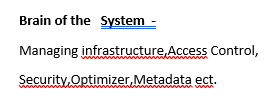
**Snewflake:**

**Snowflake Architecture:**

**Cloud Service **

**Query Processing** Virtual Virtual virtual -**Muscle of the system-**

Warehouse Warehouse Warehouse Performs MMP

(Massive parallel Processing)

**Storage -Hybrid Columnar Storage-**

Saved in blobs

**Virtual Warehouse Sizes:**

**Xs -1(server) -Billed by Second- with 1 minute as minimum**

**S -2(server) -Multi-Clustering(more queries)**

**M -4(servers)**

**L—8**

**XL-16**

**4XL-128**

**Multi clustering**

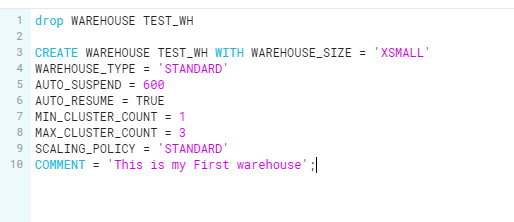
**S \* \***

**S \* \***

**S \* \***

**Create Warehouse**

CREATE WAREHOUSE TEST\_WH WITH WAREHOUSE\_SIZE = 'XSMALL' WAREHOUSE\_TYPE = 'STANDARD' AUTO\_SUSPEND = 600 AUTO\_RESUME = TRUE MIN\_CLUSTER\_COUNT = 1 MAX\_CLUSTER\_COUNT = 3 SCALING\_POLICY = 'STANDARD' COMMENT = 'This is my First warehouse';



**Scaling:**

**>Auto Scaling 🡪 When to start an additional cluster**

**Scalling**

**Standared (Default) Economy**

**(Favor starting additiona WH(SAWH)) (Favors Conserving credits rather then SAWH)**

**Standered:**

1.Immediately when either a query is queued or the system detects that there are more queries then can be executed by the currentlyavailable cluster.

2. After 2 to 3 consecutive successful checks(performed at 1 minute intervals),

Which determine whether the load on the least-loaded cluster could be redistributed to the other cluster

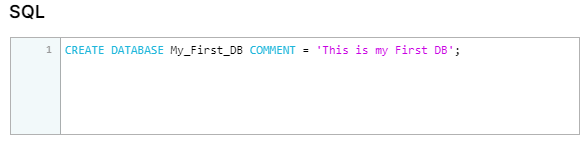
**Economy:**

1.Only if the system estimates there’s enough query load to keep the cluster busy for at least 6 minutes

2.After 5 to 6 consecutive successdul check

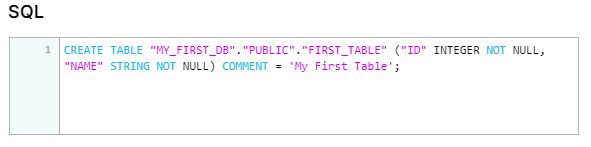
**Creationg DB:**

**CREATE DATABASE My\_First\_DB COMMENT = 'This is my First DB';**

****

**Creating the Table :**

**CREATE TABLE "MY\_FIRST\_DB"."PUBLIC"."FIRST\_TABLE" ("ID" INTEGER NOT NULL, "NAME" STRING NOT NULL) COMMENT = 'My First Table';**

****

**//Creating the table / Meta data**

**CREATE TABLE "OUR\_FIRST\_DB"."PUBLIC"."LOAN\_PAYMENT" (**

**"Loan\_ID" STRING,**

**"loan\_status" STRING,**

**"Principal" STRING,**

**"terms" STRING,**

**"effective\_date" STRING,**

**"due\_date" STRING,**

**"paid\_off\_time" STRING,**

**"past\_due\_days" STRING,**

**"age" STRING,**

**"education" STRING,**

**"Gender" STRING);**

**//Check that table is empy**

**USE DATABASE OUR\_FIRST\_DB;**

**SELECT \* FROM LOAN\_PAYMENT;**

**//Loading the data from S3 bucket**

**COPY INTO LOAN\_PAYMENT**

**FROM s3://bucketsnowflakes3/Loan\_payments\_data.csv**

**file\_format = (type = csv**

**field\_delimiter = ','**

**skip\_header=1);**

**//Validate**

**SELECT \* FROM LOAN\_PAYMENT;**

**Data WareHouse**

**HR data**

**ETL**

**Sales data**

**Data Warehouse**

**=Database that is used for reporting and data analysis**

**ETL=Extract , Transform & Load**

**Reporting**

**Different Layer**

**HR data data Science**

**Other apps**

**Sales data**

**Raw Data --🡪 data integration🡪 Access layer**

**Data Sources**

**Raw layer -- Staging Area**

**Data Integration --- Data Transformation**

**Cloud Computing**

**Data centre(infrastructure,security, electricity,software/hardware Upgrades)**

**Software-as-a Services**

**Application databases,tables,ect**

**|**

**Software Snowflake**

**| managing data storage,Virtual**

**Software as a service Data Warehouses,**

**| Upgrades/Metadata ect.**

**Operation System**

**|**

**Physical Servers**

**| Cloud Provider**

**Virtual Machines AWS,Azure,GCP**

**|**

**Physical Storage**

**GCP-Google Cloud Platform**

**Snowflack Editions**

**1.Standard - ----** introductory level

**2.Enterprise ---**additional features for the needs of large-scale enterprises (multi-cluster warehouse)

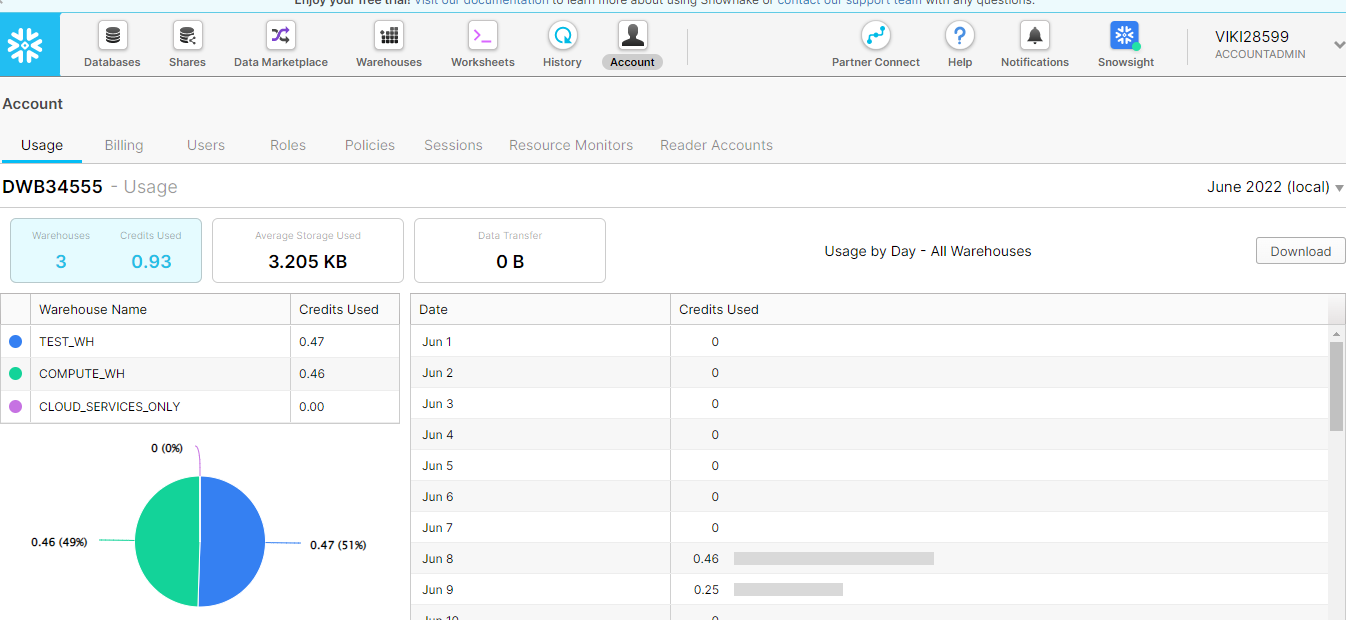
**3.Business Critical ---**even higher level of data protection for orrganizations with extremely sensitive data(Extended support)

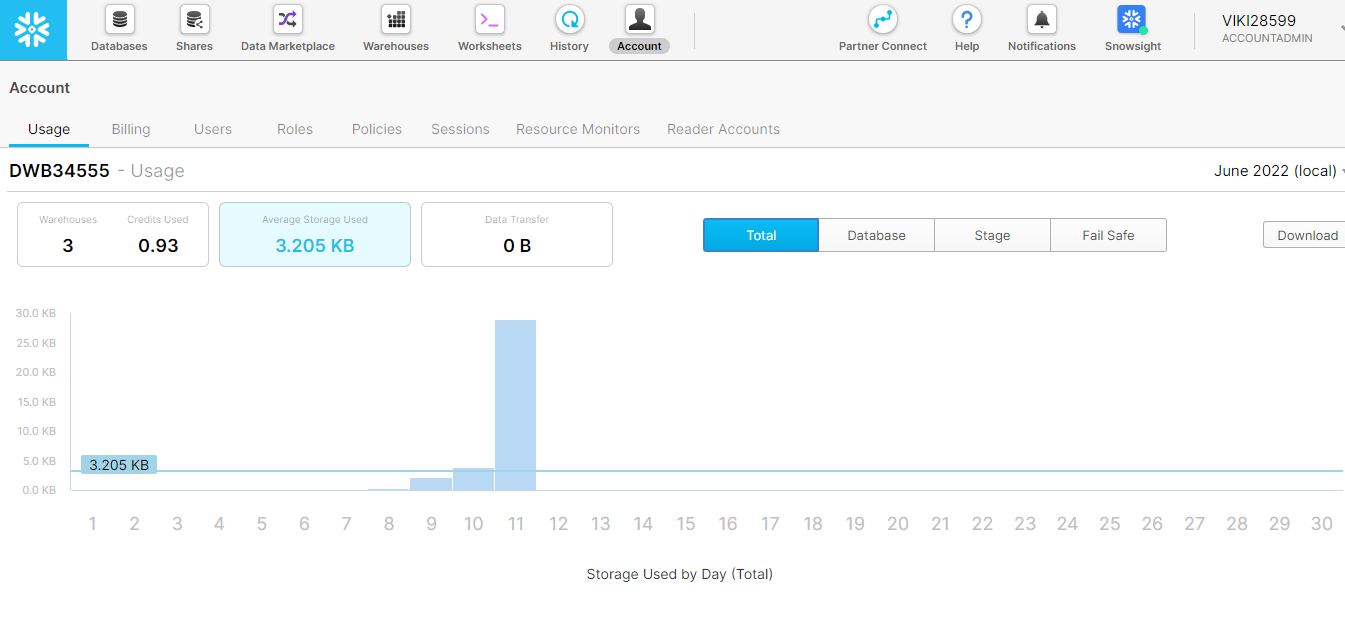
**4.Virtual Private –** Highest level of security

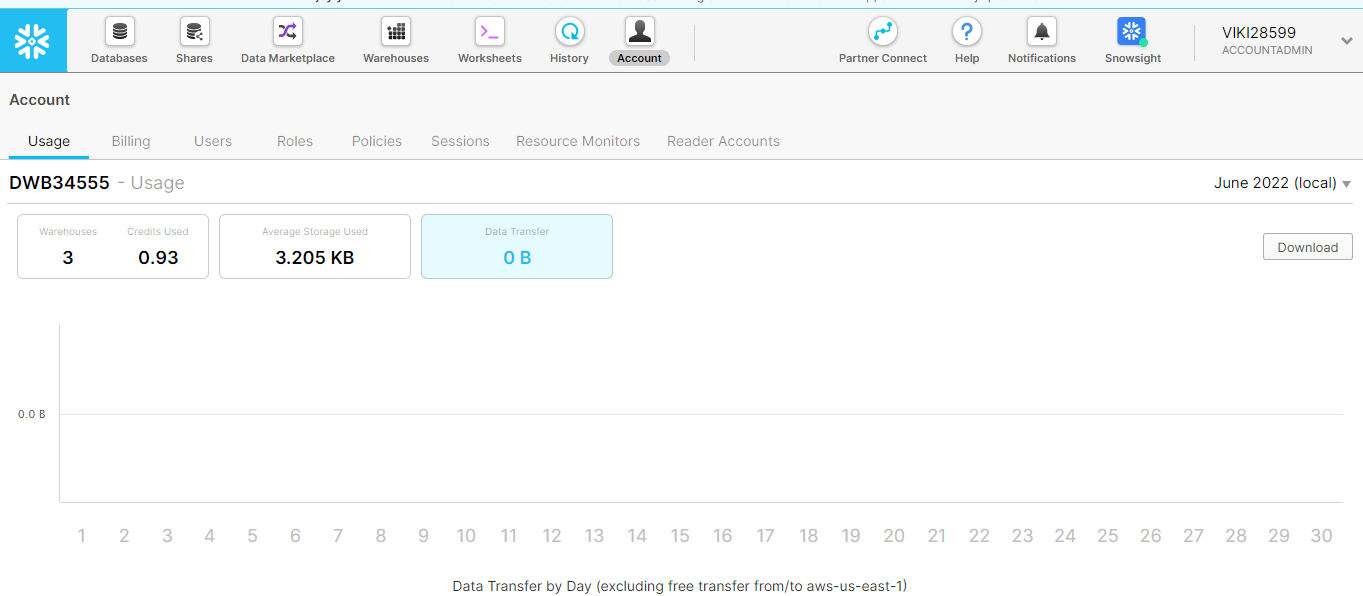
Snowflack pricking depends on region,Plotform,service, size

Compute cast

Storage cast -🡪 1.on Demand storage 2.Capacity Storage







SnowFlack Roles

ACCOUNTADMIN

SYSADMIN

SECURITYADMIN

Custom Role Custom Role

Custom Role 2

Custom Role 1

USERADMIN

Custom Role 3

PUBLIC

**ACCOUNTADIMN :**

**1.SYSADMIN and SECURITYADIMN**

**2.Top-levelrole in the system**

**3.should be granted only to a limites number of users**

**SECURITYADMIN:**

**1.USERADMINrole is granted to SECURITYADMIN**

**2.Can manage users and roles**

**3.Can manage any object grant globally**

**SYSADMIN:**

**1.create warehouses and databases(and more objects)**

**2.REcommented that all custom roles are assigned**

**USERADMIN:**

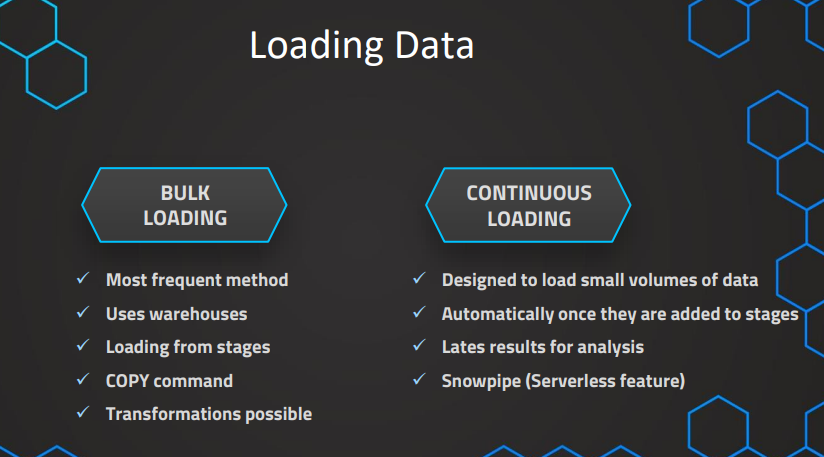
**1.Dedicated to user and role management only**

**2.Can create users and roles**

**PUBLIC:**

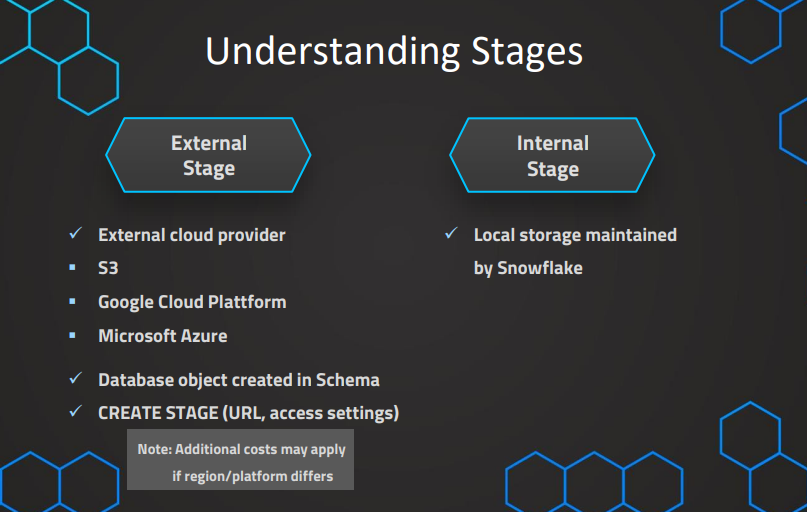
**1.Automatically granted to every user**

**2.can create own objects like every other role(available to every other user/role)**

****

**Snowpipe (Serverless feature)**

**Stages:**

****

// Database to manage stage objects, fileformats etc.

CREATE OR REPLACE DATABASE MANAGE\_DB;

CREATE OR REPLACE SCHEMA external\_stages;

// Creating external stage

CREATE OR REPLACE STAGE MANAGE\_DB.external\_stages.aws\_stage

url='s3://bucketsnowflakes3'

credentials=(aws\_key\_id='ABCD\_DUMMY\_ID' aws\_secret\_key='1234abcd\_key');

Copy Comment

// Creating ORDERS table

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS;

// First copy command

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS

FROM @aws\_stage

file\_format = (type = csv field\_delimiter=',' skip\_header=1);

// Copy command with fully qualified stage object

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS

FROM @MANAGE\_DB.external\_stages.aws\_stage

file\_format= (type = csv field\_delimiter=',' skip\_header=1);

// List files contained in stage

LIST @MANAGE\_DB.external\_stages.aws\_stage;

// Copy command with specified file(s)

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS

FROM @MANAGE\_DB.external\_stages.aws\_stage

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files = ('OrderDetails.csv');

// Copy command with pattern for file names

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS

FROM @MANAGE\_DB.external\_stages.aws\_stage

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*';

// Description of external stage

DESC STAGE MANAGE\_DB.external\_stages.aws\_stage;

// Alter external stage

ALTER STAGE aws\_stage

SET credentials=(aws\_key\_id='XYZ\_DUMMY\_ID' aws\_secret\_key='987xyz');

// Publicly accessible staging area

CREATE OR REPLACE STAGE MANAGE\_DB.external\_stages.aws\_stage

url='s3://bucketsnowflakes3';

// List files in stage

LIST @aws\_stage;

//Load data using copy command

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS

FROM @aws\_stage

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*';

Transforming Data

// Transforming using the SELECT statement

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM (select s.$1, s.$2 from @MANAGE\_DB.external\_stages.aws\_stage s)

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files=('OrderDetails.csv');

// Example 1 - Table

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT

)

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX;

// Example 2 - Table

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

PROFITABLE\_FLAG VARCHAR(30)

)

// Example 2 - Copy Command using a SQL function (subset of functions available)

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM (select

s.$1,

s.$2,

s.$3,

CASE WHEN CAST(s.$3 as int) < 0 THEN 'not profitable' ELSE 'profitable' END

from @MANAGE\_DB.external\_stages.aws\_stage s)

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files=('OrderDetails.csv');

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

// Example 3 - Table

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

CATEGORY\_SUBSTRING VARCHAR(5)

)

// Example 3 - Copy Command using a SQL function (subset of functions available)

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM (select

s.$1,

s.$2,

s.$3,

substring(s.$5,1,5)

from @MANAGE\_DB.external\_stages.aws\_stage s)

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files=('OrderDetails.csv');

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

//Example 3 - Table

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

PROFITABLE\_FLAG VARCHAR(30)

)

//Example 4 - Using subset of columns

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (ORDER\_ID,PROFIT)

FROM (select

s.$1,

s.$3

from @MANAGE\_DB.external\_stages.aws\_stage s)

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files=('OrderDetails.csv');

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX;

//Example 5 - Table Auto increment

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID number autoincrement start 1 increment 1,

AMOUNT INT,

PROFIT INT,

PROFITABLE\_FLAG VARCHAR(30)

)

//Example 5 - Auto increment ID

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (PROFIT,AMOUNT)

FROM (select

s.$2,

s.$3

from @MANAGE\_DB.external\_stages.aws\_stage s)

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files=('OrderDetails.csv');

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX WHERE ORDER\_ID > 15;

DROP TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

// Create new stage

CREATE OR REPLACE STAGE MANAGE\_DB.external\_stages.aws\_stage\_errorex

url='s3://bucketsnowflakes4'

// List files in stage

LIST @MANAGE\_DB.external\_stages.aws\_stage\_errorex;

// Create example table

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Demonstrating error message

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files = ('OrderDetails\_error.csv');

// Validating table is empty

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

// Error handling using the ON\_ERROR option

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files = ('OrderDetails\_error.csv')

ON\_ERROR = 'CONTINUE';

// Validating results and truncating table

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

SELECT COUNT(\*) FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

TRUNCATE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX;

// Error handling using the ON\_ERROR option = ABORT\_STATEMENT (default)

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files = ('OrderDetails\_error.csv','OrderDetails\_error2.csv')

ON\_ERROR = 'ABORT\_STATEMENT';

// Validating results and truncating table

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

SELECT COUNT(\*) FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

TRUNCATE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX;

// Error handling using the ON\_ERROR option = SKIP\_FILE

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files = ('OrderDetails\_error.csv','OrderDetails\_error2.csv')

ON\_ERROR = 'SKIP\_FILE';

// Validating results and truncating table

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

SELECT COUNT(\*) FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

TRUNCATE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX;

Copy Option ONERROR

// Error handling using the ON\_ERROR option = SKIP\_FILE\_<number>

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files = ('OrderDetails\_error.csv','OrderDetails\_error2.csv')

ON\_ERROR = 'SKIP\_FILE\_2';

// Validating results and truncating table

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

SELECT COUNT(\*) FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

TRUNCATE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX;

// Error handling using the ON\_ERROR option = SKIP\_FILE\_<number>

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files = ('OrderDetails\_error.csv','OrderDetails\_error2.csv')

ON\_ERROR = 'SKIP\_FILE\_0.5%';

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files = ('OrderDetails\_error.csv','OrderDetails\_error2.csv')

ON\_ERROR = SKIP\_FILE\_3

SIZE\_LIMIT = 30;

File Formater

// Specifying file\_format in Copy command

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format = (type = csv field\_delimiter=',' skip\_header=1)

files = ('OrderDetails\_error.csv')

ON\_ERROR = 'SKIP\_FILE\_3';

// Creating table

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Creating schema to keep things organized

CREATE OR REPLACE SCHEMA MANAGE\_DB.file\_formats;

// Creating file format object

CREATE OR REPLACE file format MANAGE\_DB.file\_formats.my\_file\_format;

// See properties of file format object

DESC file format MANAGE\_DB.file\_formats.my\_file\_format;

// Using file format object in Copy command

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format= (FORMAT\_NAME=MANAGE\_DB.file\_formats.my\_file\_format)

files = ('OrderDetails\_error.csv')

ON\_ERROR = 'SKIP\_FILE\_3';

// Altering file format object

ALTER file format MANAGE\_DB.file\_formats.my\_file\_format

SET SKIP\_HEADER = 1;

// Defining properties on creation of file format object

CREATE OR REPLACE file format MANAGE\_DB.file\_formats.my\_file\_format

TYPE=JSON,

TIME\_FORMAT=AUTO;

// See properties of file format object

DESC file format MANAGE\_DB.file\_formats.my\_file\_format;

// Using file format object in Copy command

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format= (FORMAT\_NAME=MANAGE\_DB.file\_formats.my\_file\_format)

files = ('OrderDetails\_error.csv')

ON\_ERROR = 'SKIP\_FILE\_3';

// Altering the type of a file format is not possible

ALTER file format MANAGE\_DB.file\_formats.my\_file\_format

SET TYPE = CSV;

// Recreate file format (default = CSV)

CREATE OR REPLACE file format MANAGE\_DB.file\_formats.my\_file\_format

// See properties of file format object

DESC file format MANAGE\_DB.file\_formats.my\_file\_format;

// Truncate table

TRUNCATE table OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX;

// Overwriting properties of file format object

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

file\_format = (FORMAT\_NAME= MANAGE\_DB.file\_formats.my\_file\_format field\_delimiter = ',' skip\_header=1 )

files = ('OrderDetails\_error.csv')

ON\_ERROR = 'SKIP\_FILE\_3';

DESC STAGE MANAGE\_DB.external\_stages.aws\_stage\_errorex;

---- VALIDATION\_MODE ----

// Prepare database & table

CREATE OR REPLACE DATABASE COPY\_DB;

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Prepare stage object

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/size/';

LIST @COPY\_DB.PUBLIC.aws\_stage\_copy;

//Load data using copy command

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

VALIDATION\_MODE = RETURN\_ERRORS

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

VALIDATION\_MODE = RETURN\_5\_ROWS

Rejected\_error

---- Use files with errors ----

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/returnfailed/';

LIST @COPY\_DB.PUBLIC.aws\_stage\_copy;

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

VALIDATION\_MODE = RETURN\_ERRORS

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

VALIDATION\_MODE = RETURN\_1\_rows

-------------- Working with error results -----------

---- 1) Saving rejected files after VALIDATION\_MODE ----

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

VALIDATION\_MODE = RETURN\_ERRORS;

// Storing rejected /failed results in a table

CREATE OR REPLACE TABLE rejected AS

select rejected\_record from table(result\_scan(last\_query\_id()));

INSERT INTO rejected

select rejected\_record from table(result\_scan(last\_query\_id()));

SELECT \* FROM rejected;

---- 2) Saving rejected files without VALIDATION\_MODE ----

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

ON\_ERROR=CONTINUE

select \* from table(validate(orders, job\_id => '\_last'));

---- 3) Working with rejected records ----

SELECT REJECTED\_RECORD FROM rejected;

CREATE OR REPLACE TABLE rejected\_values as

SELECT

SPLIT\_PART(rejected\_record,',',1) as ORDER\_ID,

SPLIT\_PART(rejected\_record,',',2) as AMOUNT,

SPLIT\_PART(rejected\_record,',',3) as PROFIT,

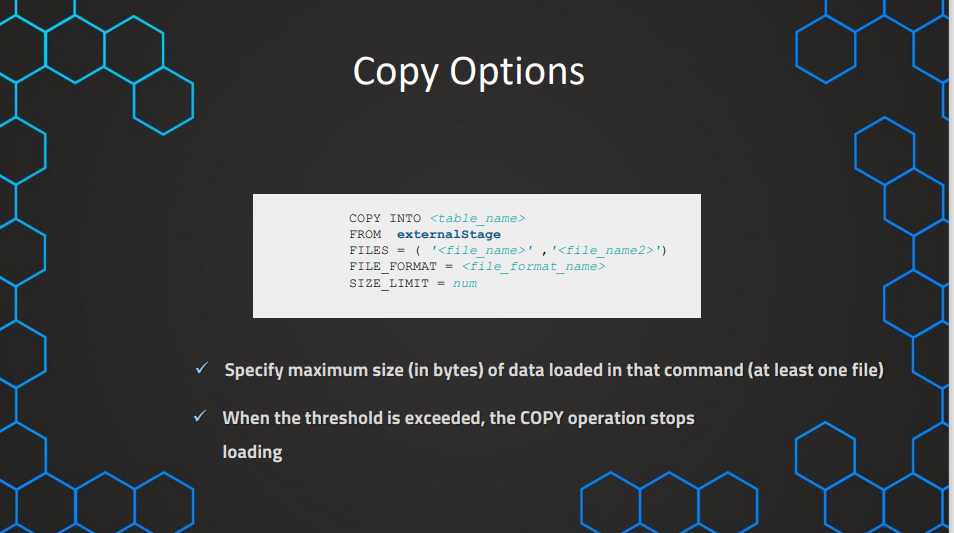
SPLIT\_PART(rejected\_record,',',4) as QUATNTITY,

SPLIT\_PART(rejected\_record,',',5) as CATEGORY,

SPLIT\_PART(rejected\_record,',',6) as SUBCATEGORY

FROM rejected;

SELECT \* FROM rejected\_values;



---- SIZE\_LIMIT ----

// Prepare database & table

CREATE OR REPLACE DATABASE COPY\_DB;

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Prepare stage object

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/size/';

// List files in stage

LIST @aws\_stage\_copy;

//Load data using copy command

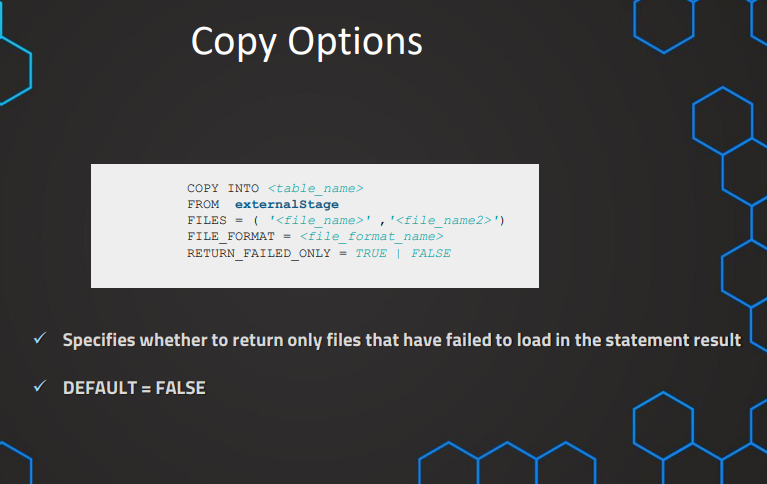
COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

SIZE\_LIMIT=20000;



---- RETURN\_FAILED\_ONLY ----

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Prepare stage object

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/returnfailed/';

LIST @COPY\_DB.PUBLIC.aws\_stage\_copy

//Load data using copy command

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

RETURN\_FAILED\_ONLY = TRUE

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

ON\_ERROR =CONTINUE

RETURN\_FAILED\_ONLY = TRUE

// Default = FALSE

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

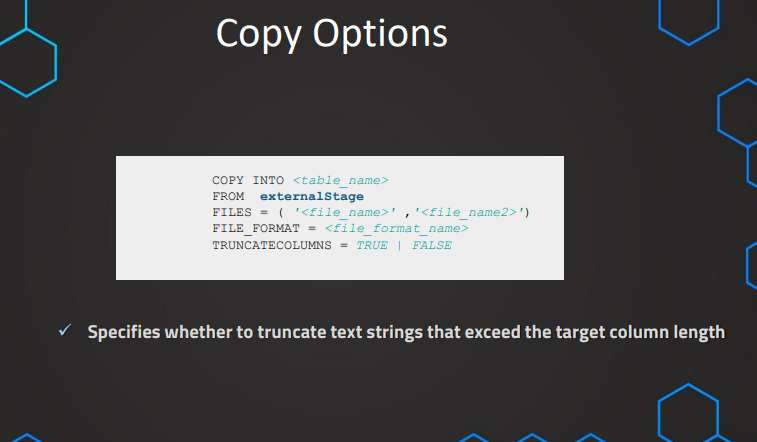
COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

ON\_ERROR =CONTINUE



---- TRUNCATECOLUMNS ----

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(10),

SUBCATEGORY VARCHAR(30));

// Prepare stage object

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/size/';

LIST @COPY\_DB.PUBLIC.aws\_stage\_copy

//Load data using copy command

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

TRUNCATECOLUMNS = true;

SELECT \* FROM ORDERS;



---- FORCE ----

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Prepare stage object

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/size/';

LIST @COPY\_DB.PUBLIC.aws\_stage\_copy

//Load data using copy command

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

// Not possible to load file that have been loaded and data has not been modified

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

SELECT \* FROM ORDERS;

// Using the FORCE option

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

FORCE = TRUE;

Load History

-- Query load history within a database --

USE COPY\_DB;

SELECT \* FROM information\_schema.load\_history

-- Query load history gloabally from SNOWFLAKE database --

SELECT \* FROM snowflake.account\_usage.load\_history

// Filter on specific table & schema

SELECT \* FROM snowflake.account\_usage.load\_history

where schema\_name='PUBLIC' and

table\_name='ORDERS'

// Filter on specific table & schema

SELECT \* FROM snowflake.account\_usage.load\_history

where schema\_name='PUBLIC' and

table\_name='ORDERS' and

error\_count > 0

// Filter on specific table & schema

SELECT \* FROM snowflake.account\_usage.load\_history

WHERE DATE(LAST\_LOAD\_TIME) <= DATEADD(days,-1,CURRENT\_DATE)

Assignment

**1. Create a table called employees with the following columns and data types:**

**customer\_id** int,

**first\_name** varchar(50),

**last\_name**varchar(50),

**email**varchar(50),

**age**int,

**department** varchar(50)

**2. Create a stage object pointing to**'s3://snowflake-assignments-mc/copyoptions/example2'

**3. Create a file format object with the specification**

TYPE = CSV

FIELD\_DELIMITER=','

SKIP\_HEADER=1;

**4. Use the copy option to only validate if there are errors and if yes what errors.**

**5. One value in the first\_name column has more than 50 characters. We assume the table column properties could not be changed.**

**What option could you use to load that record anyways and just truncate the value after 50 characters?**

**Load the data in the table using that option.**

**Questions for this assignment**

How many rows have been loaded?

Ans :

-- Create table

create or replace table employees(

customer\_id int,

first\_name varchar(50),

last\_name varchar(50),

email varchar(50),

age int,

department varchar(50));

-- create stage object

CREATE OR REPLACE STAGE EXERCISE\_DB.public.aws\_stage

url='s3://snowflake-assignments-mc/copyoptions/example2';

-- create file format object

CREATE OR REPLACE FILE FORMAT EXERCISE\_DB.public.aws\_fileformat

TYPE = CSV

FIELD\_DELIMITER=','

SKIP\_HEADER=1;

-- Use validation mode

COPY INTO EXERCISE\_DB.PUBLIC.EMPLOYEES

FROM @aws\_stage

file\_format= EXERCISE\_DB.public.aws\_fileformat

VALIDATION\_MODE = RETURN\_ERRORS;

-- Use TRUNCATECOLUMNS

COPY INTO EXERCISE\_DB.PUBLIC.EMPLOYEES

FROM @aws\_stage

file\_format= EXERCISE\_DB.public.aws\_fileformat

TRUNCATECOLUMNS = TRUE;

Load Unstructured data

Create Stage 🡪

Load Raw data 🡪

Analyse & Parse 🡪

Flatten & Load

Our data:

Json:

{

“Key”: “Value”

}

// First step: Load Raw JSON

CREATE OR REPLACE stage MANAGE\_DB.EXTERNAL\_STAGES.JSONSTAGE

url='s3://bucketsnowflake-jsondemo';

CREATE OR REPLACE file format MANAGE\_DB.FILE\_FORMATS.JSONFORMAT

TYPE = JSON;

CREATE OR REPLACE table OUR\_FIRST\_DB.PUBLIC.JSON\_RAW (

raw\_file variant);

COPY INTO OUR\_FIRST\_DB.PUBLIC.JSON\_RAW

FROM @MANAGE\_DB.EXTERNAL\_STAGES.JSONSTAGE

file\_format= MANAGE\_DB.FILE\_FORMATS.JSONFORMAT

files = ('HR\_data.json');

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

// Second step: Parse & Analyse Raw JSON

// Selecting attribute/column

SELECT RAW\_FILE:city FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW

SELECT $1:first\_name FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW

// Selecting attribute/column - formattted

SELECT RAW\_FILE:first\_name::string as first\_name FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

SELECT RAW\_FILE:id::int as id FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

SELECT

RAW\_FILE:id::int as id,

RAW\_FILE:first\_name::STRING as first\_name,

RAW\_FILE:last\_name::STRING as last\_name,

RAW\_FILE:gender::STRING as gender

FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

// Handling nested data

SELECT RAW\_FILE:job as job FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

// Handling nested data

SELECT RAW\_FILE:job as job FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

SELECT

RAW\_FILE:job.salary::INT as salary

FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

SELECT

RAW\_FILE:first\_name::STRING as first\_name,

RAW\_FILE:job.salary::INT as salary,

RAW\_FILE:job.title::STRING as title

FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

// Handling arreys

SELECT

RAW\_FILE:prev\_company as prev\_company

FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

SELECT

RAW\_FILE:prev\_company[1]::STRING as prev\_company

FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

SELECT

ARRAY\_SIZE(RAW\_FILE:prev\_company) as prev\_company

FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW;

SELECT

RAW\_FILE:id::int as id,

RAW\_FILE:first\_name::STRING as first\_name,

RAW\_FILE:prev\_company[0]::STRING as prev\_company

FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW

UNION ALL

SELECT

RAW\_FILE:id::int as id,

RAW\_FILE:first\_name::STRING as first\_name,

RAW\_FILE:prev\_company[1]::STRING as prev\_company

FROM OUR\_FIRST\_DB.PUBLIC.JSON\_RAW

ORDER BY id