SCDs for Invoices

In your dim\_claim\_invoice table, Cim\_invoice\_id is identified as the **Natural Key** or **Business Key**. Here's an explanation of its role:

**Natural Key (Business Key) Overview:**

* A **Natural Key**, also known as a **Business Key**, is a column or set of columns that represents a real-world, business-defined identifier for a record. It is typically used to uniquely identify an entity or object in the operational (source) system.

**Role of Cim\_invoice\_id:**

* In your dim\_claim\_invoice table, the Cim\_invoice\_id represents the actual invoice identifier that comes from the source system (e.g., an ERP or invoicing system). This value is typically stable and consistent because it is defined by the business and not by the database structure.
* Student will be having student\_id
* Customer will be having cust\_id
* Citizen will be having SSN, Aadhar

**Key Points About Cim\_invoice\_id:**

1. **Unique in the Source System**: The Cim\_invoice\_id is unique in the source operational system and remains constant across different versions of the same invoice. It's the real-world identifier for an invoice.
2. **Not Affected by SCD Versioning**: Even when you implement Slowly Changing Dimensions (SCD) Type 2, the Cim\_invoice\_id does not change between records; what changes are other attributes of the invoice (like Cim\_invoice\_number) that are tracked over time in your dim\_claim\_invoice table.
3. **Tracking History**: Since SCD Type 2 maintains multiple versions of the same invoice to track changes over time, the Cim\_invoice\_id helps you to link back to the original real-world entity across all historical and current records. You might have multiple rows in your table for the same Cim\_invoice\_id (representing different versions of the invoice), but each will have different surrogate keys, effective dates, and version numbers.
4. **Business Logic Dependence**: Cim\_invoice\_id is important for business operations and logic, like joining with other tables (fact tables, for example) in your data warehouse that use this identifier.

**Example Use Case:**

If an invoice originally had an ID INV123, the Cim\_invoice\_id would always be INV123 regardless of how many changes occur to the other details (like invoice number, amounts, or dates). In the background, the system tracks different versions of this invoice using surrogate keys and other SCD columns.

In contrast, the **Surrogate Key** (like Cim\_invoice\_key) is a system-generated identifier that uniquely distinguishes each row (each version) in your dimension table.

**Why Use Both?**

* **Surrogate Key** (Cim\_invoice\_key): This is for tracking the versions of the same record.
* **Id, 1,2,3,4,5 – in Bigquery – There is no concept like surrogate key - UUID**
* **Natural Key** (Cim\_invoice\_id): This represents the real-world identifier of the invoice that business users are familiar with.

This approach allows you to implement SCD Type 2 while still maintaining a connection to the real-world entity through the natural/business key (Cim\_invoice\_id).

To implement your dim\_claim\_invoice dimension table in BigQuery, you typically need two types of tables:

1. **Staging Table**: This is where raw data is loaded, transformed, and cleaned before being inserted into the final table.
2. **Final Dimension Table**: The cleaned and transformed data will be stored here. This table will be the one used in reports or other processes.

**1. Staging Table (stg\_claim\_invoice)**

This table will hold the raw data from source systems. You can load data here from various sources such as CSV, JSON, or other transactional systems. A common practice is to load incremental changes or deltas, including a metadata column to track the insert/update timestamps.

**Staging Table Structure:** source.

CREATE OR REPLACE TABLE woven-name-434311-i8.claims\_datalake.stg\_claim\_invoice (

Clm\_invoice\_id STRING,

Clm\_invoice\_date DATE,

Clm\_invoice\_number STRING,

Source\_timestamp TIMESTAMP -- Timestamp from the source system (e.g., created\_at or updated\_at)

);

**2. Final Dimension Table (dim\_claim\_invoice)**

This table will hold the cleaned and transformed data, ready to be used in analytics queries. You will have a surrogate key as the primary key, along with the actual invoice details. You also track insert/update timestamps and flags such as CurrentFlag to handle slowly changing dimensions (SCD Type 2).

**Final Dimension Table Structure:**

CREATE OR REPLACE TABLE woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice (

Clm\_invoice\_key STRING, -- Surrogate key (generated automatically)

Clm\_invoice\_id STRING, -- Natural key

Clm\_invoice\_date DATE, -- Invoice date

Clm\_invoice\_number STRING, -- Invoice number

Inserted\_timestamp TIMESTAMP, -- Time when the record was inserted

Updated\_timestamp TIMESTAMP, -- Time when the record was updated (if applicable)

CurrentFlag BOOLEAN, -- Indicates whether this is the current record (SCD Type 2)

);

### ****Loading into Staging Table****

Load the extracted data directly into the staging table in BigQuery. Here’s how the load might look if you’re using a tool or script:

INSERT INTO woven-name-434311-i8.claims\_datalake.stg\_claim\_invoice (

Clm\_invoice\_id,

Clm\_invoice\_date,

Clm\_invoice\_number,

Source\_timestamp

)

VALUES

('INV-001', '2024-10-01', '12345', '2024-10-02 14:55:00 UTC'),

('INV-002', '2024-10-02', '67890', '2024-10-03 09:12:00 UTC');

1. **ETL (Extract, Transform, Load) Logic**

To load data from the staging table to the final dimension table, you would typically implement a MERGE statement to handle inserts and updates. Here's an example: table:

MERGE INTO woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice AS target

USING (

SELECT \*

FROM woven-name-434311-i8.claims\_datalake.stg\_claim\_invoice

WHERE Source\_timestamp >= TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 1 DAY) -- Use the source timestamp for filtering

) AS source

ON target.Clm\_invoice\_id = source.Clm\_invoice\_id

WHEN MATCHED AND target.Clm\_invoice\_number != source.Clm\_invoice\_number THEN

UPDATE SET

target.Clm\_invoice\_number = source.Clm\_invoice\_number,

target.Clm\_invoice\_date = source.Clm\_invoice\_date,

target.Updated\_timestamp = source.Source\_timestamp, -- Use source timestamp

target.CurrentFlag = FALSE -- Mark the previous record as inactive

WHEN NOT MATCHED THEN

INSERT (

Clm\_invoice\_key, Clm\_invoice\_id, Clm\_invoice\_date,

Clm\_invoice\_number, Inserted\_timestamp, CurrentFlag

)

VALUES (

GENERATE\_UUID(), -- Generate unique surrogate key

source.Clm\_invoice\_id,

source.Clm\_invoice\_date,

source.Clm\_invoice\_number,

source.Source\_timestamp, -- Use source timestamp

TRUE

);

INSERT INTO woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice (

Clm\_invoice\_key,

Clm\_invoice\_id,

Clm\_invoice\_date,

Clm\_invoice\_number,

Inserted\_timestamp,

CurrentFlag

)

SELECT

GENERATE\_UUID(), -- Generate new UUID as the primary key

source.Clm\_invoice\_id,

CAST(source.Clm\_invoice\_date AS DATE), -- Ensure Clm\_invoice\_date is cast to DATE type

CAST(source.Clm\_invoice\_number AS STRING), -- Ensure Clm\_invoice\_number is cast to STRING

CURRENT\_TIMESTAMP(), -- Use current timestamp for the inserted time

TRUE -- Set the CurrentFlag to TRUE

FROM (

SELECT

\*,

ROW\_NUMBER() OVER (PARTITION BY Clm\_invoice\_id ORDER BY CURRENT\_TIMESTAMP()) AS rn -- Assign row numbers for each invoice\_id

FROM

woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice

WHERE

CurrentFlag = FALSE

) AS target

JOIN

woven-name-434311-i8.claims\_datalake.stg\_claim\_invoice AS source

ON

target.Clm\_invoice\_id = source.Clm\_invoice\_id

WHERE

target.rn = 1 -- Select only the first unique record for each Clm\_invoice\_id

AND CAST(target.Clm\_invoice\_number AS STRING) != CAST(source.Clm\_invoice\_number AS STRING);

### 5. ****Incremental Load Based on**** Source\_timestamp

To manage incremental loads (especially if the source data is large), you can filter the data based on the Source\_timestamp. This will allow you to load only the records that have changed since the last ETL run.

You can keep track of the last Source\_timestamp processed and use it in your next ETL batch

**Summary**

* **Staging Table**: Instead of generating a Load\_timestamp during the load, you are pulling a Source\_timestamp from the source server.
* **ETL Process**: Use this Source\_timestamp in the MERGE query to update or insert records into the final dimension table.
* **Incremental Loads**: Filter source data based on Source\_timestamp to ensure you're processing only new or updated records.

TEST CASES

To ensure that the MERGE query works correctly, you can define several test cases based on the two inserted records in your staging table (stg\_claim\_invoice) and the current state of the final dimension table (dim\_claim\_invoice). Each test case will validate different scenarios, such as updates, inserts, and no-op operations.

Here are **three test cases** to test the MERGE query functionality.

**1. Test Case 1: Insert New Records**

**Description:**

* Insert a new record in the staging table (stg\_claim\_invoice) with no matching Clm\_invoice\_id in the final dimension table (dim\_claim\_invoice).
* The expected behavior is that the new record is inserted into the final table.

**Staging Table (New Record Example):**

INSERT INTO woven-name-434311-i8.claims\_datalake.stg\_claim\_invoice (

Clm\_invoice\_id, Clm\_invoice\_date, Clm\_invoice\_number, Source\_timestamp

) VALUES (

'INV-003', '2024-10-03', '54321', '2024-10-03 10:00:00 UTC'

);

**Expected Behavior:**

* The new invoice record (INV-003) should be inserted into the dim\_claim\_invoice table with a new surrogate key, Inserted\_timestamp, and CurrentFlag = TRUE.

**Post-MERGE Check:**

SELECT \*

FROM woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice

WHERE Clm\_invoice\_id = 'INV-003';

* Ensure that the record for INV-003 exists in the dim\_claim\_invoice table and CurrentFlag = TRUE.

**2. Test Case 2: Update Existing Record**

**Description:**

* Insert a record into the staging table (stg\_claim\_invoice) with an Clm\_invoice\_id that already exists in the final table but with a different Clm\_invoice\_number.
* The expected behavior is that the existing record in the final table should be updated with the new Clm\_invoice\_number, and the Updated\_timestamp should reflect the Source\_timestamp.

**Staging Table (Update Existing Record Example):**

INSERT INTO woven-name-434311-i8.claims\_datalake.stg\_claim\_invoice (

Clm\_invoice\_id, Clm\_invoice\_date, Clm\_invoice\_number, Source\_timestamp

) VALUES (

'INV-001', '2024-10-01', '99999', '2024-10-04 10:30:00 UTC' -- Clm\_invoice\_number updated

);

**Expected Behavior:**

* The invoice record (INV-001) should be updated in the dim\_claim\_invoice table:
  + Clm\_invoice\_number should be updated to '99999'.
  + Updated\_timestamp should be updated to '2024-10-04 10:30:00 UTC'.
  + The CurrentFlag of the existing record should be set to FALSE, and a new version should be inserted with CurrentFlag = TRUE.

**Post-MERGE Check:**

SELECT \*

FROM woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice

WHERE Clm\_invoice\_id = 'INV-001';

* Ensure that there are two records for INV-001:
  + The old record has CurrentFlag = FALSE.
  + The new record has Clm\_invoice\_number = '99999', CurrentFlag = TRUE, and Updated\_timestamp is set to the Source\_timestamp.

**3. Test Case 3: No Changes (No-Op)**

**Description:**

* Insert a record into the staging table (stg\_claim\_invoice) that already exists in the final dimension table with the same Clm\_invoice\_id and Clm\_invoice\_number.
* The expected behavior is that no changes should be made to the final table (no updates or inserts).

**Staging Table (No-Op Example):**

INSERT INTO woven-name-434311-i8.claims\_datalake.stg\_claim\_invoice (

Clm\_invoice\_id, Clm\_invoice\_date, Clm\_invoice\_number, Source\_timestamp

) VALUES (

'INV-002', '2024-10-02', '67890', '2024-10-04 11:00:00 UTC' -- Same data as in dim\_claim\_invoice

);

**Expected Behavior:**

* The record for INV-002 in the dim\_claim\_invoice table should not be modified since the Clm\_invoice\_number is the same as the current one.
* No new rows should be inserted, and the Updated\_timestamp should remain unchanged.

**Post-MERGE Check:**

SELECT \*

FROM woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice

WHERE Clm\_invoice\_id = 'INV-002';

* Verify that no new rows are inserted for INV-002 and the existing row has not been updated.

**General Validation Queries:**

After each test case, you can run the following queries to check the state of the data:

1. **Check for duplicates**:

SELECT Clm\_invoice\_id, COUNT(\*)

FROM woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice

GROUP BY Clm\_invoice\_id

HAVING COUNT(\*) > 1;

1. **Check if CurrentFlag is correctly set**:

SELECT Clm\_invoice\_id, CurrentFlag, COUNT(\*)

FROM woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice

GROUP BY Clm\_invoice\_id, CurrentFlag

ORDER BY Clm\_invoice\_id;

1. **Check for latest updates**:

SELECT \*

FROM woven-name-434311-i8.claims\_datalake.dim\_claim\_invoice

ORDER BY Updated\_timestamp DESC

LIMIT 5;

These test cases ensure that the MERGE operation handles inserts, updates, and no-ops correctly. You can validate the results using the post-MERGE queries provided.