School of Business
University of Dundee

University of Dundee

School of Business

Assignment 1

Module: AC51047
Advanced Big Data Analysis

Submitted by
Ana Das
2622436

Dated: 04/11/2024

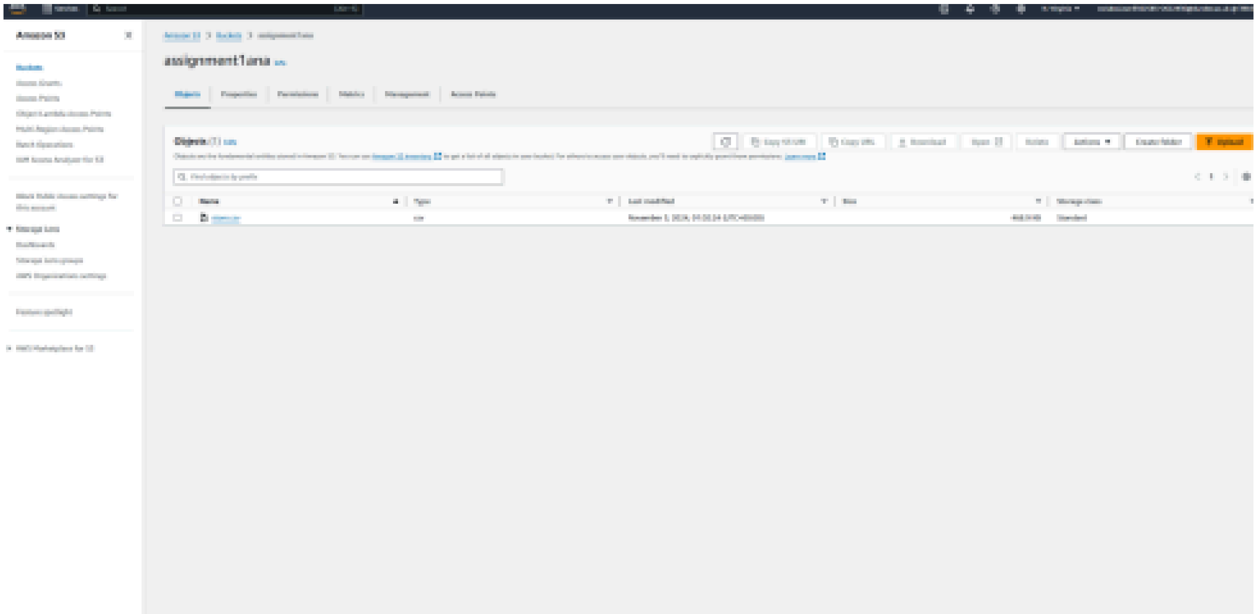
1. S3 Bucket Creation:

In the Learner Lab window, wait for the button to go green, then click AWS. Click on the previously created S3. Then create a bucket by choosing the Create bucket option on the initial S3 screen. Give it a unique name and create.

Next, upload an object. Click on the name of the bucket you have just created and add the file store.csv to this bucket.

Note: Before uploading the store.csv file, I cleaned the data, removed the space between the attribute names, and removed any special characters.

Amazon S3 buckets are essential for scalable, secure cloud data storage, offering high durability, encryption, and detailed access control. They integrate with AWS services for analytics and data processing and support versioning for recovery. S3 is also cost-effective, making it ideal for storing data for analysis and distribution.



2. Key-Value Stores: Amazon DynamoDB was used for the assignment here

2.1. Setting the Keys:

I chose RowID as the partition key for this table because it uniquely identifies each row. I did not include a sort key, as the partition key alone is sufficient to uniquely identify the rows, making a combined key unnecessary.

Alternatively, OrderID (as the partition key) with Product Name (as a sort key) could also serve as a unique identifier. However, using Customer ID as the partition key along with Product Name would be less effective, as each customer can place multiple orders, and each order may contain multiple products. This setup would add unnecessary complexity.

Update: During the assignment, I realized that using OrderID as the partition key and Product Name as the sort key would have been a more effective approach for this task. However, I lacked the necessary credits to start over and implement this structure.

2.2. Efficiently Executed Queries:

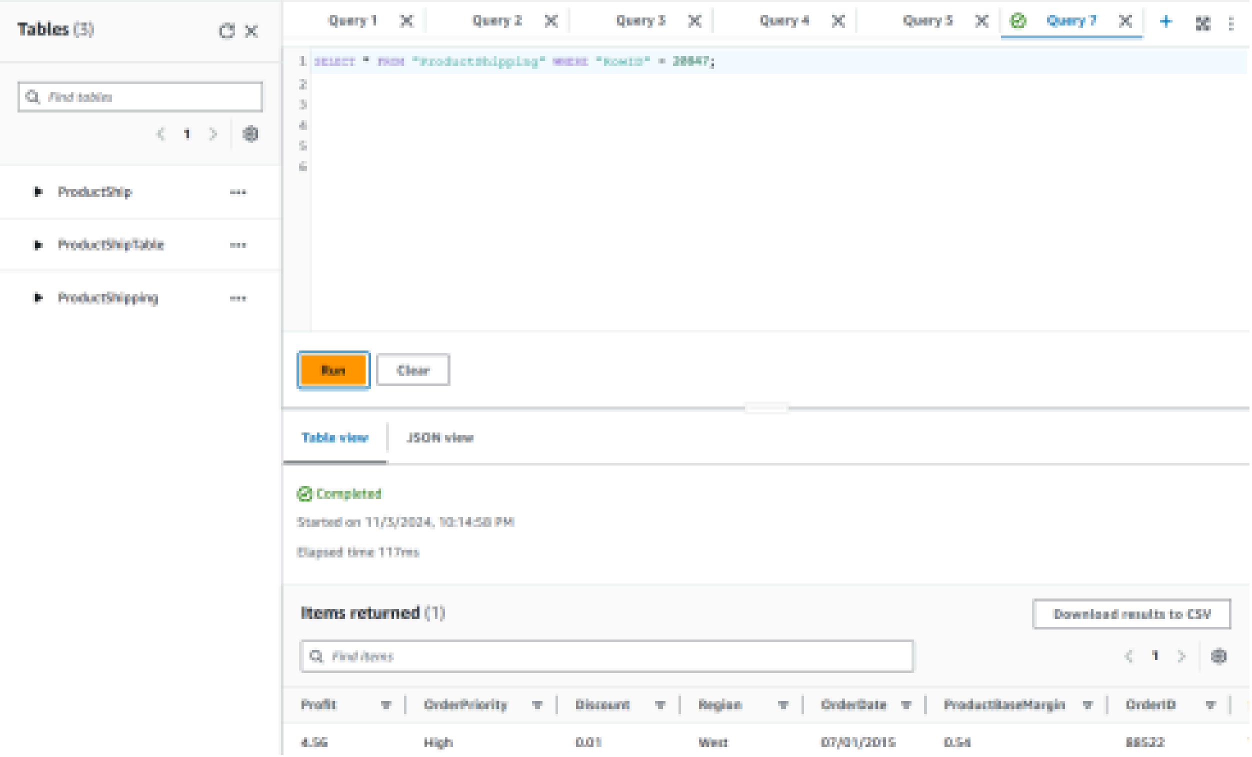
2.2.1. Example 1:

2.2.1.a. Query in English:

Select all from the table for a given Row ID

2.2.1.b. PartiQL Query:

```
SELECT * FROM "ProductShipping"
WHERE "RowID" = 20847;
```



```
SELECT * FROM "ProductShipping" WHERE "RowID" = 20847;
```

Completed

Started on 11/3/2024, 10:14:58 PM

Elapsed time 117ms

Items returned (1)

Profit	OrderPriority	Discount	Region	OrderDate	ProductBaseMargin	OrderID
4.56	High	0.01	West	07/01/2015	0.54	88522

2.2.1.c. Explanation:

This query directly retrieves a single row using RowID, the primary key. As DynamoDB handles indexing for the primary key by default, this operation is very efficient and does not require a table scan and additional filters. Since DynamoDB uses a hash function to store items based on the partition key, querying by primary key (RowID in this case) ensures that the operation is highly efficient and cost-effective, even without secondary indexes.

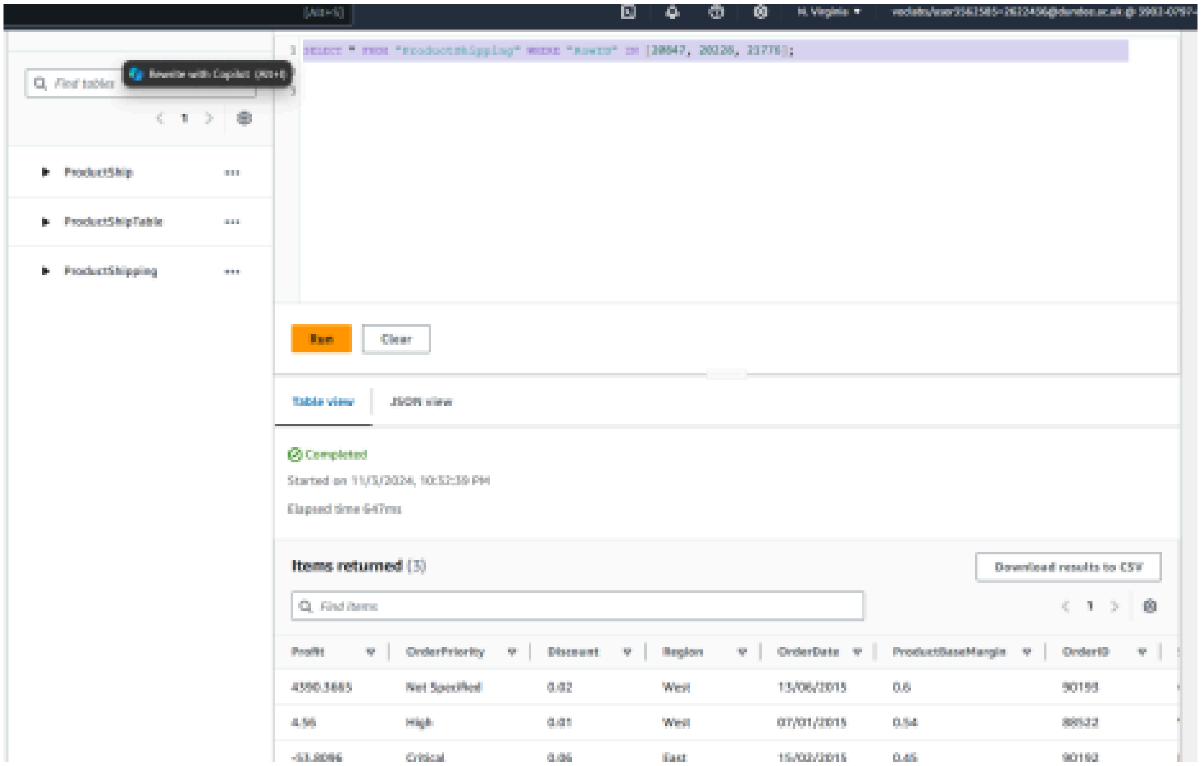
2.2.2. Example 2:

2.2.2.a. Query in English:

Select all from the table for a multiple Row ID

2.2.2.b. PartiiQL Query:

```
SELECT * FROM "ProductShipping"
WHERE "RowID" IN [20847, 20228, 21776];
```



```
1 SELECT * FROM "ProductShipping" WHERE "RowID" IN [20847, 20228, 21776];
```

Completed

Started on 11/3/2024, 10:52:39 PM

Elapsed time 647ms

Items returned (3)

Profit	OrderPriority	Discount	Region	OrderDate	ProductBaseMargin	OrderID
4390.3665	Not Specified	0.02	West	13/06/2015	0.6	90193
4.56	High	0.01	West	07/01/2015	0.54	88522
-53.8096	Critical	0.06	East	15/02/2015	0.45	90192

2.2.2.c. Explanation:

This query is highly efficient because `RowID` serves as the primary key, allowing DynamoDB to directly access specific items without performing a full table scan. Using the `IN` operator enables DynamoDB to fetch multiple items in a single operation, making it faster and more efficient than executing separate queries for each `RowID`. This method is effective as it leverages direct access to specific rows via the primary key, avoiding the need for extra indexing or scans.

2.2.3. How to Execute These Queries Using the DynamoDB (PartiQL Editor):

- Go to the AWS Management Console and open the DynamoDB service.
- Navigate to Tables and select the ProductShipping table (the table name should match exactly).
- Select Explore table items from the left-hand menu and go to the PartiQL editor tab.
- In the editor, enter the PartiQL query mentioned above.
- Click Run to execute the query.
- The results will be displayed in the output window.

2.3. Non Efficiently Executed Queries:

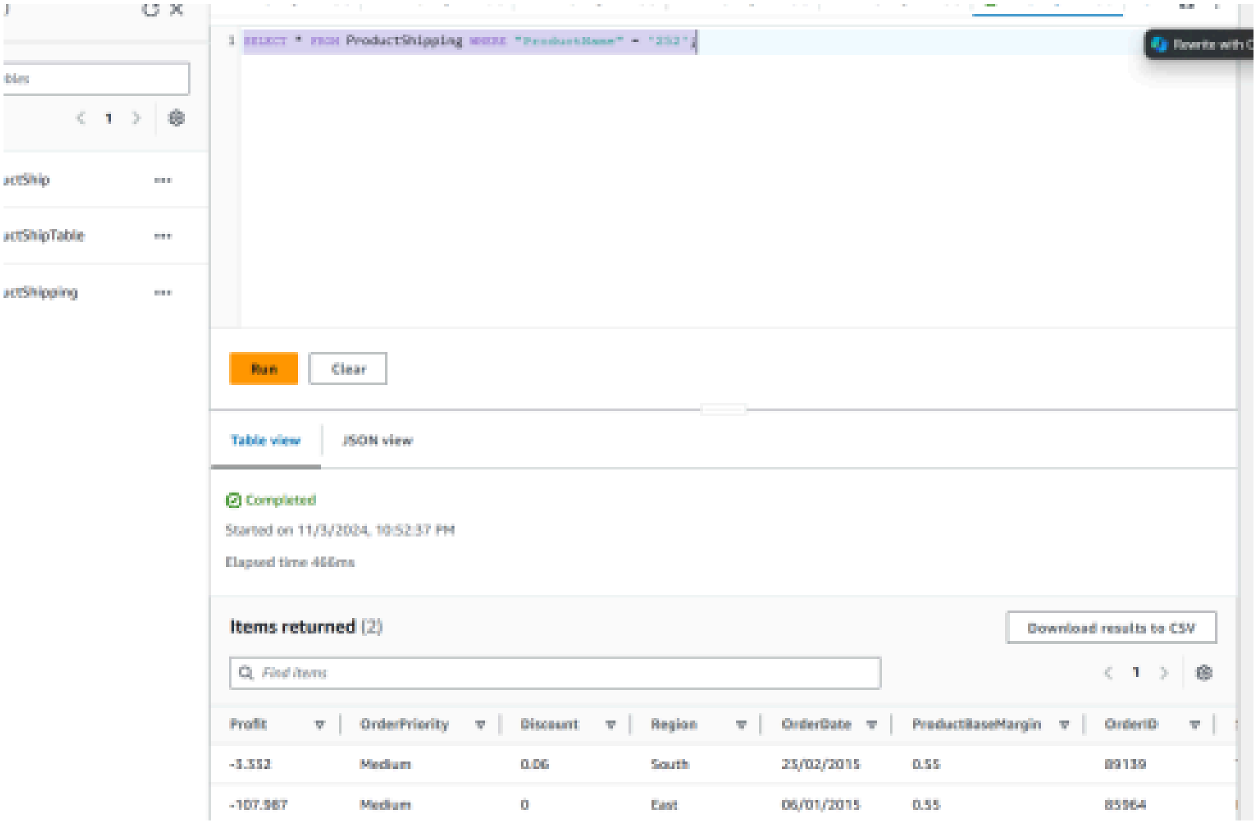
2.3.1. Example 1:

2.3.1.a. Query in English:

Select all from the table for a specific product 252

2.3.1.b. PartiQL Query:

SELECT * FROM ProductShipping WHERE "ProductName" = '252';



```
SELECT * FROM ProductShipping WHERE "ProductName" = '252';
```

Completed

Started on 11/3/2024, 10:52:37 PM

Elapsed time 466ms

Items returned (2)

Profit	OrderPriority	Discount	Region	OrderDate	ProductBaseMargin	OrderID
-3.552	Medium	0.06	South	23/02/2015	0.55	89139
-107.987	Medium	0	East	06/01/2015	0.55	85964

2.3.1.c. Explanation:

Since ProductName is not indexed, DynamoDB may need to scan the table, leading to slower performance as the table grows.

2.3.2. Example 2:

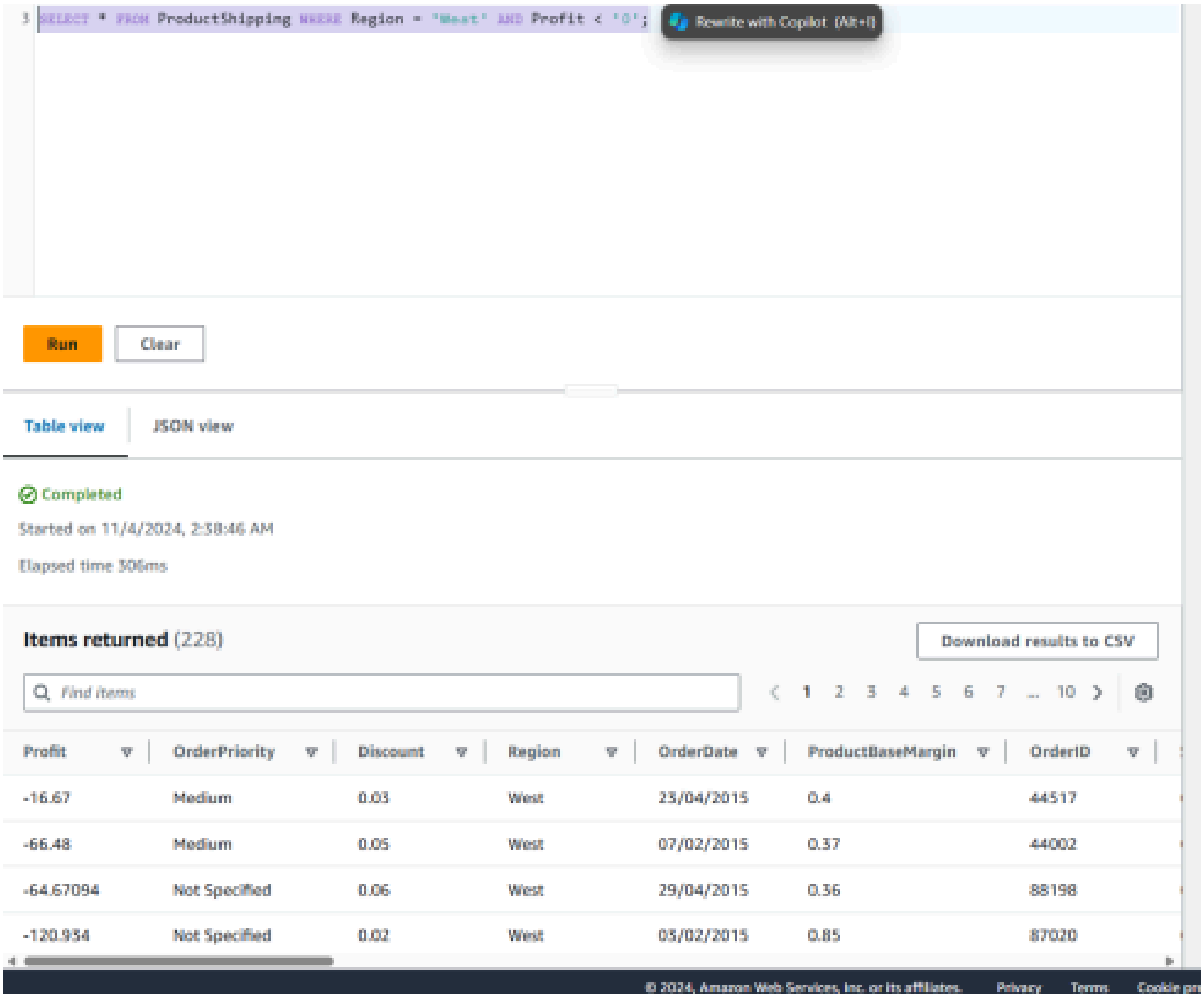
2.3.2.a. Query in English:

Select all from the table for the West region where there are losses

2.3.2.b. PartiiQL Query:

SELECT * FROM ProductShipping WHERE Region = 'West' AND Profit < '0';

Note: For some reason the "Profit" attribute is imported as a string value instead of a number; hence, this query did not run well. However, the concept of the solution is the same. We can increase the efficiency by creating the indexing.



```
SELECT * FROM ProductShipping WHERE Region = 'West' AND Profit < '0';
```

Completed

Started on 11/4/2024, 2:38:46 AM

Elapsed time 306ms

Items returned (228)

Profit	OrderPriority	Discount	Region	OrderDate	ProductBaseMargin	OrderID
-16.67	Medium	0.03	West	23/04/2015	0.4	44517
-66.48	Medium	0.05	West	07/02/2015	0.37	44002
-64.67094	Not Specified	0.06	West	29/04/2015	0.36	88198
-120.954	Not Specified	0.02	West	03/02/2015	0.85	87020

2.3.2.c. Explanation:

With RowID as the partition key and no sort key or additional indexing, DynamoDB cannot optimize the search based on Region or Profit. Every item in the ship table must be scanned to check if it meets the Region = 'West' and Profit < 0 criteria. Since RowID is the partition key, DynamoDB has no direct way to access records based on Region or Profit. It would have to scan each item in the table to find the matching records. Without any indexes on Region or Profit, DynamoDB cannot optimize the search by focusing only on specific segments of the data, leading to slower execution

2.3.3. How to Execute These Queries Using the DynamoDB (PartiQL Editor):

- Go to the AWS Management Console and open the DynamoDB service.
- Navigate to Tables and select the ProductShipping table (the table name should match exactly).
- Select Explore table items from the left-hand menu and go to the PartiQL editor tab.
- In the editor, enter the PartiQL query mentioned above.
- Click Run to execute the query.
- The results will be displayed in the output window.

2.4. How to improve efficiency of query C:

2.4.1. Sample:

SELECT * FROM ProductShipping WHERE Region = 'West' AND Profit < '0';

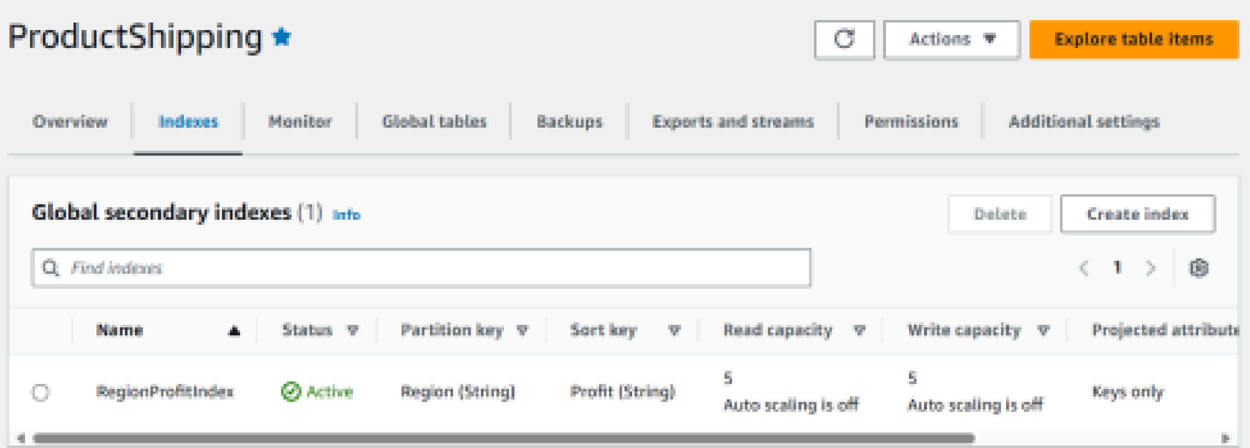
2.4.2. What would you do to make the execution of the queries for the part c more efficient?

To improve execution, we can create a Global Secondary Index (GSI) with "Region" as a Partition Key & "Profit" as a Sort Key. This index will allow direct access to records based on Region, sorted by Profit, making it possible to efficiently retrieve records with Profit < 0 in the 'West' region.

2.4.3. How would this make the execution of the queries faster?

2.4.3.a. First, we will create the GSI:

- Open the AWS DynamoDB console and go to the ship table
- Go to the Indexes tab and click Create Index
- After saving, DynamoDB will create the index and populate it with data from the ship table. The time to complete this step depends on the table's size.
- Once the index is ready, we will update the query to use the GSI we just created.

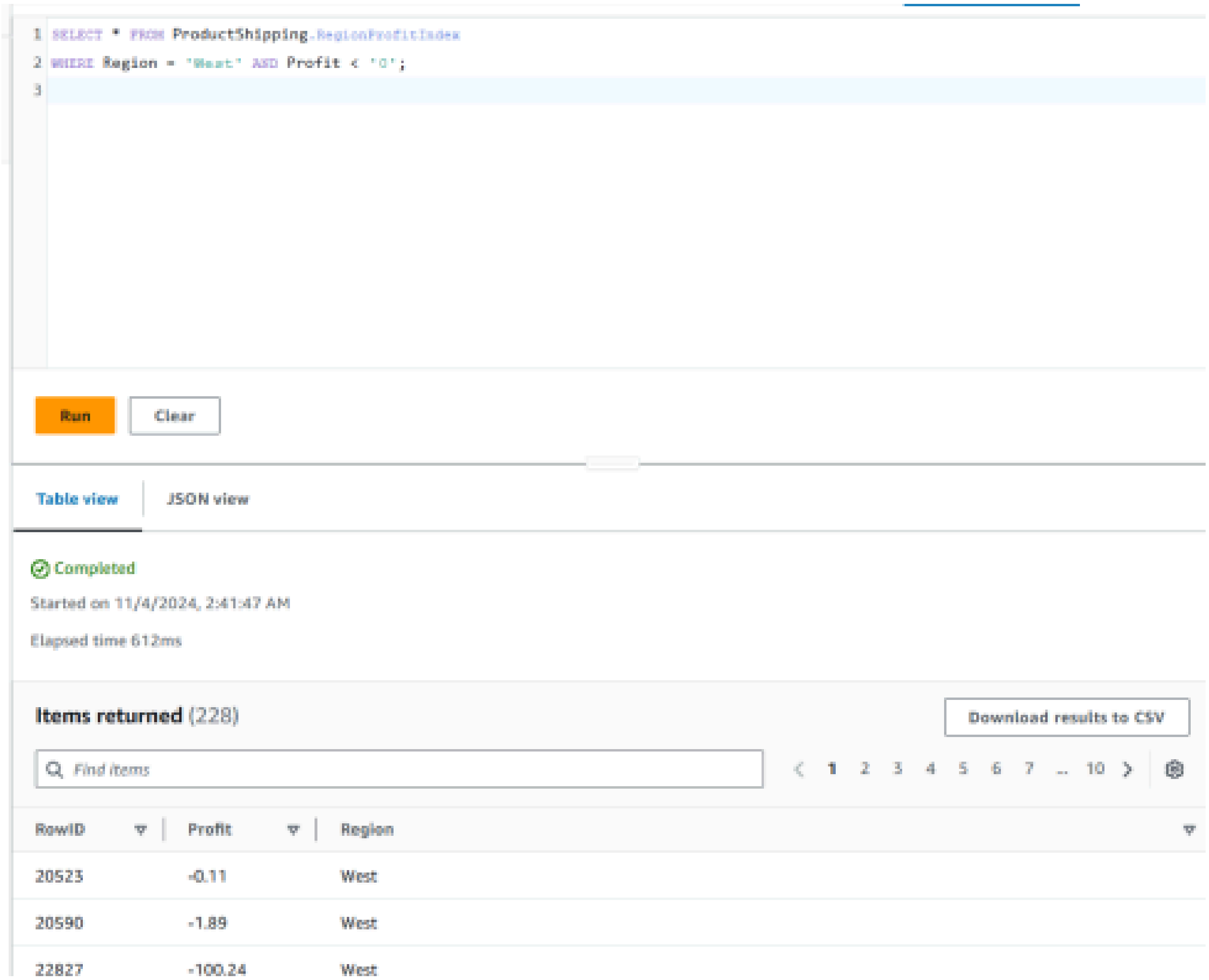


Global secondary indexes (1) Info

	Name ▲	Status	Partition key	Sort key	Read capacity	Write capacity	Projected attribut
○	RegionProfitIndex	✓ Active	Region (String)	Profit (String)	5 Auto scaling is off	5 Auto scaling is off	Keys only

2.4.3.b. Enter updated Query:

SELECT * FROM "ProductShipping"."RegionProfitIndex"
WHERE "Region" = 'West' AND "Profit" < ‘0’;



```
SELECT * FROM ProductShipping.RegionProfitIndex
WHERE Region = 'West' AND Profit < '0';
```

Completed

Started on 11/4/2024, 2:41:47 AM

Elapsed time 612ms

Items returned (228)

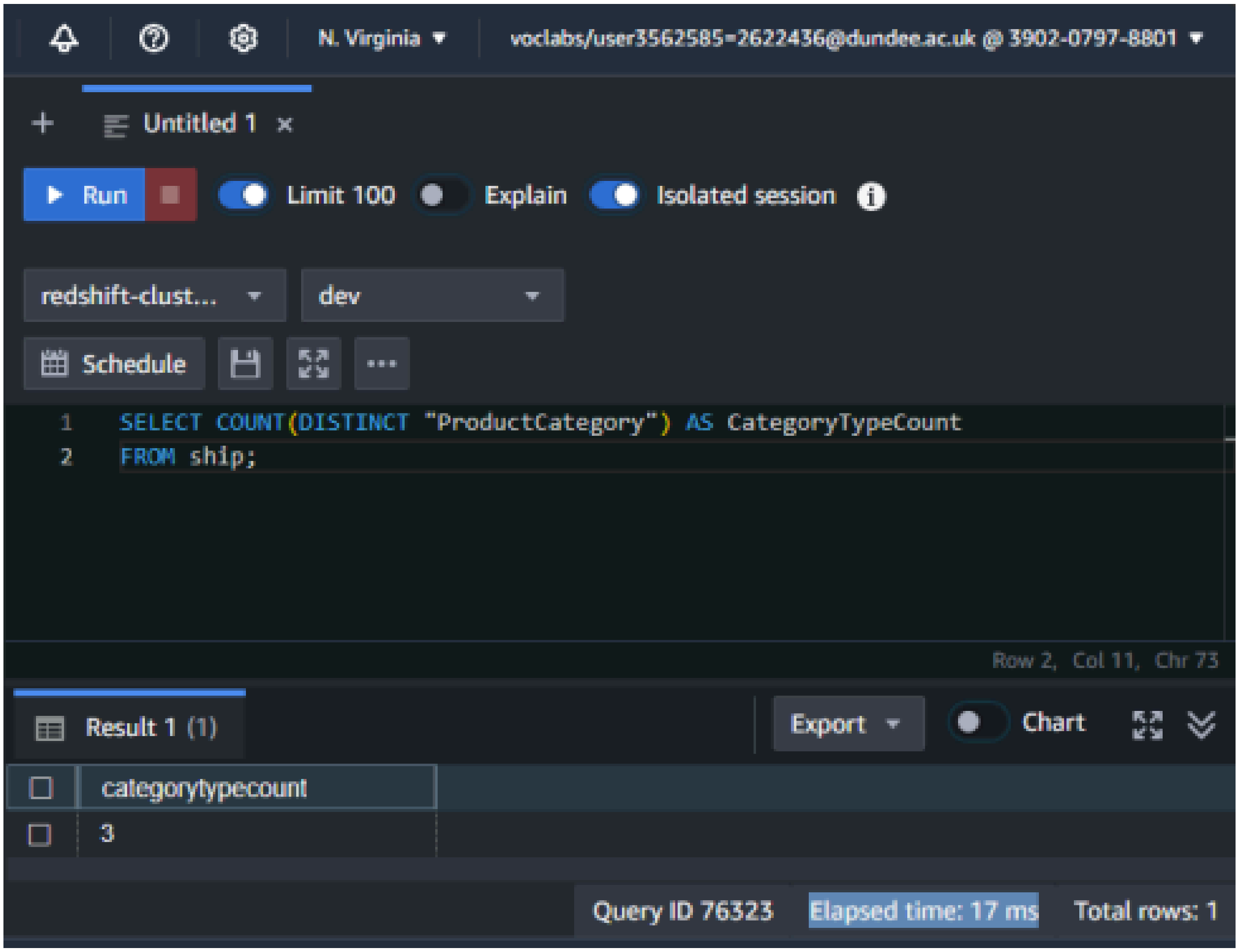
RowID	Profit	Region
20523	-0.11	West
20590	-1.89	West
22827	-100.24	West

3. Data Warehousing:

3.1. How many different product categories do there exist in the dataset?

3.1.a. SQL Query:

SELECT COUNT(DISTINCT "ProductCategory") AS CategoryTypeCount
FROM ship;



```
SELECT COUNT(DISTINCT "ProductCategory") AS CategoryTypeCount
FROM ship;
```

```
□ categorytypecount
□ 3
```

Query ID 76323   Elapsed time: 17 ms   Total rows: 1

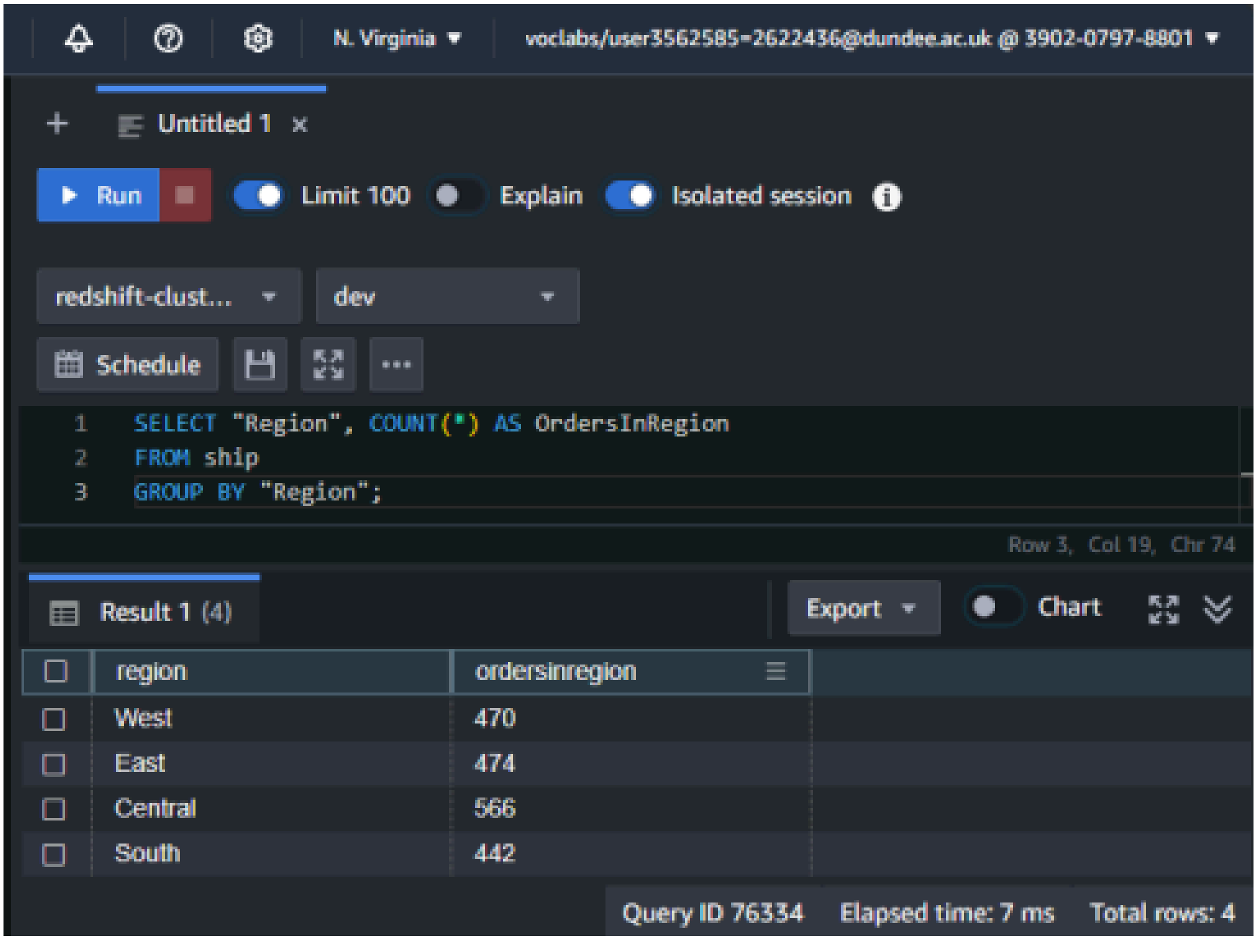
3.1.b. Explanation:

- The COUNT function counts the number of unique entries in the "ProductCategory" column and renames the result column as CategoryTypeCount for easier reference in the output. Using DISTINCT ensures that each category type is only counted once, so duplicate categories will not affect the count.
- FROM ship: This specifies that the data is being retrieved from the ship table.
- The query will return a single value labeled CategoryTypeCount, which is the total number of unique product categories in the ProductCategory column of the ship table.
- The ProductCategory column has three unique categories — "Furniture," "Office Supplies," and "Technology" — the result would be 3

3.2. How many orders were there in each of the regions?

3.2.a. SQL Query:

```
SELECT "Region", COUNT(*) AS OrdersInRegion
FROM ship
GROUP BY "Region";
```



● Limit 100 ○ Explain ● Isolated session

```
SELECT "Region", COUNT(*) AS OrdersInRegion
FROM ship
GROUP BY "Region";
```

Row 3, Col 19, Chr 74

region	ordersinregion
West	470
East	474
Central	566
South	442

Query ID 76334 Elapsed time: 7 ms Total rows: 4

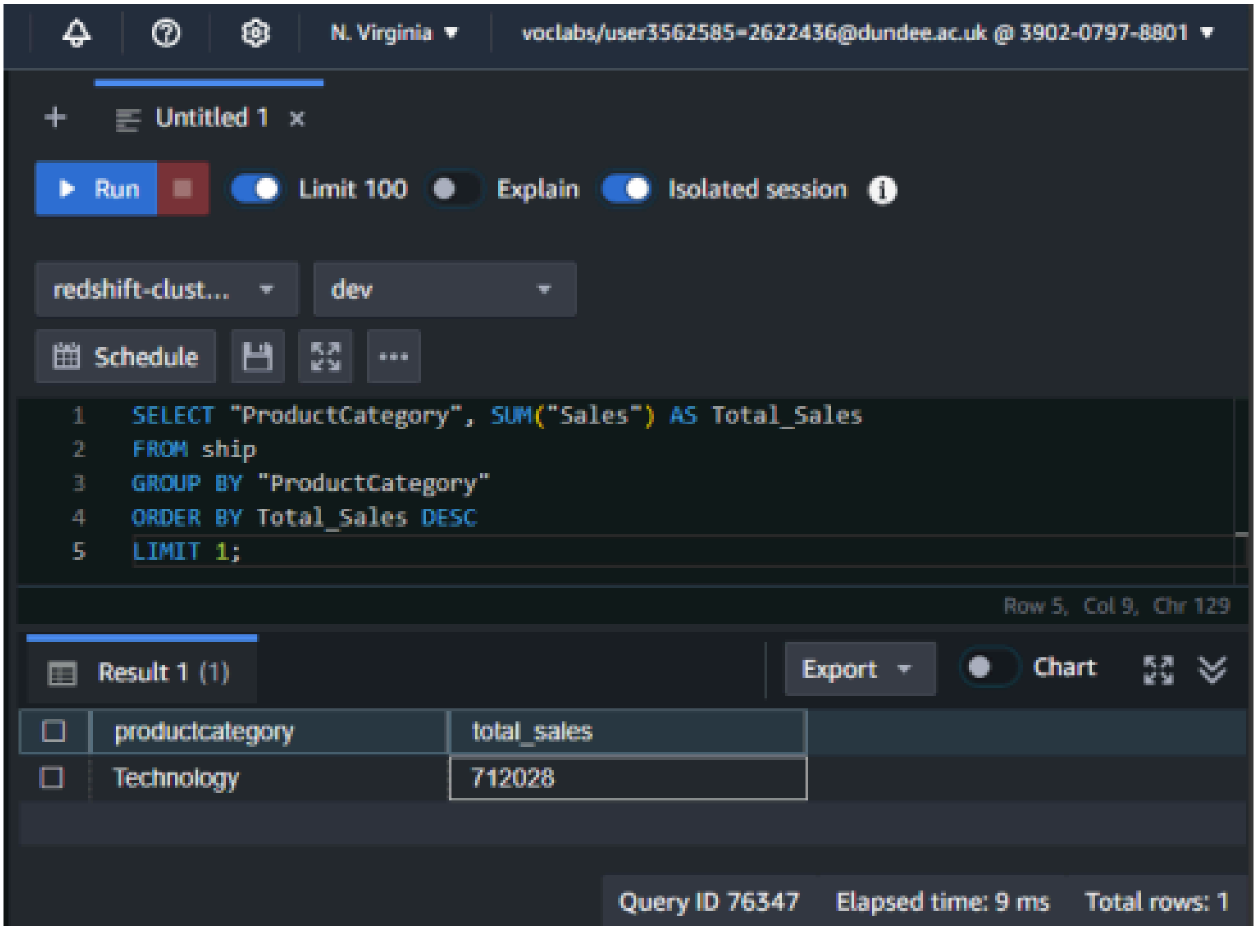
3.2.b. Explanation:

- The query will group the data based on each unique value in Region column.
- Using COUNT(*) here means it will count every row in each group, regardless of column values, to get the total number of orders in each region.
- AS OrdersInRegion will rename the count result as OrdersInRegion
- GROUP BY "Region" will group the data by each unique value in the "Region" column, so that the COUNT(*) function can calculate the number of orders for each region separately.
- The result will display each unique region along with the total number of orders associated with it.

3.3. Which product category has the highest total sales amount?

3.3.a. SQL Query:

```
SELECT "ProductCategory", SUM("Sales") AS Total_Sales
FROM ship
GROUP BY "ProductCategory"
ORDER BY Total_Sales DESC
LIMIT 1;
```



```sql
SELECT "ProductCategory", SUM("Sales") AS Total_Sales
FROM ship
GROUP BY "ProductCategory"
ORDER BY Total_Sales DESC
LIMIT 1;
```

Row 5, Col 9, Chr 129

Result 1 (1)

productcategory	total_sales
Technology	712028

Query ID 76347 Elapsed time: 9 ms Total rows: 1

Notes: To see the Top 3 categories, we can set LIMIT = 3 or we can remove Limit value if we wish to see the total sales of each category.

3.3.b. Explanation:

- This query finds the product category with the highest total sales in the ship table. The SUM function calculates the total sales amount for each product category. It adds up all values in the "Sales" column within each product category. AS Total_Sales renames the summed sales amount as Total_Sales in the output, making it clear that it represents the total sales for each category. GROUP BY "ProductCategory" groups the data by each unique value in the "ProductCategory" column, so that the SUM("Sales") function can calculate the total sales for each category separately. LIMIT 1 limits the result to just the top row, which corresponds to the product category with the highest total sales.
- The query will return a single row containing, ProductCategory & Total_Sales. Here "Technology" has the highest total sales amount.

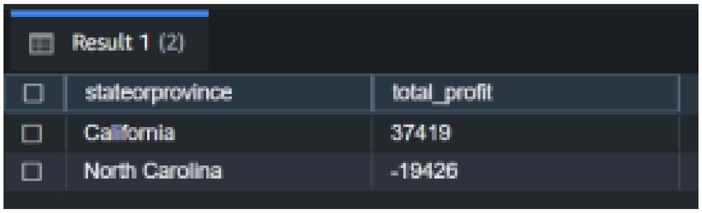
3.4. Which states have recorded the biggest and the smallest profit?

Note: *Initially, I tried to take the max total profit and min total profit, however; the data reflects negative values, which means they are on loss. Hence, to be able to fetch only the profits, I tried to set the values strictly >0*

3.4.a. SQL Query:

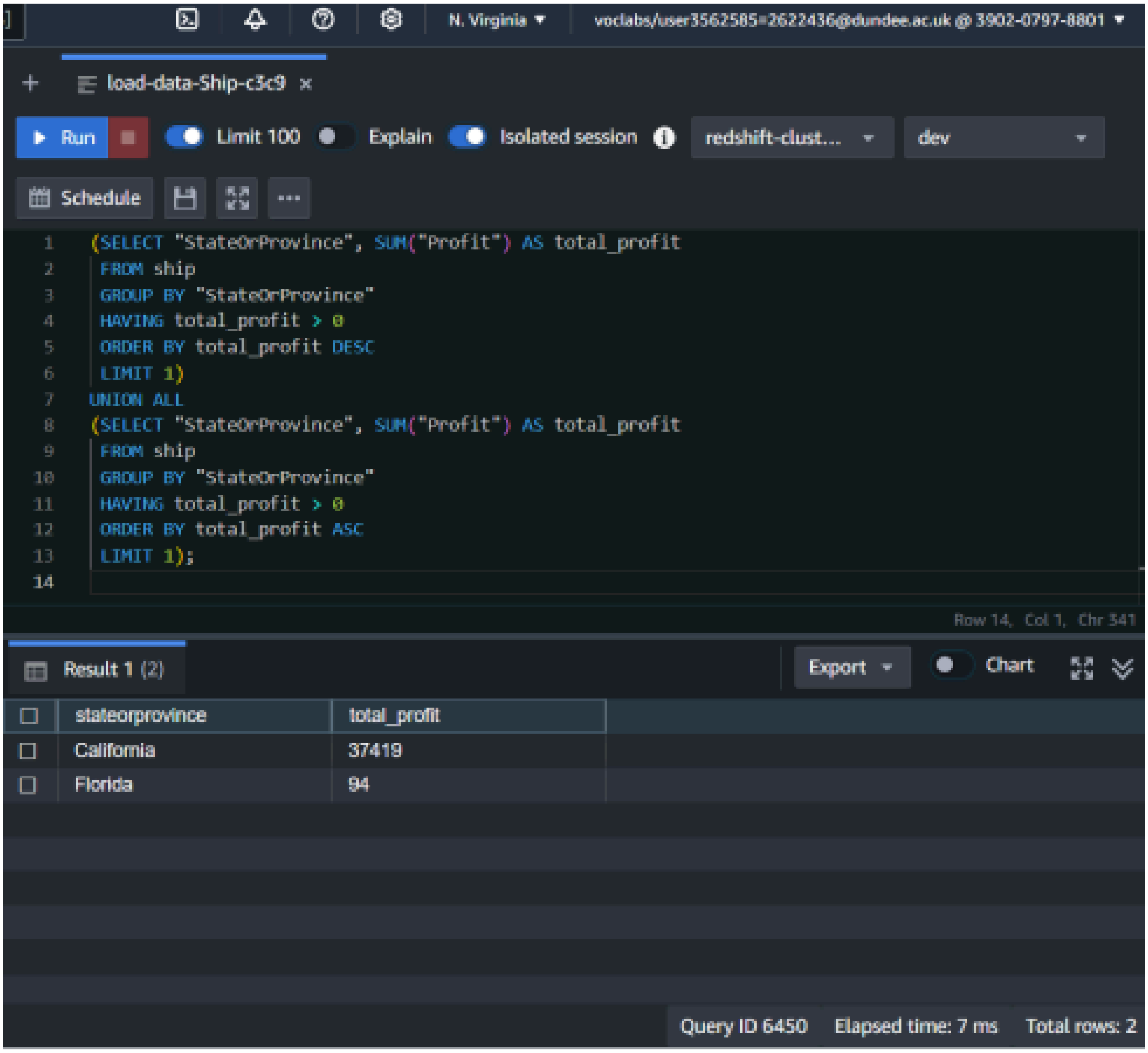
```
(SELECT "StateOrProvince", SUM("Profit") AS total_profit
 FROM ship
 GROUP BY "StateOrProvince"
 HAVING total_profit > 0
 ORDER BY total_profit DESC
 LIMIT 1)
UNION ALL
(SELECT "StateOrProvince", SUM("Profit") AS total_profit
 FROM ship
 GROUP BY "StateOrProvince"
 HAVING total_profit > 0
 ORDER BY total_profit ASC
 LIMIT 1);
```

3.4.b. Before setting the value to >0:



□	stateorprovince	total_profit
□	California	37419
□	North Carolina	-19426

3.4.c. After Setting the value to >0



```
(SELECT "StateOrProvince", SUM("Profit") AS total_profit
 FROM ship
 GROUP BY "StateOrProvince"
 HAVING total_profit > 0
 ORDER BY total_profit DESC
 LIMIT 1)
UNION ALL
(SELECT "StateOrProvince", SUM("Profit") AS total_profit
 FROM ship
 GROUP BY "StateOrProvince"
 HAVING total_profit > 0
 ORDER BY total_profit ASC
 LIMIT 1);
```

□	stateorprovince	total_profit
□	California	37419
□	Florida	94

Query ID 6450 Elapsed time: 7 ms Total rows: 2

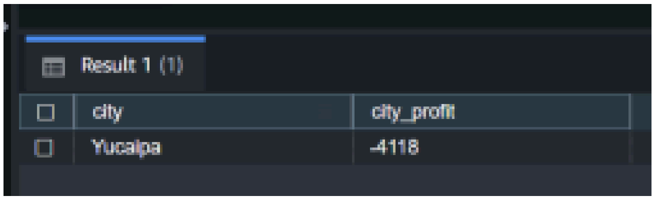
3.4.b. Explanation:

This SQL query finds the state or province with the highest and lowest positive total profit from the `ship` table. The first subquery calculates and returns the state or province with the highest total profit by summing the `Profit` for each `StateOrProvince`, filtering for positive totals, and sorting in descending order, limited to one result. The second subquery similarly calculates the total profit but returns the state or province with the lowest positive total profit, sorting in ascending order and limited to one result. The `UNION ALL` combines both subquery results into a single output, showing both the highest and lowest positive total profits.

3.5. Which city in the most profitable state has the least profit?

3.5.a. SQL Query:

```
WITH MostProfitableState AS (
    SELECT "StateOrProvince"
    FROM ship
    GROUP BY "StateOrProvince"
    ORDER BY SUM("Profit") DESC
    LIMIT 1
)
SELECT "City", SUM("Profit") AS city_profit
FROM ship
 WHERE "StateOrProvince" = (SELECT "StateOrProvince" FROM
   MostProfitableState)
GROUP BY "City"
ORDER BY city_profit ASC
LIMIT 1;
```

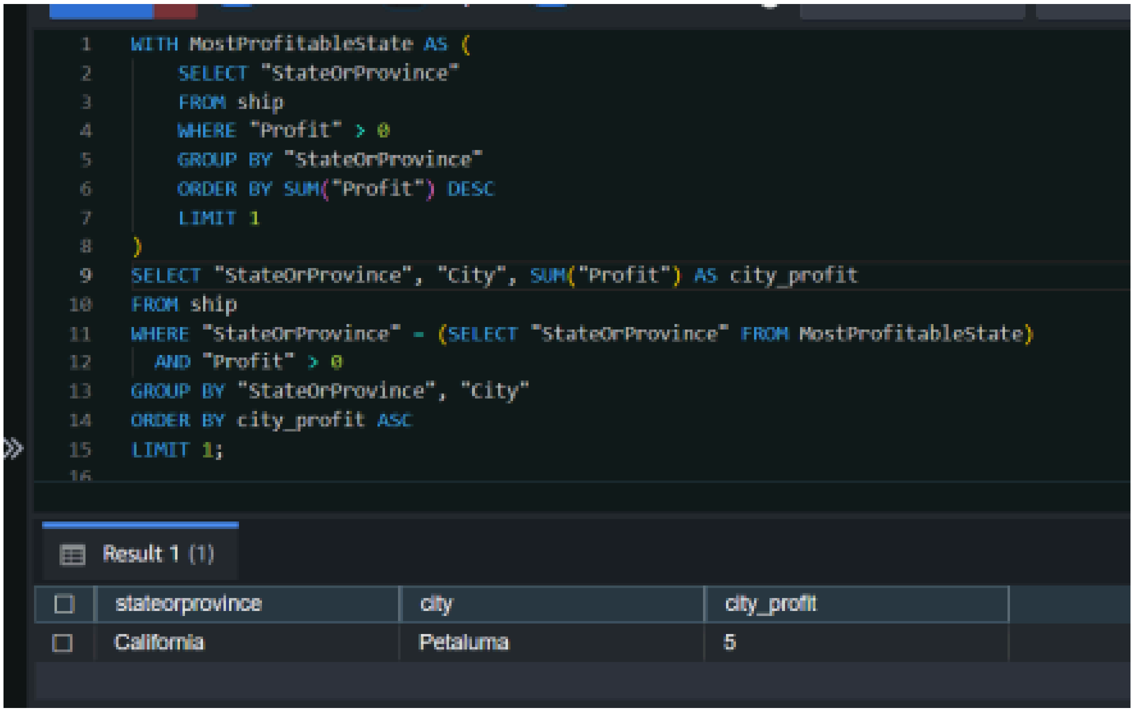


Result 1 (1)

□	city	city_profit
□	Yucaipa	-4118

In this, we can see that the profit displays in negative; hence, it denotes a loss rather. Hence, we will modify our query a bit.

3.5.b. Here is the Modified SQL Query:



```
WITH MostProfitableState AS (
    SELECT "StateOrProvince"
    FROM ship
    WHERE "Profit" > 0
    GROUP BY "StateOrProvince"
    ORDER BY SUM("Profit") DESC
    LIMIT 1
)
SELECT "StateOrProvince", "City", SUM("Profit") AS city_profit
FROM ship
WHERE "StateOrProvince" = (SELECT "StateOrProvince" FROM MostProfitableState)
  AND "Profit" > 0
GROUP BY "StateOrProvince", "City"
ORDER BY city_profit ASC
LIMIT 1;
```

Result 1 (1)

□	stateorprovince	city	city_profit
□	California	Petaluma	5

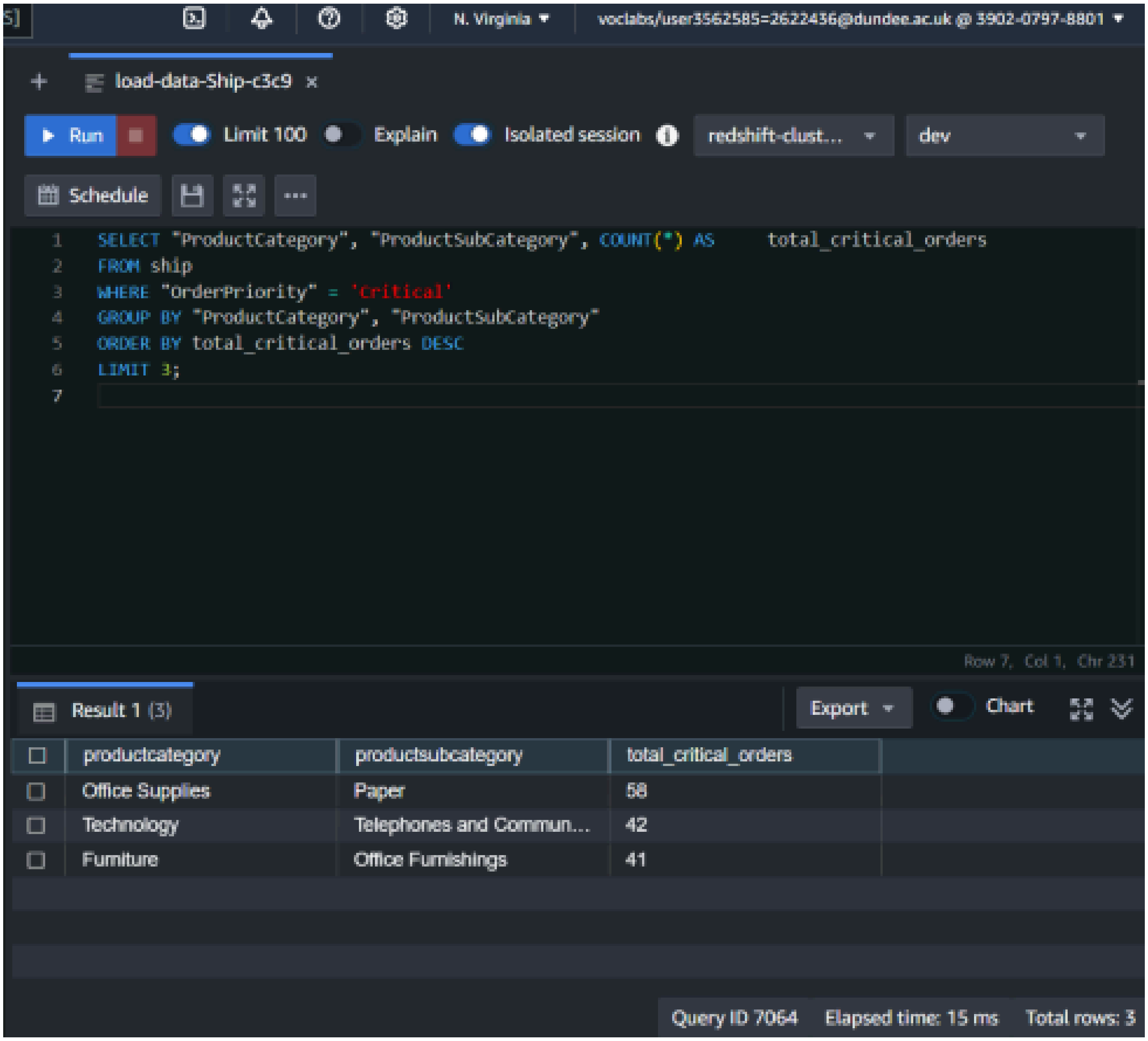
3.5.c. Explanation:

- WITH MostProfitableState AS () calculates the most profitable state by grouping by StateOrProvince, summing the profits, and ordering the results in descending order. LIMIT 1 ensures we only get the state with the highest total profit.
- SELECT "City", SUM("Profit") AS city_profit selects the City and calculate the total profit (SUM("Profit")) for each city within the most profitable state.
- ORDER BY city_profit ASC LIMIT 1: This orders the cities in ascending order by their profit, and LIMIT 1 retrieves the city with the smallest profit.
- WHERE "Profit" > 0 ensures that we only consider records where Profit is positive, both when calculating the most profitable state and when finding the city with the least positive profit.
- This query will return the StateOrProvince, City, and total city_profit for the city with the smallest positive profit in the most profitable state.

3.6. Which product subcategory had the most orders that were of the critical priority?

3.6.a. SQL Query:

```
SELECT "ProductCategory", "ProductSubCategory", COUNT(*) AS
total_critical_orders
FROM ship
HERE "OrderPriority" = 'Critical'
GROUP BY "ProductCategory", "ProductSubCategory"
ORDER BY total_critical_orders DESC
LIMIT 3;
```



Limit 100 ● Explain ○ Isolated session ● redshift-clust... dev

```
SELECT "ProductCategory", "ProductSubCategory", COUNT(*) AS     total_critical_orders
FROM ship
WHERE "OrderPriority" = 'Critical'
GROUP BY "ProductCategory", "ProductSubCategory"
ORDER BY total_critical_orders DESC
LIMIT 3;
```

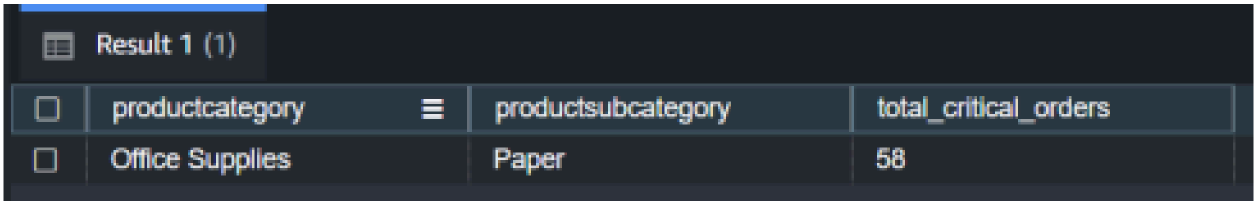
Row 7, Col 1, Chr 231

Result 1 (3)

□	productcategory	productsubcategory	total_critical_orders
□	Office Supplies	Paper	58
□	Technology	Telephones and Commun...	42
□	Furniture	Office Furnishings	41

Query ID 7064 Elapsed time: 15 ms Total rows: 3

If you set the limit to 1, we get the only highest answer:
LIMIT 1;



Result 1 (1)

□	productcategory	productsubcategory	total_critical_orders
□	Office Supplies	Paper	58

3.6.b. Explanation:

- *COUNT()* AS total_critical_orders counts the number of orders for each ProductSubcategory where the priority is "Critical". GROUP BY

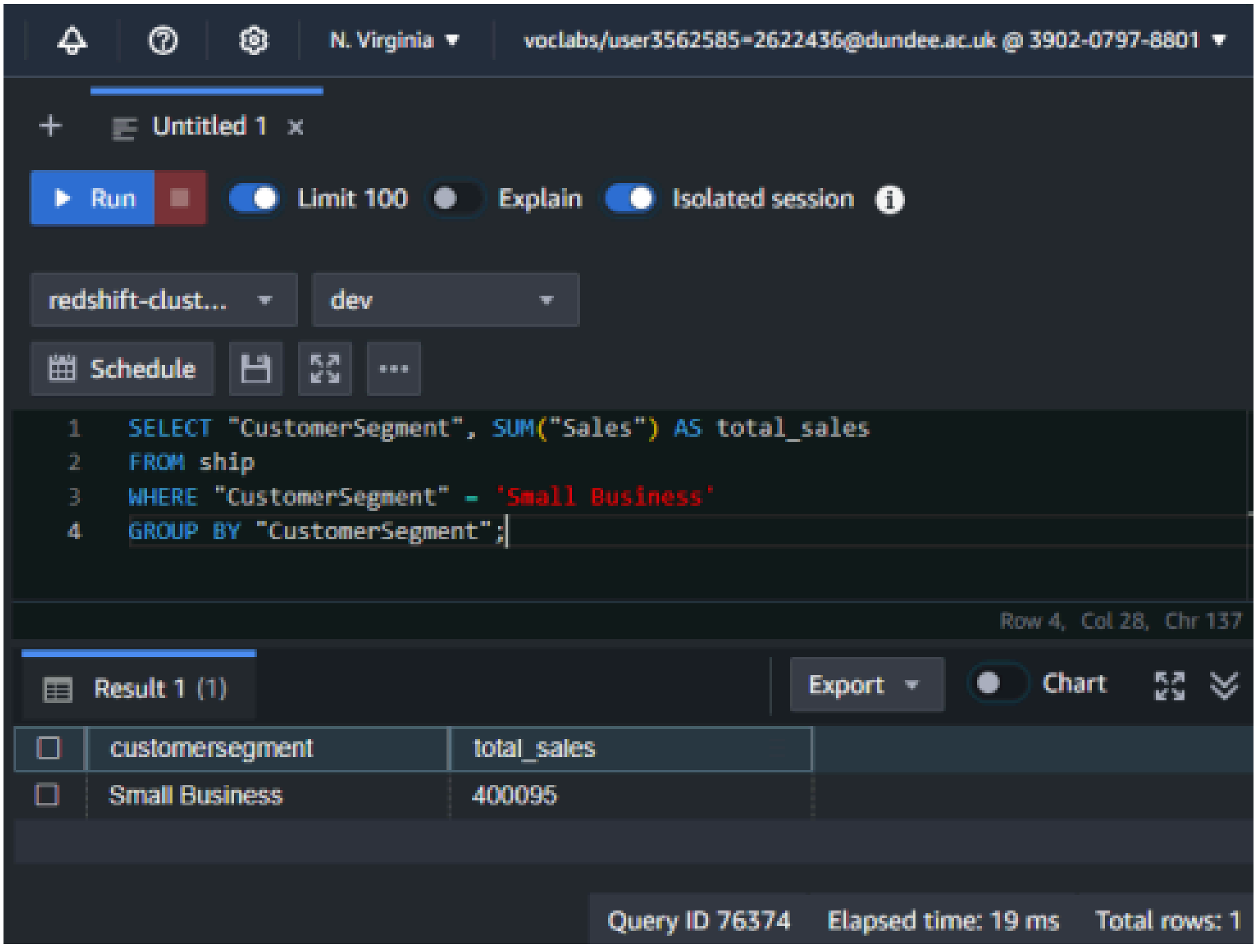
"ProductCategory" and "ProductSubcategory" groups the results by both ProductCategory and ProductSubcategory to get the order count for each unique combination.

- ORDER BY total_critical_orders DESC sorts the results in descending order based on the order count, so the subcategory with the highest number of "Critical" orders is first. LIMIT 1 limits the result to only the top row, showing the subcategory with the highest count of "Critical" orders.
- This query will return the ProductCategory, ProductSubcategory, and the count of critical orders for the subcategory with the most critical priority orders.

3.7. What is the total amount of sales for 'Small Business' customer segment?

3.7.a. SQL Query:

```
SELECT "CustomerSegment", SUM("Sales") AS total_sales
FROM ship
WHERE "CustomerSegment" = 'Small Business'
GROUP BY "CustomerSegment";
```



```
SELECT "CustomerSegment", SUM("Sales") AS total_sales
FROM ship
WHERE "CustomerSegment" = 'Small Business'
GROUP BY "CustomerSegment";
```

Row 4, Col 28, Chr 137

□	customersegment	total_sales
□	Small Business	400095

Query ID 76374 Elapsed time: 19 ms Total rows: 1

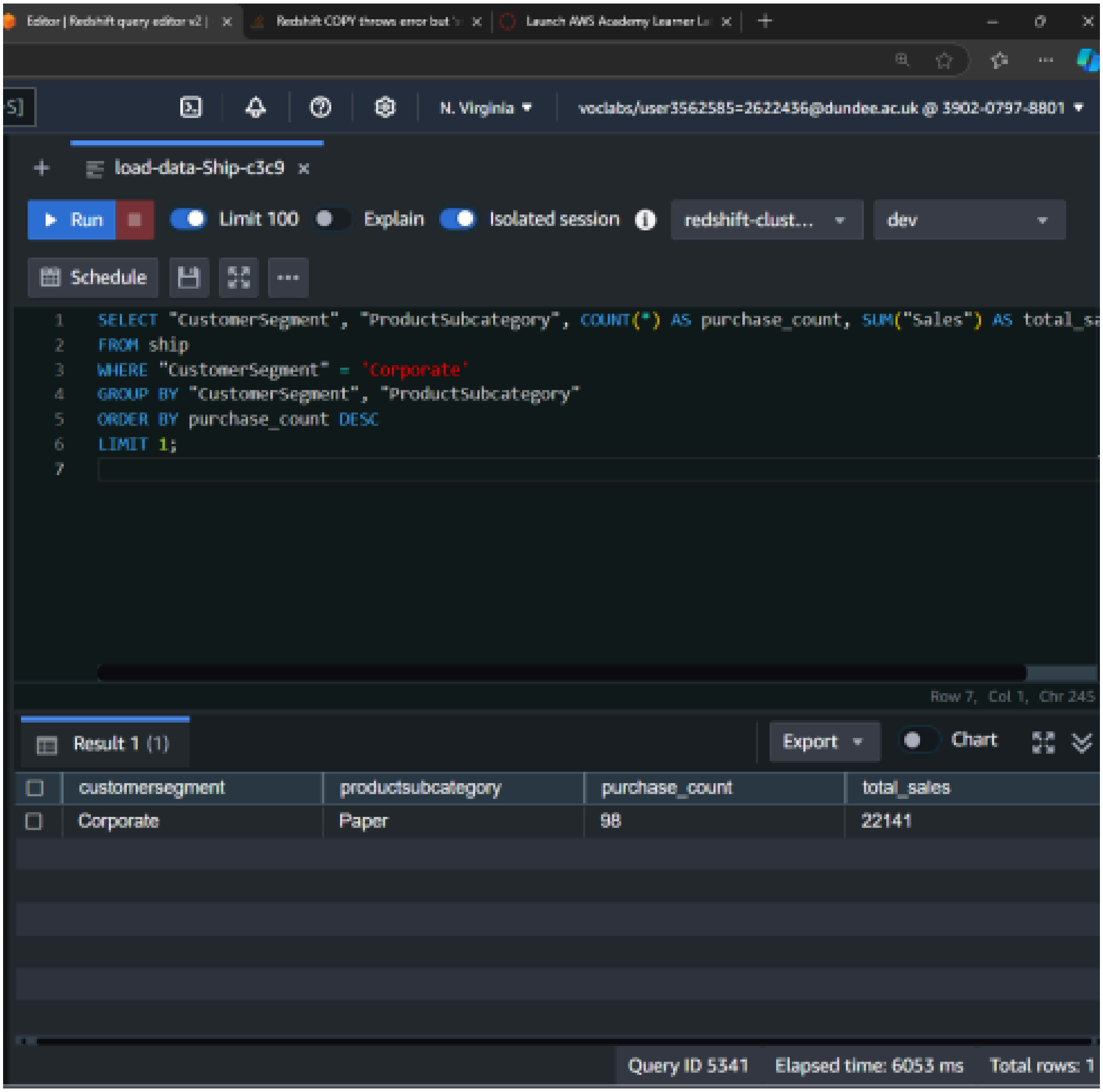
3.7.b. Explanation:

- SUM("Sales") AS total_sales calculates the total amount of sales for all records that match the specified condition. WHERE "CustomerSegment" = 'Small Business' filters the records to only those where the CustomerSegment is "Small Business".
- This query will return a single value, total_sales, which represents the total sales amount for all orders made by customers in the "Small Business" segment.

3.8. What product sub-category was the most often purchased in the ‘Corporate’ customer segment, and what is the total amount of sales in all of these transactions?

3.8.a. SQL Query:

```
SELECT "CustomerSegment", "ProductSubcategory", COUNT(*) AS purchase_count,
SUM("Sales") AS total_sales
FROM ship
WHERE "CustomerSegment" = 'Corporate'
GROUP BY "CustomerSegment", "ProductSubcategory"
ORDER BY purchase_count DESC
LIMIT 1;
```



Editor | Redshift query editor v2 |
Redshift COPY throws error but
Launch AWS Academy Learner La
N. Virginia
voclabs/user3562585=2622436@dundee.ac.uk @ 3902-0797-8801
load-data-Ship-c3c9
Run
Schedule
Export
Chart

Limit 100 ● Explain ○ Isolated session ● redshift-clust... dev

```
SELECT "CustomerSegment", "ProductSubcategory", COUNT(*) AS purchase_count, SUM("Sales") AS total_sa
FROM ship
WHERE "CustomerSegment" = 'Corporate'
GROUP BY "CustomerSegment", "ProductSubcategory"
ORDER BY purchase_count DESC
LIMIT 1;
```

Row 7, Col 1, Chr 245

Result 1 (1)

□	customersegment	productsubcategory	purchase_count	total_sales
□	Corporate	Paper	98	22141

Query ID 5341 Elapsed time: 6053 ms Total rows: 1

3.8.b. Explanation:

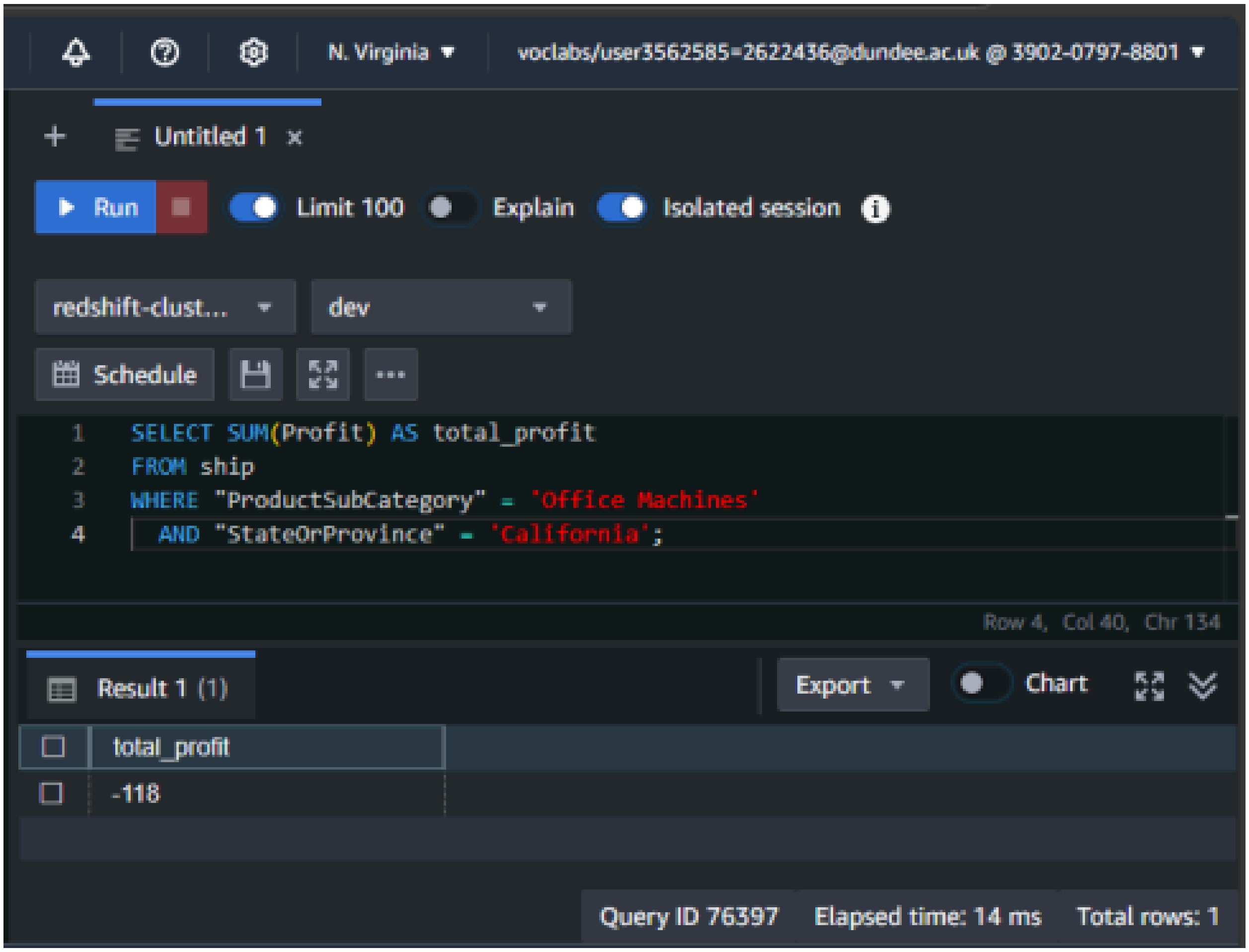
- COUNT(*) AS purchase_count counts the number of purchases for each product sub-category in the "Corporate" segment. SUM("Sales") AS total_sales sums the sales for each sub-category to get the total amount of sales in those transactions.
- WHERE "CustomerSegment" = 'Corporate' filters to only include records where the CustomerSegment is "Corporate". ORDER BY purchase_count DESC sorts the results in descending order of purchase count, so the most frequently purchased sub-category appears first. And LIMIT 1 limits the result to only the top row, showing the most-purchased product sub-category in the "Corporate" segment.

- This query will return the ProductSubcategory with the highest purchase count in the "Corporate" segment, along with the number of purchases (purchase_count) and the total sales amount (total_sales) for that sub-category.

3.9. What is the total profit for all of the products of the 'Office Machines' subcategory in the 'California' state?

3.9.a. SQL Query:

```
SELECT SUM(Profit) AS total_profit
FROM ship
WHERE "ProductSubCategory" = 'Office Machines'
  AND "StateOrProvince" = 'California';
```



```
SELECT SUM(Profit) AS total_profit
FROM ship
WHERE "ProductSubCategory" = 'Office Machines'
 AND "StateOrProvince" = 'California';
```

□	total_profit
□	-118

Query ID 76397 Elapsed time: 14 ms Total rows: 1

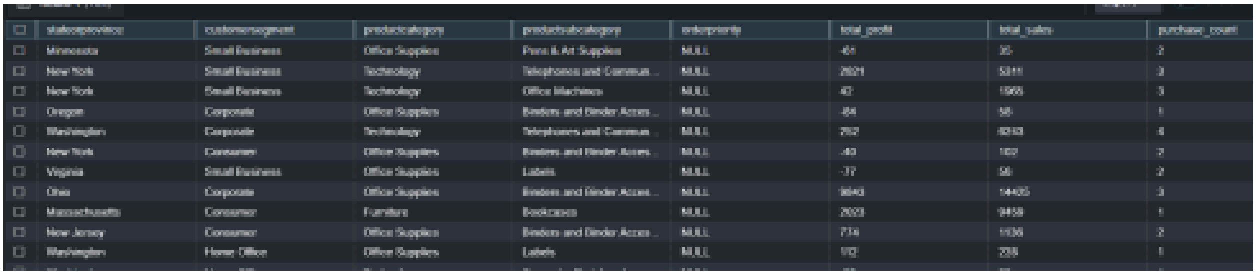
3.9.b. Explanation:

This SQL query calculates the total profit from the `ship` table for products in the 'Office Machines' subcategory sold in the state of California. The `SUM(Profit) AS total_profit` function aggregates the profit values for all relevant rows that match the conditions specified in the `WHERE` clause. The `WHERE` clause filters the results to include only those records where the `ProductSubCategory` is 'Office Machines' and the `StateOrProvince` is 'California'. The output will be a single value representing the total profit generated from the sale of office machines in California. In this case, the value is negative, which means there is a loss in this category.

4. Cube:

4.1. Creating the cube:

```
CREATE TABLE SalesCube AS
SELECT
    "StateOrProvince",
    "CustomerSegment",
    "ProductCategory",
    "ProductSubcategory",
    "OrderPriority",
    SUM("Profit") AS total_profit,
    SUM("Sales") AS total_sales,
    COUNT(*) AS purchase_count
FROM ship
GROUP BY CUBE("StateOrProvince", "CustomerSegment", "ProductCategory",
"ProductSubcategory", "OrderPriority");
```



□	stateorprovince	customersegment	productcategory	productsubcategory	orderpriority	total_profit	total_sales	purchase_count
□	Minnesota	Small Business	Office Supplies	Pens & Art Supplies	NULL	-61	35	2
□	New York	Small Business	Technology	Telephones and Commun...	NULL	2621	5311	3
□	New York	Small Business	Technology	Office Machines	NULL	42	1965	3
□	Oregon	Corporate	Office Supplies	Binders and Binder Acces...	NULL	-64	58	1
□	Washington	Corporate	Technology	Telephones and Commun...	NULL	252	8243	4
□	New York	Consumer	Office Supplies	Binders and Binder Acces...	NULL	-40	102	2
□	Virginia	Small Business	Office Supplies	Labels	NULL	-77	56	2
□	Ohio	Corporate	Office Supplies	Binders and Binder Acces...	NULL	9840	14405	3
□	Massachusetts	Consumer	Furniture	Bookcases	NULL	2023	9459	1
□	New Jersey	Consumer	Office Supplies	Binders and Binder Acces...	NULL	774	1136	2
□	Washington	Home Office	Office Supplies	Labels	NULL	112	228	1

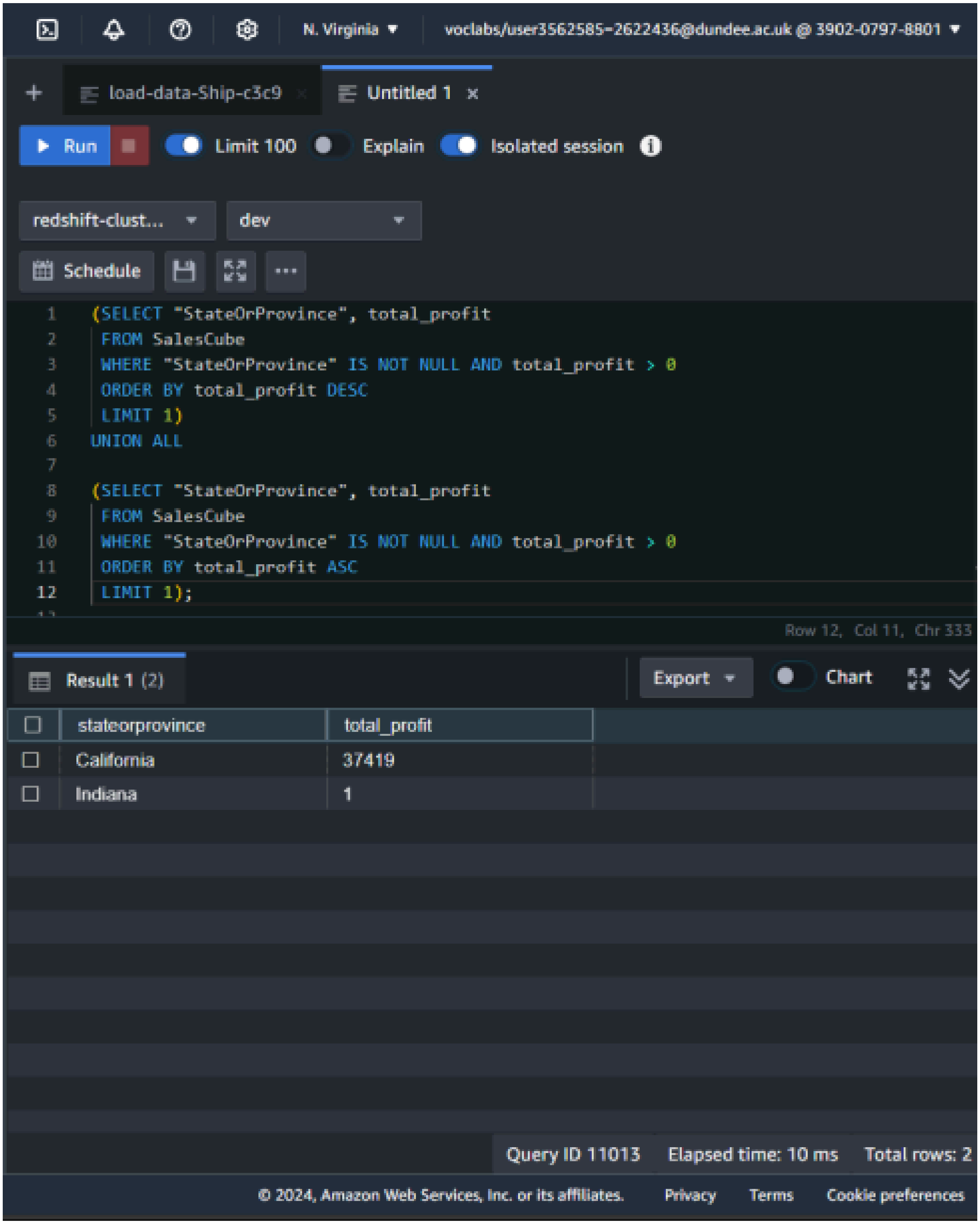
The SQL statement creates a new table, `SalesCube`, which aggregates sales data from the `ship` table using a data cube structure. By using `GROUP BY CUBE`, the query generates summary data for all possible combinations of five attributes: `"StateOrProvince"`, `"CustomerSegment"`, `"ProductCategory"`, `"ProductSubcategory"`, and `"OrderPriority"`. This means that the resulting table will contain subtotals for each combination, from individual attributes (like just `"StateOrProvince"`) to combinations of two, three, or more attributes, up to an overall total. For each combination, the query calculates `total_profit` (the sum of `"Profit"`), `total_sales` (the sum of `"Sales"`), and `purchase_count` (the count of rows that match each grouping). The resulting `SalesCube` table allows for multi-dimensional analysis, enabling users to drill down into detailed sales performance and profit metrics across various segments, locations, product categories, and priorities. This cube structure is especially useful for reporting and analytics, as it provides pre-aggregated data that can be quickly queried to answer a variety of business questions about profitability and sales trends.

4.1.a. Which states have recorded the biggest and the smallest profit?

```
(SELECT "StateOrProvince", total_profit
 FROM SalesCube
 WHERE "StateOrProvince" IS NOT NULL AND total_profit > 0
 ORDER BY total_profit DESC
 LIMIT 1)
UNION ALL

(SELECT "StateOrProvince", total_profit
 FROM SalesCube
 WHERE "StateOrProvince" IS NOT NULL AND total_profit > 0
 ORDER BY total_profit ASC
 LIMIT 1);
```

This SQL code uses the `SalesCube` table to find and display the `StateOrProvince` with the highest and lowest positive total profit. The first `SELECT` statement retrieves the state with the maximum total profit by filtering out any rows where `StateOrProvince` is `NULL` and ensuring `total_profit` is greater than 0. It orders the results in descending order and limits the output to one row (the top result). The second `SELECT` statement performs a similar function but orders the results in ascending order, returning the state with the lowest positive total profit. The `UNION ALL` combines these two results into a single output that displays both the state with the highest profit and the state with the lowest positive profit, creating a comprehensive comparison in one query.



```
(SELECT "StateOrProvince", total_profit
 FROM SalesCube
 WHERE "StateOrProvince" IS NOT NULL AND total_profit > 0
 ORDER BY total_profit DESC
 LIMIT 1)
UNION ALL

(SELECT "StateOrProvince", total_profit
 FROM SalesCube
 WHERE "StateOrProvince" IS NOT NULL AND total_profit > 0
 ORDER BY total_profit ASC
 LIMIT 1);
```

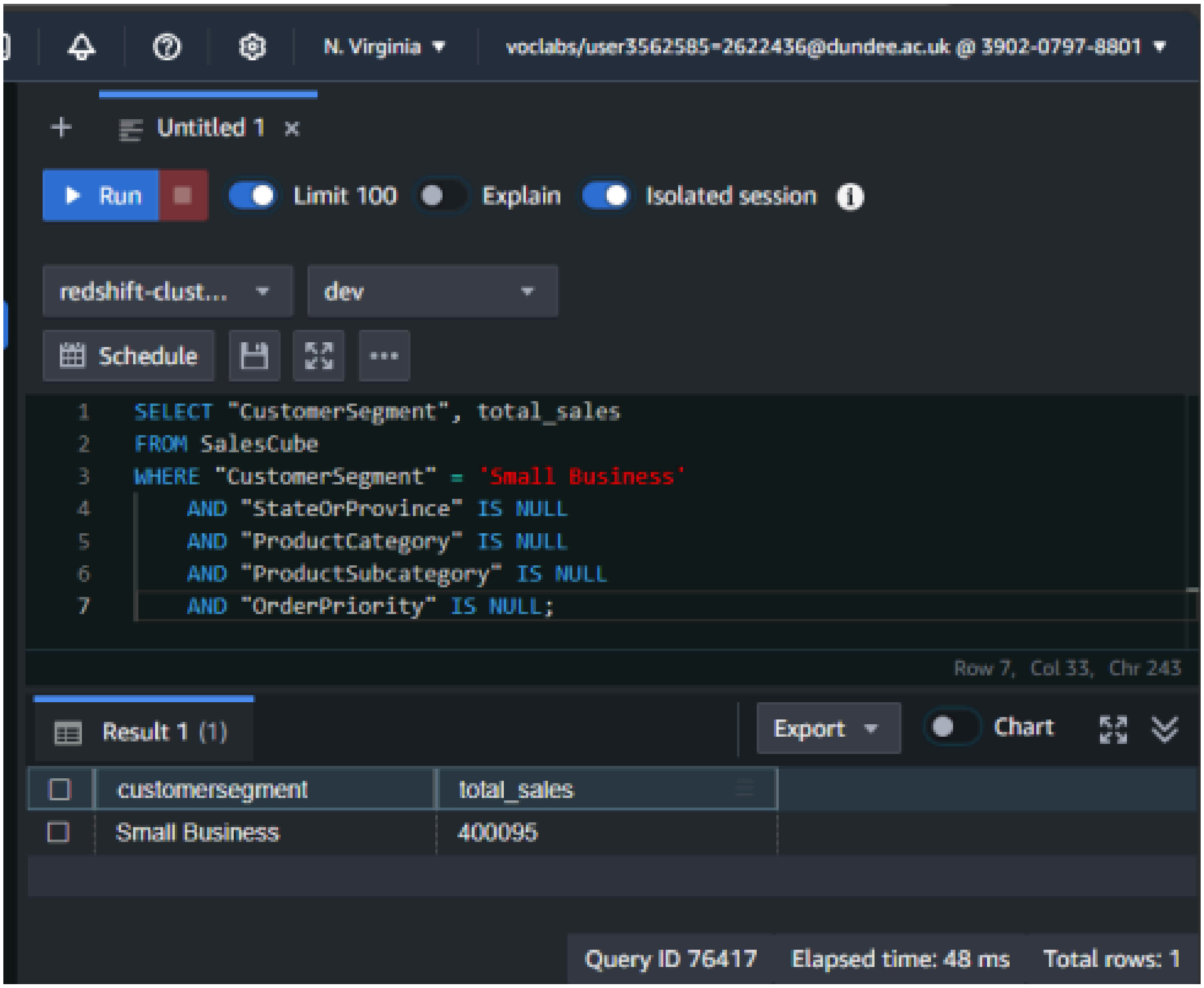
□	stateorprovince	total_profit
□	California	37419
□	Indiana	1

Query ID 11013 Elapsed time: 10 ms Total rows: 2

4.1.b. What is the total amount of sales for the ‘Small Business’ customer segment?

```
SELECT "CustomerSegment", total_sales
FROM SalesCube
WHERE "CustomerSegment" = 'Small Business'
    AND "StateOrProvince" IS NULL
    AND "ProductCategory" IS NULL
    AND "ProductSubcategory" IS NULL
    AND "OrderPriority" IS NULL;
```

This SQL query retrieves the total sales for the 'Small Business' customer segment from the `SalesCube` table, specifically for records where all other dimensions—`StateOrProvince`, `ProductCategory`, `ProductSubcategory`, and `OrderPriority`—are `NULL`. This filtering ensures that only the aggregate total sales value for the 'Small Business' segment, without any breakdown by location, product type, or order priority, is returned. The result provides a high-level summary of sales attributed solely to the 'Small Business' segment, making it useful for understanding overall performance within this segment, independent of other attributes.



```
SELECT "CustomerSegment", total_sales
FROM SalesCube
WHERE "CustomerSegment" = 'Small Business'
    AND "StateOrProvince" IS NULL
    AND "ProductCategory" IS NULL
    AND "ProductSubcategory" IS NULL
    AND "OrderPriority" IS NULL;
```

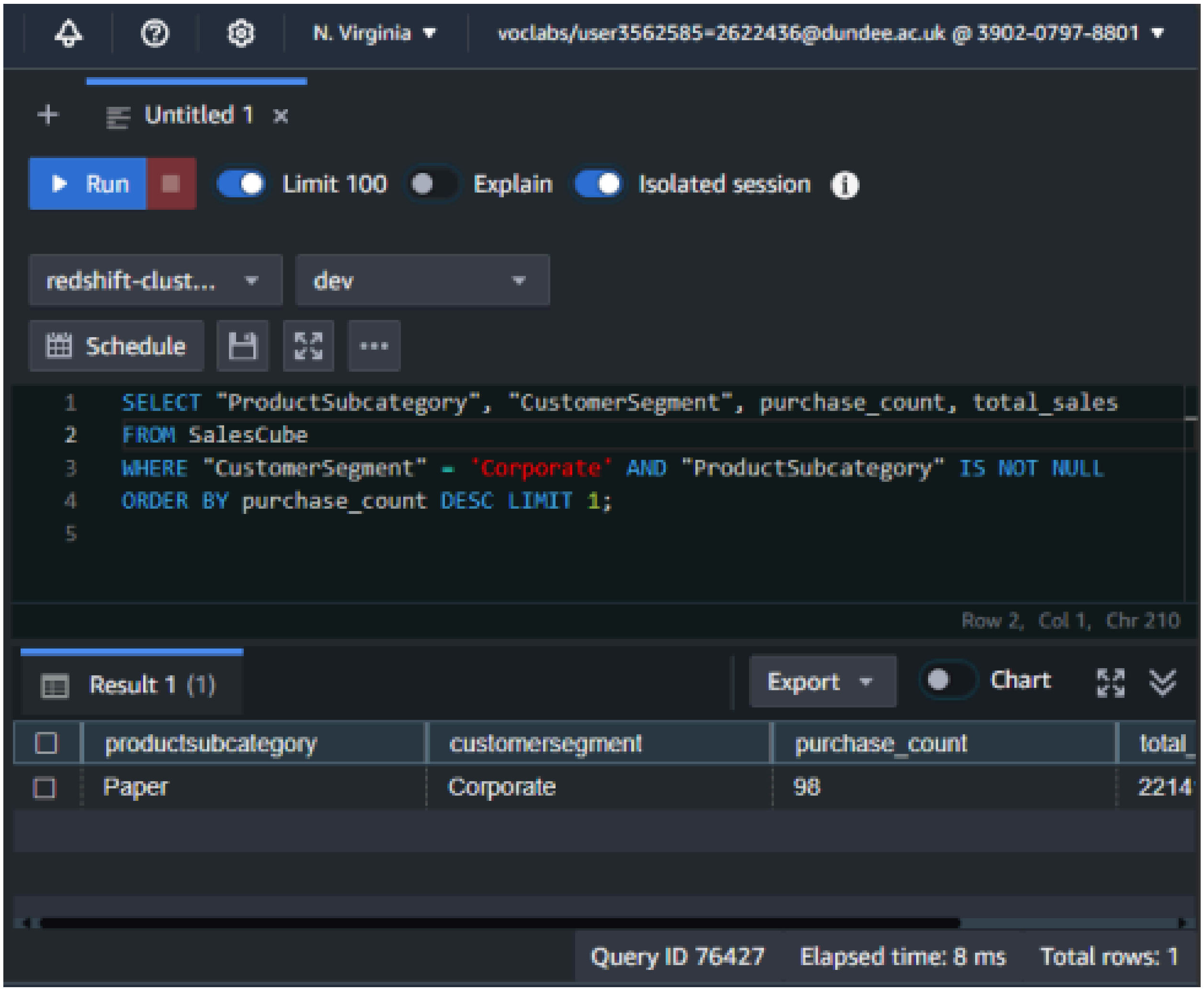
customersegment	total_sales
Small Business	400095

Query ID 76417 Elapsed time: 48 ms Total rows: 1

4.1.c. What product sub-category was the most often purchased in the 'Corporate' customer segment, and what is the total amount of sales in all of these transactions?

```
SELECT "ProductSubcategory", "CustomerSegment", purchase_count, total_sales
FROM SalesCube
WHERE "CustomerSegment" = 'Corporate' AND "ProductSubcategory" IS NOT NULL
ORDER BY purchase_count DESC LIMIT 1;
```

This SQL query retrieves data from the `SalesCube` table to find the most purchased product subcategory by the 'Corporate' customer segment. It selects the columns `ProductSubcategory`, `CustomerSegment`, `purchase_count`, and `total_sales`, filtering for rows where `CustomerSegment` is 'Corporate' and `ProductSubcategory` is specified (not `NULL`). The `ORDER BY purchase_count DESC` sorts the results in descending order based on `purchase_count`, ensuring that the product subcategory with the highest purchase count for the 'Corporate' segment appears at the top. The `LIMIT 1` clause returns only the top result, providing insight into which product subcategory is most frequently purchased by corporate customers and its corresponding sales.



```
SELECT "ProductSubcategory", "CustomerSegment", purchase_count, total_sales
FROM SalesCube
WHERE "CustomerSegment" = 'Corporate' AND "ProductSubcategory" IS NOT NULL
ORDER BY purchase_count DESC LIMIT 1;
```

Row 2, Col 1, Chr 210

Result 1 (1)

□	productsubcategory	customersegment	purchase_count	total_
□	Paper	Corporate	98	2214

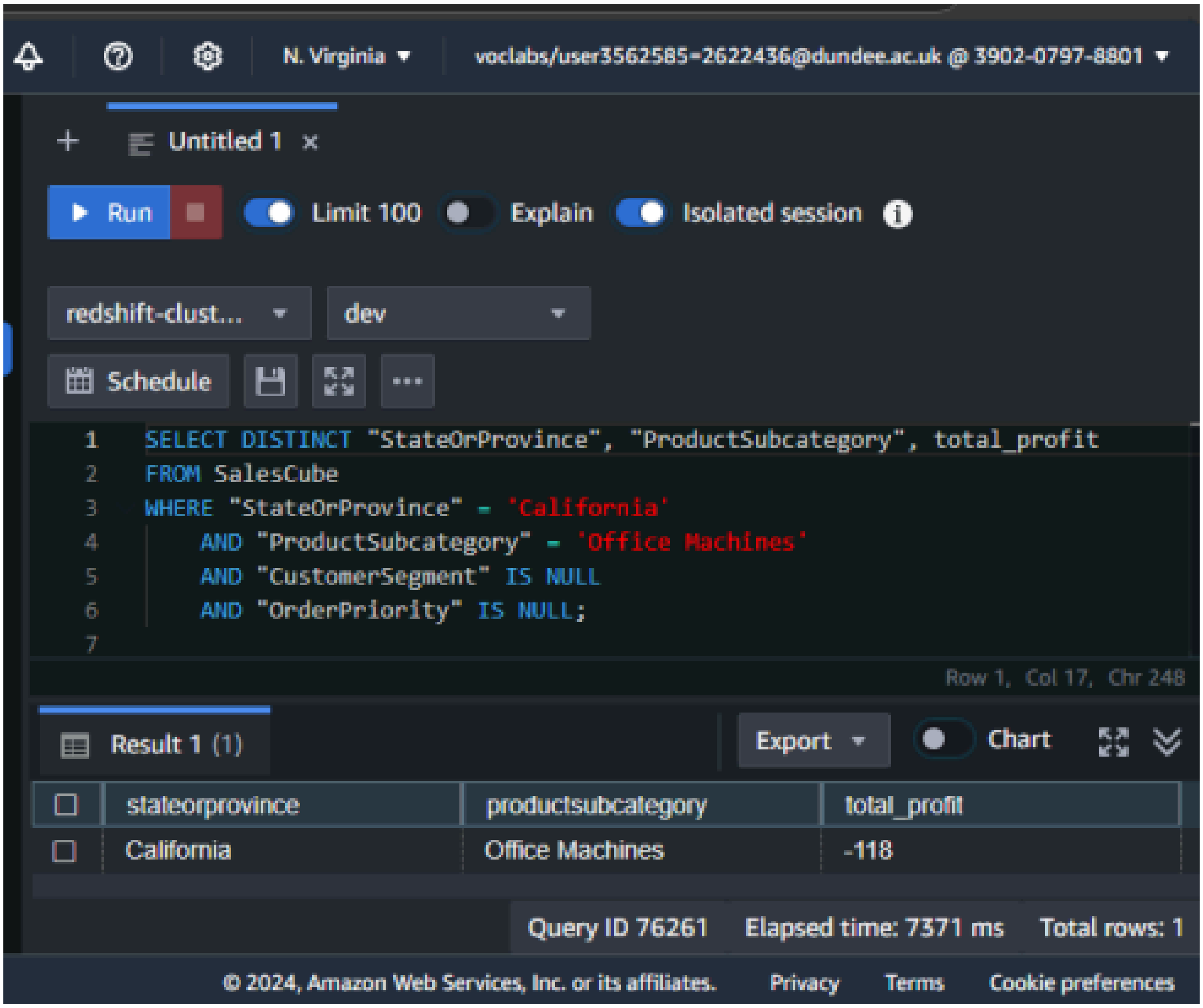
Query ID 76427 Elapsed time: 8 ms Total rows: 1

4.1.d. What is the total profit for all products of the 'Office Machines' subcategory in the 'California' state?

```
SELECT DISTINCT "StateOrProvince", "ProductSubcategory", total_profit
FROM SalesCube
WHERE "StateOrProvince" = 'California'
   AND "ProductSubcategory" = 'Office Machines'
   AND "CustomerSegment" IS NULL
   AND "OrderPriority" IS NULL;
```

This SQL query retrieves the total profit from the `SalesCube` table specifically for the 'Office Machines' product subcategory in the state of California. It selects `StateOrProvince`, `ProductSubcategory`, and `total_profit`, applying filters to focus only on entries where `StateOrProvince` is 'California' and `ProductSubcategory` is

'Office Machines', while also ensuring that `CustomerSegment` and `OrderPriority` are `NULL`. This filtering narrows the result to a high-level profit summary for office machines in California, without breaking down by customer segment or order priority. The output provides a targeted view of profitability for this specific product subcategory within California.



```
SELECT DISTINCT "StateOrProvince", "ProductSubcategory", total_profit
FROM SalesCube
WHERE "StateOrProvince" = 'California'
    AND "ProductSubcategory" = 'Office Machines'
    AND "CustomerSegment" IS NULL
    AND "OrderPriority" IS NULL;
```

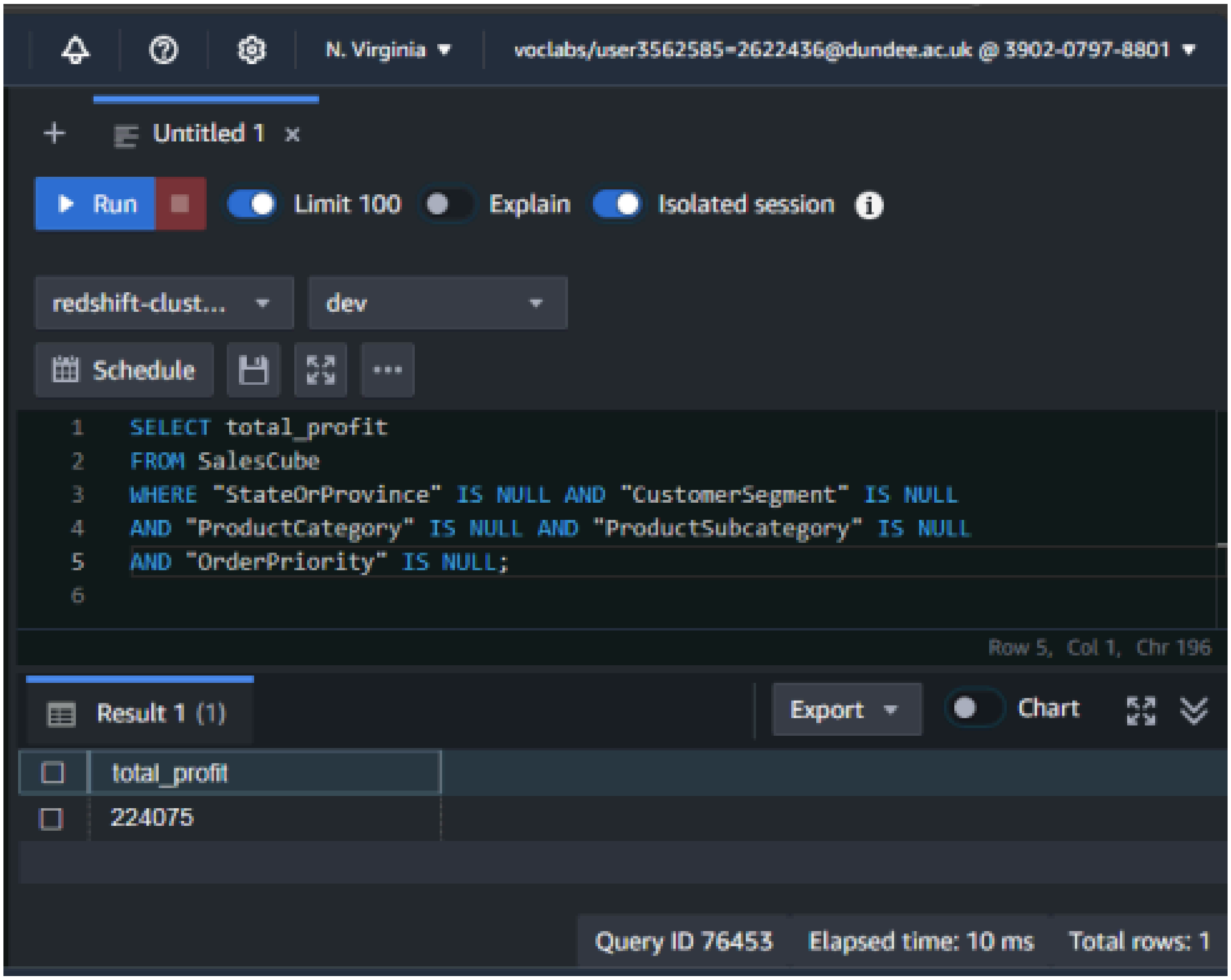
stateorprovince	productsubcategory	total_profit
California	Office Machines	-118

Query ID 76261 Elapsed time: 7371 ms Total rows: 1

4.1.e.What is the total profit achieved over all transactions?

```
SELECT total_profit
FROM SalesCube
WHERE "StateOrProvince" IS NULL AND "CustomerSegment" IS NULL AND
"ProductCategory" IS NULL AND "ProductSubcategory" IS NULL AND "OrderPriority" IS
NULL;
```

The code first creates a cube table called `SalesCube` by aggregating data from a table named `ship`. Using the `GROUP BY CUBE` clause, it calculates aggregate measures—`total_profit`, `total_sales`, and `purchase_count`—for every possible combination of the specified dimensions: `"StateOrProvince"`, `"CustomerSegment"`, `"ProductCategory"`, `"ProductSubcategory"`, and `"OrderPriority"`. The `CUBE` operation generates subtotals across all combinations of these columns, including a grand total across all rows where all dimensions are `NULL` (indicating no specific grouping applied). In the subsequent `SELECT` query, only the row where each of these five dimensions is `NULL` is retrieved. This specific row represents the grand total for `total_profit` across the entire dataset, as it includes sales and profit data for all records without any filtering by individual dimension values. This approach helps obtain a comprehensive summary of profits regardless of specific categories or regions.



```
SELECT total_profit
FROM SalesCube
WHERE "StateOrProvince" IS NULL AND "CustomerSegment" IS NULL
AND "ProductCategory" IS NULL AND "ProductSubcategory" IS NULL
AND "OrderPriority" IS NULL;
```

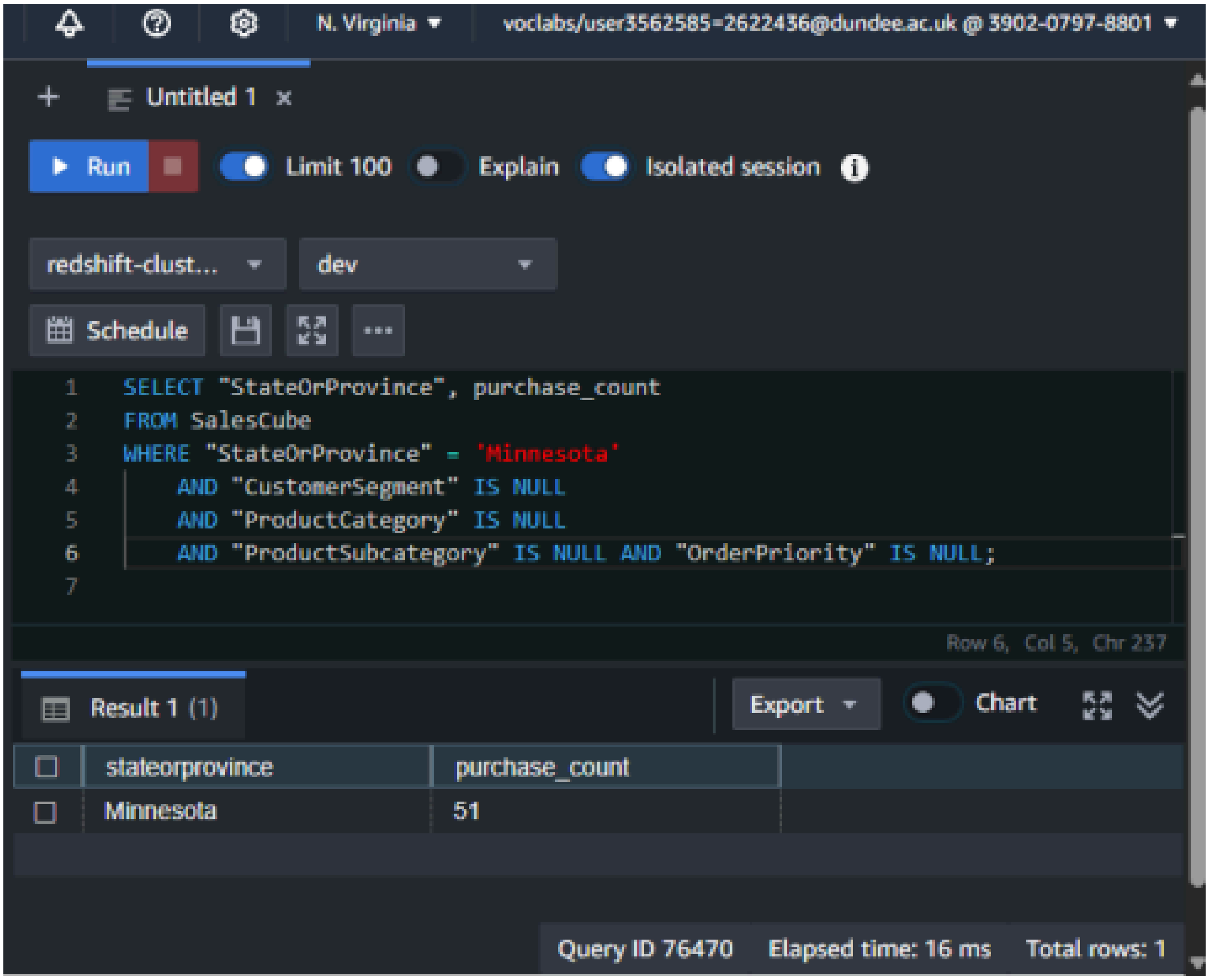
```
total_profit
224075
```

4.1.f. How many transactions were there in the Minnesota state?

SELECT "StateOrProvince", purchase_count
FROM SalesCube

WHERE "StateOrProvince" = 'Minnesota' AND "CustomerSegment" IS NULL AND "ProductCategory" IS NULL AND "ProductSubcategory" IS NULL AND "OrderPriority" IS NULL;

This code retrieves data from the `SalesCube` table specifically for the `"StateOrProvince"` of 'Minnesota'. By specifying `WHERE "StateOrProvince" = 'Minnesota'`, the query focuses only on rows where the data is aggregated at the state level for Minnesota, while keeping the other dimensions (`"CustomerSegment"`, `"ProductCategory"`, `"ProductSubcategory"`, and `"OrderPriority"`) set to `NULL`. This combination of `NULL` values in the other fields indicates that the result shows totals for all customer segments, product categories, subcategories, and order priorities in Minnesota without breaking down by these additional attributes. The query returns two columns: `"StateOrProvince"` (which will display as 'Minnesota' in this case) and `purchase_count`, representing the total count of purchases for all records related to Minnesota. This approach provides an aggregate view of purchase count for the entire state without subdividing into smaller segments.



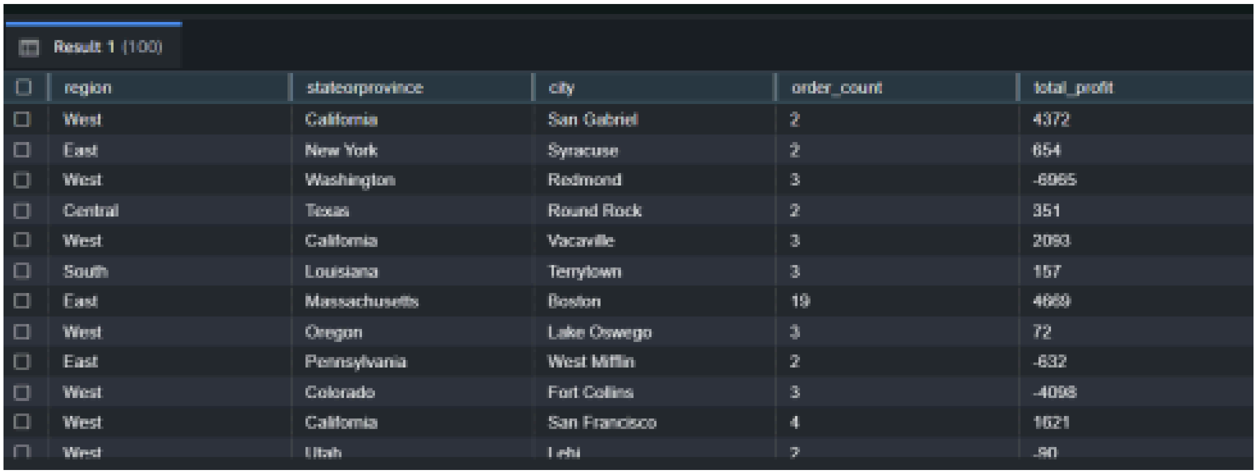
```
SELECT "StateOrProvince", purchase_count
FROM SalesCube
WHERE "StateOrProvince" = 'Minnesota'
    AND "CustomerSegment" IS NULL
    AND "ProductCategory" IS NULL
    AND "ProductSubcategory" IS NULL AND "OrderPriority" IS NULL;
```

stateorprovince	purchase_count
Minnesota	51

Query ID 76470 Elapsed time: 16 ms Total rows: 1

4.2. Create another cube for rollup and drilldown operations:

```
   CREATE TABLE RegionProfitCube AS
SELECT
   "Region",
   "StateOrProvince",
   "City",
   COUNT(*) AS order_count,
   SUM("Profit") AS total_profit
FROM ship
GROUP BY CUBE("Region", "StateOrProvince", "City");
```



□	region	stateorprovince	city	order_count	total_profit
□	West	California	San Gabriel	2	4372
□	East	New York	Syracuse	2	654
□	West	Washington	Redmond	3	-6965
□	Central	Texas	Round Rock	2	351
□	West	California	Vacaville	3	2093
□	South	Louisiana	Terrytown	3	157
□	East	Massachusetts	Boston	19	4669
□	West	Oregon	Lake Oswego	3	72
□	East	Pennsylvania	West Mifflin	2	-632
□	West	Colorado	Fort Collins	3	-4098
□	West	California	San Francisco	4	1621
□	West	Utah	Lehi	2	-90

To support roll-up and drill-down operations for questions about the number of orders in each region, the state with the biggest and smallest profit, and the city in the most profitable state with the least profit, we can create a cube that aggregates data by Region, StateOrProvince, and City.

This cube allows for roll-up (e.g., from city to state to region) and drill-down operations (e.g., from region to state to city), giving flexibility to answer each of the mentioned questions efficiently.

Region: For aggregating data at the regional level to answer the number of orders per region.
StateOrProvince: For identifying the state with the largest and smallest profit.
City: For drilling down to the city level within the most profitable state to find the city with the least profit.

4.3. Explain what the benefit is of creating data cubes in parts b) and c) and why are the queries on these data cubes more efficient than queries on the top-level table, as in the part a). What are the drawbacks of creating these cubes?

4.3.1. Benefits of Creating Data Cubes in Parts (b) and c :

Improved Query Performance: By pre-aggregating data at various levels (e.g., by Region, StateOrProvince, City, CustomerSegment, ProductCategory, etc.,), these data cubes reduce the need to perform extensive calculations at query time. Instead, the results are largely precomputed, enabling faster retrieval.

Optimized Roll-Up and Drill-Down: Data cubes provide a multi-dimensional view of the data, which facilitates efficient roll-up (summarizing to higher levels) and drill-down (detailed analysis at lower levels). For example, part (b) focuses on customer segments and product categories, useful for sales and profitability analysis, while part (c) is tailored for geographical analysis. This structure aligns well with specific analytical needs, making queries more efficient for each specific use case.

Reduced Data Scanning: Since the cube structures in parts (b) and (c) only store aggregated results rather than every individual transaction, queries on these cubes require less data to be scanned. For instance, querying the total profit by state or the number of orders by region can be performed directly on the aggregated data, rather than on the raw data of millions of individual transactions, which would be needed in the top-level table.

4.3.2. Why Queries on These Data Cubes Are More Efficient Than on the Top-Level Table (Part a):

Reduced Data Volume: The data cubes store pre-aggregated metrics at a higher level of granularity, so the volume of data queried is significantly less. For instance, calculating total profit by state on the top-level table would require scanning and summing individual rows for every transaction within each state. In contrast, the data cube provides the profit total for each state directly.

Pre-computed Aggregations: Since key aggregations (such as SUM for profit or COUNT for order count) are computed ahead of time, the database doesn't need to perform these calculations during each query execution. This saves on processing time and resources, especially for complex calculations and repeated queries.

Optimized Indexing and Partitioning: Each cube is designed to focus on dimensions relevant to specific queries, such as customer segment analysis or regional profitability. This makes it possible to use indexing and partitioning strategies tailored to those specific dimensions, improving access times for these queries.

4.3.3. Drawbacks of the cubes:

Storage Overhead: Creating multiple data cubes increases storage requirements, as each cube is an additional copy of the data with different levels of aggregation. Although the storage for a cube is generally less than the raw data, it still consumes extra space.

Increased Maintenance Complexity: When data changes (e.g., new transactions or updates to existing records), each cube must be recalculated or incrementally updated. This can increase the complexity of data maintenance and require additional processing resources to keep all cubes synchronized with the latest data.

Potential Redundancy: Certain dimensions or aggregations may overlap between cubes, leading to redundancy. For example, both parts (b) and (c) might include StateOrProvince in their aggregation levels, creating duplicate storage of similar data in different contexts.

Limited Flexibility: Data cubes are optimized for specific types of queries, which means they may not support ad-hoc queries or analyses that fall outside the predefined aggregations. If a new analysis requirement emerges that isn't covered by the existing cubes, a new cube might need to be created, or queries may need to run on the more general top-level table.