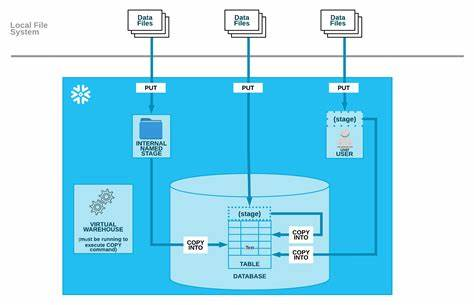
**Practical for snowflake:**

1. **Loading the data from local to SF (INTERNAL STAGING)**



We are create a table for table stage:

(in snowflake worksheet)

create or replace table customer\_csv (

customer\_pk number(38,0),

salutation varchar(10),

first\_name varchar(20),

last\_name varchar(20),

gender varchar(1),

marital\_status varchar(10),

day\_of\_birth date,

birth\_country varchar(60),

email\_address varchar(60),

city\_name varchar(20),

zip\_code varchar (10),

country\_name varchar(20),

gmt\_timezone\_offset number(10,2),

preferred\_cust\_flag boolean,

registration\_time timestamp\_ltz(9)

);

select \* from customer\_csv;

**--uploading data file to user stage** (in local cmd)

put file://C:\Users\BhandariRaviraj\customer\_10k.csv @~/staged;

**--uploading data file to table stage**

put file://C:\Users\BhandariRaviraj\customer\_10k.csv @%customer\_csv;

**--file format for named stage** (in local cmd)

create or replace file format customer\_ff

type='csv', field\_delimiter = ',', field\_optionally\_enclosed\_by='\042', skip\_header = 1;

**--creating named stage** (in local cmd)

create or replace stage customer\_stage

//file\_format=(type='csv', field\_delimiter = ',', field\_optionally\_enclosed\_by='\042', skip\_header = 1);

file\_format=customer\_ff;

**--upload data file to named stage (customer\_stage)**(in local cmd)

put file://C:\Users\BhandariRaviraj\customer\_10k.csv @customer\_stage;

**--listing from internal stages(**in snowflake worksheet**)**

list @~/staged; --user stage

list @%customer\_csv; -- table stage

list @customer\_stage; --named stage

**Copying/ loading data files from stages(user, named, table) to table**

**--load data from user stage**

copy into customer\_csv

from @~/staged

file\_format=(type='csv', field\_delimiter = ',', field\_optionally\_enclosed\_by='\042', skip\_header = 1);

select \* from customer\_csv;

**--load data from table stage**

copy into customer\_csv

from @%customer\_csv

file\_format=(type='csv', field\_delimiter = ',', field\_optionally\_enclosed\_by='\042', skip\_header = 1);

**--load data from named stage (customer\_stage)**

copy into cutomer\_csv

from @customer\_stage/customer\_10k.csv;

\*\* If error occurs for column mismatch try to stage with below command:

put file://C:\Users\BhandariRaviraj\test.csv @~/final AUTO\_COMPRESS=TRUE OVERWRITE=TRUE;

COPY INTO mytable

FROM @~/test/final.csv.gz

FILE\_FORMAT = (type='csv', field\_delimiter = ',', field\_optionally\_enclosed\_by='\042', skip\_header = 1)

ON\_ERROR = 'skip\_file';

**-- copy command properties**

**pattern** = '.\*/.\*/.\*[.]csv[.]gz'

**FILES** = ('file\_name')

**file\_format** = format\_name

**ON\_ERROR** = {CONTINUE | SKIP\_FILE | SKIP\_FILE\_<num> | 'SKIP\_FILE\_<num>%' | ABORT\_STATEMENT}

**SIZE\_LIMIT** = <num>

**PURGE** = TRUE | FALSE

**LOAD\_UNCERTAIN\_FILES** = TRUE | FALSE

**trim\_space** = false | true

**error\_on\_column\_count\_mismatch** = true | false;

**-- loading error commands**

--**1. on error => skip the file**

COPY INTO emp.emp\_customer\_csv

From @/file\_name

FILE\_FORMAT = emp.emp.customer\_tsv\_ff

ON\_ERROR = 'SKIP\_FILE'

PURGE = TRUE;

--**2. on error** **=> abort statement**

COPY INTO emp.emp.customer\_csv

FROM @/file\_name

FILE\_FORMAT = emp.emp.customer\_csv\_ff

ERROR = 'ABORT\_STATEMENT'

PURGE = TRUE;

--**3. on error** **=> skip upto 10 error records**

COPY INTO emp.emp.customer\_csv\_ff

FROM @/file\_name

FILE\_FORMAT = emp.emp.customer\_c[sv\_ff

ON\_ERROR = 'SKIP\_FILE\_10'

PURGE = TRUE;

--**4. On erro**r **=> continue even if there is an error**

COPY INTO emp.emp.cutomer\_csv

FROM @/FILENAME

FILE\_FORMAT = emp.emp.customer\_csv\_ff

ON\_ERROR = 'CONTINUE'

PURGE = TRUE;

**---loading psv file**

create or replace table customer\_psv (

CUSTOMER\_PK number(38,0),

SALUTATION varchar(10),

FIRST\_NAME varchar(20),

LAST\_NAME varchar(20),

GENDER varchar(1),

MARITAL\_STATUS varchar(20),

DAY\_OF\_BIRTH date,

BIRTH\_COUNTRY varchar(60),

EMAIL\_ADDRESS varchar(60),

CITY\_NAME varchar(50),

ZIP\_CODE varchar (50),

COUNTRY\_NAME varchar(50),

GMT\_TIMEZONE\_OFFSET number(10,2),

PREFERRED\_CUST\_FLAG boolean,

REGISTRATION\_TIME VARCHAR(30)

);

select \* from customer\_psv;

**--create ff for csv**

create or replace file format customer\_ff

type='csv', field\_delimiter =',', field\_optionally\_enclosed\_by = '\042', skip\_header=1;

**--stage area**

create or replace stage customer\_stg

file\_format = customer\_ff;

**-- uploading to stage area**

put file://C:\Users\BhandariRaviraj\customer\_10k.\* @customer\_stg;

**-- list**

list @customer\_stg;

**-- loading to customer csv**

copy into customer\_csv

from @customer\_stg/customer\_10k.csv.gz

on\_error = 'continue'**;**

**-- loading to customer csv with specified file**

copy into customer\_csv

from @customer\_stg/customer\_10k.csv.gz

files = ('customer\_10k.csv.gz')

on\_error = 'continue';

**--create ff for psv**

create or replace file format customer\_ff

type='csv', field\_delimiter ='|', field\_optionally\_enclosed\_by = '\042', skip\_header=1;

**-- loading to customer psv**

copy into customer\_psv

from @customer\_stg

files = ('customer\_10k.psv.gz')

on\_error = 'continue';

**– loading tsv data**

create or replace table customer\_tsv (

CUSTOMER\_PK number(38,0),

SALUTATION varchar(10),

FIRST\_NAME varchar(20),

LAST\_NAME varchar(20),

GENDER varchar(1),

MARITAL\_STATUS varchar(20),

DAY\_OF\_BIRTH date,

BIRTH\_COUNTRY varchar(60),

EMAIL\_ADDRESS varchar(60),

CITY\_NAME varchar(50),

ZIP\_CODE varchar (50),

COUNTRY\_NAME varchar(50),

GMT\_TIMEZONE\_OFFSET number(10,2),

PREFERRED\_CUST\_FLAG boolean,

REGISTRATION\_TIME VARCHAR(30)

);

**--create ff for tsv**

create or replace file format customer\_ff

type='csv', field\_delimiter ='\t', field\_optionally\_enclosed\_by = '\042', skip\_header=1;

**-- loading to customer psv**

copy into customer\_tsv

from @customer\_stg

files = ('customer\_10k.tsv.gz')

on\_error = 'continue';

select \* from customer\_tsv;

**--- loading multiple csv files in 1 go**

create or replace table customer\_multiple (

CUSTOMER\_PK number(38,0),

SALUTATION varchar(10),

FIRST\_NAME varchar(20),

LAST\_NAME varchar(20),

GENDER varchar(1),

MARITAL\_STATUS varchar(20),

DAY\_OF\_BIRTH date,

BIRTH\_COUNTRY varchar(60),

EMAIL\_ADDRESS varchar(60),

CITY\_NAME varchar(50),

ZIP\_CODE varchar (50),

COUNTRY\_NAME varchar(50),

GMT\_TIMEZONE\_OFFSET number(10,2),

PREFERRED\_CUST\_FLAG boolean,

REGISTRATION\_TIME VARCHAR(30)

);

select \* from customer\_multiple;

**-- create a file format**

create or replace file format customer\_multi\_ff

type = 'csv', field\_delimiter = ',', field\_optionally\_enclosed\_by = '\042', skip\_header=1;

**-- create a stage to load all data files in 1 go**

create or replace stage customer\_multi

file\_format = customer\_multi\_ff;

list @customer\_multi;

**-- upload file to stage**

put file://C:\Users\BhandariRaviraj\customer\_10k.\*csv @customer\_multi;

**-- copy file from stage to table**

copy into customer\_multiple

from @customer\_multi

file\_format = customer\_multi\_ff

on\_error = 'CONTINUE'

pattern = ',\*[.]csv[.]gz';

select \* from customer\_multiple;

**—-Loading file for big\_data**

**--create a table for big data**

create or replace table customer\_bigdata (

CUSTOMER\_PK number(38,0),

SALUTATION varchar(10),

FIRST\_NAME varchar(20),

LAST\_NAME varchar(20),

GENDER varchar(1),

MARITAL\_STATUS varchar(20),

DAY\_OF\_BIRTH date,

BIRTH\_COUNTRY varchar(60),

EMAIL\_ADDRESS varchar(60),

CITY\_NAME varchar(50),

ZIP\_CODE varchar (50),

COUNTRY\_NAME varchar(50),

GMT\_TIMEZONE\_OFFSET number(10,2),

PREFERRED\_CUST\_FLAG boolean,

REGISTRATION\_TIME VARCHAR(30)

);

select \* from customer\_bigdata;

**--create a file format**

create or replace file format customer\_multi\_ff

type = 'csv', record\_delimiter= '\n', field\_delimiter = ',', field\_optionally\_enclosed\_by = '\042', skip\_header=1;

**--create a stage to load all data files in 1 go**

create or replace stage customer\_bigdata

file\_format = customer\_multi\_ff;

**--upload the data**

copy into customer\_bigdata

from @customer\_bigdata

file\_format = customer\_multi\_ff

on\_error = 'CONTINUE'

pattern = '.\*[.]csv[.]gz'

purge = true;

select \* from customer\_bigdata;

**----loading data files with column mismatch**

--create a table for multiple files

create or replace table column\_mismatch (

CUSTOMER\_PK number(38,0),

SALUTATION varchar(10),

FIRST\_NAME varchar(20),

LAST\_NAME varchar(20),

GENDER varchar(1),

MARITAL\_STATUS varchar(20),

DAY\_OF\_BIRTH date,

BIRTH\_COUNTRY varchar(60),

EMAIL\_ADDRESS varchar(60),

CITY\_NAME varchar(50),

ZIP\_CODE varchar (50),

COUNTRY\_NAME varchar(50),

GMT\_TIMEZONE\_OFFSET number(10,2),

PREFERRED\_CUST\_FLAG boolean,

REGISTRATION\_TIME VARCHAR(30)

);

select \* from column\_mismatch;

**--create a file format**

create or replace file format customer\_multi\_ff

type = 'csv', field\_delimiter = ',', field\_optionally\_enclosed\_by = '\042', skip\_header=1, error\_on\_column\_count\_mismatch = false;

**--create a stage to load all data files in 1 go**

create or replace stage column\_stage

file\_format = customer\_multi\_ff;

**--upload the data**

put file://C:\Users\BhandariRaviraj\customer\_10k.csv @column\_stage auto\_compress =false;

list @column\_stage;

**--copy file from stage to table**

copy into column\_mismatch

from @column\_stage

file\_format = customer\_multi\_ff

on\_error = 'CONTINUE';

select \* from column\_mismatch;

**----loading data files with special character**

**--create a table**

create or replace table user\_email(

id number,

first\_name varchar(100),

last\_name varchar(100),

email varchar(100),

gender varchar(6),

about\_me varchar(500)

);

**--create file\_format**

**--create a file format where field\_optionally\_enclosed\_by parameter will take double quote (\042: for double quote)**

create or replace file format csv\_double\_q\_ff

type = 'csv'

compression = 'none'

field\_delimiter = ','

record\_delimiter = '\n'

skip\_header = 1

field\_optionally\_enclosed\_by = '\042'

trim\_space = false

error\_on\_column\_count\_mismatch = true

escape = '\134';

**--create a file format where field\_optionally\_enclosed\_by parameter will take double quote (\047: for single quote)**

create or replace file format csv\_single\_q\_ff

type = 'csv'

compression = 'none'

field\_delimiter = ','

record\_delimiter = '\n'

skip\_header = 1

field\_optionally\_enclosed\_by = '\047'

trim\_space = false

error\_on\_column\_count\_mismatch = true

escape = '\134';

**-- create a stage to load all data files in 1 go**

create or replace stage special\_char;

list @special\_char;

**--upload file to stage**

put file://C:\Users\BhandariRaviraj\snowflakedata\specialchar\\*.csv @special\_charauto\_compress = false;

**--copy file from stage to table(simple csv file )**

copy into user\_email

from @special\_char/01\_sample\_user\_email.csv

file\_format = csv\_double\_q\_ff

on\_error = 'CONTINUE';

select \* from user\_email;

truncate table user\_email;

**--- file with double quotes**

copy into user\_email

from @special\_char/02\_sample\_email\_double\_quotes.csv

file\_format = csv\_double\_q\_ff

on\_error = 'CONTINUE';

**--- file with single quotes**

copy into user\_email

from @special\_char/03\_sample\_email\_single\_quote.csv

file\_format = csv\_single\_q\_ff

on\_error = 'CONTINUE';

**— file with mixed quotes**

list @special\_char;

copy into user\_email

from @special\_char/04\_sample\_email\_quote\_mixed.csv

file\_format = csv\_double\_q\_ff

on\_error = 'CONTINUE';

select \* from user\_email;

truncate table user\_email;

**--- file with new line character**

copy into user\_email

from @special\_char/05\_sample\_email\_new\_line.csv

file\_format = csv\_double\_q\_ff

on\_error = 'CONTINUE';

**--- file with new line character and double quotes**

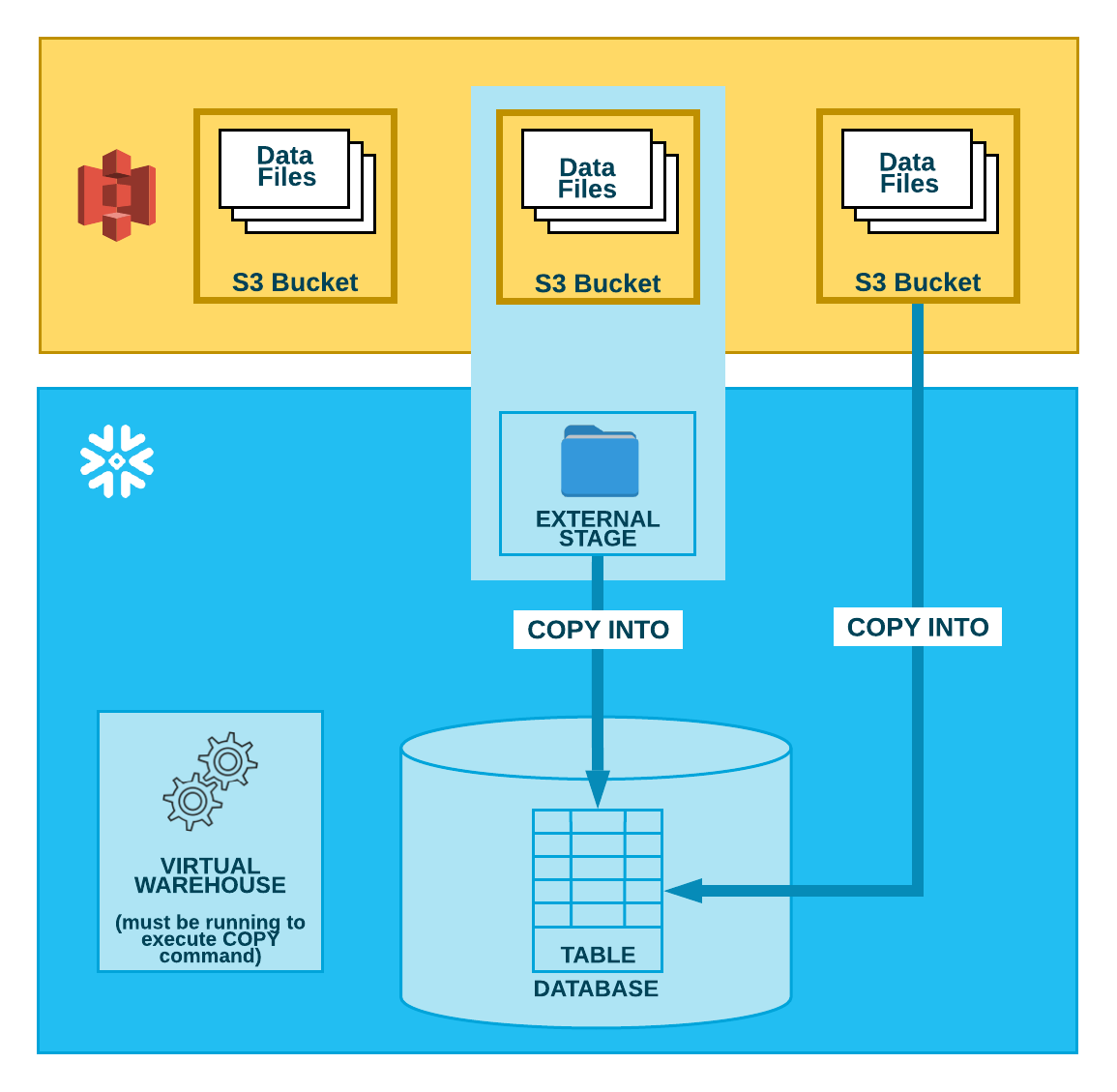
copy into user\_email

from @special\_char/04\_1\_sample\_email\_double\_q.csv

file\_format = csv\_double\_q\_ff

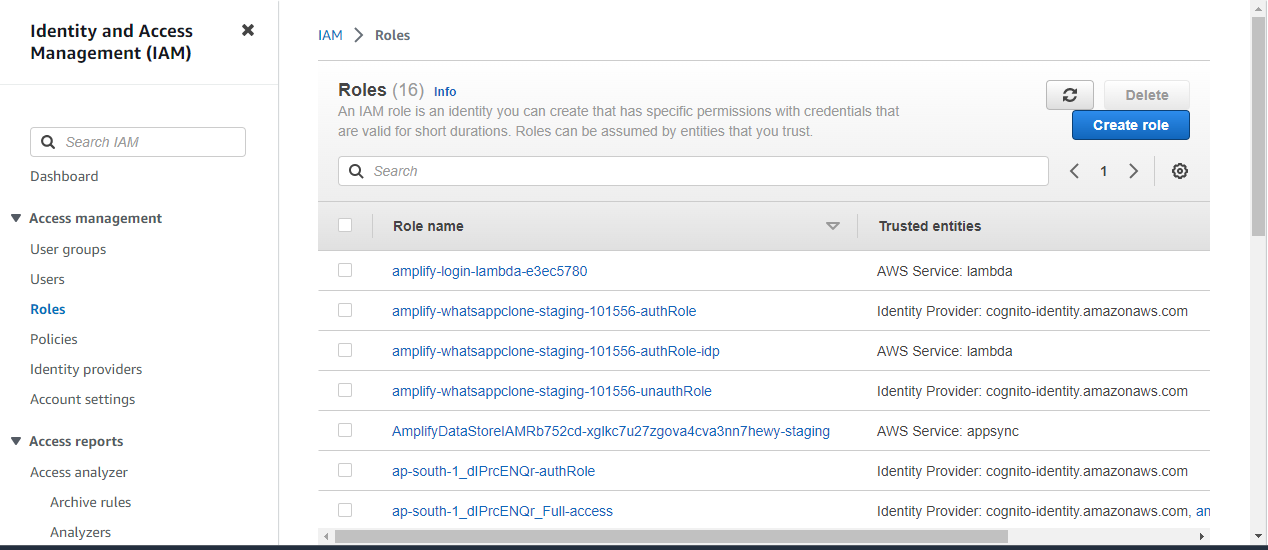
on\_error = 'CONTINUE';

1. **Loading the data from S3 to SF (EXTERNAL STAGING)**

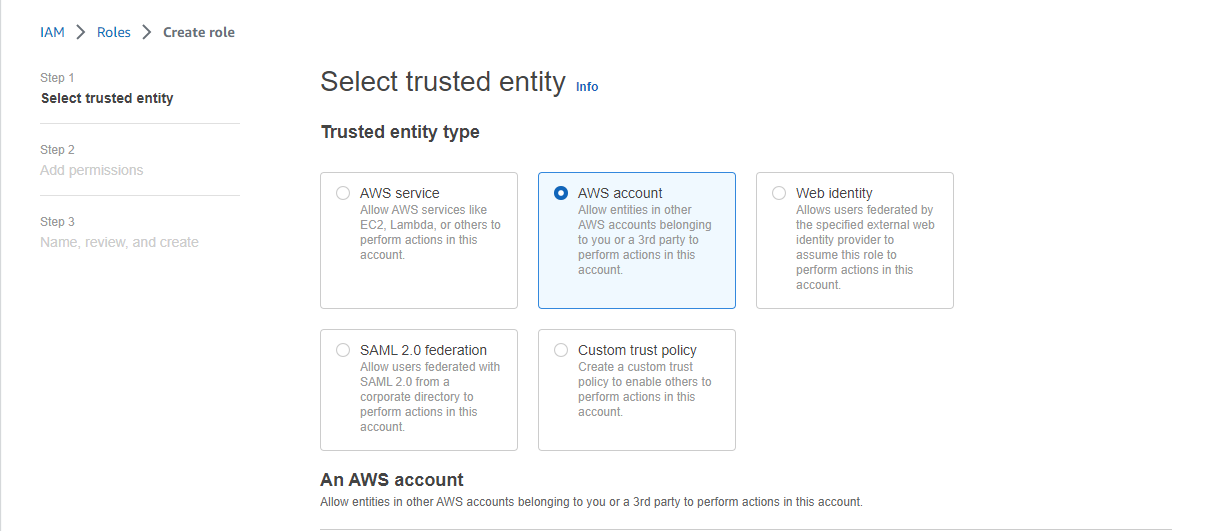


**– Create AWS account**

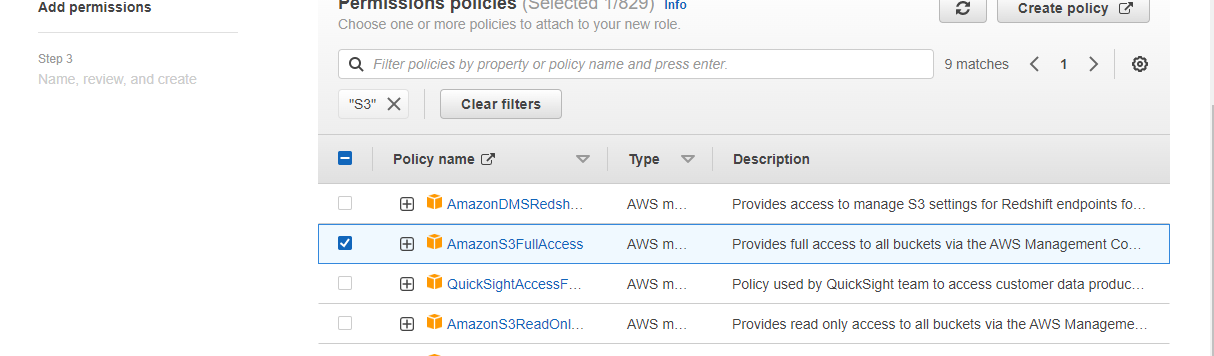
**–1. Create IAM role**

****

1. Search **IAM** and above page will appear
2. Create a role then page will appear select **AWS account** and next

****

1. Search for **S3 full access** as in below image then click on next

****

1. After this it will ask for **role name** give a role name and create a role
2. Locate your role and open it their copy the **ARN number** in notepad

* Note: Required services of **AWS:**
* ARN: Amazon resource names
* S3- simple storage service
* IAM- integrated account management
* DMS(Terraform)
* Lambda

Required services of **Azure:**

- Blob Storage

- ADF

- Data bricks

**– Creating storage object and exchanging the ARN’s**

**–create storage integration object**

create or replace storage integration sky\_aws\_int

type = EXTERNAL\_STAGE

STORAGE\_PROVIDER = S3

ENABLED = TRUE

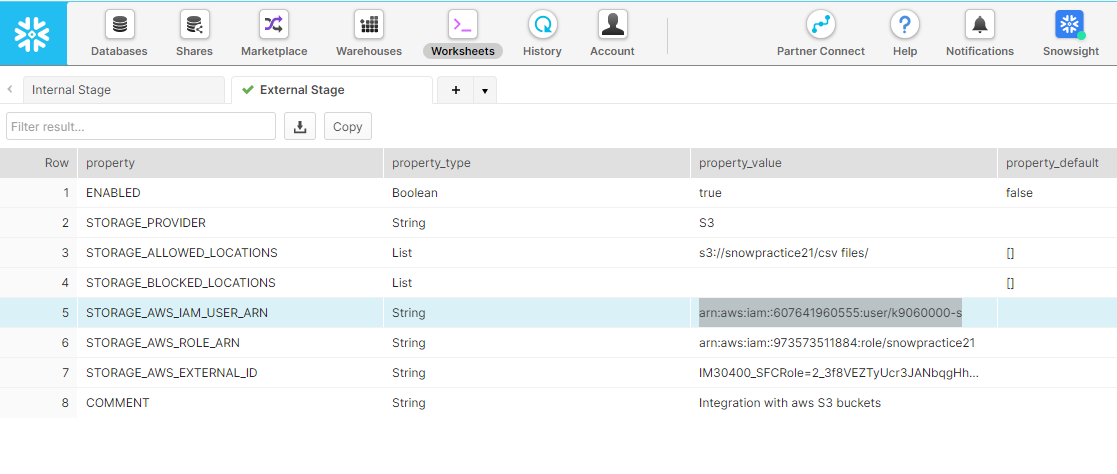
STORAGE\_AWS\_ROLE\_ARN = 'arn:aws:iam::973573511884:role/snowpractice21'

STORAGE\_ALLOWED\_LOCATIONS = ('s3://snowpractice21/csv files/')

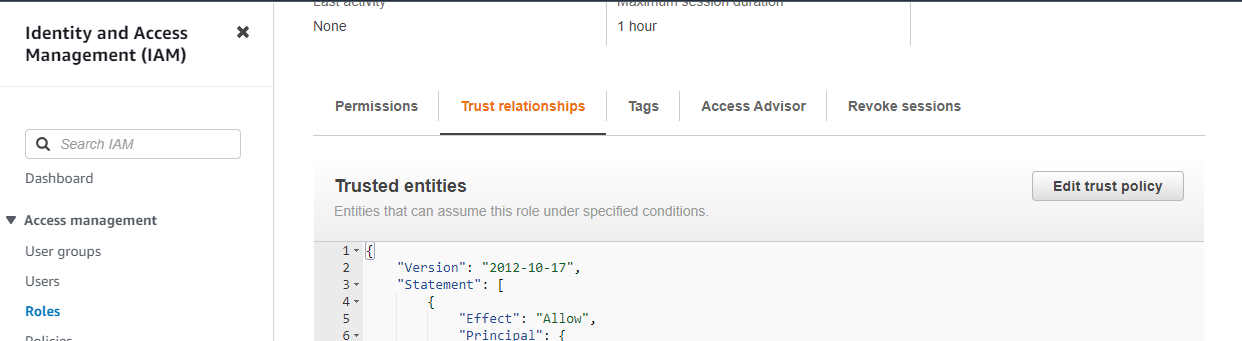
COMMENT = 'Integration with aws S3 buckets';

**-- get external\_id and update it in S3**

**DESC integration sky\_aws\_int;**



Now copy the **STORAGE\_AWS\_IAM\_USER\_ARN** and paste it in the roles



The **edit trust policy** and change the ARn no. that you copied and click on **update the policy**

**-- create a stage**

create or replace stage stage\_aws

URL = 's3://snowpractice21/csv files/'

STORAGE\_INTEGRATION = sky\_aws\_int;

list @stage\_aws;

**--loading data from aws s3 using copy command**

**--create a table**

create or replace table user\_email(

id number,

first\_name varchar(100),

last\_name varchar(100),

email varchar(100),

gender varchar(6),

about\_me varchar(500)

);

select \* from user\_email;

**--create a ff\_object**

create or replace file format emp.emp.aws\_ff

type = csv

field\_delimiter = ','

skip\_header = 1

field\_optionally\_enclosed\_by = '\042'

empty\_field\_as\_null = TRUE;

**--create stage**

create or replace stage stage\_aws

URL = 's3://snowpractice21/csv files/'

STORAGE\_INTEGRATION = sky\_aws\_int

file\_format = aws\_ff;

list @stage\_aws;

**--copying data from aws s3**

copy into user\_email

from @stage\_aws

files = ('01\_sample\_user\_email.csv');

select \* from user\_email;

**3. Zero Copy Clone**

**--cloning a table**

create or replace table user\_email\_clone

CLONE user\_email;

select \* from user\_email\_clone;

**--cloning a schema**

create or replace schema tasks\_clone

CLONE tasks;

**--cloning a database**

create or replace database emp\_clone

CLONE emp;

**--updating data in source table and cloned objects and observing both the tables**

**-- making changes in source table**

select \* from user\_email;

select \* from user\_email\_clone;

update user\_email set first\_name = 'Zach' where id = 1;

**--listing records of cloned**

select \* from user\_email where id = 1;

**---Time Travel**

**--setting a tt to a table**

create or replace table time\_travel\_ex (id number, name string)

DATA\_RETENTION\_TIME\_IN\_DAYS = 1;

select \* from time\_travel\_ex

**--setting a tt to schema**

create or replace schema time\_travel\_ex

DATA\_RETENTION\_TIME\_IN\_DAYS = 1;

**--setting tt for db**

create or replace database time\_travel\_ex

DATA\_RETENTION\_TIME\_IN\_DAYS = 1;

**-- altering retention period on table**

alter table time\_travel\_ex

set DATA\_RETENTION\_TIME\_IN\_DAYS = 1;

**--listing a retention period**

show tables like 'time\_travel\_ex';

show schemas like 'time\_travel\_ex';

show databases like 'time\_travel\_ex';

Show parameters like '%retention%' in account;

show parameters like '%DATA\_RETENTION\_TIME\_IN\_DAYS%' in table practice2;

show parameters like '%DATA\_RETENTION\_TIME\_IN\_DAYS%' in schema practice;

show parameters like '%DATA\_RETENTION\_TIME\_IN\_DAYS%' in database practice;

**--querying historical data**

**--case1: updating some data in user\_email ---> ““OFFSET””**

select \* from user\_email where id =1; *---first name was zach*

update user\_email set first\_name = 'snowflake' where id =1; *---change to snowflake*

select \* from user\_email where id =1; *---update to snowflake*

*--querying historical data by offset.*

select \* from user\_email at (offset => -60\*5) where id = 1; *— 5 is in minutes*

**--case2: updating some data in user\_email ---> ""TIMESTAMP""**

select \* from user\_email;

update user\_email set first\_name = 'snow' where id = 1;

**--querying historical data by timestamp.**

select current\_timestamp;

select \* from user\_email at (timestamp => '2023-04-21 05:11:31.768 -0700' ::timestamp);

**--case3: updating some data in user\_email ---> ""BEFORE STATEMENT""**

select \* from EMPLOYEE;

UPDATE employee set empname = 'Borgave' where empid =1; *---amogh was name change to borgave*

**--querying historical data by offset.**

select \* from employee before (statement => '01abc662-3200-b63d-0003-e7120002db62'); --copy query id from history

**--restoring data from table**

create or replace table restore\_user\_email CLONE user\_email at (offset => -60\*40);

select \* from restore\_user\_email;

create or replace table res\_employee CLONE employee at (timestamp => '2023-04-21 05:11:31.768 -0700' ::timestamp);

select \* from res\_employee; *–same for schema and database*

create or replace table res\_employee CLONE employee before (statement => '01abc662-3200-b63d-0003-e7120002db62');

select \* from res\_employee;

**--CACHING**

**--Two type of cache**

**--1. result cache**

**--2. local cache**

**----Result cache**

**--query is fetching from storage layer (remote disk) --> 1st time --> more time**

select \* from COLUMN\_MISMATCH; **--515ms**

**--2nd time -- query -- fetching from result cache**

select \* from COLUMN\_MISMATCH; **--60ms**

**--3rd time -- sub query -- fetching from metadata**

select count(\*) from column\_mismatch;  **--92ms**

**--4th time -- sub query -- fetching from remote disk**

select \* from column\_mismatch where gender = 'F'; **--400ms**

select \* from column\_mismatch where gender = 'M'; **--422ms**

**---Local Disk cache**

**--turn off result cache**

alter session set use\_cached\_result = false;

**--query is fetching from storage layer (remote disk) --> 1st time --> more time**

select \* from COLUMN\_MISMATCH; **--R=515ms L=605MS**

**--2nd time -- query -- fetching from result cache**

select \* from COLUMN\_MISMATCH;  **--R=60ms L=425MS**

**--3rd time -- sub query -- fetching from metadata**

select count(\*) from column\_mismatch; **--R=92ms L=27MS**

**--4th time -- sub query -- fetching from remote disk**

select \* from column\_mismatch where gender = 'F'; **--R=400ms L=406MS**

select \* from column\_mismatch where gender = 'M';  **--R=422ms L=460MS**

**--Unloading data to EXTERNAL STAGE**

**--unloading options**

-**-OVERWRITE** = TRUE | FALSE - specifies to overwrite existing files

--**SINGLE** = TRUE | FALSE - specifies whether to generate a single file or multiple files

--**MAX\_FILE\_SIZE** = NUMBER - maximum file size

-**-INCLUDE\_QUERY\_ID** =TRUE | FALSE - specifies whether to uniquely identify unloaded files by universally unique identifier

--**DETAILED\_OUTPUT** = TRUE | FALSE - shows the path and name for each file, its size, and the number of rows that were unloaded to the file

**--create storage integration object**

create or replace storage integration sky\_aws\_int

type = EXTERNAL\_STAGE

STORAGE\_PROVIDER = S3

ENABLED = TRUE

STORAGE\_AWS\_ROLE\_ARN = 'arn:aws:iam::973573511884:role/snowpractice21'

STORAGE\_ALLOWED\_LOCATIONS = ('s3://snowpractice21/unloading/ , s3://snowpractice21/csv files/')

COMMENT = 'integration with aws s3 buckets';

**--altering storage integration**

ALTER STORAGE integration sky\_aws\_int

SET STORAGE\_ALLOWED\_LOCATIONS = ('s3://snowpractice21/csv/' , 's3://snowpractice21/unloading/');

**-- get external\_id and update it in S3**

DESC integration sky\_aws\_int;

**--create or replace file\_format**

create or replace file format unloading\_ff

type = 'csv', field\_delimiter = '|', skip\_header = 0, empty\_field\_as\_null = true;

**--create a stage**

create or replace stage unload\_stage

url = 's3://snowpractice21/unloading/'

storage\_integration = sky\_aws\_int

file\_format = unloading\_ff;

**--copy command for unloading data to external stage**

copy into @unload\_stage

from customer\_psv;

list @unload\_stage;

**--copy command for unloading data to external stage with data name**

copy into @unload\_stage/customer

from customer\_psv;

**--copy command for unloading data to external stage with unload properties**

copy into @unload\_stage/customer

from customer\_psv

max\_file\_size = 2000000

overwrite = true

single = true  *--when data is big it will create single file*

detailed\_output = true;

**-- Unloading data to internal stage**

-**-create file format object**

create or replace FILE FORMAT unload\_ff

type = csv

field\_delimiter = '|'

skip\_header = 0

empty\_field\_as\_null = TRUE;

**-- create a stage**

create or replace stage int\_stage

file\_format = unload\_ff ;

**-- copying file into internal stage by copy command**

copy into @int\_stage/customer/

from customer\_psv

single = true

overwrite = true

detailed\_output = true;

**--list a stage**

list @int\_stage;

**-- downloading file from internal stage to local machine by using GET command**

get @int\_stage/customer/data file://C:\Users\BhandariRaviraj\snowflakedata;

**--downloading file from table --@% (table stage)**

get @%customer\_psv/customer/data file://C:\Users\BhandariRaviraj\snowflakedata;

**--downloading file from user stage --@~**

get @~user\_stage/customer/data file://C:\Users\BhandariRaviraj\snowflakedata;

**--DATA SHARING TO SNOWFLAKE USERS**

**--Data sharing properties**

--1. grant usage --> database, schema

--2. grant select --> tables

**--creating database data\_sharing**

create or replace database share\_db;

use database share\_db;

create or replace schema share\_db.share;

**--create table**

create or replace table customer\_csv\_share (

CUSTOMER\_PK number(38,0),

SALUTATION varchar(10),

FIRST\_NAME varchar(20),

LAST\_NAME varchar(20),

GENDER varchar(1),

MARITAL\_STATUS varchar(20),

DAY\_OF\_BIRTH date,

BIRTH\_COUNTRY varchar(60),

EMAIL\_ADDRESS varchar(60),

CITY\_NAME varchar(50),

ZIP\_CODE varchar (50),

COUNTRY\_NAME varchar(50),

GMT\_TIMEZONE\_OFFSET number(10,2),

PREFERRED\_CUST\_FLAG boolean,

REGISTRATION\_TIME VARCHAR(30)

);

select \* from customer\_csv\_share;

**-- ff for loading**

create or replace file format fileformat\_share

type = csv

field\_delimiter = ','

skip\_header = 1

field\_optionally\_enclosed\_by = '\042'

empty\_field\_as\_null = TRUE;

**--creating a external stage**

create or replace stage aws\_stage

url='s3://snowpractice21/csv files/'

STORAGE\_INTEGRATION = sky\_aws\_int;

ALTER STORAGE integration sky\_aws\_int

SET STORAGE\_ALLOWED\_LOCATIONS = ('s3://snowpractice21/csv/' , 's3://snowpractice21/unloading/' , 's3://snowpractice21/csv files/');

list @aws\_stage;

**--not loading csv filr from aws to our table**

copy into customer\_csv\_share

from @aws\_stage

file\_format = fileformat\_share

files = ('customer\_1.csv');

select \* from customer\_csv\_share;

**-- we need to do these in providers account (sender's account) who want to share the data**

**--create a share**

create or replace share sky\_share;

**--list a share**

show shares;

**-- granting permission to share to use database**

grant usage on database share\_db to share sky\_share;

**-- granting permission to share to use schema**

grant usage on schema share\_db.share to share sky\_share;

**-- granting permission to share to use table**

grant select on table share\_db.share.customer\_csv\_share to share sky\_share;

**-- granting permission to use complete schema by share**

grant select on all tables in schema share\_db.share to share sky\_share;

**-- granting permission to use complete database by share**

grant select on all tables in database share\_db to share sky\_share;

**--showing grants permission to share**

show grants to share sky\_share;

**--giving a account name of consumer (receiver)**

alter share sky\_share add account = pk83112; -- consumer

-- this account is having the permission to use my db, sc, tb

**--things to do in consumer account (receiver's account: i have taken richa's account)**

show shares;

create or replace database db\_share from share im30400.sky\_share; ---providers account

select \* from db\_share.share.customer\_csv\_share;

**-- TABLES**

**--types of tables**

**--1. permanent**

**--2. transient**

**--3. temporary**

create or replace schema table\_types;

**--where to see table type**

SHOW TABLES in SCHEMA EMP;

**--creating trans, temp table**

CREATE OR REPLACE TRANSIENT TABLE table\_types.tran\_table(id number);

CREATE OR REPLACE TEMPORARY TABLE table\_types.temp\_table(name string);

SHOW TABLES in SCHEMA table\_types;

**--1. PERMANENT TABLES/SCHEMAS**

**create or replace table permanent (id number);**

**--default table type in sf**

**--these are the regular and common tables**

**--tables exists until we drop them explicitly**

**--time travel retention period is 0-90 days depends on sf edition**

**--7 dys of fail safe period**

**--2. TRANSIENT Tables/Schemas**

**--similar to permanent tables but with 1 day retention period**

**--no fail-safe period**

**--tables exist until we drop them**

**--useful when data protection is not required**

**--defining stage tables as Transient**

**--create transient table**

CREATE OR REPLACE TRANSIENT TABLE table\_types.tran\_table(id number);

**--create transient schema**

CREATE OR REPLACE TRANSIENT SCHEMA table\_schema; --all table in transient table will be transient by default

CREATE OR REPLACE table table\_schema.tran\_table(id number);

show tables in schema table\_schema;

**-- we cannot alter the retention period of transient tables its default 1 day**

alter table table\_schema.tran\_table

set data\_retention\_time\_in\_days = 1;

**-- can we undrop table schema, db transient in 24 hr**

drop table table\_schema.tran\_table;

**--undrop this transient table within 24 hrs**

undrop table table\_schema.tran\_table;

--same for transient schema

**--3. TEMPORARY tables/schema**

**--table exists only within the session**

**--1 day retention period**

**--no fail safe period**

**--we can only create table**

**--create a temporary table**

CREATE OR REPLACE TEMPORARY TABLE temp\_table(id number);

CREATE OR REPLACE TEMPORARY TABLE temp\_table2 (name string);

insert into temp\_table2 values ('Yogesh');

insert into temp\_table2 values ('Atharva');

insert into temp\_table2 values ('Varun');

insert into temp\_table2 values ('Jeet');

insert into temp\_table2 values ('Amogh');

select \* from temp\_table2; --temp table will only be assessed from 1 session (worksheet)

select \* from temp\_table2 order by name DESC;

**--drop and undrop**

drop table temp\_table2;

undrop table temp\_table2;

desc table temp\_table2;

**---SENARIO 1**

**---when we drop the temporary table and create another temporary table with the same name**

**---then we can not undrop the table that we dropped so here we can do is:**

**---1. alter table <tablename> rename to <tablename>;**

**---SENARIO 2**

**---if we are creating the table with same name in temporary and permanent, then for —that worksheet it will take temporary**

**---and we need to access permanently from another worksheet.**

**--to check which kind of table**

show schemas in database emp;

**--EXTERNAL STAGE (AMAZON S3 CLOUD)**

use database emp;

**--create a schema for external tables**

create schema if not exists ext\_table;

create or replace schema ext\_table;

use schema ext\_table;

**--create file format object**

create or replace file format csv\_ff

type = csv

field\_delimiter =','

skip\_header = 1

empty\_field\_as\_null = TRUE;

**--create storage integration object**

create or replace storage integration sky\_aws\_int

type = EXTERNAL\_STAGE

storage\_provider = S3

ENABLED = TRUE

STORAGE\_AWS\_ROLE\_ARN = 'arn:aws:iam::973573511884:role/snowpractice21'

STORAGE\_ALLOWED\_LOCATIONS = ('s3://snowpractice21/csv files/')

COMMENT = 'Interagtion with aws s3 buckets';

**-- create stage object with integration object and file format object**

create or replace stage ext\_stage

url = 's3://snowpractice21/csv files/'

storage\_integration = sky\_aws\_int

file\_format = csv\_ff;

--list stage

list @ext\_stage;

**--create a external table in simple way**

**--method 1: data will be in json format**

create or replace external table user\_email\_ext

with

location = @ext\_stage

pattern = '.\*user.\*'

file\_format = csv\_ff;

select \* from user\_email\_ext;

DESC integration sky\_aws\_int;

**--method 2: data will be in tabular format**

create or replace external table user\_ext (

id number as (value:c1::number),

first\_name varchar as (value:c2::varchar),

last\_name varchar as (value:c3::varchar),

email varchar as (value:c4::varchar),

gender varchar as (value:c5::varchar),

aboutme varchar as (value:c6::varchar)

)

with

location = @ext\_stage

pattern = '.\*user.\*'

file\_format = csv\_ff;

select \* from user\_ext;

**--practice queries for external**

select \* from user\_ext where gender = 'F';

select \* from user\_ext where email like '%@gmail.com';

**--to see external tables**

desc external table user\_ext;

**--total external tables**

show external tables;

**--to see the files if referring or how many files are loaded**

select distinct metadata$filename from user\_ext;

**--SNOWPIPE**

**--loading data from aws S3 using SNOWPIPE**

**--create storage integration object**

create or replace storage integration sky\_aws\_int

type = EXTERNAL\_STAGE

STORAGE\_PROVIDER = S3

ENABLED = TRUE

STORAGE\_AWS\_ROLE\_ARN = 'arn:aws:iam::973573511884:role/snowpractice21'

STORAGE\_ALLOWED\_LOCATIONS =('s3://snowpractice21/csv files/', 's3://snowpractice21/csv/')

COMMENT = 'Integration with aws s3 buckets';

**--create a table**

create or replace table customer\_csv (

CUSTOMER\_PK number(38,0),

SALUTATION varchar(10),

FIRST\_NAME varchar(20),

LAST\_NAME varchar(20),

GENDER varchar(1),

MARITAL\_STATUS varchar(20),

DAY\_OF\_BIRTH date,

BIRTH\_COUNTRY varchar(60),

EMAIL\_ADDRESS varchar(60),

CITY\_NAME varchar(50),

ZIP\_CODE varchar (50),

COUNTRY\_NAME varchar(50),

GMT\_TIMEZONE\_OFFSET number(10,2),

PREFERRED\_CUST\_FLAG boolean,

REGISTRATION\_TIME VARCHAR(30)

);

select \* from customer\_csv;

**--create a ff**

create or replace file format aws\_ff

type = csv

field\_delimiter = ','

skip\_header = 1

field\_optionally\_enclosed\_by = '\042'

empty\_field\_as\_null = TRUE;

**--create a stage**

create or replace stage pipe\_stage

url = 's3://snowpractice21/csv/'

storage\_integration = sky\_aws\_int

file\_format = aws\_ff;

**--list stage**

list @pipe\_stage;

**--create a pipe**

--new file will be automaticallly loaded in table

create or replace pipe aws\_pipe

auto\_ingest = True

as

copy into customer\_csv

from @pipe\_stage;

**--desc pipe**

desc pipe aws\_pipe;

--arn:aws:sqs:ap-south-1:607641960555:sf-snowpipe-AIDAY26SIQBVTSB2GUX6B-Ko9-PS

**Paste the above Arn in the aws bucket properties in event notification**

We need to create an event notification and paste the arn copied.

**--2 methods for setting notifications to the snowpipe:**

**--a. SQS Queue --> copy arn of snowflake and paste in event notification**

**--b. SNS Topic --> create a sns topic and create event notification**

**--a. SQS queue ::**

--list the records of the table

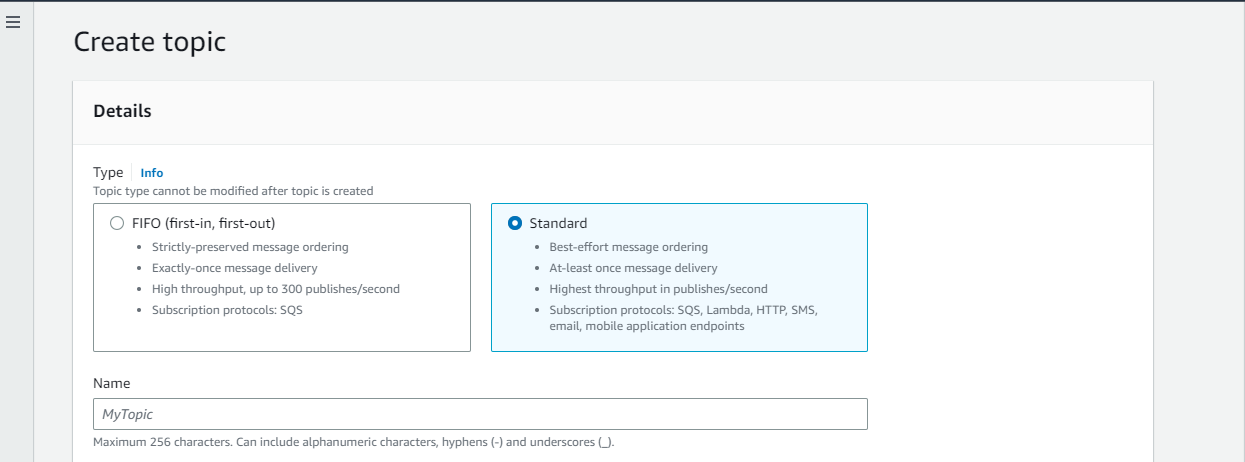
select \* from customer\_csv;

**--check pipe status**

select system$pipe\_status('aws\_pipe');

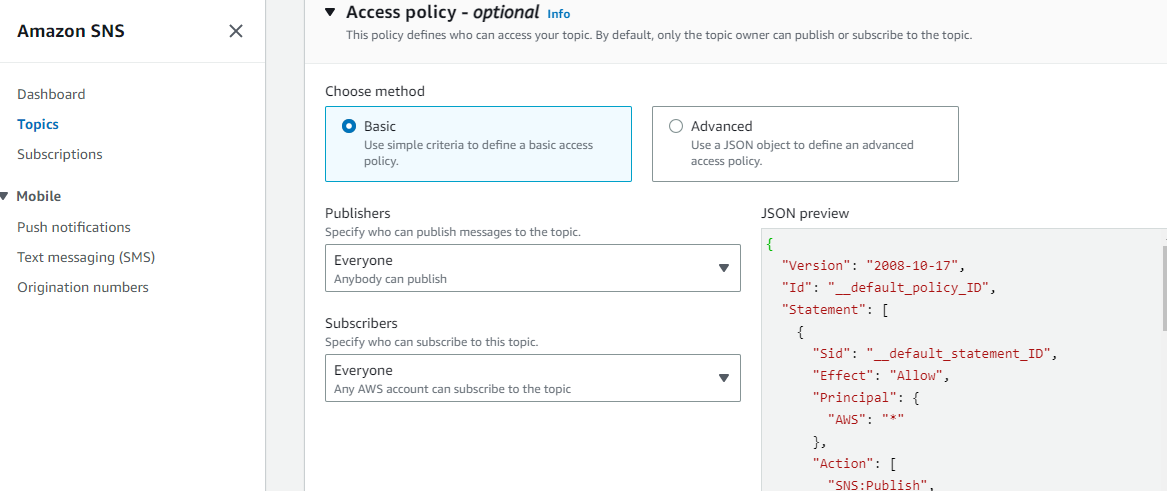
Now for SNS we need to create an SNS on aws

Step1. : create an topic and crete standard as in image



Step2: Go to access policy and select everyone for subscriber and publishers

And create a policy

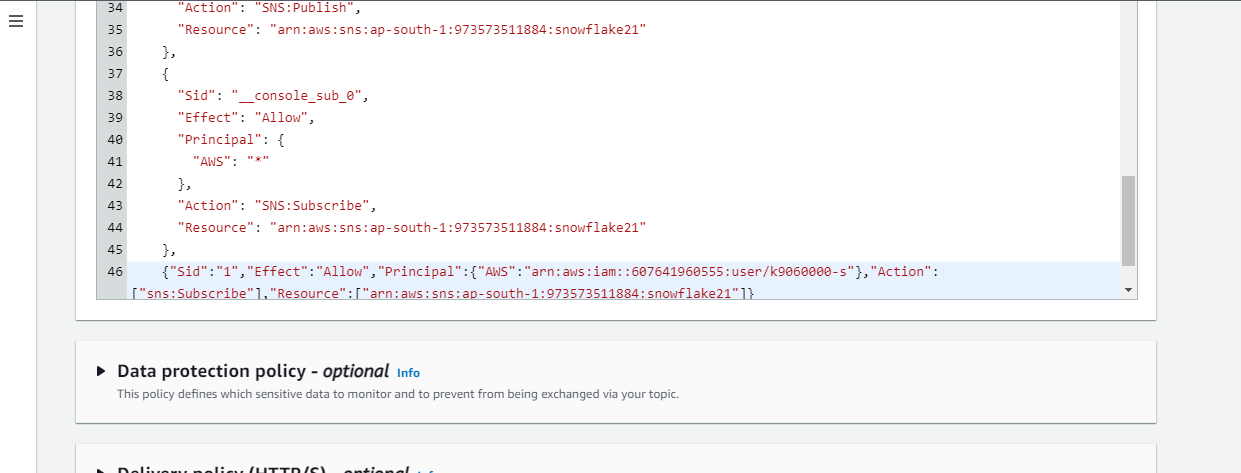


Step3: write the below query

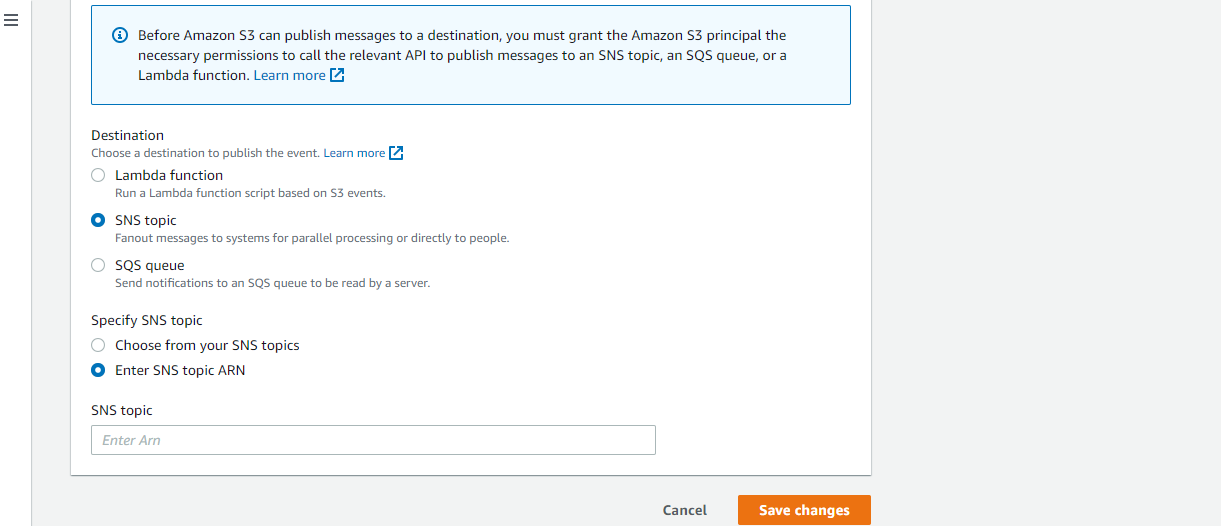
**--b. SNS queue**

select system$get\_aws\_sns\_iam\_policy ('arn:aws:sns:ap-south-1:973573511884:snowflake21');

Step4: After this it will generate arn copy that and paste it in our policy that we created in json file as in image below and save changes



Step5: create a event notification and save changes



Step6: open worksheet and

**--create a pipe**

**--new file will be automatically loaded in table**

create or replace pipe aws\_pipe\_sns

auto\_ingest = True

aws\_sns\_topic = 'arn:aws:sns:ap-south-1:973573511884:snowflake21'

as

copy into customer\_csv

from @pipe\_stage;

**--desc pipe**

desc pipe aws\_pipe\_sns;

**--listing records of table**

select \* from customer\_csv;

**--JSON file data**

**--process of loading**

**--a. create a stage that points to json files in AWS?azure**

**--b. create stage table to store the raw data**

**--c.create with single variant column**

**--d. copy data to stage table using copy command**

**--now parse the RAW data based on the json file content**

**--normal json**

**--json with array**

**--json with nested**

**--json with all types**

**--load this parsed data to a table**

**--1. JSON data and variant colum**n

**--variant column: holds all types of data into single column**

**--loading data from internal stage**

**--creating schema**

create or replace schema js\_on;

**--create file format**

create or replace file format my\_json\_format

type = json

strip\_outer\_array = true;

**--creating internal stage**

create or replace stage stage\_json

file\_format = my\_json\_format;

**--list the stage**

list @stage\_json;

**--upload file to stage**

put file://C:\Users\BhandariRaviraj\jsondata\internal\_stage\author.json @stage\_json;

**--creating table with variant column**

create or replace table author (json\_data variant);

**--copy command to load json data into author table with variant column**

copy into author

from @stage\_json/author.json

file\_format = my\_json\_format;

--file =('')

**--listing records in author stage table**

select \* from author;

**--parsing of json data into column format**

**--querying (listing) individual columns with json type data**

select

json\_data:AuthorName as AuthorName,

json\_data:Category as Category

from author;

**--accessing array json data**

--getting array size

select array\_size(json\_data:Category[0]:best\_edition) size from author;

**--listing records of array**

select json\_data:Category[0]:best\_edition from author;

**--accessing nested json data**

**--by giving indexing**

**--ex:1**

select

json\_data:AuthorName,

json\_data:Category[0]:CategoryName,

json\_data:Category[1]:CategoryName

from author;

**--ex: 2**

select

json\_data:AuthorName,

json\_data:Category[0]:Genre[0]:GenreName,

json\_data:Category[0]:Genre[0]:Novel[0]:sales

from author;

**--parse json data to get columns and table**

select \* from author;

**--parsing json data to columns and it will remove quotes --by giving datatypes of columns and casting to specific data type**

**select**

json\_data:AuthorName::String as Author\_Name,

json\_data:Category[0]:CategoryName::string as Fiction\_Category ,

json\_data:Category[1]:CategoryName::string as Non\_Fiction\_Category

from author;

**--more detailed**

select

JSON\_DATA:AuthorName::String as Author\_Name,

JSON\_DATA:Category[0]:CategoryName::string as Fiction\_Category ,

JSON\_DATA:Category[0]:Genre[0]:GenreName::string,

JSON\_DATA:Category[0]:Genre[1]:GenreName::string,

JSON\_DATA:Category[1]:CategoryName::string,

JSON\_DATA:Category[1]:Genre[0]:GenreName::string,

JSON\_DATA:Category[1]:Genre[1]:GenreName::string

from author;

**--2 methods for getting data into our main table**

**--a.by using flattening**

**--b.by just prasing all data and union all**

**--a. Flattening**

**--is a process of unpacking the semi-structured data into a columnar format by converting arrays into different rows of data.**

--using the lateral flatten function we can explode arrays into individual JSON objects.

--flattern data into columnar format --> converting diff arrays into rows -->

**--flattening all data into rows from json arrays**

select

JSON\_DATA:AuthorName::string as Author\_Name,

Flatten\_Category.VALUE:CategoryName::string as Category\_Name,

Flatten\_Genre.VALUE:GenreName::string as Genre\_Name,

Flatten\_Novel.VALUE:Novel::string as Novel\_Name,

Flatten\_Novel.VALUE:Sales::number as Sales\_in\_Millions

from author

,LATERAL FLATTEN (input => JSON\_DATA:Category) as Flatten\_Category

,LATERAL FLATTEN (input => Flatten\_Category.VALUE:Genre) as Flatten\_Genre

,LATERAL FLATTEN (input => Flatten\_Genre.VALUE:Novel) as Flatten\_Novel

**--loading this flattened data (column type) into our main table**

create or replace table author\_data as

select

JSON\_DATA:AuthorName::string as Author\_Name,

Flatten\_Category.VALUE:CategoryName::string as Category\_Name,

Flatten\_Genre.VALUE:GenreName::string as Genre\_Name,

Flatten\_Novel.VALUE:Novel::string as Novel\_Name,

Flatten\_Novel.VALUE:Sales::number as Sales\_in\_Millions

from author

,LATERAL FLATTEN (input => JSON\_DATA:Category) as Flatten\_Category

,LATERAL FLATTEN (input => Flatten\_Category.VALUE:Genre) as Flatten\_Genre

,LATERAL FLATTEN (input => Flatten\_Genre.VALUE:Novel) as Flatten\_Novel

select \* from author\_data;

select \* from author;

**--b. substr, substring, parse the data in copy command**

**--create a table**

create or replace table home\_sales(

city string,

zip string,

state string,

type string default 'Residential',

sale\_date timestamp\_ntz,

price string

);

**--list stage**

list @stage\_json

**--upload file**

put file://C:\Users\BhandariRaviraj\jsondata\internal\_stage\sales.json @stage\_json;

**--copy command $--accessing the column names, substr --extracting**

copy into home\_sales(city, state, zip , sale\_date, price)

from (select substr($1:location.state\_city,4),

substr($1:location.state\_city,1,2),

$1:location.zip,

to\_timestamp\_ntz($1:sale\_date),

$1:price

from @stage\_json/sales.json)

on\_error = "CONTINUE";

**--list records**

select \* from home\_sales;

**--working json with table stage**

--create a table by ff properties for stage table

create or replace table home\_table(json variant)

stage\_file\_format = (type= 'JSON');

select \* from home\_table;

**--list stage**

list @%home\_table;

**--upload file to table stage**

put file://C:\Users\BhandariRaviraj\jsondata\internal\_stage\sales.json @%home\_table;

**--list records from stage only**

select \* from home\_table;

select \* from @%home\_table;

**--copy command to load data into table**

copy into home\_table;

**-- loading JSON data from external stage**

create or replace schema jj\_son;

use schema jj\_son;

**--create storage integration object**

create or replace storage integration sky\_aws\_int

type = EXTERNAL\_STAGE

storage\_provider = S3

ENABLED = TRUE

STORAGE\_AWS\_ROLE\_ARN = 'arn:aws:iam::973573511884:role/snowpractice21'

STORAGE\_ALLOWED\_LOCATIONS = ('s3://snowpractice21/csv files/', 's3://snowpractice21/others/')

COMMENT = 'Interagtion with aws s3 buckets';

**--creating ff**

create or replace file format my\_json

type = json

strip\_outer\_array = true;

**--create a stage**

create or replace stage json\_stage

storage\_integration = sky\_aws\_int

file\_format = my\_json

url = 's3://snowpractice21/others/';

**--list stage**

list @json\_stage;

**--create a stage table to raw json data**

create or replace table author

(json\_data variant);

**--list records**

select \* from author;

desc table author;

**--copy command to load raw data to stage table**

copy into author

from @json\_stage/author.json

file\_format = my\_json;

**--parsing of json data into column format**

**--accessing simple json data**

**--using $ for query json data**

select $1:AuthorName from author;

**--accessing data from stage location directly**

select $1:AuthorName from @json\_stage/author.json (file\_format => 'my\_json');

**--querying (listing) individual columns with json type data**

select

json\_data:AuthorName as AuthorName,

json\_data:Category as Category

from author;

**--accessing array json data**

**--getting array size**

select array\_size(json\_data:Category[0]:best\_edition) size from author;

**--listing records of array**

select json\_data:Category[0]:best\_edition from author;

**--accessing nested json data**

**--by giving indexing**

--ex:1

select

json\_data:AuthorName,

json\_data:Category[0]:CategoryName,

json\_data:Category[1]:CategoryName

from author;

--ex: 2

select

json\_data:AuthorName,

json\_data:Category[0]:Genre[0]:GenreName,

json\_data:Category[0]:Genre[0]:Novel[0]:sales

from author;

**--parse json data to get columns and table**

select \* from author;

--parshing json data to columns and it will remove quotes --by giving datatypes of columns and casting to specific dattype

select

json\_data:AuthorName::String as Author\_Name,

json\_data:Category[0]:CategoryName::string as Fiction\_Category ,

json\_data:Category[1]:CategoryName::string as Non\_Fiction\_Category

from author;

**--more detailed**

select

JSON\_DATA:AuthorName::String as Author\_Name,

JSON\_DATA:Category[0]:CategoryName::string as Fiction\_Category ,

JSON\_DATA:Category[0]:Genre[0]:GenreName::string,

JSON\_DATA:Category[0]:Genre[1]:GenreName::string,

JSON\_DATA:Category[1]:CategoryName::string,

JSON\_DATA:Category[1]:Genre[0]:GenreName::string,

JSON\_DATA:Category[1]:Genre[1]:GenreName::string

from author;

**--2 methods for getting data into our main table**

**--a.by using flattening**

**--b.by just prasing all data and union all**

**--a. Flattening**

**--is a process of inpacking the semi-structured data into a columnar format by converting arrys into different rows of data.**

**--using the lateral flatten function we can explode arrays into individual JSON object.**

**--flattern data into columnar format --> converting diff arrays into rows -->**

**--flattening all data into rows from json arrays**

create or replace table authors\_data as

select

JSON\_DATA:AuthorName::string as Author\_Name,

Flatten\_Category.VALUE:CategoryName::string as Category\_Name,

Flatten\_Genre.VALUE:GenreName::string as Genre\_Name,

Flatten\_Novel.VALUE:Novel::string as Novel\_Name,

Flatten\_Novel.VALUE:Sales::number as Sales\_in\_Millions

from author

,LATERAL FLATTEN (input => JSON\_DATA:Category) as Flatten\_Category

,LATERAL FLATTEN (input => Flatten\_Category.VALUE:Genre) as Flatten\_Genre

,LATERAL FLATTEN (input => Flatten\_Genre.VALUE:Novel) as Flatten\_Novel

**--loading this flattend data (column type) into our main table**

create or replace table authors\_data as

select

JSON\_DATA:AuthorName::string as Author\_Name,

Flatten\_Category.VALUE:CategoryName::string as Category\_Name,

Flatten\_Genre.VALUE:GenreName::string as Genre\_Name,

Flatten\_Novel.VALUE:Novel::string as Novel\_Name,

Flatten\_Novel.VALUE:Sales::number as Sales\_in\_Millions

from author

,LATERAL FLATTEN (input => JSON\_DATA:Category) as Flatten\_Category

,LATERAL FLATTEN (input => Flatten\_Category.VALUE:Genre) as Flatten\_Genre

,LATERAL FLATTEN (input => Flatten\_Genre.VALUE:Novel) as Flatten\_Novel

select \* from authors\_data;

select \* from author;

**--b. substr, substring, parse the data in copy command**

**--create a table**

create or replace table home\_sales(

city string,

zip string,

state string,

type string default 'Residential',

sale\_date timestamp\_ntz,

price string

);

**--list stage**

list @json\_stage;

**--copy command $--accessing the column names, substr --extracting**

copy into home\_sales(city, state, zip , sale\_date, price)

from (select substr($1:location.state\_city,4),

substr($1:location.state\_city,1,2),

$1:location.zip,

to\_timestamp\_ntz($1:sale\_date),

$1:price

from @json\_stage/sales.json)

on\_error = "CONTINUE";

**--list records**

select \* from home\_sales;

**--OBJECT CONSTRUCT**

**--creating json object type data with column name and values -- using object\_construc function**

select object\_construct ('firstName','Raviraj','lastName','Bhandari', 'gender','male','age',22,'number','7030655778','address',null);

**--creating json type data -we get column name -col1, col2**

select object\_construct(\*) from values ('firstName','Raviraj','lastName','Bhandari', 'gender','male','age',22,'number','7030655778','address',null);

**--parsing json data (create tabular type data by giving column name) (simple 2 columns and no nesting)**

select 'People' as Groupid, 'Raviraj' as FirstName, 'Bhandari' as LastName UNION Select 'People' as Groupid, 'Yogesh' as FirstName, 'Jadhav' as LastName;

**--checking all records in column format of JSON**

select \* from (select 'People' as Groupid, 'Raviraj' as FirstName, 'Bhandari' as LastName UNION select 'People' as Groupid, 'Yogesh' as FirstName, 'Jadhav' as LastName);

**--ARRAY\_AGG**

**--doing with array\_agg to create json data of address of people.json file**

select groupid, array\_agg(object\_construct('firstname',firstname ,'lastname',lastname)) from (Select 'People' as Groupid, 'Raviraj' as FirstName, 'Bhandari' as LastName UNION

Select 'People' as Groupid, 'Yogesh' as FirstName, 'Jagtap' as LastName )group by groupid;

**--doing with array\_agg to get json type of data of ADDRESS of people.json file**

select addressid, array\_agg(object\_construct('city',city, 'state',state, 'cities',cities, 'states',states))

from(Select 'address' as addressid, 'Mumbai' as city, 'Maharashtra' as state, 'Lucknow' as cities, 'UP' as states

UNION

select 'address' as addressid, 'Pune' as city, 'Maharashtra' as state, 'Gaziabad' as cities, 'UP' as states

UNION ALL

select 'address' as addressid, 'Nashik' as city, 'Maharashtra' as state, 'Noida' as cities, 'UP' as states) group by addressid;

**--parsing json data into tabular format for address**

select \* from (Select 'address' as addressid, 'Mumbai' as city, 'Maharashtra' as state, 'Lucknow' as cities, 'UP' as states

UNION

select 'address' as addressid, 'Pune' as city, 'Maharashtra' as state, 'Gaziabad' as cities, 'UP' as states

UNION ALL

select 'address' as addressid, 'Nashik' as city, 'Maharashtra' as state, 'Noida' as cities, 'UP' as states);

**--doing with array\_agg to get nested json type data of PEOPLE.json file**

select groupid, array\_agg(object\_construct('firstname', firstname, 'lastname', lastname, 'gender', gender, 'age', age, 'number', number))

from (select 'Employee' as groupid, 'Raviraj' as firstname, 'Bhandari' as lastname, 'male' as gender, '22' as age, '7030655778' as number

UNION

select 'Employee' as Groupid, 'Yogesh' as firstname, 'Jagtap' as lastname, 'female' as gender, '22' as age, '6969696969' as number) group by groupid;

**--JSON PROCESSING**

**--object construct: converting normal data to json data**

**--array\_aggregate: returns the input values (of type array), if the input is empty, an empty ARRAy is returned**

**--parse\_json: convert json data to normal data**

**--directly insert json data into table: colon notation "accessing json data"**

**--type\_of function**

**--dealing with date and timestamp**

**--flattening JSON data into separate tables:**

**--create json table with variant column**

create or replace table json\_table(json\_col variant);

**--insert into variant col table**

insert into json\_table(json\_col) values ('{"firstName" : "Raviraj", "empid": 1}'); *--error accepting variant got varchar*

**-- insert records with parse\_json**

insert into json\_table(json\_col)

select parse\_json('{"firstName" : "Raviraj", "empid": 1}');

select \* from json\_table;

**--insert multiple values in column**

insert into json\_table(json\_col)

select parse\_json(Column1) from values

('{"firstName" : "Yogesh", "empid" : 1}'),

('{"firstName" : "Amogh", "empid" : 2}'),

('{"firstName" : "Jeet", "empid" : 3}'),

('{"firstName" : "Varun", "empid" : 4}'),

('{"firstName" : "Atharva", "empid" : 5}');

select \* from json\_table;

**--creating & inserting records in json table (employee\_table)**

**--directly inserting json data into table**

**--create an employee table with one variant column**

create or replace table employee\_table(

emp\_json variant);

**-- insert one single record**

insert into employee\_table(emp\_json)

select parse\_json(

'{

"name": "Yogesh",

"age": 22,

"height\_in\_ft": 5.11,

"married": true,

"has\_kids": false,

"stock\_options": null

}');

**-- run the select sql**

select \* from employee\_table;

**--colon notation to access JSON elements**

select

emp\_json:name,

emp\_json:age,

emp\_json:height\_in\_ft,

emp\_json:married,

emp\_json:has\_kids,

emp\_json:stock\_options

from employee\_table;

**--accessing data with alias notation**

select

emp\_json:age::integer as age,

emp\_json:name::string as Name,

emp\_json:height\_in\_ft::decimal as height,

emp\_json:married as is\_married,

emp\_json:has\_kids as has\_kids,

emp\_json:stock\_options as stock\_options

from employee\_table;

**--adding typeof() function to check how snowflake treat these name/value pair**

select

typeof(emp\_json:name) as "Name",

typeof(emp\_json:age) as age,

typeof(emp\_json:height\_in\_ft) as height,

typeof(emp\_json:married) as is\_married,

typeof(emp\_json:has\_kids) as has\_kids,

typeof(emp\_json:stock\_options) as stock\_options

from employee\_table;

**--inserting nested json records for accessing it and practice**

insert into employee\_table(emp\_json)

select parse\_json(

'{

"employee": {

"name":"Yogesh",

"age": 22,

"height\_in\_ft":5.11,

"married": true,

"has\_kids": false,

"stock\_options": null,

"phone": [

"69696969777",

"21212121217"

],

"Address": {

"street": "pan-card road",

"city":"Pune",

"state":"Maharashtra"

}

}

}'

);

select \* from employee\_table;

**--accessing the records in nested json data with (.) notation for nested one and (:) will also work**

select

emp\_json:employee.name::string as name,

emp\_json:employee.age as age,

emp\_json:employee.height\_in\_ft as height,

emp\_json:employee.married as is\_married,

emp\_json:employee.has\_kids as has\_kids,

emp\_json:employee.stock\_options as stock\_options,

typeof(emp\_json:employee.phone) as all\_phone\_type,

ARRAY\_SIZE(emp\_json:employee.phone) as how\_many\_phone,

emp\_json:employee.phone[0] as work\_phone,

emp\_json:employee.phone[1] as office\_phone,

typeof(emp\_json:employee:Address) as address\_type,

emp\_json:employee:Address:street as street,

emp\_json:employee.Address.city as city,

emp\_json:employee.Address.State as state

from employee\_table;

**--apply other function and mathematical operation without casting it**

select

emp\_json:employee.age as age,

(emp\_json:employee.height\_in\_ft) \* (12\*2.54) as height\_in\_cm

from employee\_table;

**--date and timestamp data types in snowflake JSON**

select current\_timestamp;

insert into employee\_table(emp\_json)

select parse\_json(

'{

"name" : "Jeet",

"age" : "22",

"height\_in\_ft" : 5.11,

"dob":"2000-05-01",

"dob\_timestamp": "2023-04-27 06:43:45.925 -0700",

"married": true,

"has\_kids": false,

"stock\_options": null

}'

);

select \* from employee\_table;

**--access timestamp**

select emp\_json:dob::date,

emp\_json:dob\_timestamp::timestamp

from employee\_table order by 1 desc;

**--Flatting JSON data and loading into snowflake tables**

**--create the table**

create or replace table employee\_table(

emp\_json variant);

select \* from employee\_table;

**--1st record**

insert into employee\_table(emp\_json)

select parse\_json(

'{

"employee": {"name": "Yogesh", "age": 22, "heighr\_in\_ft": 5.11,"married":true,"has\_kids":false,

"stock\_options": null, "email":"yogesh@gmail.com","phone": ["+91 1234567890","+91 0987654321"], "Address": {"street":"passport office", "city":"pune","State":"maharashtra"}

}

}'

);

**--2nd record**

insert into employee\_table(emp\_json)

select parse\_json(

'{

"employee": {"name": "Amogh", "age": 22, "heighr\_in\_ft": 6.00,"married":true,"has\_kids":false,

"stock\_options": null, "email":"amogh@gmail.com","phone": ["+91 1234561212","+91 0987652121"], "Address": {"street":"wadgoan sheri", "city":"pune","State":"maharashtra"}

}

}'

);

select \* from employee\_table;

**--creating sequence object for attaching t3 table (emp, phone, address) for primary and foreign key**

**--create sequence -for incremental number as primary key and foreign key**

create or replace sequence emp\_seq

start 1

increment 1

comment = 'employee sequence';

create or replace sequence phone\_seq

start 1

increment 1

comment = 'phone sequence';

create or replace sequence address\_seq

start 1

increment 1

comment = 'address sequence';

**--create master table to load json data --in column format**

**--employee master table**

**--using seq objects as an auto generated number (nextval)**

create or replace table employee\_master(

emp\_pk integer default emp\_seq.nextval,

name string,

age number(3),

height\_in\_cm decimal(6,3),

is\_married boolean,

has\_kids boolean,

stock\_options integer,

email varchar(100)

);

**-- child table holding all the phones**

**--having relationship with emp master table**

create or replace table emp\_phones(

phone\_pk integer default phone\_seq.nextval,

emp\_fk number,

phone\_type varchar(20),

phone\_number varchar(30)

);

**-- child table holding all the address**

**--having relationship with emp master table**

create or replace table emp\_address(

address\_pk integer default address\_seq.nextval,

emp\_fk number,

street\_address varchar(200),

city varchar(50),

state varchar(50)

);

**DATA SAMPLING**

**--a. Bernoulli(or row):** includes each row with a probability of p/100,

--similar to flipping a weighted coin for each row.

**--b. SYSTEM (or Block):** includes each block of rows with a probability of p/100.

-- This method does not support fixed-size sampling.

**--c. Repeatable | SEED (seed):** For identifying the data

--Specifies a seed value to make the sampling deterministic. Can be any integer

--The following keywords can be interchangeably:

**SAMPLE | TABLESAMPLE**

**BERNOULLI | ROW**

**SYSTEM | BLOCK**

**REPEATABLE | SEED**

--The number of rows returned depends on the sampling method specified:

**--1. Fraction based**

-- if no seed is specified, SAMPLE generates different results when the same query is repeated

--if a table does not change, and the same seed and probability are specified, SAMPLE generates the same result.

--sampling on a copy of a table might not return the same result as sampling on the original table, even if the same

--2 methods with example

**--ROW:**: return a sample of a table in which each row has a 20.3% probability of being included in the sample:

select \* from test tablesample bernoulli (20.3);

**--BLOCK**:: return a sample of a table in which each block of row has a 3% probability of being included in the sample, and set

select \* from testtable sample system (3) seed (82);

**--2. Fixed size**

--if the table is larger than the requested number of rows, the number of requested rows is always returned.

--if the table is smaller than the requested number of rows, the entire table is returned.

**--example:**

select \* from example\_table sample system (10 rows);

select \* from example\_table sample row(10 rows) seed(99);

**--sampling with a seed is not supported on view or subqueries**

select \* from table1 sample system (10 rows);

select \* from table1 sample row (10 rows) seed (99);

**--some example**

1. select \* from table\_sample row(10); --return a sample with 10% of rows

2. select \* from tablename tablesample block(20) --return a sample with data from 20% block

3. select \* from tablename sample system (10) seed (111);

--return a sample with data from 10% of blocks and same data set if use seed11 next time

4. select \* from tablename tablesample (100) --return an entire table, including all rows into the sample

5. select \* from tablename tablesample row (0); -- return an empty sample

6. select \* from tablename sample (10 rows); --return a fixed size sample of 10 rows

**--performing data sample on the data**

**--1.On Fixed data**

select \* from customer\_csv sample row(10); --generate approx data with sampling of 10%

select \* from customer\_csv sample block (10);

**--just selecting data from sample db**

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER;

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER sample bernoulli (10);

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER sample system (10);

**--now we have sample some data now lets store that data**

**--with row or bernoulli method**

**create or replace table customer\_1 as**

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER sample (10);

select count(1) from customer\_1;

**create or replace table customer\_2 as**

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER sample row (10);

select count(1) from customer\_2;

select count(1) from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER;

**--with system or block**

**create or replace table customer\_3 as**

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER sample block (10);

select count(1) from customer\_3;

**create or replace table customer\_4 as**

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER sample block (10) seed(5);

select count(1) from customer\_4;

**--2.On Fixed sampling**

create or replace table customer\_5 as

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER sample row (1000 rows) ;

select count(1) from customer\_5;

**--check the seed is same or not**

select \* from customer\_3

minus

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER sample block (10);

select count(1) from customer\_3;

select \*from customer\_4

minus

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10.CUSTOMER sample block (10) seed(5);

select count(1) from customer\_4;

**--User defined function**

**--allows you to perform operations that are not available through the built-in, system defined function.**

**--SF UDF's can return scalar(just a value or a string) and tabular results.**

**--2 types of UDF available in SF**

**--1. SCALAR 2. TABULAR**

**--SCALAR: returns output for each input we are passing (varchar or any string or value).**

**--TABULAR: returns a table (can return zero, one or multiple rows)**

**--SF supports 4 languages for writing UDFs**

**--SQL**

**--JavaScript**

**--Java**

**--Python**

**--Snowflake supports UDF overloading: means support functions with the same name but different params.**

**--– Proc\_Calculate\_Area() is different from Proc\_Calculate\_Area(Radius FLOAT)**

**--– Proc\_Calculate\_Area (Radius FLOAT) is different from**

**--– Proc\_Calculate\_Area (Length number, width number)**

**--SCALAR UDF**

create or replace schema udf\_proc;

**--create a simple UDF**

create or replace function simple\_sql\_udf()

returns string

language SQL

as

$$

select 'simple sql language udf'

$$ ;

select simple\_sql\_udf();

**--function for cal area of circle**

create or replace function area\_of\_circle(radius float)

returns float

as

$$

pi()\* radius \* radius

$$ ;

select area\_of\_circle(5.5);

**--function for cal multi**

create or replace function cal\_multi(a float, b float)

returns float

as

$$

case

when a is null then 0

else

a\*b

end

$$ ;

select cal\_multi(5,2);

select cal\_multi(5,null);

**--scenario 1- fixed tax of 10%**

create or replace function cust\_tax(price float)

returns float

as

$$

(price \* 10)/100

$$ ;

**--grant the access on scenario1**

grant usage on function cust\_tax(float) to public;

**--listing records and lets the tax**

select O\_ORDERKEY, O\_ORDERDATE, O\_TOTALPRICE, CUST\_TAX(O\_TOTALPRICE) FROM snowflake\_sample\_data.tpch\_sf1.orders;

**--scenario 2 -variable tax**

create or replace function cust\_tax(price float, tax\_per float)

returns float

as

$$

(price \* tax\_per)/100

$$ ;

**--grant the access on scenario1**

grant usage on function cust\_tax(float, float) to public;

**--listing records and lets the tax**

select O\_ORDERKEY, O\_ORDERDATE, O\_TOTALPRICE, CUST\_TAX(O\_TOTALPRICE, 8) FROM snowflake\_sample\_data.tpch\_sf1.orders;

**--TABULAR UDF**

create or replace function tab()

returns table (msg varchar)

as

$$

select 'Raj'

union

select 'Raviraj'

$$;

select msg from table (tab());

Create or replace table employee(

employee varchar (30),

dept\_name varchar (20),

salary varchar (20),

manager varchar (20)

);

insert into employee values ('Monika', 'Associate', '1lac', 'Vishal');

insert into employee values ('Vishal', 'Analyst', '1.2lac', 'Niharika');

insert into employee values ('Niharika', 'Manager', '1.4lac', 'Vaibhav');

insert into employee values ('Vaibhav', 'CEO', '1.9lac', 'Vaibhav');

select \* from employee;

**--function for like keyword**

create or replace function emp\_fun()

returns table(employee varchar, dept\_name varchar, salary varchar, manager varchar)

as

$$

select \* from employee where employee like '%v'

$$;

select \* from table (emp\_fun());

**--creating a simple orders table**

create or replace table orders(

order\_id number,

customer\_id\_fk number,

item\_id\_fk number,

retail\_price number(10,2),

purchase\_price number(10,2),

sold\_quantity number(3),

country\_code varchar(2)

);

**--inserting records**

insert into orders values

(1,1,1,99.2,89.6,2,'US'),

(2,8,2,17.1,11,10,'IN'),

(3,5,1,827,900.99,5,'JP'),

(4,10,4,200,172,7,'DE');

select \* from orders;

**--function for cal profit**

create or replace function calculate\_profit(retail\_price number, purchase\_price number, sold\_quantity number)

returns number(10,2)

as

$$

select ((retail\_price - purchase\_price) \* sold\_quantity)

$$;

**--call function by scalar**

select calculate\_profit(100, 90, 6);

**--call function by table**

select item\_id\_fk, retail\_price, purchase\_price, sold\_quantity, calculate\_profit(retail\_price, purchase\_price, sold\_quantity)

as profit\_udf from orders;

**--call profit without function and observing same profit for both**

select item\_id\_fk, retail\_price, purchase\_price, sold\_quantity,

((retail\_price - purchase\_price) \* sold\_quantity) as profit from orders;

**--creating some more table**

create or replace table countries

(country\_code char(2), country\_name varchar);

insert into countries(country\_code, country\_name) VALUES

('FR', 'FRANCE'),

('US', 'UNITED STATES'),

('IN', 'INDIA'),

('SP', 'SPAIN');

**create or replace table user\_addresses**

(user\_id integer, country\_code char(2));

insert into user\_addresses (user\_id, country\_code) values

(100, 'SP'),

(123, 'FR'),

(567, 'US'),

(426, 'IN');

**--create a function to fetch country name of user\_id**

create or replace function get\_country\_for\_user(ID number)

returns table (user\_id number, country\_name varchar)

as

$$

select id, C.country\_name from user\_addresses A, countries C

where A.user\_id = ID

and C.country\_code = A.country\_code

$$;

**--fetch country name for specified user id**

select \* from table (get\_country\_for\_user(123));

**--fetch country name for all user id**

select F. \* from user\_addresses, table (get\_country\_for\_user(user\_id)) F;

--select \* from user\_addresses, table (get\_country\_for\_user(user\_id));

**--desc functions**

show functions like 'get\_country\_for\_user%';

**--desc function**

desc function get\_country\_for\_user(number);

**//table function example 1**

CREATE OR REPLACE table dates\_of\_interest (event\_date DATE);

INSERT INTO dates\_of\_interest (event\_date) VALUES

('2021-06-21'::DATE),

('2022-06-21'::DATE);

select \* from dates\_of\_interest ;

CREATE OR REPLACE FUNCTION record\_high\_temperatures\_for\_date(d DATE)

RETURNS TABLE (event\_date DATE, city VARCHAR, temperature NUMBER)

as

$$

SELECT d, 'New York', 65.0

UNION ALL

SELECT d, 'Los Angeles', 69.0

$$;

SELECT

doi.event\_date as "Date",

record\_temperatures.city,

record\_temperatures.temperature

FROM dates\_of\_interest AS doi,

TABLE(record\_high\_temperatures\_for\_date(doi.event\_date)) AS record\_temperatures

ORDER BY doi.event\_date, city;

**//table example2**

CREATE OR REPLACE table genders (genderType varchar);

INSERT INTO genders (genderType) VALUES

('male'::VARCHAR),

('female'::VARCHAR);

select \* from genders ;

CREATE OR REPLACE FUNCTION ubertaxifares(g varchar)

RETURNS TABLE (genderType varchar, city VARCHAR, fares NUMBER)

as

$$

SELECT g, 'New York', 32.0

UNION ALL

SELECT g, 'Los Angeles', 14.0

$$;

SELECT

gend.genderType as "Gender",

ubertaxirates.city,

ubertaxirates.fares

FROM genders AS gend,

TABLE(ubertaxifares(gend.genderType)) AS ubertaxirates

ORDER BY gend.genderType, city;

**//table function example3**

create or replace table countries

(country\_code char(2), country\_name varchar);

insert into countries(country\_code, country\_name) VALUES

('FR', 'FRANCE'),

('US', 'UNITED STATES'),

('IN', 'INDIA'),

('SP', 'SPAIN');

create or replace table user\_addresses

(user\_id integer, country\_code char(2));

insert into user\_addresses (user\_id, country\_code) values

(100, 'SP'),

(123, 'FR'),

(567, 'US'),

(426, 'IN');

**--create a function to fetch country name of user\_id**

create or replace function get\_country\_for\_user(ID number)

returns table (user\_id number, country\_name varchar)

as

$$

select id, C.country\_name from user\_addresses A, countries C

where A.user\_id = ID

and C.country\_code = A.country\_code

$$;

--fecth country name for specified user id

select \* from table (get\_country\_for\_user(123));

**--fecth country name for all user id**

select F. \* from user\_addresses, table (get\_country\_for\_user(user\_id)) F;

--select \* from user\_addresses, table (get\_country\_for\_user(user\_id));

**--desc functions**

show functions like 'get\_country\_for\_user%';

**--desc function**

desc function get\_country\_for\_user(number);

**//table function example4**

CREATE OR REPLACE table lottery(statuss varchar);

INSERT INTO lottery(statuss) VALUES

('couponswon'::varchar),

('foodcardswon'::varchar);

select \* from lottery;

CREATE OR REPLACE FUNCTION lotterycoupons(c varchar)

RETURNS TABLE (statuss varchar, couponswon varchar, foodcardswon varchar)

as

$$

SELECT c, 'AmazonCoupons', 'First'

UNION ALL

SELECT c, 'SwiggyCashCards', 'Second'

$$;

select lott.statuss as Statuss,

lotterycoupons.couponswon,

lotterycoupons.foodcardswon

from lottery as lott,

table(lotterycoupons(lott.statuss)) as lotterycoupons

order by foodcardswon;

**//example5**

create or replace function tab()

returns table (msg varchar(50))

as

$$

select 'Raj'

union

select 'Raviraj'

$$;

select msg from table (tab());

**--STREAMS**

**--create schema for schemas**

create or replace schema streams;

**--table for stream**

create or replace table employee

(empid number,

empname varchar,

salary number,

age number,

department varchar,

location varchar);

**--inserting records**

insert into employee values

(1,'Amogh',700001, 22, 'AWS', 'Hubli'),

(2,'Yogesh',700002, 23, 'RPA', 'Latur'),

(3,'Atharva',700003, 22, 'GAD', 'Pune'),

(4,'Varun',700004, 21, 'React', 'Patna'),

(5,'Abeer',700005, 23, 'React', 'Jodpur'),

(6,'Zuzu',700006, 23, 'MS', 'Latur'),

(7,'Shweta',700007, 24, 'AWS', 'Satara');

select \* from employee;

**--create a stream on employee table**

create or replace stream employee\_stream on table employee;

select \* from employee\_stream;

**--INSERT some records to employee table**

insert into employee values

(8,'Aniket',700001, 22, 'SQL', 'PCMC'),

(9,'Gaurav',700002, 22, 'Python', 'Prayagraj');

select \* from employee;

select \* from employee\_stream;

**--DELETE some records**

DELETE FROM employee

WHERE empid = 7;

delete from employee where empid = 1;

truncate table employee;

select \* from employee;

select \* from employee\_stream;

**--UPDATE some records**

update employee set salary = 49000 where empid =1;

update employee set location = 'pune' where empid =7;

**--listing stream**

drop stream employee\_stream;

create or replace stream employee\_stream on table employee;

show streams in schema streams;

desc stream employee\_stream;

**--MULTIPLE STREAMS ON SINGLE TABLE**

--we can create multiple streams on single table,

create or replace stream employee\_stream2 on table employee;

show streams;

**--INSERT some records to employee tabel**

insert into employee values

(8,'Aniket',700001, 22, 'SQL', 'PCMC'),

(9,'Gaurav',700002, 22, 'Python', 'Prayagraj');

select \* from employee\_stream;

select \* from employee\_stream2;

**--APPEND ONLY STREAM**: just capture the data which is being inserted,

--update or deleting will not be updated

**--TABLE FOR STREAM**

CREATE OR REPLACE TABLE employee\_append

(

empid number,

empname varchar,

salary number,

age number,

department varchar,

location varchar

);

**--inserting records to source table**

insert into employee\_append values

(1,'Amogh',700001, 22, 'AWS', 'Hubli'),

(2,'Yogesh',700002, 23, 'RPA', 'Latur'),

(3,'Atharva',700003, 22, 'GAD', 'Pune'),

(4,'Varun',700004, 21, 'React', 'Patna'),

(5,'Abeer',700005, 23, 'React', 'Jodpur'),

(6,'Zuzu',700006, 23, 'MS', 'Latur'),

(7,'Shweta',700007, 24, 'AWS', 'Satara');

**--create a stream on emp\_source**

create or replace stream emp\_append\_stream on table employee\_append

append\_only = true;

**--insert some records**

insert into employee\_append values

(8,'Aniket',700001, 22, 'SQL', 'PCMC'),

(9,'Gaurav',700002, 22, 'Python', 'Prayagraj');

select \* from employee\_stream;

select \* from employee\_stream2;

select \* from employee\_append;

**--update some records**

update employee\_append set salary = 700002 where empid =2;

update employee\_append set location = 'pune' where empid =7;

**--delete some records**

delete from employee\_append where empid = 1;

show streams;

**-- Consuming data from streams**

CREATE OR REPLACE TABLE emp\_source

(

empid number,

empname varchar,

salary number,

age number,

department varchar,

location varchar

);

--inserting records to source table

insert into emp\_source values

(1,'Amogh',700001, 22, 'AWS', 'Hubli'),

(2,'Yogesh',700002, 23, 'RPA', 'Latur'),

(3,'Atharva',700003, 22, 'GAD', 'Pune'),

(4,'Varun',700004, 21, 'React', 'Patna'),

(5,'Abeer',700005, 23, 'React', 'Jodpur'),

(6,'Zuzu',700006, 23, 'MS', 'Latur'),

(7,'Shweta',700007, 24, 'AWS', 'Satara');

**--create or replace table -- whatever changes is captured by stream will store into target table**

create or replace table emp\_target

(

empid int,

empname varchar (30),

salary float,

age int,

department varchar(20),

location varchar(20)

);

select \* from emp\_source;

select \* from emp\_target;

**--create a stream**

create or replace stream emp\_stream on table emp\_source;

**--insert some records**

insert into emp\_source values

(8,'Aniket',700001, 22, 'SQL', 'PCMC'),

(9,'Gaurav',700002, 22, 'Python', 'Prayagraj');

select \* from emp\_stream;

**begin transaction;**

**--updating (storing) record captured by stream into target table**

insert into emp\_target

select empid, empname, salary, age, department, location from emp\_stream

where METADATA$ACTION = 'INSERT' AND METADATA$ISUPDATE = False;

**commit;**

select \* from emp\_target;

**--Streams on External Tables**

**--create a file format**

create or replace file format csv\_ff

type = csv

field\_delimiter = ','

skip\_header = 1

empty\_field\_as\_null = TRUE;

**--create storage integration object**

create or replace storage integration aws\_ss

TYPE = EXTERNAL\_STAGE

STORAGE\_PROVIDER = S3

ENABLES = TRUE

STORAGE\_AWS\_ROLE\_ARN = 'arn:aws:iam::973573511884:role/snowpractice21'

STORAGE\_ALLOWED\_LOCATIONS = ('s3://snowpractice21/csv files/', 's3://snowpractice21/others/')

COMMENT = 'INTEGRATION WITH S3 BUCKETS';

**--create stage object with integration object and file format**

create or replace stage ext\_stage

url = 's3://snowpractice21/csv files/'

storage\_integration = aws\_ss

file\_format = csv\_ff;

**--list stage**

list @ext\_stage;

**--create table --user email**

create or replace external table user\_external (

id number as(value:c1::number),

first\_name varchar as (value:c2::varchar),

last\_name varchar as (value:c3::varchar),

email varchar as (value:c4::varchar),

gender varchar as (value:c5::varchar),

aboutme varchar as (value:c6::varchar)

)

with

location = @ext\_stage

pattern = '.\*user.\*'

file\_format = csv\_ff ;

select \* from user\_external;

**--creating stream on external table**

**--we cannot create standard stream on external table**

**--only insert\_only work on external table**

create or replace stream exxt\_stream on table external table

insert\_only = true; --it will fail

select \* from extt\_stream;

**--now whenever new data files are arriving on s3 location, stream will capture them.**

**--retention period = 0 for all table**

**--TASKS**

**--1. SIMPLE TASK WITH TIME**

create or replace schema tas\_ks;

use schema tas\_ks;

**--create a sample table for inserting data using tasks**

create or replace table customer

(

id int autoincrement start = 1 increment =1,

name varchar(20),

location varchar(20),

load\_time timestamp

);

**--create task to insert data for every minute to customer table**

create or replace task customer\_task

warehouse = employees

schedule = '1 minute'

as

insert into customer (name, location, load\_time)

values('amogh', 'hubli', current\_timestamp);

**--listing records**

select \* from customer;

**--to see tasks**

show tasks;

**--describing the tasks**

desc task customer\_task;

**--we have to resume the task so that our task should run and do their work**

**--starting and suspending tasks**

alter task customer\_task resume; --to start a task

alter task customer\_task suspend; --to stop task

**--lets see if data is inserted or not after 1 mintue**

select \* from customer; --running

desc task customer\_task;

**--2.USING CORN METHOD**

**--using corn to load the data for every minute**

select current\_timestamp;

create or replace task customer\_cron

warehouse = employees

schedule = 'using cron \* \* \* \* \* utc' --0-6 0=sun, 1=mon --- 6=sat, L=last week

as

insert into customer (name, location, load\_time)

values ('yogesh','Latur', current\_timestamp);

**--describing the tasks**

**--listing records**

select \* from customer;

desc task customer\_cron;

**--resume and suspend a task**

alter task customer\_cron resume;

alter task customer\_cron suspend;

**--some more example for cron**

**--every day at 7 am utc timezone --7:00 am**

schedule = 'using cron 0 7 \* \* \* utc'

**--every day at 10 am and 10 pm --10:00 am , 22:00 pm**

schedule = 'using cron 0 10,22 \* \* \* utc'

**--every month last day at 11pm**

schedule = 'using cron 0 23 L \* \* utc'

**--every monday at 6:30 am**

schedule = 'using cron 30 6 \* \* 1 utc'

**--LET CLONE THE TASK AND IT WORKS**

create or replace task customer\_copy CLONE customer\_cron;

desc task customer\_copy;

**--DAG OF TASKS**

/\*

ROOT TABLE --> EMP\_ADMIN

CHILD TABLES --> SALES TABLE, HR TABLE, MARKETING TABLE

first we will create task on emp\_admin root table then child tables tasks will be automatically executed in order

we will get sales dept data in sales dept

we will get hr dept in hr dept

we will get market dept data in market dept

\*/

**--create a root table**

create or replace table emp\_admin

(

empid int autoincrement start = 1 increment = 1,

emp\_name varchar(20),

dept\_name varchar(44),

load\_time timestamp

);

select \* from emp\_admin;

--root task (DAG task)

create or replace task emp\_admin

warehouse = employees

schedule = '1 minute'

as

insert into emp\_admin(emp\_name, dept\_name,load\_time)

values ('Atharva', 'sales', current\_timestamp);

desc task emp\_admin;

**--child task we will create - sales, hr, market /// / root task --> emp\_admin**

**--child task for loading sales data into sales table**

create or replace task sales\_task

warehouse = employees

after emp\_admin

as

create or replace table sales\_tble

as

select \* from emp\_admin

where dept\_name = 'sales';

desc task sales\_task;

**--child task for loading hr data into hr table**

create or replace task hr\_task

after emp\_admin

as

create or replace table hr\_tble

as

select \* from emp\_admin

where dept\_name = 'hr';

desc task hr\_task;

**--child task for loading marketing data into marketing table**

create or replace task market\_task

as

create or replace table market\_tble

as

select \* from emp\_admin

where dept\_name = 'market';

desc task market\_task;

**--add dependencies**

alter task market\_task add after sales\_task;

alter task market\_task add after hr\_task;

**--start the tasks: 1st child task 2 parent task**

alter task market\_task resume;

alter task hr\_task resume;

alter task sales\_task resume;

alter task emp\_admin resume;

alter task market\_task suspend;

alter task hr\_task suspend;

alter task sales\_task suspend;

alter task emp\_admin suspend;

show tasks;

select \* from emp\_admin;

select \* from sales\_tble;

select \* from hr\_tble;

select \* from market\_tble;

**--altering the task for hr table**

alter task emp\_admin suspend;

alter task emp\_admin

modify

as

insert into emp\_admin(emp\_name, dept\_name,load\_time)

values ('Amogh', 'hr', current\_timestamp);

alter task emp\_admin resume;

select \* from emp\_admin;

**--altering the task for market table**

alter task emp\_admin suspend;

alter task emp\_admin

modify

as

insert into emp\_admin(emp\_name, dept\_name,load\_time)

values ('Yogesh', 'market', current\_timestamp);

alter task emp\_admin resume;

select \* from emp\_admin;

**--TASK HISTORY**

**--to see all tasks history with last executed task first**

select \* from table (information\_schema.task\_history()) order by scheduled\_time desc;

**--TASK ON STORED PROCEDURES**

--employee table for task

create or replace table employee (emp\_id number, emp\_name varchar, emp\_address varchar);

**--sequence**

create sequence if not exists emp\_id;

**--create a procedure for inserting records on table**

create or replace procedure employee\_insert(name varchar, address varchar)

returns varchar

language javascript

as

$$

var command = "INSERT INTO Employee (emp\_id, emp\_name, emp\_address) values (emp\_id.nextval, '"+NAME+"', '"+ADDRESS+"')";

var cmd1\_dict = {sqlText: command};

var stmt = snowflake.createStatement(cmd1\_dict);

var rs = stmt.execute();

return 'success';

$$;

**--call the procedure**

call employee\_insert ('Yogesh', 'Latur');

SELECT \* from employee;

**--create task and calling a stored procedure for insertion of records**

create or replace task emp\_task

warehouse = employees

schedule = '1 minute'

as

call employee\_insert('Amogh', 'Hugli');

alter task emp\_task resume;

alter task emp\_task suspend;

SELECT \* from employee;

**TASK + STREAM**

create or replace database store;

create or replace schema mystreams; --store all streams

create or replace schema stage\_tbls; --store all source tables

create or replace schema intg\_tbls; --store all the target table

**--stage table (source table)**

create or replace table stage\_tbls.stg\_empl

(

EMPID INT,

EMPNAME VARCHAR(30),

SALARY FLOAT,

AGE INT,

DEPT VARCHAR(10),

LOCATION VARCHAR(20)

);

--stream on source table

CREATE OR REPLACE STREAM MYSTREAMS.STREAM\_EMPL ON TABLE STAGE\_TBLS.STG\_EMPL;

show streams in schema mystreams;

select \* from mystreams.stream\_empl;

**--Inserts**

insert into stage\_tbls.stg\_empl values

(1,'Amogh',700001, 22, 'AWS', 'Hubli'),

(2,'Yogesh',700002, 23, 'RPA', 'Latur'),

(3,'Atharva',700003, 22, 'GAD', 'Pune'),

(4,'Varun',700004, 21, 'React', 'Patna'),

(5,'Abeer',700005, 23, 'React', 'Jodpur'),

(6,'Zuzu',700006, 23, 'MS', 'Latur'),

(7,'Shweta',700007, 24, 'AWS', 'Satara');

select \* from stage\_tbls.stg\_empl;

**--listing capture by stream**

select \* from mystreams.stream\_empl;

**--target table**

create or replace table intg\_tbls.empl

(

empid int,

empname varchar(30),

salary float,

age int,

dept varchar(15),

location varchar(20),

insert\_dt date,

lst\_updt\_dt date

);

select \* from intg\_tbls.empl;

**--sending (updating) new records (insertion) into target table**

insert INTO intg\_tbls.empl

(empid, empname, salary, age, dept, location, insert\_dt, lst\_updt\_dt)

select empid, empname, salary, age, dept, location, current\_date, null from mystreams.stream\_empl

where METADATA$ACTION = 'INSERT' AND METADATA$ISUPDATE = FALSE;

select \* from intg\_tbls.empl;

**--UPDATES**

**--doing some of the updation**

update stage\_tbls.stg\_empl set salary=700001 where empid=2;

update stage\_tbls.stg\_empl set location= 'Pune' where empid=5;

select \* from stage\_tbls.stg\_empl;

**--listing captured by the stream**

select \* from mystreams.stream\_empl;

**--sending (updating) the updated data into the target which is captured by the the stream**

MERGE INTO intg\_tbls.empl E

USING MYSTREAMS.STREAM\_EMPL S

ON E.EMPID = S.EMPID

WHEN MATCHED

AND S.METADATA$ACTION = 'INSERT' AND S.METADATA$ISUPDATE = TRUE

THEN UPDATE

SET E.EMPNAME = S.EMPNAME,

E.SALARY = S.SALARY,

E.AGE = S.AGE,

E.DEPT = S.DEPT,

E.LOCATION = S.LOCATION,

E.LST\_UPDT\_DT = CURRENT\_DATE;

select \* from intg\_tbls.empl;

**--DELETE**

select \* from stage\_tbls.stg\_empl;

**--deleteing some records**

delete from stage\_tbls.stg\_empl where empid in (3,4);

**-- sending (deleted) the deleted data into the target table which is captured by the stream**

MERGE INTO intg\_tbls.empl E

USING MYSTREAMS.STREAM\_EMPL S

ON E.EMPID = S.EMPID

WHEN MATCHED

AND S.METADATA$ACTION = 'DELETE' AND S.METADATA$ISUPDATE = FALSE

THEN DELETE;

--listing captured by the stream

select \* from mystreams.stream\_empl;

select \* from intg\_tbls.empl;

**--ALL CHANGES AT A SAME TIME**

**--(updation, deletion & insertion)**

select \* from stage\_tbls.stg\_empl;

**--insert**

insert into stage\_tbls.stg\_empl values

(3,'Atharva',700003, 22, 'GAD', 'Pune'),

(4,'Varun',700004, 21, 'React', 'Patna');

**--updates**

update stage\_tbls.stg\_empl

set salary = 700002, location ='Pune' where empid=1;

**--delete**

delete from stage\_tbls.stg\_empl where empid in (6);

**--listing captured by the stream**

select \* from mystreams.stream\_empl;

**--sending all insert updates delete to target table**

MERGE INTO intg\_tbls.empl E

USING MYSTREAMS.STREAM\_EMPL S

ON E.EMPID = S.EMPID

WHEN MATCHED

AND S.METADATA$ACTION = 'DELETE' AND S.METADATA$ISUPDATE = FALSE

THEN DELETE

WHEN MATCHED

AND S.METADATA$ACTION = 'INSERT' AND S.METADATA$ISUPDATE = TRUE

THEN UPDATE

SET E.EMPNAME = S.EMPNAME,

E.SALARY = S.SALARY,

E.AGE = S.AGE,

E.DEPT = S.DEPT,

E.LOCATION = S.LOCATION,

E.LST\_UPDT\_DT = CURRENT\_DATE

WHEN NOT MATCHED

AND METADATA$ACTION = 'INSERT' AND METADATA$ISUPDATE = FALSE

THEN INSERT (empid, empname, salary, age, dept, location, insert\_dt, lst\_updt\_dt)

VALUES (S.EMPID, S.EMPNAME, S.SALARY, S.AGE, S.DEPT, S.LOCATION, CURRENT\_DATE, NULL);

select \* from intg\_tbls.empl;

select \* from stage\_tbls.stg\_empl;

**--DOING WITH TASKS**

**--with tasks**

**create or replace schema mytasks;**

**--creating tasks**

create or replace task mytasks.task\_empl\_data\_load

warehouse = EMPLOYEES

schedule = '1 minute'

when system$stream\_has\_data('mystreams.stream\_empl')

AS

MERGE INTO intg\_tbls.empl E

USING MYSTREAMS.STREAM\_EMPL S

ON E.EMPID = S.EMPID

WHEN MATCHED

AND S.METADATA$ACTION = 'DELETE' AND S.METADATA$ISUPDATE = FALSE

THEN DELETE

WHEN MATCHED

AND S.METADATA$ACTION = 'INSERT' AND S.METADATA$ISUPDATE = TRUE

THEN UPDATE

SET E.EMPNAME = S.EMPNAME,

E.SALARY = S.SALARY,

E.AGE = S.AGE,

E.DEPT = S.DEPT,

E.LOCATION = S.LOCATION,

E.LST\_UPDT\_DT = CURRENT\_DATE

WHEN NOT MATCHED

AND METADATA$ACTION = 'INSERT' AND METADATA$ISUPDATE = FALSE

THEN INSERT (empid, empname, salary, age, dept, location, insert\_dt, lst\_updt\_dt)

VALUES (S.EMPID, S.EMPNAME, S.SALARY, S.AGE, S.DEPT, S.LOCATION, CURRENT\_DATE, NULL);

desc task mytasks.task\_empl\_data\_load;

**--resume and suspend the task**

alter task mytasks.task\_empl\_data\_load resume;

alter task mytasks.task\_empl\_data\_load suspend;

select \* from stage\_tbls.stg\_empl;

**--doing some insert, update, delete**

**--insert**

insert into stage\_tbls.stg\_empl values

(6,'Zuveriya',700003, 22, 'Sharepoint', 'Latur'),

(8,'Aniket',700004, 22, 'SQL', 'PCMC');

insert into stage\_tbls.stg\_empl values

(10,'Gaurav',700010, 22, 'Python', 'Prayagraj');

**--updates**

update stage\_tbls.stg\_empl

set salary = 700007, location ='Laturr' where empid=2;

**--delete**

delete from stage\_tbls.stg\_empl where empid in (3);

select \* from intg\_tbls.empl;

select \* from stage\_tbls.stg\_empl;

--listing captured by the stream

select \* from mystreams.stream\_empl;

**--STORED PROCEDURAL LANGUAGE**

**1. simple hello world procedure**

**2. simple example of using arguments passed to a stored procedure**

**3. using an argument in a SQL statement (Binding) --we use [:] for argument**

**4. Returning tabular data**

**5. calling a stored procedure from another stored procedure**

**6. some more examples**

**--1. simple hello world procedure**

**--creating procedure for hello world**

create or replace procedure output\_message (message varchar)

returns varchar not null

language SQL

as

$$

begin

return message;

end;

$$

;

**--call the stored procedure**

call output\_message ('Hello Raviraj, its start stored procedure');

**--2. simple example of using arguments passed to a stored procedure**

**--procedure for returning a greater values between 2 values**

create or replace procedure return\_greater(num\_1 integer, num\_2 integer)

returns integer not null

language SQL

as

$$

begin

if(num\_1>num\_2) then

return num\_1;

else

return num\_2;

end if;

end;

$$;

**--calling**

call return\_greater(10,40);

**--3. using an argument in a SQL statement (Binding) --we use [:] for argument**

create or replace table invoices (id integer, price number);

insert into invoices values (1, 1000);

insert into invoices values (2, 2000);

insert into invoices values (3, 3000);

insert into invoices values (4, 4000);

insert into invoices values (5, 5000);

select \* from invoices;

**--stored procedure for returning the data by passing the id**

create or replace procedure find\_invoice\_by\_id (id varchar)

returns table (id integer, price number(12,2))

language sql

as

$$

DECLARE

res resultset DEFAULT (select \* from invoices where id = :id);

BEGIN

return table (res);

END;

$$;

**--calling**

call find\_invoice\_by\_id('3');

**--5. calling a stored procedure from another stored procedure**

**--create a table for use in the example**

create or replace table int\_table (value integer);

**--stored procedures to insert the values to the table**

**--create first stored procedure to be called from another stored procedure**

create or replace procedure insert\_value(value integer)

returns varchar not null

language sql

as

$$

begin

insert into int\_table values(:value);

return 'rows inserted:' || SQLROWCOUNT;

END;

$$;

**--create second stored procedure to be called from another stored procedure**

create or replace procedure insert\_two\_values(value1 integer, value2 integer)

returns varchar not null

language SQL

as

$$

begin

call insert\_value(:value1);

call insert\_value(:value2);

return 'finished calling stored procedures';

END;

$$;

call insert\_two\_values(20,21);

select \* from int\_table;

--sequence

create or replace sequence seq1;

select seqref.a a, seqref.a b from (select seq1.nextval a from dual) seqref;

create or replace table abc(col1 varchar);

insert into abc values ('a'),('b'),('c'),('d'),('e'),('f'),('g'),('h'),('i'),('j');

alter table abc add column col2 number;

create or replace sequence seq1;

update abc set col2 = seq1.nextval;

select \* from abc;

--data masking

create or replace masking policy date\_mask\_idm as (val date)

return date -> case when current\_role() in ('admin')

then val

else date\_trunc('year', val)END;

select date\_trunc('year', current\_date);

select current\_timestamp;

select date\_trunc('year', current\_timestamp);

select date\_trunc('month', current\_timestamp);

select date\_trunc('day', current\_timestamp);

--how to retrieve undrop table data after rename

create or replace table abcd (col1 int, col2 varchar(5));

insert into abcd values(1,'abc'), (2,'xyz');

drop table abcd;

create or replace table abcd (col1 int, col2 varchar(5), col3 date);

insert into abcd values(1,'abc', current\_date);

select \* from abcd;

alter table abcd rename to abcd\_2;

undrop table abcd;

alter table abcd rename to abcd\_day1;

alter table abcd\_2 rename to abcd;

select \* from abcd;

select \* from abcd\_day1;

**-- query for listagg**

select FULL\_NAME, LISTAGG (LANG,',') as LANG\_SPEAK

from EMP\_lang

group by FULL\_NAME;

create or replace table tab(id int,val varchar(10));

insert into tab values (1, 'a'), (1, 'b'), (1, 'c'), (2, 'x'), (2, 'y'), (3, 'a'), (3, 'b');

select id, LISTAGG(val,'|')AS "values"

from tab group by id;

**–Scripting**

show databases;

create or replace database snowscript;

show schemas;

create or replace schema scripts;

**-- this is format for classic console**

execute immediate

$$

declare

result varchar2 default 'success';

begin

create or replace table emp(empno number, ename varchar2(30));

create or replace table dept(deptno number, dname varchar2(30));

insert into emp values(123, 'Rj'), (456,'Aj');

insert into dept values (21, 'It'), (11, 'Hr');

return result;

end;

$$

**-- now without using $$**

declare

result varchar2 default 'success';

begin

create table if not exists emp(empno number, ename varchar2(30));

create table if not exists dept(deptno number, dname varchar2(30));

insert into emp values(123, 'Rj'), (456,'Aj');

insert into dept values (21, 'It'), (11, 'Hr');

return result;

end;

**--variable**

declare

v\_name varchar2(20):='rj';

begin

insert into message\_info values(:v\_name, 'At the starting of the main block');

end;

**--exception handling**

**--exception is a runtime error or warning condition can be pre defined and used defined**

**--exceptions can be occured declaration section, execution section or even at exception section of anonymous block**

**--predefined exceptions raised imlicitly and user defined execptions need be raised by user explicitly**

declare

v\_empno number:=123;

v\_ename varchar(10);

begin

select ename into :v\_ename from emp where empno=:v\_empno limit 1;

return 'name of the employee:-->'||:v\_ename||' for the given empno -->'||:v\_empno;

end;

**--handling exception that occurs in execution section**

declare

v\_empno number:=123;

v\_ename varchar(10);

begin

select ename into :v\_ename from emp where empno=:v\_empno;

return 'Name of the employee: -->' ||:v\_ename;

exception

when other then

return object\_construct('Error type', 'other Exception', 'SQLCODE', sqlcode, 'SQLERRM', sqlerrm, 'SQLSTATE', sqlstate);

end;

**--but expection can be one of the handle so:**

declare

v\_empno number:=123;

v\_ename varchar(10);

begin

select ename into :v\_ename from emp where empno=:v\_empno;

return 'Name of the employee: -->' ||:v\_ename;

exception

when EXPRESSION\_ERROR then

return object\_construct('Error type', 'Expression Exception', 'SQLCODE', sqlcode, 'SQLERRM', sqlerrm, 'SQLSTATE', sqlstate);

when STATEMENT\_ERROR then

return object\_construct('Error type', 'Statement Exception', 'SQLCODE', sqlcode, 'SQLERRM', sqlerrm, 'SQLSTATE', sqlstate);

when other then

return object\_construct('Error type', 'other Exception', 'SQLCODE', sqlcode, 'SQLERRM', sqlerrm, 'SQLSTATE', sqlstate);

end;

**--handling exception that occurs in declaration section**

begin

declare

v\_empno number:='empno123';

v\_ename varchar (10);

begin

select ename into :v\_ename from emp where empno=:v\_empno limit 1;

return 'Name of the employee: -->'||:v\_ename;

exception

when other then

return object\_construct('Error type', 'Inner loop Exception', 'SQLCODE', sqlcode, 'SQLERRM', sqlerrm, 'SQLSTATE', sqlstate);

end;

exception

when other then

return object\_construct('Error type', 'Outer loop Exception', 'SQLCODE', sqlcode, 'SQLERRM', sqlerrm, 'SQLSTATE', sqlstate);

end;

**--user defined exception**

declare

date\_of\_birth date :='2000-10-21';

age\_limit\_exception exception(-20021, 'Age is out range for this position');

begin

let age number:=datediff(year,date\_of\_birth,current\_date());

if (age>=35) then

raise age\_limit\_exception;

end if;

exception

when age\_limit\_exception then

return object\_construct('Error type', 'User define loop Exception', 'SQLCODE', sqlcode, 'SQLERRM', sqlerrm, 'SQLSTATE', sqlstate);

end;

**--passing exception from inner block to outer block**

begin

declare

v\_empno number:='empno1234';

v\_ename varchar(10);

begin

select ename into :v\_ename from emp where empno=:v\_empno limit 1;

return 'Name of the Employee:-->'||:v\_ename;

exception

when other then

raise; --passing exception to outer block

end;

exception

when other then

return object\_construct('Error type', 'Outer loop Exception', 'SQLCODE', sqlcode, 'SQLERRM', sqlerrm, 'SQLSTATE', sqlstate);

end;

–-**Cursor**

**--cursor is a pointer which points the records in active set and fetches records one after another till end**

use database snowscript;

use schema scripts;

**--stages in cursor**

**--1. declaration 2. open 3. fetch 4. close**

declare

emp\_cur cursor for select \* from emp;

v\_empno number;

v\_ename varchar;

begin

open emp\_cur;

fetch emp\_cur into :v\_empno,:v\_ename;

fetch emp\_cur into :v\_empno,:v\_ename;

close emp\_cur;

return :v\_empno||' '||:v\_ename;

end;

**--for loop**

Here we don't need to specify parameters for loop will do that

declare

emp\_cur cursor for select \* from emp;

– v\_empno number;

– v\_ename varchar;

– v\_counter integer :=0;

begin

-- open emp\_cur;

-- fetch emp\_cur into :v\_empno,:v\_ename;

-- fetch emp\_cur into :v\_empno,:v\_ename;

-- close emp\_cur;

-- return :v\_empno||' '||:v\_ename;

for v\_for\_cursor in emp\_cur loop

return v\_for\_cursor.empno||' '||v\_for\_cursor.ename;

end loop;

end;