

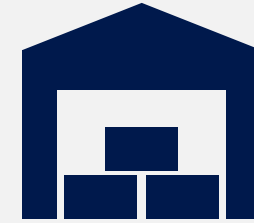


CREATION OF APPLICATIONS WITH REAL-TIME DATA INGESTION



1. Creation of a Warehouse

Optimizing Data Warehouse Performance



```
1  -- CREATING A WORKSPACE
2  -----
3
4
5  --Sysadmin role to allow the creation of Warehouse
6  use role sysadmin;
7
8  -----
9  --1.CREATING A WAREHOUSE
10 -----
11 --Starting with a XSMALL warehouse and we t
12 create warehouse if not exists pipe_wh
13     warehouse_size = 'XSMALL'
14     auto_suspend=120
15     initially_suspended=true;
16
17 use warehouse pipe_wh
18
19 -----
```

	name	size	md5
1	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:38:36 GMT-0700	1,942,537	a7de9103e89417e7e5649c9a74451z
2	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:40:57 GMT-0700	2,745,319	5895865197726103f7254d556d35e
3	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:42:00 GMT-0700	2,760,603	f63155d5ddd77e30b99bb865dafcf3e
4	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:42:58 GMT-0700	2,608,630	6090af4eb6b0d608b9eade463dd024

Query Details

- Query duration: 1.3s
- Rows: 567
- Query ID: 01af0b59-0000-26a4...

- Creating a data warehouse named "pipe_wh."
- Starting with an "XSMALL" warehouse size.
- Setting auto-suspend to 120 minutes.
- Initially suspending the warehouse.

2. Creation of a Database

LOREM IPSUM DOLOR SIT AMET

The screenshot shows the Snowflake Snowpipe web interface. The browser address bar shows 'app.snowflake.com/jinoszv/mu85263/w5Su8Fsi8poL#query'. The interface has a sidebar on the left with 'Databases' and 'Worksheets' tabs. Under 'Databases', there is a tree view showing 'CITIBIKE' with sub-items 'INFORMATION_SCHEMA' and 'PUBLIC'. The main area displays a SQL query in the 'CITIBIKE.PUBLIC' schema. The query is as follows:

```
--2.CREATING A DATABASE(working area)
create database if not exists citibike;
--specify the default schema for the session
use schema citibike.public;

--3.Table creation of trips - trips data loaded from snowpipe as csv
create or replace table trips_stream (
  tripduration integer,
  starttime timestamp,
  stoptime timestamp,
  start_station_id integer,
  end_station_id integer,
  bikeid integer,
  usertype string,
```

The bottom of the interface shows a 'Results' tab with a table of query results. The table has columns 'name', 'size', and 'md5'. The results are as follows:

	name	size	md5
1	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:38:36 GMT-0700	1,942,537	a7de9103e89417e7e5649c9a74451e
2	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:40:57 GMT-0700	2,745,319	5895865197726103f7254d556d35e
3	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:42:00 GMT-0700	2,760,603	f63155d5ddd77e30b99bb865dafcf3e
4	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:42:56 GMT-0700	2,600,600	6000cf4cb8b0d608b800de462dd024

On the right side of the results table, there is a 'Query Details' panel showing 'Query duration' as 1.3s, 'Rows' as 567, and 'Query ID' as 01af0b59-0000-26a4-...

- Creating a database named "citibike" if it doesn't already exist.
- Specifying the default schema for the session as "citibike.public."

3. Creation of Tables

Creating trips_stream and trips_stream tables



The screenshot shows the Snowflake Snowpipe interface. On the left, the 'Databases' sidebar is expanded, showing the 'CITIBIKE' database and its 'PUBLIC' schema. Under 'Tables', 'JSON_WEATHER_STREAM' and 'TRIPS_STREAM' are listed. The main panel displays a SQL query for creating these tables. The query includes comments and SQL syntax for creating a table with a variant field. The 'Results' tab at the bottom shows a table with 4 rows of query results.

```
34      start_station_id integer,  
35      end_station_id integer,  
36      bikeid integer,  
37      usertype string,  
38      birth_year integer,  
39      gender integer,  
40      program_id integer  
41    );  
42  
43  
44    --4. Create a weather table - data loaded from snowpipe as JSON  
45    -----  
46    -- The data being semi structured, we will user a variant field  
47    create or replace table json_weather_stream(  
48      v variant  
49    );  
50  
51    -----  
52    --5. CREATING STAGE  
53    -----
```

	name	size	md5
1	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:38:36 GMT-070	1,942,537	a7de9103e89417e7e5649c9a74451e
2	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:40:57 GMT-070	2,745,319	5895865197726103f7254d556d35e
3	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:42:00 GMT-070	2,760,603	f63155d5ddd77e30b99bb865dafcf3e
4	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:42:56 GMT-070	2,600,820	6080af4eb6b0d609b0eade463d4024

Query Details

- Query duration: 1.3s
- Rows: 567
- Query ID: 01af0b59-0000-26a4-...

- The trips_stream table captures information about bike trips, including user details and station IDs.
- The json_weather_stream table is designed to handle semi-structured weather data in JSON format.
- Using a variant field allows flexibility in handling diverse weather data structures.

4. Creation of Stages and pipes

Efficient Data Management and Automation

The screenshot displays the Snowflake Snowpipe interface. On the left, the 'Databases' sidebar shows the 'CITIBIKE' database with a 'PUBLIC' schema. Under 'Tables', 'JSON_WEATHER_STREAM' and 'TRIPS_STREAM' are listed. Under 'Stages', 'PIPE_DATA_TRIPS' and 'PIPE_DATA_WEATHER' are listed. Under 'Pipes', 'TRIPS_PIPE' and 'WEATHER_PIPE' are listed. The main panel shows the SQL script for creating these stages and pipes. The script includes comments for auto-ingest and SNS notifications, and uses the 'create or replace pipe' and 'copy into' commands. The 'Results' tab at the bottom shows a table with columns 'name', 'size', and 'md5', containing four rows of data.

```
--TRIPS PIPE
--to auto load information
--copy into TRIPS_STREAM
--  from @pipe_data_trips;

create or replace pipe trips_pipe
  auto_ingest=true
  --to be notified when there is activity
  aws_sns_topic='arn:aws:sns:us-east-1:484577546576:snowpipe_sns_lab'
as
  copy into TRIPS_STREAM
  from @pipe_data_trips;

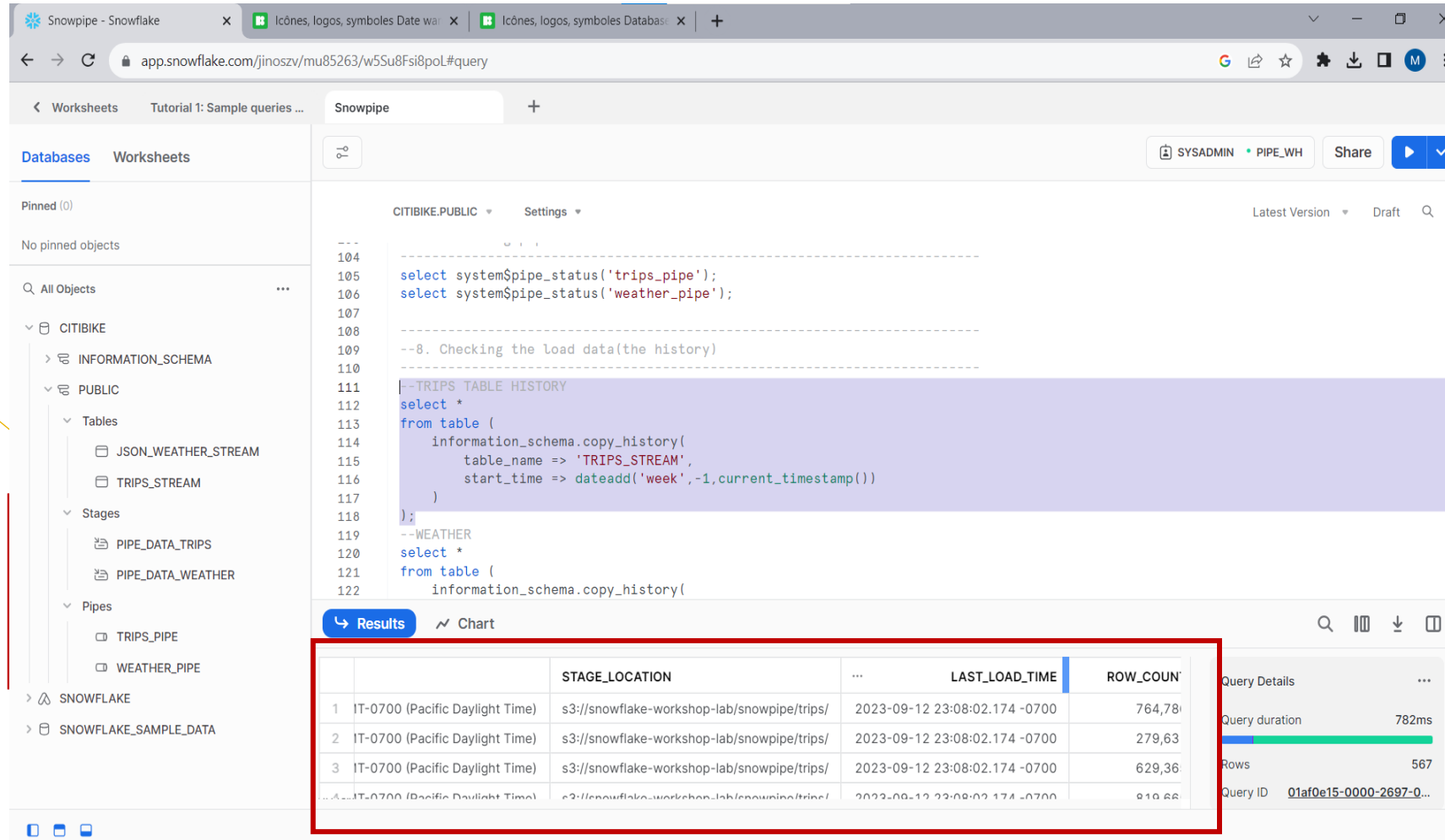
--WEATHER PIPE
create or replace pipe weather_pipe
  auto_ingest=true
  --to be notified when there is activity
  aws_sns_topic='arn:aws:sns:us-east-1:484577546576:snowpipe_sns_lab'
```

	name	size	md5
1	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:38:36 GMT-07C	1,942,537	a7de9103e89417e7e5649c9a74451e
2	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:40:57 GMT-07C	2,745,319	5895865197726103f7254d556d35e
3	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:42:00 GMT-07C	2,760,603	f63155d5ddd77e30b99bb865dafcf3e
4	s3://snowflake-workshop-lab/snowpipe/trips/Thu Aug 31 2023 10:42:58 GMT-07C	2,808,828	6880ef4eb6b0d608b8e0dc463d4034

- **Trips Data Stage:**
 - Stores CSV-formatted trip data.
 - Location: 's3://snowflake-workshop-lab/snowpipe/trips/'
- **Weather Data Stage:**
 - Stores JSON-formatted weather data.
 - Location: 's3://snowflake-workshop-lab/weather/'
 - Use list commands to browse stored files in each stage.
- **Trips Pipe:**
 - Automatically loads trip data into TRIPS_STREAM.
 - Monitors activity through AWS SNS.
- **Weather Pipe:**
 - Automatically loads weather data into json_weather_stream.
 - Monitors activity through AWS SNS.
- Use of show pipes to view and manage data loading processes.

5. Checking the history after using pipes

Tracking Data Loading Activities



```
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
```

```
select system$pipe_status('trips_pipe');
select system$pipe_status('weather_pipe');

--8. Checking the load data(the history)

--TRIPS TABLE HISTORY
select *
from table (
  information_schema.copy_history(
    table_name => 'TRIPS_STREAM',
    start_time => dateadd('week',-1,current_timestamp())
  )
);

--WEATHER
select *
from table (
  information_schema.copy_history(
```

		STAGE_LOCATION	...	LAST_LOAD_TIME	ROW_COUNT
1	IT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/		2023-09-12 23:08:02.174 -0700	764,781
2	IT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/		2023-09-12 23:08:02.174 -0700	279,63
3	IT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/		2023-09-12 23:08:02.174 -0700	629,36
...	IT-0700 (Pacific Daylight Time)	s3://snowflake-workshop-lab/snowpipe/trips/		2023-09-12 23:08:02.174 -0700	910,68

- This SQL query retrieves the copy history of the 'TRIPS_STREAM' table.
- It allows you to monitor and review recent data loading activities.
- The START_TIME parameter is set to the past week for a specific time frame.

6. Checking data in our trip Table

Queries on Structured and Unstructured Data

The screenshot shows the Snowflake web interface. On the left, the 'Databases' sidebar is expanded to 'CITIBIKE', then 'PUBLIC', and finally 'Tables', where 'TRIPS_STREAM' is selected. The main pane displays a SQL query:

```
123 table_name => 'JSON_WEATHER_STREAM',
124 start_time => dateadd('hour', -1, current_timestamp())
125 )
126 );
127
128 --Number of rows
129 select count (*) from json_weather_stream;
130 select count (*) from trips_stream;
131
132 -- trips data visualisation
133 select *
134 from trips_stream
135 limit 5;
136
137 -- weather data visualisation
138 select *
139 from json_weather_stream;
140
141
142
```

Below the query, the 'Results' tab is active, showing a table with 5 rows and 6 columns. The table is highlighted with a red rectangle:

	TRIPDURATION	STARTTIME	STOPTIME	START_STATION_ID	END_STATION_ID
1	1,148	2018-09-19 00:07:55.609	2018-09-19 00:27:03.865	389	3571
2	1,148	2018-09-19 00:07:55.609	2018-09-19 00:27:03.865	389	3571
3	1,148	2018-09-19 00:07:55.609	2018-09-19 00:27:03.865	389	3571
4	1,148	2018-09-19 00:07:55.609	2018-09-19 00:27:03.865	389	3571

Query details on the right show a duration of 526ms and 5 rows.

The screenshot shows the Snowflake web interface. On the left, the 'Databases' sidebar is expanded to 'CITIBIKE', then 'PUBLIC', and finally 'Tables', where 'TRIPS_STREAM' is selected. The main pane displays a SQL query:

```
156 where v:city.name in ('New York','Seattle','San Francisco','Miami');
157
158
159
160
161 SELECT 'Total' AS conditions,
162 SUM(iff(v:city.name::string = 'New York', 1, 0)) AS nyc_freq
163 FROM json_weather_stream w, LATERAL FLATTEN(input => w.v:weather) wf
164 WHERE v:city.name IN ('New York', 'Seattle', 'San Francisco', 'Miami')
165 UNION ALL
166 select value::main::string as conditions,
167 sum(iff(v:city.name::string='New York',1,0)) as nyc_freq
168 --sum(iff(v:city.name::string='Seattle',1,0)) as seattle_freq,
169 --sum(iff(v:city.name::string='San Francisco',1,0)) as san_francisco_freq,
170 --sum(iff(v:city.name::string='Miami',1,0)) as miami_freq
171 from json_weather_stream w,
172 lateral flatten (input => w.v:weather) wf
173 where v:city.name in ('New York','Seattle','San Francisco','Miami')
174 group by conditions;
```

Below the query, the 'Results' tab is active, showing a table with 4 rows and 2 columns. The table is highlighted with a red rectangle:

	CONDITIONS	NYC_FREQ
1	Total	70,993
2	Clear	18,279
3	Clouds	19,674
4	Mist	11,665

Query details on the right show a duration of 93ms and 13 rows.



Thanks.



Makan DANSOKO



+33652728863



makandansoko@gmail.com



<https://makandansoko.github.io/github-portfolio/>