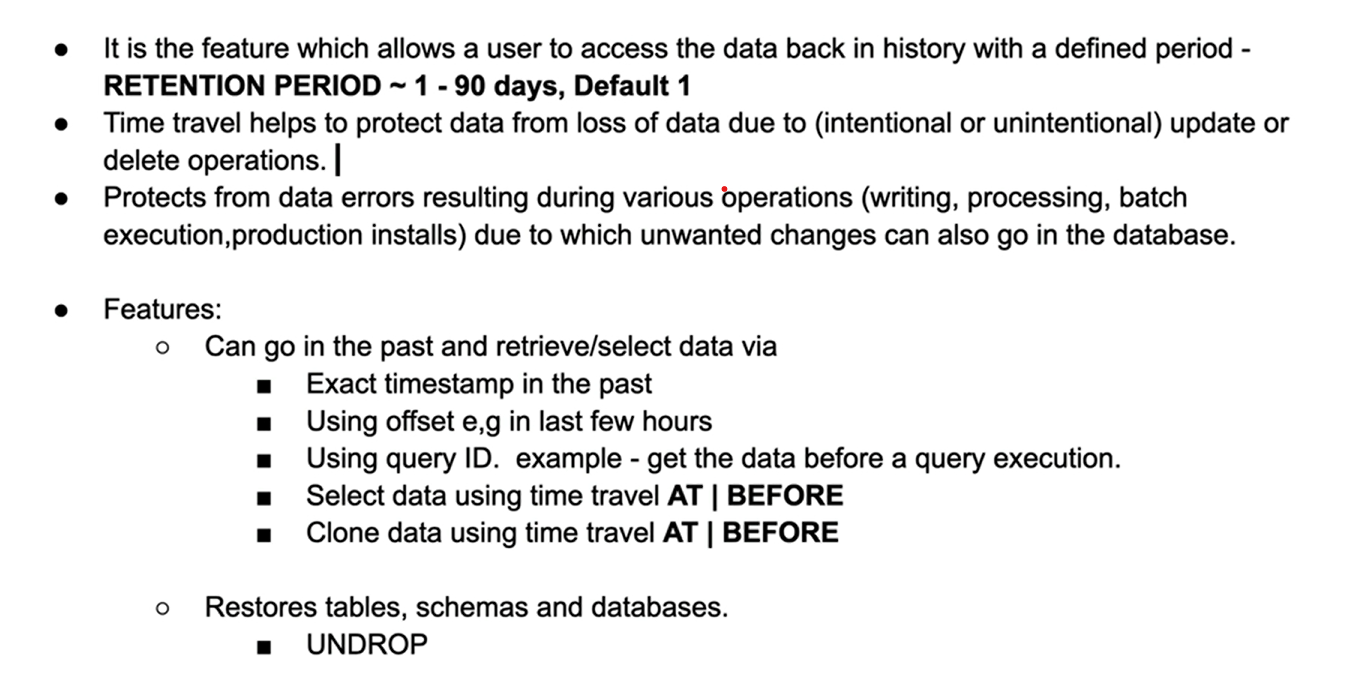
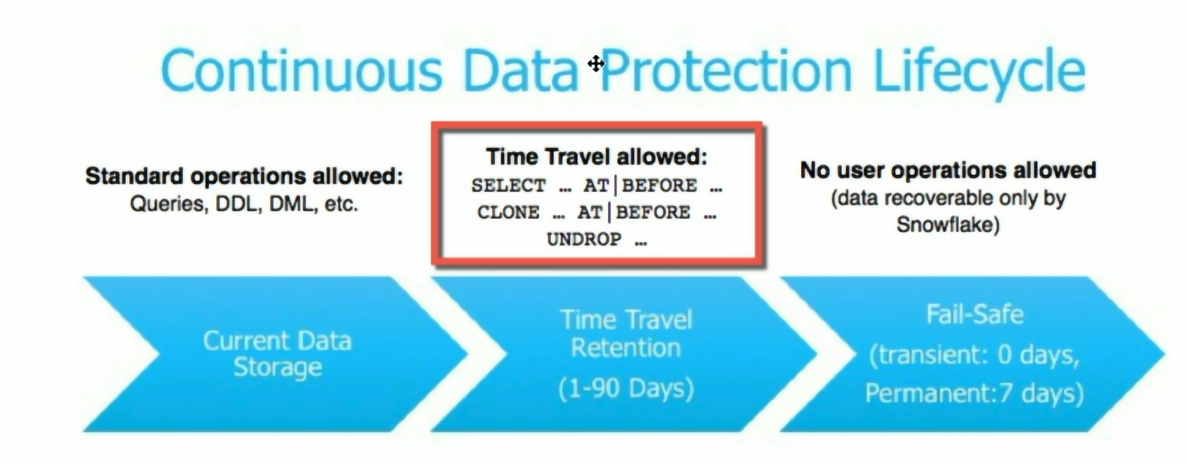
**TIME TRAVEL**

**Time Travel:**

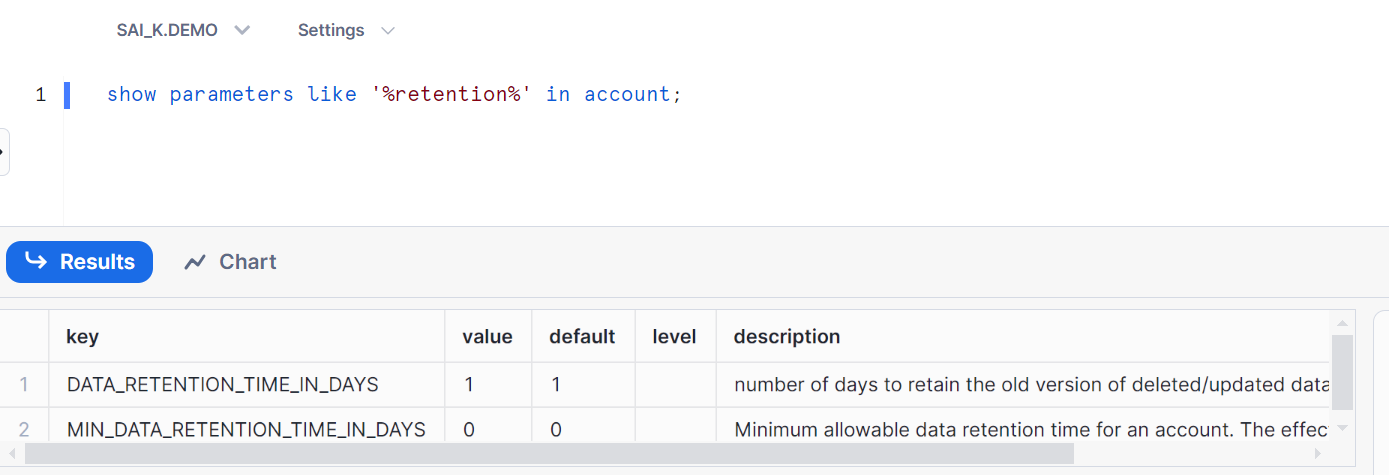


**How Time Travel works?**



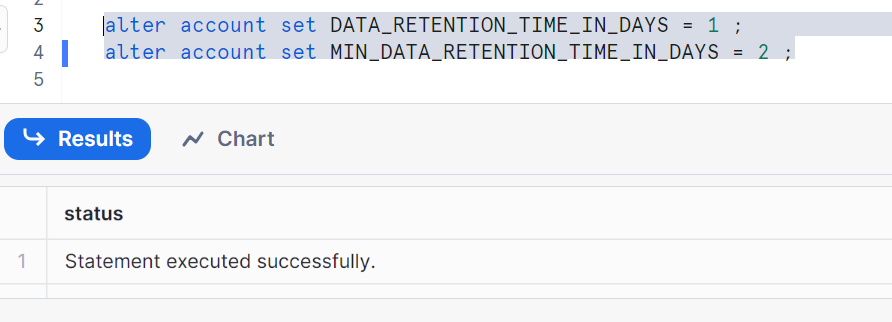
Let’s understand in practical step by step

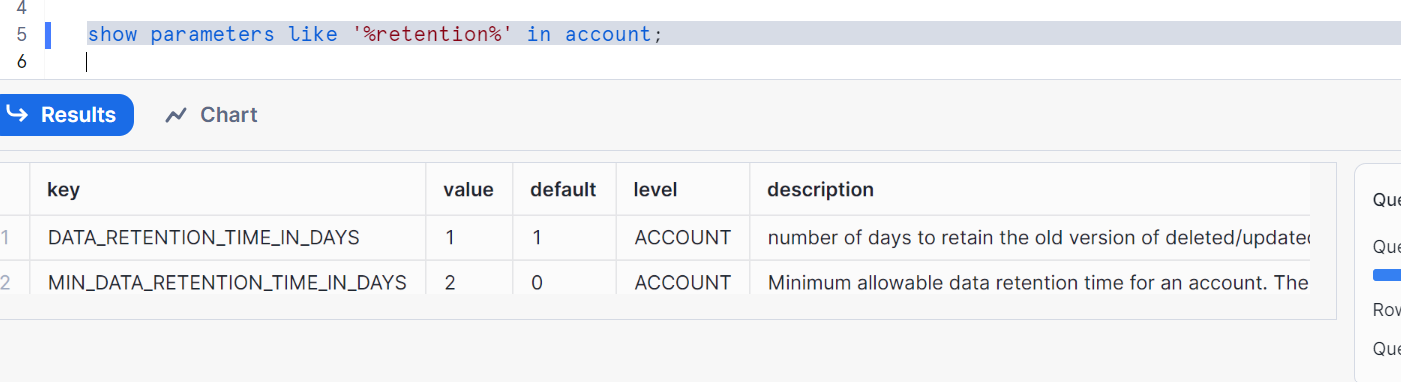
Please read the description in account level as shown below to understand better



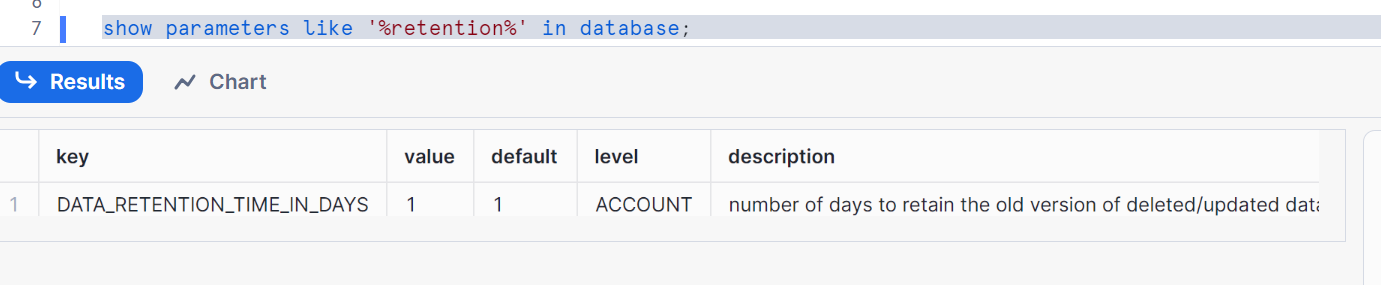
Default is 1 and 0, Value can be 1 to 90 days

We can alter as well



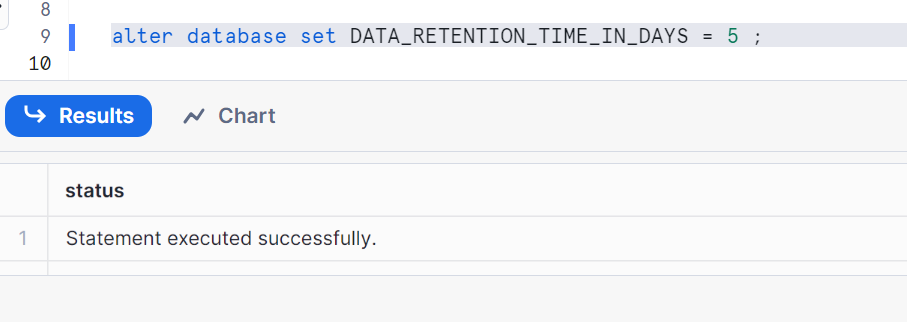


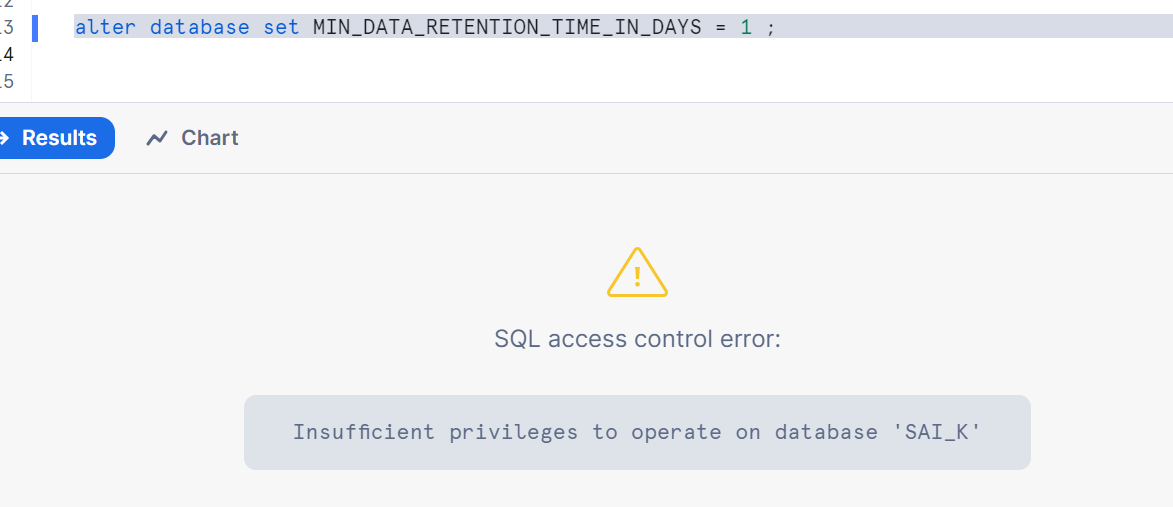
Let’s check in DB level

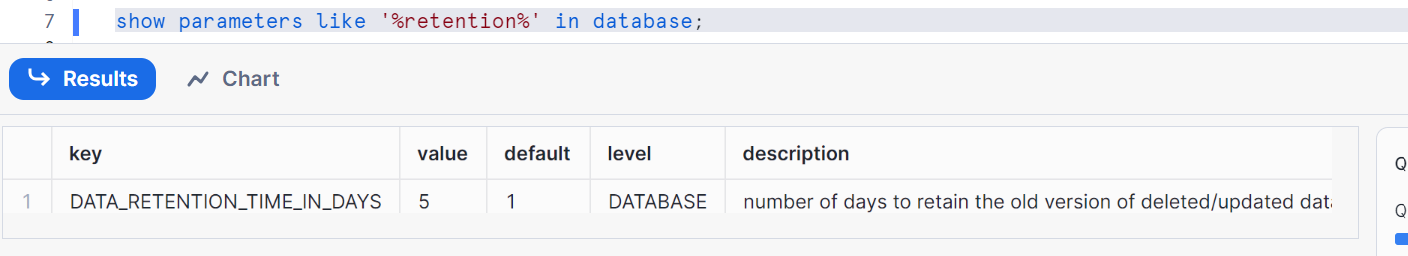


Only DATA\_RETENTION\_TIME\_IN\_DAYS is in account level implies

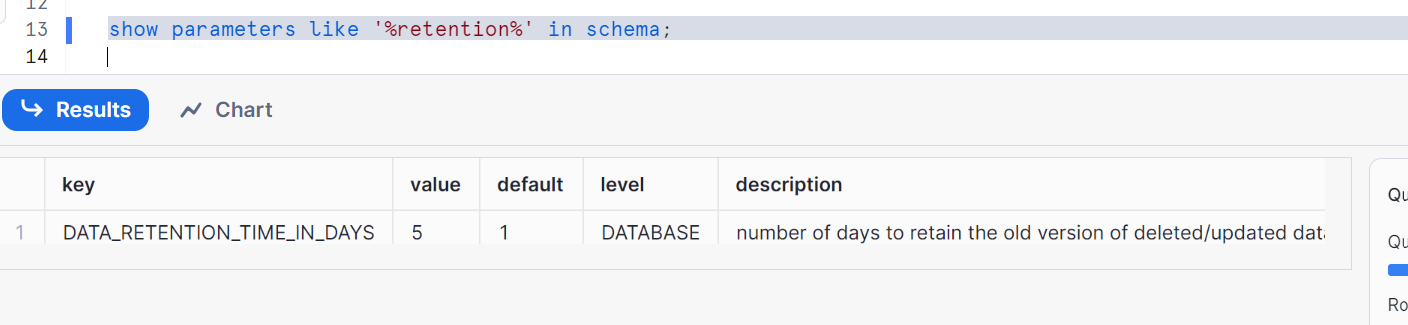
MIN\_ DATA\_RETENTION\_TIME\_IN\_DAYS can’t be altered in DB, let’s check



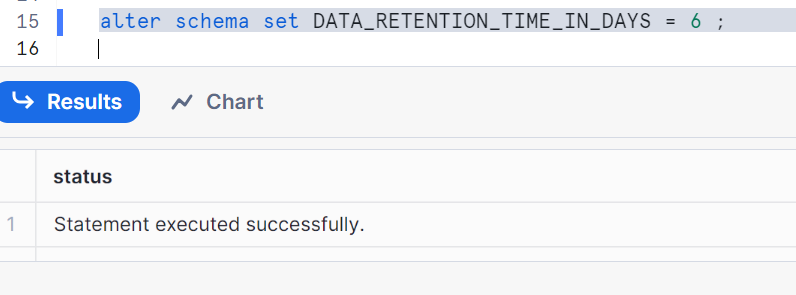


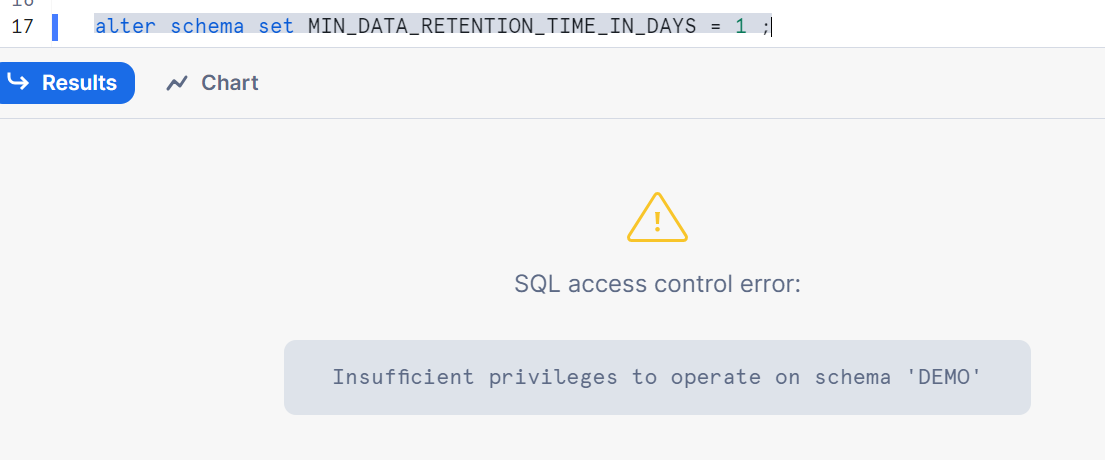


Let’s see for schemas



Here, in schema level also applies the same as that of DB



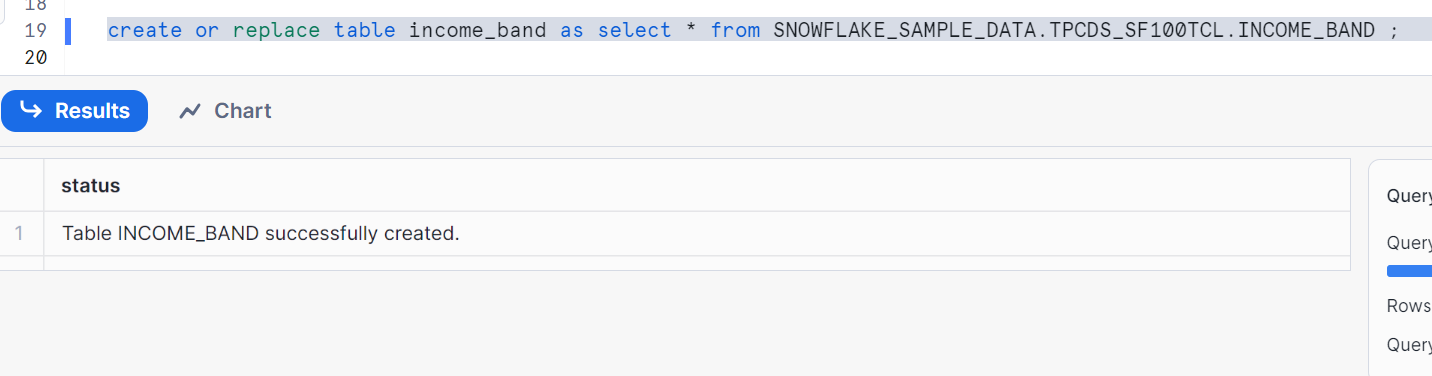


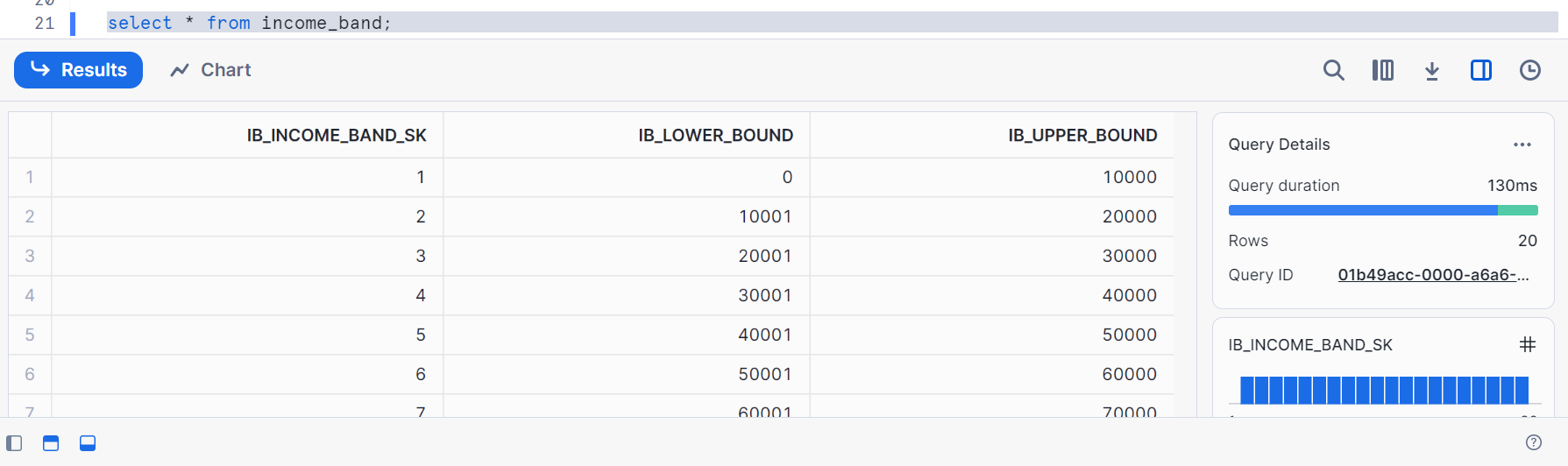
**Effective retention time** =

Max (data\_retention\_time\_in\_days, min\_data\_retention\_time\_in\_days)

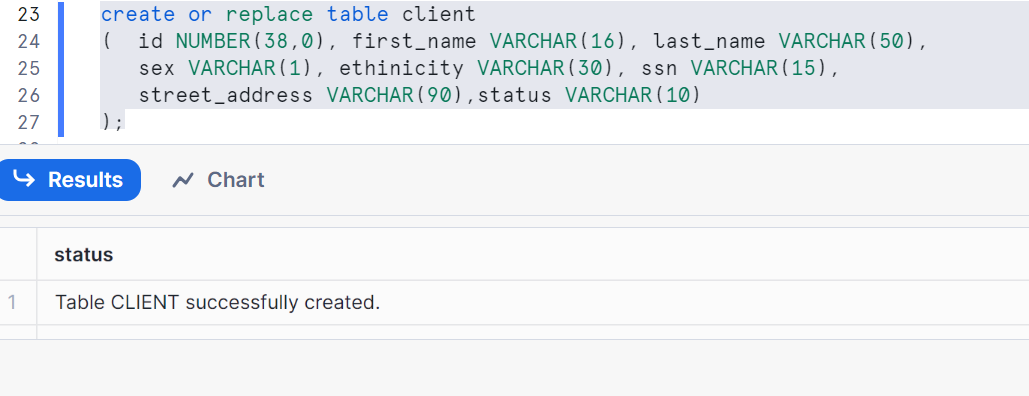
**Let’s retrieve data using ‘at’ & ‘before’:**

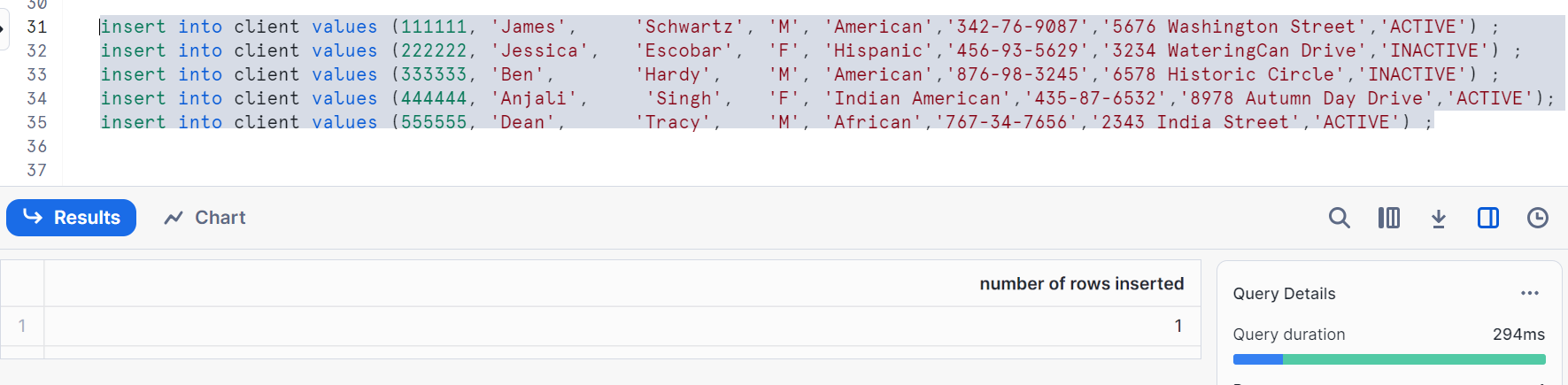
Firstly, let’s create a table

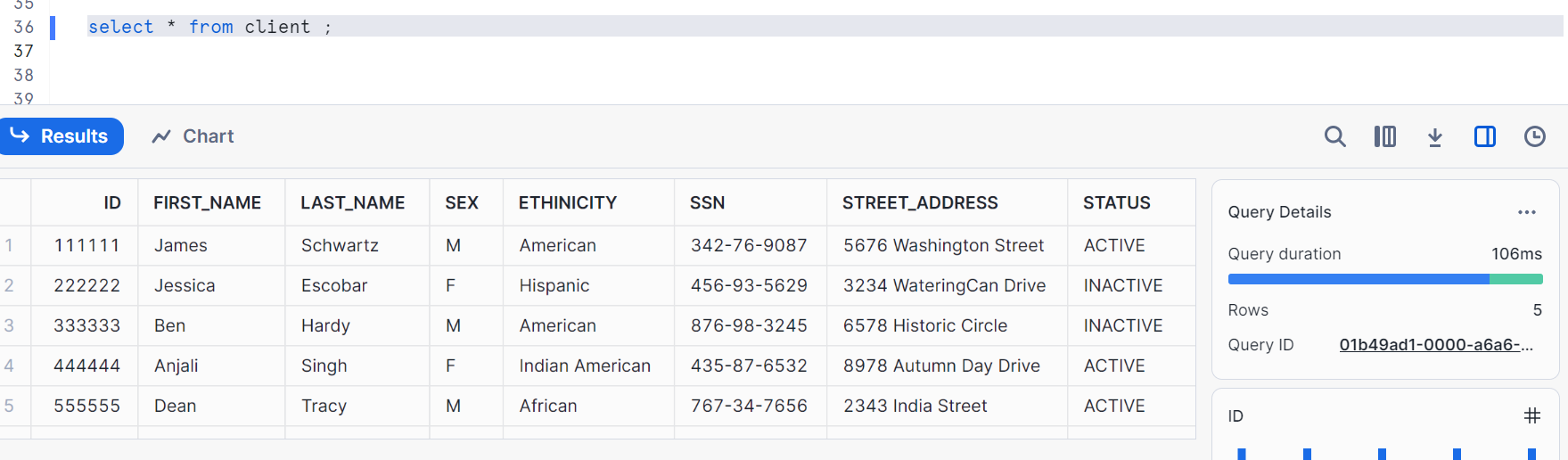
****

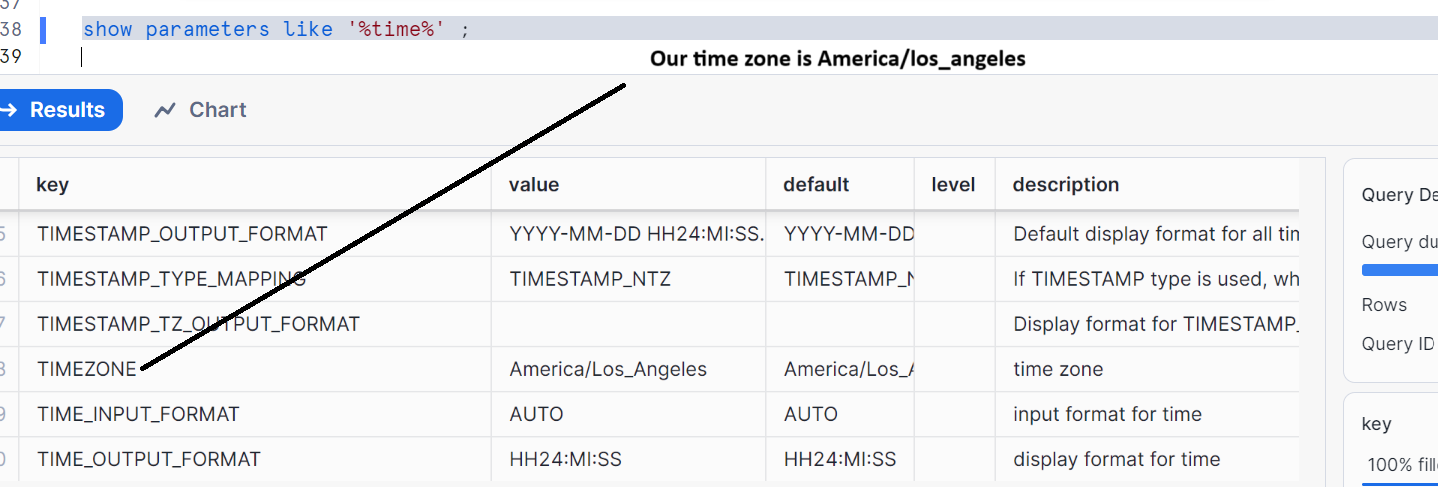
****

Create another table named client

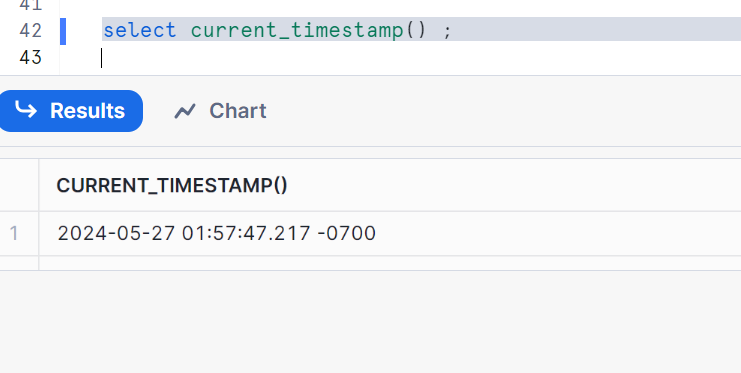
****

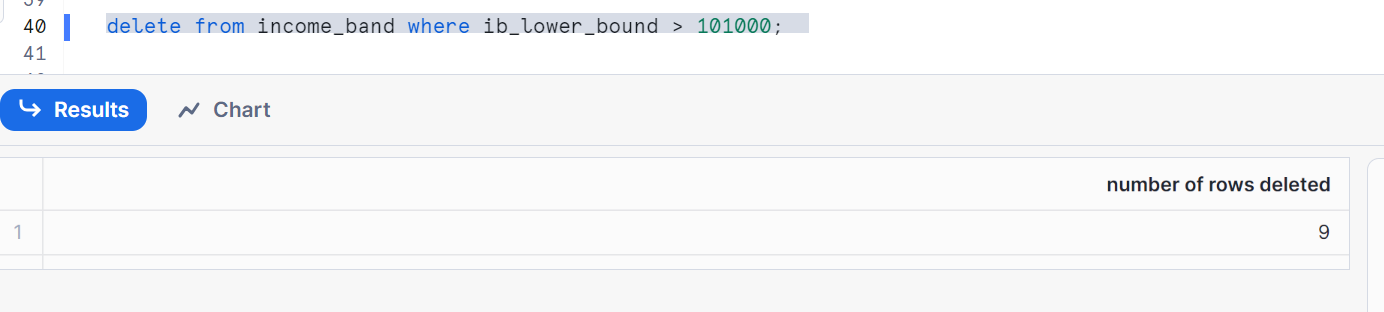
****

****



Before performing any accidental operations, first copy your current timestamp aside



Let’s perform accidental delete i.e. wanted to delete > 121000 but mistakenly we deleted ib\_lower\_bound>101000 accidentally

Copy the query id of above in your notepad

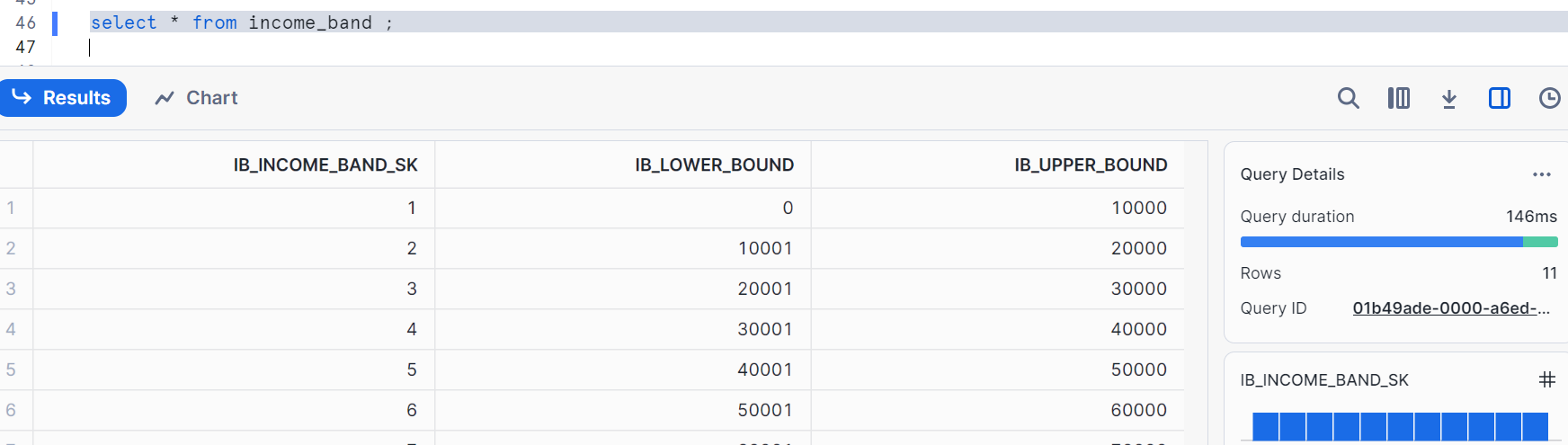


Another accidental operation i.e. accidental update, should have been only for Id=222222 i.e. we forgot to set for Id 222222 in table client



Copy the Query ID of above

Now, see the data changes in table i.e. current status

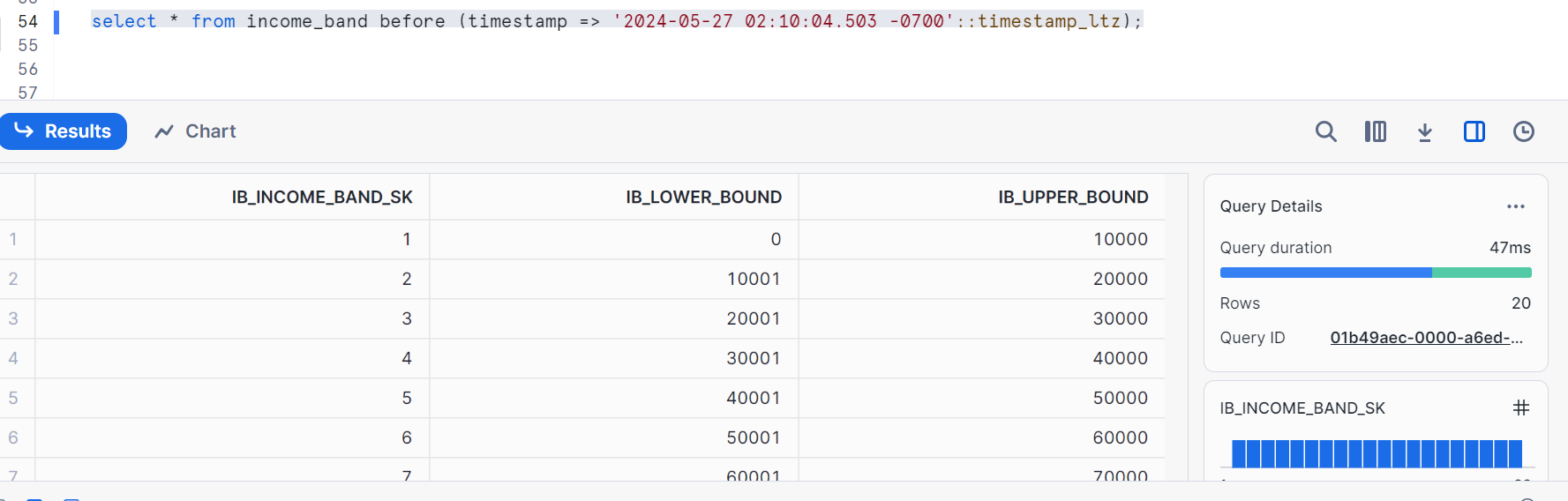




Now, we can query using time travel BEFORE | AT

Paste your timestamp 

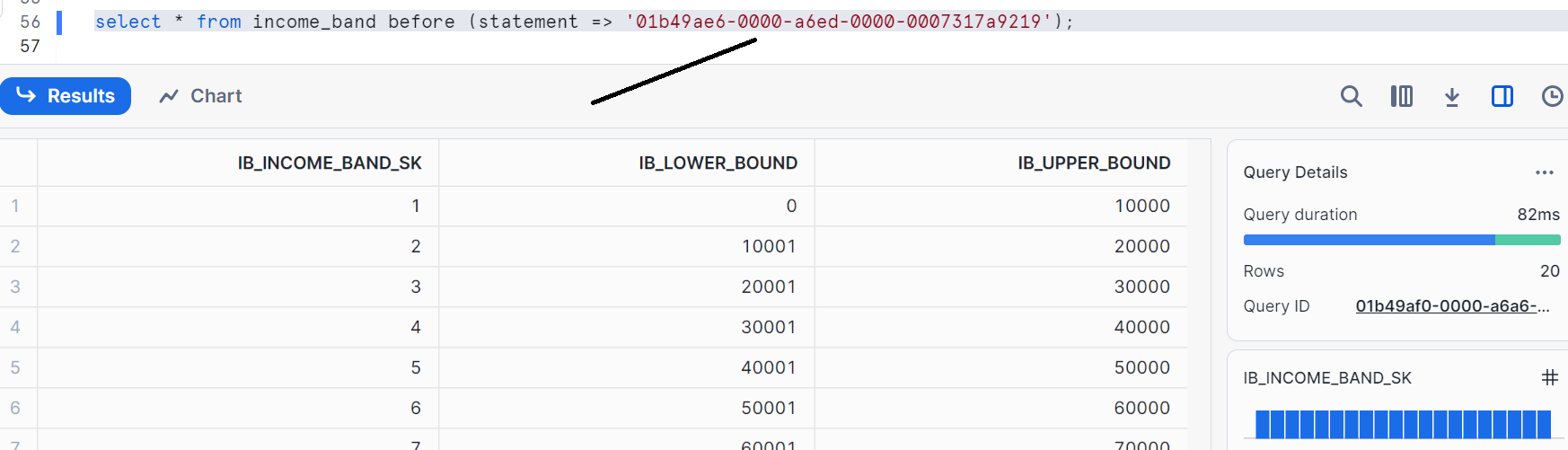
Shows 5 rows with correct status as that of without accidental operations



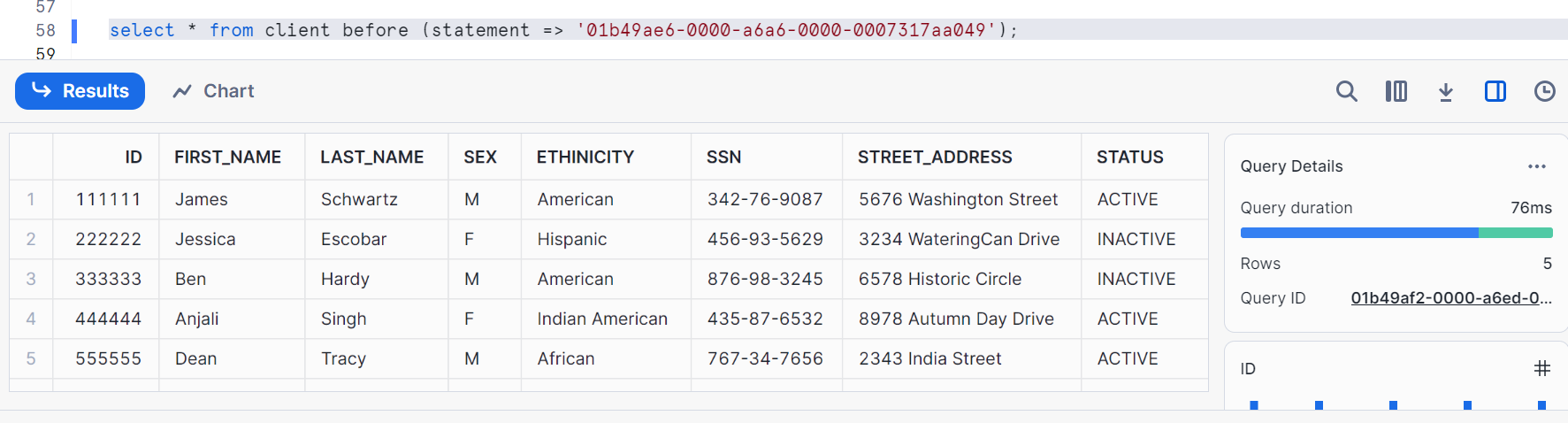
Similarly check for the client, you should get all 20 rows

Now, **using BEFORE via query ID**

Copy your query id as shown

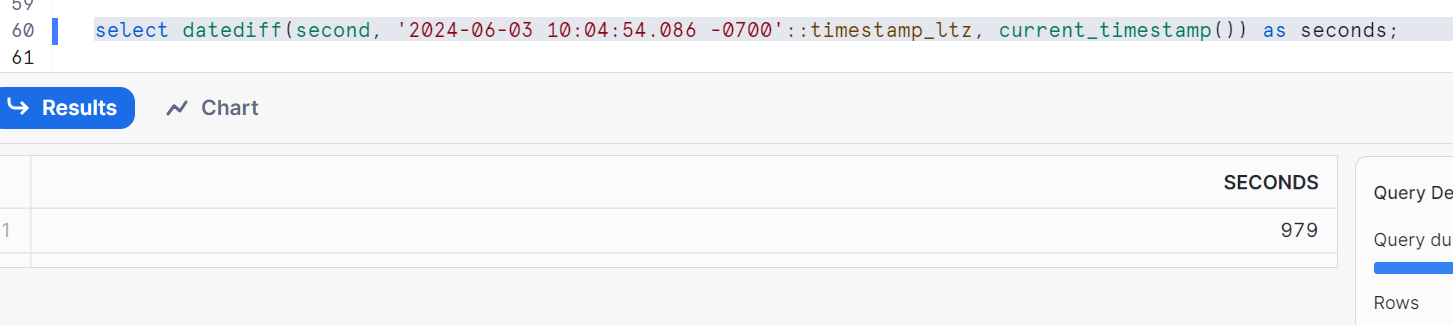


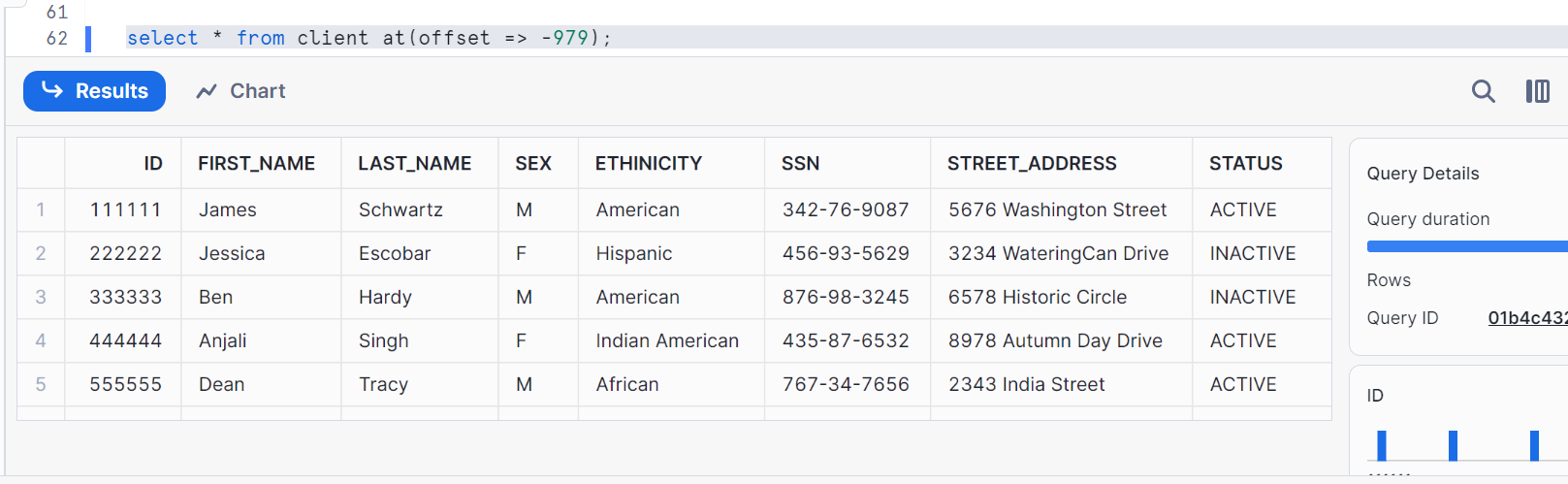
We got our 20 rows

We got our 5 rows

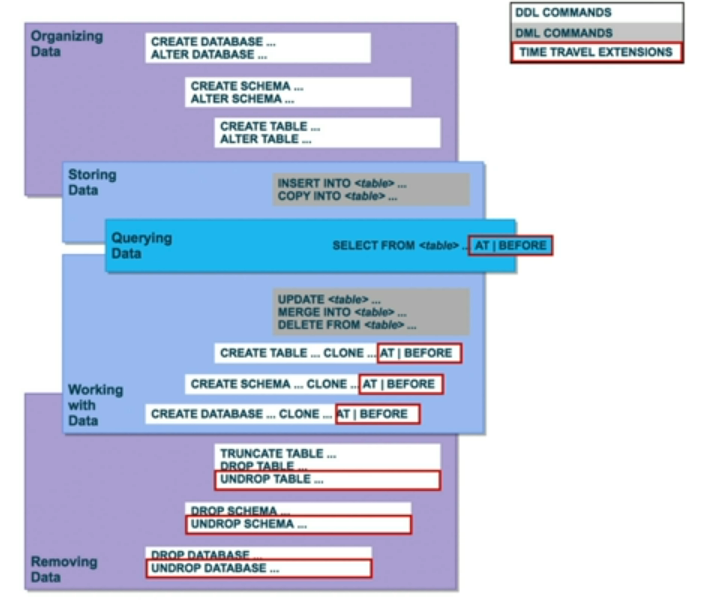
**Another method using offset**

Get the seconds first i.e. difference in seconds of current time and time when we had not accidental operations



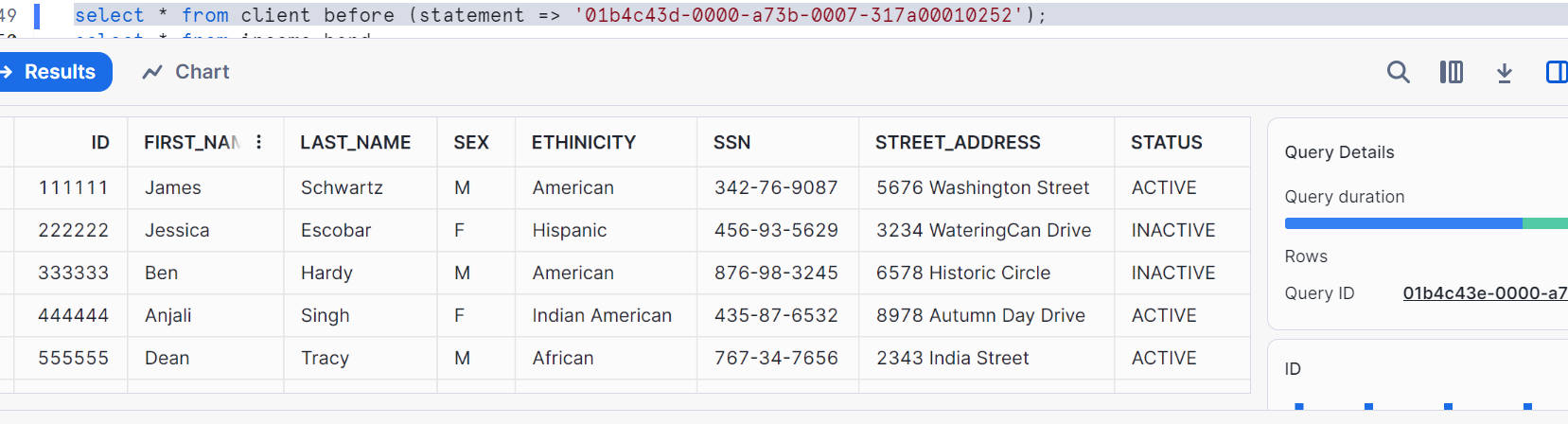


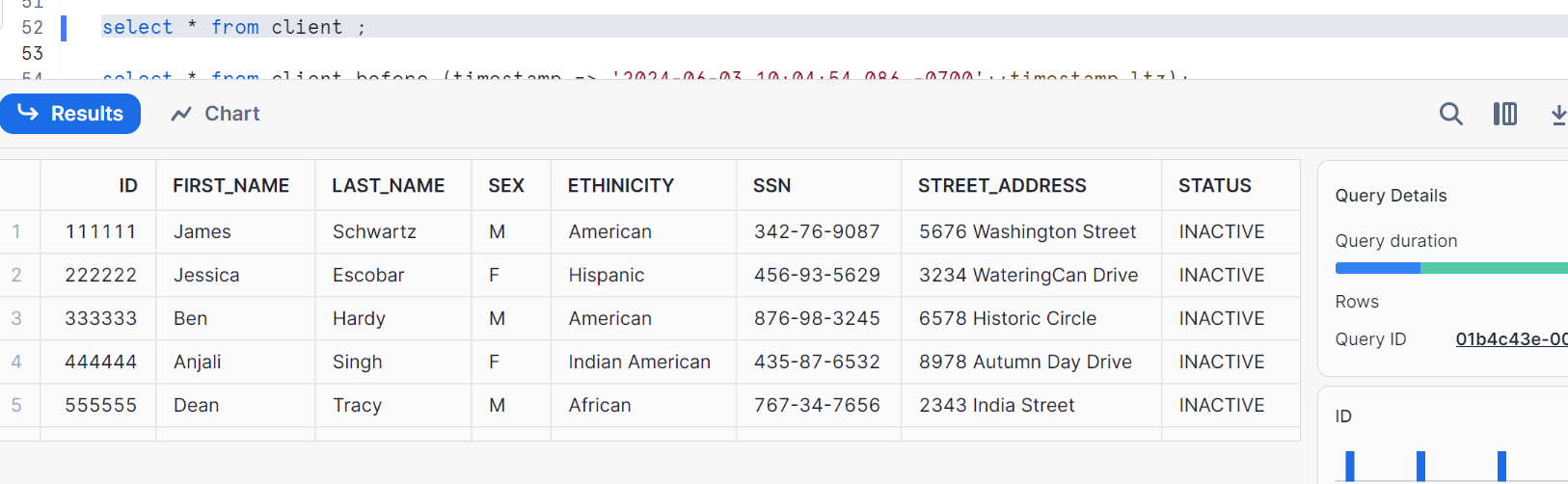
**Snowflake data lifecycle with time travel**

****

Now let’s do accidental operation on client table i.e. first accidental update (query ID 1) and then accidental delete (query ID 2)

Now, Paste your query id 1 on client table so that you have to get the data after the update operation by checking the real table





We had done delete and update implies second query Id pasted will give me the data before update operation but

Do the same for income\_band

**Data restoration (restoring your data)**

1. Direct method
2. Indirect method

**Direct data restoration:**

Let’s do practical

I had done two accidental operations on client table as shown, I copied my query Id’s

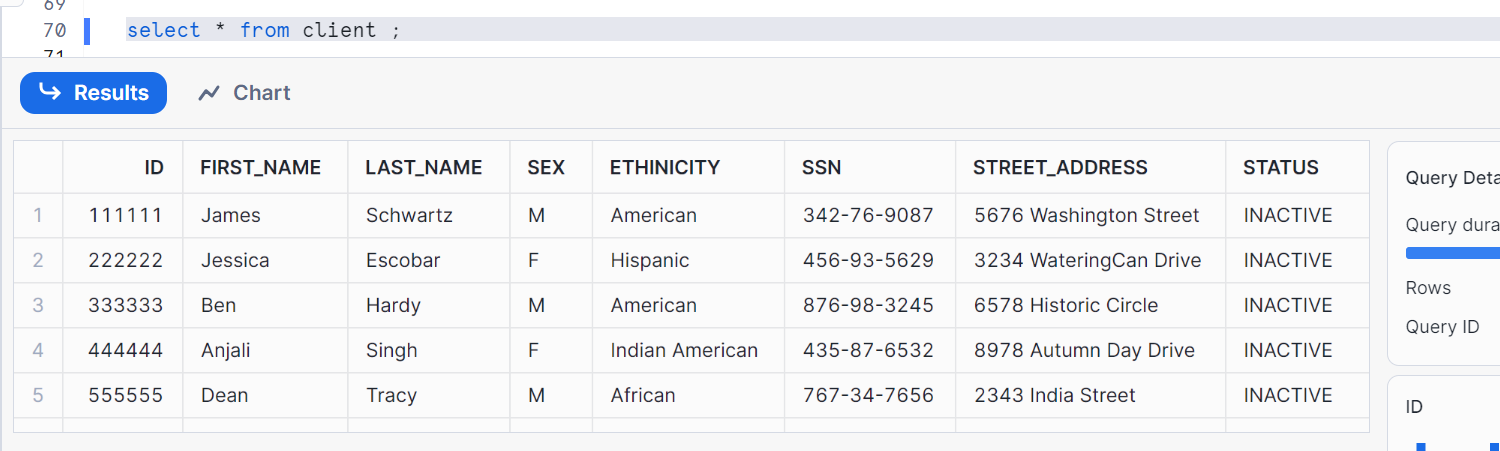


Now, I want to get the data before my delete operation i.e. I had done update operation. So, my table need to have all status as inactive

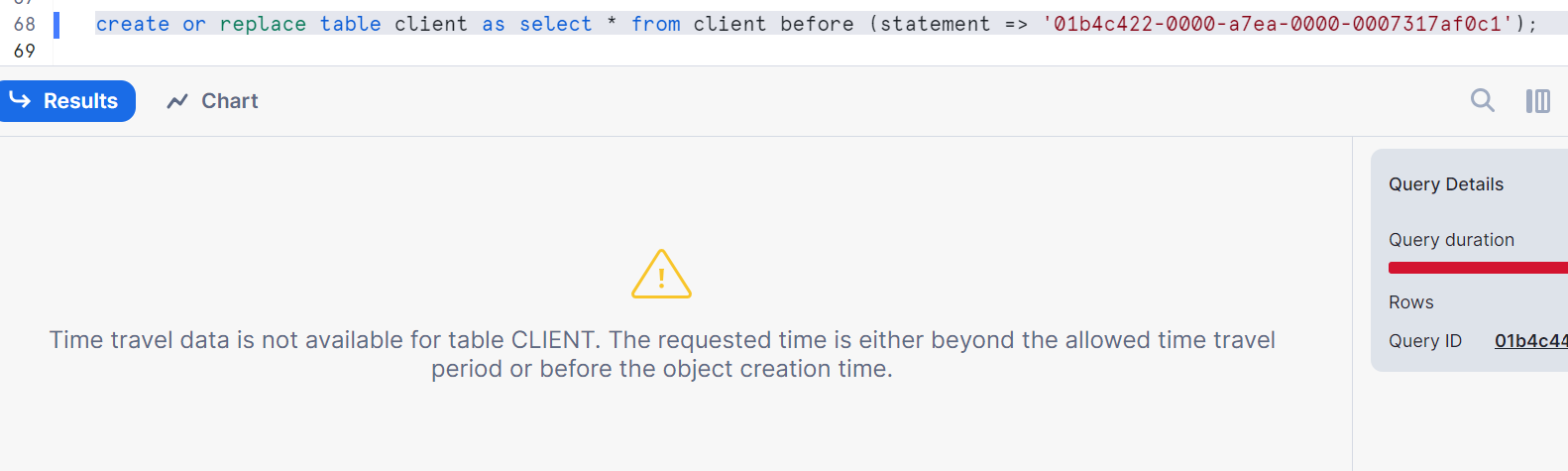
Here is the syntax

**Create or replace table client as select \* from client before (statement => 'paste your query ID of delete operation here');**

Now check the table



Now, I need to get the data before update operation i.e. my original data. Let’s perform the same



That means we cannot go back to first query ID because we have used create or replace table client for second ID as main table client is actually got recreated and it has lost time travel history.

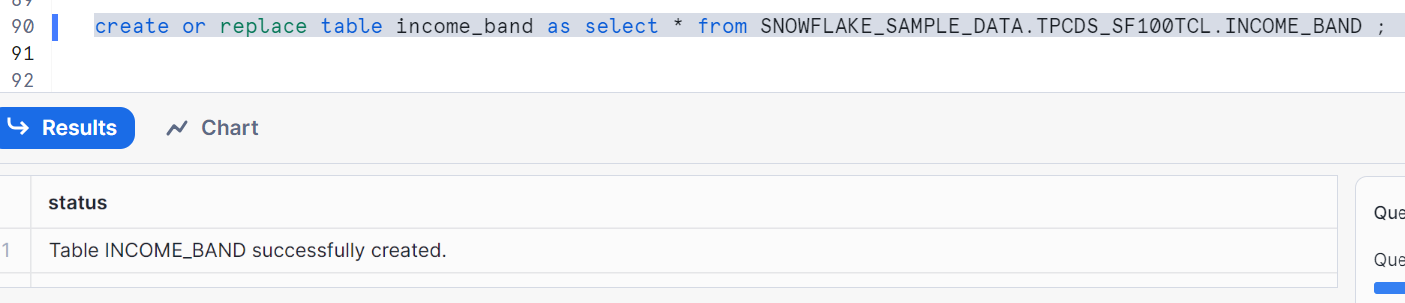
Hence, direct method of creating or replacing a table has this particular limitation

If you are using it, you have to be 100% sure what particular time frame/query ID you are using and once you do that, you will not be able to use again for another period.

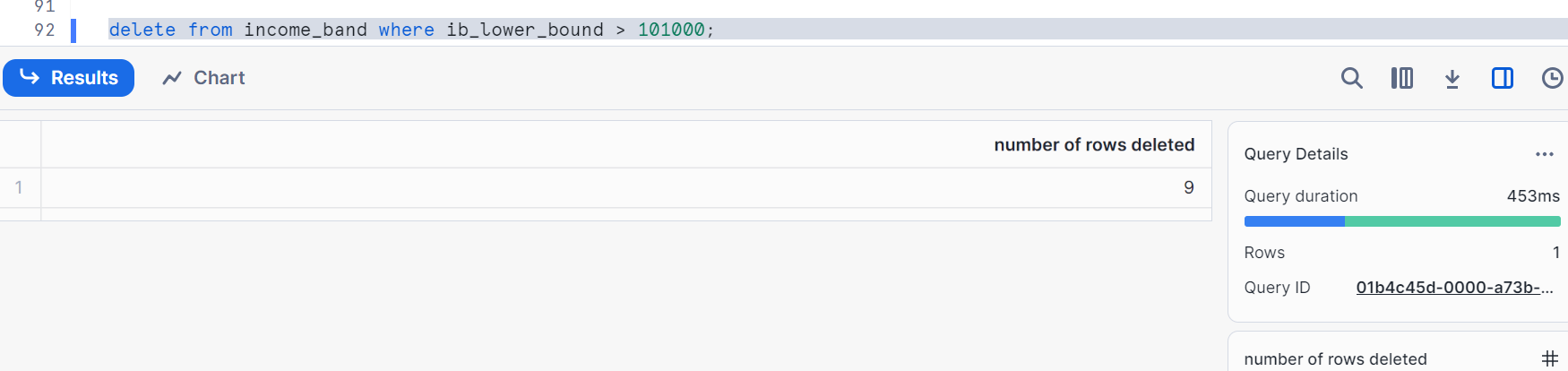
Hence to avoid this, we use indirect method

**Indirect data restoration:**

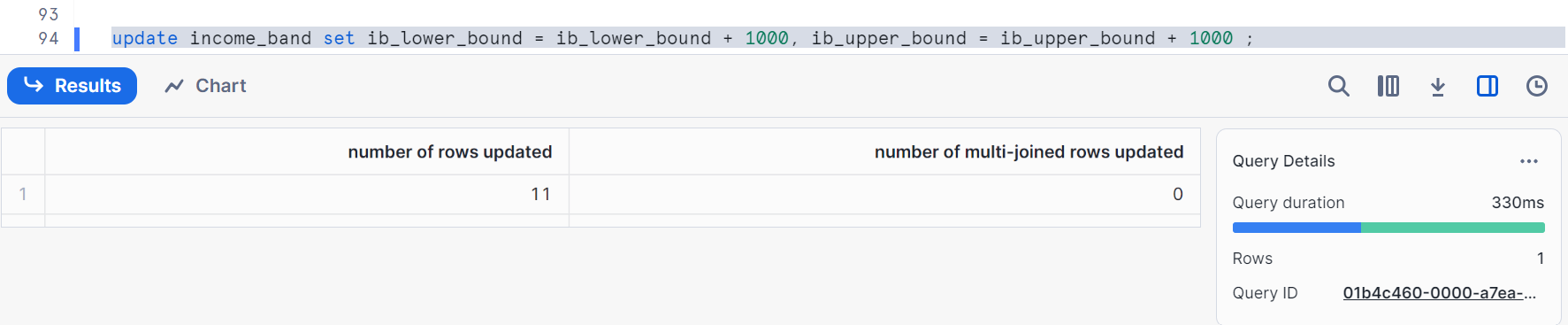
Let’s do in practical



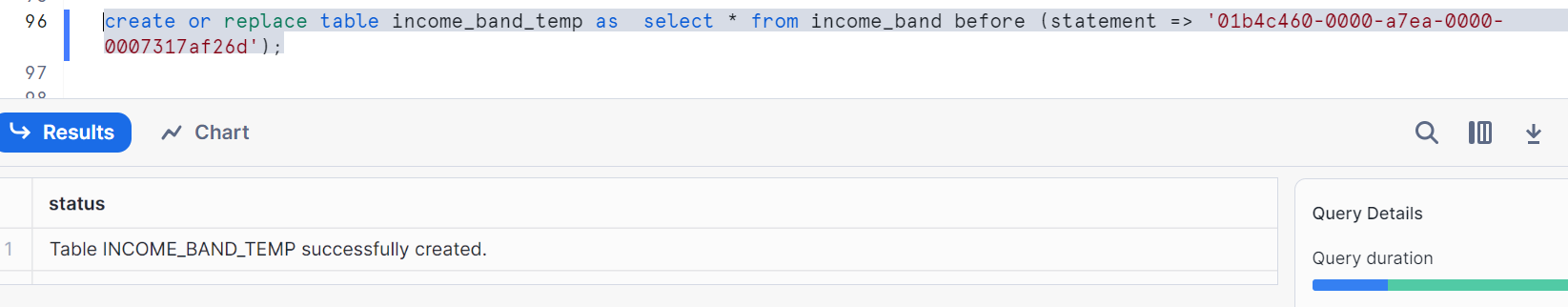
First accidental delete i.e. wanted to delete > 121000



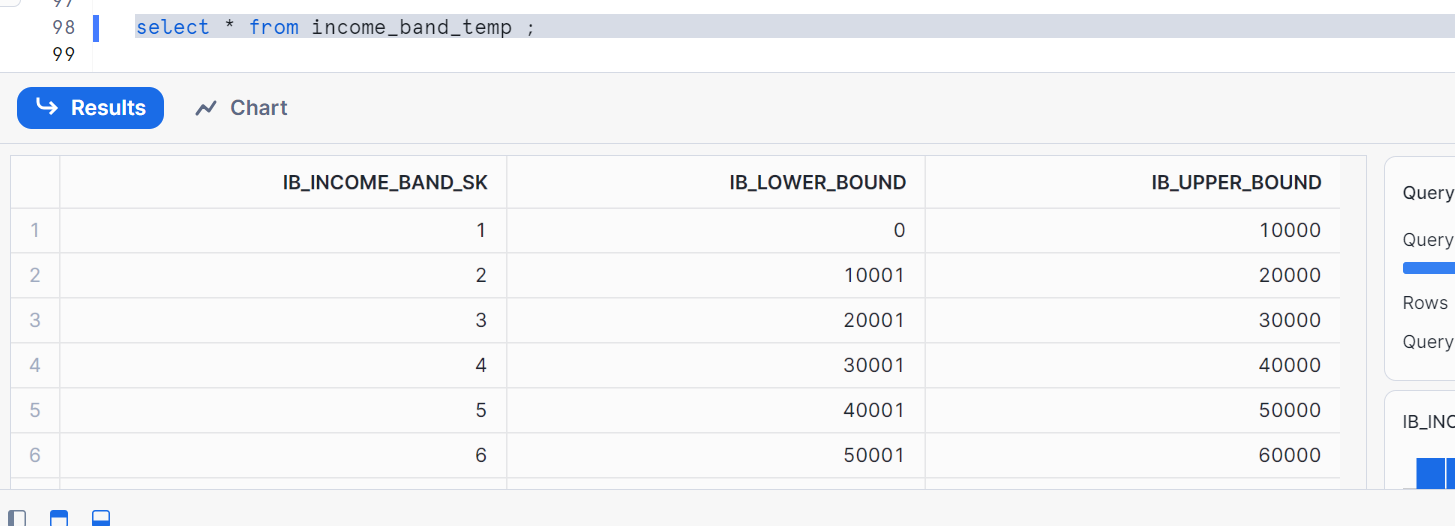
Second accidental update i.e. shift the band by 100, accidentally shifted by 1000.



Creating a temporary table and pasted second query ID as shown

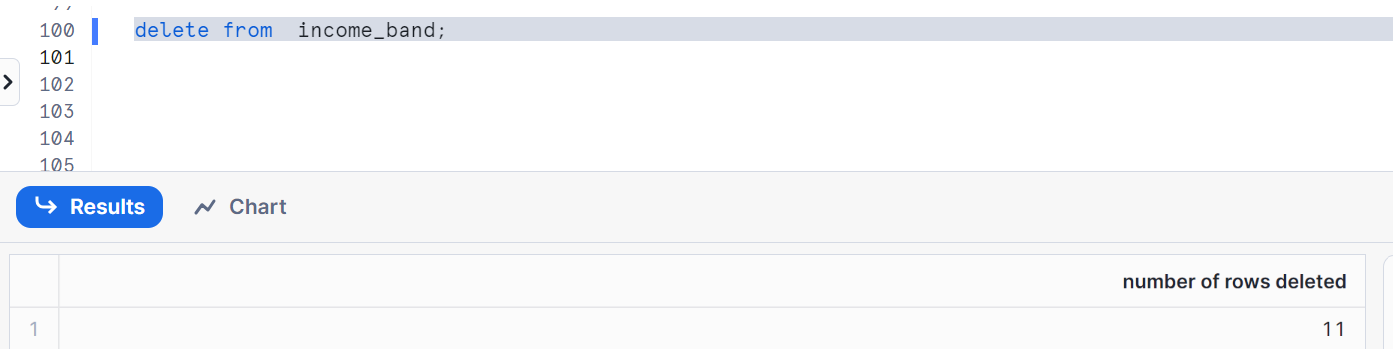


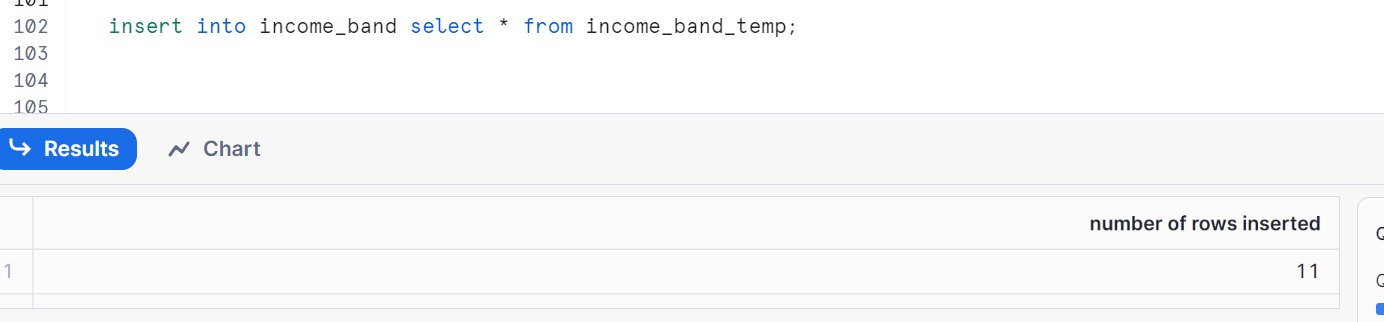
Let’s check data

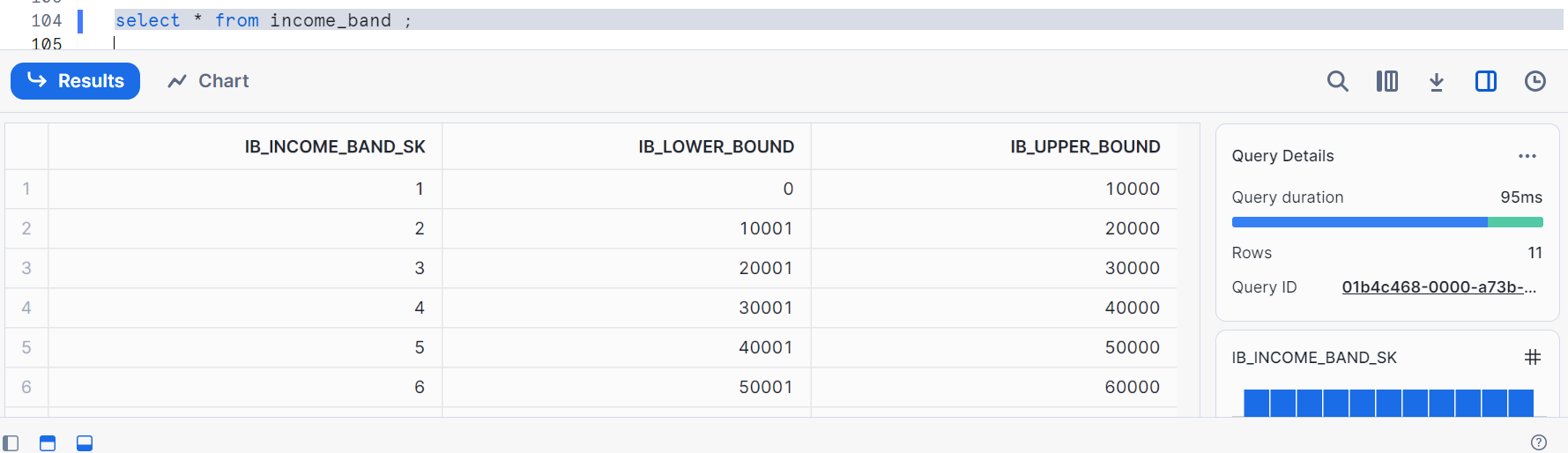


I got my records which are not increased by 1000

Now delete the main table and restore from temp table

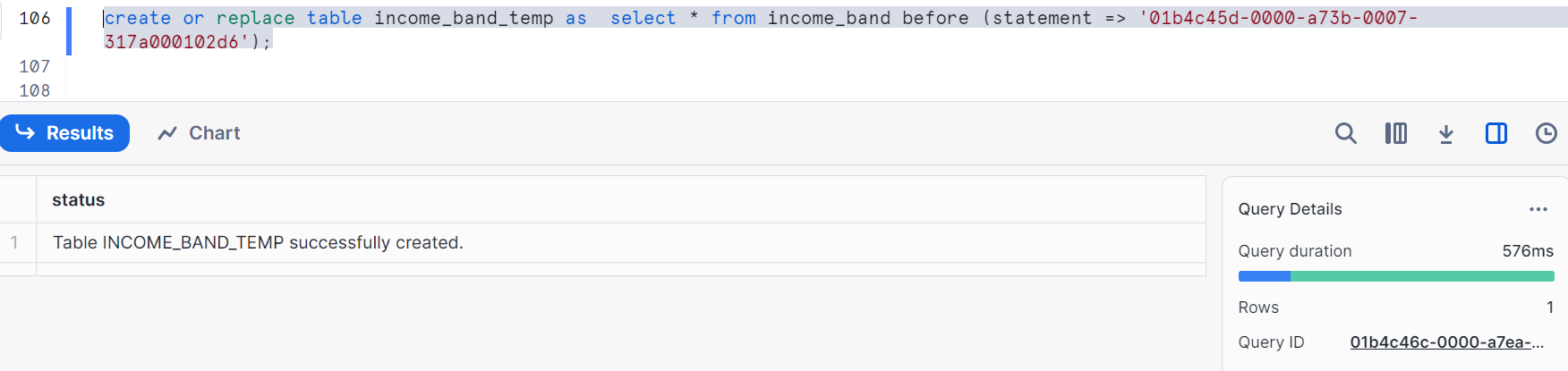




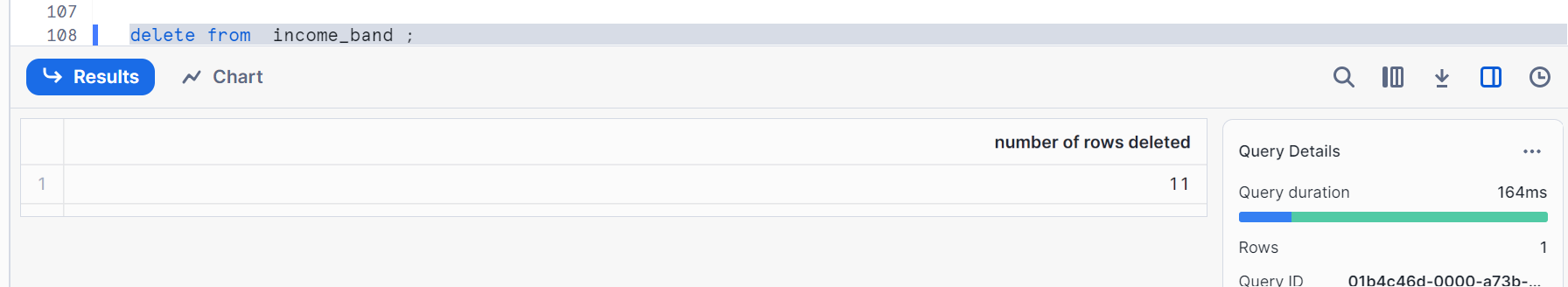


We went back one step, now we realize that we need to go back further

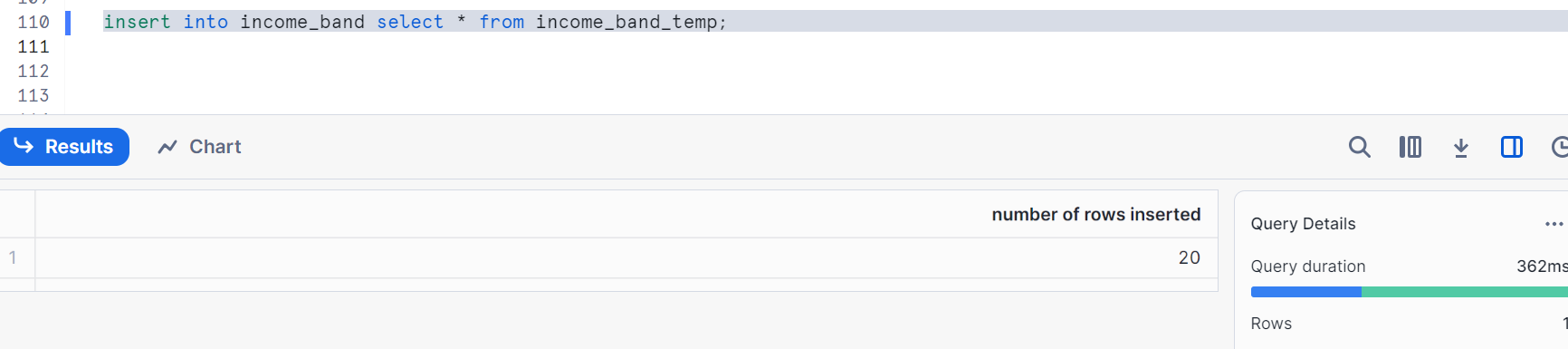
Again we can build the temp table using first query, let’s see



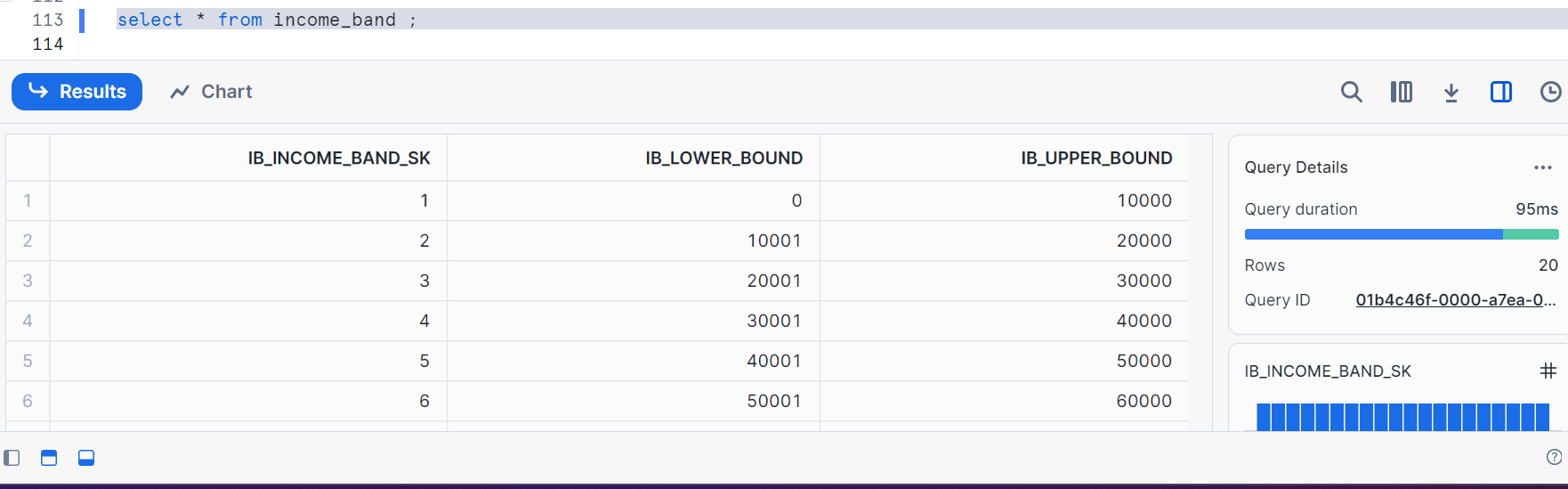
Delete the main table



Restore from temp table



Check the data

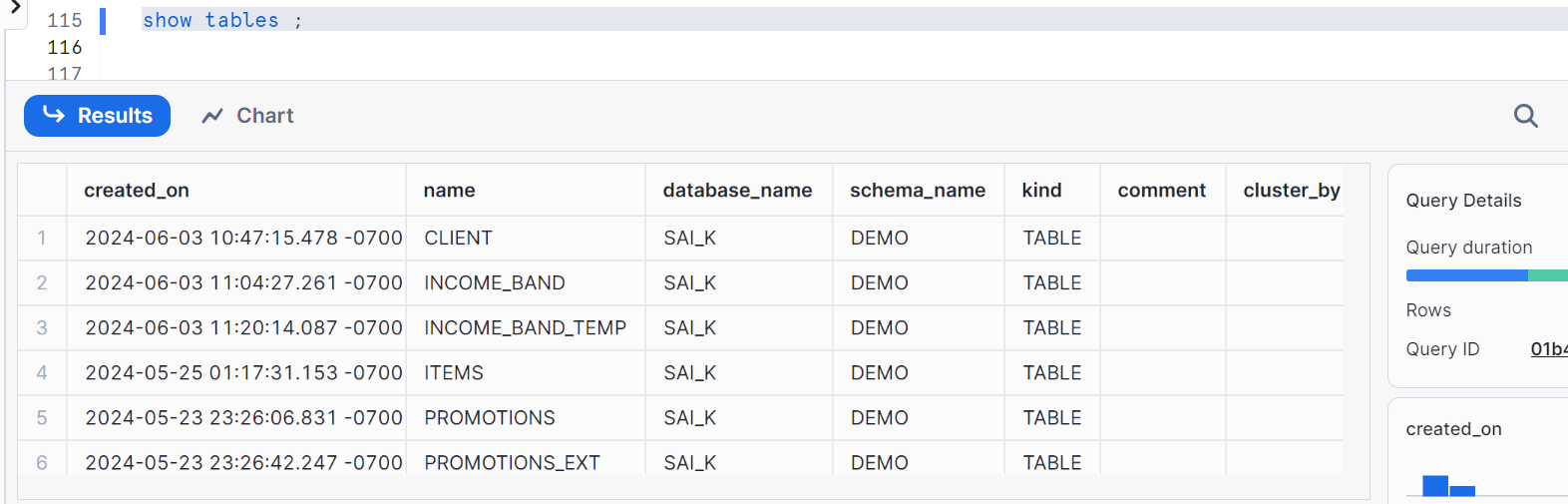


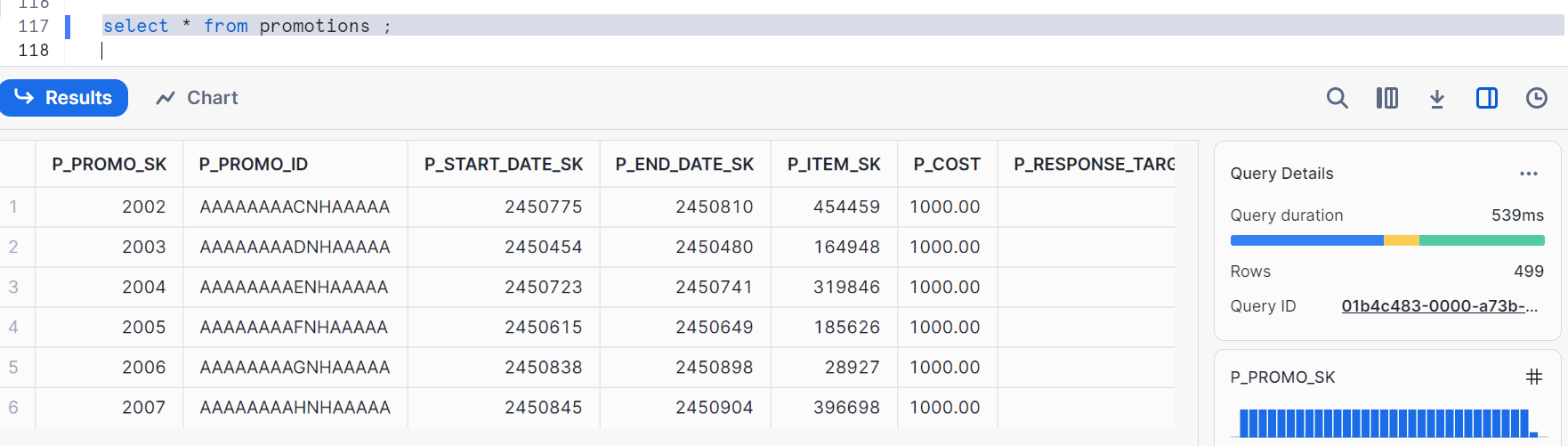
* The idea is you should not drop the table to have its data retention.
* The moment a table is dropped it loses time travel and is considered as a new object,
* Create or replace actually drops the object.

**UNDROP:**

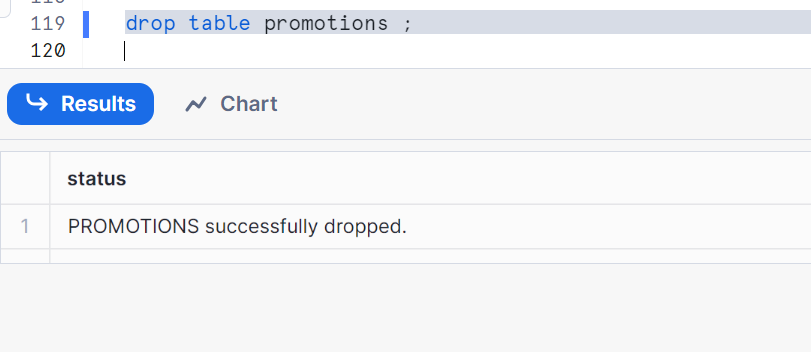
To retain our tables, schemas, database which we dropped

Let’s see in practical

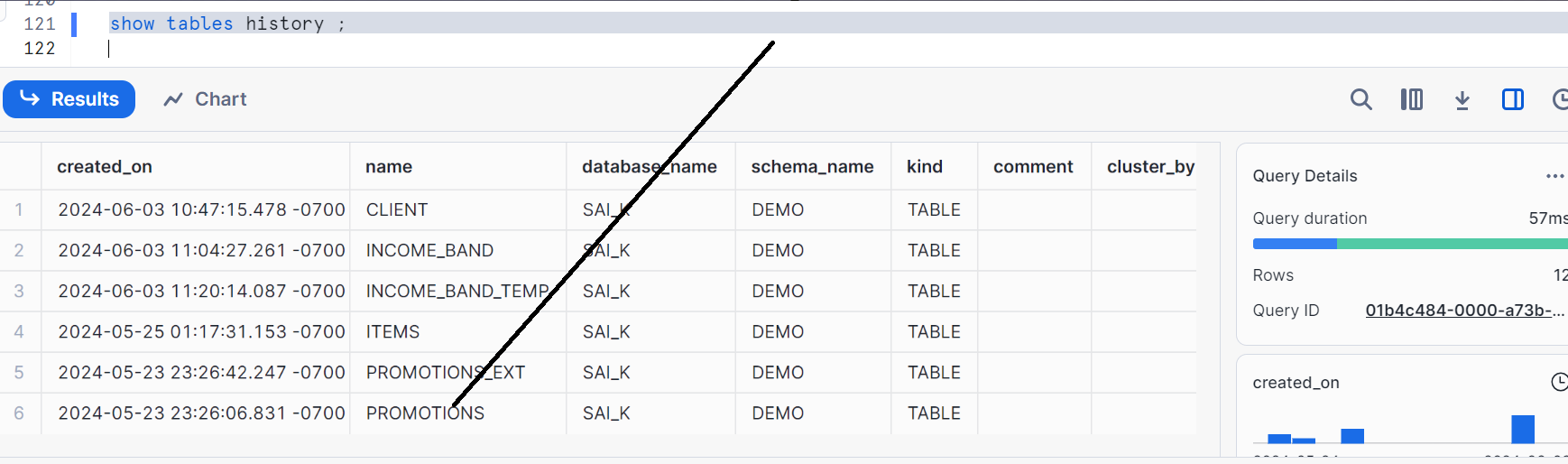




Let’s drop it

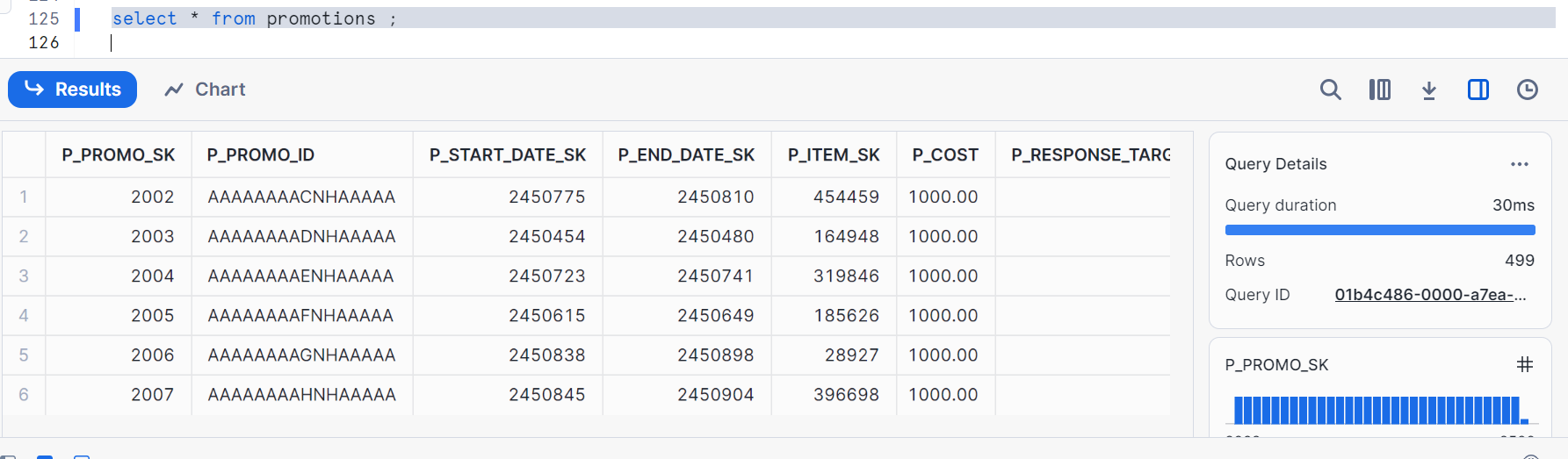


Please follow the syntax

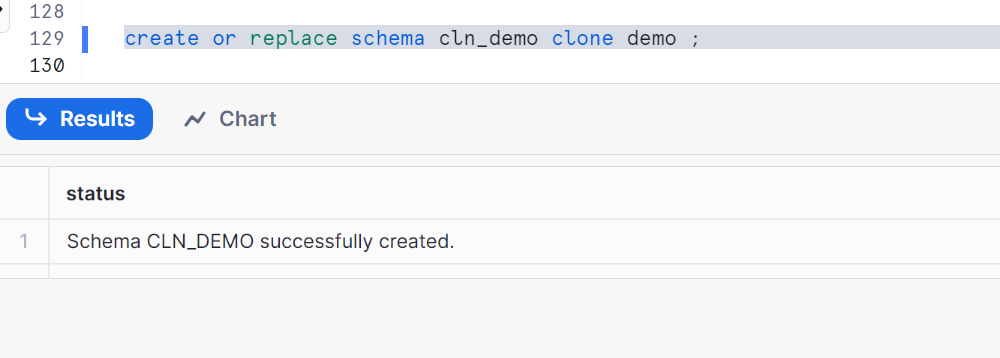


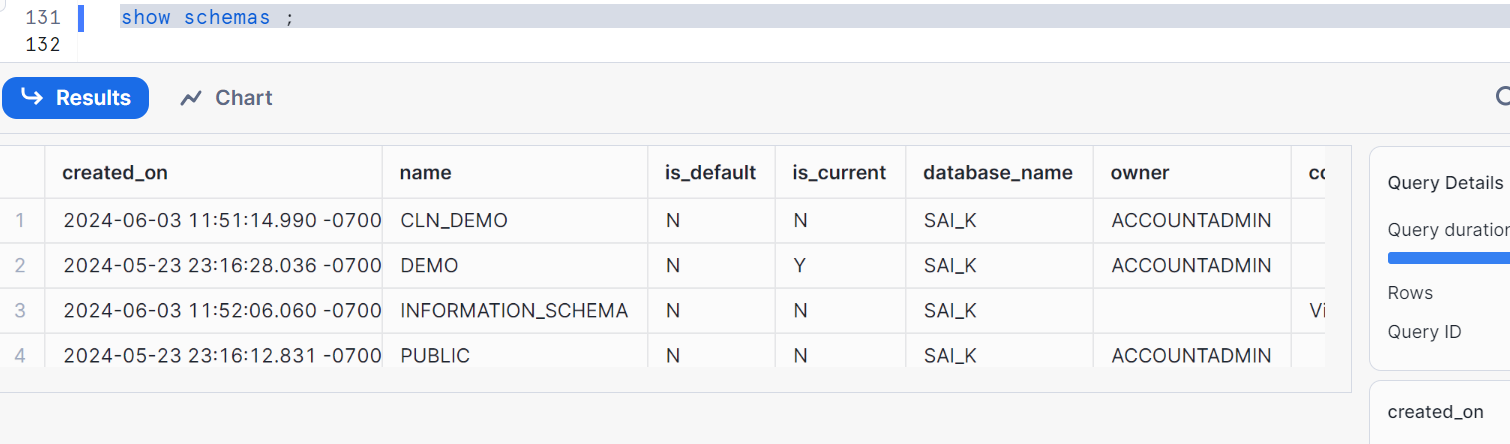
Visible, Hence can be undropped

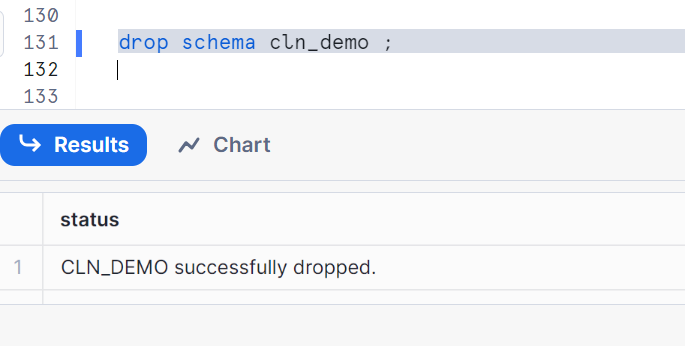


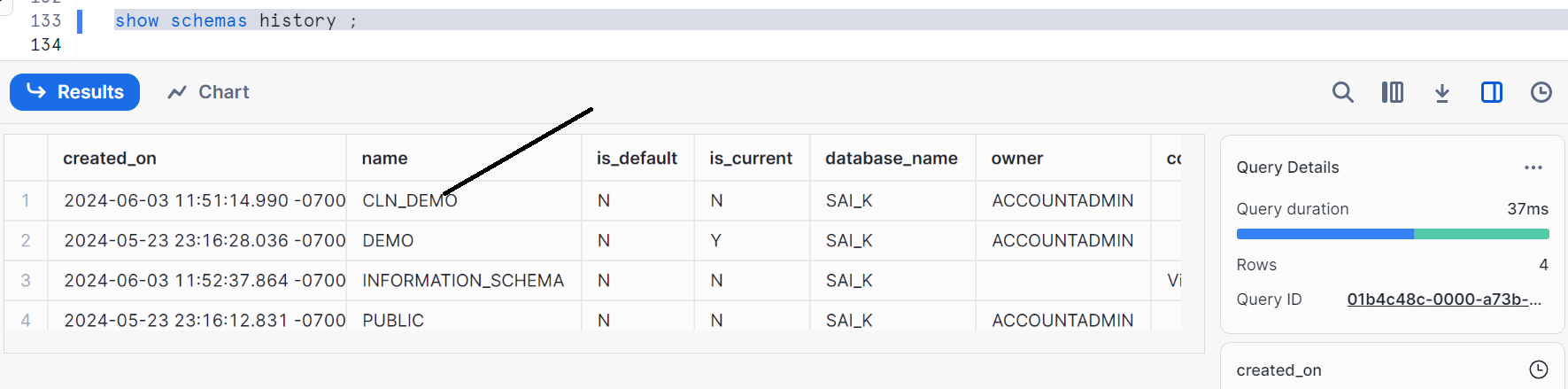


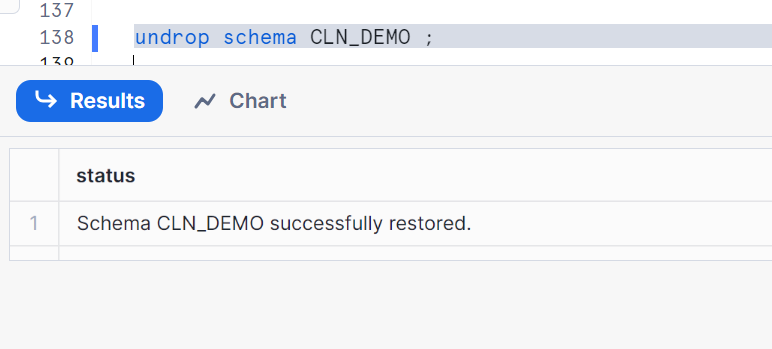
For schemas

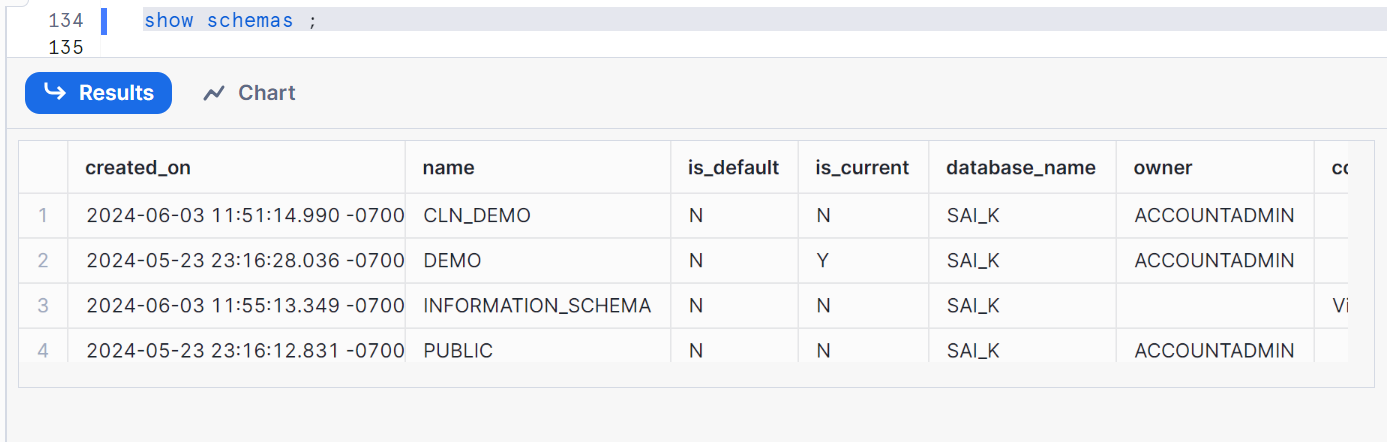






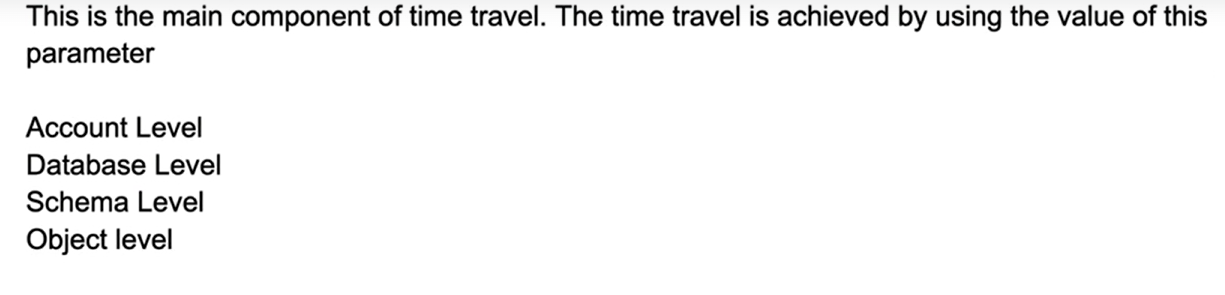


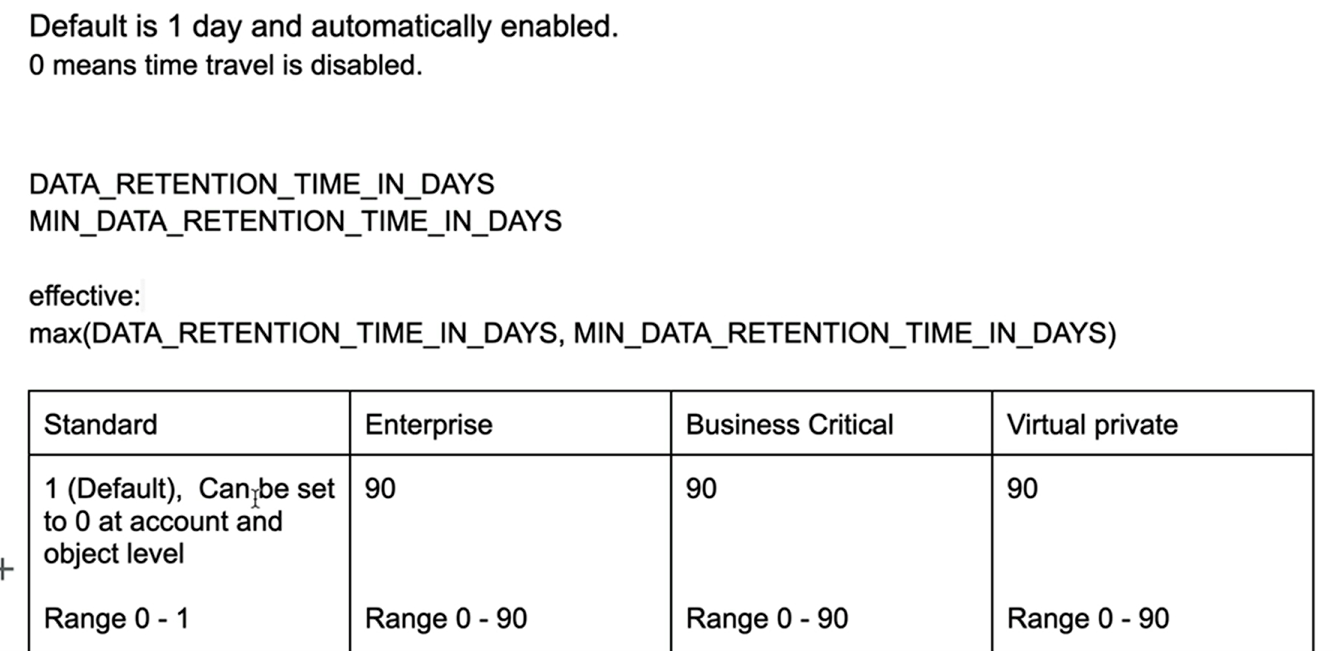




Similarly for databases as well

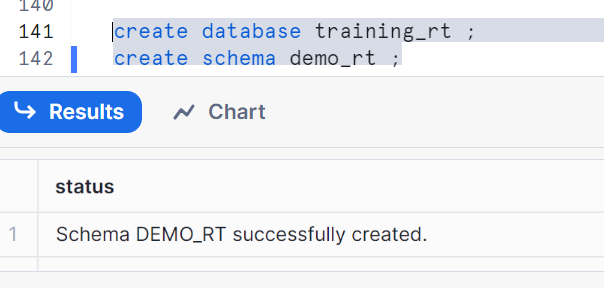
**Retention time:**

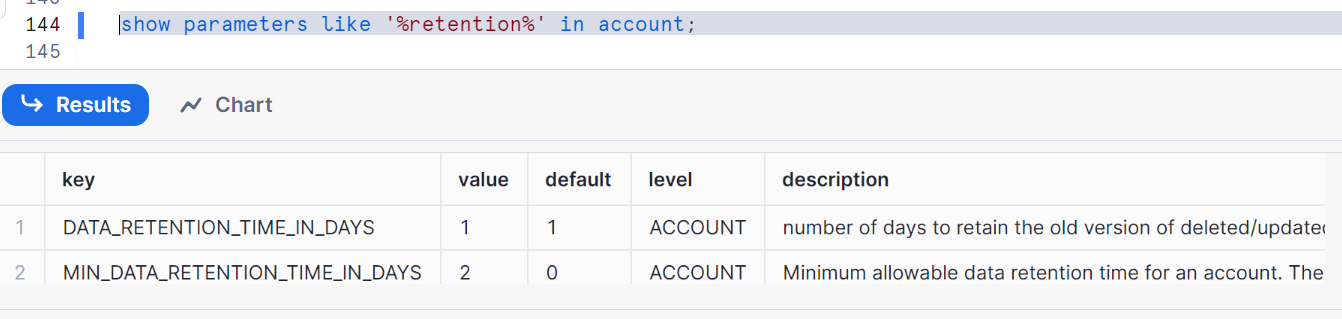
****

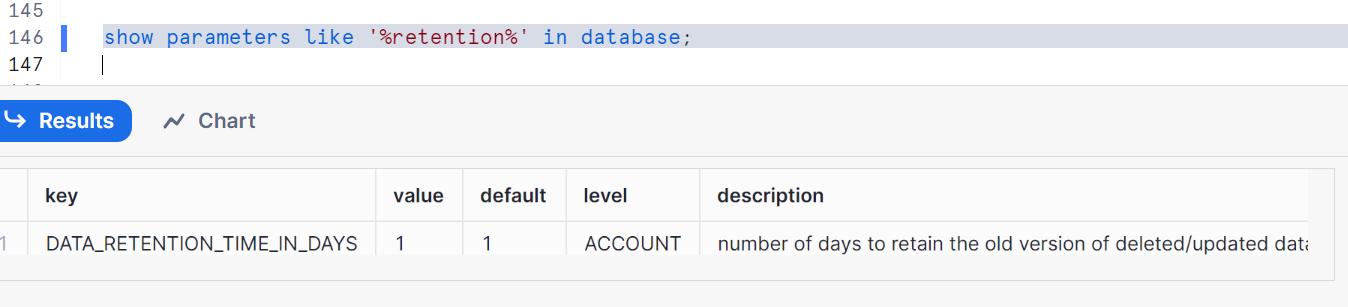
****

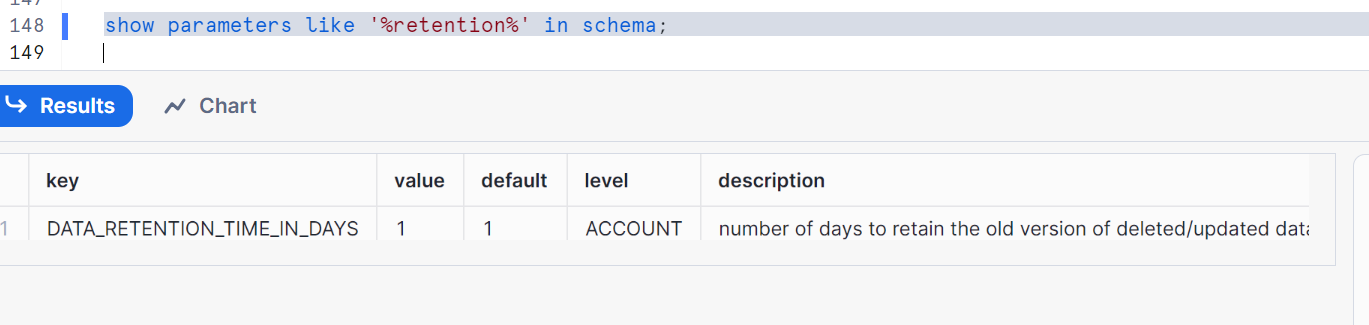
Let’s do hands on

Creating new database and schema

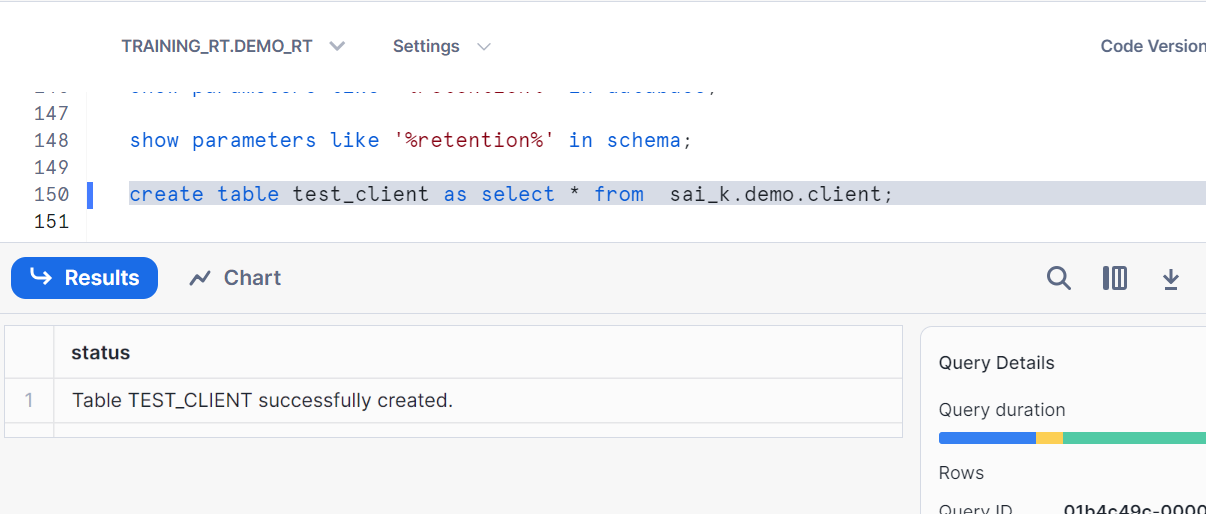




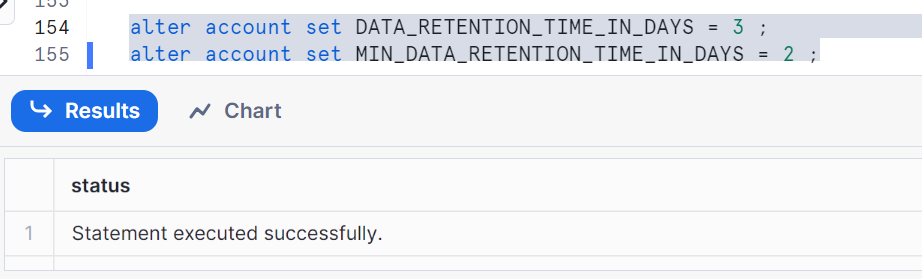


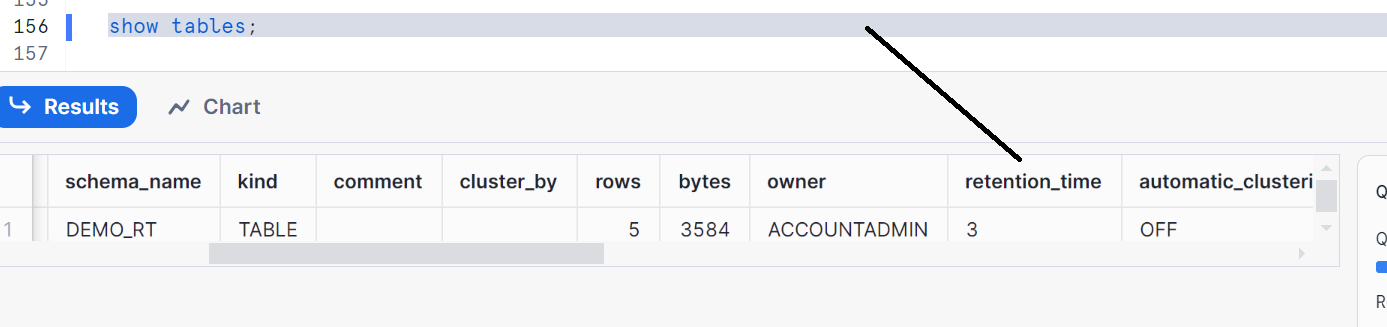


Let’s create a table in training\_rt DB and demo\_rt schema

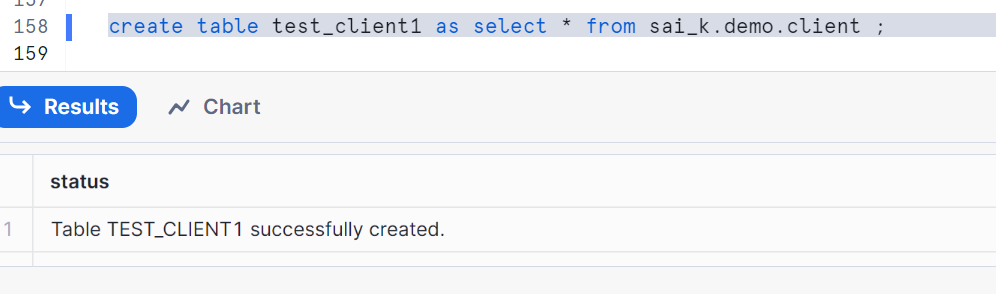


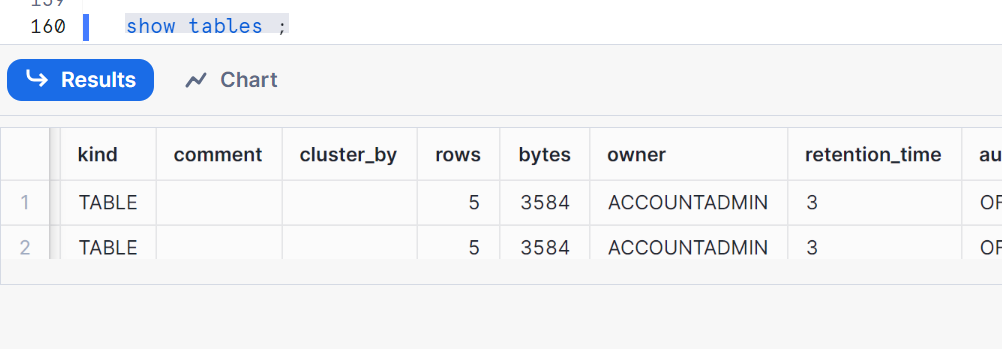
Let’s alter





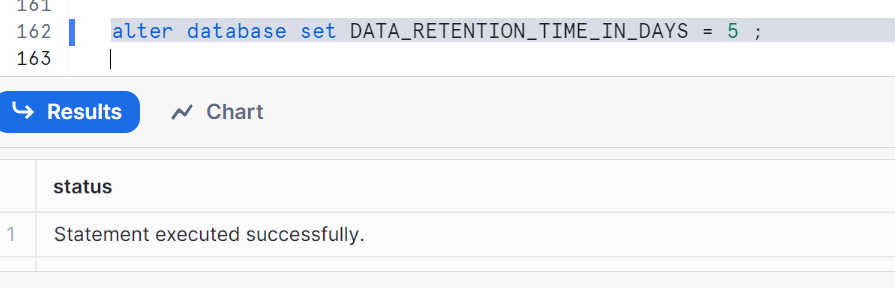
Creating new table and check

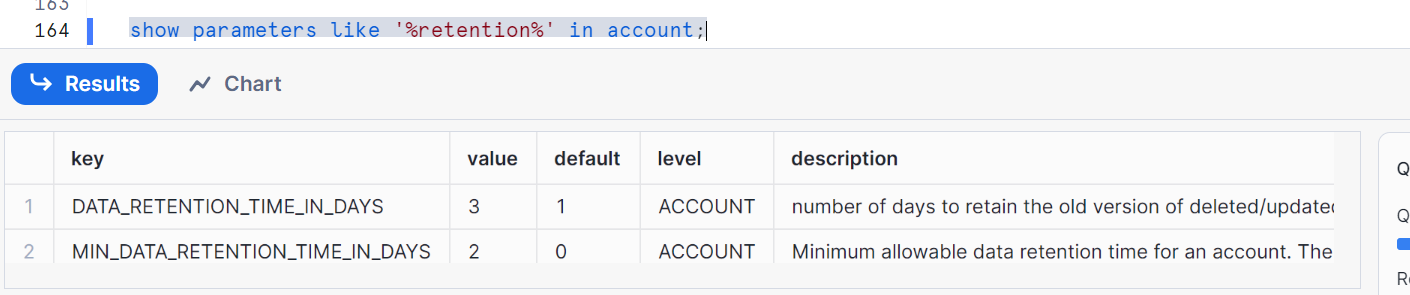


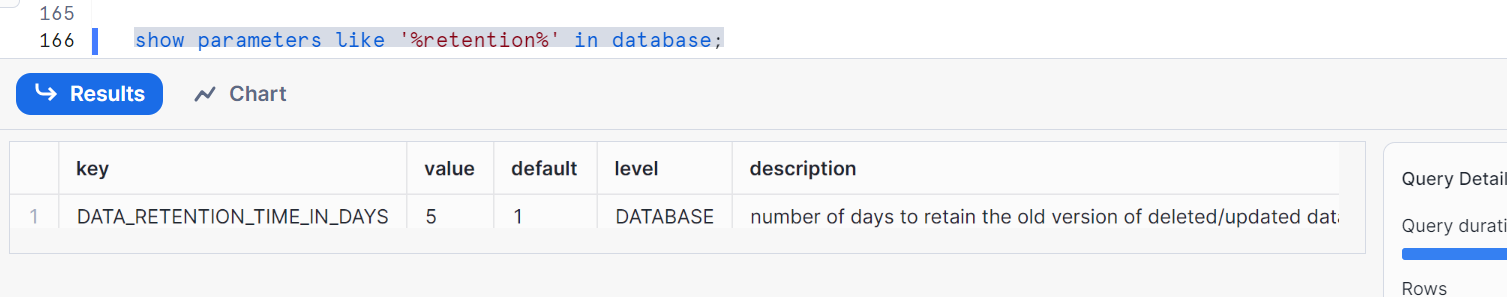


Retention time is 3, taken from account

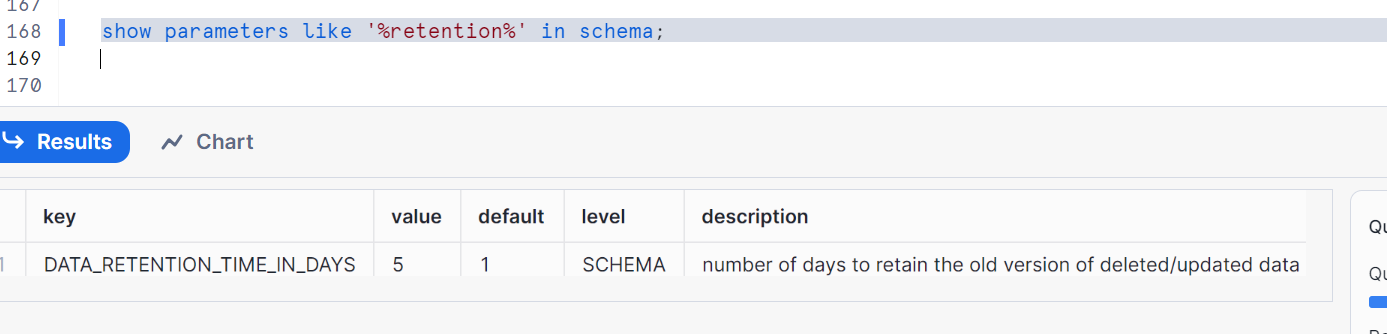
Now, change retention at DB level



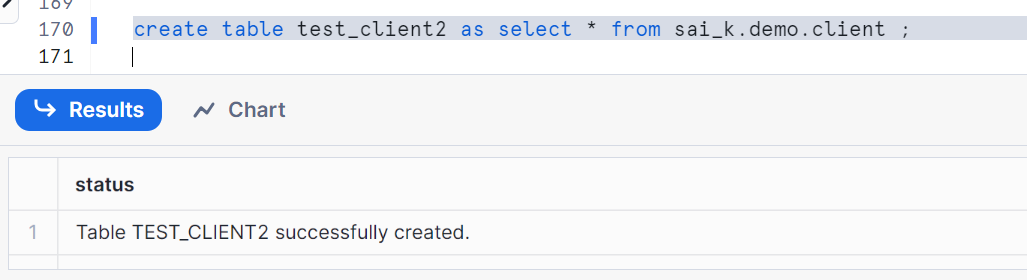


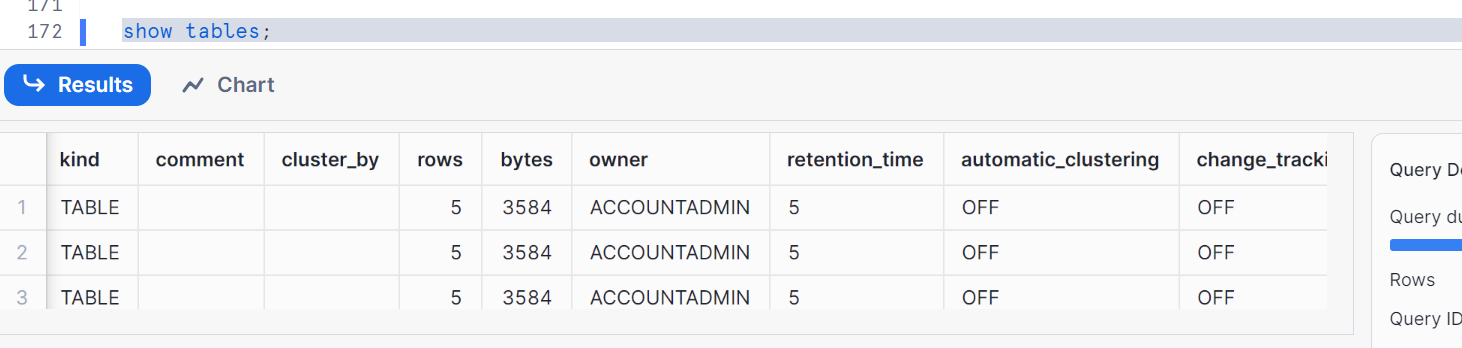


Overrides account, since modified at DB and flows to schema as there is no override at schema



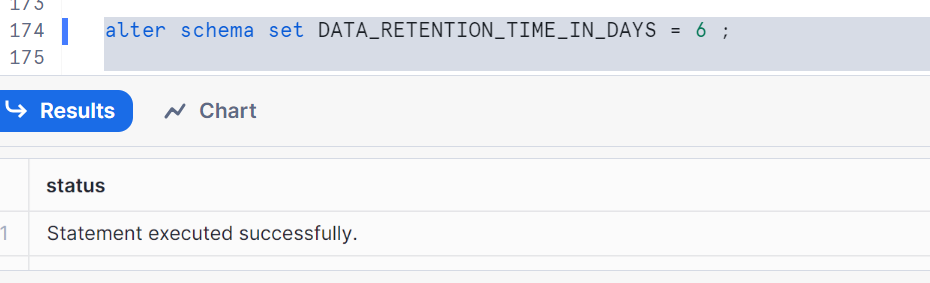
Gets from database

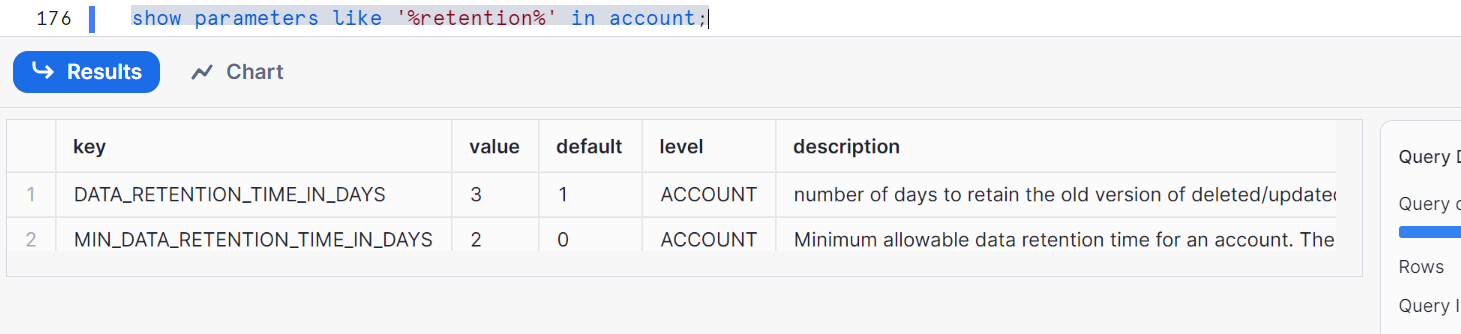




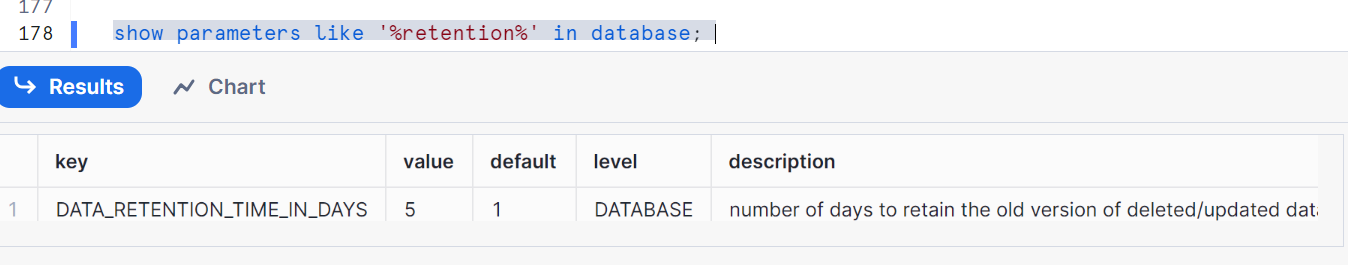
5 for new table as well, 5 for all (prev)

Now let’s set at schema level

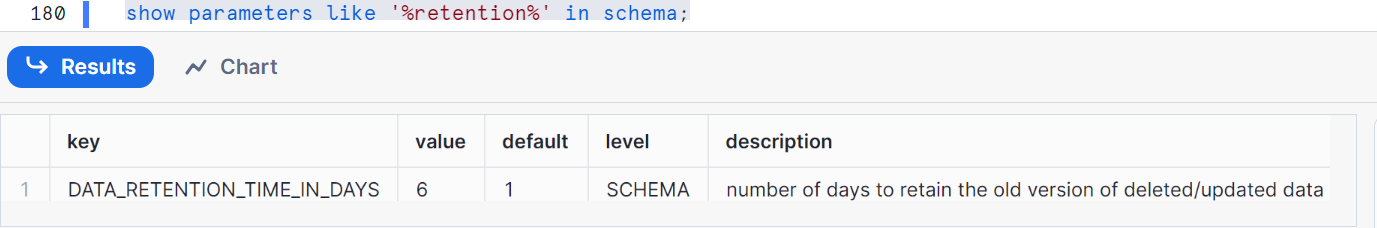




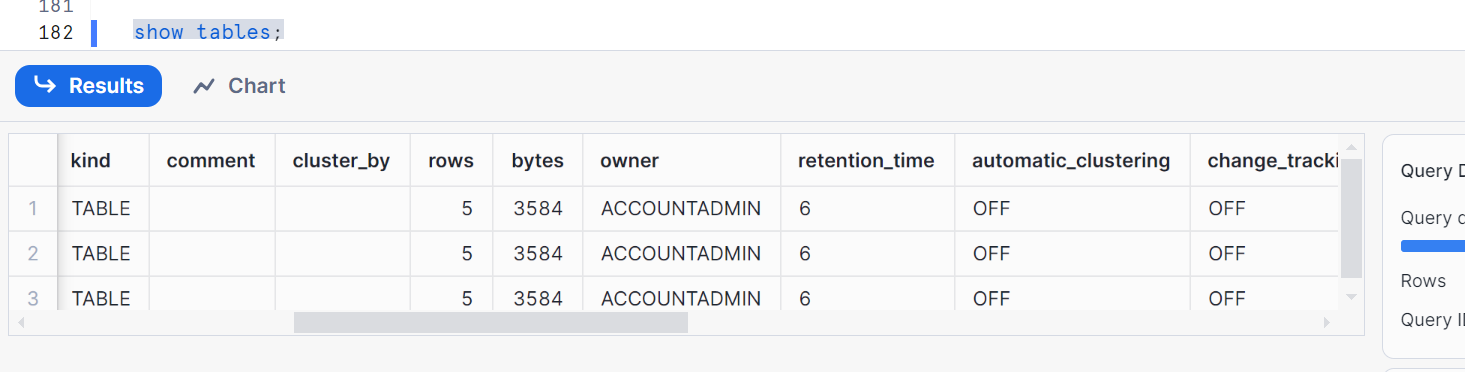
Value is 3



Value is 5

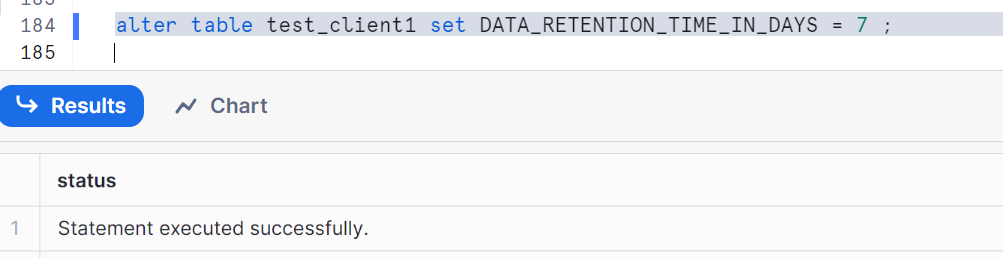


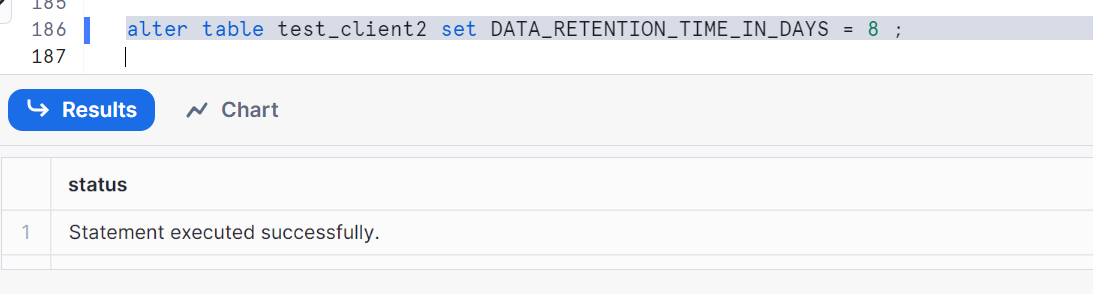
Value is 6, schema override will flow to tables

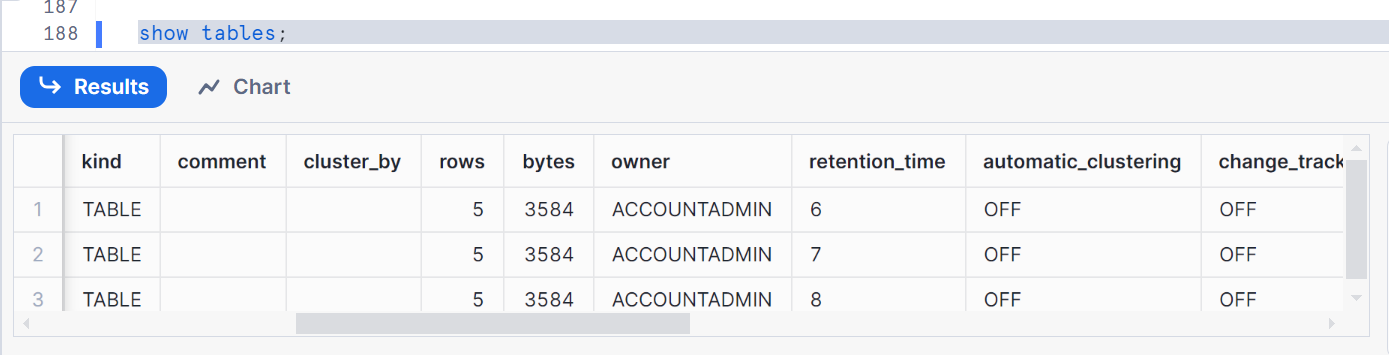


Retention time is 6 for new and existing

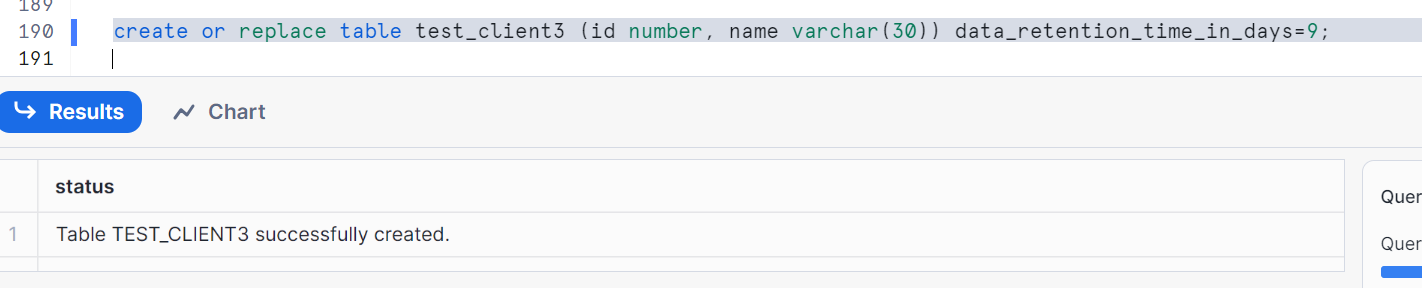
**Setting at table level**

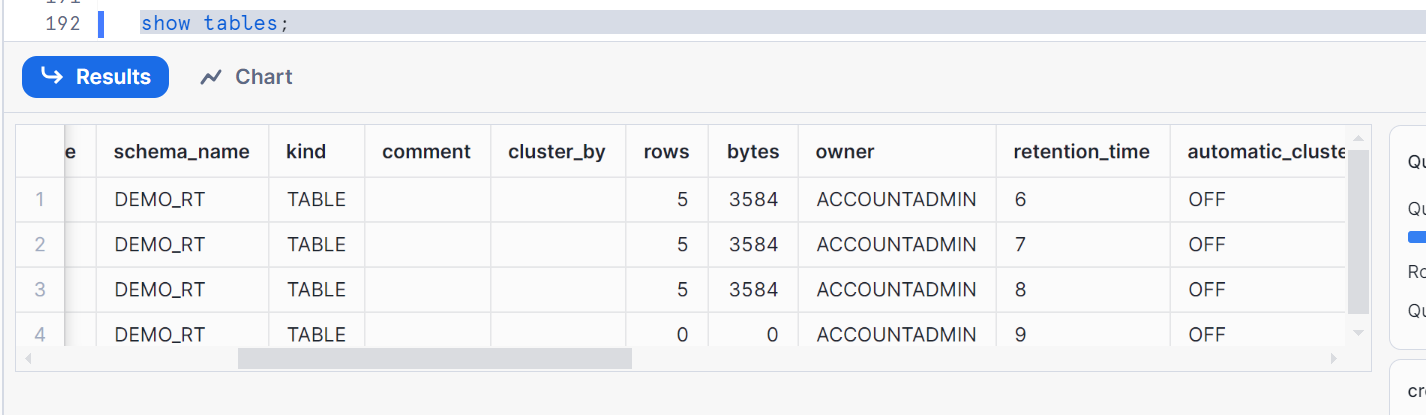
****

****

****

**Setting up at create table level**

****

****

**Increasing retention - it increases the time period for which the data is available in time travel => 5 -> 10, does not apply to data already in fail safe**

**Decreasing retention - it reduces the time period for which the data is available in time travel => 6 -> 4. Some data move to fail-safe**

**-- check the storage cost and different buckets**

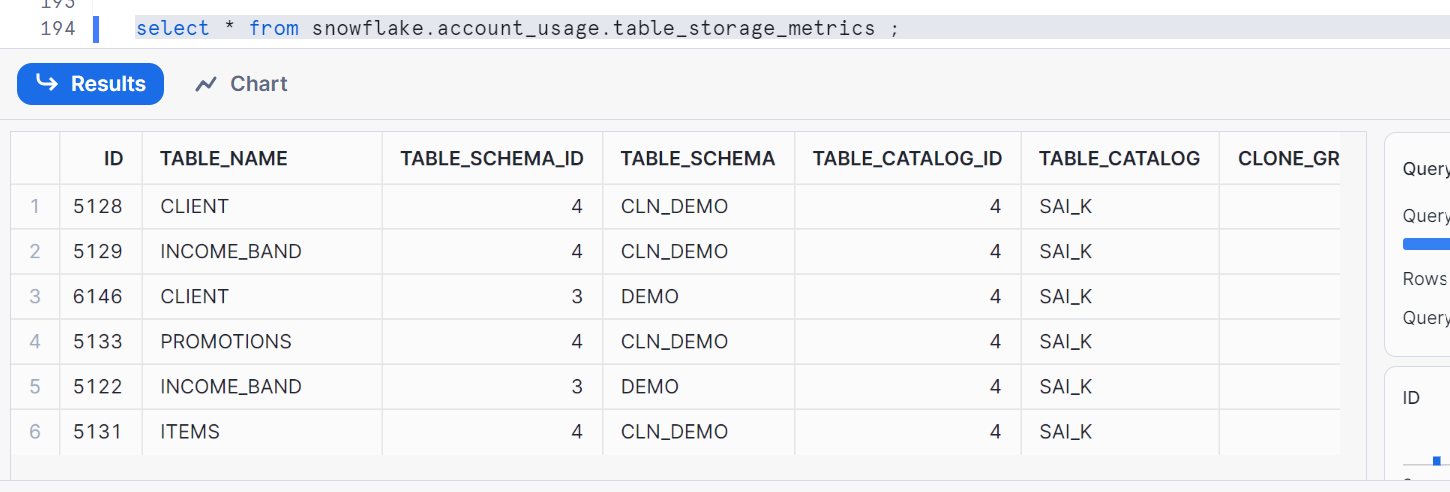
**-- UI only shows Stage, Database and Fail Safe**

**-- Since time travel stores the data to be available it incurs the cost.**

**-- We can check the cost –bucket for time travel via UI as well -- as using SQL queries**

**-- check the storage cost and different buckets**

Below query shows time travel bytes



**Note**: fail safe will be explained in upcoming post

**Thanks**