Travel Data Pipeline | Complete & Refine Solution

Naam: Ajimi Mohamed
Datum: 17/05/2024

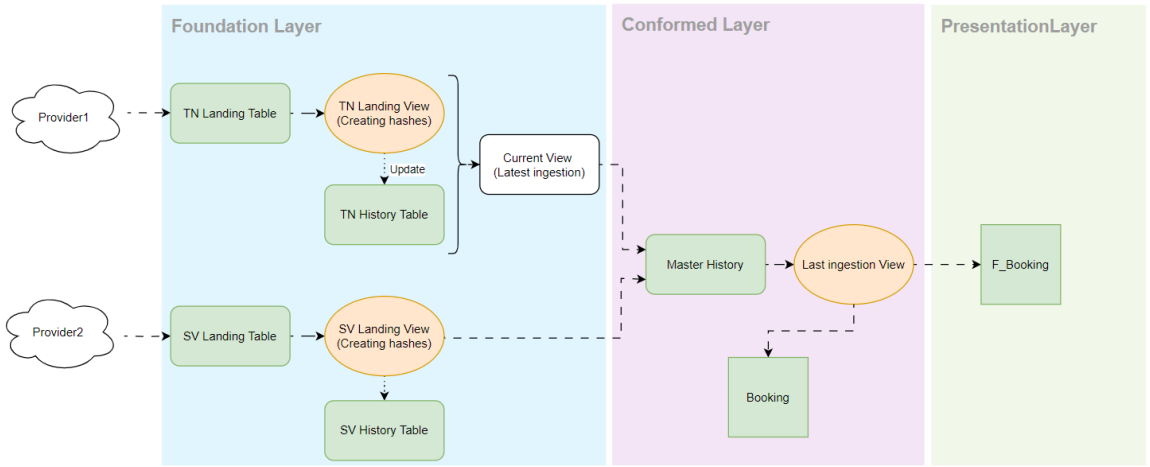
1. Inhoud

2. Overview

This documentation outlines the data ingestion and processing workflow for a client system that integrates data from two distinct providers.

3. Data Flow and Processing

1.



3.1. Foundation Layer

1. Provider1:

Each weekly dataset includes:

- **New Bookings:** All new bookings that have been made since the last dataset.
- **Updated Bookings:** Updated versions of any bookings that have changed since their last inclusion in a dataset.
- **Unchanged Bookings:** Crucially, the data set also includes all previous bookings that have not changed, continuing to provide the latest version of these bookings as they were last reported.

This means that every week, the dataset from Provider1 is a complete snapshot of all active bookings (both new and unchanged) up to that point.

Example Scenario:

- **Week 1**: Receives 3 bookings (A, B, C).

- **Week 2**: Receives updates for bookings A and B, and 2 new bookings D and E. The dataset includes the latest versions of A and B, the unchanged booking C, and the new bookings D and E.
- **Week 3**: Booking B is updated again; bookings A, C, D, and E remain unchanged. The dataset this week includes the latest version of B, alongside the unchanged latest versions of A, C, D, and E.

2. Provider2:

Each dataset includes:

- **New and Recently Updated Bookings Only:** Unlike Provider1, Provider2's dataset includes only those bookings that are either newly created or have been updated since the last dataset. It does not include unchanged bookings from one week to the next unless they are newly updated or added.
- **Snapshot of Changes:** Each week's data from Provider2 is effectively a snapshot of new or recently altered bookings, without carrying over data for unchanged bookings from previous weeks.

Example Scenario:

- **Week 1**: Sends data for 4 new bookings (F, G, H, I).
- **Week 2**: Books G and H are updated, and 1 new booking J is added. The dataset includes only the updated versions of G and H and the new booking J.

3. Pipeline

For Provider1 :

Row Hashing and Version Tracking:

The introduction of `row_hash`, `ingestion_date`, `isCurrent` and `new_row_hash` serves to meticulously track changes across the data received each week. This ensures that even if a booking reverts to a previous state, the system can recognize and correctly handle such occurrences.

How isCurrent Works:

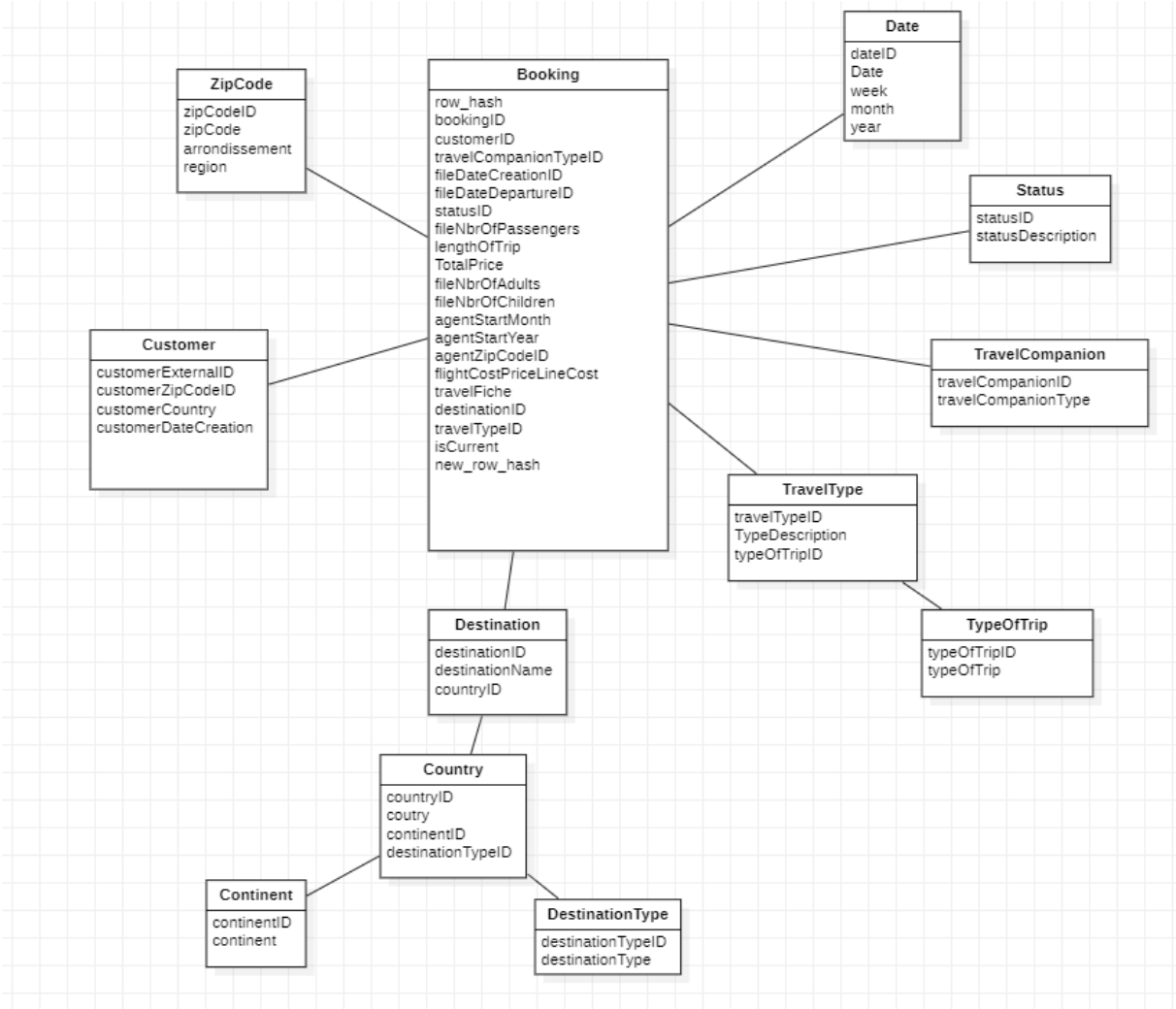
- **Tracking Latest Versions**: When new data is ingested each week, every record in the landing view or landing table is initially considered as potentially the latest. As the new data is processed, the system needs to determine which records represent the most current state of each booking.
- **Update Mechanism**: During the data ingestion and update process in (`Provider1-booking-landing-View`):

1. **Identifying Changes**: The system compares the `row_hash` of the newly ingested data against the existing records in the history table. It looks specifically at records where `isCurrent` is true, which represent the latest versions of bookings.
2. **Updating Flags**: If a booking from the new dataset matches an existing booking by ID but has a different `row_hash` (indicating a change), the system will:
    - Set the `isCurrent` flag to TRUE for the new record, indicating that it is now the latest version.
    - Reset the `isCurrent` flag to FALSE for the previous record in the history table, indicating that it is no longer the latest version.
3. **Handling Unchanged Data**: For bookings that continue to appear week after week without changes, their `row_hash` will match, and they will retain their `isCurrent` status as TRUE in the database. This ensures that the system continuously acknowledges them as the current, valid entries.

For Provider2:

Simpler Integration Strategy: Since Provider2 only provides data on new or recently updated bookings, the approach here is more straightforward. The same columns (`row_hash`, `ingestion_date`, `new_row_hash`) are added to track changes efficiently but without the complexity of managing unchanged historical bookings.

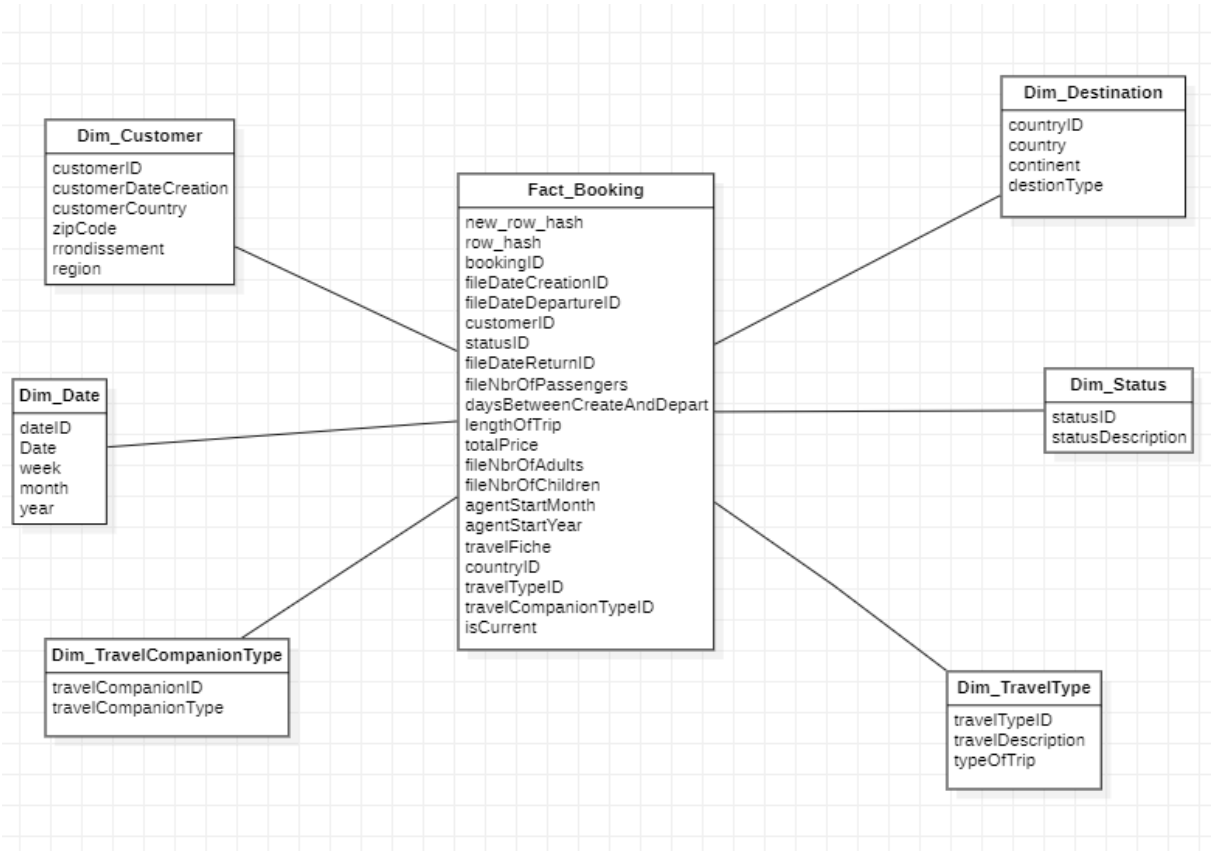
3.2. Conformed Layer :



Conflict Resolution: When identical booking IDs appear from both providers within a single ingestion cycle, possibly due to bookings being present in both systems, the system employs a rule-based mechanism to decide which booking to retain. The selection criterion is based on the highest `TotalPrice`.
Implementation of B2B and B2C Filters: Planned for future integration, this filter will further refine the selection process, prioritizing bookings based on business logic and customer segments.

Master History Table: This is where data from both providers is merged into a single source of truth. Here, essential attributes like `Id`, `ingestion_date`, and the `source` of the data (Provider1 or Provider2) are consolidated. This table is pivotal for ensuring data consistency across the system.
Data from the `Master History Table` is then used to populate the `Booking` table in the Conformed Layer.

3.3. Presentation Layer :



F_Booking Table: This final table is designed for external reporting and analytics.

Additional calculations are done in the View Current-Booking-View:

Current-Booking-View QUERY SHARE

SCHEMA DETAILS LINEAGE DATA PROFILE DATA QU

Filter Enter property name or value

Field name	Type	Mode	Key
YearCreation	INTEGER	NULLABLE	-
MonthCreation	INTEGER	NULLABLE	-
WeekCreation	INTEGER	NULLABLE	-
YearDeparture	INTEGER	NULLABLE	-
MonthDeparture	INTEGER	NULLABLE	-
WeekDeparture	INTEGER	NULLABLE	-
YTDNew	STRING	NULLABLE	-
Destination	STRING	NULLABLE	-
Continent	STRING	NULLABLE	-
DestType	STRING	NULLABLE	-
LengthOfTrip	INTEGER	NULLABLE	-
TravelCompanionType	STRING	NULLABLE	-
DaysBetweenCreateAndDepart	INTEGER	NULLABLE	-
BookingWindow	STRING	NULLABLE	-
TravelFiche	STRING	NULLABLE	-
FileNbrOfPassengers	INTEGER	NULLABLE	-
TotalPrice	INTEGER	NULLABLE	-
FileNbrOfAdults	INTEGER	NULLABLE	-
FileNbrOfChildren	INTEGER	NULLABLE	-

4. Procedures and Views

4.1. Master Procedure Overview:

This query is responsible for the automation of the pipeline. Once the data is in the landing tables, we execute this query.

```
BEGIN
  CALL `btc-dev-414511.sandbox_test.sp_insert_travelnote_booking_current`();
  CALL `btc-dev-414511.ConformedLayer_btc.sp_insert_master_history`();
  CALL `btc-dev-414511.ConformedLayer_btc.sp_update_insert_booking`();
  CALL `btc-dev-414511.PresentationLayer_btc.sp_update_insert_f_booking`();
  CALL `btc-dev-414511.sandbox_test.sp_update_travelnote_booking_history`();
END;
```

4.2. Explanation:

- **Landing View**:

Query

```
SELECT *,
  CAST(FARM_FINGERPRINT(TO_JSON_STRING(t)) AS STRING) AS row_hash,
  CURRENT_DATE() AS ingestion_date,  -- or use CURRENT_TIMESTAMP() for date and time
  CAST(FARM_FINGERPRINT(CONCAT(TO_JSON_STRING(t), CAST(CURRENT_DATE() AS STRING))) AS STRING) AS new_row_hash
FROM `btc-dev-414511.sandbox_test.Servico-booking-landing`  t
```

  - **Purpose**: The landing view prepares incoming raw data by adding necessary metadata like hash values , ingestion dates and isCurrent(for Provider1), which help in identifying new and changed records.

- **sp_insert_provider1_booking_current**:

```
BEGIN
  BEGIN
    INSERT INTO `btc-dev-414511.sandbox_test.TravelNote-booking-Current`
    SELECT DISTINCT
      lt.*
    FROM `btc-dev-414511.sandbox_test.TravelNote-booking-landing-View` lt
    LEFT JOIN `btc-dev-414511.sandbox_test.TravelNote-booking-history` bh
      ON lt.Id = bh.Id AND bh.isCurrent = TRUE
    WHERE bh.row_hash <> lt.row_hash OR bh.row_hash IS NULL;
  EXCEPTION WHEN ERROR THEN
    INSERT INTO `btc-dev-414511.ConformedLayer_btc.error_log` (error_timestamp, procedure_name, error_message)
    VALUES (CURRENT_TIMESTAMP(), 'sp_insert_travelnote_booking_current', ERROR_MESSAGE());
  END;
END
```

  - **Purpose**: This procedure captures new and updated bookings from Provider1. It checks the current data against previously stored data to identify and insert only the latest changes or new entries and stores the changes in “Current” table that we will use as a base to merge Provider1 and Provider2 data (could be changed to a View instead of a table).

- **sp_insert_master_history:**

```
BEGIN
  BEGIN
    INSERT INTO `btc-dev-414511.ConformedLayer_btc.master-history`
    SELECT
      new_row_hash,
      row_hash,
      Id,
      ingestion_date,
      source
    FROM (
      SELECT *,
        ROW_NUMBER() OVER (PARTITION BY Id ORDER BY TotalPrice DESC) AS global_rn
      FROM (
        SELECT
          new_row_hash,
          row_hash,
          Id,
          TotalPrice,
          CURRENT_DATE() AS ingestion_date,
          'TravelNote' AS source
        FROM (
          SELECT *,
            ROW_NUMBER() OVER (PARTITION BY Id ORDER BY TotalPrice DESC) AS rn
          FROM `btc-dev-414511.sandbox_test.TravelNote-booking-Current`
        )
        WHERE rn = 1

        UNION ALL

        SELECT
          new_row_hash,
          row_hash,
          Id,
          TotalPrice,
          CURRENT_DATE() AS ingestion_date,
          'Servico' AS source
        FROM (
          SELECT *,
            ROW_NUMBER() OVER (PARTITION BY Id ORDER BY TotalPrice DESC) AS rn
          FROM `btc-dev-414511.sandbox_test.Servico-booking-landing-View`
        )
        WHERE rn = 1
      ) combined
    )
    WHERE global_rn = 1;
  EXCEPTION WHEN ERROR THEN
    INSERT INTO `btc-dev-414511.ConformedLayer_btc.error_log` (error_timestamp, procedure_name, error_message)
    VALUES (CURRENT_TIMESTAMP(), 'sp_insert_master_history', ERROR_MESSAGE());
  END;
END
```

  - **Purpose**: This procedure captures new and updated bookings from Provider1. It checks the current data against previously stored data to identify and insert only the latest changes or new entries and stores the changes in “Current” table that we will use as a base to merge Provider1 and Provider2 data (could be changed to a View instead of a table).

- **Master View:**

```sql
SELECT
  main.new_row_hash,
  main.row_hash,
  main.Id AS bookingID,
  main.ingestion_date,
  main.source,
  COALESCE(tn.AgentZipCode, sv.AgentZipCode) AS AgentZipCode,
  CAST(COALESCE(tn.FileDateCreation, sv.FileDateCreation) AS DATE) AS FileDateCreation,
  SAFE.DATE(SAFE.PARSE_TIMESTAMP('%Y-%m-%d %H:%M:%E*S', COALESCE(tn.FileDateDeparture, sv.FileDateDeparture))) AS FileDateDeparture,
  SAFE.DATE(SAFE.PARSE_TIMESTAMP('%Y-%m-%d %H:%M:%E*S', COALESCE(tn.FileDateReturn, sv.FileDateReturn))) AS FileDateReturn,
  COALESCE(tn.TravelTypeDescription, sv.TravelTypeDescription) AS TravelTypeDescription,
  COALESCE(tn.StatusDescription, sv.StatusDescription) AS StatusDescription,
  SAFE_CAST(COALESCE(tn.fileNbrOfPassengers, sv.fileNbrOfPassengers) AS INTEGER) AS fileNbrOfPassengers,
  COALESCE(tn.Destination, sv.Destination) AS Destination,
  COALESCE(tn.TotalPrice, sv.TotalPrice) AS TotalPrice,
  COALESCE(tn.CustomerZipcode, sv.CustomerZipcode) AS CustomerZipcode,
  SAFE.DATE(SAFE.PARSE_TIMESTAMP('%Y-%m-%d %H:%M:%E*S', COALESCE(tn.CustomerDateCreation, sv.CustomerDateCreation))) AS CustomerDateCreation,
  COALESCE(tn.CustomerExternalId, sv.CustomerExternalId) AS CustomerExternalId,
  COALESCE(tn.AgentStartMonth, sv.AgentStartMonth) AS AgentStartMonth,
  COALESCE(tn.AgentStartYear, sv.AgentStartYear) AS AgentStartYear,
  SAFE_CAST(COALESCE(tn.LengthOfTrip, sv.LengthOfTrip) AS INTEGER) AS LengthOfTrip,
  COALESCE(tn.Adults, sv.Adults) AS fileNbrOfAdults,
  COALESCE(tn.Children, sv.Children) AS fileNbrOfChildren,
  COALESCE(tn.DestinationType, sv.DestinationType) AS DestinationType,
  COALESCE(tn.TravelFiche, sv.TravelFiche) AS TravelFiche,
  COALESCE(tn.TravelCompanionType, sv.TravelCompanionType) AS TravelCompanionType
FROM (
    SELECT *,
           MAX(ingestion_date) OVER () AS max_ingestion_date
    FROM `btc-dev-414511.ConformedLayer_btc.master-history`
) main
LEFT JOIN `btc-dev-414511.sandbox_test.TravelNote-booking-Current` tn
  ON main.new_row_hash = tn.new_row_hash
LEFT JOIN `btc-dev-414511.sandbox_test.Servico-booking-landing-View` sv
  ON main.new_row_hash = sv.new_row_hash
WHERE main.ingestion_date = main.max_ingestion_date;
```

  - **Purpose**: This view combines and harmonizes data from both Provider1 and Provider2, providing a unified view of all bookings. We also perform data transformation and type checks.

- **sp_update_insert_booking:**

```
CREATE OR REPLACE PROCEDURE `btc-dev-414511.ConformedLayer_btc.sp_update_insert_booking`()
BEGIN
  BEGIN
    UPDATE `btc-dev-414511.ConformedLayer_btc.Booking`
    SET isCurrent = FALSE
    WHERE bookingID IN (
        SELECT BookingID FROM `btc-dev-414511.ConformedLayer_btc.Current-master-View`
    ) AND isCurrent = TRUE;

    INSERT INTO `btc-dev-414511.ConformedLayer_btc.Booking`
    (
        row_hash,
        bookingID,
        travelCompanionTypeID,
        fileDateCreationID,
        fileDateDepartureID,
        fileNbrOfPassengers,
        statusID,
        lengthOfTrip,
        totalPrice,
        fileNbrOfAdults,
        fileNbrOfChildren,
        agentZipCodeID,
        agentStartMonth,
        agentStartYear,
        travelFiche,
        destinationID,
        travelTypeID,
        isCurrent,
        new_row_hash
    )
    SELECT
        mtv.row_hash,
        mtv.BookingID,
        tc.TravelCompanionID,
        da_creation.DateID AS fileDateCreationID,
        da_departure.DateID AS fileDateDepartureID,
        mtv.fileNbrOfPassengers,
        s.StatusID,
        mtv.lengthOfTrip,
        mtv.totalPrice,
        mtv.fileNbrOfAdults,
        mtv.fileNbrOfChildren,
        z.zipCodeID,
        mtv.agentStartMonth,
        mtv.agentStartYear,
        mtv.travelFiche,
        d.destinationID,
        tt.typeOfTripID,
        TRUE AS isCurrent,
        mtv.new_row_hash
    FROM `btc-dev-414511.ConformedLayer_btc.Current-master-View` mtv
    LEFT JOIN `btc-dev-414511.ConformedLayer_btc.TravelCompanion` tc
        ON mtv.travelCompanionType = tc.TravelCompanionType
    LEFT JOIN `btc-dev-414511.ConformedLayer_btc.TypeOfTrip` tt
        ON mtv.TravelTypeDescription = tt.typeOfTrip
    LEFT JOIN `btc-dev-414511.ConformedLayer_btc.Status` s
        ON mtv.StatusDescription = s.StatusDescription
    LEFT JOIN `btc-dev-414511.ConformedLayer_btc.Destination` d
        ON mtv.Destination = d.DestinationName
    LEFT JOIN `btc-dev-414511.ConformedLayer_btc.ZipCode` z
        ON mtv.agentZipCode = z.zipCode
    LEFT JOIN `btc-dev-414511.ConformedLayer_btc.Date` da_creation
        ON mtv.fileDateCreation = da_creation.CalendarDate
    LEFT JOIN `btc-dev-414511.ConformedLayer_btc.Date` da_departure
        ON mtv.fileDateDeparture = da_departure.CalendarDate;
  EXCEPTION WHEN ERROR THEN
    INSERT INTO `btc-dev-414511.ConformedLayer_btc.error_log` (error_timestamp, procedure_name, error_message)
    VALUES (CURRENT_TIMESTAMP(), 'sp_update_insert_booking', ERROR_MESSAGE());
  END;
END;
```

  - **Purpose**: This procedure updates the conformed layer with the master data that has been cleaned and consolidated in previous steps. It ensures that the booking records are not only current but also accurately reflect the most recent information across all sources, supporting reliable operational decisions. It also tracks the latest version of a booking using isCurrent.

- **sp_update_insert_f_booking:**

```
BEGIN
  BEGIN
    UPDATE `btc-dev-414511.PresentationLayer_btc.F_Booking`
    SET isCurrent = FALSE
    WHERE bookingID IN (
        SELECT BookingID FROM `btc-dev-414511.ConformedLayer_btc.Current-master-View`
    ) AND isCurrent = TRUE;

    INSERT INTO `btc-dev-414511.PresentationLayer_btc.F_Booking`
    (
        row_hash,
        bookingID,
        travelCompanionTypeID,
        fileDateCreationID,
        fileDateDepartureID,
        fileNbrOfPassengers,
        statusID,
        lengthOfTrip,
        totalPrice,
        fileNbrOfAdults,
        fileNbrOfChildren,
        agentStartMonth,
        agentStartYear,
        travelFiche,
        countryID,
        travelTypeID,
        isCurrent,
        new_row_hash
    )
    SELECT
        s.row_hash,
        s.bookingID,
        s.travelCompanionTypeID,
        s.fileDateCreationID,
        s.fileDateDepartureID,
        s.fileNbrOfPassengers,
        s.statusID,
        s.lengthOfTrip,
        s.totalPrice,
        s.fileNbrOfAdults,
        s.fileNbrOfChildren,
        s.agentStartMonth,
        s.agentStartYear,
        s.travelFiche,
        d.countryID,
        s.travelTypeID,
        TRUE AS isCurrent,
        s.new_row_hash
    FROM `btc-dev-414511.ConformedLayer_btc.Booking-View` s
    JOIN `btc-dev-414511.ConformedLayer_btc.Current-master-View` b
        ON s.new_row_hash = b.new_row_hash
    JOIN `btc-dev-414511.ConformedLayer_btc.Destination` d
        ON s.destinationID = d.destinationID;
  EXCEPTION WHEN ERROR THEN
    INSERT INTO `btc-dev-414511.ConformedLayer_btc.error_log` (error_timestamp, procedure_name, error_message)
    VALUES (CURRENT_TIMESTAMP(), 'sp_update_insert_f_booking', ERROR_MESSAGE());
  END;
END
```

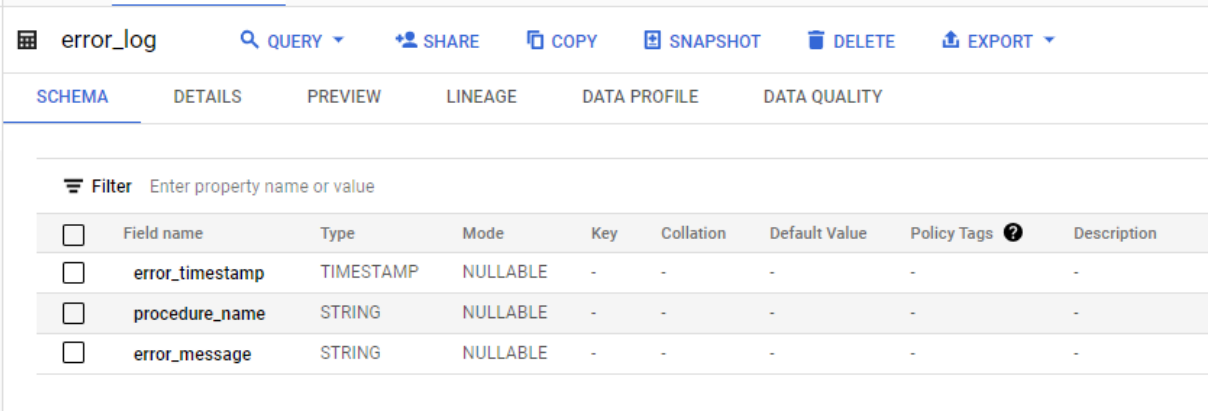
  - **Purpose**: Updates the presentation layer.

- **sp_update_provider1_booking_history**:

```
BEGIN
  BEGIN
    -- Update isCurrent flag to FALSE in TravelNote-booking-history
    UPDATE `btc-dev-414511.sandbox_test.TravelNote-booking-history`
    SET isCurrent = FALSE
    WHERE Id IN (SELECT Id FROM `btc-dev-414511.sandbox_test.TravelNote-booking-Current`)
      AND isCurrent = TRUE;

    -- Insert new records into TravelNote-booking-history with isCurrent set to TRUE
    INSERT INTO `btc-dev-414511.sandbox_test.TravelNote-booking-history`
    SELECT *,
           TRUE AS isCurrent
    FROM `btc-dev-414511.sandbox_test.TravelNote-booking-Current` t;

    -- Delete all records from TravelNote-booking-landing
    DELETE FROM `btc-dev-414511.sandbox_test.TravelNote-booking-landing` WHERE TRUE;

    -- Delete all records from TravelNote-booking-Current
    DELETE FROM `btc-dev-414511.sandbox_test.TravelNote-booking-Current` WHERE TRUE;
  EXCEPTION WHEN ERROR THEN
    -- Log error in error_log table
    INSERT INTO `btc-dev-414511.ConformedLayer_btc.error_log` (error_timestamp, procedure_name, error_message)
    VALUES (CURRENT_TIMESTAMP(), 'sp_update_travelnote_booking_history', ERROR_MESSAGE());
  END;
END
```

  - **Purpose**: This final step updates the historical records for Provider1 bookings. It marks previous versions of bookings as outdated and inserts the latest versions as current.
    It also deletes the Landing and Provider1 Current table for future ingestions.

5. Error Logging:



5.1. What is Logged:

- **Error Details**: This includes the timestamp of when the error occurred, the name of the procedure where the error was detected, and the error message describing the nature of the issue.

5.2. Specific Table and Dataset:

- **Table Name**: `error_log`
- **Dataset Location**: located in the `ErrorLog` dataset within the project.
- **Database Schema**: The `error_log` table contains the following columns:
  - **`error_timestamp`**: Records the exact time when the error occurred.
  - **`procedure_name`**: Identifies which procedure encountered the error.
  - **`error_message`**: Stores a descriptive message about the error, providing insight into what went wrong.