Snowflake Tutorial

Summary	In this codelab you will get detailed insight into snowflake architecture. Once we are familiar with snowflake architecture we will leverage the knowledge to understand some of the very important features of snowflake like, time travel, clone or zero copy , drop and undrop and copy commands. We will see different methods to load data and unload data from snowflake. Configuring snowflake with salesforce. We will do data load and unload from salesforce. We will also see how to execute queries on data stored in aws s3 and google gcs from snowflake. We discuss best practices to create development databases using snowflake sampling techniques and different sampling algorithms of snowflake.
URL	https://docs.google.com/document/d/1V99Bl51cO-yRXkyY7S89RvFbO_gSfJ1oyYm7aRUK9zk/edit?usp=sharing
Category	Snowflake
Author	Uthsav Shetty

What is Snowflake?

Snowflake Inc. is a cloud-based data-warehousing startup that was founded in 2012.

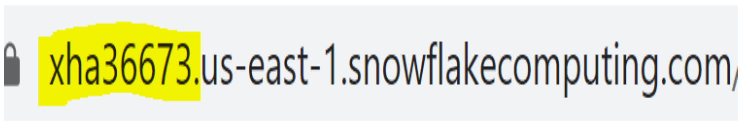
Snowflake offers a cloud-based data storage and analytics service, generally termed "data warehouse-as-a-service". It allows corporate users to store and analyze data using cloud-based hardware and software. Snowflake runs on Amazon S3 and on Microsoft Azure. It is being rolled out on Google Cloud Platform in 2019. Its Snowflake Data Exchange allows customers to discover, exchange and securely share data.

How to Create an Enterprise Edition account ?

- Go to trail.snowflake
- Create an enterprise edition and chose AWS as the hosting service.
- Region - US East N.Virginia
- Rest fill all details to create a trial account.
- Before that, create an AWS educate account to avail $100 credits.
- You'll receive an email and activate the account.

Snowflake Web Console Introduction

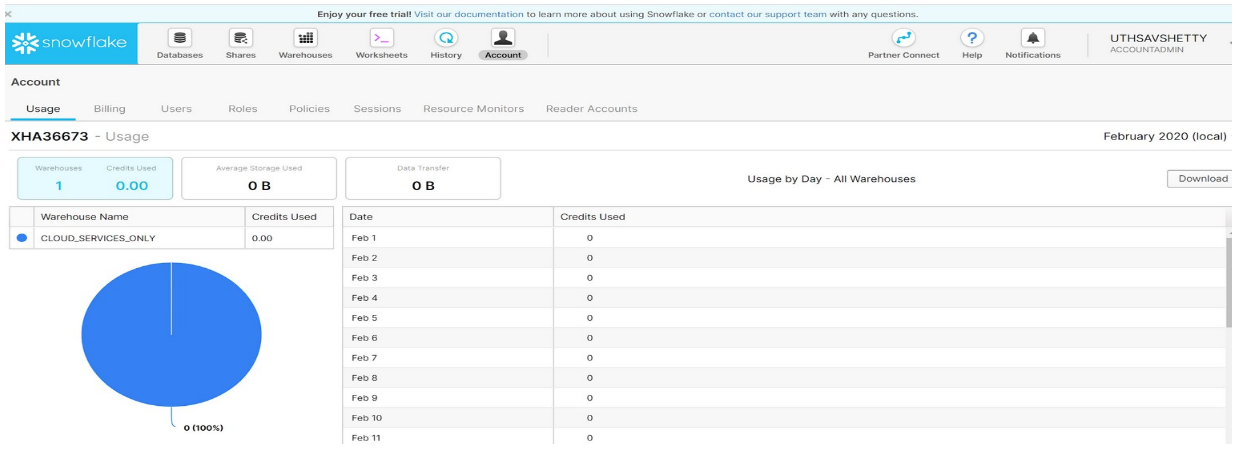
- Create Username after you receive the verification email.
- This will be your Unique Snowflake ID.



xha36673.us-east-1.snowflakecomputing.com/

- You can see a list of components on the web console like databases, Shares, Warehouses, Worksheets, Partner Connect & History.
- Shares means you can share databases across Snowflakes account
- Warehouses (don't get confused with data warehousing concept) means a group of machines which helps you to process the data based on the volume of data you want to proceed. You can create different sizes of warehouses like Large, X-Large or 2X-Large.
- Worksheets - can write all queries and create many worksheets in a particular account.
- History - can check all the queries which got executed and can filter the results based on parameters..
- Partner Connect - using these 3rd party tools can migrate data to snowflake databases.
- Through Help - can go to documentation and download several drivers like JDBC, ODBC, Python or Spark connectors.
- Can switch between roles using the dropdown menu. Like Admin - You can manage all the users all the rules and policies and also resource monitors and reader accounts

-



Enjoy your free trial! Visit our documentation to learn more about using Snowflake or contact our support team with any questions.

Account

XHA36673 - Usage

February 2020 (local)

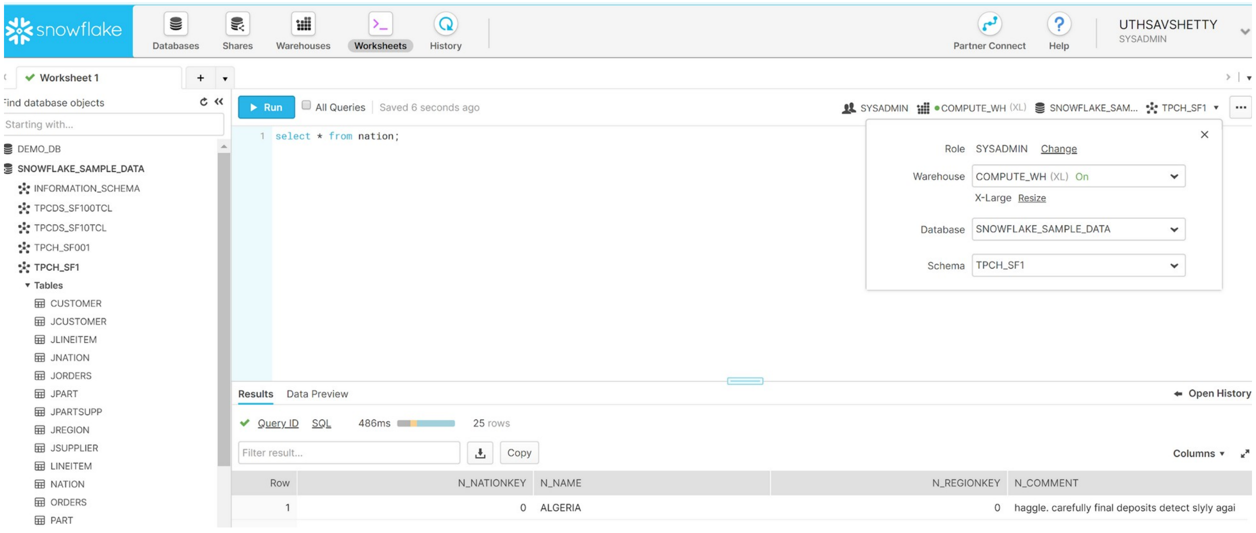
Warehouses	Credits Used	Average Storage Used	Data Transfer
1	0.00	0 B	0 B

Usage by Day - All Warehouses

	Warehouse Name	Credits Used
●	CLOUD_SERVICES_ONLY	0.00

Date	Credits Used
Feb 1	0
Feb 2	0
Feb 3	0
Feb 4	0
Feb 5	0
Feb 6	0
Feb 7	0
Feb 8	0
Feb 9	0
Feb 10	0
Feb 11	0

How to execute a query in Snowflake?



```
select * from nation;
```

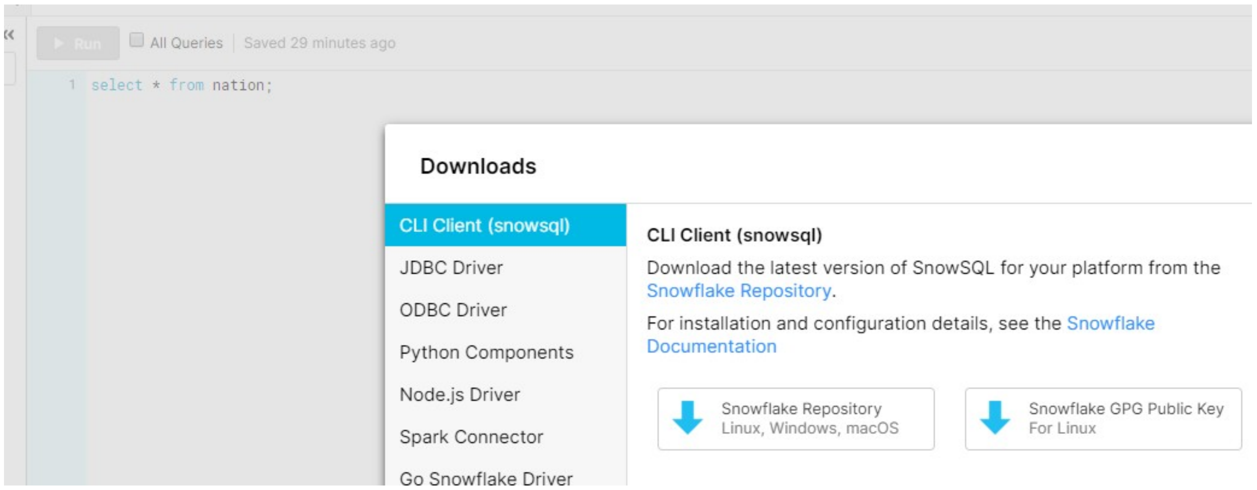
Role SYSADMIN Change
Warehouse COMPUTE_WH (XL) On
X-Large Resize
Database SNOWFLAKE_SAMPLE_DATA
Schema TPCH_SF1

Query ID SQL 486ms 25 rows

Row	N_NATIONKEY	N_NAME	N_REGIONKEY	N_COMMENT
1	0	ALGERIA	0	haggle. carefully final deposits detect slyly agai

- Query the table - nation from database - SNOWFLAKE_SAMPLE_DATA and follow the screenshot before querying it
- Each query you execute will have a query ID.
- Copy the query ID below and will direct you to the profile page where you high level details of this query (how much data got scanned & time it took to return the result).
- Limitation of using Snowflake web console - Cannot check for the syntax error before I start executing these set of queries.

Login into Snowflake from SnowflakeCLI



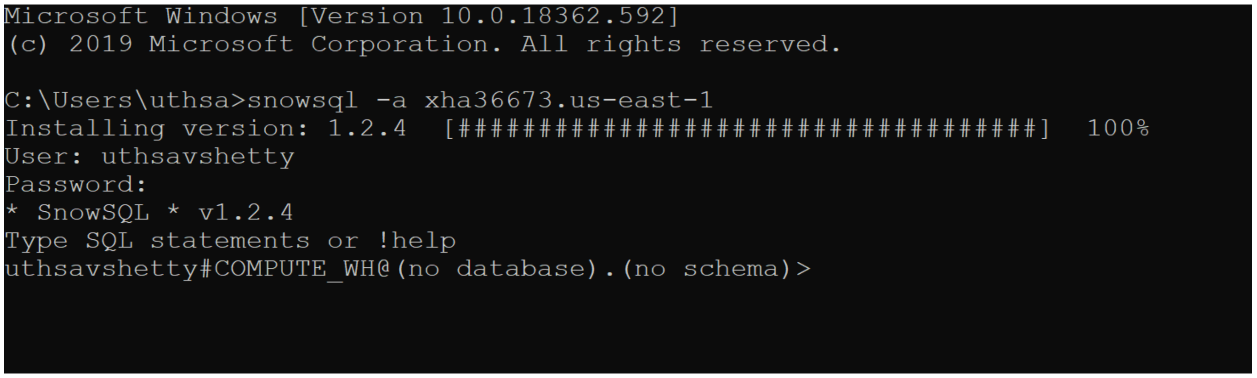
```
select * from nation;
```

CLI Client (snowsql)

Download the latest version of SnowSQL for your platform from the Snowflake Repository.

For installation and configuration details, see the Snowflake Documentation

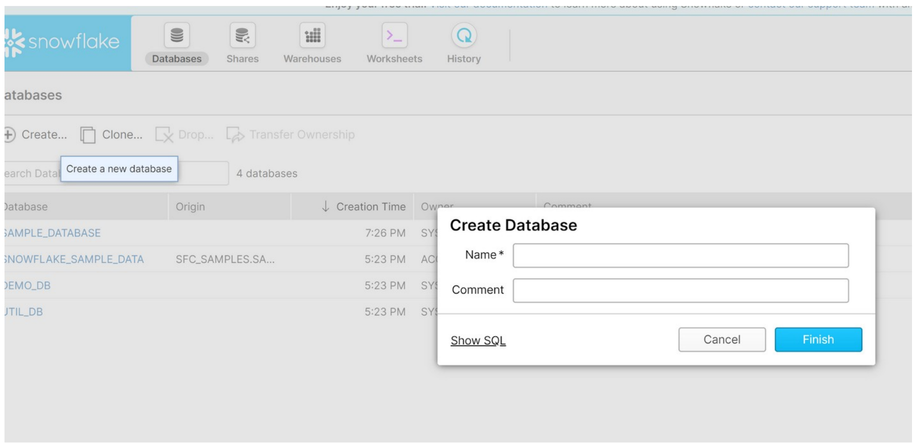
- You can log into snowflake using command line interface.



```
Microsoft Windows [Version 10.0.18362.592]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\uthsa>snowsql -a xha36673.us-east-1
Installing version: 1.2.4  [####################################]  100%
User: uthsavshetty
Password:
* SnowSQL * v1.2.4
Type SQL statements or !help
uthsavshetty#COMPUTE_WH@(no database).(no schema)>
```

- -a > account and your account name in snowflake is already been discussed. You can create databases and tables either by executing the query or through a web console.



atabase	Origin	Creation Time	Owner	Comment
SAMPLE_DATABASE		7:26 PM	SYS	
SNOWFLAKE_SAMPLE_DATA	SFC_SAMPLES.SA...	5:23 PM	AC	
DEMO_DB		5:23 PM	SYS	
UTIL_DB		5:23 PM	SYS	

Create Database

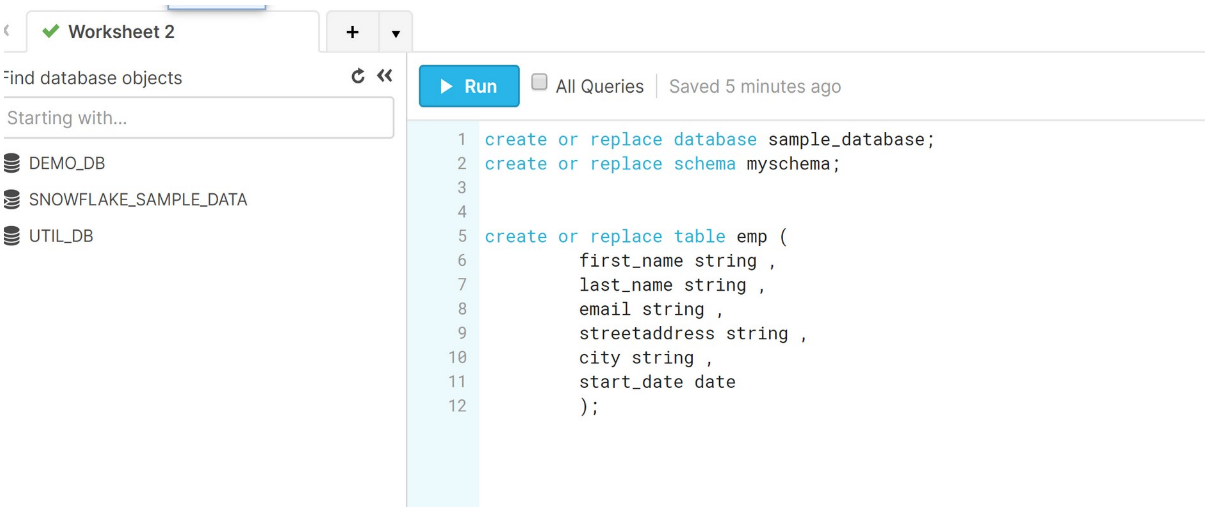
Name *

Comment

Show SQL

Cancel

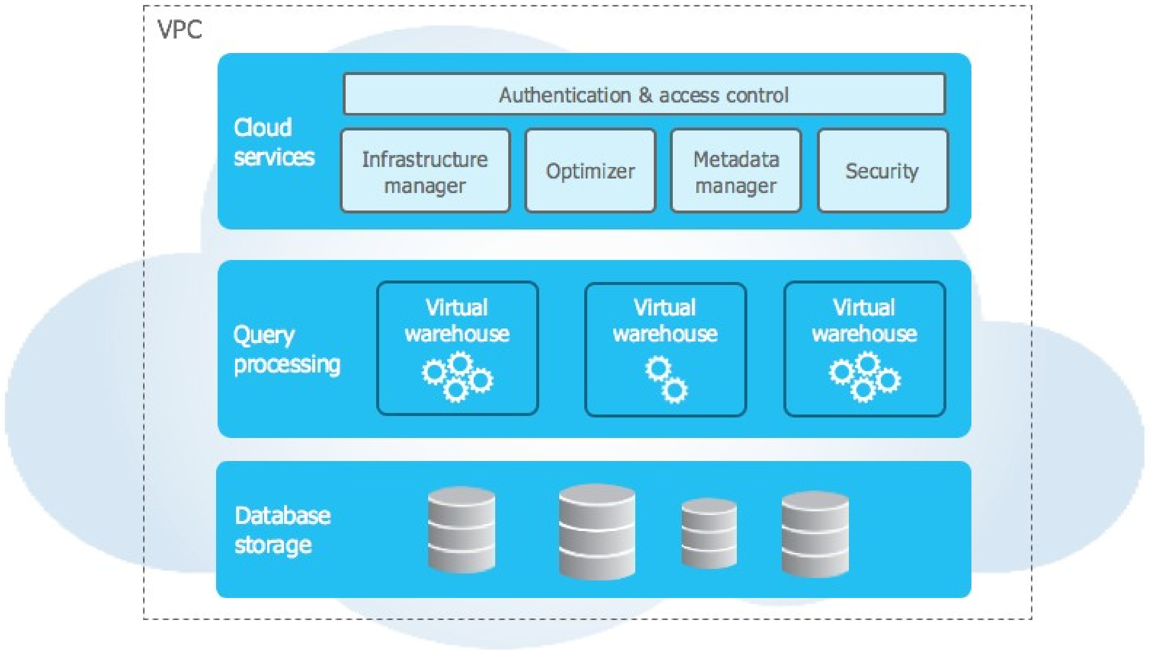
Finish



```
create or replace database sample_database;
create or replace schema myschema;


create or replace table emp (
         first_name string ,
         last_name string ,
         email string ,
         streetaddress string ,
         city string ,
         start_date date
         );
```

Snowflake Architecture



- Snowflake's architecture is a hybrid of traditional shared-disk database architectures and shared-nothing database architectures. Similar to shared-disk architectures, Snowflake uses a central data repository for persisted data that is accessible from all compute nodes in the data warehouse. But similar to shared-nothing architectures, Snowflake processes queries using MPP (massively parallel processing) compute clusters where each node in the cluster stores a portion of the entire data set locally. This approach offers the data management simplicity of a shared-disk architecture, but with the performance and scale-out benefits of a shared-nothing architecture.

- Snowflake's unique architecture consists of three key layers:
- Database Storage
- Query Processing
- Cloud Services

Database Storage

When data is loaded into Snowflake, Snowflake reorganizes that data into its internal optimized, compressed, columnar format. Snowflake stores this optimized data in cloud storage.

Snowflake manages all aspects of how this data is stored — the organization, file size, structure, compression, metadata, statistics, and other aspects of data storage are handled by Snowflake. The data objects stored by Snowflake are not directly visible nor accessible by customers; they are only accessible through SQL query operations run using Snowflake.

Query Processing

Query execution is performed in the processing layer. Snowflake processes queries using "virtual warehouses". Each virtual warehouse is an MPP compute cluster composed of multiple compute nodes allocated by Snowflake from a cloud provider.

Each virtual warehouse is an independent compute cluster that does not share compute resources with other virtual warehouses. As a result, each virtual warehouse has no impact on the performance of other virtual warehouses.

For more information, see Virtual Warehouses.

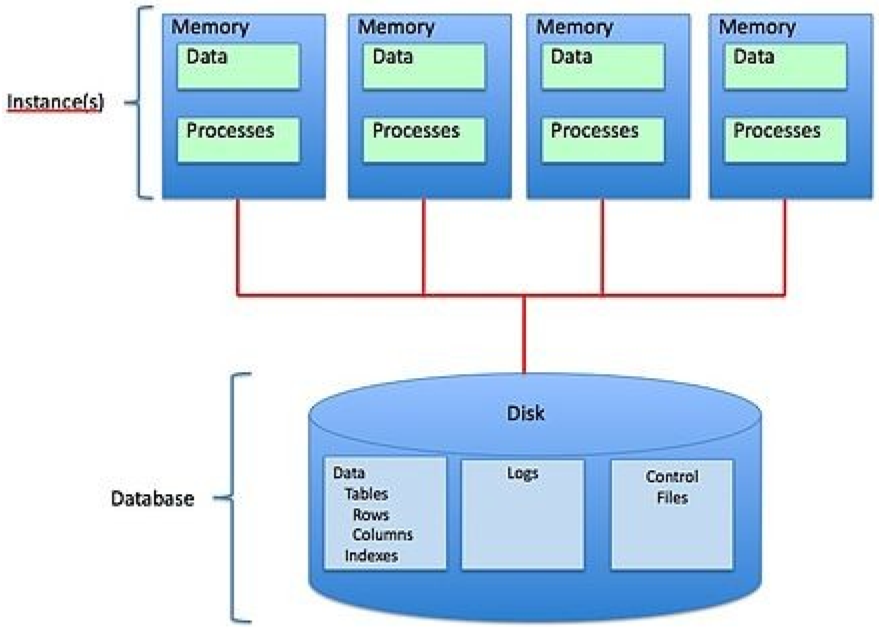
Cloud Services

The cloud services layer is a collection of services that coordinate activities across Snowflake. These services tie together all of the different components of Snowflake in order to process user requests, from login to query dispatch. The cloud services layer also runs on compute instances provisioned by Snowflake from the cloud provider. Among the services in this layer:

- Authentication
- Infrastructure management
- Metadata management
- Query parsing and optimization
- Access control

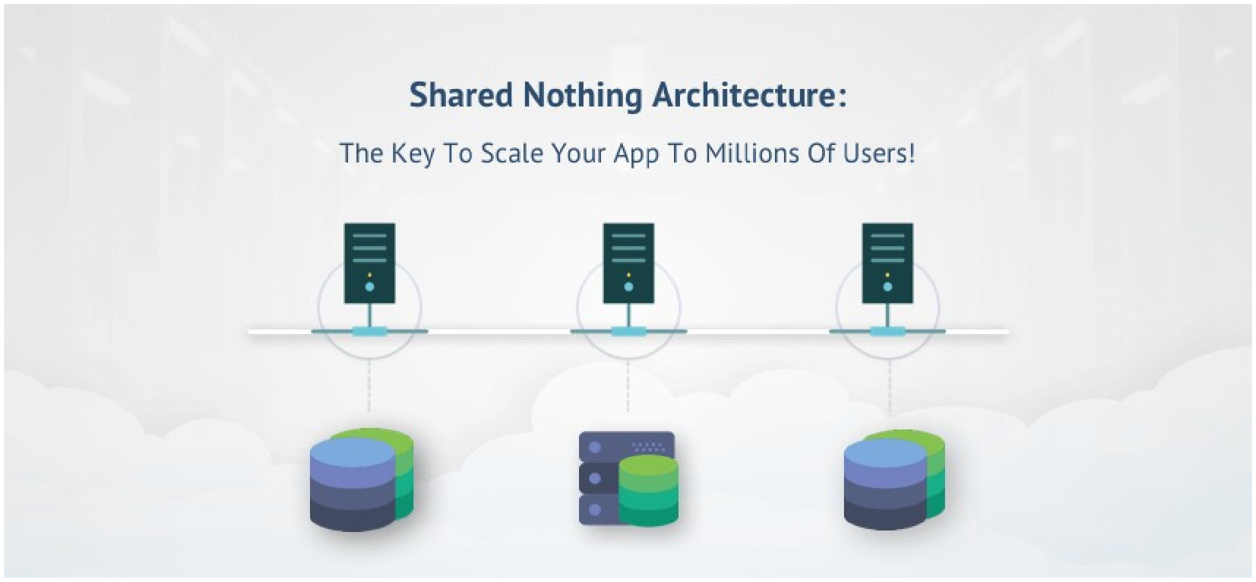
- You have to understand the Snowflake architecture because there are few features like time travel drop and data cloning which are dependent on this architecture.
- If you understand the architecture better, it will help you in understanding how the process is happening in the backend.
- This will help you to save the cost while using a snowflake database.
- One major thing you will be charged for every query you execute.

Shared Disk Architecture



- In this architecture, there will be one main disk which will be accessed by multiple databases. Every database node will try to read data from the shared disk.
- Scalability is limited - as node increases, will be difficult for all databases nodes to access this disk because this will be a bottleneck for the main disk.
- Hard to maintain data consistency across the cluster - 2 or more databases nodes try to read & write data at the same time, will be difficult to maintain consistency.
- Bottleneck of communication with shared disk.

Shared Nothing Architecture


Shared Nothing Architecture:
The Key To Scale Your App To Millions Of Users!

- Eliminates bottleneck problem.
- Scales processing and compute together
- Moves data storage close to compute

- **Problems**
- Data distributed across the cluster requires shuffling between nodes.
- Performance is heavily dependent on how data is distributed across the nodes in the system.
- Compute can’t be sized independently of storage.

Caching in Snowflake Data Warehouse

Snowflake Caching

- Like any modern database, Snowflake uses various levels of caching to maximize query performance, and reduce machine load.

System Architecture

Before starting it's worth considering the underlying Snowflake architecture, and explaining when Snowflake caches data. The diagram below illustrates the overall architecture which consists of three layers:-

1. **Service Layer:** Which accepts SQL requests from users, coordinates queries, managing transactions and results. Logically, this can be assumed to hold the *result cache* – a cached copy of the results of every query executed.
2. **Compute Layer:** Which actually does the heavy lifting. This is where the actual SQL is executed across the nodes of a *Virtual Data Warehouse*. This layer holds a cache of data queried, and is often referred to as *Local Disk I/O* although in reality this is implemented using SSD storage. All data in the compute layer is temporary, and only held as long as the virtual warehouse is active.
3. **Storage Layer:** Which provides long term storage of results. This is often referred to as *Remote Disk*, and is currently implemented on either Amazon S3 or Microsoft Blob storage.

Snowflake Caching Layers

illustrates the levels at which data and results are cached for subsequent use. These are:-

1. **Result Cache:** Which holds the results of every query executed in the past 24 hours. These are available across virtual warehouses, so query results returned to one user is available to any other user on the system who executes the same query, provided the underlying data has not changed.
2. **Local Disk Cache:** Which is used to cache data used by SQL queries. Whenever data is needed for a given query it's retrieved from the *Remote Disk* storage, and cached in SSD and memory.
3. **Remote Disk:** Which holds the long term storage. This level is responsible for data resilience, which in the case of Amazon Web Services, means 99.999999999% durability. Even in the event of an entire data centre failure.

Test Sequence

The test sequence was as follows:-

1. **Run from cold:** Which meant starting a new virtual warehouse (with no local disk caching), and executing the query.
2. **Run from warm:** Which meant disabling the result caching, and repeating the query. This makes use of the local disk caching, but not the result cache.
3. **Run from hot:** Which again repeated the query, but with the result caching switched on.

Each query ran against 60Gb of data, although as Snowflake returns only the columns queried, and was able to automatically compress the data, the actual data transfers were around 12Gb. As Snowflake is a columnar data warehouse, it automatically returns the columns needed rather then the entire row to further help maximize query performance.

Run from Cold

This query returned in around 20 seconds, and demonstrates it scanned around 12Gb of compressed data, with 0% from the local disk cache. This means it had no benefit from disk caching.

Run from Warm

- This query was executed immediately after, but with the result cache disabled, and it completed in 1.2 seconds – around 16 times faster. In this case, the *Local Disk* cache (which is actually SSD on Amazon Web Services) was used to return results, and disk I/O is no longer a concern.

In the above case, the disk I/O has been reduced to around 11% of the total elapsed time, and 99% of the data came from the (local disk) cache. While querying 1.5 billion rows, this is clearly an excellent result.

Run from Hot

- This query returned results in milliseconds, and involved re-executing the query, but with this time, the result cache enabled. Normally, this is the default situation, but it was disabled purely for testing purposes.

Although more information is available in the Snowflake Documentation, a series of tests demonstrated the result cache will be reused unless the underlying data (or SQL query) has changed. As a series of additional tests demonstrated inserts, updates and deletes which don't affect the underlying data are ignored, and the result cache is used, provided data in the micro-partitions remains unchanged.

Finally, results are normally retained for 24 hours, although the clock is reset every time the query is re-executed, up to a limit of 30 days, after which results query the remote disk.

The sequence of tests was designed purely to illustrate the effect of data caching on Snowflake. The tests included:-

- **Raw Data:** Including over 1.5 billion rows of TPC generated data, a total of over 60Gb of raw data
- **Initial Query:** Took 20 seconds to complete, and ran entirely from the remote disk. Quite impressive.
- **Second Query:** Was 16 times faster at 1.2 seconds and used the *Local Disk* (SSD) cache.
- **Result Set Query:** Returned results in 130 milliseconds from the result cache (intentionally disabled on the prior query).

Notes:

- Always need a virtual warehouse to execute queries
- Always use limit clause with select * from queries (to process such huge volume of data, charges will be applied)

- During development activity always keep auto suspend time limit to a higher value at least 15 mins.
- Share your virtual warehouse when a group of users are working on the common tables.
- Never disable cloud service layer result cache to avoid extra charges.
- Reusing query results in snowflake is free.

- If there's any changes to the underlying table, Query will not use reserved cached query result (goes to data storage layer) instead it submits the queries to the virtual warehouse to process the data.

Micro - Partitions & Data Clustering

Traditional data warehouses rely on static partitioning of large tables to achieve acceptable performance and enable better scaling. In these systems, a *partition* is a unit of management that is manipulated independently using specialized DDL and syntax; however, static partitioning has a number of well-known limitations, such as maintenance overhead and data skew, which can result in disproportionately-sized partitions.

Snowflake has implemented a powerful and unique form of partitioning, called *micro-partitioning*, that delivers all the advantages of static partitioning without the known limitations, as well as providing additional significant benefits.

All data in Snowflake tables is automatically divided into **micro-partitions**, which are contiguous units of storage. Each micro-partition contains between 50 MB and 500 MB of uncompressed data (note that the actual size in Snowflake is smaller because data is always stored compressed). Groups of rows in tables are mapped into individual micro-partitions, organized in a columnar fashion. This size and structure allows for extremely granular pruning of very large tables, which can be comprised of millions, or even hundreds of millions, of micro-partitions.

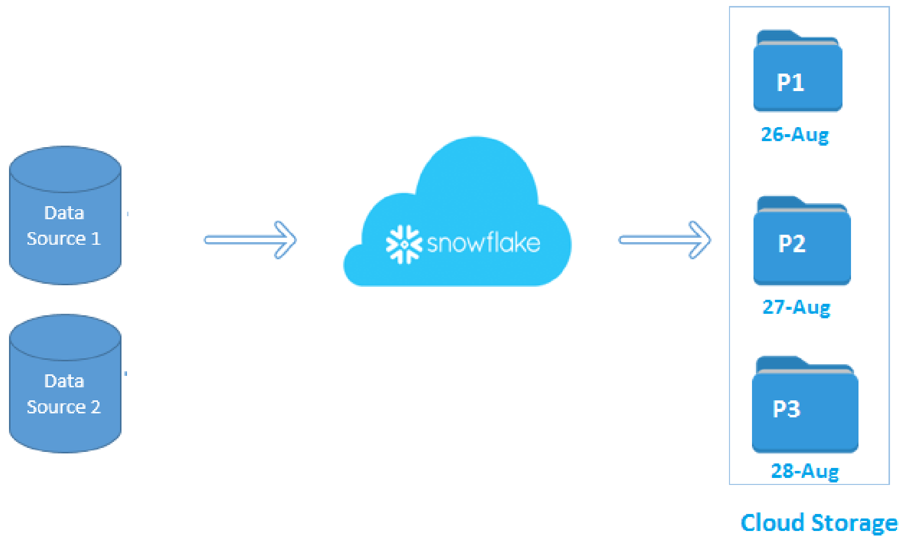
Snowflake stores metadata about all rows stored in a micro-partition, including:

- The range of values for each of the columns in the micro-partition.
- The number of distinct values.
- Additional properties used for both optimization and efficient query processing.

- Micro - partitioning is automatically performed on all snowflake tables. Tables are transparently partitioned using the ordering of the data as it is inserted/loaded.

- **Clustering** is the process of grouping records within micro partition and if you provide defined clustering key on a snowflake table then snowflake will automatically group data in micro partition by the defined key.
- In Snowflake, all data in tables is automatically divided into micro-partitions. Tables are transparently partitioned using the ordering of the data as it is inserted/loaded.

For example: Let's assume you will have a single fact table with one year of historical sales data with a date column. You are loading daily transaction data, snowflake partitions them in the same order as the data arrives. For simplicity, we have one micro-partition per day in the below diagram. But you will have **N** number of micro-partitions in a day.
When the user queries a date, Snowflake knows exactly which micro-partitions has that data (Based on the micro-partition metadata). It will then only scan the portion of the micro-partitions that contain the data.
Statistics are collected for each block independently on block creation, which is handled by the Snowflake engine transparently to the users.



Tables are automatically divided into micro-partitions, we can't partition our tables on our own. We do have a workaround for this, you can sort the table data by a field like transaction date. It will allow Snowflake to use min-max statistics of micro-partitions to prune the ones that do not contain the relevant dates for the queries that filter on the date. This is called clustered tables. Snowflake will maintain the data clustered for you transparently, but of course for a fee for the compute and storage resources needed to achieve this.

- To know about snowflake clustering refer this link:

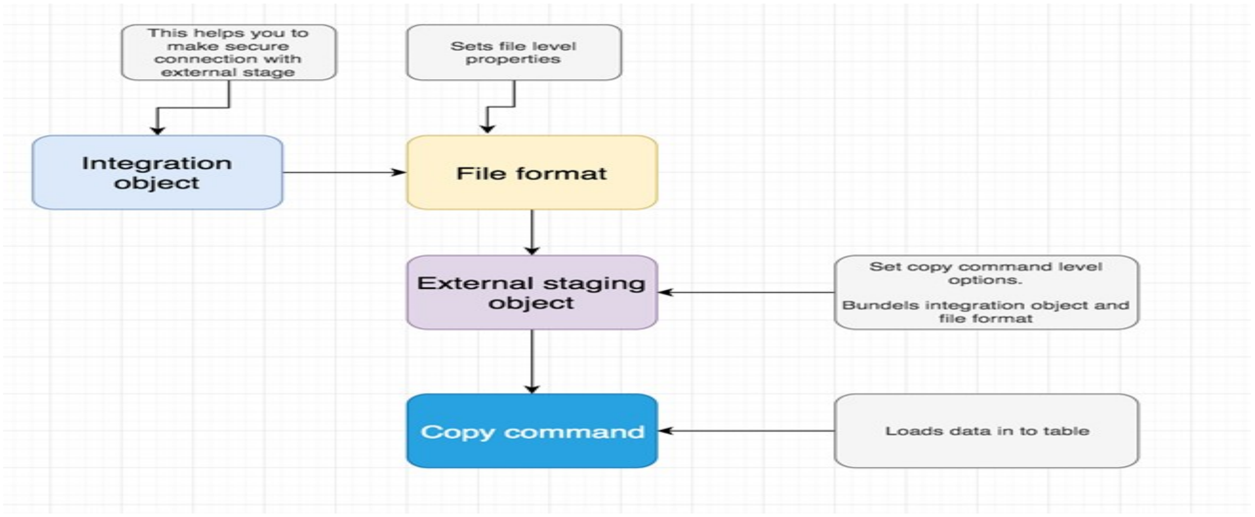
  https://docs.snowflake.net/manuals/user-guide/tables-clustering-keys.html

---

Loading data in Snowflake

- 2 staging areas : Load using internal staging area (managed by snowflake) and load using external staging area (external blob storages like amazon, gcp & azure)

Loading data from AW S3 to Snowflake


This helps you to make secure connection with external stage
Sets file level properties
Integration object
File format
External staging object
Set copy command level options.
Bundels integration object and file format
Copy command
Loads data in to table

1. Create AWS S3 Bucket
2. Create AWS S3 policy (need to create a policy for the bucket which was created)

```
    {
"Version": "2012-10-17",
"Statement": [
  {
```

```
            "Effect": "Allow",
            "Action": [
              "s3:PutObject",
              "s3:GetObject",
              "s3:GetObjectVersion",
              "s3:DeleteObject",
              "s3:DeleteObjectVersion"
            ],
            "Resource": "arn:aws:s3:::uthsavsnowflake/*"
        },
        {
            "Effect": "Allow",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::uthsavsnowflake",
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "<prefix>/*"
                    ]
                }
            }
        }
    ]
}
```

3. Create AWS role
    - Click on IAM->Copy Account Number -> Click on roles -> Another AWS account -> Choose the policy which you created -> Give a name to the role name -> create

4. Create Snowflake Integration
    - --- AWS S3 Configuration.

```
create or replace storage integration s3_int
type = external_stage
storage_provider = s3
enabled = true
storage_aws_role_arn = 'arn:aws:iam::***************:role/Snowflake'
 storage_allowed_locations = ('s3://uthsavsnowflake/employee/');

DESC INTEGRATION s3_int;
```

- After running this command, copy storage_aws_role_arn and storage_aws_role_external_id on IAM (-> edit trust relationships) -> update

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "storage_aws_role_arn"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "storage_aws_external_id"
        }
      }
    }
  ]
}
```

- Now AWS and Snowflake can talk to each other and integration object is created.

5. Upload data to AWS S3 using aws console
6. Query AWS from Snowflake
   Here will create file format and external staging object

   - create or replace file format my_csv_format
     type = csv field_delimiter = ',' skip_header = 1 null_if = ('NULL', 'null')
     empty_field_as_null = true;

   - desc file format my_csv_format

   - create or replace stage my_s3_stage
     storage_integration = s3_int
     url = 's3://snowflake069/employee/'
     file_format = my_csv_format;

   - select t.$1 as first_name,t.$2 last_name,t.$3 email,t.$4 from @my_s3_stage/ t;
7. Load data from s3 to snowflake
   - create or replace table emp_ext_stage (first_name string , last_name string , email string , streetaddress string , city string , start_date date
     );
   - copy into emp_ext_stage
     from (select t.$1 , t.$2 , t.$3 , t.$4 , t.$5 , t.$6 from @my_s3_stage/ t)
     --pattern = '.*employees0[1-5].csv'
     on_error = 'CONTINUE';

   - TRUNCATE TABLE emp_ext_stage;

   - SELECT * FROM emp_ext_stage

- select * from table(validate(emp_ext_stage, job_id=>'018fe737-03d4-9502-0000-e5fd000122b6'));

8. Unload data to S3
   - create or replace file format my_csv_unload_format
     type = csv field_delimiter = ',' skip_header = 1 null_if = ('NULL', 'null')
     empty_field_as_null = true compression = gzip;

   - alter storage integration s3_int set storage_allowed_locations=('s3://snowflake069/employee/','s3://snowflake069/emp_unload/','s3://snowflake069/zip_folder/')

   - desc integration s3_int

   - create or replace stage my_s3_unload_stage
     storage_integration = s3_int
     url = 's3://snowflake069/emp_unload/'
     file_format = my_csv_unload_format;

   - copy into @my_s3_unload_stage
     from
     emp_ext_stage

   - copy into @my_s3_unload_stage/select_
     from
     (select first_name, email from emp_ext_stage)

   - copy into @my_s3_unload_stage/parquet_
     From emp_ext_stage
     FILE_FORMAT=(TYPE='PARQUET' SNAPPY_COMPRESSION=TRUE)

Loading data from GCP to Snowflake

1. Create GCP account
2. Create Bucket
3. Upload Employee folder on GCP bucket
4. Create GCP storage integration object
   - create or replace storage integration gcs_int
     type = external_stage
     storage_provider = gcs
     enabled = true
     storage_allowed_locations = ('gcs://bucketname/employee/')
   - DESC INTEGRATION gcs_int;

After running this query, copy STORAGE_GCP_ACCOUNT value from the query result --> click on GCP bucket -> add members -> paste that value -> Role - Storage Admin

- Now GCP will able to communicate with Snowflake

5. Query GCP data
    - create or replace file format my_csv_format
      type = csv field_delimiter = ',' skip_header = 1 null_if = ('NULL', 'null')
      empty_field_as_null = true;

    - desc file format my_csv_format

    - create or replace stage my_gcs_stage
       storage_integration = gcs_int
       url = 'gcs://snowflake069/emp/'
      file_format = my_csv_format;

    - select t.$1 as first_name,t.$2 last_name,t.$3 email,t.$4 from @my_gcs_stage/ t;

6 . Load data into Snowflake

```
-create or replace table emp_gcs_table ( first_name string , last_name string ,email
 string , streetaddress string , city string, start_date date);

-copy into emp_gcs_table from (select t.$1 , t.$2 , t.$3 , t.$4 , t.$5 , t.$6 from
 @my_gcs_stage/ t)
--pattern = '.*employees0[1-5].csv'
 on_error = 'CONTINUE';


 - TRUNCATE TABLE emp_ext_stage;
-  SELECT * FROM emp_gcs_table
```

7 . Unload GCP

```
-   create or replace storage integration gcs_int
    type = external_stage
    storage_provider = gcs
    enabled = true
    storage_allowed_locations = ('gcs://snowflake069/emp/')

-   create or replace file format my_csv_unload_format
    type = csv field_delimiter = ',' skip_header = 1 null_if = ('NULL', 'null')
    empty_field_as_null = true compression = gzip;
```

- alter storage integration gcs_int set
  storage_allowed_locations=('gcs://snowflake069/emp/','gcs://snowflake069/
  emp_unload/')

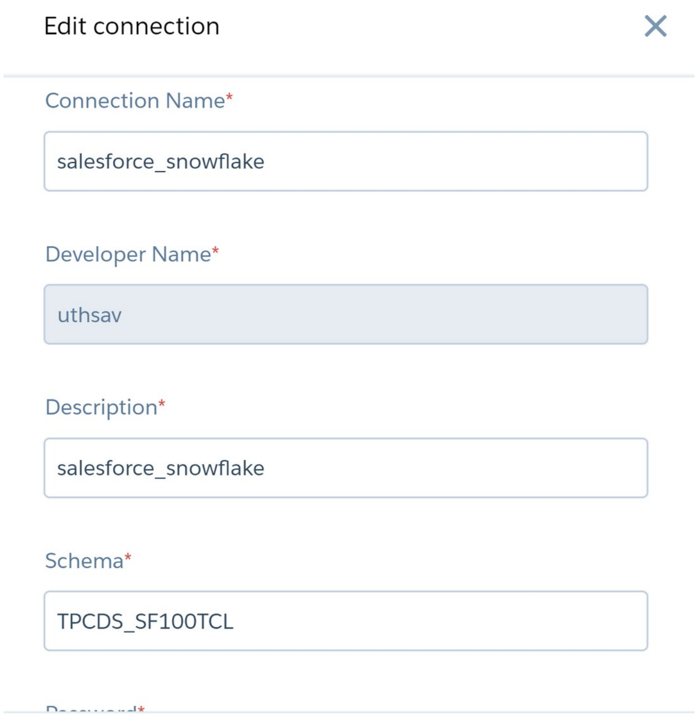
- create or replace stage my_gcs_unload_stage
  storage_integration = gcs_int
  url = 'gcs://snowflake069/emp_unload/'
   file_format = my_csv_unload_format;

- select * from emp_gcs_table

- copy into @my_gcs_unload_stage
  From emp_gcs_table

Einstein Analytics Snowflake Integration

1. Connection Setup
    - Go to Einstein Analytics Console -> Launch Analytics Studio -> Create Dataset -> External Dataset -> Open Snowflake Connector -> Follow these Instructions for setup



Edit connection

Connection Name*

salesforce_snowflake

Developer Name*

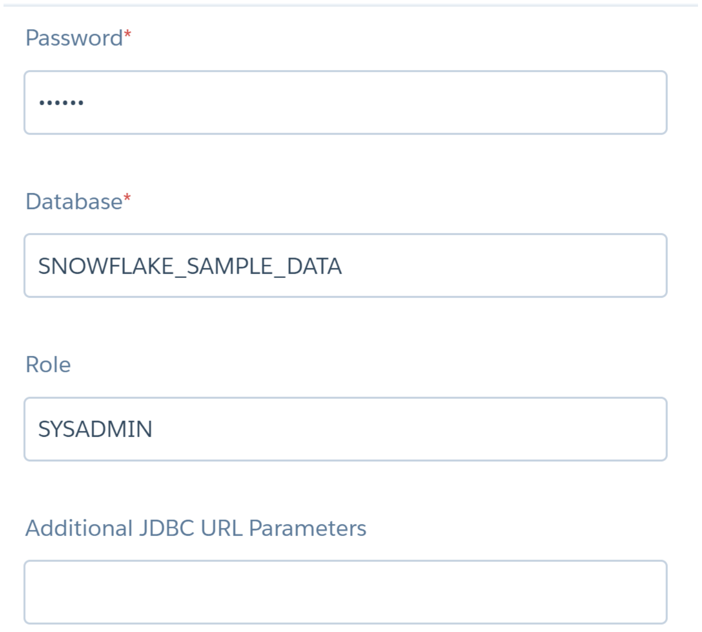
uthsav

Description*

salesforce_snowflake

Schema*

TPCDS_SF100TCL



Password*

••••••

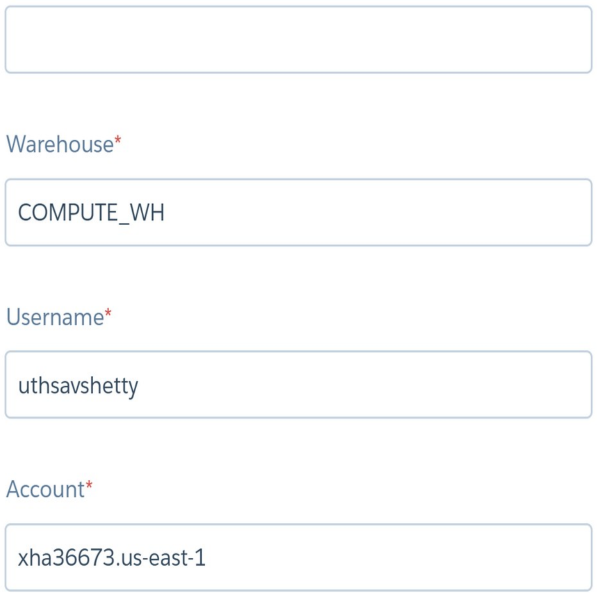
Database*

SNOWFLAKE_SAMPLE_DATA

Role

SYSADMIN

Additional JDBC URL Parameters



Warehouse*

COMPUTE_WH

Username*

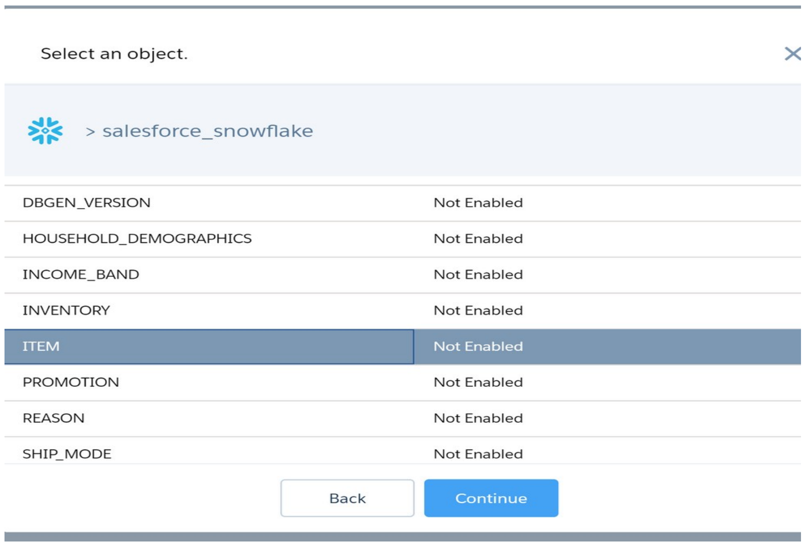
uthsavshetty

Account*

xha36673.us-east-1

- Password - Snowflake user password
- Account - will get it from snowflake URL
- Use database - SNOWFLAKE_SAMPLE_DATA
- Use Schema - TPCDS_SF100TCL
- Use Table - ITEM

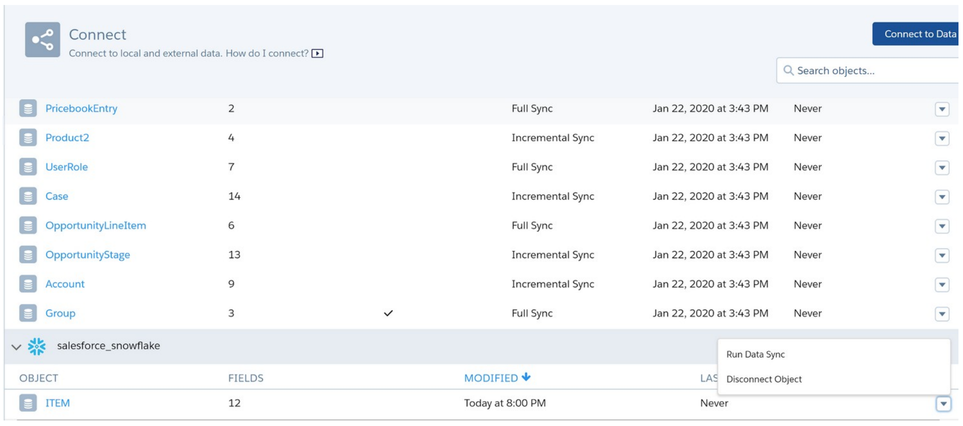
2. Steps

- 

Select an object.

> salesforce_snowflake

DBGEN_VERSION	Not Enabled
HOUSEHOLD_DEMOGRAPHICS	Not Enabled
INCOME_BAND	Not Enabled
INVENTORY	Not Enabled
ITEM	Not Enabled
PROMOTION	Not Enabled
REASON	Not Enabled
SHIP_MODE	Not Enabled



Connect

Connect to local and external data. How do I connect?

PricebookEntry	2		Full Sync	Jan 22, 2020 at 3:43 PM	Never
Product2	4		Incremental Sync	Jan 22, 2020 at 3:43 PM	Never
UserRole	7		Full Sync	Jan 22, 2020 at 3:43 PM	Never
Case	14		Incremental Sync	Jan 22, 2020 at 3:43 PM	Never
OpportunityLineItem	6		Full Sync	Jan 22, 2020 at 3:43 PM	Never
OpportunityStage	13		Incremental Sync	Jan 22, 2020 at 3:43 PM	Never
Account	9		Incremental Sync	Jan 22, 2020 at 3:43 PM	Never
Group	3	✓	Full Sync	Jan 22, 2020 at 3:43 PM	Never

salesforce_snowflake

OBJECT	FIELDS	MODIFIED	LAS
ITEM	12	Today at 8:00 PM	Never

References

https://docs.snowflake.net/manuals/index.html
https://docs.aws.amazon.com/
https://cloud.google.com/docs

Dataset

https://drive.google.com/drive/folders/1xOLeyXKf1E8YKLWoP8lgclJa2bQtIMPl?usp=sharing