc using index\_total\_quantity\_sold (cost=100.75 rows=1000) (actual time=0.051..0.325 rows=1000 loops=1) -> Filter: (i.ProductId is not null) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000) -> Single-row index lookup on i using PRIMARY (InventoryId=oc.InventoryId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000) -> Single-row index lookup on p using PRIMARY (ProductId=i.ProductId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)

### 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);
```

EXPLAIN ANALYZE  
SELECT  
p.ProductId,  
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;

RESULTS

EXPLAIN

-> Sort: total\_quantity\_sold DESC (actual time=812.721..812.758 rows=326 loops=1) -> Table scan on <temporary> (actual time=810.479..810.585 rows=326 loops=1) -> Aggregate using temporary table (actual time=809.452..809.452 rows=326 loops=1) -> Filter: (month(o.OrderDate) = <cache>(month(curdate())))) (cost=100803.39 rows=1000000) (actual time=39.754..419.828 rows=151000 loops=1) -> Inner hash join (no condition) (cost=100803.39 rows=1000000) (actual time=5.956..256.537 rows=1000000 loops=1) -> Covering index scan on o using idx\_order\_orderdate (cost=0.10 rows=1000) (actual time=0.042..1.532 rows=1000 loops=1) -> Hash -> Nested loop inner join (cost=800.75 rows=1000) (actual time=0.154..5.596 rows=1000 loops=1) -> Nested loop inner join (cost=450.75 rows=1000) (actual time=0.142..3.875 rows=1000 loops=1) -> Table scan on oc (cost=100.75 rows=1000) (actual time=0.116..0.638 rows=1000 loops=1) -> Filter: (i.ProductId is not null) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1000) -> Single-row index lookup on i using PRIMARY (InventoryId=oc.InventoryId) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1000) -> Single-row index lookup on p using PRIMARY (ProductId=i.ProductId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)

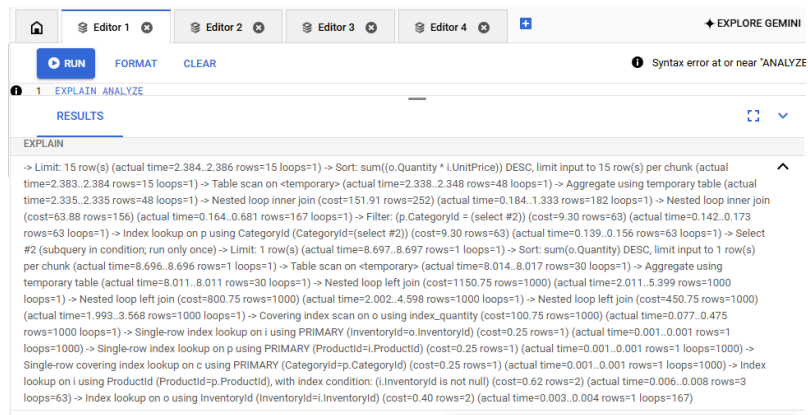
- Conclusion for Query 2: We are selecting Design 3 as the indexing strategy as we can see that the cost is decreasing from the original cost and also, we are using single-column index on frequently queried columns and are using composite index with filtering columns

### 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;
```

```
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

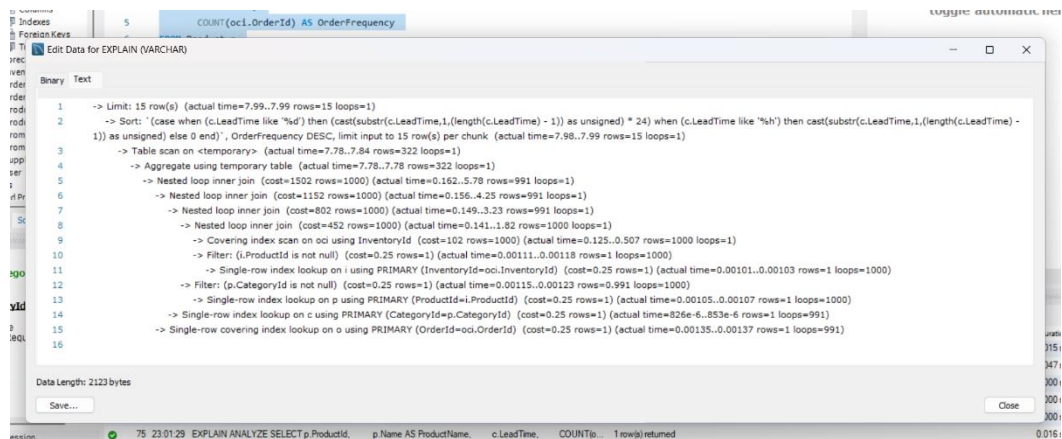
CAST(SUBSTRING(c.LeadTime, 1, LENGTH(c.LeadTime) - 1) AS UNSIGNED)

ELSE 0

END,

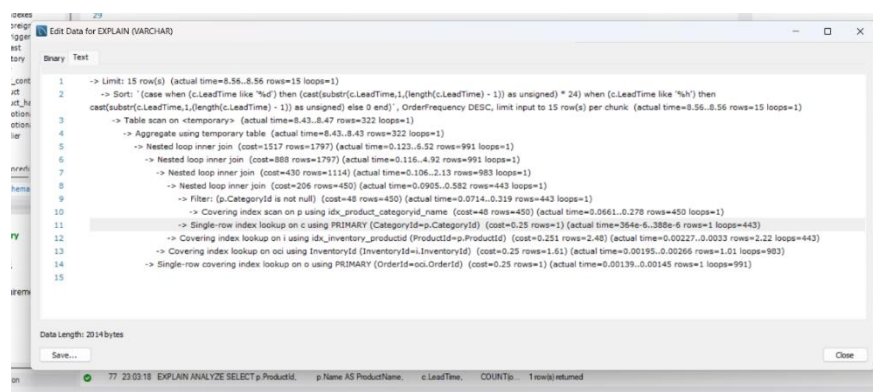
## OrderFrequency DESC

LIMIT 15;



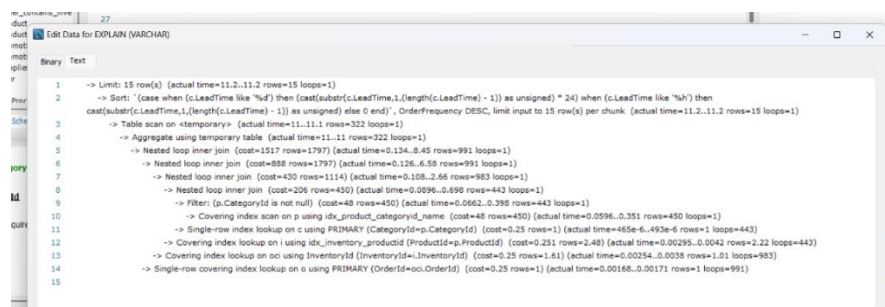
- Index Design 1: Index on Name to speed up grouping and ordering by product name

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);

```

1  --> Limit: 15 row(s) (actual time=10.8..10.8 rows=15 loops=1)
2  --> 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)
3  --> Table scan on <temporary> (actual time=10.6..10.7 rows=322 loops=1)
4  --> Aggregate using temporary table (actual time=10.6..10.6 rows=322 loops=1)
5  --> Nested loop inner join (cost=1517 rows=1797) (actual time=0.118..7.42 rows=991 loops=1)
6  --> Nested loop inner join (cost=888 rows=1797) (actual time=0.113..5.79 rows=991 loops=1)
7  --> Nested loop inner join (cost=430 rows=1114) (actual time=0.104..2.57 rows=993 loops=1)
8  --> Nested loop inner join (cost=206 rows=450) (actual time=0.0904..0.735 rows=443 loops=1)
9  --> Filter: (p.CategoryId is not null) (cost=48 rows=450) (actual time=0.0757..0.479 rows=443 loops=1)
10 --> Covering index scan on p using idx_product_categoryid_name (cost=48 rows=450) (actual time=0.0713..0.394 rows=450 loops=1)
11 --> 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)
12 --> 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)
13 --> Covering index lookup on ocl using InventoryId (InventoryId=i.InventoryId) (cost=0.25 rows=1.61) (actual time=0.0023..0.00311 rows=1.01 loops=983)
14 --> Single-row covering index lookup on o using PRIMARY (OrderId=ocl.OrderId) (cost=0.25 rows=1) (actual time=0.00144..0.00147 rows=1 loops=991)
15

```

- 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.