

Steven Evans
SRE220000
BUAN 6335
Snowflake Lab 3

CUSTOMER TABLE CREATED IN SNOWSIGHT

Search objects

BLUEJAY_DB

BOA_DB

CAMEL_DB

CATFISH_DB

CAT_DB

CHEETAH_DB

CHIPMUNK_DB

COBRA_DB

INFORMATION_SCHEMA

MY_SCHEMA

PUBLIC

Tables

CUSTOMER

CUSTOMER

150K Rows

C_CUSTKEY

C_FIRSTNAME

C_LASTNAME

CHIPMUNK_DB.PUBLIC

Settings

```
67
68
69 -- 3.1.4 Create some data to query.
70 -- Here we'll create a table and populate it with data from the customer
71 -- table. We'll also suspend the virtual warehouse to make sure our
72 -- subsequent query doesn't just pull from cached data.
73
74 CREATE TABLE customer AS
75 SELECT
76     c_custkey
```

Results

Chart

	C_CUSTKEY	C_FIRSTNAME	C_LASTNAME
1	10332	Lori	Bridges
2	18972	Otto	Cox
3	87683	Sierra	Morales
4	135845	Henry	Smith
5	146127	Peter	Van Zandt
6	56423	Patty	Ichinose
7	61727	Gina	Janeschild
8	40153	Wally	Gardner
9	40719	John	Gatsby
10	34700	Wilfredo	Fontanilla
11	123630	Yuki	Rahal
12	63108	Sebastian	Jax
13	69189	John	Kirby

Query Details

Query duration

627ms

Rows

150K

Query ID

01b87269-0004-30ff-0...

Show more

C_CUSTKEY

C_FIRSTNAME

Lori

Henry

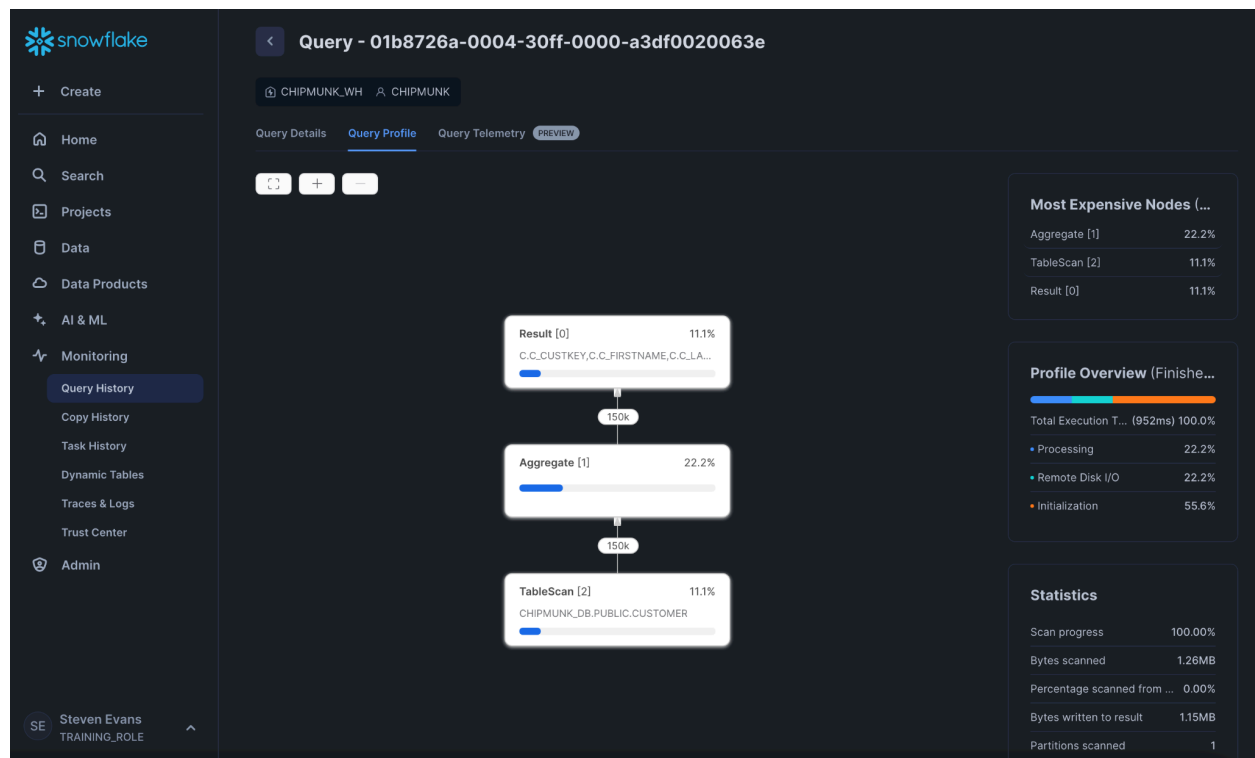
Gina

385

385

Ask Copilot

VIEWING THE QUERY PROFILE FOR SELECT DISTINCT * FROM CUSTOMER C;

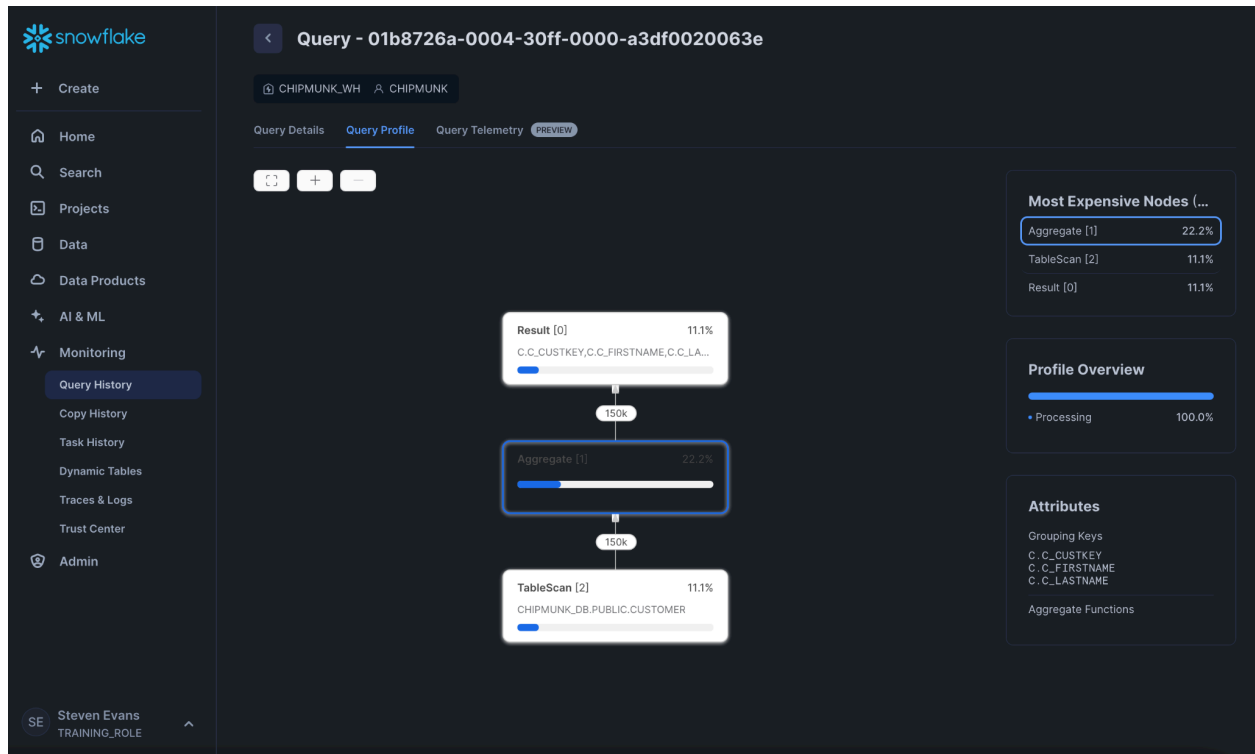


Important Noticings:

22.2 % of the execution timer was done via processing the data, 22.2% was completed via Remote Disk I/O, and 55.6% was executed throughout the initialization. 1 partition was scanned and 1.15 Bytes were written to the result.

- It is important to note that the Dick I/O is fairly low here indicating that this is the reason why the query did not take as long to run, if a more complex query were run and the time to complete were high, having a high Disk I/O time would be indicative that we need to do more filtering in the query to reduce the runtime.

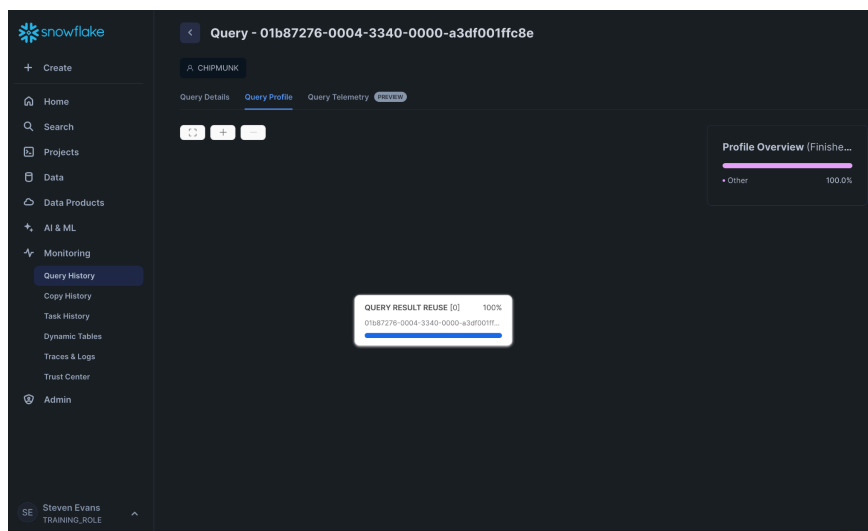
3.1 LOOKING AT THE DIFFERENT TABS IN THE QUERY PROFILE TO DISCOVER HOW THE QUERY WAS PROCESSED.



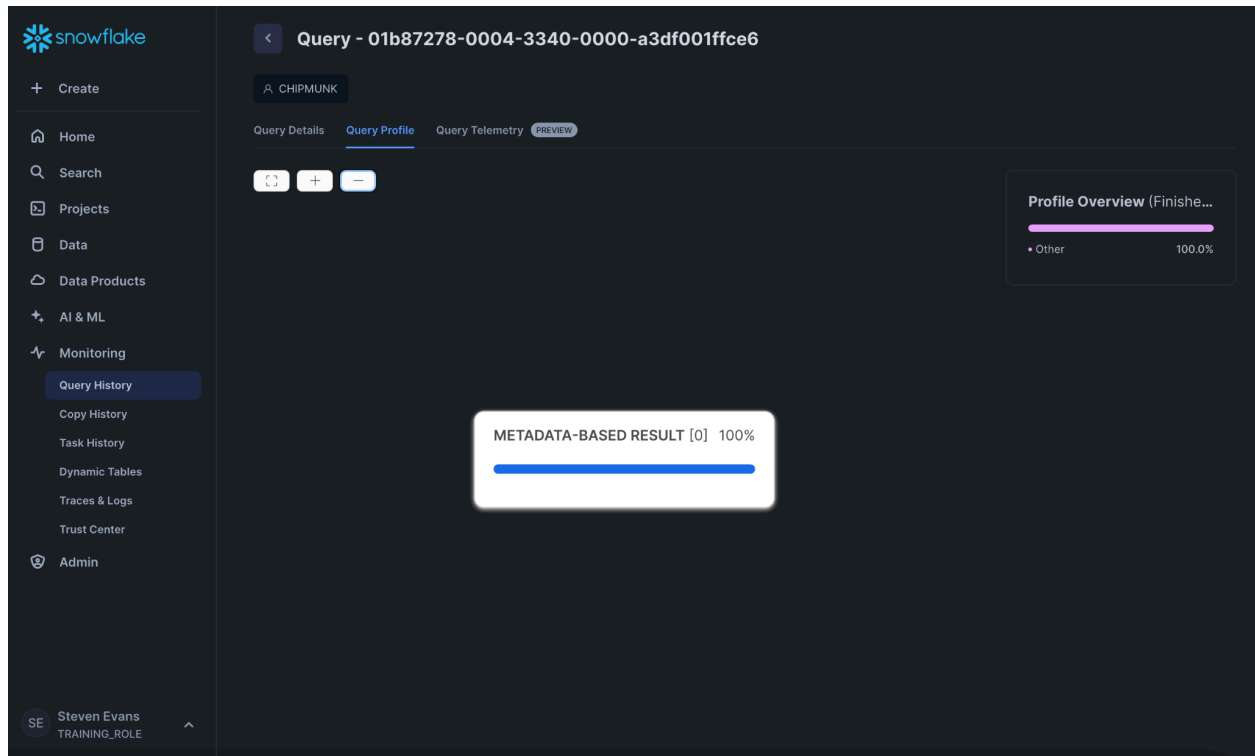
Important Noticings:

- In the Aggregate tab 100% was done via processing, indicating that the 22.2% of the overall execution time is due to this single step
- In the Result and TableScan tabs 100% is due to Disk I/O indicating that the Disk I/O usage is due to producing results for the query and scanning the necessary table to find the correct information.

After running the query again I can see that the metadat for the query was cached and reused. The faster execution time makes sense here



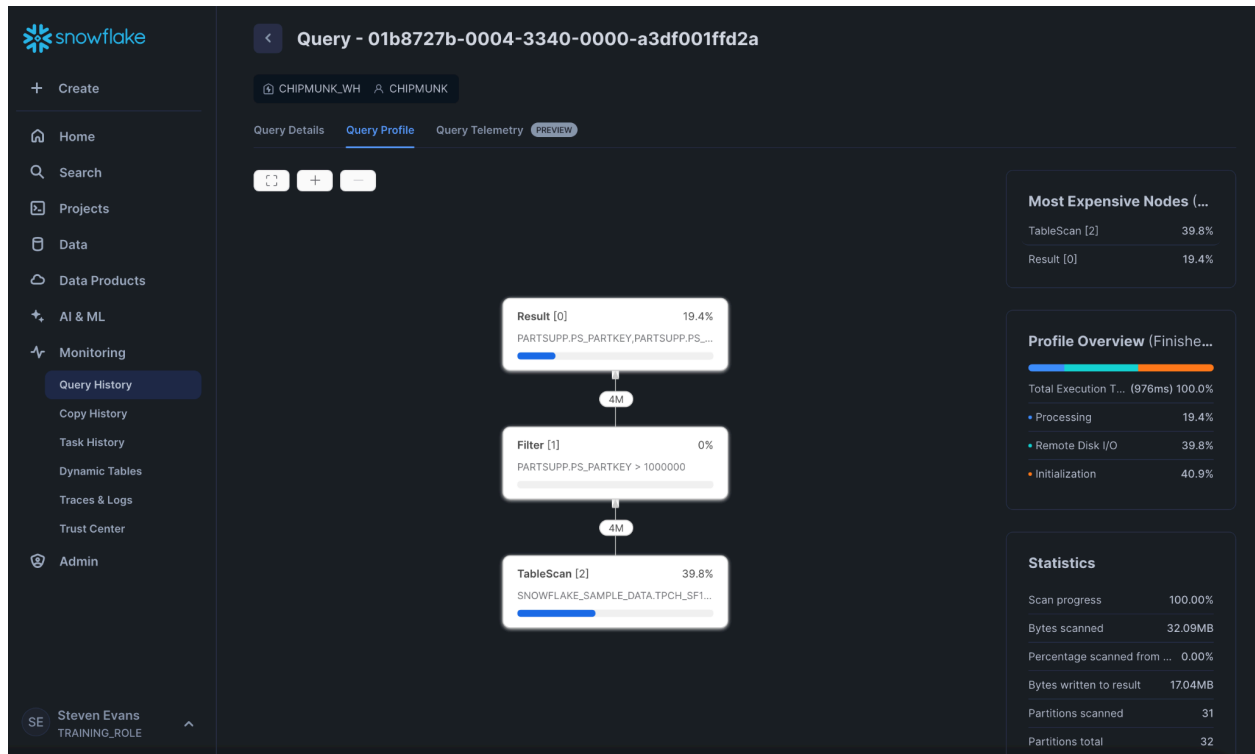
3.2 QUERY PROFILE FOR MIN AND MAX USAGE IN QUERIES



Important Noticings:

When running a query of this nature Snowflake executes the command based on the metadata stored in the cache. This shows that this information is stored within the database and utilizes no disk space.

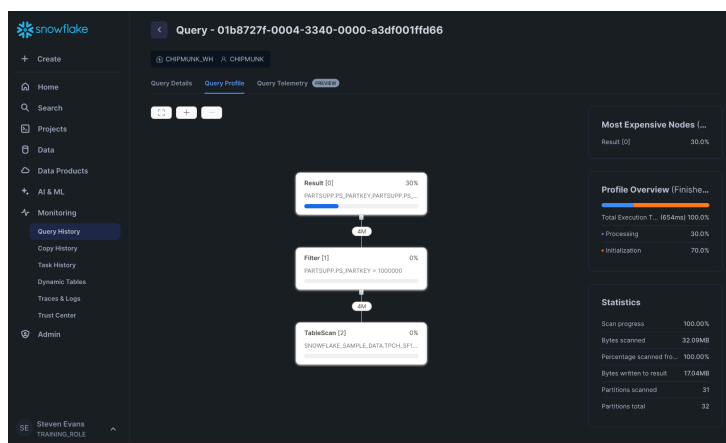
3.3 Running a new query using a where specific identifier



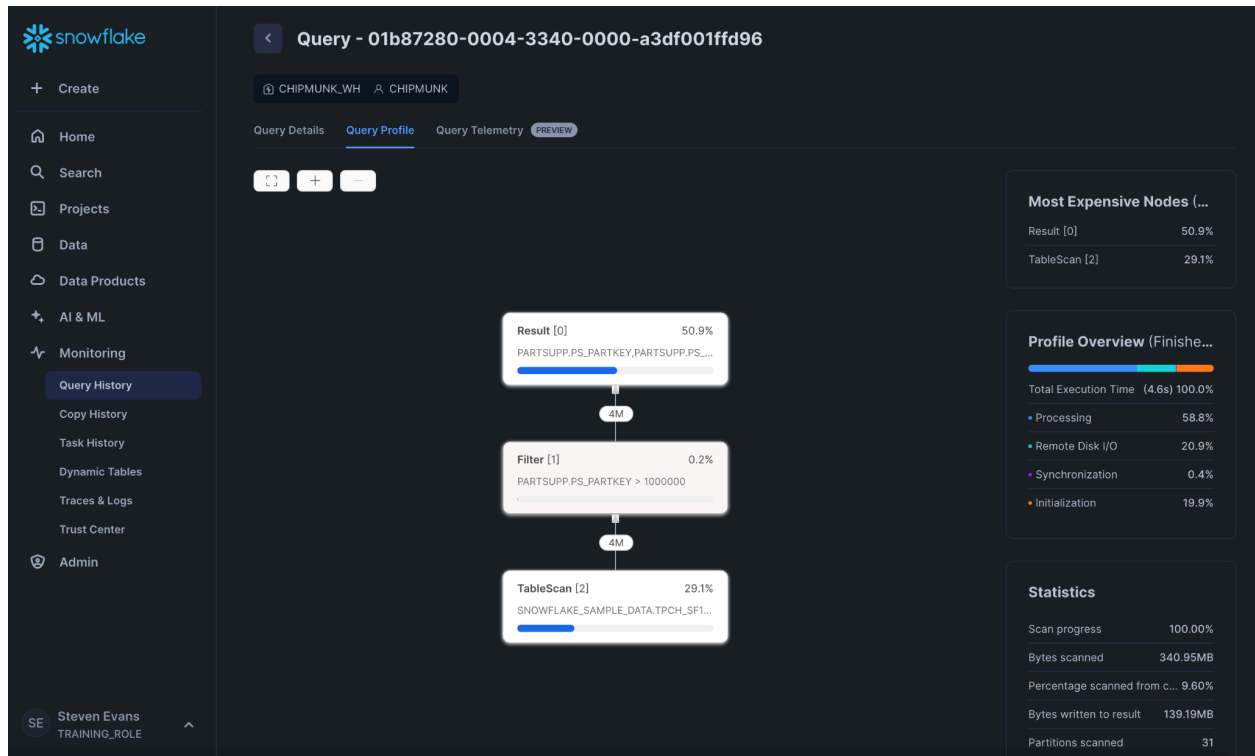
Important Noticings:

Here we can see that because this is a new query on a newly resumed virtual warehouse there was no caching used for its execution, There was some processing ran during the Result phase and Disk I/O used to scan the table.

- After running the query again, there was no disk I/O used to scan the table only processing done on the result tab



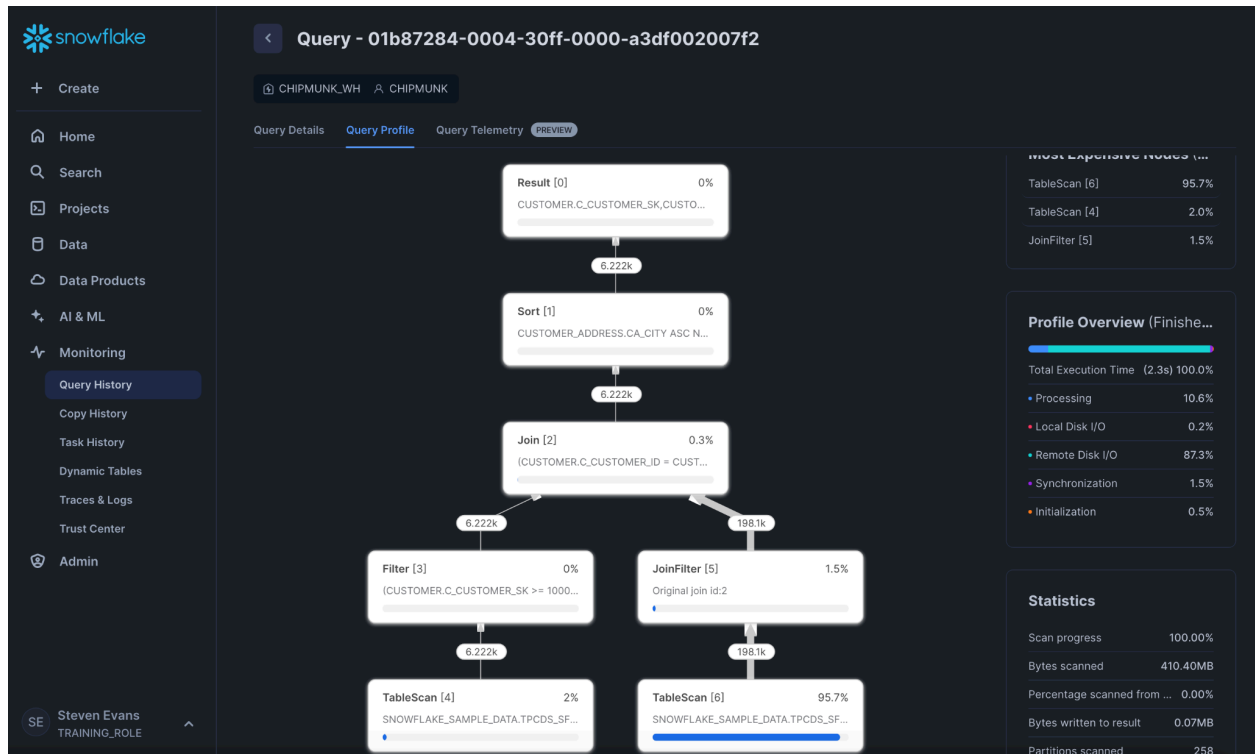
3.3.6 Running the same query but adding some columns



Important Noticings:

Upon adding some columns we can see that some Disk I/O space was used to execute the query. This explains why the query took much longer to run as opposed to the second run of the initial query where we pulled info from the metadata cache.

3.4 Running a query including joins of multiple tables and where clauses

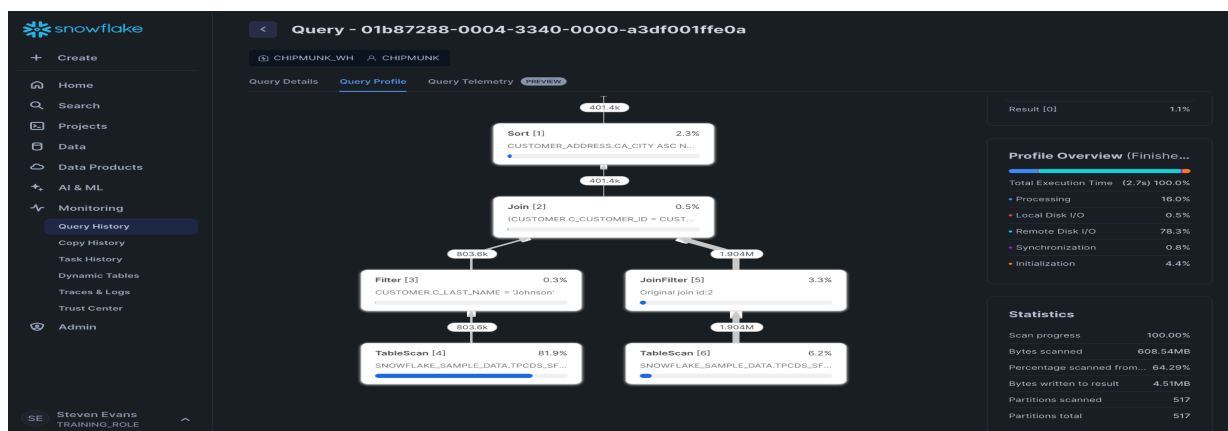


Important Noticings:

Only about half of the total partitions were scanned indicating that this query did not need to scan the entire database to find the necessary information to run the query.

If we look below at the second query without partition pruning using where clauses we can see that every partition was scanned to run the query and more remote disk I/O space was used in the process.

- This demonstrates the importance of partition pruning and understanding how the table's data is organized so that when running queries we can optimize and reduce runtime and remote disk usage. This in turn will lower the overall cost of compute for an organization.



3.5.3 Running a query that generates spillage



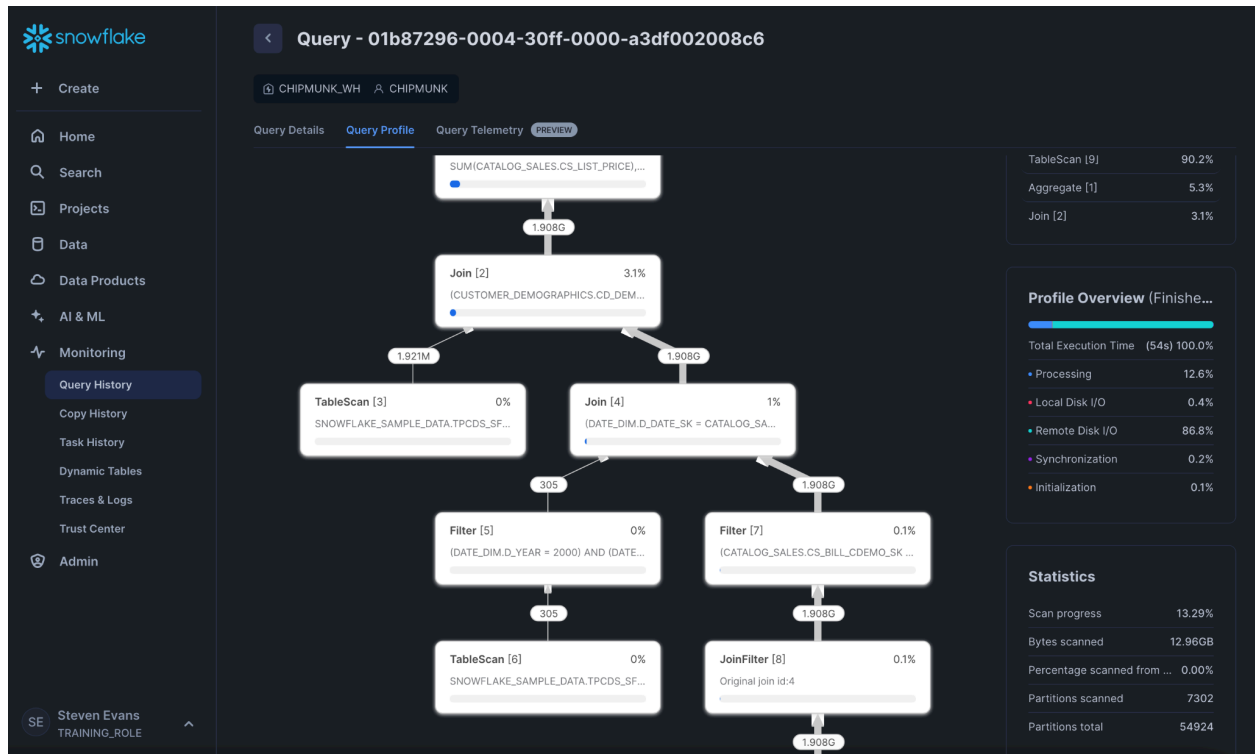
Important Noticings:

Because of the number of partitions scanned during this query we can see that 3.75 GB of bytes were spilled into local storage. This explains why the query took much longer to run than prior queries. The primary cause of this spillage within the queries are the aggregate nodes where multiple sum and counts aggregate functions were run for execution.

Throughout the lab I will be focusing on different methods used to remedy this.

- This query doesn't really need the outer query, we can remove it and remove the culprit of the spillage which is the `cs_order_number` column

3.5.6 Remedying the spillage by removing the outer query and the `cs_order_number` column



Important Noticings:

I can see how the removal of these two pieces of the query resulted in there being no spillage and still getting the same output from the query. The query also took around a minute less time to execute than the first. It is important also to note that there is only one aggregate node present in this query profile.

3.6 Running EXAPLIN on the query profiles to look at the different query nodes and understand their significance.

The screenshot displays the Snowflake SQL Editor interface. The top navigation bar includes tabs for various tutorials. The left sidebar shows a database hierarchy with 'CUSTOMER' selected. The main editor area contains a SQL query with an 'EXPLAIN' statement. Below the query, the 'Results' tab is active, showing a table with 10 rows representing the execution plan. The right sidebar shows the schema for the 'CUSTOMER' table.

SQL Query:

```

553 / inner_query
554 GROUP BY
555     cd_gender;
556
557
558 -- 3.6.2 Click on the operation column header and select the up arrow to sort
559 -- the rows alphabetically by operation type.
560 -- Note that there are 12 rows that correspond to the execution nodes
561 -- that you would see in the Query Profile. Also, note that two of the
562 -- rows are aggregate rows. The node below executes the averaging of the
563 -- list price, sales price, and quantity:
564
565 -- 3.6.3 Run the EXPLAIN statement for the second query.
566
567 EXPLAIN
  
```

Execution Plan Results:

step	id	parentOperators	operation	objects	alias	expressions	Column
1	1	[0]	Aggregate	null	null	aggExprs: [SUM((SUM(CATALOG_SALE	operation
2	1	[1]	Aggregate	null	null	aggExprs: [SUM(CATALOG_SALES.CS_1	100% filled
3	1	[5]	Filter	null	null	(DATE_DIM.D_YEAR = 2000) AND (DATI	7 distinct values
4	1	[5]	Filter	null	null	(CATALOG_SALES.CS_BILL_CDEMO_SK	TableScan 3
5	null	null	GlobalStats	null	null	null	Aggregate 2
6	1	[2]	InnerJoin	null	null	joinKey: (CUSTOMER_DEMOGRAPHICS	InnerJoin 2
7	1	[3]	InnerJoin	null	null	joinKey: (DATE_DIM.D_DATE_SK = CATA	Filter 2
8	1	[8]	JoinFilter	null	null	joinKey: (CUSTOMER_DEMOGRAPHICS	GlobalStats
9	1	0	Result	null	null	CUSTOMER_DEMOGRAPHICS.CD_GENI	Result 1
10	1	[3]	TableScan	SNOWFL	null	CD_DEMO_SK, CD_GENDER	

CUSTOMER Table Schema:

Column	Data Type
C_CUSTKEY	NUMBER(38,0)
C_FIRSTNAME	VARCHAR(25)
C_LASTNAME	VARCHAR(25)

Important Noticings- First Query: Spillage

- Based on the information presented in the output of the explain query I can see that there are two rows with aggregate expressions. This aligns with what I saw earlier in the query profile showing an unnecessary aggregate node from getting the average a second time.

The screenshot displays the Snowflake SQL IDE interface. The top bar shows the current database as `SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL`. The left sidebar lists the database schema structure, including `CHIPMUNK_DB` and `CUSTOMER`. The main editor shows the following SQL query:

```

SELECT
  cd_gender
, AVG(cs_list_price) lp
, AVG(cs_sales_price) sp
, AVG(cs_quantity) qu
FROM
  catalog_sales
, date_dim
, customer_demographics
WHERE
  cs_sold_date_sk = d_date_sk
AND
  cs_bill_cdemo_sk = cd_demo_sk
AND
  d_year IN (2000)

```

Below the query, the 'Results' tab is active, displaying the execution plan. The plan consists of 10 steps:

step	id	parentOperators	operator	objects	alias	expressions	Column
1	1	[0]	Aggregate	null	null	aggExprs: [SUM(CATALOG_SALES.CS_L...	operation
2	1	[4]	Filter	null	null	(DATE_DIM.D_YEAR = 2000) AND (DATI	100% filled
3	1	[4]	Filter	null	null	(CATALOG_SALES.CS_BILL_CDEMO_SK	7 distinct values
4	null	null	GlobalStats	null	null	null	TableScan 3
5	1	[1]	InnerJoin	null	null	joinKey: (CUSTOMER_DEMOGRAPHICS	InnerJoin 2
6	1	[2]	InnerJoin	null	null	joinKey: (DATE_DIM.D_DATE_SK = CATA	Filter 2
7	1	[7]	JoinFilter	null	null	joinKey: (CUSTOMER_DEMOGRAPHICS	GlobalStats 1
8	1	0	Result	null	null	CUSTOMER_DEMOGRAPHICS.CD_GENI	Result
9	1	[2]	TableScan	SNOWFL.	null	CD_DEMO_SK, CD_GENDER	Ask Copilot
10	1	[5]	TableScan	SNOWFL.	null	D_DATE_SK, D_YEAR, D_MOY	Aggregate 1

Important Noticings - Second Query: No Spillage

- Here I can see that the aggregate node running the average a second time is no longer present. This shows the importance and functionality of the explain command to show the different nodes of the query and how it can be leveraged to optimize queries by removing bottlenecks in the query writing process.