# Change data capture in Snowflake

Description:

CDC (Change data capture) is the practive of detecting the deltas coming from a source and optimizing the propagation to tables according to the business need and archetecture.

There are 2 types that we will be using of CDC.

SC1: The final table will have only one record per natural key. therefore the process will insert if it is a new record, and update if there was an attribute change.

SC2: The final table will have history and keep the version of each record through time. Therefore it will insert a new record, and insert/update when an attribute changed. Columns like DBT_VALID_TO, DBT_VALID_FROM, CURR_IND will be used to show the change over time. CURR_IND will be Y for latest attribution of that natural key.

Each type will have 2 stored procedures. One to output the merge statement and one to run the merge statment.

Code for SC1:

**SC1 - Stored Proc for viewing Merge Statment**

```
create or replace procedure sp_merge_type1scd_sql_sp(targetTbl string, sourceTbl string, naturalkeys string)
returns string
language javascript
comment = 'Merge with type-1 using natural keys'
execute as caller
as
$$
    //Process Source & Target keys from Input
    const tkeys = NATURALKEYS.split(',');
    const skeys = NATURALKEYS.split(',');
    const tcols = new Set();
    const scols = new Set();
    if(tkeys.length != skeys.length){
        return "ERROR: Source and Target must have the same number of entries"
    }

    // Set Join condition between source & target.
    // Variable will be reused throughout the procedure with different aliases
    var joincondition = "";
    for (let i=0;i<tkeys.length;i++){
         joincondition = joincondition.concat("tgt."+tkeys[i].trim()+" = src."+skeys[i].trim()+" and ");
    } ;
    joincondition = joincondition.substring(0, joincondition.length-4);

    //Begin Merge
    var merge = "merge into " + TARGETTBL + " t using " + SOURCETBL + " s "
    merge = merge.concat("  on " + joincondition.replace(/src/g,'s').replace(/tgt/g,'t')+ " ");

       //Get columns from source and target
    var res = snowflake.execute( {sqlText: "describe table "+TARGETTBL} );
    while(res.next()){
        tcols.add(res.getColumnValue(1));
    }
    var res = snowflake.execute( {sqlText: "describe table "+SOURCETBL} );
    while(res.next()){
        scols.add(res.getColumnValue(1));
    }

    var insert = "\t(";
    var upd_ls = "";
    var vals = "\tvalues ( \n";
    var titer = tcols.values();

    //Construct the list of values for insert
    for(let entry of titer){
```

```
        if(tcols.has(entry)){
            insert = insert.concat("\n\t" + entry +", ")
            vals = vals.concat("\t s." + entry + ", \n");

        }
    }

    var update = "\t";
    var upd_ls = "";
    var vals2 = "\t \n";
    var titer = tcols.values();
     //Construct the list of values for update
    for(let entry of titer){
        if(tcols.has(entry)){
            update = update.concat("\t " + entry + "=s." + entry + ", \n")
        }
    }
    var merge = merge.concat(" when matched then update set\n");
    var update = update.trim();
    var update_final = update.slice(0, -1);
    var merge = merge.concat(update_final);

    var merge = merge.concat(" when not matched then insert\n");
    var insert = insert.substring(0, insert.length-2).concat(")\n");
    var vals = vals.substring(0, vals.length-3).concat(")\n");
    var merge = merge.concat(insert).concat(vals);

    return merge;
    $$
;
```

**SC1 - Stored Proc to run the merge statement**

```
create or replace procedure sp_merge_type1scd_run_sp(targetTbl string, sourceTbl string, naturalkeys string)
returns string
language javascript
comment = 'Merge with type-1 using natural keys'
execute as caller
as
$$
    //Process Source & Target keys from Input
    const tkeys = NATURALKEYS.split(',');
    const skeys = NATURALKEYS.split(',');
    const tcols = new Set();
    const scols = new Set();
    if(tkeys.length != skeys.length){
        return "ERROR: Source and Target must have the same number of entries"
    }

    // Set Join condition between source & target.
    // Variable will be reused throughout the procedure with different aliases
    var joincondition = "";
    for (let i=0;i<tkeys.length;i++){
         joincondition = joincondition.concat("tgt."+tkeys[i].trim()+" = src."+skeys[i].trim()+" and ");
    } ;
    joincondition = joincondition.substring(0, joincondition.length-4);

    //Begin Merge
    var merge = "merge into " + TARGETTBL + " t using " + SOURCETBL + " s "
    merge = merge.concat("  on " + joincondition.replace(/src/g,'s').replace(/tgt/g,'t')+ " ");

       //Get columns from source and target
    var res = snowflake.execute( {sqlText: "describe table "+TARGETTBL} );
    while(res.next()){
        tcols.add(res.getColumnValue(1));
    }
    var res = snowflake.execute( {sqlText: "describe table "+SOURCETBL} );
    while(res.next()){
        scols.add(res.getColumnValue(1));
```

```
    }

    var insert = "\t(";
    var upd_ls = "";
    var vals = "\tvalues ( \n";
    var titer = tcols.values();

    //Construct the list of values for insert
    for(let entry of titer){
        if(tcols.has(entry)){
            insert = insert.concat("\n\t" + entry +", ")
            vals = vals.concat("\t s." + entry + ", \n");

        }
    }

    var update = "\t";
    var upd_ls = "";
    var vals2 = "\t \n";
    var titer = tcols.values();
     //Construct the list of values for update
    for(let entry of titer){
        if(tcols.has(entry)){
            update = update.concat("\t " + entry + "=s." + entry + ", \n")
        }
    }
    var merge = merge.concat(" when matched then update set\n");
    var update = update.trim();
    var update_final = update.slice(0, -1);
    var merge = merge.concat(update_final);

    var merge = merge.concat(" when not matched then insert\n");
    var insert = insert.substring(0, insert.length-2).concat(")\n");
    var vals = vals.substring(0, vals.length-3).concat(")\n");
    var merge = merge.concat(insert).concat(vals);

    var res = snowflake.execute( {sqlText: merge} );
    res.next();
    var insertCount = res.getColumnValue(1);
    var updateCount = res.getColumnValue(2);
    return "Merge completed. " + insertCount + " rows inserted. " + updateCount + " rows updated.";
    $$
;
```

Code for SC2:

**SC2 - Stored Proc for viewing Merge Statment**

```
create or replace procedure sp_merge_type2scd_sql_sp(targetTbl string, sourceTbl string, naturalkeys string)
returns string
language javascript
comment = 'Merge with type-2 using natural keys'
execute as caller
as
$$
    //Process Source & Target keys from Input
    const tkeys = NATURALKEYS.split(',');
    const skeys = NATURALKEYS.split(',');
    const tcols = new Set();
    const scols = new Set();
    if(tkeys.length != skeys.length){
        return "ERROR: Source and Target must have the same number of entries"
    }

    // Set Join condition between source & target.
    // Variable will be reused throughout the procedure with different aliases
    var joincondition = "";
```

```
    for (let i=0;i<tkeys.length;i++){
         joincondition = joincondition.concat("tgt."+tkeys[i].trim()+" = src."+skeys[i].trim()+" and ");
    } ;
    joincondition = joincondition.substring(0, joincondition.length-4);

    //Begin Merge
    var merge = "merge into " + TARGETTBL + " t using ( \n"
    merge = merge.concat("with \n");

    //Constructing the CTEs so that the tables can be queried once
    merge = merge.concat("\tCTE_SRC_HASH AS ( select *, SHA1(TO_JSON(OBJECT_CONSTRUCT(*))) as ROW_HASH from " +
SOURCETBL + " ), \n");
    merge = merge.concat("\tCTE_TGT_CURR AS ( select " + NATURALKEYS + ", ROW_HASH from " + TARGETTBL + " where
CURR_IND = 'Y') \n");
    merge = merge.concat("\t-- Insert for new record \n");
    merge = merge.concat("\tselect NULL as MERGE_KEY \n");
    merge = merge.concat("\t ,S1.* \n");
    merge = merge.concat("\t from CTE_SRC_HASH S1 \n");
    merge = merge.concat("\t where not exists \n");
    merge = merge.concat("\t (select 1 from CTE_TGT_CURR T1 where " + joincondition.replace(/src/g,'S1').replace
(/tgt/g,'T1') +") \n");
    merge = merge.concat("\tunion all  \n");
    merge = merge.concat("\t-- Insert for updated record \n");
    merge = merge.concat("\tselect NULL as MERGE_KEY \n");
    merge = merge.concat("\t ,S2.* \n");
    merge = merge.concat("\t from CTE_SRC_HASH S2 inner join CTE_TGT_CURR T2 ON " + joincondition.replace(/src
/g,'S2').replace(/tgt/g,'T2')+ " \n");
    merge = merge.concat("\t and S2.ROW_HASH != T2.ROW_HASH \n");
    merge = merge.concat("\tunion all \n");
    merge = merge.concat("\t-- Update for updated record \n");
    merge = merge.concat("\tselect 'Y' as MERGE_KEY \n");
    merge = merge.concat("\t ,S3.* \n");
    merge = merge.concat("\t from CTE_SRC_HASH S3 inner join CTE_TGT_CURR T3 ON \n");
    merge = merge.concat("\t "+ joincondition.replace(/src/g,'S3').replace(/tgt/g,'T3')  + " \n");
    merge = merge.concat("\t and S3.ROW_HASH != T3.ROW_HASH \n");
    merge = merge.concat(") s \n");

    //Join the contructed view with the target table
    merge = merge.concat("on \n");
    merge = merge.concat( joincondition.replace(/src/g,'S3').replace(/tgt/g,'T3') + " and s.MERGE_KEY is not
null \n" );

    //Expire records for updated records
    merge = merge.concat("when matched then update set CURR_IND = 'N', DBT_UPDATED_AT =current_timestamp(),
DBT_VALID_TO =current_timestamp() \n");

    //Get columns from source and target
    var res = snowflake.execute( {sqlText: "describe table "+TARGETTBL} );
    while(res.next()){
        tcols.add(res.getColumnValue(1));
    }
    var res = snowflake.execute( {sqlText: "describe table "+SOURCETBL} );
    while(res.next()){
        scols.add(res.getColumnValue(1));
    }

    var insert = "\t(";
    var upd_ls = "";
    var vals = "\tvalues ( \n";
    var titer = tcols.values();

    //Construct the list of values for insert
    //For the insert values, conditionally replace CURR_IND, DBT_UPDATED_AT,  DBT_VALID_FROM & DBT_VALID_TO
    for(let entry of titer){
        if(tcols.has(entry)){
            insert = insert.concat("\n\t" + entry +", ");
            if(entry == 'CURR_IND'){
                vals = vals.concat("\t 'Y', \n");
            } else if(entry == 'DBT_VALID_FROM'){
                 vals = vals.concat("\t current_timestamp(), \n");
            } else if(entry == 'DBT_VALID_TO'){
```

```
                vals = vals.concat("\t NULL, \n");
            } else if(entry == 'DBT_UPDATED_AT'){
                vals = vals.concat("\t current_timestamp(), \n");
            } else {
                vals = vals.concat("\t s." + entry + ", \n");
            }
        }
    }

    var merge = merge.concat("when not matched then insert\n");
    var insert = insert.substring(0, insert.length-2).concat(")\n");
    var vals = vals.substring(0, vals.length-3).concat(")\n");
    var merge = merge.concat(insert).concat(vals);

    return merge;
    $$
;
```

**SC2 - Stored Proc to run the merge statement**

```
create or replace procedure sp_merge_type2scd_run_sp(targetTbl string, sourceTbl string, naturalkeys string)
returns string
language javascript
comment = 'Merge with type-2 using natural keys'
execute as caller
as
$$
    //Process Source & Target keys from Input
    const tkeys = NATURALKEYS.split(',');
    const skeys = NATURALKEYS.split(',');
    const tcols = new Set();
    const scols = new Set();
    if(tkeys.length != skeys.length){
        return "ERROR: Source and Target must have the same number of entries"
    }

    // Set Join condition between source & target.
    // Variable will be reused throughout the procedure with different aliases
    var joincondition = "";
    for (let i=0;i<tkeys.length;i++){
         joincondition = joincondition.concat("tgt."+tkeys[i].trim()+" = src."+skeys[i].trim()+" and ");
    } ;
    joincondition = joincondition.substring(0, joincondition.length-4);

    //Begin Merge
    var merge = "merge into " + TARGETTBL + " t using ( \n"
    merge = merge.concat("with \n");

    //Constructing the CTEs so that the tables can be queried once
    merge = merge.concat("\tCTE_SRC_HASH AS ( select *, SHA1(TO_JSON(OBJECT_CONSTRUCT(*))) as ROW_HASH from " +
SOURCETBL + " ), \n");
    merge = merge.concat("\tCTE_TGT_CURR AS ( select " + NATURALKEYS + ", ROW_HASH from " + TARGETTBL + " where
CURR_IND = 'Y') \n");
    merge = merge.concat("\t-- Insert for new record \n");
    merge = merge.concat("\tselect NULL as MERGE_KEY \n");
    merge = merge.concat("\t ,S1.* \n");
    merge = merge.concat("\t from CTE_SRC_HASH S1 \n");
    merge = merge.concat("\t where not exists \n");
    merge = merge.concat("\t (select 1 from CTE_TGT_CURR T1 where " + joincondition.replace(/src/g,'S1').replace
(/tgt/g,'T1') +") \n");
    merge = merge.concat("\tunion all  \n");
    merge = merge.concat("\t-- Insert for updated record \n");
    merge = merge.concat("\tselect NULL as MERGE_KEY \n");
    merge = merge.concat("\t ,S2.* \n");
    merge = merge.concat("\t from CTE_SRC_HASH S2 inner join CTE_TGT_CURR T2 ON " + joincondition.replace(/src
/g,'S2').replace(/tgt/g,'T2')+ " \n");
    merge = merge.concat("\t and S2.ROW_HASH != T2.ROW_HASH \n");
    merge = merge.concat("\tunion all \n");
```

```
    merge = merge.concat("\t-- Update for updated record \n");
    merge = merge.concat("\tselect 'Y' as MERGE_KEY \n");
    merge = merge.concat("\t ,S3.* \n");
    merge = merge.concat("\t from CTE_SRC_HASH S3 inner join CTE_TGT_CURR T3 ON \n");
    merge = merge.concat("\t "+ joincondition.replace(/src/g,'S3').replace(/tgt/g,'T3')  + " \n");
    merge = merge.concat("\t and S3.ROW_HASH != T3.ROW_HASH \n");
    merge = merge.concat(") s \n");

    //Join the contructed view with the target table
    merge = merge.concat("on \n");
    merge = merge.concat( joincondition.replace(/src/g,'S').replace(/tgt/g,'T') + " and s.MERGE_KEY is not null
\n" );

    //Expire records for updated records
    merge = merge.concat("when matched then update set CURR_IND = 'N', DBT_UPDATED_AT =current_timestamp(),
DBT_VALID_TO =current_timestamp() \n");

    //Get columns from source and target
    var res = snowflake.execute( {sqlText: "describe table "+TARGETTBL} );
    while(res.next()){
        tcols.add(res.getColumnValue(1));
    }
    var res = snowflake.execute( {sqlText: "describe table "+SOURCETBL} );
    while(res.next()){
        scols.add(res.getColumnValue(1));
    }

    var insert = "\t(";
    var upd_ls = "";
    var vals = "\tvalues ( \n";
    var titer = tcols.values();

    //Construct the list of values for insert
    //For the insert values, conditionally replace CURR_IND, DBT_UPDATED_AT,  DBT_VALID_FROM & DBT_VALID_TO
    for(let entry of titer){
        if(tcols.has(entry)){
            insert = insert.concat("\n\t" + entry +", ");
            if(entry == 'CURR_IND'){
                vals = vals.concat("\t 'Y', \n");
            } else if(entry == 'DBT_VALID_FROM'){
                 vals = vals.concat("\t current_timestamp(), \n");
            } else if(entry == 'DBT_VALID_TO'){
                 vals = vals.concat("\t NULL, \n");
            } else if(entry == 'DBT_UPDATED_AT'){
                 vals = vals.concat("\t current_timestamp(), \n");
            } else {
                vals = vals.concat("\t s." + entry + ", \n");
            }
        }
    }

    var merge = merge.concat("when not matched then insert\n");
    var insert = insert.substring(0, insert.length-2).concat(")\n");
    var vals = vals.substring(0, vals.length-3).concat(")\n");
    var merge = merge.concat(insert).concat(vals);

    var res = snowflake.execute( {sqlText: merge} );
    res.next();
    var insertCount = res.getColumnValue(1);
    var updateCount = res.getColumnValue(2);
    return "Merge completed. " + insertCount + " rows inserted. " + updateCount + " rows updated.";
    $$
;
```

Control table to autogenerate SP and Tasks for automation:

Teams will be able to insert into a driving table the metadata information around the CDC process in order to autogenerate the code needed and the automation piece.

**Driving table DDL**

```
create or replace table CDC_DRIVE_TBL
(SOURCE_NAME VARCHAR,
 TARGET_NAME VARCHAR,
 NATURAL_KEYS VARCHAR,
 CDC_TYPE VARCHAR,
 TASK_NAME VARCHAR,
 SCHEDULING VARCHAR,
 WAREHOUSE VARCHAR
);
```

**Insert Statmenet Example**

```
insert into CDC_DRIVE_TBL
values ('HRDP_LND_DV_DB.PUBLIC.POC_DIM_LEARNER_STG_VW', 'HRDP_LND_DV_DB.PUBLIC.
POC_DIM_LEARNER_CD1','DLRN_PK_LEARNER_KEY_DLRN','CD1','HRDP_LND_DV_DB.PUBLIC.POC_DIM_LEARNER_CD1_TK','USING
CRON 0 0 8 * * UTC', 'HRDP_DBT_BASE_WH');

insert into CDC_DRIVE_TBL
values ('HRDP_LND_DV_DB.PUBLIC.POC_DIM_LEARNER_STG_VW', 'HRDP_LND_DV_DB.PUBLIC.
POC_DIM_LEARNER_CD2','DLRN_PK_LEARNER_KEY_DLRN','CD2','POC_DIM_LEARNER_CD1', 'USING CRON 0 0 8 * * UTC',
'HRDP_DBT_BASE_WH');
```

Once information is inserted into the table, a view can then generate the needed objects DDL:

**Driving View DDL**

```
CREATE OR REPLACE SECURE VIEW CDC_DRIVE_VW AS
select *,
'CALL '||case when CDC_TYPE='CD1' then 'sp_merge_type1scd_run_sp' when  CDC_TYPE='CD2' then
'sp_merge_type2scd_run_sp' else 'ERROR' end || '(\'' || TARGET_NAME || '\',\''||SOURCE_NAME||'\',
\''||NATURAL_KEYS||'\');' as SPROC_RUN,
'CALL '||case when CDC_TYPE='CD1' then 'sp_merge_type1scd_sql_sp' when  CDC_TYPE='CD2' then
'sp_merge_type2scd_sql_sp' else 'ERROR' end || '(\'' || TARGET_NAME || '\',\''||SOURCE_NAME||'\',
\''||NATURAL_KEYS||'\');' as SPROC_SQL,
'CREATE TASK '|| TASK_NAME||' WAREHOUSE = '||WAREHOUSE||case when SCHEDULING like '%USING CRON %' then '
SCHEDULE = \''||SCHEDULING||'\'' else ' AFTER '||SCHEDULING||'' end||' AS  '|| SPROC_RUN ||';' as  TASK_SQL
from CDC_DRIVE_TBL;
```

Create Stream on the View, in order to capture all the changes to the driving tables:

**Create Stream**

```
CREATE OR REPLACE STREAM CDC_DRIVE_STREAM ON VIEW CDC_DRIVE_VW;
```

Finally we need to Stored Procedure to use the stream on a reoccurring basis to enable the tasks in the driving table:

**Stored Procedure to use CDC Stream**

```
create or replace procedure TASK_CREATION()
returns string
language javascript
comment = 'USES STREAM TO CREATE TASKS FOR PIPELINE'
execute as caller
as
$$
   var create_temp_table = `create or replace temporary table temp_stream_cdc (name varchar, sql_command
varchar);`;
   snowflake.execute({sqlText: create_temp_table });
   var stream_driving_view =`insert into temp_stream_cdc select 'alter task '||task_name||' resume;', TASK_SQL
from CDC_DRIVE_STREAM`;
   snowflake.execute({sqlText: stream_driving_view });
   var temp_table_task =`select * from temp_stream_cdc`;
   var stream_resultset = snowflake.execute({sqlText: temp_table_task });
   while (stream_resultset.next())
   {
        snowflake.createStatement({sqlText: stream_resultset.getColumnValue(2)}).execute();
        snowflake.createStatement({sqlText: stream_resultset.getColumnValue(1)}).execute();
   }
   return 'SUCCESS'
$$
;
```

Finally the Stored Procedure can be ran or scheduled:

call TASK_CREATION();

or

**Create Task task**

```
CREATE TASK CDC_TASK

WAREHOUSE = <>

SCHEDULE =<>

AS  CALL TASK_CREATION{};
```