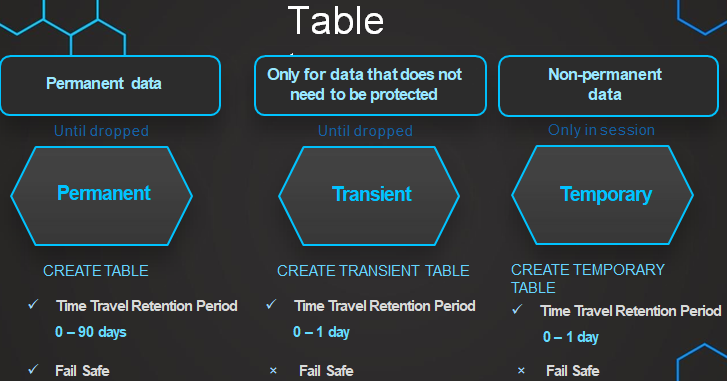
**Table types**

Tables are database objects that contain all the data, like in any other SQL language



Databases and tables used for ETL processing is that the data in those tables will be repeatedly inserted and deleted. If such tables are not specifically configured, you will be unnecessarily incurring costs for the time travel and fail-safe

that is stored with every data change that happens for those tables

Timetravel and Fail safe

Query or restore data from past without any backup

Time travel enables accessing historical data (i.e. data that has been changed or deleted) at any point within a defined period. **If we drop a table, with time travel we can restore it**. You can use it with Databases, Schemas & Tables.

To deal Data issues sometimes we cannot determine when the data was changed or whether the data has been lost altogether.

Snowflake provides an incredibly unique way of **going back in time through** the Time

Travel feature.

**Use cases of Time Travel:**

* Restoring data-related objects that might have been accidentally or intentionally deleted.
* Duplicating and backing up data from key points in the past.
* Query data in the past that has since been updated or deleted.

No tasks are required to enable Time Travel. **By default, it’s enabled with a 1-day retention period**. However, **we can increase to 90 days** if we have (at least) the Snowflake Enterprise Edition. **It requires additional storage which will be reflected in your monthly storage charges**.

## RESTORE OBJECTS USING TIME TRAVEL

To restore objects, we use the command “**UNDROP**”. You can use it with Databases, Schemas, or Tables as we mentioned before. **If we try to restore an object with a name that already exists, Snowflake will give an error**.

DROP TABLE mytable; -> **We accidentally drop a table** UNDROP TABLE mytable; -> **We restore it using Time Travel**

In this example, we are trying to UNDROP an object with the same name, so an error will be produced.

DROP TABLE mytable; CREATE TABLE mytable (c1 varchar); UNDROP TABLE mytable; -> **It will trigger an error**

## QUERYING OVER HISTORICAL DATA

We can also query over historical data. Imagine that a table has changed a lot during the last week, and **we want to query it how it was 7 days ago**. We can also do that thanks to Time Travel. There are different ways to do it:

* **By timestamp**

SELECT \*   
FROM my\_table   
**AT(timestamp => 'Mon, 01 May 2021 08:00:00 -0700'::timestamp\_tz);**

SELECT \*   
FROM my\_table   
**BEFOFRE(timestamp => 'Mon, 01 May 2021 08:00:00 -0700'::timestamp\_tz);**

* **By offset →** In this example, we select the historical data from a table as of 15 minutes ago:

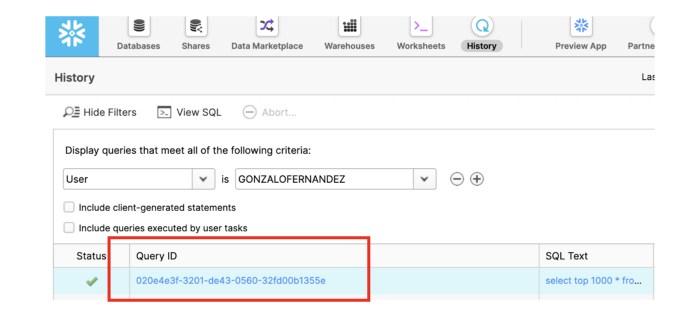
SELECT \*   
FROM my\_table   
**AT(offset => -60\*60);**

SELECT \*   
FROM my\_table   
**BEFORE(offset => -60\*60);**

* **By query statement ID** → We can see the Query Statement ID in the history of queries:

SELECT \*   
FROM my\_table   
AT(STATEMENT => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');

SELECT \*   
FROM my\_table   
BEFORE(STATEMENT => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');



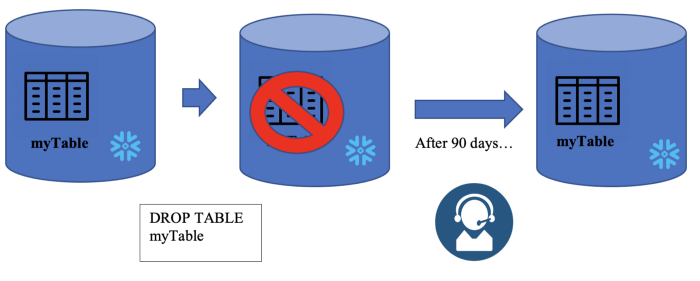
How to find the Snowflake Query Statement ID in the Snowflake UI.

# FAIL-SAFE

Fail-safe ensures **historical data is protected in the event of a system failure or other catastrophic event**, e.g. a hardware failure or security breach. It provides a **(NON-CONFIGURABLE) 7-day period** during which **historical data may be recoverable by Snowflake support**.

This is important, you cannot recover this data by yourself, **you have to ask the Snowflake support**.

Failsafe starts immediately after the Time Travel retention period ends, and it also requires **additional storage** as Time Travel.



Fail Safe in Snowflake

Imagine that we have 20 days Time Travel period. We would have all our data protected for 20 (Time Travel) + 7 (Fail Safe) days. If the Time Travel period is 90 days, the data would be protected for 97 days.

Only permanent tables have Fail-Safe period as we saw in this table.

**Why Fail-safe?**

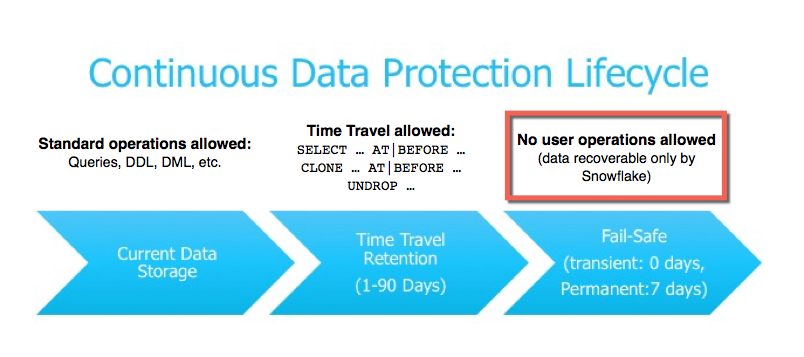
Although data is backed up at regular intervals, there can be situations between the two back-ups during which the data has been changed accidentally or deleted due to some mistake. So the fail-safe provides a quick and efficient way to recover such data with minimal cost and minimal effort.

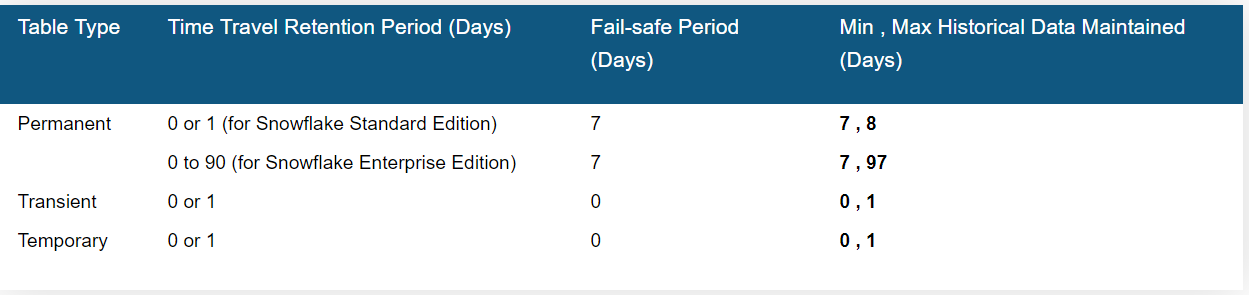
**Fail Safe Key points**

* Protection of historical data in case of disaster/ **catastrophic event**, e.g. a hardware failure or security breach
* It’s Non configurable 7 day period for permanent tables
* Period starts immediately after Timetravel ends
* No user Interaction & recoverable only by snowflake
* Contributes to storage cost

It is for use *only* by Snowflake to recover data that may have been lost or damaged due to extreme operational failures.

Fail-safe is a data recovery service and is intended only for use when all other recovery options have been attempted





* Snowflake provides similar capabilities traditional databases
* Since Snowflake has been designed for the cloud from the ground up, it has small configurations that allow control over how data is managed in a database or a table
* How temporary data is maintained and destroyed when not required.
* These capabilities are required when designing an ETL system or structuring data according to a data model.

We can check the fail-safe information for any table using the following query. Basically, it will show the amount of memory a table is occupying in the fail-safe region. (ACCOUNTADMIN has the privilege to see this information)

SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.TABLE\_STORAGE\_METRICS;

--WHERE TABLE\_NAME = 'customer';

--Convert the result into GBs

SELECT TABLE\_NAME,

ACTIVE\_BYTES / (1024\*1024\*1024) AS STORAGE\_USED,

TIME\_TRAVEL\_BYTES / (1024\*1024\*1024) AS TIME\_TRAVEL\_STORAGE,

FAILSAFE\_BYTES / (1024\*1024\*1024) AS FAILSAFE\_STORAGE

FROM SNOWFLAKE.ACCOUNT\_USAGE.TABLE\_STORAGE\_METRICS;

--WHERE TABLE\_NAME = 'customer';