

Analyzing Trip Data in Snowflake

Analyzing Trip Data in Snowflake

Shweta Ajay Shinde

Masters in Data Analytics, San Jose State University

Data 226: Data warehousing

Instructor: Keeyong Han

9th Sept 2024

Analyzing Trip Data in Snowflake

1. Creating objects like Database, Schema, and Table in Snowflake.

We are using a **COMPUTE_WH (X-Small)** warehouse for the following steps.

- SQL query to create database:- **CREATE DATABASE trip_db;**

The screenshot shows the Snowflake web interface. The query editor contains the SQL command: `1 Create database trip_db;`. The results pane displays a single row with the status: "Database TRIP_DB successfully created." The query details on the right indicate a duration of 223ms and 1 row returned.

status
1 Database TRIP_DB successfully created.

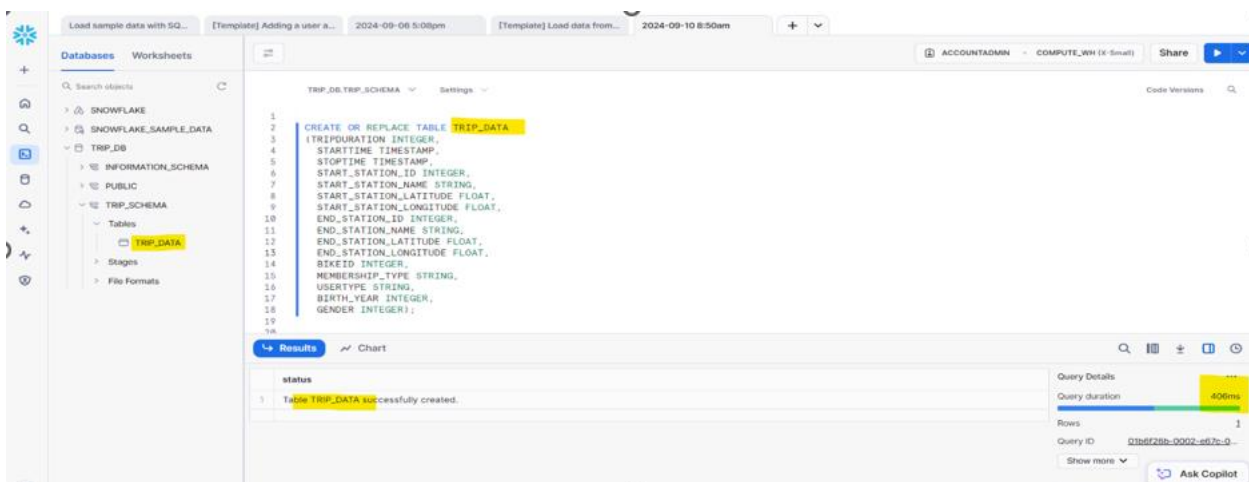
- SQL query to create schema:- **CREATE SCHEMA trip_schema;**

The screenshot shows the Snowflake web interface with the query editor containing the SQL commands: `1 Create database trip_db;`, `2 CREATE SCHEMA trip_schema;`, and `3 SHOW SCHEMAS;`. The results pane displays a table with schema information. The query details on the right indicate a duration of 91ms and 3 rows returned.

	created_on	name	is_default	is_current	database_name	owner	comment
1	2024-09-10 08:58:34.603 -0700	INFORMATION_SCHEMA	N	N	TRIP_DB		Views describing the contents of sc
2	2024-09-10 08:52:10.274 -0700	PUBLIC	N	N	TRIP_DB	ACCOUNTADMIN	
3	2024-09-10 08:58:33.163 -0700	TRIP_SCHEMA	N	Y	TRIP_DB	ACCOUNTADMIN	

Analyzing Trip Data in Snowflake

- Query to create table:-
CREATE OR REPLACE TABLE TRIP_DATA
(TRIPDURATION INTEGER,
STARTTIME TIMESTAMP,
STOPTIME TIMESTAMP,
START_STATION_ID INTEGER,
START_STATION_NAME STRING,
START_STATION_LATITUDE FLOAT,
START_STATION_LONGITUDE FLOAT,
END_STATION_ID INTEGER,
END_STATION_NAME STRING,
END_STATION_LATITUDE FLOAT,
END_STATION_LONGITUDE FLOAT,
BIKEID INTEGER,
MEMBERSHIP_TYPE STRING,
USERTYPE STRING,
BIRTH_YEAR INTEGER,
GENDER INTEGER);



2. Create Stage 'trip_stage' in Snowflake

Next, create a stage to hold the files. Here's how you can create an external stage:

- SQL query to create stag
create stage trip_stage url='s3://snowflake-workshop-lab/citibike-trips-csv/';

Analyzing Trip Data in Snowflake

The screenshot shows the Snowflake web interface. The left sidebar displays the database structure: SNOWFLAKE, SNOWFLAKE_SAMPLE_DATA, TRIP_DB, INFORMATION_SCHEMA, PUBLIC, TRIP_SCHEMA, and Stages. The main panel shows a query editor with the following SQL code:

```
1
2
3 create stage trip_stage url='s3://snowflake-workshop-lab/citibike-trips-csv/';
4
5
6
```

The query results show a single row with the status: "Stage area 'TRIP_STAGE' successfully created." The right sidebar displays query details: Query duration 389ms, Rows 1, Query ID 01b6f1f0-0002-e6f1-00...

- Query to list the stage
- list @trip_stage;**

The screenshot shows the Snowflake web interface displaying the results of the query `list @trip_stage;`. The results table lists 11 files with columns: name, size, md5, and last_modified.

	name	size	md5	last_modified
1	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_0_0.csv.gz	3072073	cfc69e04228a94d1337ab383a3af7472	Sun, 10 Jul 2022 18:19:41 GMT
2	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_1_0.csv.gz	2877852	92a1c064a3c632f338b57d5c6531e97d	Sun, 10 Jul 2022 18:19:41 GMT
3	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_2_0.csv.gz	3174598	39faac098802c1b29f2d4d99f31378be	Sun, 10 Jul 2022 18:19:41 GMT
4	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_3_0.csv.gz	3031012	cd0dca1dcfa309c0bb4bd40d1265c3a9	Sun, 10 Jul 2022 18:19:41 GMT
5	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_4_0.csv.gz	3005838	fb24c0cc5fb6ee54d2aa4d50265792c7	Sun, 10 Jul 2022 18:19:41 GMT
6	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_5_0.csv.gz	3099881	441efe806352c57a50c4f31afccb2e3	Sun, 10 Jul 2022 18:19:41 GMT
7	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_6_0.csv.gz	3060952	372765a1ca713eeb1d56f79e0956e48f	Sun, 10 Jul 2022 18:19:41 GMT
8	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_7_0.csv.gz	3302808	0298668f685d5e6f48df9fe79836f1c9	Sun, 10 Jul 2022 18:19:41 GMT
9	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_1_0_0.csv.gz	3494317	cc73ecc8f0408f089d5e63bc5515192d	Sun, 10 Jul 2022 18:19:41 GMT
10	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_1_1_0.csv.gz	3325846	2951cea70970fe9bf475d34b539de8cf	Sun, 10 Jul 2022 18:19:41 GMT
11	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_1_2_0.csv.gz	3461223	dce5effbeb1ed7db67fbd81986b38cec	Sun, 10 Jul 2022 18:19:41 GMT

The right sidebar displays query details: Query duration 657ms, Rows 376, Query ID 01b6f1f1-0002-e688-00... The 'name' column is 100% filled, and the 'size' column is 2293464.

3. Create File Format

Create a file format to read data from CSV file.

- Query to create file format
- ```
CREATE FILE FORMAT TRIP_FILE_FORMAT TYPE = 'CSV'
FIELD_OPTIONALLY_ENCLOSED_BY = ''
SKIP_HEADER = 1
FIELD_DELIMITER = ',';
```

## Analyzing Trip Data in Snowflake

The screenshot shows the Snowflake web interface. The left sidebar displays the database structure: SNOWFLAKE, SNOWFLAKE\_SAMPLE\_DATA, TRIP\_DB, INFORMATION\_SCHEMA, PUBLIC, TRIP\_SCHEMA, Stages (TRIP\_STAGE), and File Formats (TRIP\_FILE\_FORMAT). The main query editor shows the following SQL code:

```
1 CREATE FILE FORMAT trip_file_format
2 TYPE = 'CSV'
3 FIELD_OPTIONALLY_ENCLOSED_BY = ''
4 SKIP_HEADER = 1
5 FIELD_DELIMITER = ',';
```

The 'Results' tab shows a single row: "File format 'TRIP\_FILE\_FORMAT' successfully created." The 'Query Details' panel on the right indicates a query duration of 62ms, 1 row, and a query ID of 01b6f21e-0002-e58d-0... The status bar at the bottom shows "status 100% filled".

- Query to view the data in file using file format.

```
SELECT $1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14,$15,$16
FROM @trip_stage/trips_2013_0_1_0.csv.gz
(FILE_FORMAT => 'trip_file_format');
```

The screenshot shows the Snowflake web interface with the following SQL query in the editor:

```
1 SELECT $1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14,$15,$16
2 FROM @trip_stage/trips_2013_0_1_0.csv.gz
3 (FILE_FORMAT => 'trip_file_format');
```

The 'Results' tab displays a table with 16 columns labeled \$1 through \$16. The table contains 13 rows of trip data. The first column (\$1) represents a unique trip ID, and the last column (\$16) represents a count.

|    | \$1  | \$2                 | \$3                 | \$4  | \$5             | \$6        | \$7        | \$8 | \$9           | \$10      | \$11       | \$12  | \$13 | \$14       | \$15 | \$16 |
|----|------|---------------------|---------------------|------|-----------------|------------|------------|-----|---------------|-----------|------------|-------|------|------------|------|------|
| 1  | 1469 | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 417  | Barclay St & C  | 40.7129122 | -74.010202 | 232 | Cadman Plaz   | 40.695976 | -73.990148 | 15785 |      | Customer   |      | 0    |
| 2  | 1855 | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 306  | Cliff St & Fult | 40.7082350 | -74.005300 | 309 | Murray St & V | 40.714978 | -74.013012 | 15168 |      | Subscriber | 1971 | 1    |
| 3  | 1891 | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 306  | Cliff St & Fult | 40.7082350 | -74.005300 | 309 | Murray St & V | 40.714978 | -74.013012 | 19104 |      | Subscriber | 1977 | 2    |
| 4  | 1021 | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 2000 | Front St & We   | 40.7025506 | -73.989402 | 395 | Bond St & Sc  | 40.688070 | -73.984106 | 17944 |      | Customer   |      | 0    |
| 5  | 1995 | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 216  | Columbia Hei    | 40.7003786 | -73.995480 | 224 | Spruce St & M | 40.711463 | -74.005524 | 16829 |      | Customer   |      | 0    |
| 6  | 1310 | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 442  | W 27 St & 7     | 40.746647  | -73.993915 | 422 | W 59 St & 10  | 40.770513 | -73.988036 | 15641 |      | Subscriber | 1965 | 2    |
| 7  | 1479 | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 511  | E 14 St & Ave   | 40.7293866 | -73.977724 | 491 | E 24 St & Par | 40.740963 | -73.986022 | 17675 |      | Customer   |      | 0    |
| 8  | 84   | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 525  | W 34 St & 11    | 40.7559415 | -74.002116 | 525 | W 34 St & 11  | 40.755941 | -74.002116 | 18079 |      | Customer   |      | 0    |
| 9  | 2038 | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 389  | Broadway & E    | 40.7104455 | -73.965250 | 389 | Broadway & E  | 40.710445 | -73.965250 | 19075 |      | Customer   |      | 0    |
| 10 | 896  | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 290  | 2 Ave & E 58    | 40.7602025 | -73.964784 | 297 | E 15 St & 3 A | 40.734232 | -73.986923 | 20503 |      | Customer   |      | 0    |
| 11 | 361  | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 348  | W Broadway      | 40.7249096 | -74.001547 | 328 | Watts St & Gr | 40.724055 | -74.009656 | 18429 |      | Customer   |      | 0    |
| 12 | 1728 | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 487  | E 20 St & FDF   | 40.7331425 | -73.975738 |     |               |           |            | 15432 |      | Subscriber | 1981 | 1    |
| 13 | 787  | 2013-06-09 15:00:00 | 2013-06-09 15:00:00 | 329  | Greenwich St    | 40.7201522 | -74.010365 | 249 | Harrison St & | 40.718709 | -74.009000 | 19567 |      | Subscriber | 1953 | 1    |

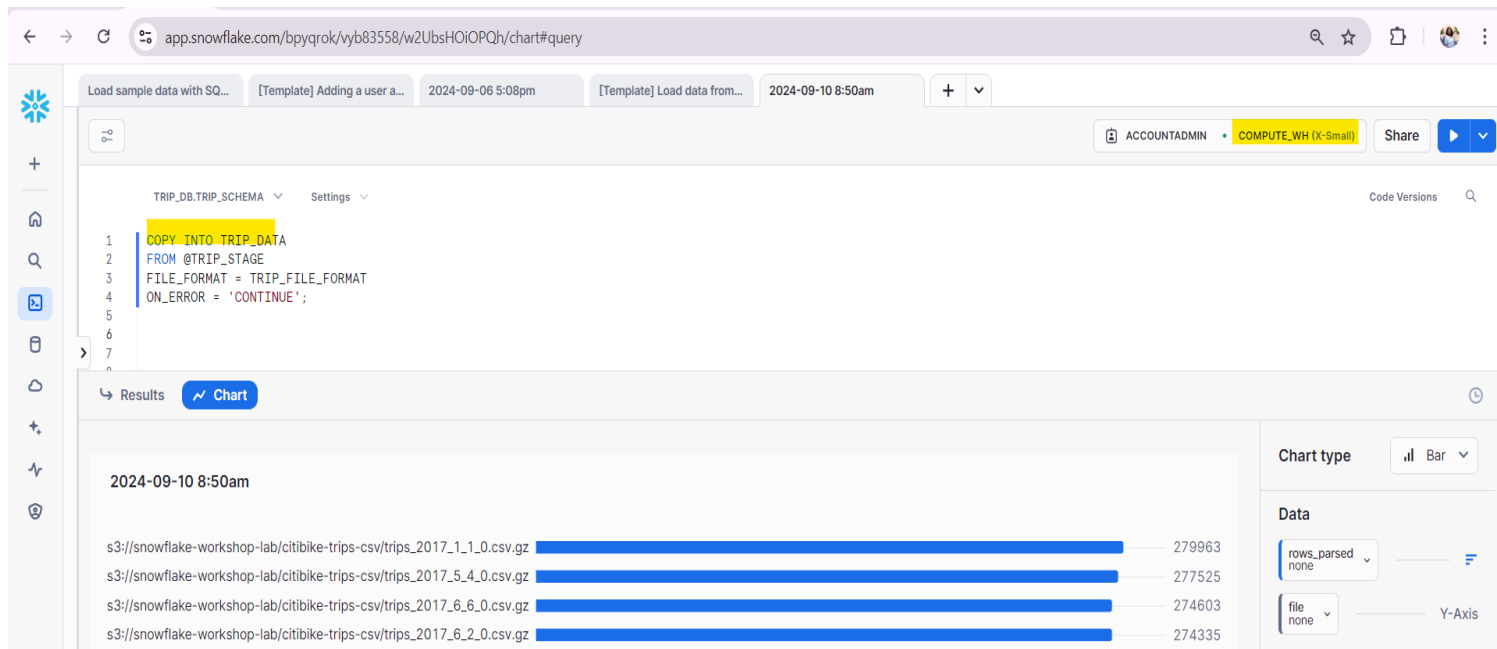
## Analyzing Trip Data in Snowflake

### 4. Load Data into the Table

After data file is uploaded to the stage, you can now load it into the trip\_data table:

- Query to copy data into table

```
COPY INTO TRIP_DATA
FROM @TRIP_STAGE
FILE_FORMAT = TRIP_FILE_FORMAT
ON_ERROR = 'CONTINUE';
```



**select \* from trip\_data;**

The screenshot shows the Snowflake web interface displaying the results of the query 'select \* from trip\_data;'. The results are shown in a table format with the following columns:

|    | TRIPDURATION | STARTTIME               | STOPTIME                | START_STATION_ID | START_STATION_NAME            | START_STATION_LATITUDE | START_STATION_LONGITUDE | END_STATION_ID | END_STATION_NAME        | END_STATION_LATITUDE |
|----|--------------|-------------------------|-------------------------|------------------|-------------------------------|------------------------|-------------------------|----------------|-------------------------|----------------------|
| 1  | 1855         | 2013-06-09 15:07:06.000 | 2013-06-09 15:38:01.000 | 306              | Cliff St & Fulton St          | 40.70823502            | -74.00530063            | 309            | Murray St & West St     | 40.70823502          |
| 2  | 1891         | 2013-06-09 15:07:06.000 | 2013-06-09 15:38:37.000 | 306              | Cliff St & Fulton St          | 40.70823502            | -74.00530063            | 309            | Murray St & West St     | 40.70823502          |
| 3  | 1310         | 2013-06-09 15:07:09.000 | 2013-06-09 15:28:59.000 | 442              | W 27 St & 7 Ave               | 40.746647              | -73.993915              | 422            | W 59 St & 10 Ave        | 40.746647            |
| 4  | 787          | 2013-06-09 15:07:18.000 | 2013-06-09 15:20:25.000 | 329              | Greenwich St & North Moore St | 40.72015227            | -74.010365009           | 249            | Harrison St & Hudson St | 40.72015227          |
| 5  | 261          | 2013-06-09 15:07:21.000 | 2013-06-09 15:11:42.000 | 257              | Lispenard St & Broadway       | 40.71939226            | -74.00247214            | 146            | Hudson St & Reade St    | 40.71939226          |
| 6  | 644          | 2013-06-09 15:07:24.000 | 2013-06-09 15:18:08.000 | 293              | Lafayette St & E 8 St         | 40.730206605           | -73.991026282           | 2004           | 6 Ave & Broome St       | 40.730206605         |
| 7  | 1427         | 2013-06-09 15:07:52.000 | 2013-06-09 15:31:39.000 | 301              | E 2 St & Avenue B             | 40.72217444            | -73.98368779            | 439            | E 4 St & 2 Ave          | 40.72217444          |
| 8  | 2756         | 2013-06-09 15:07:54.000 | 2013-06-09 15:53:50.000 | 435              | W 21 St & 6 Ave               | 40.74173969            | -73.99415556            | 146            | Hudson St & Reade St    | 40.74173969          |
| 9  | 486          | 2013-06-09 15:07:56.000 | 2013-06-09 15:16:02.000 | 303              | Mercer St & Spring St         | 40.72362738            | -73.99949601            | 355            | Bayard St               | 40.72362738          |
| 10 | 508          | 2013-06-09 15:07:59.000 | 2013-06-09 15:18:25.000 | 268              | Howard St & Centre St         | 40.71910537            | -73.99973337            | 311            | Norfolk St              | 40.71910537          |

The bottom right corner of the interface features an 'Ask Copilot' button.

## Analyzing Trip Data in Snowflake

5. Write an SQL query to produce a report that shows, for each hour of the day, the following:

1. The total number of trips that started during that hour.
2. The average duration of these trips in mins.
3. The average distance traveled during these trips in kms.

- SQL query

**SELECT**

**DATE\_PART(HOUR, STARTTIME) AS HOUR,**

**COUNT(\*) AS TOTAL\_NUM\_OF\_TRIP,**

**ROUND(AVG(TRIPDURATION / 60),4) AS AVERAGE\_DURATION,**

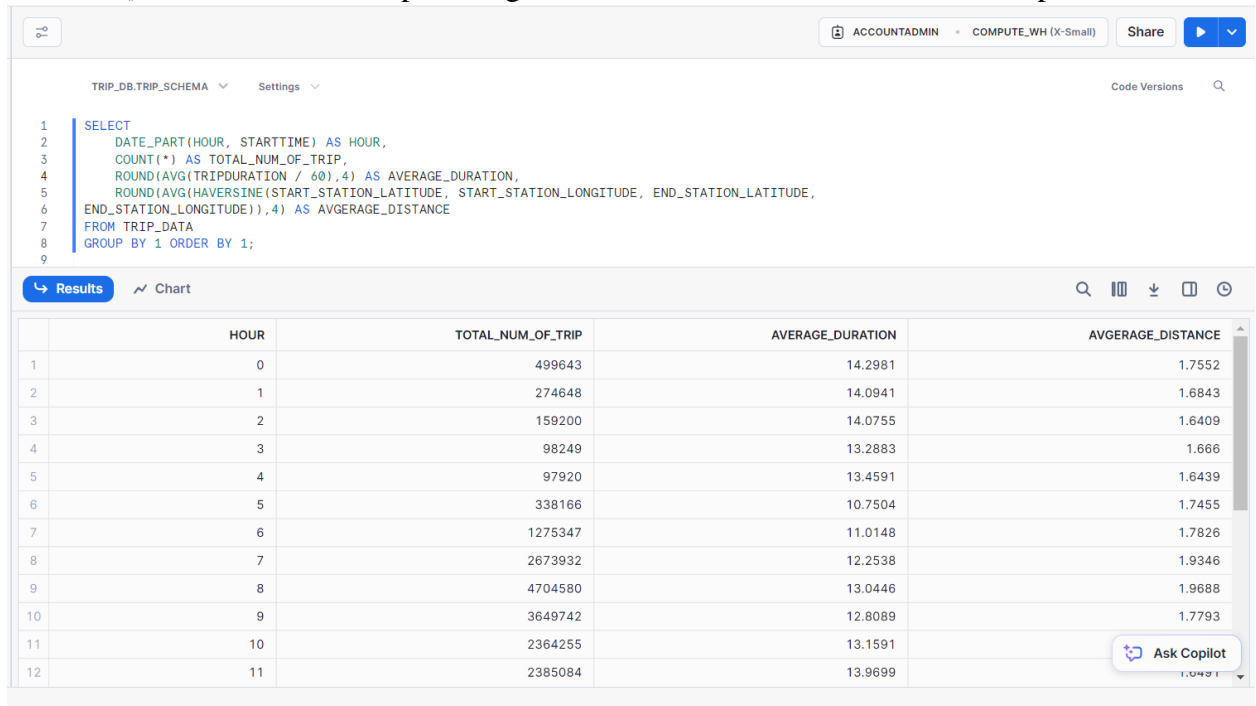
**ROUND(AVG(HAVERSINE(START\_STATION\_LATITUDE,  
START\_STATION\_LONGITUDE, END\_STATION\_LATITUDE,  
END\_STATION\_LONGITUDE)),4) AS AVGERAGE\_DISTANCE**

**FROM TRIP\_DATA**

**GROUP BY 1 ORDER BY 1;**

Explanation:

- DATE\_PART(HOUR, STARTTIME) extracts the hour from the STARTTIME timestamp.
- COUNT(\*) gives the total number of trips for each hour.
- AVG(TRIPDURATION / 60.0) calculates the average trip duration in minutes (assuming TRIPDURATION is stored in seconds).
- HAVERSINE function calculates the distance between the start and end coordinates of each trip using the Haversine formula.
- ROUND() round the number up to the given number, in this case it rounds till 4 places.



The screenshot shows a Snowflake SQL query editor with the following query:

```

1 SELECT
2 DATE_PART(HOUR, STARTTIME) AS HOUR,
3 COUNT(*) AS TOTAL_NUM_OF_TRIP,
4 ROUND(AVG(TRIPDURATION / 60),4) AS AVERAGE_DURATION,
5 ROUND(AVG(HAVERSINE(START_STATION_LATITUDE, START_STATION_LONGITUDE, END_STATION_LATITUDE,
6 END_STATION_LONGITUDE)),4) AS AVGERAGE_DISTANCE
7 FROM TRIP_DATA
8 GROUP BY 1 ORDER BY 1;
9

```

The results are displayed in a table with the following columns: HOUR, TOTAL\_NUM\_OF\_TRIP, AVERAGE\_DURATION, and AVGERAGE\_DISTANCE. The table shows data for each hour of the day from 0 to 11.

|    | HOUR | TOTAL_NUM_OF_TRIP | AVERAGE_DURATION | AVGERAGE_DISTANCE |
|----|------|-------------------|------------------|-------------------|
| 1  | 0    | 499643            | 14.2981          | 1.7552            |
| 2  | 1    | 274648            | 14.0941          | 1.6843            |
| 3  | 2    | 159200            | 14.0755          | 1.6409            |
| 4  | 3    | 98249             | 13.2883          | 1.666             |
| 5  | 4    | 97920             | 13.4591          | 1.6439            |
| 6  | 5    | 338166            | 10.7504          | 1.7455            |
| 7  | 6    | 1275347           | 11.0148          | 1.7826            |
| 8  | 7    | 2673932           | 12.2538          | 1.9346            |
| 9  | 8    | 4704580           | 13.0446          | 1.9688            |
| 10 | 9    | 3649742           | 12.8089          | 1.7793            |
| 11 | 10   | 2364255           | 13.1591          |                   |
| 12 | 11   | 2385084           | 13.9699          |                   |

## Analyzing Trip Data in Snowflake

### Data Result

| HOURL | TOTAL_NUM_OF_TRIP | AVERAGE_DURATION | AVERAGE_DISTANCE |
|-------|-------------------|------------------|------------------|
| 0     | 499643            | 14.2981          | 1.7552           |
| 1     | 274648            | 14.0941          | 1.6843           |
| 2     | 159200            | 14.0755          | 1.6409           |
| 3     | 98249             | 13.2883          | 1.666            |
| 4     | 97920             | 13.4591          | 1.6439           |
| 5     | 338166            | 10.7504          | 1.7455           |
| 6     | 1275347           | 11.0148          | 1.7826           |
| 7     | 2673932           | 12.2538          | 1.9346           |
| 8     | 4704580           | 13.0446          | 1.9688           |
| 9     | 3649742           | 12.8089          | 1.7793           |
| 10    | 2364255           | 13.1591          | 1.6973           |
| 11    | 2385084           | 13.9699          | 1.6491           |
| 12    | 2771192           | 13.974           | 1.6038           |
| 13    | 2899224           | 14.0809          | 1.5985           |
| 14    | 2969842           | 14.7             | 1.6678           |
| 15    | 3130067           | 14.9695          | 1.7113           |
| 16    | 3767896           | 14.8513          | 1.7962           |
| 17    | 5457805           | 14.7013          | 1.9442           |
| 18    | 5295007           | 14.7997          | 1.9237           |
| 19    | 3675781           | 14.5261          | 1.7884           |
| 20    | 2461409           | 14.5986          | 1.737            |
| 21    | 1716035           | 14.4014          | 1.7365           |
| 22    | 1283920           | 14.5091          | 1.7496           |
| 23    | 854615            | 14.4283          | 1.7302           |

### 6. Create X-LARGE Warehouse

Now, create a new warehouse **TRIP\_XLARGE\_WH** of size **X-LARGE**.

- SQL query to create X-LARGE WH  
**CREATE WAREHOUSE TRIP\_XLARGE\_WH**  
**WAREHOUSE\_SIZE = 'XLARGE'**  
**AUTO\_SUSPEND = 300**  
**AUTO\_RESUME = TRUE;**

Or



## Analyzing Trip Data in Snowflake

We can manually create the warehouse by following the below steps.

The screenshot shows the Snowflake 'Warehouses' page. A modal for creating a new warehouse is open. The modal title is 'New Warehouse'. It shows 'Creating as ACCOUNTADMIN'. The name field is 'TRIP\_XLARGE\_WH'. There is an 'Add Comment' button. The 'Type' is 'Standard' and the 'Size' is 'X-Large'. The 'Advanced Options' section shows 'The current advanced settings are auto-resume: on, auto-suspend: 5mins, multi-cluster: off, query acceleration: off'. At the bottom, there are 'Cancel' and 'Create Warehouse' buttons. A red arrow points to the 'Create Warehouse' button.

After dropping and recreating the data table, we will repeat steps 4 and 5 to analyze the performance of both the X-Small and X-Large warehouses.

|                       |                           |         |              |
|-----------------------|---------------------------|---------|--------------|
| drop table trip_data; | 01b6f2fc-0002-e67c-0000-1 | Success | SHWETASHINDE |
|-----------------------|---------------------------|---------|--------------|

### 4. Create trip\_data using X-Large Warehouse

The screenshot shows the Snowflake SQL editor. The SQL query is: `CREATE OR REPLACE TABLE TRIP_DATA (TRIPDURATION INTEGER, STARTTIME TIMESTAMP, STOPTIME TIMESTAMP, START_STATION_ID INTEGER, START_STATION_NAME STRING, START_STATION_LATITUDE FLOAT, START_STATION_LONGITUDE FLOAT, END_STATION_ID INTEGER, END_STATION_NAME STRING, END_STATION_LATITUDE FLOAT, END_STATION_LONGITUDE FLOAT, BIKEID INTEGER, MEMBERSHIP_TYPE STRING, USERTYPE STRING, BIRTH_YEAR INTEGER, GENDER INTEGER);`. The 'Results' tab shows 'Table TRIP\_DATA successfully created.' The 'Query Details' panel shows 'Query duration: 232ms' and 'Rows: 1'.

## Analyzing Trip Data in Snowflake

We are load data in the table using X-Large Warehouse

The screenshot shows the Snowflake web interface. At the top, the user is logged in as ACCOUNTADMIN and the current warehouse is TRIP\_XLARGE\_WH (X-Large). The query editor shows a SQL query to load data from a stage into a table.

```

1 COPY INTO TRIP_DATA
2 FROM @TRIP_STAGE
3 FILE_FORMAT = TRIP_FILE_FORMAT
4 ON_ERROR = 'CONTINUE';
5

```

The Results tab is active, displaying a table with the following columns: file, status, rows\_parsed, rows\_loaded, error\_limit, errors\_seen, and file\_size. The table contains 8 rows of data, showing the progress of loading various CSV files.

|   | file                                                                   | status           | rows_parsed | rows_loaded | error_limit | errors_seen | file_size |
|---|------------------------------------------------------------------------|------------------|-------------|-------------|-------------|-------------|-----------|
| 1 | s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_4_7_0.csv.gz | PARTIALLY_LOADED | 108414      | 97306       | 108414      | 11108       | N         |
| 2 | s3://snowflake-workshop-lab/citibike-trips-csv/trips_2015_4_6_0.csv.gz | PARTIALLY_LOADED | 165902      | 146123      | 165902      | 19779       | N         |
| 3 | s3://snowflake-workshop-lab/citibike-trips-csv/trips_2018_5_7_0.csv.gz | PARTIALLY_LOADED | 104433      | 99714       | 104433      | 4719        | N         |
| 4 | s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_3_0_0.csv.gz | PARTIALLY_LOADED | 114247      | 98259       | 114247      | 15988       | N         |
| 5 | s3://snowflake-workshop-lab/citibike-trips-csv/trips_2015_6_3_0.csv.gz | PARTIALLY_LOADED | 159338      | 142723      | 159338      | 16615       | N         |
| 6 | s3://snowflake-workshop-lab/citibike-trips-csv/trips_2018_6_4_0.csv.gz | LOADED           | 120897      | 120897      | 120897      | 0           | N         |
| 7 | s3://snowflake-workshop-lab/citibike-trips-csv/trips_2015_5_1_0.csv.gz | PARTIALLY_LOADED | 166889      | 136492      | 166889      | 30397       | N         |
| 8 | s3://snowflake-workshop-lab/citibike-trips-csv/trips_2017_0_3_0.csv.gz | PARTIALLY_LOADED | 263260      | 255208      | 263260      | 8052        | N         |

On the right, the Query Details panel shows a query duration of 11s, 376 rows, and a query ID of 01b6f30c-0002-e66c-0...

5. We will run the SQL query to produce a report that shows, for each hour of the day using X-Large Warehouse

The screenshot shows the Snowflake web interface with a new SQL query entered in the query editor. The query is designed to produce a report showing trip statistics by hour.

```

1 SELECT
2 DATE_PART(HOUR, STARTTIME) AS HOUR,
3 COUNT(*) AS TOTAL_NUM_OF_TRIP,
4 ROUND(AVG(TRIPDURATION / 60), 4) AS AVERAGE_DURATION,
5 ROUND(AVG(HAVERSINE(START_STATION_LATITUDE, START_STATION_LONGITUDE, END_STATION_LATITUDE), 4) AS AVGERAGE_DISTANCE
6 FROM TRIP_DATA
7 GROUP BY 1 ORDER BY 1;
8
9

```

The Results tab is active, displaying a table with the following columns: HOUR, TOTAL\_NUM\_OF\_TRIP, AVERAGE\_DURATION, and AVGERAGE\_DISTANCE. The table contains 12 rows of data, representing the hours of the day from 0 to 11.

|    | HOUR | TOTAL_NUM_OF_TRIP | AVERAGE_DURATION | AVGERAGE_DISTANCE |
|----|------|-------------------|------------------|-------------------|
| 1  | 0    | 499643            | 14.2981          | 1.7552            |
| 2  | 1    | 274648            | 14.0941          | 1.6843            |
| 3  | 2    | 159200            | 14.0755          | 1.6409            |
| 4  | 3    | 98249             | 13.2883          | 1.666             |
| 5  | 4    | 97920             | 13.4591          | 1.6439            |
| 6  | 5    | 338166            | 10.7504          | 1.7455            |
| 7  | 6    | 1275347           | 11.0148          | 1.7826            |
| 8  | 7    | 2673932           | 12.2538          | 1.9346            |
| 9  | 8    | 4704580           | 13.0446          | 1.9688            |
| 10 | 9    | 3649742           | 12.8089          | 1.7793            |
| 11 | 10   | 2364255           | 13.1591          |                   |
| 12 | 11   | 2385084           | 13.9699          |                   |

An "Ask Copilot" button is visible in the bottom right corner of the results area.

## Analyzing Trip Data in Snowflake

Analyze the performance of the node upon changing configurations from X-Small to X-Large in Snowflake.

In the analysis, we observed that the X-Small warehouse took longer to execute the query compared to the X-Large warehouse. Specifically:



- The **X-Large Warehouse** (*TRIP\_XLARGE\_WH*) delivers much faster query completion, indicating a higher level of performance and capacity compared to the X-Small Warehouse. The X-Large Warehouse offers **better performance**, but this comes with a **higher cost**. Its advanced capabilities are reflected in the expense of using this option.
- In contrast, the **X-Small Warehouse** (*COMPUTE\_WH*) demonstrates **slower performance**, taking more time to execute similar operations. This slower speed suggests potential limitations in handling large volumes of data or complex queries effectively. The X-Small Warehouse is more **budget-friendly**, but it comes with slower performance. This lower cost may result in reduced efficiency and longer processing times.

Please find below the screenshots comparing both Warehouses in terms of Efficiency and cost.

### Analysis of Efficiency:-



- **CREATE** table TRIP\_DATA

The **X-Large** Warehouse took **232ms** to create the table, while the **X-Small** Warehouse took **406ms** for the same task.

|                                                 |                          |         |              |                |                                                                                             |                       |
|-------------------------------------------------|--------------------------|---------|--------------|----------------|---------------------------------------------------------------------------------------------|-----------------------|
| CREATE OR REPLACE TABLE TRIP_DATA (TRIPDURATION | 01b6f26b-0002-e67c-0000- | Success | SHWETASHINDE | COMPUTE_WH     |  406ms | 9/10/2024, 4:07:12 PM |
| CREATE OR REPLACE TABLE TRIP_DATA (TRIPDURATION | 01b6f2fd-0002-e68d-0000- | Success | SHWETASHINDE | TRIP_XLARGE_WH |  232ms | 9/10/2024, 6:33:55 PM |

- **COPY** query to load data into TRIP\_DATA



The **X-Large** Warehouse took **11 seconds** to load 54,803,559 rows into the table, while the **X-Small** Warehouse took **1 minute and 10 seconds** for the same task.

|   | SQL TEXT                                         | QUERY ID                 | STATUS  | USER         | WAREHOUSE      | DURATION                                                                                     | STARTED               |
|---|--------------------------------------------------|--------------------------|---------|--------------|----------------|----------------------------------------------------------------------------------------------|-----------------------|
| 1 | COPY INTO TRIP_DATA FROM @TRIP_STAGE FILE_FORMAT | 01b6f30c-0002-e66c-0000- | Success | SHWETASHINDE | TRIP_XLARGE_WH |  11s    | 9/10/2024, 6:48:43 PM |
| 2 | COPY INTO TRIP_DATA FROM @TRIP_STAGE FILE_FORMAT | 01b6f297-0002-e67c-0000- | Success | SHWETASHINDE | COMPUTE_WH     |  1m 10s | 9/10/2024, 4:51:27 PM |

## Analyzing Trip Data in Snowflake

- **SELECT** query on TRIP\_DATA

The **X-Large** Warehouse took **1.7 seconds** to select data from table , while the **X-Small** Warehouse took **3.1 seconds** for the same task.

|                                                  |                         |         |              |                |                                                                                          |                       |
|--------------------------------------------------|-------------------------|---------|--------------|----------------|------------------------------------------------------------------------------------------|-----------------------|
| SELECT DATE_PART(HOUR, STARTTIME) AS HOUR, COUNT | 01b6f2b7-0002-e66e-0000 | Success | SHWETASHINDE | COMPUTE_WH     |  3.1s | 9/10/2024, 5:23:04 PM |
| SELECT DATE_PART(HOUR, STARTTIME) AS HOUR, COUNT | 01b6f310-0002-e66d-0000 | Success | SHWETASHINDE | TRIP_XLARGE_WH |  1.7s | 9/10/2024, 6:52:50 PM |

### Analysis of Cost:-

- Cost of both the Warehouse

The **X-Large** Warehouse has an expense of **\$5.25** to execute 3 to 4 queries , while the **X-Small** Warehouse costs **\$1.92** for handling more than 10 queries.

### Top warehouses by cost

[View All >](#)

|                                                                                                  |                                                                                    |      |
|--------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|------|
|  TRIP_XLARGE_WH |  | 5.25 |
|  COMPUTE_WH     |   | 1.92 |

## Conclusion

For much larger datasets, the X-Large Warehouse will likely perform even better. It will take less time to load data compared to the X-Small Warehouse. As the amount of data grows, the time difference between the two warehouses will increase. The X-Large Warehouse will handle large tasks more efficiently, significantly cutting down the time needed for loading and processing data.

While the X-Large Warehouse comes with a higher cost, its performance justifies the expense for handling large-scale operations. In contrast, the X-Small Warehouse is more budget-friendly, making it ideal for smaller datasets or less complex tasks where high performance is less critical. It is cost-effective for development, testing, small-scale operations or small dataset.