SQL RDBMS Implementation for Ecommerce

(with Front End)

# Nabeel Khan

School of Engineering and Applied Science – Data Science

Buffalo, USA

nkhan6@buffalo.edu

# Imran Anwar Rahman Khan

# School of Engineering and Applied Science – Data Science

# Buffalo, USA

# imrananw@buffalo.edu

# Vijaya Rana

# School of Engineering and Applied Science – Data Science

# Buffalo, USA

# vrana3 @buffalo.edu

**ABSTRACT*— In this paper, we discuss the Relational Database implementation for an e-commerce platform. An e-commerce platform needs to store many data points, of users, sellers, customers, orders, delivery times, etc. In this paper, we explain our approach to implementing an RDBMS system for application in an eCommerce system scenario.***

**I. INTRODUCTION**

Storing information in an RDBMS system is helpful for any business, especially e-commerce businesses. It can help them solve many business analytics related problems, such as:

● Top Selling Category of products (Daily, Weekly, Monthly, Yearly)

● Most purchased category by location (city, state)

● Most Popular/Frequent payment method for the users over time:

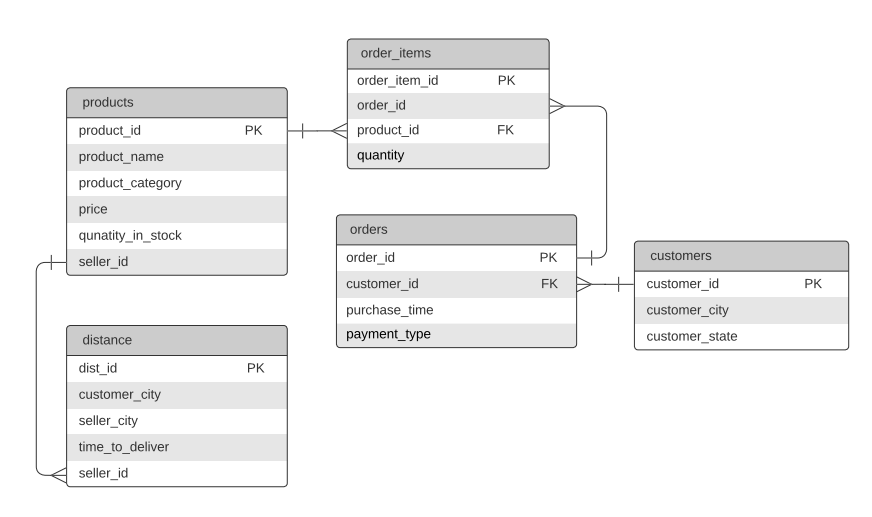
● Most Valuable payment method of the users over time

● Breakdown of payment type per category

● Breakdown of payment type by location

These figures can help in providing useful insights into the performance of the business and help with important decisions.

**II. E/R DIAGRAM**



**III. TASKS FOR THE PROJECT**

These are the important listed tasks for the project

* Extract Data from Kaggle and augmented it using Python.
* Data wrangling & cleaning with pandas.
* Validations to maintaining referential integrity
* Dashboard creation to implement run time functionalities
* Unit Testing with all scenarios for the two dashboards results

**IV. STRUCTURE FOR THE DATABASE**

In our database we have final tables with the below functionalities

Products - This table stores the following information:

* product\_name
* product\_category
* price
* quantity\_in\_stock.

Orders-This table stores the following information

* payment type
* order id
* customer\_id
* purchase time

Customers -This table stores the following information:

* customer\_id
* customer\_city
* customer\_state

Order\_items--This table stores the following information:

* order\_item\_id
* order\_id
* product\_id
* quantity

Distance-This table store the following information

* distance\_id
* customer\_city
* seller\_city
* time\_to\_deliver
* seller\_id

We **assume** that we have a fixed number of repeated customers who will be placing repeated orders with the business.

**V. FINAL LIST OF TABLES**

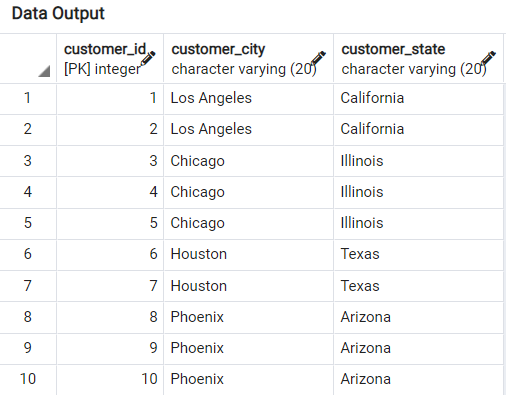
To satisfy the BCNF conditions and to implement some new functionalities we modified the existing tables and below is the final 5 tables we are using

* products
* order\_items
* distance
* orders
* customers

**VI. JUSTIFICATION OF RELATIONS IN BCNF**

**1 - Customers Relation:**

CREATE TABLE customers(customer\_id INT PRIMARY KEY,customer\_city VARCHAR(20),customer\_state VARCHAR(3));



*Valid Functional Dependencies :*

**customer\_id → {customer\_id, customer\_city, customer\_state}**

Only one real FD exists with others being subsets or derivations from the above FD.

We can see that here that

* There exists no multi-valued attribute, hence the relation customers is in **1st Normal Form (1NF)**.
* The relation is in 1NF and there exists no partial dependency. There exists no proper subset of the candidate key customer\_id, hence there can exist no partial dependency which indicates that the relation is in **2nd Normal Form (2NF).**
* The relation is in 2NF and there also exists no transitive functional dependency for Non Prime Attributes which means that the relation customers in in **3rd Normal Form (3NF).**
* Hence relation is in **3NF**
* If we closure of customer\_id

{ customer\_id}+ = {customer\_city, customer\_state}

Therefore, the LHS is a super key as its closure includes all the attributes.

* **Thus we can say relation in BCNF**

**2 - Order\_items Relation:**

CREATE TABLE order\_items

(

order\_item\_id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

order\_id INT,

product\_id INT,

quantity INT,

FOREIGN KEY (product\_id) REFERENCES products(product\_id)ON UPDATE CASCADE ON DELETE SET NULL

);



*Valid Functional Dependencies :*

**order\_item\_id → {order\_item\_id**, **order\_id**, **product\_id**, **quantity**}

Only one real FD exists with others being subsets or derivations from the above FD.

We can see that here that

* There exists no multi-valued attribute, hence the relation order\_items is in **1st Normal Form (1NF)**.
* The relation is in 1NF and there exists no partial dependency. There exists no proper subset of the candidate key order\_items\_id, hence there can exist no partial dependency which indicates that the relation is in **2nd Normal Form (2NF).**
* The relation is in 2NF and there also exists no transitive functional dependency for Non Prime Attributes which means that the relation order\_items is in **3rd Normal Form. (3NF).**
* Hence relation is in **3NF**
* If we closure of order\_item\_id

{ order\_item\_id }+ = {order\_item\_id, order\_id, product\_id, quantity}

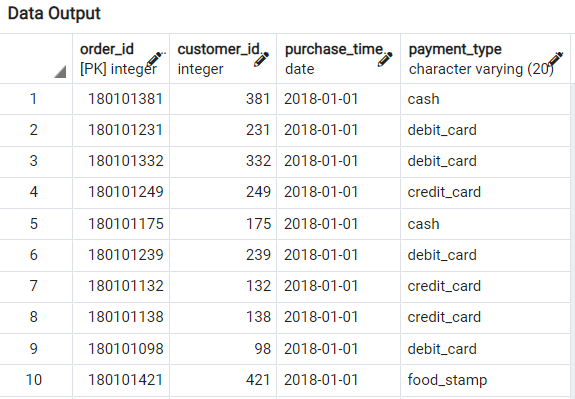
Therefore, the LHS is a super key as its closure includes all the attributes.

* **Thus we can say relation in BCNF**

**3 - Orders Relation:**

CREATE TABLE orders ( order\_id INT PRIMARY KEY,customer\_id INT, purchase\_time DATE, payment\_type VARCHAR(20),FOREIGN KEY (customer\_id)

REFERENCES customers(customer\_id),FOREIGN KEY (order\_id) REFERENCES orders(order\_id)ON UPDATE CASCADE ON DELETE SET NULL**);**

****

*Valid Functional Dependencies* :

**order\_id → {order\_id, customer\_id, purchase\_time,payment\_type}**

Only one real FD exists with others being subsets or derivations from the above FD.

We can see that here that

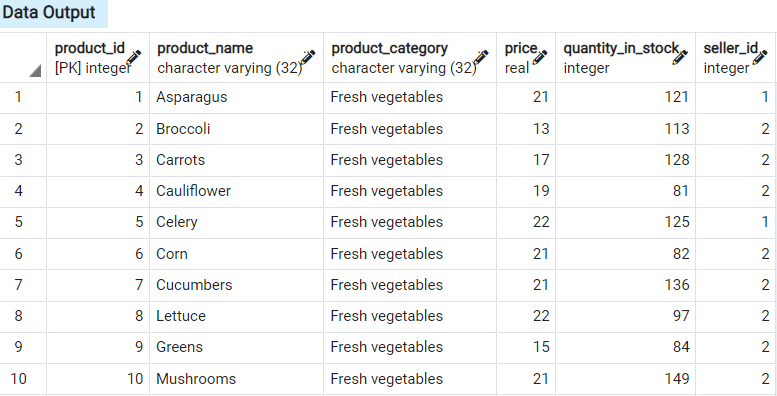
* There exists no multi-valued attribute, hence the relation orders is in **1st Normal Form (1NF)**.
* The relation is in 1NF and there exists no partial dependency. There exists no proper subset of the candidate key order\_id, hence there can exist no partial dependency which indicates that the relation is in **2nd Normal Form (2NF).**
* The relation is in 2NF and there also exists no transitive functional dependency for Non Prime Attributes which means that the relation orders is in **3rd Normal Form. (3NF).**
* Hence relation is in **3NF**
* If we closure of **order\_id**

{ order\_id }+ = {order\_id, customer\_id, purchase\_time,payment\_type}Therefore, the LHS is a super key as its closure includes all the attributes.

* **Thus we can say relation in BCNF**

**4 - Products Relation:**

CREATE TABLE products(product\_id SERIAL PRIMARY KEY,product\_name VARCHAR(32),product\_category VARCHAR(32),price REAL,quantity\_in\_stock INT, seller\_id INT );



*Valid Functional Dependencies :*

**product\_id →**

**{product\_id, product\_name, product\_category, price, quantity\_in\_stock, seller\_id}**

Only one real FD exists with others being subsets or derivations from the above FD.

We can see that here that

* There exists no multi-valued attribute, hence the relation Orders is in **1st Normal Form (1NF)**.
* The relation is in 1NF and there exists no partial dependency. There exists no proper subset of the candidate key product\_id, hence there can exist no partial dependency which indicates that the relation is in **2nd Normal Form (2NF).**
* The relation is in 2NF and there also exists no transitive functional dependency for Non Prime Attributes which means that the relation products is in **3rd Normal Form. (3NF).**
* Hence relation is in **3NF**
* If we closure of customer\_id

{ product\_id }+ = { product\_id, product\_name, product\_category, price quantity\_in\_stock, seller\_id}

Therefore, the LHS is a super key as its closure includes all the attributes.

* **Thus we can say relation in BCNF**

**5 - Distances Relation:**

CREATE TABLE distances

(dist\_id INT PRIMARYKEY, customer\_city VARCHAR(20), seller\_city VARCHAR(20), time\_to\_deliver INT, seller\_idINT **);**

****

*Valid Functional Dependencies* :

**dist\_id → {customer\_city, seller\_city, time\_to\_deliver, seller\_id}**

Only one real FD exists with others being subsets or derivations from the above FD.

We can see that here that there exists no multi-valued attribute, hence the relation Distances is in **1st Normal Form (1NF)**.

* The relation is in 1NF and there exists no partial dependency. There exists no proper subset of the candidate key transaction\_id, hence there can exist no partial dependency which indicates that the relation is in **2nd Normal Form (2NF).**
* The relation is in 2NF and there also exists no transitive functional dependency for Non Prime Attributes which means that the relation transactions is in **3rd Normal Form. (3NF).**
* Hence relation is in **3NF**

**VII. DATASET CHALLENGES FACED & RESOLUTION**

* ***Challenge 1*** *-* Our dataset contains 100,000 orders, and due to scale we found it challenging to insert data in our database, due to erroneous entries.
* ***Solution.*** We used Python for data prepping, missing value handling, erroneous value handling, and datatype correction. Additionally we had to do brainstorming to figure out which attributes we needed to filter out.
* ***Challenge 2 -*** Generating the unique and consistent order\_id but different order\_item\_id at run time from the dashboard was challenging.
* ***Solution -*** We assumed order\_id to be a combination of date and customer id, hence all order\_items from a customer on a single day will automatically fall into a single unique order\_id. This allowed us to keep our frontend simple by only keeping a single order item.

**VIII. DATABASE QUERIES WITH EXECUTION RESULTS**

**Note:** Few variables in these queries were selected from the frontend

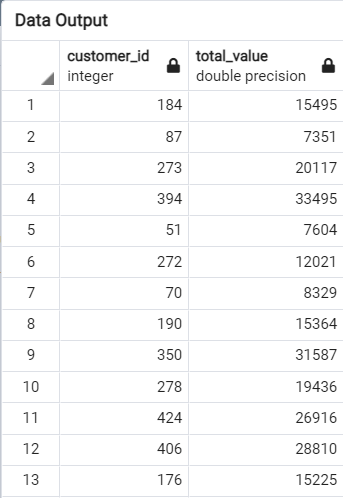
**1 - Top N most frequent customers:**

SELECT

orders.customer\_id, sum(order\_items.quantity \* products.price) as total\_value FROM orders INNER JOIN order\_items ON orders.order\_id = order\_items.order\_id

INNER JOIN products

ON order\_items.product\_id=products.product\_id

**

**2 - Top N most selling product\_categories (in past M days):**

SELECT

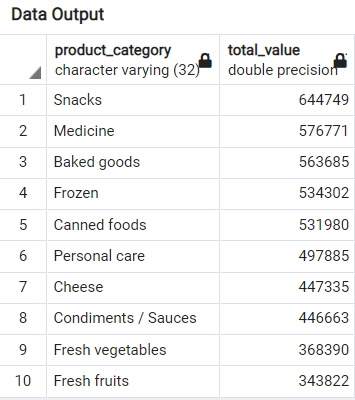
products.product\_category, sum(order\_items.quantity \* products.price) as total\_value

FROM order\_items

INNER JOIN products ON order\_items.product\_id = products.product\_id

INNER JOIN orders ON orders.order\_id = order\_items.order\_id

WHERE orders.purchase\_time > current\_date - {m} GROUP BY products.product\_category ORDER BY total\_value DESC LIMIT {n}

****

**3 - Mapping of customer IDs with the products that they have bought:**

SELECT customers.customer\_id, products.product\_name FROM products

INNER JOIN order\_items ON products.product\_id = order\_items.order\_id INNER JOIN orders ON order\_items.order\_id = orders.order\_id

INNER JOIN customers ON orders.customer\_id = customers.customer\_id

****

**4 - Inserting data into orders table:**

INSERT into orders (order\_id, customer\_id, purchase\_time, payment\_type) VALUES (""" + vals\_1 + ") ON CONFLICT DO NOTHING

\* These values are taken from the front-end panel

(from the data which the user enters)

**5 - Inserting data into order\_items table:**

INSERT into order\_items (order\_id ,product\_id, quantity) VALUES(""" + vals\_2 + ")”

\* These values are taken from the front-end panel

(from the data which the user enters)

**6 - Calculating the delivery time between seller and customer cities:**

SELECT time\_to\_deliver, seller\_city

FROM distances

WHERE seller\_id = {} AND customer\_city = {}

**7 - Getting Various Products options**

**(to show on front-end - after user selects the category):**

SELECT DISTINCT product\_id, product\_name, price, seller\_id from products WHERE product\_category = '{}'

**IX. QUERY EXECUTION ANALYSIS AND WAYS TO IMPROVE THEM**

**Problematic Query #1:**

Query to get the product\_name and product\_category for the products sold to customers in states (New York)

SELECT product\_name, product\_category

FROM products

WHERE product\_id IN (SELECT product\_id

FROM order\_items

WHERE order\_id IN(SELECT order\_id FROM orders

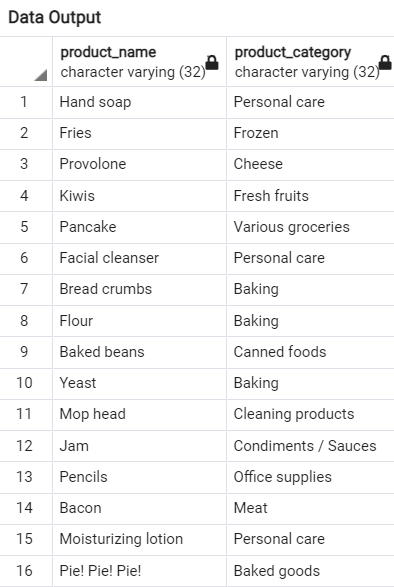
WHERE customer\_id IN(SELECT customer\_id

FROM customers

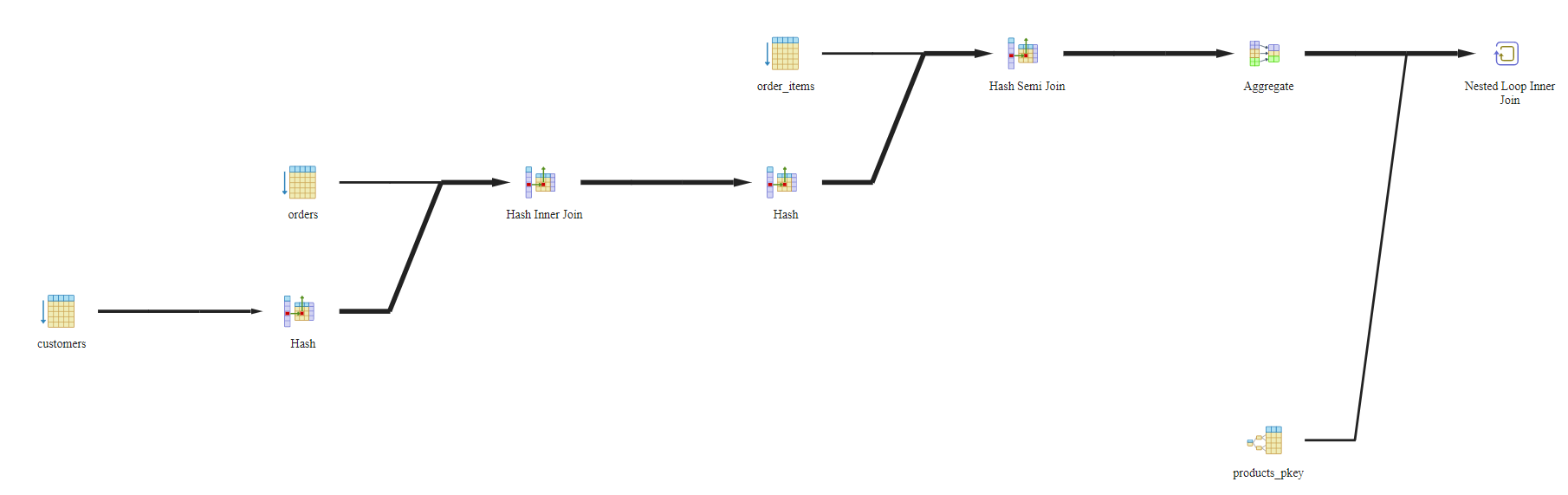
WHERE customer\_state IN ('New York'))))

**Run Time = 189 ms**

**OUTPUT**



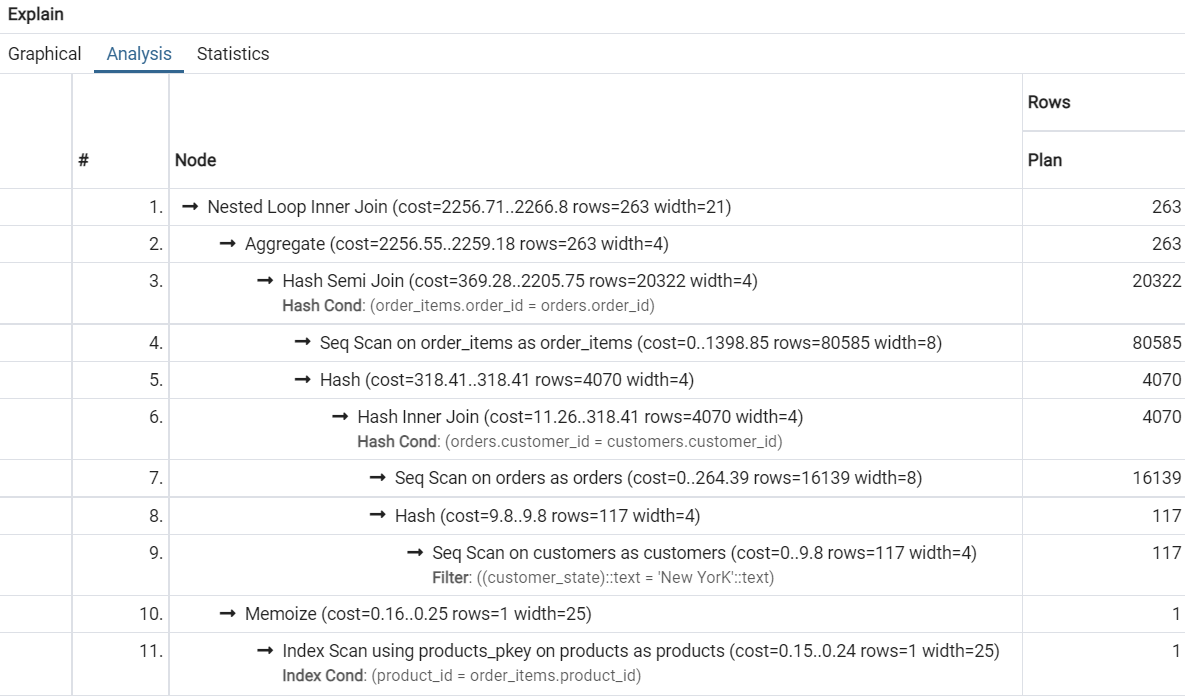
**EXECUTION PLAN**

****

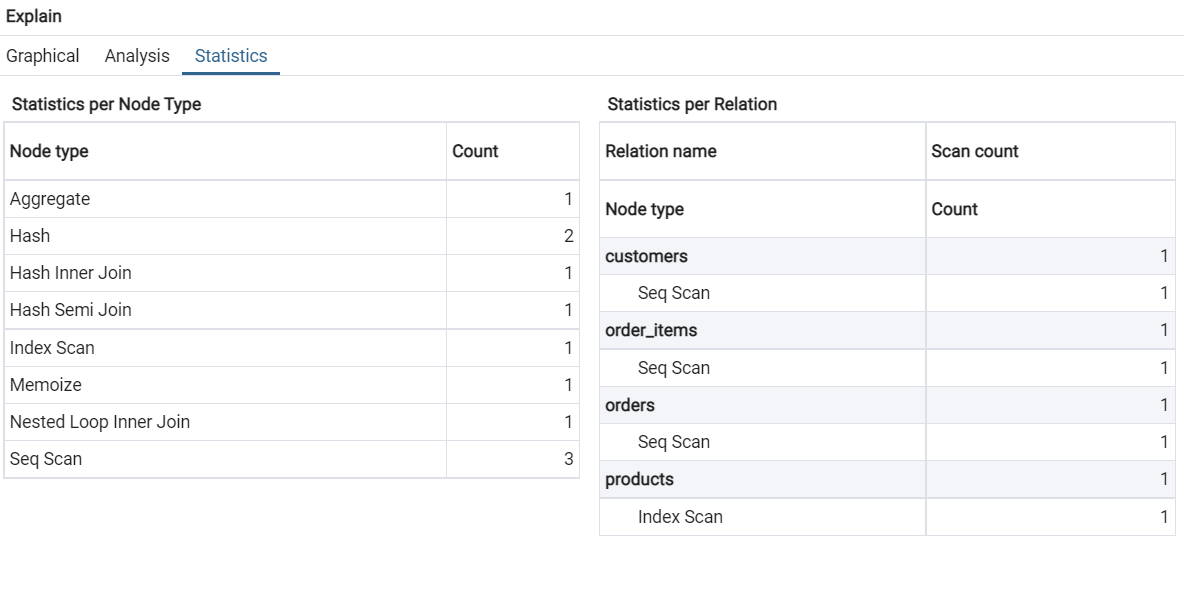
**QUERY PLAN**

****

**ANALYSIS**

****

**STATISTICS**

****

**Note: We achieved the cost of 2256.71 ,which we will try to improve**

**Improvement for Problematic Query #1:**

CREATE INDEX ind ON products(product\_id)

SELECT product\_name,product\_category FROM products

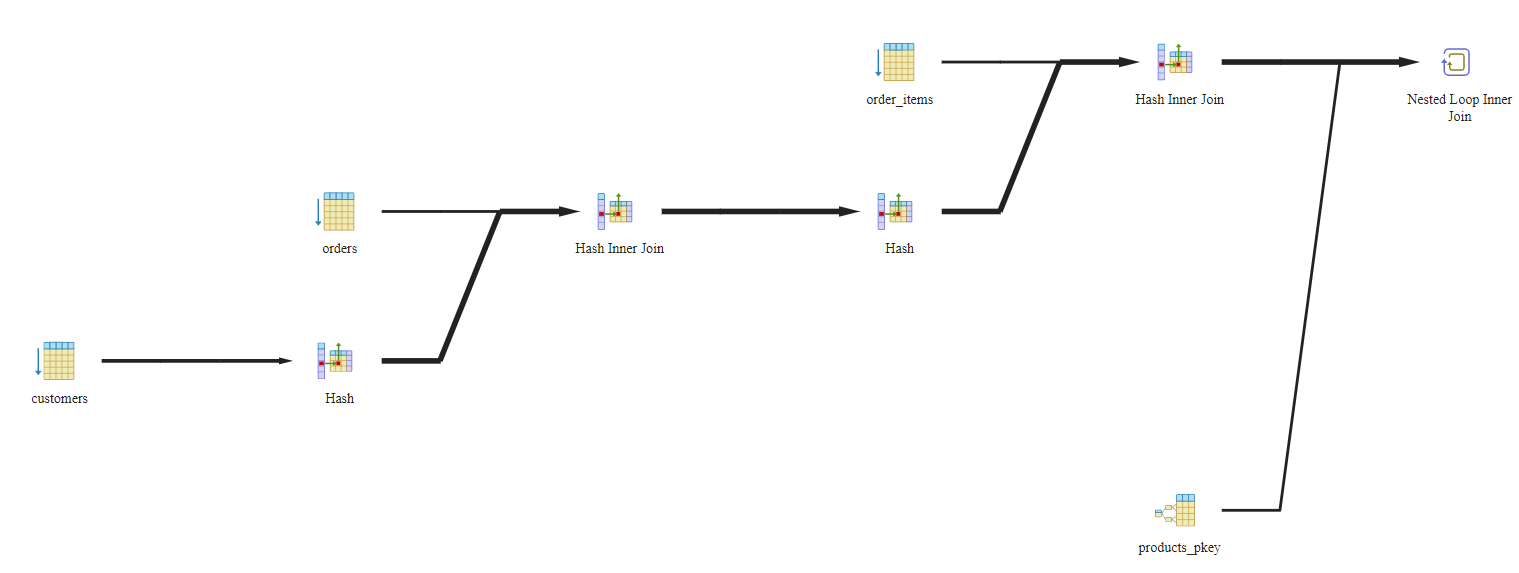
INNER JOIN order\_items ON products.product\_id = order\_items.product\_id

INNER JOIN orders ON orders.order\_id = order\_items.order\_id INNER JOIN customers ON customers.customer\_id = orders.customer\_id

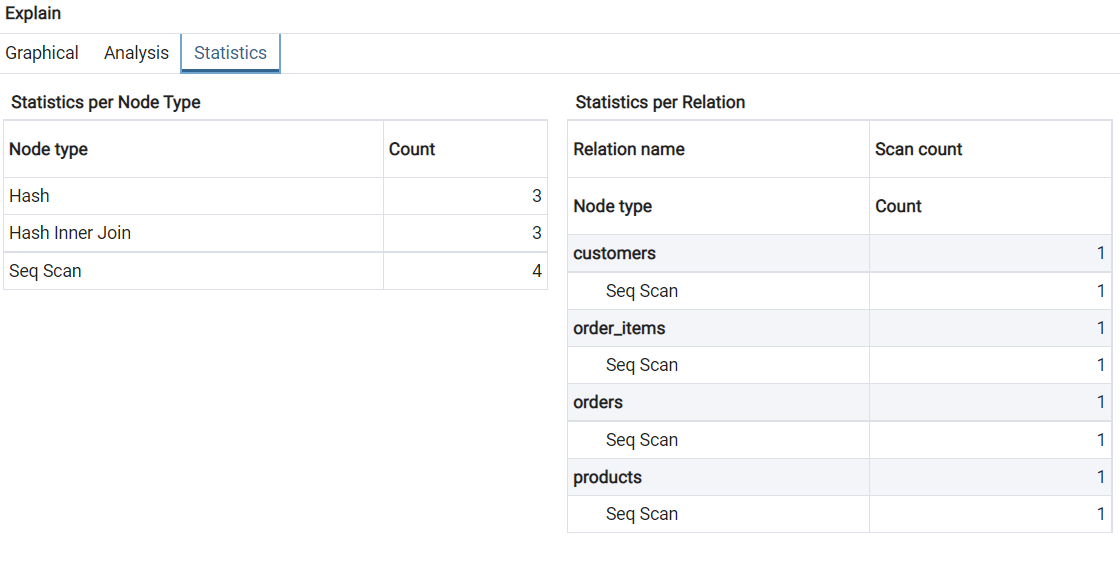
WHERE customers.customer\_state IN ('New York')

**Run Time = 70ms**

**EXECUTION PLAN**



**STATISTICS**



The nested query is seen to take a lot of cost. This can be improved. We made the nested query into a natural join query in addition to adding an index for the product\_id column in the products relation.

Note:We achieved the cost of 378.22 ,which is highly improved

**Problematic Query #2:**

SELECT DISTINCT distances.seller\_id, distances.seller\_city FROM distances

INNER JOIN products ON distances.seller\_id = products.seller\_id

INNER JOIN order\_items ON products.product\_id =order\_items.product\_id

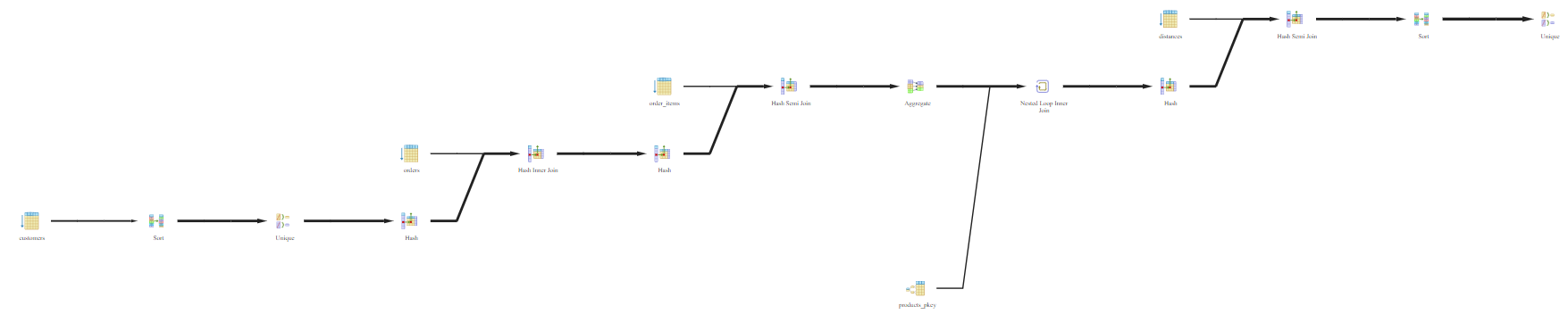
INNER JOIN orders ON order\_items.order\_id = orders.order\_id

INNER JOIN customers ON customers.customer\_id = orders.customer\_id

WHERE customers.customer\_state = 'NY'

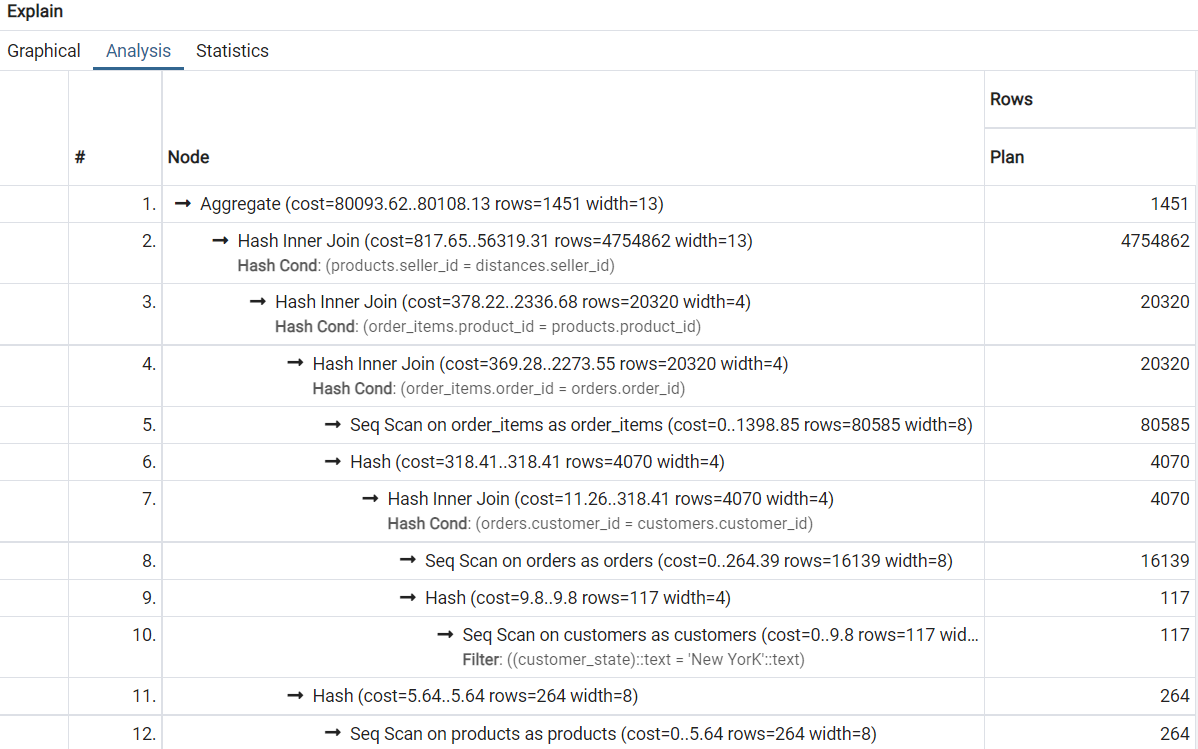
SELECT DISTINCT seller\_id,seller\_city

**Run Time = 965ms**

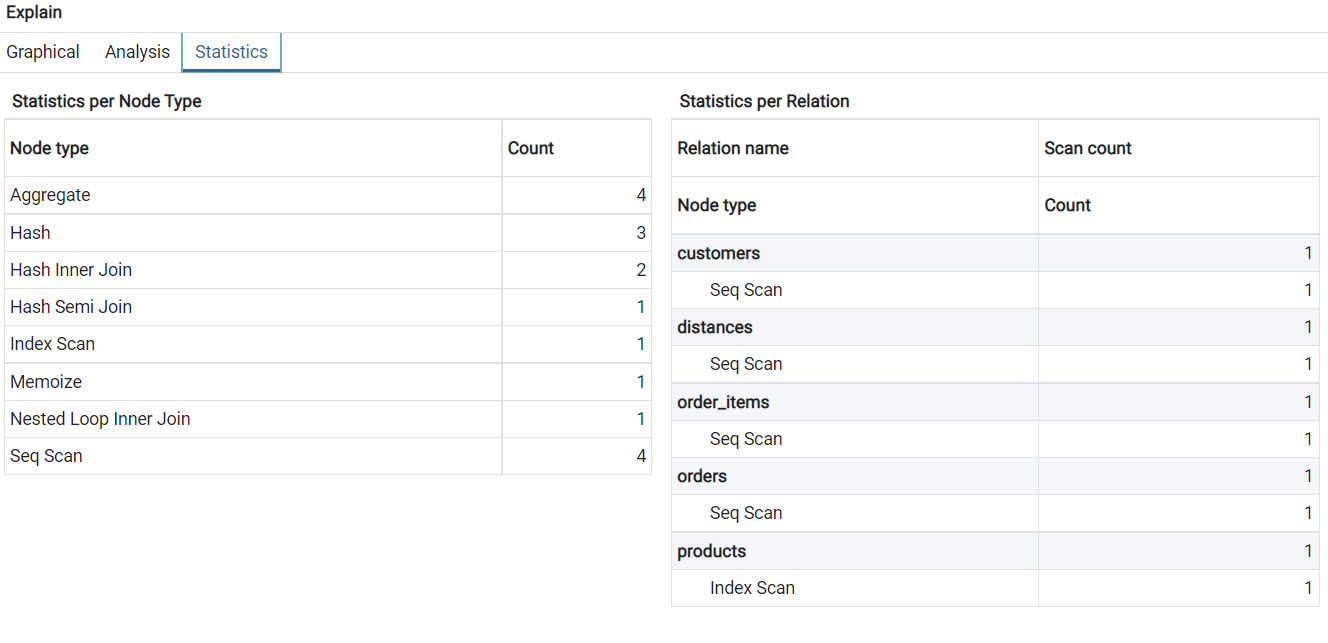
****

****

**ANALYSIS**

****

**STATISTICS**

****

Note:We achieved the cost of 2725.68 ..2740 ,which we will try to improve

**Improvement for Problematic Query #2:**

FROM distances WHERE seller\_id IN

(SELECT seller\_id FROM products

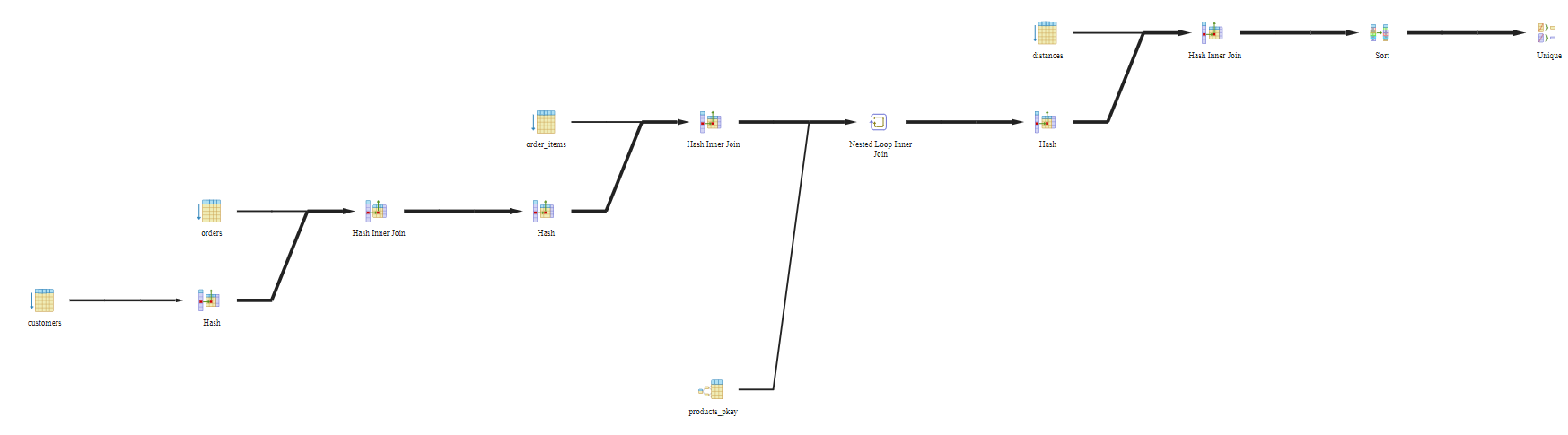
WHERE product\_id IN

(SELECT product\_id FROM order\_items

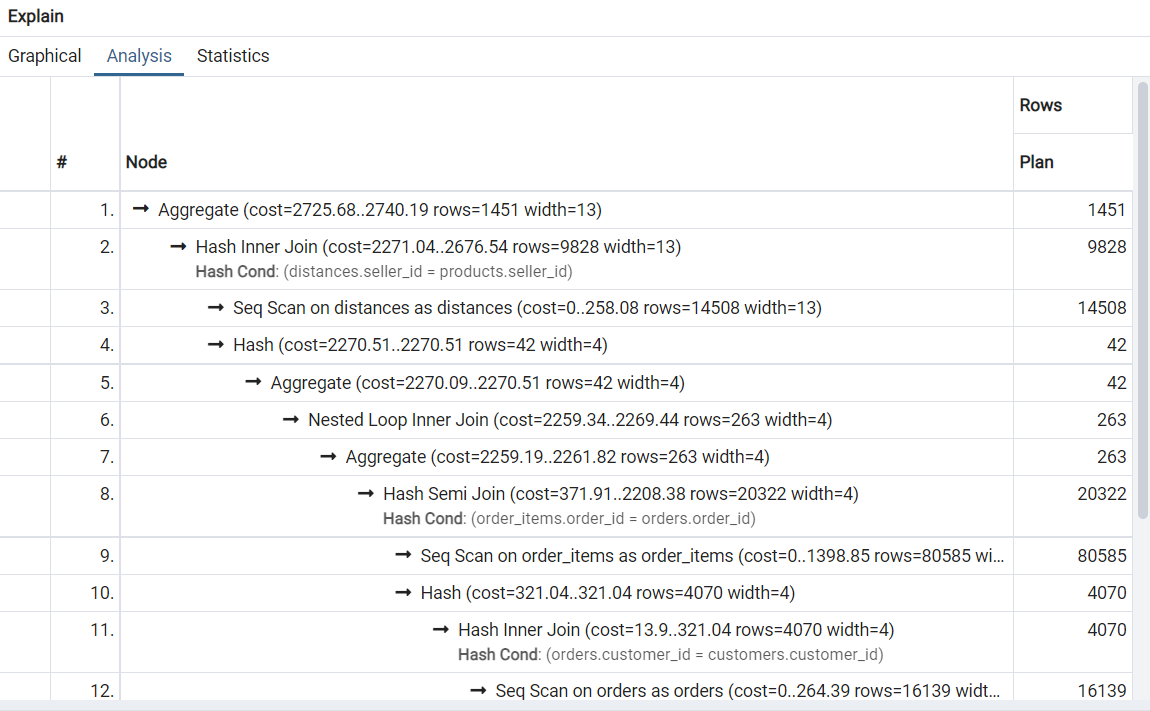
WHERE order\_id IN(SELECT order\_id FROM ORDERS WHERE customer\_id IN

(SELECT DISTINCT customer\_id FROM customers WHERE customer\_state = 'NY'))))

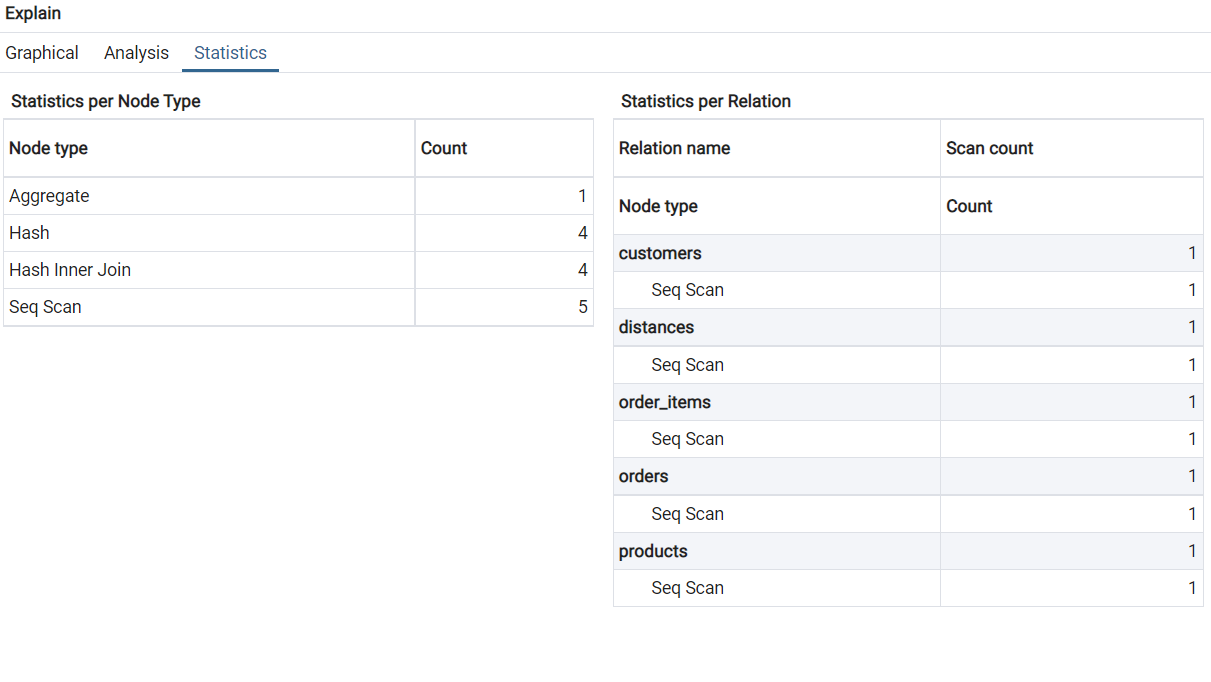
**RUN TIME: 65 ms**

****

**ANALYSIS**

****

**STATISTICS**

****

**SOLUTION APPROACH**

The nested query is seen to take a lot of cost. This can be improved. This can be improved.We made the second nested query into a natural join query in addition to adding an index for the num purchases column in the places relation.

**Problematic Query #3:**

SELECT customers.customer\_id, products.product\_name FROM products

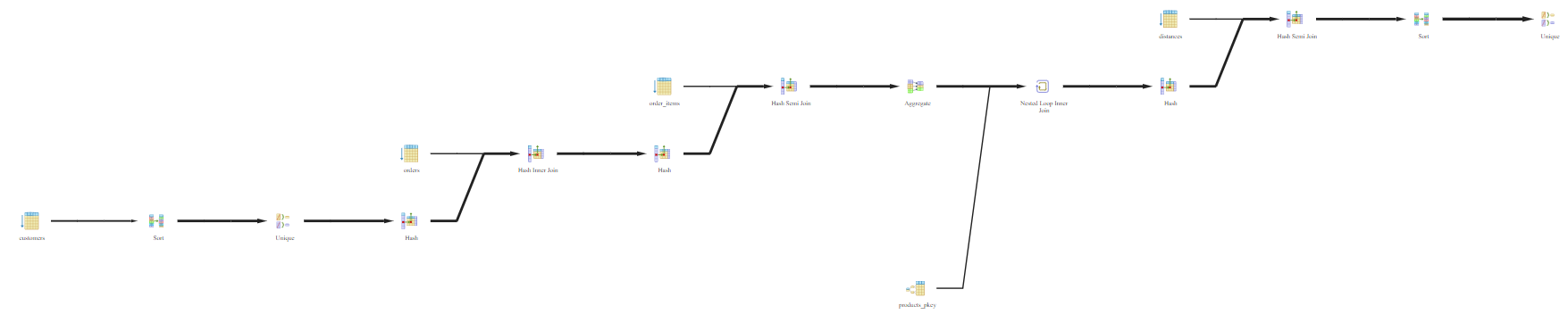
INNER JOIN order\_items ON products.product\_id = order\_items.order\_id INNER JOIN orders ON order\_items.order\_id = orders.order\_id

INNER JOIN customers ON orders.customer\_id = customers.customer\_id

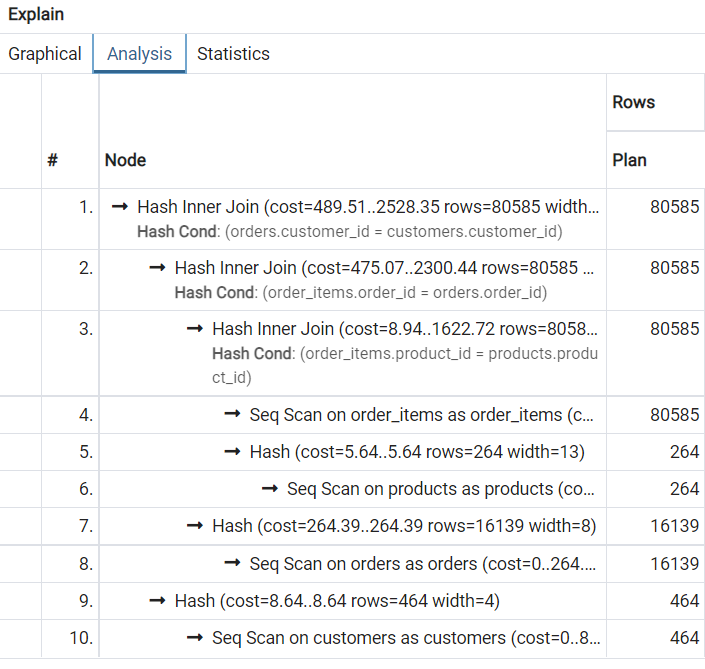
**Run Time = 103 ms**

**OUTPUT**

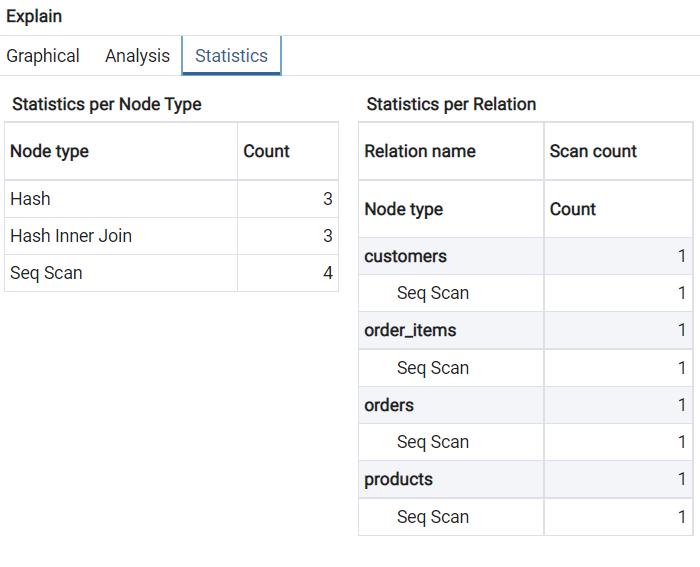
****

****

**ANALYSIS**



**STATISTICS**



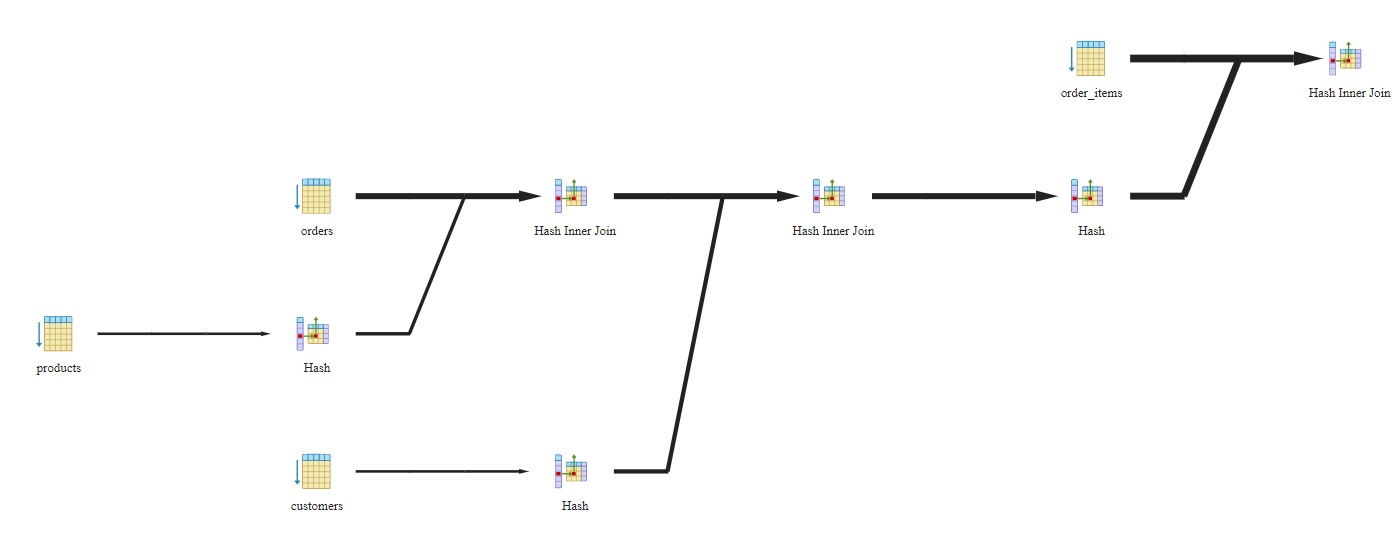
**Improvement for Problematic Query #3:**

SELECT customers.customer\_id, products.product\_name FROM products

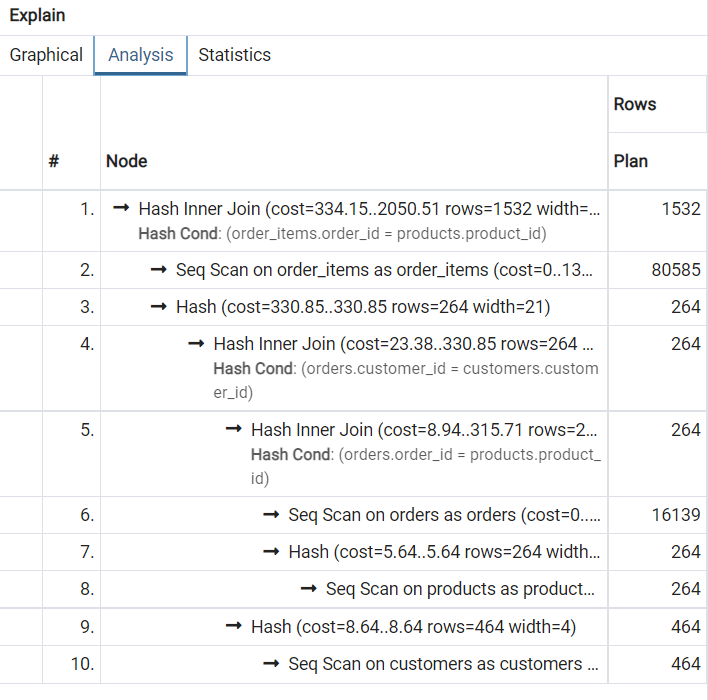
INNER JOIN order\_items ON products.product\_id = order\_items.product\_id INNER JOIN orders ON order\_items.order\_id = orders.order\_id

INNER JOIN customers ON orders.customer\_id = customers.customer\_id

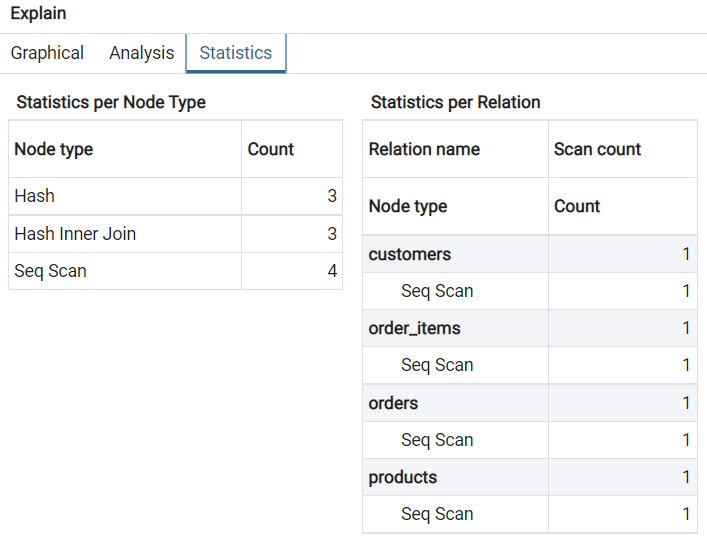
**Run Time = 48 ms**

****

**ANALYSIS**

****

**STATISTICS**

****

**SOLUTION APPROACH**

The nested query is seen to take a lot of cost. This can be improved. The filter places.numwebpurchases > 15 also takes a considerable amount of time. This can be improved.

We made the second nested query into a natural join query in addition to adding an index for the numwebpurchases column in the places relation.

**X. FRONT END**

We have implemented two dashboard functionality for end to end process to give more realistic impact

**1- Customer portal**

This helps the customer this help customer to place the order.We have implemented additional 3 functionality (highlighted yellow) as below

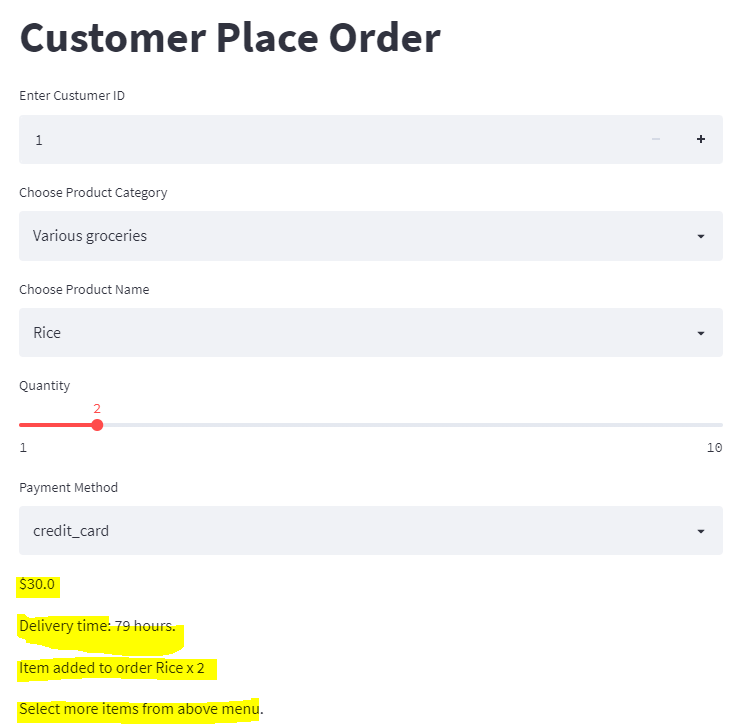
* Functionality to calculate the delivery time
* Functionality to calculate the total price
* Functionality to add more items to cart and accordingly new price will be updated **

Fig 1.1

***2:* Summary dashboard**

This helps to give the overall picture of the customer and purchase and revenue.We implement below functionalities

* On the basis of the number of previous days selected,we can choose the number of records we want to see.

Now using this we can see the top n customers,products\_categories

* Additionally we are calculating the total purchase of top

customers,and total purchase within each top product category

* Functionality to view the above statistics as bar graph
* Functionality to view the monthly revenue generated on the basis of number of orders placed and there price.



Fig 1.2

**WEBSITE URL REFERENCES**

* Hosted Static version of the frontend <https://lanbeee.github.io/DMQL>
* Olist dataset <https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce>
* Olist company profile

<https://pitchbook.com/profiles/company/102473-65#signals>

* Brazil - Market Challenges. Retrieved July 01, 2021

<https://www.trade.gov/country-commercial-guides/brazil-market-challenges>

**M .TEAM MEMBERS CONTRIBUTION**

We all contributed equally overall. Nabeel mostly handled frontend, Imran handled problematic queries, and Vijaya handled database and frontend interaction.