Module 13: Snowpipe and Snowflake Next Gen UI

1.1 Steps to Prepare Your Experiment Environment

1.1.1 This experiment only works with AWS cloud. The Dynamic Data Masking feature needs Snowflake ENTERPRISE edition. Hence, we suggest you select the region which is physically closest to you. And select the Enterprise edition so you can leverage some advanced capabilities that are not available in lower Edition.

1.1.2 We'll use https://github.com/GeorgeNiece/snowflake-core-4day/blob/main/experiments/Snowpipe_lab_scripts.sql and download the "Snowpipe_lab_scripts.sql" file to your local machine. This file contains prewritten SQL commands and we will use this file later in the experiment.

Module 13.2:The Next Generation Snowflake User Interface

> **About the screen captures, sample code, and environment**
> Screen captures in this experiment depict examples and results that may slightly vary from what you may see when you complete the experiment.

## 2.1 Logging Into the Snowflake User Interface (UI)

2.1.1 Open a browser window and enter the URL of your Snowflake 30-day trial environment.

2.1.2 You should see the login screen below. Enter your unique credentials to log in.

2.2 Close any Welcome Boxes and Tutorials

2.2.1 You may see "welcome" and "helper" boxes in the UI when you log in for the first time. Also a "Enjoy your free trial…" ribbon at the top of the UI. Minimize and close them by clicking on the items in the red boxes on the screenshot below.
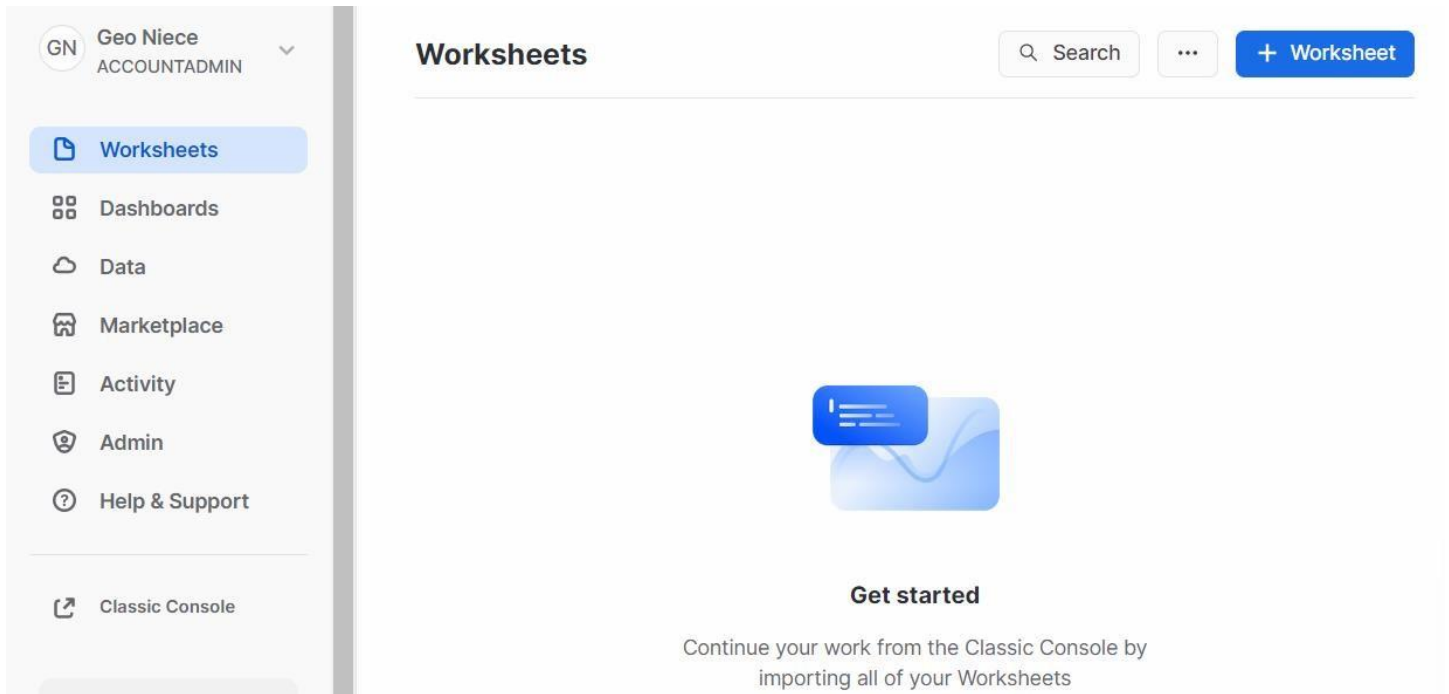
2.3 Navigate the new Snowflake UI App

  2.3.1 Exploring the new Snowflake UI.

2.4 Navigating the Snowflake UI

First let's get you acquainted with Snowflake's Next Generation User Interface! This section covers the basic components of the user interface to help you orient yourself.

  2.4.1 The left menu allows you to switch between the different areas of Snowflake. Please click on Worksheet(on the left menu) and then +Worksheet(on the top right) :

2.4.2 The **Worksheet** tab opens an interface for submitting SQL queries, performing DDL and DML operations and viewing results as your queries/operations complete.

Copy and paste the entire SQL text from the .sql file that you downloaded before.

All of the SQL commands you need to run for the remainder of this experiment will now appear on the new worksheet. Do not run any of the SQL commands yet. We will come back to them later in the experiment and execute them one at a time.

⛔ **Warning - Do Not Copy/Paste SQL From our PDF to a Worksheet**
Copy-pasting the SQL code from this PDF into a Snowflake worksheet will result in formatting errors and the SQL will not run correctly.

Please open the .sql file with a text editor and then copy/paste all the text from the .sql file into the "Worksheet"

ℹ️ **Worksheets vs the UI**
Much of the configurations in this experiment will be executed via this prewritten SQL in the Worksheet in order to save time. These configurations could also be done via the UI in a less technical manner but would take more time.

Module 13.3: Preparing to Load Data

Let's start by preparing to load the structured & semi structured data into Snowflake. This module will walk you through the steps to:
- Create a database
- Create an external stage
- Set the context for the worksheet

3.1 Create a Database and Stage

3.1.1 First, let's create a database called CITIBIKE that will be used for loading the data. At the top of the Snowflake UI, click the Worksheets tab. You should see the worksheet with all the SQL we loaded in a prior step.

3.1.2 First, we need to set the context appropriately within the Worksheet. We will set the worksheet context via SQL commands in the worksheet. Select the highlighted SQL commands and run them to set the context.
We will set the following context: Role:
SYSADMIN
Warehouse: PIPE_WH (M)
Database: CITIBIKE
Schema = PUBLIC

create or replace warehouse pipe_wh with warehouse_size = 'medium' warehouse_type = 'standard' auto_suspend = 120 auto_resume = true;
use role sysadmin;
use warehouse pipe_wh; create
database if not exists citibike;
use schema public;

3.1.3 Let's now create a table called TRIPS_STREAM that will be used for loading the structured data. We will be using the Worksheets tab in the Snowflake UI to run the DDL (data definition language) to create the table. Based on a prior step, the SQL text below should be showing on the worksheet.

```
create or replace table trips_stream
(tripduration integer, starttime
  timestamp, stoptime
  timestamp, start_station_id
  integer, end_station_id
  integer,
  bikeid integer,
  usertype string,
  birth_year integer, gender
  integer, program_id
  integer);
```

Many Options to Run Commands.
SQL commands can be executed through the UI (limited), via the Worksheets tab, using our SnowSQL command line tool, a SQL editor of your choice via ODBC/JDBC, or through our Python or Spark connectors.

As mentioned earlier, in this experiment we will run some operations via prewritten SQL in the worksheet (as opposed to using the UI) to save time.

3.1.4 Run the query by placing your cursor anywhere in the command and clicking the blue "Run" button at the top of the page or by hitting Ctrl/Cmd+Enter on your keyboard.

Warning
In this experiment, never check the "All Queries" box at the top of the worksheet. We want to run SQL queries one at a time in a specific order; not all at once.

```
 7
 8    --create trips table to load data from pipe
 9    create or replace table trips_stream
10    (tripduration integer,
11      starttime timestamp,
12      stoptime timestamp,
13      start_station_id integer,
14      end_station_id integer,
15      bikeid integer,
16      usertype string,
17      birth_year integer,
18      gender integer,
19      program_id integer);
20
```

3.2 Create an External Stage

We are working with structured data that will be staged in a public, external S3 bucket. Before we can use this data, we first need to create a Stage that specifies the location of our external bucket.

NOTE - For this experiment we are using an AWS-East bucket. In the real-world, to prevent data egress/transfer costs, you would want to select a staging location from the same cloud provider and region that your Snowflake environment is in.

3.2.1 Let's create the external stage for TRIPS data. We have files landing on these AWS S3 buckets periodically which mimics the real-time data ingestion scenario in Production environments.

```
24
25    --create stage for trips
26    create or replace stage pipe_data_trips url = 's3://snowflake-workshop-lab/snowpipe/trips/' file_format=(type=csv);
27
```

NOTE - We have defined the file format along with the external stage definition. You may define it with the COPY statement for the Snowpipe as well.

3.2.2 Next, we will list the files in the external stages to see files that are already present in the buckets.

```
27
28    list @pipe_data_trips;
29
```

NOTE - The S3 bucket for this experiment is public so you can leave the key fields empty. In the "real world" this bucket would likely require key information.

You should see the output in the Results window in the bottom pane:

	name	size	md5	last_modified
1	s3://snowflake-workshop-lab/snowpipe/trips/trips_stream_Mon Jul 26 2021 12:03:	1,653,595	87d328806738143fcf683b7e8cd839e4	Mon, 26 Jul 2021 1
2	s3://snowflake-workshop-lab/snowpipe/trips/trips_stream_Mon Jul 26 2021 12:04:	2,055,584	3f24adfc876752fff3117fcd763ac4a5	Mon, 26 Jul 2021 1
3	s3://snowflake-workshop-lab/snowpipe/trips/trips_stream_Mon Jul 26 2021 12:05:	2,123,239	5e809c37c034e561cda06236d7afc9a8	Mon, 26 Jul 2021 1
4	s3://snowflake-workshop-lab/snowpipe/trips/trips_stream_Mon Jul 26 2021 12:06:	2,364,654	80064e9c02aa8405499cc242705f8f08	Mon, 26 Jul 2021 1

3.3 Create the pipe

Before we can load the data into Snowflake, we have to create the Pipes that defines the data to be loaded via COPY command.

> 3.3.1 Now we will create the pipe via "CREATE PIPE" SQL command. We are using a predefined SNS_Topic for auto-detection of new files in the external S3 buckets that we are using for our experiment. COPY statement defines which stage the files will be tracked from and which table the data will be loaded into.

```
30     -- create the trips pipe using SNS topic
31     create or replace pipe trips_pipe auto_ingest=true
32     aws_sns_topic='arn:aws:sns:us-east-1:484577546576:snowpipe_sns_lab'
33     as copy into trips_stream from @pipe_data_trips/;
34
```

The following options for automating Snowpipe using Amazon SQS are supported:

- **Option 1.** New S3 event notification: Create an event notification for the target path in your S3 bucket. The event notification informs Snowpipe via an SQS queue when files are ready to load.
  This is the most common option.

Important !!!
If a conflicting event notification exists for your S3 bucket, use Option 2 instead.

- **Option 2.** Existing event notification: Configure Amazon Simple Notification Service (SNS)as a broadcaster to share notifications for a given path with multiple endpoints (or "subscribers," e.g. SQS queues or AWS Lambda workloads), including the Snowflake SQS queue for Snowpipe automation. An S3 event notification published by SNS informs Snowpipe via an SQS queue when files are ready to load.

> 3.3.2 We can check the pipe status via the System function "system$pipe_status()"

```
36
37     --check trips pipe status
38     select system$pipe_status('trips_pipe');
39
```

- When there are no files pending to be loaded, then the status looks like below : Notice the pending FIle count as 0

	SYSTEM$PIPE_STATUS('TRIPS_PIPE')
1	{"executionState":"RUNNING","pendingFileCount":0,"notificationChannelName":"arn:aws:sqs:us-east-1:582086746320:sf-snowpipe-AIDAYPBYS4DIK5UQPRUVA-sm

- Once there are files pending to be loaded, then the status looks like below : Notice the pending FIle count as 1

	SYSTEM$PIPE_STATUS('TRIPS_PIPE')
1	{"executionState":"RUNNING","oldestFileTimestamp":"2021-07-26T19:09:39.319Z","pendingFileCount":1,"notificationChannelName":"arn:aws:sqs:us-east-1:58208

Module 13.4: Verify Data Loading

For this module, we will verify that the pipe is loading the data automatically.

4.1 Check copy_history to track files that have been loaded

For loading data with Snowpipe, no compute resources are needed to be run explicitly. Snowflake uses its own managed Virtual Warehouses to load data via Snowpipe. This helps save cost and is a better approach than having to run a Warehouse all the time for continuously streaming data

4.1.1 We will run the query against the "information_schema.copy_history" table function to track the files that have been loaded.

```
39
40     -- show the files that have been processed
41     select *
42     from table(information_schema.copy_history(table_name=>'TRIPS_STREAM', start_time=>dateadd('hour', -1,
       CURRENT_TIMESTAMP()))));
43
```

The results look like this after a few files have been loaded :

	FILE_NAME	STAGE_LOCATION	LAST_LOAD_TIME
1	trips_stream_Mon Jul 26 2021 11:29:56 GMT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/	2021-07-26 11:31:52.122 -0700
2	trips_stream_Mon Jul 26 2021 11:30:55 GMT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/	2021-07-26 11:31:52.122 -0700
3	trips_stream_Mon Jul 26 2021 11:31:46 GMT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/	2021-07-26 11:32:44.035 -0700
4	trips_stream_Mon Jul 26 2021 11:32:55 GMT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/	2021-07-26 11:34:04.016 -0700
5	trips_stream_Mon Jul 26 2021 11:33:46 GMT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/	2021-07-26 11:34:52.052 -0700
6	trips_stream_Mon Jul 26 2021 11:34:55 GMT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/	2021-07-26 11:35:52.025 -0700
7	trips_stream_Mon Jul 26 2021 11:35:46 GMT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/	2021-07-26 11:36:52.077 -0700
8	trips_stream_Mon Jul 26 2021 11:36:46 GMT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/	2021-07-26 11:37:52.018 -0700

4.1.2 We can check table records to verify data.
- Trips Data(Structured)

```
64    select * from trips_stream limit 5;
```

	TRIPDURATION	STARTTIME	STOPTIME	START_STATION_ID	END_STATION_ID	BIKEID	USERTY
1	330	2018-08-02 10:24:30.960	2018-08-02 10:30:01.654	268	3,260	30,042	Subscri
2	330	2018-08-02 10:24:30.960	2018-08-02 10:30:01.654	268	3,260	30,042	Subscri
3	330	2018-08-02 10:24:30.960	2018-08-02 10:30:01.654	268	3,260	30,042	Subscri
4	500	2018-08-02 10:31:01.459	2018-08-02 10:39:22.182	268	426	21,089	Subscri
5	500	2018-08-02 10:31:01.459	2018-08-02 10:39:22.182	268	426	21,089	Subscri

Module 13.5: Analyzing & Securing the Data

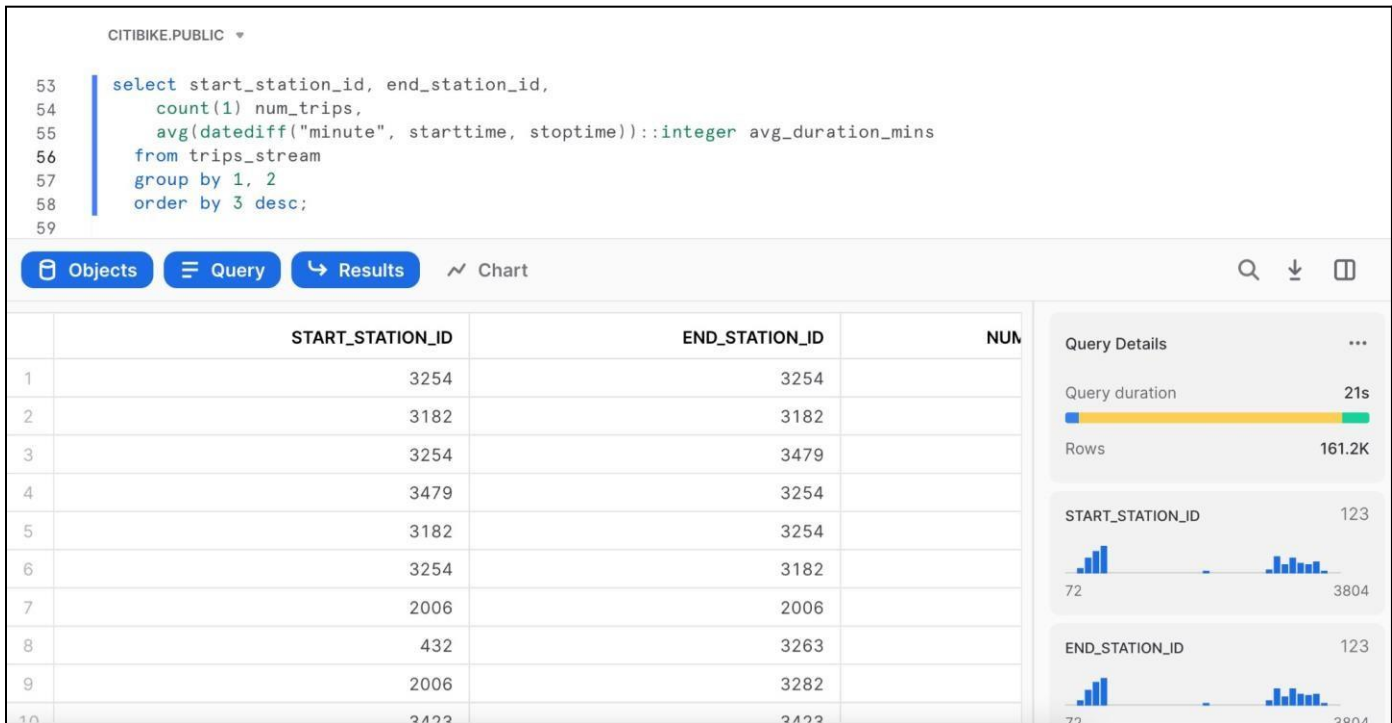For this module, we will analyze the data that is being auto-ingested and we will dynamically mask the data for unauthorized roles.

5.1 We can now analyze the data via SQL

5.1.1 "What are the most popular cycle routes and how long do they take?"

```
select start_station_id, end_station_id, count(1)
   num_trips,  avg(datediff("minute",  starttime,
   stoptime))::integer  avg_duration_mins  from
   trips_stream
```
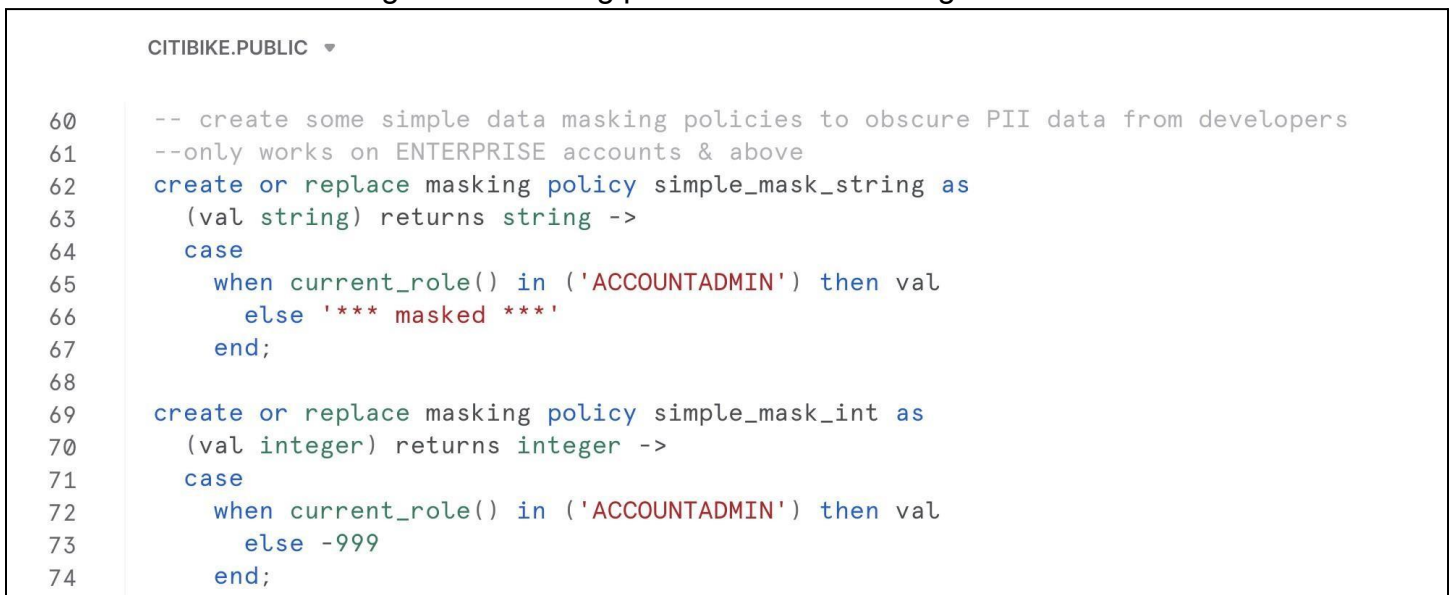
```
            group by 1, 2
            order by 3 desc;
```

```
CITIBIKE.PUBLIC  ▾

53    select start_station_id, end_station_id,
54        count(1) num_trips,
55        avg(datediff("minute", starttime, stoptime))::integer avg_duration_mins
56    from trips_stream
57    group by 1, 2
58    order by 3 desc;
59
```

	START_STATION_ID	END_STATION_ID	NUM
1	3254	3254	
2	3182	3182	
3	3254	3479	
4	3479	3254	
5	3182	3254	
6	3254	3182	
7	2006	2006	
8	432	3263	
9	2006	3282	
10	3423	3423	

Query Details   ···

Query duration   21s

Rows   161.2K

START_STATION_ID   123

72   3804

END_STATION_ID   123

72   3804

5.2 We can now secure the data via Dynamic Data Masking(DDM)

Dynamic Data Masking is a Column-level Security feature that uses masking policies to selectively mask data at query time that was previously loaded in plain-text into Snowflake.

5.2.1 We can create generic masking policies : One for Strings & One for Numbers

```
CITIBIKE.PUBLIC  ▾

60    -- create some simple data masking policies to obscure PII data from developers
61    --only works on ENTERPRISE accounts & above
62    create or replace masking policy simple_mask_string as
63      (val string) returns string ->
64      case
65        when current_role() in ('ACCOUNTADMIN') then val
66          else '*** masked ***'
67      end;
68
69    create or replace masking policy simple_mask_int as
70      (val integer) returns integer ->
71      case
72        when current_role() in ('ACCOUNTADMIN') then val
73          else -999
74      end;
```

Note how we create the policies in such a way that only ACCOUNTADMIN role will have access to the data and all other roles will see masked data.

5.2.2 We can apply the masking policies to the TRIPS_STREAM table columns
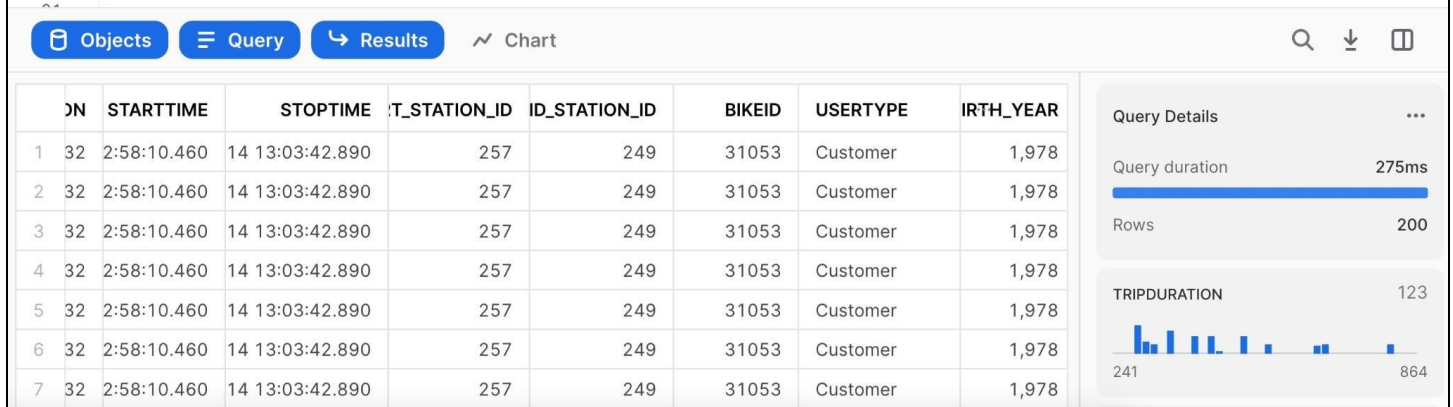
```
76
77      -- apply them to TRIPS table to protect our PII data
78      alter table trips_stream modify column bikeid set masking policy simple_mask_int;
79      alter table trips_stream modify column birth_year set masking policy simple_mask_int;
80      alter table trips_stream modify column gender set masking policy simple_mask_int;
81      alter table trips_stream modify column usertype set masking policy simple_mask_string;
82
```

5.2.3 Grant permissions to role SYSADMIN on the objects

```
82
83      -- grant explicit permissions to SYSADMIN on the environment
84      grant all on warehouse pipe_wh to role sysadmin;
85      grant all on database citibike to role sysadmin;
86      grant all on all schemas in database citibike to role sysadmin;
87      grant all on all tables in database citibike to role sysadmin;
88
```

5.2.4   Let's take a look at the data with ACCOUNTADMIN privileges.

```
88
89      -- data is now secure from developers but available to ACCOUNTADMIN
90    | select * from trips_stream limit 200;
```

Objects   Query   Results   Chart   🔍 ↓ ⬚

	ON	STARTTIME	STOPTIME	T_STATION_ID	ID_STATION_ID	BIKEID	USERTYPE	IRTH_YEAR
1	32	2:58:10.460	14 13:03:42.890	257	249	31053	Customer	1,978
2	32	2:58:10.460	14 13:03:42.890	257	249	31053	Customer	1,978
3	32	2:58:10.460	14 13:03:42.890	257	249	31053	Customer	1,978
4	32	2:58:10.460	14 13:03:42.890	257	249	31053	Customer	1,978
5	32	2:58:10.460	14 13:03:42.890	257	249	31053	Customer	1,978
6	32	2:58:10.460	14 13:03:42.890	257	249	31053	Customer	1,978
7	32	2:58:10.460	14 13:03:42.890	257	249	31053	Customer	1,978

Query Details   ...

Query duration   275ms

Rows   200

TRIPDURATION   123

241   864

Please note how none of the columns are masked as ACCOUNTADMIN role has access to the columns.

5.2.5 Now we can test our policies by switching the role to SYSADMIN & run the same query as we ran in the above step.

```
91
92     -- switch role to SYSADMIN now
93     use role sysadmin;
94
95     -- notice that for SYSADMIN role, the 4 columns(BIKEID, USERTYPE, BIRTH_YEAR, GENDER) are masked
96   | select * from trips_stream limit 200;
```


Please note how the 4 columns that we applied the masking policies to are now masked for SYSADMIN role.

Summary & Next Steps

This experiment was designed as a hands-on introduction to Snowpipe to simultaneously teach you how to use it, while showcasing some of its key capabilities and differentiators. We covered how to ● Create stages, databases, tables and warehouses

- Create pipe for auto ingestion of data
- Auto Ingest structured and semi-structured data
- Query data including joins between tables