Module 7:   Using Time Travel
Overview

Snowflake's Time Travel capability enables accessing historical data at any point within a pre-configurable period of time. The default period of time is 24 hours with the Basic Subscription and with Snowflake Enterprise Edition it can be up to 90 days. Most data warehouses cannot offer this functionality; with Snowflake it's easy!

Some useful applications of this include:

- Restoring data-related objects (tables, schemas, and databases) that may have been accidentally or intentionally deleted
- Duplicating and backing up data from key points in the past
- Analyzing data usage/manipulation over specified periods of time

i	**TIME TRAVEL** Anyone who's ever deleted the wrong table, and that is a lot of people will tell you that there is a moment of panic. With Snowflake, it's the opposite, provided you know of the **UNDROP** command.

7.1 Drop and Undrop a Table
First let's see how we can restore data objects that have been accidentally or intentionally deleted.

7.1.1 From the worksheet, run the following command which will drop (remove) the json_weather_data table:

drop table json_weather_data;

7.1.2 Now run a SELECT statement on the json_weather_data table. In the "Results" pane you should see an error because the underlying table has been dropped.
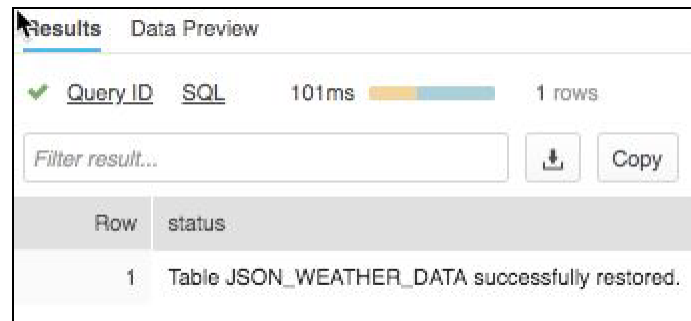
select * from json_weather_data limit 10;

7.1.3 Now restore the table:

```
undrop table json_weather_data;
```

7.1.4 The json_weather_data table should be restored.

7.2 Roll Back a Table

Now let's look rolling back a table to a previous state to fix an unintentional DML error that replaces all the station names in the Citibike database TRIPS table with the word "oops."

7.2.1 First make sure the worksheet is in the proper context:

```
use role sysadmin;
use warehouse compute_wh;
use database citibike;
use schema public;
```

7.2.2 Then run the following command that replaces all the station names in the table with the word "oops".

```
update trips set start_station_name = 'oops';
```

7.2.3 Now run a query that returns the top 20 stations by # of rides - notice how we've screwed up the station names so we only get one row:

```
select
start_station_name as "station",
count(*) as "rides"
from  trips
group by 1
order by 2 desc
limit 20;
```

7.2.4 Normally, we would need to scramble and hope we have a backup lying around. But in Snowflake, we can simply run commands to find the query ID of the last UPDATE command & store it in a variable called $QUERY_ID…
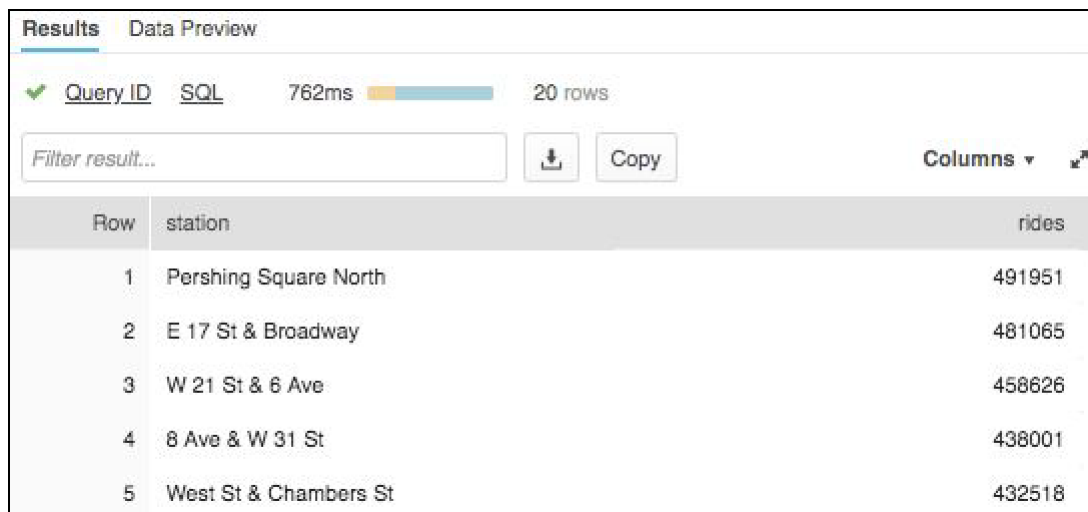
```
set query_id = (select
query_id from
table(information_schema.query_history_by_session (result_limit=>5))
where query_text like 'update%' order by start_time limit 1);
```

7.2.5 Then re-create the table as of before the update:

```
create or replace table trips as
(select * from trips before (statement => $query_id));
```

7.2.6 Run the SELECT statement again to check that the station names have been restored:

```
select
start_station_name as "station",
count(*) as "rides"
from  trips group
by 1 order by 2
desclimit 20;
```

Results   Data Preview

✔ Query ID   SQL   762ms   20 rows

Filter result...   ⬇ Copy   Columns ▾ ↗

Row	station	rides
1	Pershing Square North	491951
2	E 17 St & Broadway	481065
3	W 21 St & 6 Ave	458626
4	8 Ave & W 31 St	438001
5	West St & Chambers St	432518

7.2.7 Each database has a retention setting. Select the Worksheets tab from our top level navigation and enter the following SQL

```
show databases;
```

7.2.8 Highlight and run the SQL to view our DB retention_time. Note that all of our Snowflake databases currently have a 1 day retention_time. In our Enterprise edition we're allowed 1-90 days, but setting to something much higher eats our trial credits.

Row	created_on	name	is_default	is_current	origin	owner	comment	retention_time
1	2021-11-20 1...	CITIBIKE	N	Y		SYSADMIN		1
2	2021-11-07 ...	DEMO_DB	N	N		SYSADMIN	demo datab...	1
3	2021-11-07 ...	SNOWFLAK...	N	N	SFC_SAMPL...	ACCOUNTA...	TPC-H, Ope...	1

7.2.9 Data retention time can be altered. Let's modify our retention time for our Citibike database. Enter the following SQL.

Update database citibike set data_retention_time_in_days = 0;

7.2.10 Highlight the SQL and run to update. If your database or table has retention_time=0 in that case, you will not be able to use time travel feature as it won't work.
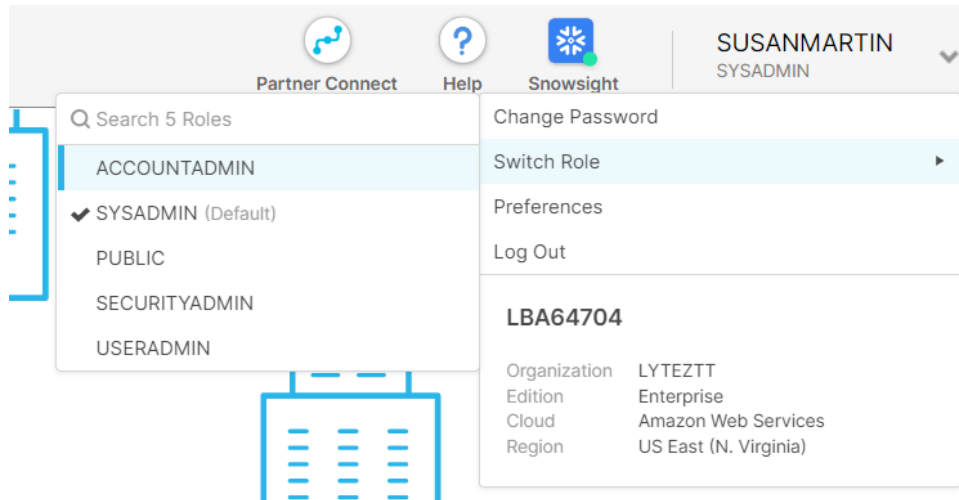
RETENTION TIME
Monitoring database retention times is key in both cost optimization and for high availability/fault tolerance discussions. Too high retention_time adds cost, too low or 0 eliminates the time travel safety net.

7.2.11 Once this retention period is over, your data will be moved to Fail-Safe.

7.2.12 Note that the Fail-Safe usage is not available to the SYSADMIN role. To view that you have to Switch Role to ACCOUNTADMIN as we noted earlier. Select the ⌄ by the USERNAME, then choose **Switch Role** to **ACCOUNTADMIN**.

7.2.13 Now we can check the Fail Safe Usage, if we had been using the account for multiple days, and see the movement of our data to failsafe that will be available after 2 days in your trial account.