**Storing Data.**

**// CREATE Database & Schema to manage stage objects & file formats etc.**

CREATE OR REPLACE DATABASE MANAGE\_DB;

USE 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');

**// 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');

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

**// Publicly accessible staging area**

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

url='s3://bucketsnowflakes3';

**// List files in stage**

LIST @aws\_stage;

**// Create Database & Schema to store data into it for further use.**

CREATE DATABASE OUR\_FIRST\_DB;

USE DATABASE OUR\_FIRST\_DB;

CREATE OR REPLACE SCHEMA PUBLIC;

**//Load data using copy command**

**// 1st Create 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.\*';

**// Transforming using the SELECT statement**

**// Example 1 - Table**

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

    ORDER\_ID VARCHAR(30),

    AMOUNT INT

    )

**// Example 1 - Copy Command using a SQL function**

   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');

   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 4 - 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

**About Error Message.**

**// 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;

**// 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';

**// ( ON\_ERROR = ' SKIP\_FILE\_N ' ) Here N means if we have more than N errors in file then skip that 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;

**// 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%';

**// ( ON\_ERROR = ' SKIP\_FILE\_N% ' ) Here N% means if we have more than N% errors in file then skip that 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;

**// 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));

**About** **File Format**

**// Creating Separate schema to keep things organized for file format.**

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;

**// Store the data 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;

**//Again Run the following 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';

**// 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;

**// 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));

**// Creat new, 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

**---- 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 ----**

**// Create New 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));

**// Creat new, 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));

**// Creat new, 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

**---- 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));

**// Creat new, 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

**---- 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;

**-- 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)

**// First step: Load Raw JSON**

**Create Database & Schema**

Create database manage\_db;

CREATE OR REPLACE SCHEMA MANAGE\_DB.EXTERNAL\_STAGES;

CREATE OR REPLACE SCHEMA MANAGE\_DB.FILE\_FORMATS;

**Create new,Stage object**

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

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

**Create new, file format object**

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

 TYPE = JSON;

**Create new Table**

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

**Note city is a attribute in the data and raw\_file is the column**

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;

 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;

**Note: job.salary means salary is in array { } in the job attribute**

**// Handling arrays**

SELECT

    RAW\_FILE:prev\_company as prev\_company

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

SELECT

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

    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

**To get how many no of companies an emp work previously.**

SELECT

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

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

SELECT

    RAW\_FILE:spoken\_languages as spoken\_languages

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

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

SELECT

     array\_size(RAW\_FILE:spoken\_languages) as spoken\_languages

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

SELECT

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

     array\_size(RAW\_FILE:spoken\_languages) as spoken\_languages

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

Note: array\_size is used to count the number of langues spoken

SELECT

    RAW\_FILE:spoken\_languages[0] as First\_language

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

SELECT

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

    RAW\_FILE:spoken\_languages[0] as First\_language

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

SELECT

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

    RAW\_FILE:spoken\_languages[0].language::STRING as First\_language,

    RAW\_FILE:spoken\_languages[0].level::STRING as Level\_spoken

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

SELECT

    RAW\_FILE:id::int as id,

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

    RAW\_FILE:spoken\_languages[0].language::STRING as First\_language,

    RAW\_FILE:spoken\_languages[0].level::STRING as Level\_spoken

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:spoken\_languages[1].language::STRING as First\_language,

    RAW\_FILE:spoken\_languages[1].level::STRING as Level\_spoken

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:spoken\_languages[2].language::STRING as First\_language,

    RAW\_FILE:spoken\_languages[2].level::STRING as Level\_spoken

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

ORDER BY ID

SELECT

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

    f.value:language::STRING as First\_language,

   f.value:level::STRING as Level\_spoken

from OUR\_FIRST\_DB.PUBLIC.JSON\_RAW, table(flatten(RAW\_FILE:spoken\_languages)) f;

**// Option 1: CREATE TABLE AS**

CREATE OR REPLACE TABLE Languages AS

select

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

f.value:language::STRING as First\_language,

f.value:level::STRING as Level\_spoken

from OUR\_FIRST\_DB.PUBLIC.JSON\_RAW, table(flatten(RAW\_FILE:spoken\_languages)) f;

SELECT \* FROM Languages;

truncate table languages;

**// Option 2: INSERT INTO**

INSERT INTO Languages

select

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

f.value:language::STRING as First\_language,

f.value:level::STRING as Level\_spoken

from OUR\_FIRST\_DB.PUBLIC.JSON\_RAW, table(flatten(RAW\_FILE:spoken\_languages)) f;

SELECT \* FROM Languages;

**Dealing with Parquet data.**

**// Create file format and stage object**

CREATE OR REPLACE FILE FORMAT MANAGE\_DB.FILE\_FORMATS.PARQUET\_FORMAT

    TYPE = 'parquet';

CREATE OR REPLACE STAGE MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE

    url = 's3://snowflakeparquetdemo'

    FILE\_FORMAT = MANAGE\_DB.FILE\_FORMATS.PARQUET\_FORMAT;

**// Preview the data**

LIST  @MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE;

SELECT \* FROM @MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE;

**// File format in Queries**

CREATE OR REPLACE STAGE MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE

    url = 's3://snowflakeparquetdemo'

SELECT \*

FROM @MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE

(file\_format => 'MANAGE\_DB.FILE\_FORMATS.PARQUET\_FORMAT')

**// Quotes can be omitted in case of the current namespace**

USE MANAGE\_DB.FILE\_FORMATS;

SELECT \*

FROM @MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE

(file\_format => MANAGE\_DB.FILE\_FORMATS.PARQUET\_FORMAT)

CREATE OR REPLACE STAGE MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE

    url = 's3://snowflakeparquetdemo'

    FILE\_FORMAT = MANAGE\_DB.FILE\_FORMATS.PARQUET\_FORMAT;

**// Syntax for Querying unstructured data**

SELECT

$1:\_\_index\_level\_0\_\_,

$1:cat\_id,

$1:date,

$1:"\_\_index\_level\_0\_\_",

$1:"cat\_id",

$1:"d",

$1:"date",

$1:"dept\_id",

$1:"id",

$1:"item\_id",

$1:"state\_id",

$1:"store\_id",

$1:"value"

FROM @MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE;

**// Date conversion**

SELECT 1;

SELECT DATE(365\*60\*60\*24);

**// Querying with conversions and aliases**

SELECT

$1:\_\_index\_level\_0\_\_::int as index\_level,

$1:cat\_id::VARCHAR(50) as category,

DATE($1:date::int ) as Date,

$1:"dept\_id"::VARCHAR(50) as Dept\_ID,

$1:"id"::VARCHAR(50) as ID,

$1:"item\_id"::VARCHAR(50) as Item\_ID,

$1:"state\_id"::VARCHAR(50) as State\_ID,

$1:"store\_id"::VARCHAR(50) as Store\_ID,

$1:"value"::int as value

FROM @MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE;

**// Adding metadata**

SELECT

$1:\_\_index\_level\_0\_\_::int as index\_level,

$1:cat\_id::VARCHAR(50) as category,

DATE($1:date::int ) as Date,

$1:"dept\_id"::VARCHAR(50) as Dept\_ID,

$1:"id"::VARCHAR(50) as ID,

$1:"item\_id"::VARCHAR(50) as Item\_ID,

$1:"state\_id"::VARCHAR(50) as State\_ID,

$1:"store\_id"::VARCHAR(50) as Store\_ID,

$1:"value"::int as value,

METADATA$FILENAME as FILENAME,

METADATA$FILE\_ROW\_NUMBER as ROWNUMBER,

TO\_TIMESTAMP\_NTZ(current\_timestamp) as LOAD\_DATE

FROM @MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE;

SELECT TO\_TIMESTAMP\_NTZ(current\_timestamp)

**// Create destination table**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.PARQUET\_DATA (

ROW\_NUMBER int,

index\_level int,

cat\_id VARCHAR(50),

date date,

dept\_id VARCHAR(50),

id VARCHAR(50),

item\_id VARCHAR(50),

state\_id VARCHAR(50),

store\_id VARCHAR(50),

value int,

Load\_date timestamp default TO\_TIMESTAMP\_NTZ(current\_timestamp))

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.PARQUET\_DATA

**// Load the parquet data**

COPY INTO OUR\_FIRST\_DB.PUBLIC.PARQUET\_DATA

FROM (SELECT

METADATA$FILE\_ROW\_NUMBER,

$1:\_\_index\_level\_0\_\_::int,

$1:cat\_id::VARCHAR(50),

DATE($1:date::int ),

$1:"dept\_id"::VARCHAR(50),

$1:"id"::VARCHAR(50),

$1:"item\_id"::VARCHAR(50),

$1:"state\_id"::VARCHAR(50),

$1:"store\_id"::VARCHAR(50),

$1:"value"::int,

TO\_TIMESTAMP\_NTZ(current\_timestamp)

FROM @MANAGE\_DB.EXTERNAL\_STAGES.PARQUETSTAGE);

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.PARQUET\_DATA;

**//  Create virtual warehouse for data scientist & DBA**

**// Data Scientists**

CREATE WAREHOUSE DS\_WH

WITH WAREHOUSE\_SIZE = 'SMALL'

WAREHOUSE\_TYPE = 'STANDARD'

AUTO\_SUSPEND = 300

AUTO\_RESUME = TRUE

MIN\_CLUSTER\_COUNT = 1

MAX\_CLUSTER\_COUNT = 1

SCALING\_POLICY = 'STANDARD';

**// DBA**

CREATE WAREHOUSE DBA\_WH

WITH WAREHOUSE\_SIZE = 'XSMALL'

WAREHOUSE\_TYPE = 'STANDARD'

AUTO\_SUSPEND = 300

AUTO\_RESUME = TRUE

MIN\_CLUSTER\_COUNT = 1

MAX\_CLUSTER\_COUNT = 1

SCALING\_POLICY = 'STANDARD';

**// Create role for Data Scientists & DBAs**

CREATE ROLE DATA\_SCIENTIST;

GRANT USAGE ON WAREHOUSE DS\_WH TO ROLE DATA\_SCIENTIST;

CREATE ROLE DBA;

GRANT USAGE ON WAREHOUSE DBA\_WH TO ROLE DBA;

**// Setting up users with roles**

**// Data Scientists**

CREATE USER DS1 PASSWORD = 'DS1' LOGIN\_NAME = 'DS1' DEFAULT\_ROLE='DATA\_SCIENTIST' DEFAULT\_WAREHOUSE = 'DS\_WH'  MUST\_CHANGE\_PASSWORD = FALSE;

CREATE USER DS2 PASSWORD = 'DS2' LOGIN\_NAME = 'DS2' DEFAULT\_ROLE='DATA\_SCIENTIST' DEFAULT\_WAREHOUSE = 'DS\_WH'  MUST\_CHANGE\_PASSWORD = FALSE;

CREATE USER DS3 PASSWORD = 'DS3' LOGIN\_NAME = 'DS3' DEFAULT\_ROLE='DATA\_SCIENTIST' DEFAULT\_WAREHOUSE = 'DS\_WH'  MUST\_CHANGE\_PASSWORD = FALSE;

GRANT ROLE DATA\_SCIENTIST TO USER DS1;

GRANT ROLE DATA\_SCIENTIST TO USER DS2;

GRANT ROLE DATA\_SCIENTIST TO USER DS3;

**// DBAs**

CREATE USER DBA1 PASSWORD = 'DBA1' LOGIN\_NAME = 'DBA1' DEFAULT\_ROLE='DBA' DEFAULT\_WAREHOUSE = 'DBA\_WH'  MUST\_CHANGE\_PASSWORD = FALSE;

CREATE USER DBA2 PASSWORD = 'DBA2' LOGIN\_NAME = 'DBA2' DEFAULT\_ROLE='DBA' DEFAULT\_WAREHOUSE = 'DBA\_WH'  MUST\_CHANGE\_PASSWORD = FALSE;

GRANT ROLE DBA TO USER DBA1;

GRANT ROLE DBA TO USER DBA2;

**// Drop objects again**

DROP USER DBA1;

DROP USER DBA2;

DROP USER DS1;

DROP USER DS2;

DROP USER DS3;

DROP ROLE DATA\_SCIENTIST;

DROP ROLE DBA;

DROP WAREHOUSE DS\_WH;

DROP WAREHOUSE DBA\_WH;

**//------------------------------------------------------------------------------------------------------------------------------**

SELECT \* FROM SNOWFLAKE\_SAMPLE\_DATA.TPCDS\_SF100TCL.WEB\_SITE T1

CROSS JOIN SNOWFLAKE\_SAMPLE\_DATA.TPCDS\_SF100TCL.WEB\_SITE T2

CROSS JOIN SNOWFLAKE\_SAMPLE\_DATA.TPCDS\_SF100TCL.WEB\_SITE T3

CROSS JOIN (SELECT TOP 57 \* FROM SNOWFLAKE\_SAMPLE\_DATA.TPCDS\_SF100TCL.WEB\_SITE) T4

**Caching**

SELECT AVG(C\_BIRTH\_YEAR) FROM SNOWFLAKE\_SAMPLE\_DATA.TPCDS\_SF100TCL.CUSTOMER

**// Setting up an additional user**

CREATE ROLE DATA\_SCIENTIST;

GRANT USAGE ON WAREHOUSE COMPUTE\_WH TO ROLE DATA\_SCIENTIST;

CREATE USER DS1 PASSWORD = 'DS1' LOGIN\_NAME = 'DS1' DEFAULT\_ROLE='DATA\_SCIENTIST' DEFAULT\_WAREHOUSE = 'DS\_WH' MUST\_CHANGE\_PASSWORD = FALSE;

GRANT ROLE DATA\_SCIENTIST TO USER DS1;

**Creating Cluster**

**// Publicly accessible staging area**

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

    url='s3://bucketsnowflakes3';

**// List files in stage**

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

**//Load data using copy command**

COPY INTO OUR\_FIRST\_DB.PUBLIC.ORDERS

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

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

    pattern='.\*OrderDetails.\*';

**// Create table**

CREATE OR REPLACE TABLE ORDERS\_CACHING (

ORDER\_ID VARCHAR(30)

,AMOUNT NUMBER(38,0)

,PROFIT NUMBER(38,0)

,QUANTITY NUMBER(38,0)

,CATEGORY VARCHAR(30)

,SUBCATEGORY VARCHAR(30)

,DATE DATE)

INSERT INTO ORDERS\_CACHING

SELECT

t1.ORDER\_ID

,t1.AMOUNT

,t1.PROFIT

,t1.QUANTITY

,t1.CATEGORY

,t1.SUBCATEGORY

,DATE(UNIFORM(1500000000,1700000000,(RANDOM())))

FROM ORDERS t1

CROSS JOIN (SELECT \* FROM ORDERS) t2

CROSS JOIN (SELECT TOP 100 \* FROM ORDERS) t3

**// Query Performance before Cluster Key**

SELECT \* FROM ORDERS\_CACHING  WHERE DATE = '2020-06-09'

**// Adding Cluster Key & Compare the result**

ALTER TABLE ORDERS\_CACHING CLUSTER BY ( DATE )

SELECT \* FROM ORDERS\_CACHING  WHERE DATE = '2020-01-05'

**// Not ideal clustering & adding a different Cluster Key using function**

SELECT \* FROM ORDERS\_CACHING  WHERE MONTH(DATE)=11

ALTER TABLE ORDERS\_CACHING CLUSTER BY ( MONTH(DATE) )

**## Connection of Snowflake with AWS (S3 Bucket)**

**// Create storage integration object**

create or replace storage integration s3\_int

  TYPE = EXTERNAL\_STAGE

  STORAGE\_PROVIDER = S3

  ENABLED = TRUE

  STORAGE\_AWS\_ROLE\_ARN = ''

  STORAGE\_ALLOWED\_LOCATIONS = ('s3://<your-bucket-name>/<your-path>/', 's3://<your-bucket-name>/<your-path>/')

   COMMENT = 'This an optional comment'

**// See storage integration properties to fetch external\_id so we can update it in S3**

DESC integration s3\_int;

**// Create table first**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.movie\_titles (

  show\_id STRING,

  type STRING,

  title STRING,

  director STRING,

  cast STRING,

  country STRING,

  date\_added STRING,

  release\_year STRING,

  rating STRING,

  duration STRING,

  listed\_in STRING,

  description STRING )

**// Create file format object**

CREATE OR REPLACE file format MANAGE\_DB.file\_formats.csv\_fileformat

    type = csv

    field\_delimiter = ','

    skip\_header = 1

    null\_if = ('NULL','null')

    empty\_field\_as\_null = TRUE;

**// Create stage object with integration object & file format object**

CREATE OR REPLACE stage MANAGE\_DB.external\_stages.csv\_folder

    URL = 's3://<your-bucket-name>/<your-path>/'

    STORAGE\_INTEGRATION = s3\_int

    FILE\_FORMAT = MANAGE\_DB.file\_formats.csv\_fileformat

**// Use Copy command**

COPY INTO OUR\_FIRST\_DB.PUBLIC.movie\_titles

    FROM @MANAGE\_DB.external\_stages.csv\_folder

**// Create file format object**

CREATE OR REPLACE file format MANAGE\_DB.file\_formats.csv\_fileformat

    type = csv

    field\_delimiter = ','

    skip\_header = 1

    null\_if = ('NULL','null')

    empty\_field\_as\_null = TRUE

    FIELD\_OPTIONALLY\_ENCLOSED\_BY = '"'

  SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.movie\_titles

**## Connection of Snowflake with AZURE**

CREATE OR REPLACE DEMO\_DB;

 USE DATABASE DEMO\_DB;

**-- Create integration object that contains the access information**

CREATE STORAGE INTEGRATION azure\_integration

  TYPE = EXTERNAL\_STAGE

  STORAGE\_PROVIDER = AZURE

  ENABLED = TRUE

  AZURE\_TENANT\_ID = '9ecede0b-0e07-4da4-8047-e0672d6e403e'

  STORAGE\_ALLOWED\_LOCATIONS = ('azure://storageaccountsnow.blob.core.windows.net/snowflakecsv', 'azure://storageaccountsnow.blob.core.windows.net/snowflakejson');

**-- Describe integration object to provide access**

DESC STORAGE integration azure\_integration;

**---- Create file format & stage objects ----**

**-- create file format**

create or replace file format demo\_db.public.fileformat\_azure

    TYPE = CSV

    FIELD\_DELIMITER = ','

    SKIP\_HEADER = 1;

**-- create stage object**

create or replace stage demo\_db.public.stage\_azure

    STORAGE\_INTEGRATION = azure\_integration

    URL = 'azure://storageaccountsnow.blob.core.windows.net/snowflakecsv'

    FILE\_FORMAT = fileformat\_azure;

**-- list files**

LIST @demo\_db.public.stage\_azure;

**---- Query files & Load data ----**

**--query files**

SELECT

$1,

$2,

$3,

$4,

$5,

$6,

$7,

$8,

$9,

$10,

$11,

$12,

$13,

$14,

$15,

$16,

$17,

$18,

$19,

$20

FROM @demo\_db.public.stage\_azure;

create or replace table happiness (

country\_name varchar,

regional\_indicator varchar,

ladder\_score number(4,3),

standard\_error number(4,3),

upperwhisker number(4,3),

lowerwhisker number(4,3),

logged\_gdp number(5,3),

social\_support number(4,3),

healthy\_life\_expectancy number(5,3),

freedom\_to\_make\_life\_choices number(4,3),

generosity number(4,3),

perceptions\_of\_corruption number(4,3),

ladder\_score\_in\_dystopia number(4,3),

explained\_by\_log\_gpd\_per\_capita number(4,3),

explained\_by\_social\_support number(4,3),

explained\_by\_healthy\_life\_expectancy number(4,3),

explained\_by\_freedom\_to\_make\_life\_choices number(4,3),

explained\_by\_generosity number(4,3),

explained\_by\_perceptions\_of\_corruption number(4,3),

dystopia\_residual number (4,3));

COPY INTO HAPPINESS

FROM @demo\_db.public.stage\_azure;

SELECT \* FROM HAPPINESS;

**--- Load JSON ----**

create or replace file format demo\_db.public.fileformat\_azure\_json

    TYPE = JSON;

create or replace stage demo\_db.public.stage\_azure

    STORAGE\_INTEGRATION = azure\_integration

    URL = 'azure://storageaccountsnow.blob.core.windows.net/snowflakejson'

    FILE\_FORMAT = fileformat\_azure\_json;

LIST  @demo\_db.public.stage\_azure;

**-- Query from stage**

SELECT \* FROM @demo\_db.public.stage\_azure;

**-- Query one attribute/column**

SELECT $1:"Car Model" FROM @demo\_db.public.stage\_azure;

**-- Convert data type**

SELECT $1:"Car Model"::STRING FROM @demo\_db.public.stage\_azure;

**-- Query all attributes**

SELECT

$1:"Car Model"::STRING,

$1:"Car Model Year"::INT,

$1:"car make"::STRING,

$1:"first\_name"::STRING,

$1:"last\_name"::STRING

FROM @demo\_db.public.stage\_azure;

**-- Query all attributes and use aliases**

SELECT

$1:"Car Model"::STRING as car\_model,

$1:"Car Model Year"::INT as car\_model\_year,

$1:"car make"::STRING as "car make",

$1:"first\_name"::STRING as first\_name,

$1:"last\_name"::STRING as last\_name

FROM @demo\_db.public.stage\_azure;

Create or replace table car\_owner (

    car\_model varchar,

    car\_model\_year int,

    car\_make varchar,

    first\_name varchar,

    last\_name varchar)

COPY INTO car\_owner

FROM

(SELECT

$1:"Car Model"::STRING as car\_model,

$1:"Car Model Year"::INT as car\_model\_year,

$1:"car make"::STRING as "car make",

$1:"first\_name"::STRING as first\_name,

$1:"last\_name"::STRING as last\_name

FROM @demo\_db.public.stage\_azure);

SELECT \* FROM CAR\_OWNER;

**Alternative Method**

**-- Using a raw file table step**

truncate table car\_owner;

select \* from car\_owner;

create or replace table car\_owner\_raw (

  raw variant);

COPY INTO car\_owner\_raw

FROM @demo\_db.public.stage\_azure;

SELECT \* FROM car\_owner\_raw;

INSERT INTO car\_owner

(SELECT

$1:"Car Model"::STRING as car\_model,

$1:"Car Model Year"::INT as car\_model\_year,

$1:"car make"::STRING as car\_make,

$1:"first\_name"::STRING as first\_name,

$1:"last\_name"::STRING as last\_name

FROM car\_owner\_raw)

select \* from car\_owner;

**## Connection of Snowflake with GCP**

CREATE OR REPLACE DEMO\_DB;

 USE DATABASE DEMO\_DB;

**-- Create integration object that contains the access information**

CREATE STORAGE INTEGRATION gcp\_integration

  TYPE = EXTERNAL\_STAGE

  STORAGE\_PROVIDER = GCS

  ENABLED = TRUE

 STORAGE\_ALLOWED\_LOCATIONS = (‘gcs://snowflakeineuroncsv' , ‘gcs://snowflakeineuronjson');

**-- Describe integration object to provide access**

DESC STORAGE integration gcp\_integration;

**-- create file format--**

use demo\_db

create file format

create or replace file format demo\_db.public.fileformat\_gcp

TYPE = CSV

FIELD \_DELIMITER=’,’

SKIP HEADER = 1

**--create stage object--**

create or replace stage demo\_db.public.stage\_gcp

STORAGE INTEGRATION = gcp\_integration

URL ='gcs://snowflakeineuroncsv/'

FILE FORMAT = fileformat\_gcp

**// List**

LIST @demo\_db.public.stage\_gcp

**---- Query files & Load data ----**

**--query files**

SELECT

$1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,

$12,$13,$14,$15,$16,$17,$18,$19,$20

FROM @demo\_db.public.stage\_gcp;

create or replace table happiness (

country\_name varchar,

regional\_indicator varchar,

ladder\_score number(4,3),

standard\_error number(4,3),

upperwhisker number(4,3),

lowerwhisker number(4,3),

logged\_gdp number(5,3),

social\_support number(4,3),

healthy\_life\_expectancy number(5,3),

freedom\_to\_make\_life\_choices number(4,3),

generosity number(4,3),

perceptions\_of\_corruption number(4,3),

ladder\_score\_in\_dystopia number(4,3),

explained\_by\_log\_gpd\_per\_capita number(4,3),

explained\_by\_social\_support number(4,3),

explained\_by\_healthy\_life\_expectancy number(4,3),

explained\_by\_freedom\_to\_make\_life\_choices number(4,3),

explained\_by\_generosity number(4,3),

explained\_by\_perceptions\_of\_corruption number(4,3),

dystopia\_residual number (4,3));

COPY INTO HAPPINESS

FROM @demo\_db.public.stage\_gcp;

SELECT \* FROM HAPPINESS;

**SNOWPIPE**

**// Create table first**

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

  id INT,

  first\_name STRING,

  last\_name STRING,

  email STRING,

  location STRING,

  department STRING

  )

**// Create file format object**

CREATE OR REPLACE file format MANAGE\_DB.file\_formats.csv\_fileformat

    type = csv

    field\_delimiter = ','

    skip\_header = 1

    null\_if = ('NULL','null')

    empty\_field\_as\_null = TRUE;

**// Create stage object with integration object & file format object**

CREATE OR REPLACE stage MANAGE\_DB.external\_stages.csv\_folder

    URL = 's3://snowflakes3bucket123/csv/snowpipe'

    STORAGE\_INTEGRATION = s3\_int

    FILE\_FORMAT = MANAGE\_DB.file\_formats.csv\_fileformat

**// Create stage object with integration object & file format object**

LIST @MANAGE\_DB.external\_stages.csv\_folder

**// Create schema to keep things organized**

CREATE OR REPLACE SCHEMA MANAGE\_DB.pipes

**// Define pipe**

CREATE OR REPLACE pipe MANAGE\_DB.pipes.employee\_pipe

auto\_ingest = TRUE

AS

COPY INTO OUR\_FIRST\_DB.PUBLIC.employees

FROM @MANAGE\_DB.external\_stages.csv\_folder

**// Describe pipe**

DESC pipe employee\_pipe

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.employees

**// Handling errors**

**// Create file format object**

CREATE OR REPLACE file format MANAGE\_DB.file\_formats.csv\_fileformat

    type = csv

    field\_delimiter = '|'

    skip\_header = 1

    null\_if = ('NULL','null')

    empty\_field\_as\_null = TRUE;

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.employees

ALTER PIPE employee\_pipe refresh

**// Validate pipe is actually working**

SELECT SYSTEM$PIPE\_STATUS('employee\_pipe')

**// Snowpipe error message**

SELECT \* FROM TABLE(VALIDATE\_PIPE\_LOAD(

    PIPE\_NAME => 'MANAGE\_DB.pipes.employee\_pipe',

    START\_TIME => DATEADD(HOUR,-2,CURRENT\_TIMESTAMP())))

**// COPY command history from table to see error massage**

SELECT \* FROM TABLE (INFORMATION\_SCHEMA.COPY\_HISTORY(

   table\_name  =>  'OUR\_FIRST\_DB.PUBLIC.EMPLOYEES',

   START\_TIME =>DATEADD(HOUR,-2,CURRENT\_TIMESTAMP())))

CREATE OR REPLACE file format MANAGE\_DB.file\_formats.csv\_fileformat

    type = csv

    field\_delimiter = ','

    skip\_header = 1

    null\_if = ('NULL','null')

    empty\_field\_as\_null = TRUE;

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.employees

ALTER PIPE employee\_pipe refresh

SELECT \* FROM TABLE (INFORMATION\_SCHEMA.COPY\_HISTORY(

   table\_name  =>  'OUR\_FIRST\_DB.PUBLIC.EMPLOYEES',

   START\_TIME =>DATEADD(HOUR,-2,CURRENT\_TIMESTAMP())))

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.employees

**-- Manage pipes --**

DESC pipe MANAGE\_DB.pipes.employee\_pipe;

SHOW PIPES;

SHOW PIPES like '%employee%'

SHOW PIPES in database MANAGE\_DB

SHOW PIPES in schema MANAGE\_DB.pipes

SHOW PIPES like '%employee%' in Database MANAGE\_DB

**-- Changing pipe (alter stage or file format) --**

**// Preparation table first**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.PUBLIC.employees2 (

  id INT,

  first\_name STRING,

  last\_name STRING,

  email STRING,

  location STRING,

  department STRING

  )

**// Pause pipe**

ALTER PIPE MANAGE\_DB.pipes.employee\_pipe SET PIPE\_EXECUTION\_PAUSED = true

**// Verify pipe is paused and has pendingFileCount 0**

SELECT SYSTEM$PIPE\_STATUS('MANAGE\_DB.pipes.employee\_pipe')

**// Recreate the pipe to change the COPY statement in the definition**

CREATE OR REPLACE pipe MANAGE\_DB.pipes.employee\_pipe

auto\_ingest = TRUE

AS

COPY INTO OUR\_FIRST\_DB.PUBLIC.employees2

FROM @MANAGE\_DB.external\_stages.csv\_folder

ALTER PIPE  MANAGE\_DB.pipes.employee\_pipe refresh

**// List files in stage**

LIST @MANAGE\_DB.external\_stages.csv\_folder

SELECT \* FROM OUR\_FIRST\_DB.PUBLIC.employees2

**// Reload files manually that where aleady in the bucket**

COPY INTO OUR\_FIRST\_DB.PUBLIC.employees2

FROM @MANAGE\_DB.external\_stages.csv\_folder

**// Resume pipe**

ALTER PIPE MANAGE\_DB.pipes.employee\_pipe SET PIPE\_EXECUTION\_PAUSED = false

**// Verify pipe is running again**

SELECT SYSTEM$PIPE\_STATUS('MANAGE\_DB.pipes.employee\_pipe')

**## Time trival**

**// Setting up table**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.test (

   id int,

   first\_name string,

  last\_name string,

  email string,

  gender string,

  Job string,

  Phone string)

CREATE OR REPLACE FILE FORMAT MANAGE\_DB.file\_formats.csv\_file

    type = csv

    field\_delimiter = ','

    skip\_header = 1

CREATE OR REPLACE STAGE MANAGE\_DB.external\_stages.time\_travel\_stage

    URL = 's3://data-snowflake-fundamentals/time-travel/'

    file\_format = MANAGE\_DB.file\_formats.csv\_file;

LIST @MANAGE\_DB.external\_stages.time\_travel\_stage

COPY INTO OUR\_FIRST\_DB.public.test

from @MANAGE\_DB.external\_stages.time\_travel\_stage

files = ('customers.csv')

SELECT \* FROM OUR\_FIRST\_DB.public.test

**// Use-case: Update data (by mistake)**

UPDATE OUR\_FIRST\_DB.public.test

SET FIRST\_NAME = 'Joyen'

**// // // Using time travel: Method 1 - 2 minutes back**

SELECT \* FROM OUR\_FIRST\_DB.public.test at (OFFSET => -60\*1.5)

**// // // Using time travel: Method 2 - before timestamp**

SELECT \* FROM OUR\_FIRST\_DB.public.test before (timestamp => '2021-04-15 17:47:50.581'::timestamp)

**-- Setting up table**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.test (

   id int,

   first\_name string,

  last\_name string,

  email string,

  gender string,

  Job string,

  Phone string);

COPY INTO OUR\_FIRST\_DB.public.test

from @MANAGE\_DB.external\_stages.time\_travel\_stage

files = ('customers.csv');

SELECT \* FROM OUR\_FIRST\_DB.public.test;

2021-04-17 08:16:24.259

**-- Setting up UTC time for convenience**

ALTER SESSION SET TIMEZONE ='UTC'

SELECT DATEADD(DAY, 1, CURRENT\_TIMESTAMP)

UPDATE OUR\_FIRST\_DB.public.test

SET Job = 'Data Scientist'

SELECT \* FROM OUR\_FIRST\_DB.public.test;

SELECT \* FROM OUR\_FIRST\_DB.public.test before (timestamp => '2021-04-16 07:30:47.145'::timestamp)

**// // // Using time travel: Method 3 - before Query ID**

**// Preparing table**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.test (

   id int,

   first\_name string,

  last\_name string,

  email string,

  gender string,

  Phone string,

  Job string)

COPY INTO OUR\_FIRST\_DB.public.test

from @MANAGE\_DB.external\_stages.time\_travel\_stage

files = ('customers.csv')

SELECT \* FROM OUR\_FIRST\_DB.public.test

**// Altering table (by mistake**)

UPDATE OUR\_FIRST\_DB.public.test

SET EMAIL = null

SELECT \* FROM OUR\_FIRST\_DB.public.test

SELECT \* FROM OUR\_FIRST\_DB.public.test before (statement => '019b9ee5-0500-8473-0043-4d8300073062')

**// Setting up table**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.test (

   id int,

   first\_name string,

  last\_name string,

  email string,

  gender string,

  Job string,

  Phone string);

COPY INTO OUR\_FIRST\_DB.public.test

from @MANAGE\_DB.external\_stages.time\_travel\_stage

files = ('customers.csv');

SELECT \* FROM OUR\_FIRST\_DB.public.test;

**// Use-case: Update data (by mistake)**

UPDATE OUR\_FIRST\_DB.public.test

SET LAST\_NAME = 'Tyson';

UPDATE OUR\_FIRST\_DB.public.test

SET JOB = 'Data Analyst';

SELECT \* FROM OUR\_FIRST\_DB.public.test before (statement => '019b9eea-0500-845a-0043-4d830007402a')

**// // // Bad method**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.test as

SELECT \* FROM OUR\_FIRST\_DB.public.test before (statement => '019b9eea-0500-845a-0043-4d830007402a')

SELECT \* FROM OUR\_FIRST\_DB.public.test

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.test as

SELECT \* FROM OUR\_FIRST\_DB.public.test before (statement => '019b9eea-0500-8473-0043-4d830007307a')

**// Setting up table**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.test (

   id int,

   first\_name string,

  last\_name string,

  email string,

  gender string,

  Job string,

  Phone string);

COPY INTO OUR\_FIRST\_DB.public.test

from @MANAGE\_DB.external\_stages.time\_travel\_stage

files = ('customers.csv');

SELECT \* FROM OUR\_FIRST\_DB.public.test;

**// Use-case: Update data (by mistake)**

UPDATE OUR\_FIRST\_DB.public.test

SET LAST\_NAME = 'Tyson';

UPDATE OUR\_FIRST\_DB.public.test

SET JOB = 'Data Analyst';

SELECT \* FROM OUR\_FIRST\_DB.public.test before (statement => '019b9eea-0500-845a-0043-4d830007402a')

**// // // Good method**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.test\_backup as

SELECT \* FROM OUR\_FIRST\_DB.public.test before (statement => '019b9ef0-0500-8473-0043-4d830007309a')

TRUNCATE OUR\_FIRST\_DB.public.test

INSERT INTO OUR\_FIRST\_DB.public.test

SELECT \* FROM OUR\_FIRST\_DB.public.test\_backup

SELECT \* FROM OUR\_FIRST\_DB.public.test

**// Setting up table**

CREATE OR REPLACE STAGE MANAGE\_DB.external\_stages.time\_travel\_stage

    URL = 's3://data-snowflake-fundamentals/time-travel/'

    file\_format = MANAGE\_DB.file\_formats.csv\_file;

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.customers (

   id int,

   first\_name string,

  last\_name string,

  email string,

  gender string,

  Job string,

  Phone string);

COPY INTO OUR\_FIRST\_DB.public.customers

from @MANAGE\_DB.external\_stages.time\_travel\_stage

files = ('customers.csv');

SELECT \* FROM OUR\_FIRST\_DB.public.customers;

**// UNDROP command - Tables**

DROP TABLE OUR\_FIRST\_DB.public.customers;

SELECT \* FROM OUR\_FIRST\_DB.public.customers;

UNDROP TABLE OUR\_FIRST\_DB.public.customers;

**// UNDROP command - Schemas**

DROP SCHEMA OUR\_FIRST\_DB.public;

SELECT \* FROM OUR\_FIRST\_DB.public.customers;

UNDROP SCHEMA OUR\_FIRST\_DB.public;

**// UNDROP command - Database**

DROP DATABASE OUR\_FIRST\_DB;

SELECT \* FROM OUR\_FIRST\_DB.public.customers;

UNDROP DATABASE OUR\_FIRST\_DB;

**// Restore replaced table**

UPDATE OUR\_FIRST\_DB.public.customers

SET LAST\_NAME = 'Tyson';

UPDATE OUR\_FIRST\_DB.public.customers

SET JOB = 'Data Analyst';

**// // // Undroping a with a name that already exists**

CREATE OR REPLACE TABLE OUR\_FIRST\_DB.public.customers as

SELECT \* FROM OUR\_FIRST\_DB.public.customers before (statement => '019b9f7c-0500-851b-0043-4d83000762be')

SELECT \* FROM OUR\_FIRST\_DB.public.customers

UNDROP table OUR\_FIRST\_DB.public.customers;

ALTER TABLE OUR\_FIRST\_DB.public.customers

RENAME TO OUR\_FIRST\_DB.public.customers\_wrong;

DESC table OUR\_FIRST\_DB.public.customers

**Time Travel Cost**

SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.STORAGE\_USAGE ORDER BY USAGE\_DATE DESC;

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

**// Query time travel storage**

SELECT ID,

TABLE\_NAME,

TABLE\_SCHEMA,

TABLE\_CATALOG,

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

TIME\_TRAVEL\_BYTES / (1024\*1024\*1024) AS TIME\_TRAVEL\_STORAGE\_USED\_GB

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

ORDER BY STORAGE\_USED\_GB DESC,TIME\_TRAVEL\_STORAGE\_USED\_GB DESC;

**Fail safe**

**// Storage usage on account level**

SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.STORAGE\_USAGE ORDER BY USAGE\_DATE DESC;

**// Storage usage on account level formatted**

SELECT USAGE\_DATE,

STORAGE\_BYTES / (1024\*1024\*1024) AS STORAGE\_GB,

STAGE\_BYTES / (1024\*1024\*1024) AS STAGE\_GB,

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

FROM SNOWFLAKE.ACCOUNT\_USAGE.STORAGE\_USAGE ORDER BY USAGE\_DATE DESC;

**// Storage usage on table level**

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

**// Storage usage on table level formatted**

SELECT ID,

TABLE\_NAME,

TABLE\_SCHEMA,

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

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

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

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

ORDER BY FAILSAFE\_STORAGE\_USED\_GB DESC;