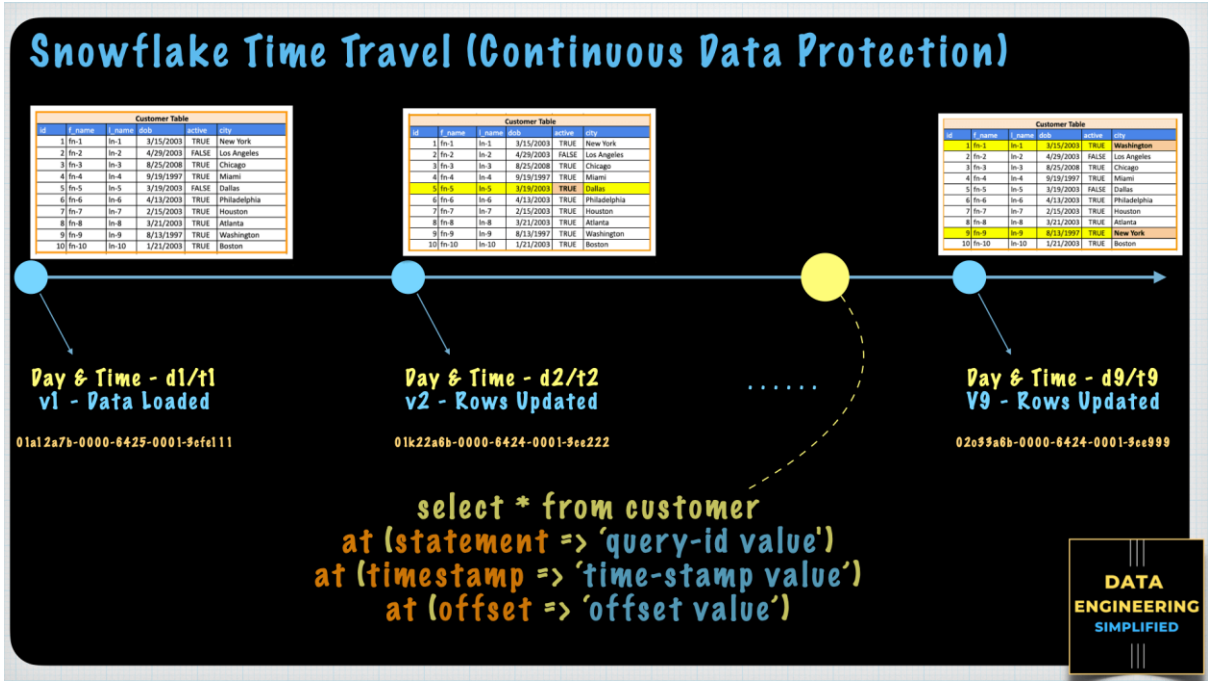
What Is Snowflake Time Travel?



Time Travel is most powerful feature introduced by snowflake, and it will be discussed in detail in this article.

Introduction to Time Travel

Data projects are complex in nature and there are many scenarios where we, as data engineers, would like to see how data looked like yesterday or at some point in time in history.

To achieve a goal like this, we put a lot of effort and bring the data with some past state and then we start our analysis or debug and that's what Snowflake's time travel feature is all about.

Snowflake Time Travel enables accessing historical data (i.e. data that has been changed or deleted) at any point within a defined period. It serves as a powerful tool for performing the following tasks:

- Restoring data-related objects (tables, schemas, and databases) that might 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.

Time travel feature in snowflake allows database or schema or tables to preserve all the changes done up to last 90 days and its extended SQL allows a data developer to fetch past state of database or schema or table without any additional burden.

Data Retention Period

A key component of Snowflake Time Travel is the data retention period.

When data in a table is modified, including deletion of data or dropping an object containing data, Snowflake preserves the state of the data before the update. The data retention period specifies the number of days for which this historical data is preserved and, therefore, Time Travel operations (SELECT, CREATE … CLONE, UNDROP) can be performed on the data.

The standard retention period is 1 day (24 hours) and is automatically enabled for all Snowflake accounts:

- For Snowflake Standard Edition, the retention period can be set to 0 (or unset back to the default of 1 day) at the account and object level (i.e. databases, schemas, and tables).
- For Snowflake Enterprise Edition (and higher):
    - For transient databases, schemas, and tables, the retention period can be set to 0 (or unset back to the default of 1 day). The same is also true for temporary tables.
    - For permanent databases, schemas, and tables, the retention period can be set to any value from 0 up to 90 days.

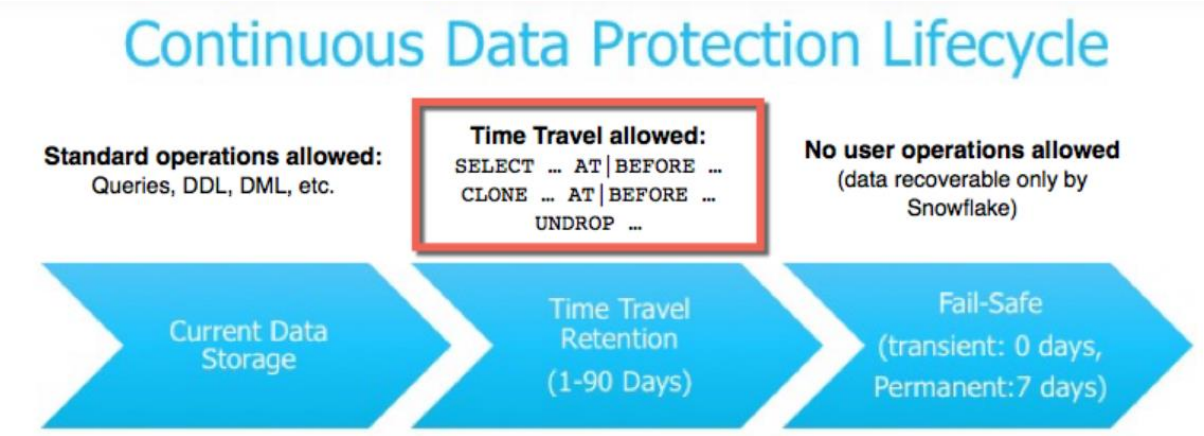
Note

A retention period of 0 days for an object effectively disables Time Travel for the object.

Advantages of Time Travel

There are many use cases and scenario where we need time travel feature desperately and you might have gone through one of them if you have managed a production grade large enterprise complex data projects.

1. You intent to modify data and mistakenly applied those DML operations (insert or update or delete) in production environment and commited them too. After some time, you realized that the changes you wanted to make is not in production and the damange is already done.
2. You have done production hot fix for a data bug and want to perform a regression testing with a state the data from past.
3. You are adding new metrics or KPI to existing data models and would like to validate before and after behaviour, in that case, you want the past state of data to simulate different state of data and validate your result.
4. If you have to restore data for any regulatory or audit purpose.



Using Time Travel, you can perform the following actions within a defined period of time:

- Query data in the past that has since been updated or deleted.
- Create clones of entire tables, schemas, and databases at or before specific points in the past.
- Restore tables, schemas, and databases that have been dropped.

Once the defined period of time has elapsed, the data is moved into Snowflake Fail-safe and these actions can no longer be performed.

Note

A long-running Time Travel query will delay moving any data and objects (tables, schemas, and databases) in the account into Fail-safe, until the query completes.

To specify the data retention period for Time Travel:

- The DATA_RETENTION_TIME_IN_DAYS object parameter can be used by users with the ACCOUNTADMIN role to set the default retention period for your account.
- The same parameter can be used to explicitly override the default when creating a database, schema, and individual table.
- The data retention period for a database, schema, or table can be changed at any time.
- The MIN_DATA_RETENTION_TIME_IN_DAYS account parameter can be set by users with the ACCOUNTADMIN role to set a minimum retention period for the account. This parameter does not alter or replace the DATA_RETENTION_TIME_IN_DAYS parameter value. However it may change the effective data retention time. When this parameter is set at the account level, the effective minimum data retention period for an object is determined by MAX(DATA_RETENTION_TIME_IN_DAYS, MIN_DATA_RETENTION_TIME_IN_DAYS).

Enabling and Disabling Time Travel

No tasks are required to enable Time Travel. It is automatically enabled with the standard, 1-day retention period.

However, you may wish to upgrade to Snowflake Enterprise Edition to enable configuring longer data retention periods of up to 90 days for databases, schemas, and tables. Note that extended data retention requires additional storage which will be reflected in your monthly storage charges. For more information about storage charges, see Storage Costs for Time Travel and Fail-safe.

Time Travel cannot be disabled for an account. A user with the ACCOUNTADMIN role can set DATA_RETENTION_TIME_IN_DAYS to 0 at the account level, which means that all databases (and subsequently all schemas and tables) created in the account have no retention period by default; however, this default can be overridden at any time for any database, schema, or table.

A user with the ACCOUNTADMIN role can also set the MIN_DATA_RETENTION_TIME_IN_DAYS at the account level. This parameter setting enforces a minimum data retention period for databases, schemas, and tables. Setting MIN_DATA_RETENTION_TIME_IN_DAYS does not alter or replace the DATA_RETENTION_TIME_IN_DAYS parameter value. It may, however, change the effective data retention period for objects. When MIN_DATA_RETENTION_TIME_IN_DAYS is set at the account level, the data retention period for an object is determined by MAX(DATA_RETENTION_TIME_IN_DAYS, MIN_DATA_RETENTION_TIME_IN_DAYS).

Time Travel can be disabled for individual databases, schemas, and tables by specifying DATA_RETENTION_TIME_IN_DAYS with a value of 0 for the object. However, if DATA_RETENTION_TIME_IN_DAYS is set to a value of 0, and MIN_DATA_RETENTION_TIME_IN_DAYS is set at the account level and is greater than 0, the higher value setting takes precedence.

⚠ Attention

Before setting DATA_RETENTION_TIME_IN_DAYS to 0 for any object, consider whether you wish to disable Time Travel for the object, particularly as it pertains to recovering the object if it is dropped. When an object with no retention period is dropped, you will not be able to restore the object.

As a general rule, we recommend maintaining a value of (at least) 1 day for any given object.

Specifying the Data Retention Period for an Object

Enterprise Edition Feature

Specifying a retention period greater than 1 day requires Enterprise Edition (or higher). To inquire about upgrading, please contact Snowflake Support.

By default, the maximum retention period is 1 day (i.e. one 24 hour period). With Snowflake Enterprise Edition (and higher), the default for your account can be set to any value up to 90 days:

- When creating a table, schema, or database, the account default can be overridden using the DATA_RETENTION_TIME_IN_DAYS parameter in the command.
- If a retention period is specified for a database or schema, the period is inherited by default for all objects created in the database/schema.

A minimum retention period can be set on the account using the MIN_DATA_RETENTION_TIME_IN_DAYS parameter. If this parameter is set at the account level,

the data retention period for an object is determined by MAX(DATA_RETENTION_TIME_IN_DAYS, MIN_DATA_RETENTION_TIME_IN_DAYS).

Changing the Data Retention Period for an Object

If you change the data retention period for a table, the new retention period impacts all data that is active, as well as any data currently in Time Travel. The impact depends on whether you increase or decrease the period:

Increasing Retention

Causes the data currently in Time Travel to be retained for the longer time period.

For example, if you have a table with a 10-day retention period and increase the period to 20 days, data that would have been removed after 10 days is now retained for an additional 10 days before moving into Fail-safe.

Note that this doesn't apply to any data that is older than 10 days and has already moved into Fail-safe.

Decreasing Retention

Reduces the amount of time data is retained in Time Travel:

- For active data modified after the retention period is reduced, the new shorter period applies.
- For data that is currently in Time Travel:
    - If the data is still within the new shorter period, it remains in Time Travel.
    - If the data is outside the new period, it moves into Fail-safe.

For example, if you have a table with a 10-day retention period and you decrease the period to 1-day, data from days 2 to 10 will be moved into Fail-safe, leaving only the data from day 1 accessible through Time Travel.

However, the process of moving the data from Time Travel into Fail-safe is performed by a background process, so the change is not immediately visible. Snowflake guarantees that the data will be moved, but does not specify when the process will complete; until the background process completes, the data is still accessible through Time Travel.

To change the retention period for an object, use the appropriate ALTER <object> command. For example, to change the retention period for a table:

```
create table mytable(col1 number, col2 date)
data_retention_time_in_days=90;

alter table mytable set data_retention_time_in_days=30;
```

⚠ Attention

Changing the retention period for your account or individual objects changes the value for all lower-level objects that do not have a retention period explicitly set. For example:

- If you change the retention period at the account level, all databases, schemas, and tables that do not have an explicit retention period automatically inherit the new retention period.
- If you change the retention period at the schema level, all tables in the schema that do not have an explicit retention period inherit the new retention period.

Keep this in mind when changing the retention period for your account or any objects in your account because the change might have Time Travel consequences that you did not anticipate or intend. In particular, we do ***not*** recommend changing the retention period to 0 at the account level.

1. Overview

Panic hits when you mistakenly delete data. Problems can come from a mistake that disrupts a process, or worse, the whole database was deleted. Thoughts of how recent was the last backup and how much time will be lost might have you wishing for a rewind button. Straightening out your database isn't a disaster to recover from with Snowflake's Time Travel. A few SQL commands allow you to go back in time and reclaim the past, saving you from the time and stress of a more extensive restore.

We'll get started in the Snowflake web console, configure data retention, and use Time Travel to retrieve historic data. Before querying for your previous database states, let's review the prerequisites for this guide.

What You'll Learn

- Snowflake account and user permissions
- Make database objects
- Set data retention timelines for Time Travel
- Query Time Travel data
- Clone past database states
- Remove database objects

- Next options for data protection

What You'll Need

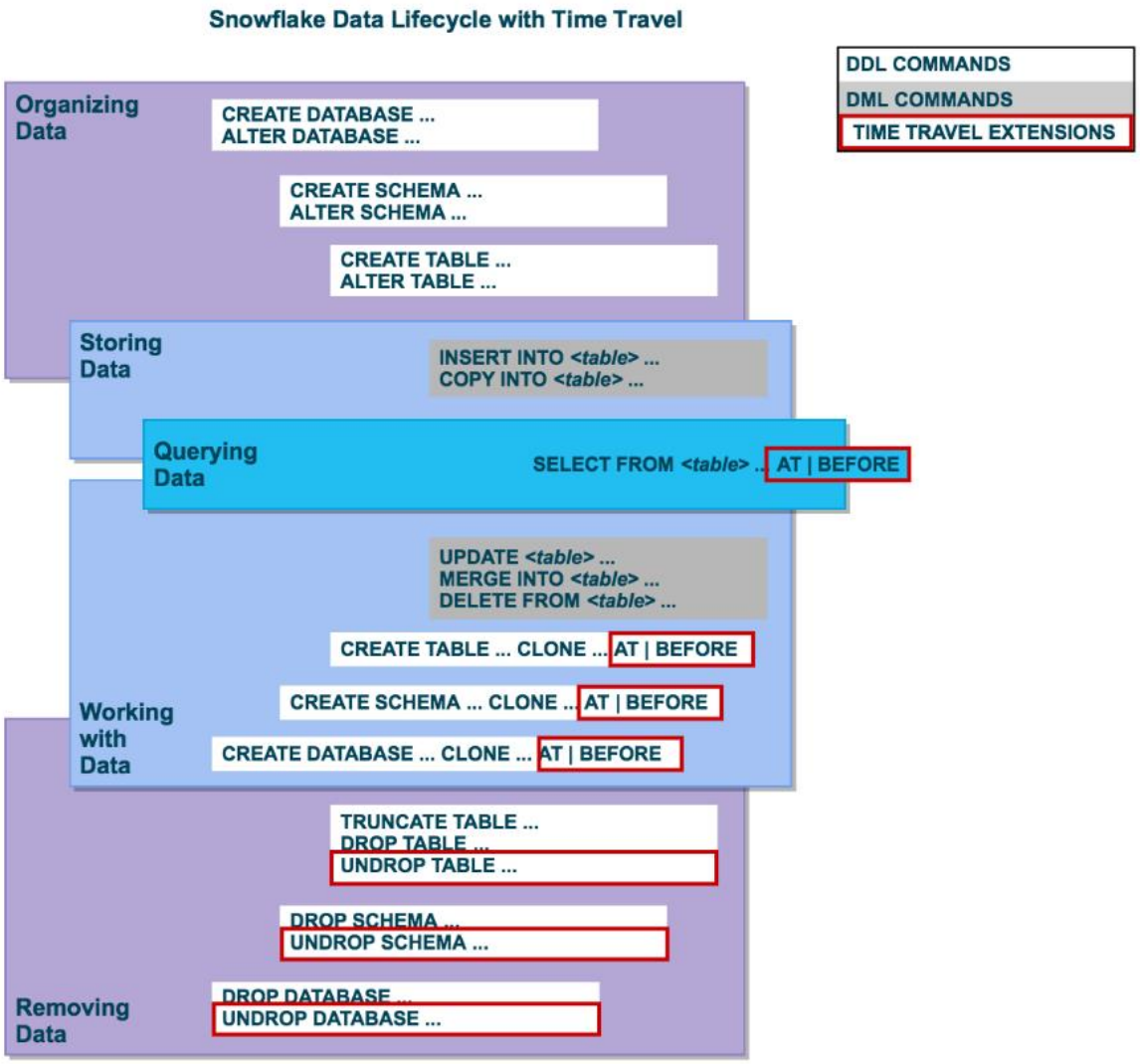
- A Snowflake Account

What You'll Build

- Create database objects with Time Travel data retention

Time Travel SQL Extensions

- To support Time Travel, the following SQL extensions have been implemented:
- AT | BEFORE clause which can be specified in SELECT statements and CREATE … CLONE commands (immediately after the object name). The clause uses one of the following parameters to pinpoint the exact historical data you wish to access:
- TIMESTAMP
- OFFSET (time difference in seconds from the present time)
- STATEMENT (identifier for statement, e.g. query ID)
- UNDROP command for tables, schemas, and databases.



Querying Historical Data

When any DML operations are performed on a table, Snowflake retains previous versions of the table data for a defined period of time. This enables querying earlier versions of the data using the AT | BEFORE clause.

This clause supports querying data either exactly at or immediately preceding a specified point in the table's history within the retention period. The specified point can be time-based (e.g. a timestamp or time offset from the present) or it can be the ID for a completed statement (e.g. SELECT or INSERT).

For example:

- The following query selects historical data from a table as of the date and time represented by the specified timestamp:

```
select * from my_table at(timestamp => 'Fri, 01 May 2015 16:20:00 -0700'::timestamp_tz);
```

- The following query selects historical data from a table as of 5 minutes ago:

```
select * from my_table at(offset => -60*5);
```

- The following query selects historical data from a table up to, but not including any changes made by the specified statement:

```
select * from my_table before(statement => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');
```

2. Get Started With the Essentials

First things first, let's get your Snowflake account and user permissions primed to use Time Travel features.

Create a Snowflake Account

Snowflake lets you try out their services for free with a trial account. A **Standard** account allows for one day of Time Travel data retention, and an **Enterprise** account allows for 90 days of data retention. An **Enterprise** account is necessary to practice some commands in this tutorial.

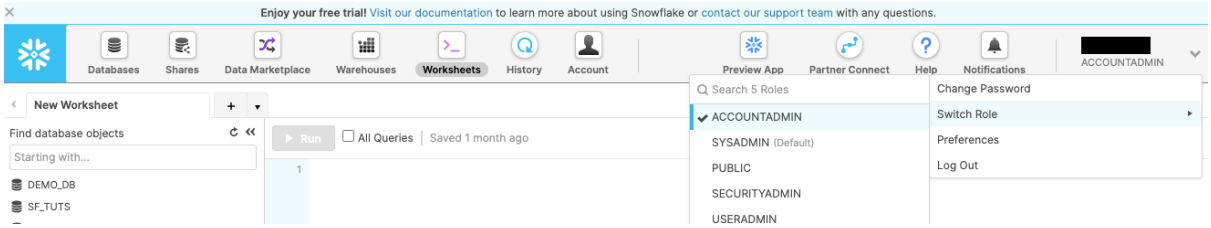
Access Snowflake's Web Console

`https://.snowflakecomputing.com/console/login`

Log in to the web interface on your browser. The URL contains your account name and potentially the region.

Increase Your Account Permission

Snowflake's web interface has a lot to offer, but for now, switch the account role from the default `SYSADMIN` to `ACCOUNTADMIN`. You'll need this increase in permissions later.



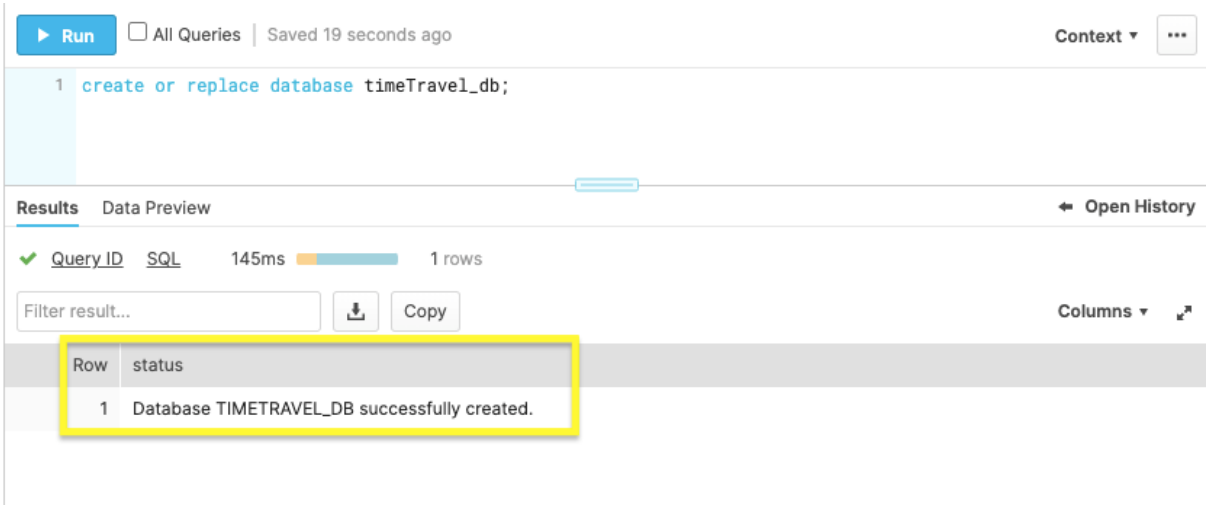
Now that you have the account and user permissions needed, let's create the required database objects to test drive Time Travel.

3. Generate Database Objects

Within the Snowflake web console, navigate to **Worksheets** and use a fresh worksheet to run the following commands.

Create Database

```
create or replace database timeTravel_db;
```



Use the above command to make a database called 'timeTravel_db'. The **Results** output will show a status message of `Database TIMETRAVEL_DB successfully created`.

Create Table

```
create or replace table timeTravel_table(ID int);
```

This command creates a table named 'timeTravel_table' on the timeTravel_db database. The **Results** output should show a status message of `Table TIMETRAVEL_TABLE successfully created`.

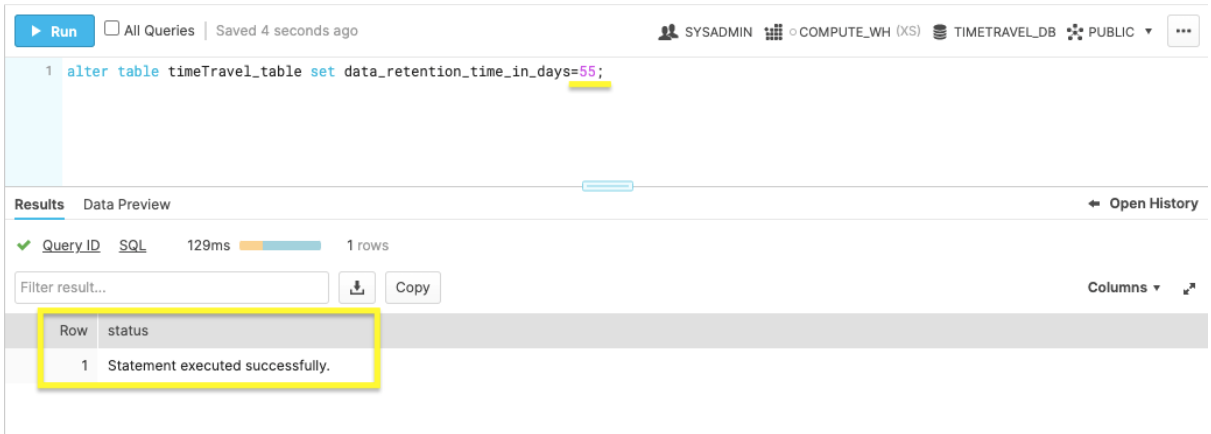
With the Snowflake account and database ready, let's get down to business by configuring Time Travel.

4. Prepare Your Database for Disaster

Be ready for anything by setting up data retention beforehand. The default setting is one day of data retention. However, if your one day mark passes and you need the previous database state back, you can't retroactively extend the data retention period. This section teaches you how to be prepared by preconfiguring Time Travel retention.

Alter Table

```
alter table timeTravel_table set data_retention_time_in_days=55;
```



The command above changes the table's data retention period to 55 days. If you opted for a **Standard** account, your data retention period is limited to the default of one day. An **Enterprise** account allows for 90 days of preservation in Time Travel.

Now you know how easy it is to alter your data retention, let's bend the rules of time by querying an old database state with Time Travel.

5. Query Your Time Travel Data

With your data retention period specified, let's turn back the clock with the `AT` and `BEFORE` clauses.

At

```
select * from timeTravel_table at(timestamp => 'Fri, 23 Oct 2020 16:20:00 -0700'::timestamp);
```

Use `timestamp` to summon the database state **at** a specific date and time.

```
select * from timeTravel_table at(offset => -60*5);
```

Employ `offset` to call the database state **at** a time difference of the current time. Calculate the offset in seconds with math expressions. The example above states, `-60*5`, which translates to five minutes ago.

Before

```
select * from timeTravel_table before(statement => '<statement_id>');
```

If you're looking to restore a database state just **before** a transaction occurred, grab the transaction's statement id. Use the command above with your statement id to get the database state right before the transaction statement was executed.

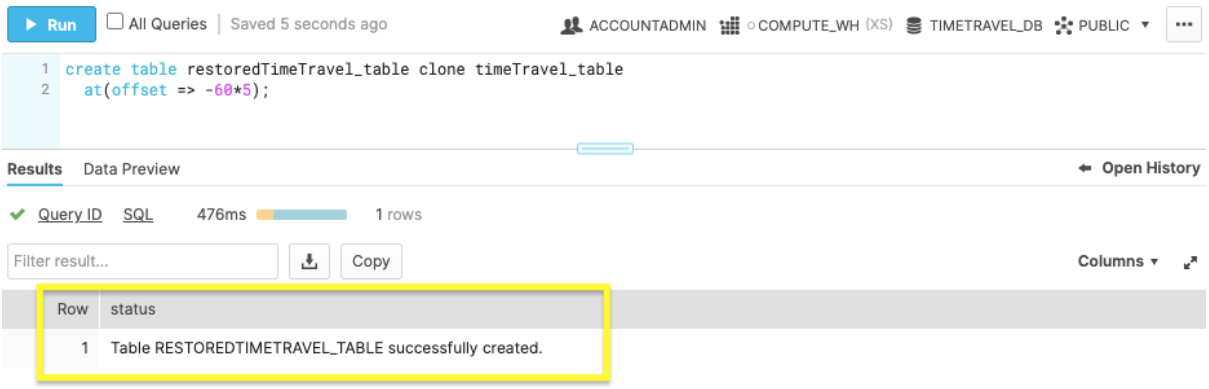
By practicing these queries, you'll be confident in how to find a previous database state. After locating the desired database state, you'll need to get a copy by cloning in the next step.

6. Clone Past Database States

With the past at your fingertips, make a copy of the old database state you need with the `clone` keyword.

Clone Table

```
create table restoredTimeTravel_table clone timeTravel_table
  at(offset => -60*5);
```



The command above creates a new table named `restoredTimeTravel_table` that is an exact copy of the table `timeTravel_table` from five minutes prior.

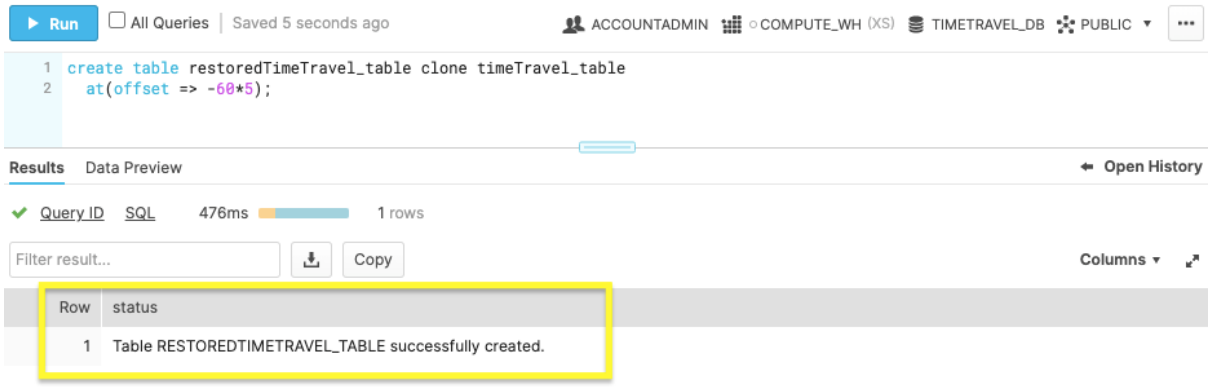
Cloning will allow you to maintain the current database while getting a copy of a past database state. After practicing the steps in this guide, remove the practice database objects in the next section.

6. Clone Past Database States

With the past at your fingertips, make a copy of the old database state you need with the `clone` keyword.

Clone Table

```
create table restoredTimeTravel_table clone timeTravel_table
  at(offset => -60*5);
```



The command above creates a new table named `restoredTimeTravel_table` that is an exact copy of the table `timeTravel_table` from five minutes prior.

Cloning will allow you to maintain the current database while getting a copy of a past database state. After practicing the steps in this guide, remove the practice database objects in the next section.