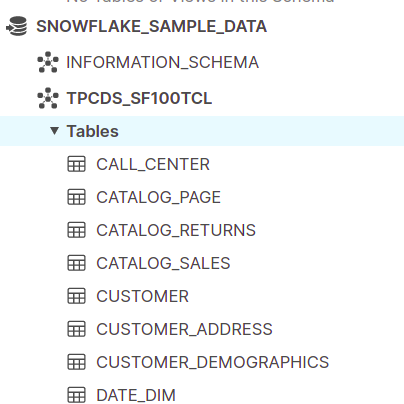
**SNOWFLAKE WORKBOOK**

**Basic queries**

**Databases and schemas are used to organize data stored in Snowflake:**

* **A database is a logical grouping of schemas. Each database belongs to a single Snowflake account.**
* **A schema is a logical grouping of database objects (tables, views, etc.). Each schema belongs to a single database.**
* **Database**
* **=>Schemas**
* **=>Tables**

****

1. **Introduction to DW , cloud and Distributed computing**
2. **Intro to snowflake architecture**
3. **Creating database,schema and table**
4. **DDL and DML operations on Table**
5. **Micropartition and cluster key**
6. **Snowsql installation and configuration**
7. **Types of stages**
8. **Loading data into user stage ,table and named stage**

--COMMAND TO LIST DATABASES

show databases;

--TO CREATE NEW DATABASE

Create database ltidb;

--TO SELECT A DATABASE

use ltidb;

--TO CREATE NEW SCHEMA

Create schema ltischema;

--TO SELECT A SCHEMA

use ltischema;

--TO LIST TABLES

show tables;

--TO LIST SCHEMAS

show schemas;

--TO CREATE A TABLE

create table emp (empno int, employeename varchar(20),salary int);

--TO CREATE A WAREHOUSE

CREATE WAREHOUSE insurance\_wh WITH WAREHOUSE\_SIZE = 'SMALL' WAREHOUSE\_TYPE = 'STANDARD' AUTO\_SUSPEND = 600 AUTO\_RESUME = TRUE MIN\_CLUSTER\_COUNT = 1 MAX\_CLUSTER\_COUNT = 2 SCALING\_POLICY = 'STANDARD';

--TO INSERT NEW RECORDS

insert into employee values(1000,'Gautam',60000);

insert into employee values(1001,'Arjun',70000);

insert into employee values(1002,'Viji',50000);

--TO VIEW THE RECORDS

select \* from ltidb.ltischema.employee;

--To update

Update employee set sal =99999 where empno =1000;

--To delete a records

Delete from employee where empno=1002;

**Snowsql -Install and configure**

**for windows**

**https://sfc-repo.snowflakecomputing.com/snowsql/bootstrap/1.2/windows\_x86\_6 4/snowsql-1.2.10-windows\_x86\_64.msi**

1. **In windows search , type snowsql**
2. **Open snowsql ,run as a administrator . It install the snowsql**

**--to get config file , run snowsql in command line ,It**

**--Direct connection to your snowflake account**

**--get account name from the url**

**https://vna82064.us-east-1.snowflakecomputing.com/console#/internal/**

**-a vna82064.us-east-1**

**-u akgeoinsys**

**snowsql -a vna82064.us-east-1 -u akgeoinsys**

**=================================================**

**--comment all other connection names**

**To configure the .snowsql/config file**

**[connections.anand]**

**accountname = rna43325.us-east-1**

**username = akgeoinsys**

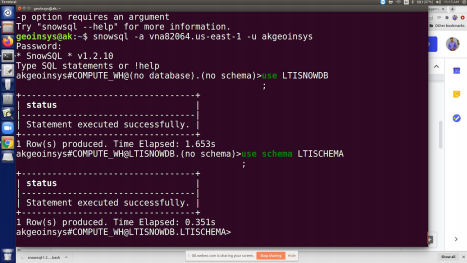
**password = Pa55word**

**--In command prompt ,type the following command**

**snowsql -c anand**

**Or**

**snowsql -a rna43325.us-east-1 -u akgeoinsys**

****

**Stages**

Creates a new named *internal* or *external* stage to use for loading data from files into Snowflake tables and unloading data from tables into files:

**Internal stage**

Stores data files internally within Snowflake. Internal stages can be either permanent or temporary. For more details, see [Choosing a Stage for Local Files](https://docs.snowflake.com/en/user-guide/data-load-local-file-system-create-stage.html).

**External stage**

References data files stored in a location outside of Snowflake. Currently, the following cloud storage services are supported:

* Amazon S3 buckets
* Google Cloud Storage buckets
* Microsoft Azure containers

The storage location can be either private/protected or public.

use role sysadmin;

use ltidb;

use schema ltischema;

create or replace table superstore(

RowID integer

, OrderID varchar(100)

, OrderDate varchar(50)

, ShipDate varchar(50)

, ShipMode varchar(50)

, CustomerID varchar(50)

, CustomerName varchar(200)

, CustomerSegment varchar(200)

, City varchar(200)

, State varchar(200)

, Country varchar(200)

, ZipCode varchar(50)

, Market varchar(100)

, Region varchar(200)

, ProductID varchar(200)

, ProductCategory varchar(200)

, ProductsubCategory varchar(200)

, ProductName varchar(500)

, Sales decimal(8,2)

, OrderQuantity integer

, Discount decimal(8,2)

, Profit decimal(8,2)

, ShippingCost decimal(8,2)

, OrderPriority varchar(50)

);

--To Download the datasets

<https://drive.google.com/drive/folders/19oqxO2builOEYEybDj_yZubYpufwod7C?usp=sharing>

Show tables;

--Implicitly created with snowflake user

--to list files in user stage (worksheet , loaded data files )

list @~;

--table stage are created in tablename

--@%tablename

list @%superstore;

show stages;

--write data into userstage

put file://path @~;

--write data into table stage

put file://path @%superstore;

truncate superstore;

1. **Data loading into stages**
2. **Querying stages**
3. **Transformation on stages data while copying**
4. **Introduction to AWS IAM and S3**
5. **Creating bucket and uploading data**
6. **Creating external stage on AWS s3 bucket using access id and secret**
7. **Copy data into snowflake table from external stage**
8. **Different types of table**
9. **Time travel and failsafe**
10. **Zero copy cloning**
11. **Types of Cache**

**--create file format to load superstore.csv file ( header present , delimiter =comma)**

CREATE or replace FILE FORMAT "LTIDB"."LTISCHEMA".CSVSALESFF COMPRESSION = 'AUTO' FIELD\_DELIMITER = ',' RECORD\_DELIMITER = '\n' SKIP\_HEADER = 1 FIELD\_OPTIONALLY\_ENCLOSED\_BY = 'NONE' TRIM\_SPACE = FALSE ERROR\_ON\_COLUMN\_COUNT\_MISMATCH = TRUE ESCAPE = 'NONE' ESCAPE\_UNENCLOSED\_FIELD = '\134'

DATE\_FORMAT = 'AUTO' TIMESTAMP\_FORMAT = 'AUTO' NULL\_IF = ('\\N');

--create internal named stage to write multiple file with local

CREATE or replace STAGE ltischema.int\_salesstage file\_format = csvsalesff;

COPY INTO ltidb.ltischema.superstore FROM @salesstage FILE\_FORMAT = 'ltidb.ltischema.csv\_fileformat' ON\_ERROR = 'CONTINUE' PURGE = TRUE;

copy into ltidb.ltischema.superstore from s3://geoinsyssnowflake

credentials=(aws\_key\_id='$AWS\_ACCESS\_KEY\_ID'

aws\_secret\_key='$AWS\_SECRET\_ACCESS\_KEY') file\_format = (type =

csv field\_delimiter = ',' skip\_header = 1);

**Types of table**

--temp table available in the created session /user

create temporary table mytemptable (id number, creation\_date date);

--transient table like permanent table except time travel >1 day and failsafe

create transient table mytranstable (id number, creation\_date date);

-- normal table are permanent table has time travel for 90 days and fail safe

create table mypermtable (id number, creation\_date date);

**Unloading**

create or replace stage unload\_stage file\_format = csvfileformat;

copy into @UNLOAD\_STAGE\_orc from

SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF1.CUSTOMER;

copy into @UNLOAD\_STAGE from

ltidb.ltischema.superstore;

use role accountadmin;

list @unload\_stage;

-----to download data from stage to local-------

-----make sure you give existing directory path-------

get @unload\_stage/data\_0\_0\_0.csv.gz file://c:\unloaddata;

get @unload\_stage/data\_0\_0\_0.csv.gz file:///home/geoinsys/data/

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF1.ORDERS;

select system$clustering\_information('superstore', '(country, city)');

show tables like 'superstore';

select table\_schema,sum(bytes)

from ltidb.information\_schema.tables

group by table\_schema;

alter table superstore suspend recluster;

alter table superstore cluster by (orderid,country);

select \* from emp;

from table(information\_schema.automatic\_clustering\_history(

date\_range\_start=>dateadd(d, -7, current\_date),

date\_range\_end=>current\_date));

================================================================

create table varia (float1 float, v variant, float2 float);

insert into varia (float1, v, float2) values (1.23, null, null);

select \* from varia;

update varia set v = to\_variant(float1); -- converts FROM a float TO a variant.

update varia set float2 = v::float; -- converts FROM a variant TO a float.

select \* from varia;

------

create table demonstration1 (

id integer,

array1 array,

variant1 variant,

object1 object

);

insert into demonstration1 (id, array1, variant1, object1)

select

1,

array\_construct(1, 2, 3),

parse\_json(' { "key1": "value1", "key2": "value2" } '),

parse\_json(' { "outer\_key1": { "inner\_key1A": "1a", "inner\_key1B": "1b" }, '

||

' "outer\_key2": { "inner\_key2": 2 } } ')

;

select \* from demonstration1;

create or replace table car\_sales

(

src variant

)

as

select parse\_json(column1) as src

from values

('{

"date" : "2017-04-28",

"dealership" : "Valley View Auto Sales",

"salesperson" : {

"id": "55",

"name": "Frank Beasley"

},

"customer" : [

{"name": "Joyce Ridgely", "phone": "16504378889", "address": "San Francisco, CA"}

],

"vehicle" : [

{"make": "Honda", "model": "Civic", "year": "2017", "price": "20275", "extras":["ext warranty", "paint protection"]}

]

}'),

('{

"date" : "2017-04-28",

"dealership" : "Tindel Toyota",

"salesperson" : {

"id": "274",

"name": "Greg Northrup"

},

"customer" : [

{"name": "Bradley Greenbloom", "phone": "12127593751", "address": "New York, NY"}

],

"vehicle" : [

{"make": "Toyota", "model": "Camry", "year": "2017", "price": "23500", "extras":["ext warranty", "rust proofing", "fabric protection"]}

]

}') v;

select \* from car\_sales;

select src from car\_sales;

desc table car\_sales;

select src:dealership from car\_sales;

--External table

select SS\_STORE\_SK from store\_sales where ss\_store\_sk=856 ;

create or replace external table patient\_ext\_variant

with location = @ST\_S3\_TIAA/healthcare

file\_format = csv\_fileformat;

select \* from patient\_ext\_variant;

select

value:c1::int as ID,

value:c2::varchar as pname ,

value:c3::varchar as drug,

value:c4::varchar as gender,

value:c5::int as amt

from patient\_ext\_variant;

create or replace external table patient\_ext

(ID INT as (value:c1::int),

PNAME varchar(20) as ( value:c2::varchar),

DRUG varchar(20) as ( value:c3::varchar),

GENDER varchar(10) as ( value:c4::varchar),

AMOUNT int as (value:c5::int))

with location = @ST\_S3\_TIAA/healthcare

AUTO\_REFRESH = true

file\_format = csv\_fileformat;

select \* from patient\_ext;

ALTER EXTERNAL TABLE patient\_ext REFRESH;

SELECT metadata$filename FROM @st\_s3\_tiaa;

select id,pname,drug,gender,amount from patient\_ext;

ALTER EXTERNAL TABLE emp\_part REFRESH;

SELECT metadata$filename FROM @st\_s3\_tiaa/part/;

CREATE or replace EXTERNAL TABLE emp\_part(

date\_part date as to\_date(substr(metadata$filename, 6, 10), 'YYYY/MM/DD'),

empno int as (value:c1::int),

ename varchar AS (value:c2::varchar),

salary int as (value:c3::int))

PARTITION BY (date\_part)

LOCATION=@st\_s3\_tiaa/part/

AUTO\_REFRESH = true

FILE\_FORMAT = csv\_fileformat;

select \* from emp\_part;

desc table emp\_part;

ALTER EXTERNAL TABLE emp\_part REFRESH;

select \* from emp\_part where date\_part = to\_date('04/02/2021');;

--types of table

delete from superstore where rowid=36328;

desc table mytemptable;

show tables;

CREATE TABLE "LTIDB"."LTISCHEMA".car1 LIKE "LTIDB"."LTISCHEMA"."CAR\_SALES";

desc table car1;

select \* from car;

select \* from carclone;

CREATE TABLE "LTIDB"."LTISCHEMA".carclone clone "LTIDB"."LTISCHEMA"."CAR\_SALES";

CREATE TABLE "LTIDB"."LTISCHEMA".car as select \* from car\_sales;

Create or Replace Database P\_database01;

show databases like '%P\_database01%'; --options column does not mention that it is transient DB

Create or Replace Transient Schema T\_schema01;

show schemas in database P\_database01; --options column does mention that it is transient Schema

Create or Replace Table T\_table01(interger\_Column int);

select get\_ddl('table','P\_database01.T\_schema01.T\_table01');

--create or replace TRANSIENT TABLE T\_TABLE01 (INTERGER\_COLUMN NUMBER (38,0) );

select \* from carclone;

drop table car;

commit;

alter table superstore set data\_retention\_time\_in\_days=2;

select current\_date(),current\_time();

select \* from superstore before (TIMESTAMP => 'Mon, 12 April 2021 01:40:00 -0700'::timestamp\_tz);

select \* from emp at(offset => -3600);

show tables ;

select \* from emp;

select \* from restored\_emp ;

delete from emp where empno=555;

select \* from emp before(statement => '019b97b3-0600-8277-0041-9f0300052136');

show databases history;

create database p\_database01;

show tables history like 'pat%' in ltidb.ltischema;

drop table superstore;

drop database p\_database01;

select \* from orders;

alter database p\_database01 rename to p\_database02;

delete from orders where o\_orderkey=1611841;

undrop database p\_database01;

create table restored\_emp clone emp

at(timestamp => 'Wed, 14 April 2021 16:10:00 -0700'::timestamp\_tz);

--cache

select \* from tpch\_sf1.orders limit 3;

create table ltidb.ltischema.orders as select \* from tpch\_sf1.orders;

select \* from ltischema.orders where o\_orderkey=1740673;

select count(\*) from ltischema.orders;

update orders set o\_totalprice=9999999.10 where o\_orderkey =1740673;

create table ltidb.ltischema.orders\_demo as select \* from ltischema.orders;

create view v\_order as select \* from ltischema.orders\_demo;

select \* from v\_order limit 100;

select \* from v\_order limit 100;

--metadata cache

desc orders;

select count (\*) from orders\_demo;

select min(o\_totalprice) from orders;

--result cache 24 hrs cache - expire time is altered same query is fired

select \* from ltischema.orders;

select current\_time();

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

--snow pipes

use role accountadmin;

use database ltidb;

use schema ltischema;

create or replace storage integration s3\_int

type = external\_stage

storage\_provider = s3

enabled = true

storage\_aws\_role\_arn = 'arn:aws:iam::574997813424:role/insurancesnowpiperole'

storage\_allowed\_locations = ('s3://geoinsyssnowflake/healthcare');

-- grant usage on integration s3\_int to role sysadmin;

desc integration s3\_int;

CREATE FILE FORMAT "LTIDB"."LTISCHEMA".csv\_pipe TYPE = 'CSV' COMPRESSION = 'AUTO' FIELD\_DELIMITER = ',' RECORD\_DELIMITER = '\n' SKIP\_HEADER = 0 FIELD\_OPTIONALLY\_ENCLOSED\_BY = 'NONE' TRIM\_SPACE = FALSE ERROR\_ON\_COLUMN\_COUNT\_MISMATCH = TRUE ESCAPE = 'NONE' ESCAPE\_UNENCLOSED\_FIELD = '\134' DATE\_FORMAT = 'AUTO' TIMESTAMP\_FORMAT = 'AUTO' NULL\_IF = ('\\N');

create or replace stage st\_ak\_s3\_pipe

storage\_integration = s3\_int

url = 's3://geoinsyssnowflake/healthcare'

file\_format = csv\_pipe;

show stages;

list @st\_ak\_s3\_pipe;

create table ltidb.ltischema.emp\_pipe (empno int ,

ename varchar(20) ,

sal int );

--grant create stage on schema to role accountadmin;

desc pipe s3\_pipe\_emp ;

ALTER PIPE s3\_pipe\_emp SET PIPE\_EXECUTION\_PAUSED=false;

SYSTEM$PIPE\_FORCE\_RESUME('S3\_PIPE');;

SYSTEM$PIPE\_FORCE\_RESUME('S3\_PIPE\_EMP');

select system$pipe\_force\_resume('s3\_pipe\_emp');

create table lti\_db.public.patient (pid int ,

pname varchar(20) ,

drug varchar(8) ,

gender varchar(20) ,

amount int );

select \* from emp\_pipe;

use role sysadmin;

alter pipe S3\_PIPE\_emp refresh;

COPY INTO "LTIDB"."LTISCHEMA"."EMP\_PIPE" FROM @"LTIDB"."LTISCHEMA"."ST\_AK\_S3\_PIPE" FILE\_FORMAT = ( FORMAT\_NAME = "LTIDB"."LTISCHEMA"."CSV\_PIPE" )

--stream and task

use role sysadmin;

use ltidb;

use schema public;

create or replace table cdc\_tbl (

empno int ,

ename varchar,

sal int

);

insert into cdc\_tbl values

(1,'aaa',9000),

(2,'bbb',8000),

(3,'ccc',7000);

select \* from cdc\_tbl;

--the final table where post cdc, data will

create or replace table final\_aktbl (

final\_empno int ,

final\_ename varchar ,

final\_sal int

);

-- 1st time data load from cdc\_table to final table, we can assume history load or onetime load

insert into final\_aktbl select \* from cdc\_tbl;

insert into final\_aktbl select \* from cdc\_tbl;

select \* from final\_aktbl;

create or replace stream

cdc\_stream on table cdc\_tbl

append\_only=true;

create or replace stream

ak\_stream on table cdc\_tbl;

show streams;

select \* from cdc\_stream;

select \* from ak\_stream;

insert into cdc\_tbl values

(4,'ddd',9000),

(5,'eee',8000),

(6,'fff',7000);

update cdc\_tbl set sal =8989 where empno=4;

delete from cdc\_tbl where empno=1;

insert into final\_aktbl select empno,ename,sal from ak\_stream;

--stream with merge

use role sysadmin;

use ltidb;

use schema public;

create or replace table cdc\_tbl (

empno int ,

ename varchar,

sal int

);

insert into cdc\_tbl values

(1,'aaa',9000),

(2,'bbb',8000),

(3,'ccc',7000);

select \* from cdc\_tbl;

--the final table where post cdc, data will

create or replace table final\_aktbl (

empno int ,

ename varchar ,

sal int

);

-- 1st time data load from cdc\_table to final table, we can assume history load or onetime load

insert into final\_aktbl select \* from cdc\_tbl;

create or replace stream

ak\_stream on table cdc\_tbl;

show streams;

select \* from ak\_stream;

insert into cdc\_tbl values

(7,'dtt',9000);

(5,'eee',8000),

(6,'fff',7000);

update cdc\_tbl set sal =1111 where empno=7;

delete from cdc\_tbl where empno=1;

alter session set ERROR\_ON\_NONDETERMINISTIC\_MERGE=false;

merge into ltidb.public.final\_aktbl as t

using (select \*

from ltidb.public.ak\_stream) as s

on t.empno=s.empno

when matched

and s.metadata$action = 'INSERT'

and s.metadata$isupdate then

update set t.ename=s.ename , t.sal =s.sal

when not matched

and s.metadata$action = 'INSERT' then

insert (empno, ename,sal ) values (s.empno,s.ename,s.sal);

merge into ltidb.public.final\_aktbl as t

using (select \*

from ltidb.public.ak\_stream) as s

on t.empno=s.empno

when matched

and s.metadata$action = 'INSERT'

and s.metadata$isupdate then update set t.ename=s.ename ,t.sal =s.sal

when matched

and s.metadata$action = 'DELETE' THEN DELETE

when not matched

and s.metadata$action = 'INSERT' then

insert (t.empno, t.ename,t.sal ) values (s.empno,s.ename,s.sal);

select \* from final\_aktbl;

delete from final\_aktbl where empno=3;

select system$stream\_has\_data('ak\_stream');

select current\_time();

select \* from ak\_stream;

--task

create or replace task cdc\_task

warehouse = compute\_wh

schedule = '5 minute'

when

system$stream\_has\_data('ak\_stream')

as

insert into final\_aktbl select empno,ename,sal from ak\_stream;

show tasks;

desc task cdc\_task;

use role accountadmin;

alter task cdc\_task resume;

alter task cdc\_task suspend;

select \* from table(information\_schema.task\_history())

order by scheduled\_time;

-- you can see only the schedule items

select \* from table(information\_schema.task\_history())

where state ='SCHEDULED' order by scheduled\_time;

-- SCD /CDC

show stages;

taskadmin;

-- Set the active role to SECURITYADMIN to show that this role can grant a role to another role

use role securityadmin;

grant role taskadmin to role sysadmin;

use role sysadmin;

create warehouse if not exists task\_warehouse with warehouse\_size = 'XSMALL' auto\_suspend = 120;

-- Create a task to schedule the MERGE statement

create or replace task populate\_nation\_history warehouse = task\_warehouse schedule = '2 minute' when system$stream\_has\_data('nation\_table\_changes')

as

merge into nation\_history nh

using nation\_change\_data m

on nh.n\_nationkey = m.n\_nationkey

and nh.start\_time = m.start\_time

when matched and m.dml\_type = 'U' then update

set nh.end\_time = m.end\_time,

nh.current\_flag = 0

when matched and m.dml\_type = 'D' then update

set nh.end\_time = m.end\_time,

nh.current\_flag = 0

when not matched and m.dml\_type = 'I' then insert

(n\_nationkey, n\_name, n\_regionkey, n\_comment,

country\_code, start\_time, end\_time, current\_flag)

values (m.n\_nationkey, m.n\_name, m.n\_regionkey, m.n\_comment,

m.country\_code, m.start\_tise role sysadmin;

create database streams\_and\_tasks;

use database streams\_and\_tasks;

create schema scd;

use schema scd;

create or replace table nation (

n\_nationkey number,

n\_name varchar(25),

n\_regionkey number,

n\_comment varchar(152),

country\_code varchar(2),

update\_timestamp timestamp\_ntz);

create or replace table nation\_history (

n\_nationkey number,

n\_name varchar(25),

n\_regionkey number,

n\_comment varchar(152),

country\_code varchar(2),

start\_time timestamp\_ntz,

end\_time timestamp\_ntz,

current\_flag int);

create or replace stream nation\_table\_changes on table nation;

show streams;

select \* from nation\_table\_changes;

create or replace view nation\_change\_data as

-- This subquery figures out what to do when data is inserted into the NATION table

-- An insert to the NATION table results in an INSERT to the NATION\_HISTORY table

select n\_nationkey, n\_name, n\_regionkey, n\_comment,

country\_code, start\_time, end\_time, current\_flag, 'I' as dml\_type

from (select n\_nationkey, n\_name, n\_regionkey, n\_comment, country\_code,

update\_timestamp as start\_time,

lag(update\_timestamp) over (partition by n\_nationkey order by update\_timestamp desc) as end\_time\_raw,

case when end\_time\_raw is null then '9999-12-31'::timestamp\_ntz else end\_time\_raw end as end\_time,

case when end\_time\_raw is null then 1 else 0 end as current\_flag

from (select n\_nationkey, n\_name, n\_regionkey, n\_comment, country\_code, update\_timestamp

from nation\_table\_changes

where metadata$action = 'INSERT'

and metadata$isupdate = 'FALSE'))

union

-- This subquery figures out what to do when data is updated in the NATION table

-- An update to the NATION table results in an update AND an insert to the NATION\_HISTORY table

-- The subquery below generates two records, each with a different dml\_type

select n\_nationkey, n\_name, n\_regionkey, n\_comment, country\_code, start\_time, end\_time, current\_flag, dml\_type

from (select n\_nationkey, n\_name, n\_regionkey, n\_comment, country\_code,

update\_timestamp as start\_time,

lag(update\_timestamp) over (partition by n\_nationkey order by update\_timestamp desc) as end\_time\_raw,

case when end\_time\_raw is null then '9999-12-31'::timestamp\_ntz else end\_time\_raw end as end\_time,

case when end\_time\_raw is null then 1 else 0 end as current\_flag,

dml\_type

from (-- Identify data to insert into nation\_history table

select n\_nationkey, n\_name, n\_regionkey, n\_comment, country\_code, update\_timestamp, 'I' as dml\_type

from nation\_table\_changes

where metadata$action = 'INSERT'

and metadata$isupdate = 'TRUE'

union

-- Identify data in NATION\_HISTORY table that needs to be updated

select n\_nationkey, null, null, null, null, start\_time, 'U' as dml\_type

from nation\_history

where n\_nationkey in (select distinct n\_nationkey

from nation\_table\_changes

where metadata$action = 'INSERT'

and metadata$isupdate = 'TRUE')

and current\_flag = 1))

union

-- This subquery figures out what to do when data is deleted from the NATION table

-- A deletion from the NATION table results in an update to the NATION\_HISTORY table

select nms.n\_nationkey, null, null, null, null, nh.start\_time, current\_timestamp()::timestamp\_ntz, null, 'D'

from nation\_history nh

inner join nation\_table\_changes nms

on nh.n\_nationkey = nms.n\_nationkey

where nms.metadata$action = 'DELETE'

and nms.metadata$isupdate = 'FALSE'

and nh.current\_flag = 1;

--merge to maintain SCD

merge into nation\_history nh -- Target table to merge changes from NATION into

using nation\_change\_data m -- nation\_change\_data is a view that holds the logic that determines what to insert/update into the NATION\_HISTORY table.

on nh.n\_nationkey = m.n\_nationkey -- n\_nationkey and start\_time determine whether there is a unique record in the NATION\_HISTORY table

and nh.start\_time = m.start\_time

when matched and m.dml\_type = 'U' then update -- Indicates the record has been updated and is no longer current and the end\_time needs to be stamped

set nh.end\_time = m.end\_time,

nh.current\_flag = 0

when matched and m.dml\_type = 'D' then update -- Deletes are essentially logical deletes. The record is stamped and no newer version is inserted

set nh.end\_time = m.end\_time,

nh.current\_flag = 0

when not matched and m.dml\_type = 'I' then insert -- Inserting a new n\_nationkey and updating an existing one both result in an insert

(n\_nationkey, n\_name, n\_regionkey, n\_comment, country\_code, start\_time, end\_time, current\_flag)

values (m.n\_nationkey, m.n\_name, m.n\_regionkey, m.n\_comment, m.country\_code, m.start\_time, m.end\_time, m.current\_flag);

-- insert new records

set update\_timestamp = current\_timestamp()::timestamp\_ntz;

begin;

insert into nation values(0,'ALGERIA',0,' haggle. carefully final deposits detect slyly agai','DZ',$update\_timestamp);

insert into nation values(1,'ARGENTINA',1,'al foxes promise slyly according to the regular accounts. bold requests alon','AR',$update\_timestamp);

insert into nation values(2,'BRAZIL',1,'y alongside of the pending deposits. carefully special packages are about the ironic forges. slyly special ','BR',$update\_timestamp);

insert into nation values(3,'CANADA',1,'eas hang ironic silent packages. slyly regular packages are furiously over the tithes. fluffily bold','CA',$update\_timestamp);

insert into nation values(4,'EGYPT',4,'y above the carefully unusual theodolites. final dugouts are quickly across the furiously regular d','EG',$update\_timestamp);

insert into nation values(5,'ETHIOPIA',0,'ven packages wake quickly. regu','ET',$update\_timestamp);

insert into nation values(6,'FRANCE',3,'refully final requests. regular ironi','FR',$update\_timestamp);

insert into nation values(7,'GERMANY',3,'l platelets. regular accounts x-ray: unusual regular acco','DE',$update\_timestamp);

insert into nation values(8,'INDIA',2,'ss excuses cajole slyly across the packages. deposits print aroun','IN',$update\_timestamp);

insert into nation values(9,'INDONESIA',2,' slyly express asymptotes. regular deposits haggle slyly. carefully ironic hockey players sleep blithely. carefull','ID',$update\_timestamp);

insert into nation values(10,'IRAN',4,'efully alongside of the slyly final dependencies. ','IR',$update\_timestamp);

insert into nation values(11,'IRAQ',4,'nic deposits boost atop the quickly final requests? quickly regula','IQ',$update\_timestamp);

insert into nation values(12,'JAPAN',2,'ously. final express gifts cajole a','JP',$update\_timestamp);

insert into nation values(13,'JORDAN',4,'ic deposits are blithely about the carefully regular pa','JO',$update\_timestamp);

insert into nation values(14,'KENYA',0,' pending excuses haggle furiously deposits. pending express pinto beans wake fluffily past t','KE',$update\_timestamp);

insert into nation values(15,'MOROCCO',0,'rns. blithely bold courts among the closely regular packages use furiously bold platelets?','MA',$update\_timestamp);

insert into nation values(16,'MOZAMBIQUE',0,'s. ironic unusual asymptotes wake blithely r','MZ',$update\_timestamp);

insert into nation values(17,'PERU',1,'platelets. blithely pending dependencies use fluffily across the even pinto beans. carefully silent accoun','PE',$update\_timestamp);

insert into nation values(18,'CHINA',2,'c dependencies. furiously express notornis sleep slyly regular accounts. ideas sleep. depos','CN',$update\_timestamp);

insert into nation values(19,'ROMANIA',3,'ular asymptotes are about the furious multipliers. express dependencies nag above the ironically ironic account','RO',$update\_timestamp);

insert into nation values(20,'SAUDI ARABIA',4,'ts. silent requests haggle. closely express packages sleep across the blithely','SA',$update\_timestamp);

insert into nation values(21,'VIETNAM',2,'hely enticingly express accounts. even final ','VN',$update\_timestamp);

insert into nation values(22,'RUSSIA',3,' requests against the platelets use never according to the quickly regular pint','RU',$update\_timestamp);

insert into nation values(23,'UNITED KINGDOM',3,'eans boost carefully special requests. accounts are. carefull','GB',$update\_timestamp);

insert into nation values(24,'UNITED STATES',1,'y final packages. slow foxes cajole quickly. quickly silent platelets breach ironic accounts. unusual pinto be','US',$update\_timestamp);

commit;

-- check the tables after merge

select \* from nation;

select \* from nation\_table\_changes;

-- MERGE statement that uses the CHANGE\_DATA view to load data into the NATION\_HISTORY table

merge into nation\_history nh -- Target table to merge changes from NATION into

using nation\_change\_data m -- NATION\_CHANGE\_DATA is a view that holds the logic that determines what to insert/update into the NATION\_HISTORY table.

on nh.n\_nationkey = m.n\_nationkey -- n\_nationkey and start\_time determine whether there is a unique record in the NATION\_HISTORY table

and nh.start\_time = m.start\_time

when matched and m.dml\_type = 'U' then update -- Indicates the record has been updated and is no longer current and the end\_time needs to be stamped

set nh.end\_time = m.end\_time,

nh.current\_flag = 0

when matched and m.dml\_type = 'D' then update -- Deletes are essentially logical deletes. The record is stamped and no newer version is inserted

set nh.end\_time = m.end\_time,

nh.current\_flag = 0

when not matched and m.dml\_type = 'I' then insert -- Inserting a new n\_nationkey and updating an existing one both result in an insert

(n\_nationkey, n\_name, n\_regionkey, n\_comment, country\_code, start\_time, end\_time, current\_flag)

values (m.n\_nationkey, m.n\_name, m.n\_regionkey, m.n\_comment,m.country\_code, m.start\_time, m.end\_time, m.current\_flag);

--query

select \* from nation\_history;

select \* from nation\_table\_changes;

begin;

update nation

set n\_comment = 'New comment for Arg', update\_timestamp = current\_timestamp()::timestamp\_ntz

where n\_nationkey = 1;

update nation

set n\_comment = 'New comment for Canada', update\_timestamp = current\_timestamp()::timestamp\_ntz

where n\_nationkey = 3;

commit;

select \* from nation where n\_nationkey in (1, 2,3);

select \* from nation\_table\_changes;

-- MERGE statement that uses the CHANGE\_DATA view to load data into the NATION\_HISTORY table

merge into nation\_history nh -- Target table to merge changes from NATION into

using nation\_change\_data m -- CHANGE\_DATA is a view that holds the logic that determines what to insert/update into the NATION\_HISTORY table.

on nh.n\_nationkey = m.n\_nationkey -- n\_nationkey and start\_time determine whether there is a unique record in the NATION\_HISTORY table

and nh.start\_time = m.start\_time

when matched and m.dml\_type = 'U' then update -- Indicates the record has been updated and is no longer current and the end\_time needs to be stamped

set nh.end\_time = m.end\_time,

nh.current\_flag = 0

when matched and m.dml\_type = 'D' then update -- Deletes are essentially logical deletes. The record is stamped and no newer version is inserted

set nh.end\_time = m.end\_time,

nh.current\_flag = 0

when not matched and m.dml\_type = 'I' then insert -- Inserting a new n\_nationkey and updating an existing one both result in an Insert

(n\_nationkey, n\_name, n\_regionkey, n\_comment, country\_code, start\_time, end\_time, current\_flag)

values (m.n\_nationkey, m.n\_name, m.n\_regionkey, m.n\_comment,

m.country\_code, m.start\_time, m.end\_time, m.current\_flag);

select \* from nation\_history;

select \* from nation\_history where n\_nationkey in (2,3) order by n\_nationkey, start\_time;

select \* from nation\_table\_changes;

--Set up TASKADMIN role

use role securityadmin;

create role taskadmin;

-- Set the active role to ACCOUNTADMIN before granting the EXECUTE TASK privilege to TASKADMIN

use role accountadmin;

grant execute task on account to roleme, m.end\_time, m.current\_flag);

show tasks;

alter task populate\_nation\_history resume;

show tasks;

select timestampdiff(second, current\_timestamp, scheduled\_time) as next\_run, scheduled\_time, current\_timestamp, name, state

from table(information\_schema.task\_history()) where state = 'SCHEDULED' order by completed\_time desc;

select timestampdiff(second, current\_timestamp, scheduled\_time) as next\_run, scheduled\_time, current\_timestamp, name, state

from table(information\_schema.task\_history()) order by completed\_time desc;

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

-- Delete data

delete from nation where n\_nationkey in (3,7);

select \* from nation\_table\_changes;

select timestampdiff(second, current\_timestamp, scheduled\_time) as next\_run, scheduled\_time, current\_timestamp, name, state

from table(information\_schema.task\_history()) order by completed\_time desc;

select \* from nation\_history where n\_nationkey in (3,7) order by n\_nationkey, start\_time;

-- Insert, update, delete in one pass

begin;

insert into nation values(26, 'COLOMBIA', 1, 'New country', 'CO', current\_timestamp()::timestamp\_ntz);

update nation

set n\_comment = 'New comment for Indonesia', update\_timestamp =

current\_timestamp()::timestamp\_ntz

where n\_nationkey = 9;

delete from nation

where n\_nationkey in (20);

commit;

select \* from nation\_table\_changes;

select \* from nation where n\_nationkey in (26,9,20);

select \* from nation\_history;

select \* from nation\_history where n\_nationkey in (26,9,20);

alter task populate\_nation\_history suspend;

show tasks;

--time travel

alter table patient set data\_retention\_time\_in\_days=30;

show tables like 'patient%';

select \* from patient where drug ='Para';

delete from patient where pid=111;

select count(\*) from patient;

select count(\*) from patient at(offset => -60\*5);

create schema restored\_schema clone ltischema at(offset => -300);

select count(\*) from patient before(timestamp => 'Sun, 11 April 2021 22:19:00 -0700'::timestamp\_tz);

select current\_timestamp();