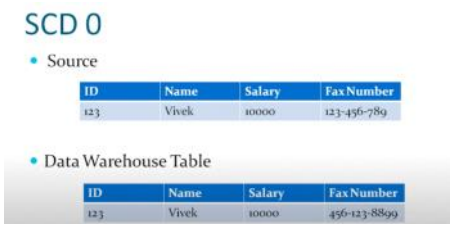
Slowly Changing Dimension (SCD)

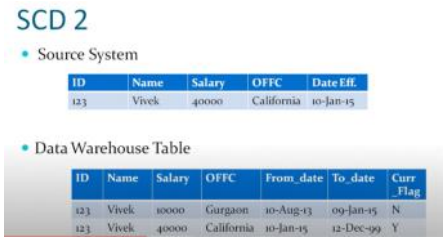
14 September 2022 10:52

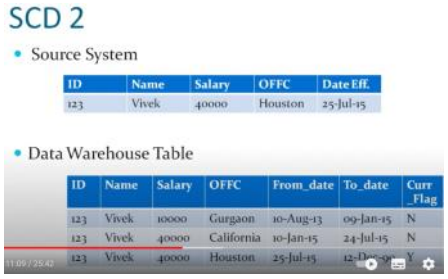
SDC0: Passive method (No Change)
Changed columns not updated in data warehouse table because the changed column(s) is not relevant anymore (i.e. FaxNumber)



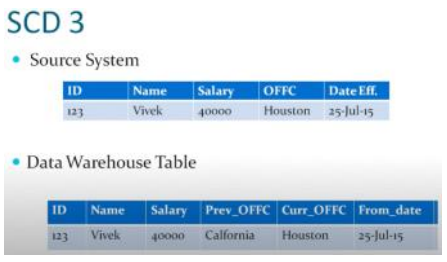
SCD1: Overwriting the old value (Latest Record Only)
Maintain the latest snapshot and no history (i.e. when the value is incorrect or when history does not matter)

SCD2: Creating a new additional record (Maintains History)
Every time when there is a change in the source system, an additional row is added to the warehouse table

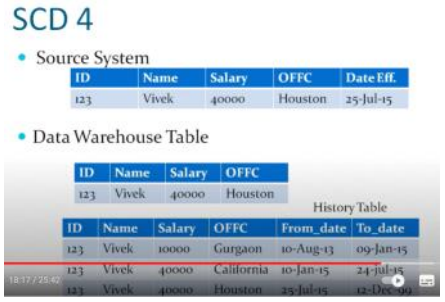




SCD3: Creating an additional Column (Rarely Use)
Every time when there is a change in the source system, an additional column is added to the warehouse table (NOT RECOMMENDED: you will need to add an additional column to every column to capture new changes)



SCD4: Using history table
Maintain both the latest snapshot and also history (i.e. keep separate tables -> a latest snapshot table and a history table)



SCD6: The hybrid approach
Add additional rows and columns

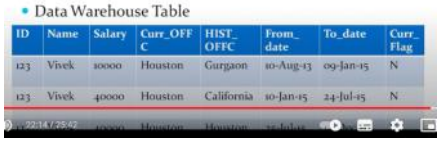
SCD 6(1+2+3)

- Source System

ID	Name	Salary	OFFC	Date Eff.
123	Vivek	40000	Houston	25-Jul-15

- Data Warehouse Table

ID	Name	Salary	Curr_OFFC	HIST_OFFC	From_date	To_date	Curr_Flag
123	Vivek	10000	Houston	Gurgaon	10-Aug-13	09-Jan-15	N
123	Vivek	40000	Houston	California	10-Jan-15	24-Jul-15	N



Curated: custom_scd2_rank (rank 1,2,3: each rank goes into the loop to create different adt_inserted_at timestamp)
Vault: vault_insert_rank (take each rank, insert into a temp table. For example, all rank 1 is loaded into the temp table, then all rank 2 is filtered and matched with the temp table with rank 1 on PK and hashdiff, so duplicates won't go in).

ADP SCD2 Example *Curated layer*:

1) Merge Into…When not matched then insert -> the purpose of this is to insert new data, records with not the same BK or the same last_modified_date). Please note that a common pattern is that you select a bunch of records from the table you want to make changes to, and provide conditions (e.g. See below) to update either the rows that match the condition.

   where CUR_REPERTOIRE.MDS.RECORDING_LOCAL.**RECORDING_LOCAL_ID** = s.**RECORDING_LOCAL_ID**
   and CUR_REPERTOIRE.MDS.RECORDING_LOCAL.**row_is_current** = 'Y'
   and CUR_REPERTOIRE.MDS.RECORDING_LOCAL.**LAST_MODIFIED_DATE** <> s.**LAST_MODIFIED_DATE**;

2) Left Join your incremental data (i.e. RAW_REPERTOIRE_LAKE.MDS.STR_RECORDING_LOCAL) to the existing table and say, when not matched insert all columns

3) Update…set -> This is for updating the old records. Think of this of updating the previous record in every loop/rank . For example, the for loop used is the following:

   IFF(b.max_rank = 1 and {{ iteration_number }} = 1, 1, {{ iteration_number + 1 }})) s

   Loop 1: IFF(b.max_rank = 1 and 1 = 1, 1, 2)) -> Assuming max_rank > 1, this will give you all records with rank = 2 in your loop 1 and you can use it to compare with the existing table and update only the records that have [existing table].LAST_MODIFIED_DATE <> [only rank= 2].LAST_MODIFIED_DATE which gives you all records with rank = 1

   Loop 2: IFF(b.max_rank = 1 and 2 = 1, 1, 3)) -> At this point, loop 2 should update record with rank = 2. This is achieved by [existing table].row_is_current = 'Y' because you will have:

   Column A | row_is_current | rank
   abs123 | N | 1
   abs123 | Y | 2 <---- (N-1) row needs updating in loop N
   abs123 | Y | 3

```
merge into CUR_REPERTOIRE.MDS.RECORDING_LOCAL t using (
    with source as (
select
  rank() over (partition by RECORDING_LOCAL_ID order by LAST_MODIFIED_DATE, adt_stage_file_name asc) rank_col,
  *
from
(
SELECT
    CURRENT_TIMESTAMP() AS ADT_INSERTED_AT,
    NULL AS ADT_UPDATED_AT,
    ADT_BATCH_DATE AS ADT_BATCH_DATE,
    ADT_STAGE_FILE_NAME AS ADT_STAGE_FILE_NAME,
    ADT_STAGE_FILE_ROW AS ADT_STAGE_FILE_ROW,
    ADT_METADATA AS ADT_METADATA,
    cast(FILECONTENTS:RECORDING_LOCAL_ID as NUMBER(38,0)) as RECORDING_LOCAL_ID,
    cast(FILECONTENTS:RECORDING_ID as NUMBER(38,0)) as RECORDING_ID,
    cast(FILECONTENTS:TERRITORY_ID as NUMBER(38,0)) as TERRITORY_ID,
    cast(FILECONTENTS:LANGUAGE_ID as NUMBER(38,0)) as LANGUAGE_ID,
    cast(FILECONTENTS:TITLE as VARCHAR(16777216)) as TITLE,
    cast(FILECONTENTS:ALTERNATIVE_TITLE as VARCHAR(16777216)) as ALTERNATIVE_TITLE,
    cast(FILECONTENTS:STANDARDISED_TITLE as VARCHAR(16777216)) as STANDARDISED_TITLE,
    cast(FILECONTENTS:STANDARDISED_ALT_TITLE as VARCHAR(16777216)) as STANDARDISED_ALT_TITLE,
    cast(FILECONTENTS:BAND_ARTIST_ID as NUMBER(38,0)) as BAND_ARTIST_ID,
    cast(FILECONTENTS:TITLE_MATCH_KEY as VARCHAR(16777216)) as TITLE_MATCH_KEY,
    cast(FILECONTENTS:SUBTITLE_MATCH_KEY as VARCHAR(16777216)) as SUBTITLE_MATCH_KEY,
    cast(FILECONTENTS:CREATED_DATE as TIMESTAMP_NTZ(9)) as CREATED_DATE,
    cast(FILECONTENTS:LAST_MODIFIED_DATE as TIMESTAMP_NTZ(9)) as LAST_MODIFIED_DATE,
    cast(FILECONTENTS:LOCAL_RECORDING_FLAG as VARCHAR(1)) as LOCAL_RECORDING_FLAG,
    cast(FILECONTENTS:DELETED_FLAG as VARCHAR(1)) as DELETED_FLAG,
    COALESCE(cast(FILECONTENTS:srcsys_is_deleted as VARCHAR(1)), 'N') as SRCSYS_IS_DELETED
FROM
    RAW_REPERTOIRE_LAKE.MDS.str_RECORDING_LOCAL) a)
        select
            i_s.*,
            i_s.RECORDING_LOCAL_ID as scd_bk,
            hash(ADT_INSERTED_AT, ADT_UPDATED_AT, ADT_BATCH_DATE, ADT_STAGE_FILE_NAME, ADT_STAGE_FILE_ROW, ADT_METADATA, RECORDING_ID, TERRITORY_ID, LANGUAGE_ID, TITLE, ALTERNATIVE_TITLE, STANDARDISED_TITLE, STANDARDISED_ALT_TITLE, BAND_ARTIST_ID, TITLE_MATCH_KEY, SUBTITLE_MATCH_KEY,
LOCAL_RECORDING_FLAG, DELETED_FLAG, SRCSYS_IS_DELETED) as tracking_hash,
            iff(i_t.RECORDING_LOCAL_ID is null, i_s.CREATED_DATE, i_s.LAST_MODIFIED_DATE) as row_start_at,
            '9999-12-31 00:00:00' as row_end_at,
            'Y' as row_is_current
        from source i_s
        left outer join (select distinct RECORDING_LOCAL_ID from CUR_REPERTOIRE.MDS.RECORDING_LOCAL) i_t -- distinct not ideal for performance, added in for handling test data
            on i_s.RECORDING_LOCAL_ID = i_t.RECORDING_LOCAL_ID
        where "RANK_COL" = 1) s
    on s.RECORDING_LOCAL_ID = t.RECORDING_LOCAL_ID
    and s.LAST_MODIFIED_DATE = t.LAST_MODIFIED_DATE
    -- and t.row_is_current = 'Y'

when not matched
then insert(ADT_INSERTED_AT, ADT_UPDATED_AT, ADT_BATCH_DATE, ADT_STAGE_FILE_NAME, ADT_STAGE_FILE_ROW, ADT_METADATA, RECORDING_ID, TERRITORY_ID, LANGUAGE_ID, TITLE, ALTERNATIVE_TITLE, STANDARDISED_TITLE, STANDARDISED_ALT_TITLE, BAND_ARTIST_ID, TITLE_MATCH_KEY, SUBTITLE_MATCH_KEY,
LOCAL_RECORDING_FLAG, DELETED_FLAG, SRCSYS_IS_DELETED, ROW_START_AT, ROW_END_AT, ROW_IS_CURRENT, TRACKING_HASH, LAST_MODIFIED_DATE, RECORDING_LOCAL_ID, CREATED_DATE)
values (s.ADT_INSERTED_AT , s.ADT_UPDATED_AT , s.ADT_BATCH_DATE , s.ADT_STAGE_FILE_NAME , s.ADT_STAGE_FILE_ROW , s.ADT_METADATA , s.RECORDING_ID , s.TERRITORY_ID , s.LANGUAGE_ID , s.TITLE , s.ALTERNATIVE_TITLE , s.STANDARDISED_TITLE , s.STANDARDISED_ALT_TITLE , s.BAND_ARTIST_ID ,
s.TITLE_MATCH_KEY , s.SUBTITLE_MATCH_KEY , s.LOCAL_RECORDING_FLAG , s.DELETED_FLAG , s.SRCSYS_IS_DELETED , s.ROW_START_AT , s.ROW_END_AT , s.ROW_IS_CURRENT , s.TRACKING_HASH , s.LAST_MODIFIED_DATE , s.RECORDING_LOCAL_ID , s.CREATED_DATE );
```

```sql
update CUR_REPERTOIRE.MDS.RECORDING_LOCAL
set row_end_at = dateadd(ms,-1,s.LAST_MODIFIED_DATE),
    adt_updated_at = current_timestamp(),
    row_is_current = 'N'
from
    (with source as (
select
  rank() over (partition by RECORDING_LOCAL_ID order by LAST_MODIFIED_DATE, adt_stage_file_name asc) rank_col,
  *
from
(
SELECT
    CURRENT_TIMESTAMP() AS ADT_INSERTED_AT,
    NULL AS ADT_UPDATED_AT,
    ADT_BATCH_DATE AS ADT_BATCH_DATE,
    ADT_STAGE_FILE_NAME AS ADT_STAGE_FILE_NAME,
    ADT_STAGE_FILE_ROW AS ADT_STAGE_FILE_ROW,
    ADT_METADATA AS ADT_METADATA,
    cast(FILECONTENTS:RECORDING_LOCAL_ID as NUMBER(38,0)) as RECORDING_LOCAL_ID,
    cast(FILECONTENTS:RECORDING_ID as NUMBER(38,0)) as RECORDING_ID,
    cast(FILECONTENTS:TERRITORY_ID as NUMBER(38,0)) as TERRITORY_ID,
    cast(FILECONTENTS:LANGUAGE_ID as NUMBER(38,0)) as LANGUAGE_ID,
    cast(FILECONTENTS:TITLE as VARCHAR(16777216)) as TITLE,
    cast(FILECONTENTS:ALTERNATIVE_TITLE as VARCHAR(16777216)) as ALTERNATIVE_TITLE,
    cast(FILECONTENTS:STANDARDISED_TITLE as VARCHAR(16777216)) as STANDARDISED_TITLE,
    cast(FILECONTENTS:STANDARDISED_ALT_TITLE as VARCHAR(16777216)) as STANDARDISED_ALT_TITLE,
    cast(FILECONTENTS:BAND_ARTIST_ID as NUMBER(38,0)) as BAND_ARTIST_ID,
    cast(FILECONTENTS:TITLE_MATCH_KEY as VARCHAR(16777216)) as TITLE_MATCH_KEY,
    cast(FILECONTENTS:SUBTITLE_MATCH_KEY as VARCHAR(16777216)) as SUBTITLE_MATCH_KEY,
    cast(FILECONTENTS:CREATED_DATE as TIMESTAMP_NTZ(9)) as CREATED_DATE,
    cast(FILECONTENTS:LAST_MODIFIED_DATE as TIMESTAMP_NTZ(9)) as LAST_MODIFIED_DATE,
    cast(FILECONTENTS:LOCAL_RECORDING_FLAG as VARCHAR(1)) as LOCAL_RECORDING_FLAG,
    cast(FILECONTENTS:DELETED_FLAG as VARCHAR(1)) as DELETED_FLAG,
    COALESCE(cast(FILECONTENTS:srcsys_is_deleted as VARCHAR(1)), 'N') as SRCSYS_IS_DELETED
FROM
    RAW_REPERTOIRE_LAKE.MDS.str_RECORDING_LOCAL) a)
        select
            a.RECORDING_LOCAL_ID,
            a.LAST_MODIFIED_DATE
        from source a
        join (select RECORDING_LOCAL_ID, max("RANK_COL") as max_rank from source group by RECORDING_LOCAL_ID) b
            on a.RECORDING_LOCAL_ID = b.RECORDING_LOCAL_ID
        where a."RANK_COL" = IFF(b.max_rank = 1 and 1 = 1, 1, 2)) s

where CUR_REPERTOIRE.MDS.RECORDING_LOCAL.RECORDING_LOCAL_ID = s.RECORDING_LOCAL_ID
and CUR_REPERTOIRE.MDS.RECORDING_LOCAL.row_is_current = 'Y'
and CUR_REPERTOIRE.MDS.RECORDING_LOCAL.LAST_MODIFIED_DATE <> s.LAST_MODIFIED_DATE;
```