**Sarah Ahmed, Sara El-sefy, Samar Ali and Ahmed Kobap**

**Client Innovation Center (CIC)**

IBM Egypt

**Version 1**

Nov 26th, 2020

**Data Integration Maintenance**

IBM Infosphere Datastage

Table of Contents

[Table of Contents 2](#_Toc58850056)

[1. Introduction 3](#_Toc58850057)

[2. Audit Maintenance solution 3](#_Toc58850058)

[2.1 Online Integration cycle 4](#_Toc58850059)

[2.1.1 Description 4](#_Toc58850060)

[2.1.2 Challenges 6](#_Toc58850061)

[2.2 Audit Maintenance sequence 6](#_Toc58850062)

[2.2.1 Audit Maintenance Part 1 6](#_Toc58850063)

[2.2.2 Audit Maintenance Part 2 14](#_Toc58850064)

[2.3 Reusable Artifacts 23](#_Toc58850065)

[2.3.1 IBM Datastage Sequence 23](#_Toc58850066)

[2.3.2 Stored Procedures 23](#_Toc58850067)

[3. Audit Maintenance solution 23](#_Toc58850068)

[3.1. Modification type maintenance solution description 24](#_Toc58850069)

[A. Unexpected modification types 24](#_Toc58850070)

[B. Duplicate modification type for the same batch 24](#_Toc58850071)

[C. Invalid modification type order 24](#_Toc58850072)

[Stored procedure snapshot 25](#_Toc58850073)

[3.2. Modification type maintenance solution running 28](#_Toc58850074)

[3.3. TELECOM\_CONSOLIDATED Example 28](#_Toc58850075)

[3.3.1. Calling the stored procedure 28](#_Toc58850076)

[3.3.2. Detecting modification type issues 28](#_Toc58850077)

[3.3.3. Logging modification type issues 29](#_Toc58850078)

[3.4. Reusable Artifacts: 29](#_Toc58850079)

# Introduction

Combining data residing in different sources into a single system or platform could be challenging due to several reasons such as the overwhelming number of sources, the vast amount of data. Possible inconsistencies and redundancies could occur and should be resolved so that the warehouse is able to provide an integrated and reconciled view of data.

This document discusses two different solutions developed that help ensure having complete and accurate data in the data warehouse. The first solution is the Audit Maintenance solution that discusses a use-case of data integration cycle that extracts data from Microsoft SQL Server DB, which gathers data from different governmental sources, then loads it into another two different Databases DB2 DB and Teradata DB. The solution aims to ensure an accurate transfer of data from source to destination. The second solution is Modification type maintenance solution which is mainly about maintaining historical data in data warehouse which is critical task that needs to be handled securely to assure data credibility and audit-ability. The two solutions will be discussed below in detail.

# Audit Maintenance solution

Data integration is one of the most important aspects of a Data Warehouse. Ensuring an error-free data transfer to destination is a crucial step for any integration cycle which is the main objective of the audit maintenance solution.

## Online Integration cycle

### Description

The on-line integration cycle starts with running ETL jobs that extracts batches of data from different governmental authorities and loading it into Microsoft internal zone tables.

Each batch run represents extracting data from different tables of a source system represented by a unique id.

Once data is loaded in MIC tables, MICRSOFT ETL\_LOG Table is updated. This table's main purpose is to keep track of the loaded data. Each record in the table is identified by a unique 'runid' representing meta-data of the table loaded in Microsoft such as the number of records inserted, the source table name, Microsoft target table name, the source system database name, the batch id ('ExecutionGUID') and several other attributes. Moreover, it updates the status column to ‘Success’ indicating that the data was successfully loaded in MIC. If there is a case of a job failure, this means that the data wasn’t moved successfully from the source system to MIC, so the status would be updated to ‘Failed’.

**Microsoft ETL\_LOG**

For example, MICRSOFT ETL\_LOG table below shows that 3329195 records were transferred from Pension\_Expense table from InsurancePensionG database and were successfully loaded in Pension\_Expense table in MIC.

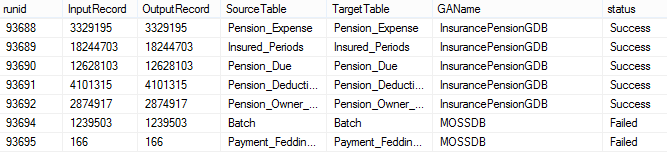


TABLE 1 MICROSOFT ETL\_LOG

**DB2 ETL\_LOG**

Afterwards, successful batches loaded in Microsoft should be transferred to IBM DB2 and then to TERA DATA for the online integration cycle to be completed. So, once the data is extracted from Microsoft and loaded successfully in IBM DB2 Model, DB2 IBM AUDIT Table is updated. This table is an extension to MICRSOFT ETL\_LOG Table as shown below. The same records in MICRSOFT ETL\_LOG are moved to DB2 IBM AUDIT Table, adding to it the number of records that were successfully loaded in DB2 model, target table name in DB2 as well as SQL to DB2 status column which is set to ‘Success’ indicating that the data was successfully transferred from MIC to DB2.

As shown below, DB2 IBM AUDIT table indicates that the 3329195 records were successfully transferred from Pension\_Expense table in MIC to Pension\_Expense table in DB2.

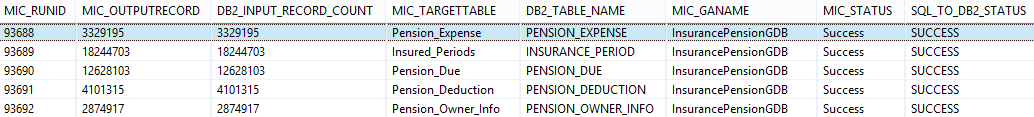


TABLE 2 DB2 IBM AUDIT

Next, data loaded in DB2 successfully should be transferred to TERADATA Model. After the data is loaded successfully in TERADATA, ‘DB2 to Tera status’ column should then be updated to ‘Success’ in DB2 IBM AUDIT Table as shown below.

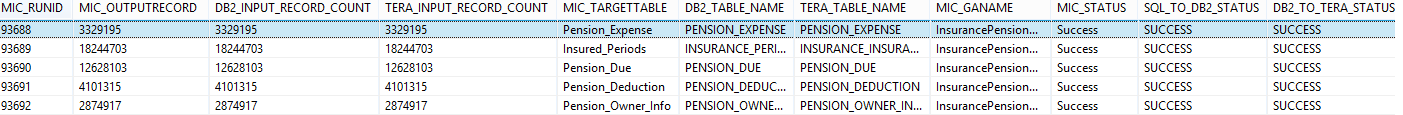


TABLE 3 DB2 IBM AUDIT

**Teradata ETL\_LOG**

So now that the data has successfully been transferred from MICROSOFT to TERADARA, the cycle is closed by copying the logs as is from IBM AUDIT TABLE in **DB2** to IBM AUDIT TABLE in **TERADATA.**

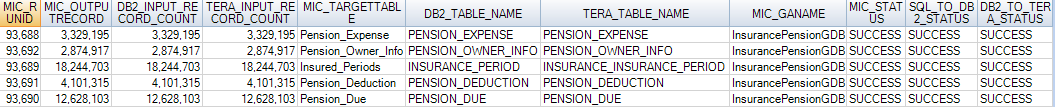


TABLE 4 TERA IBM AUDIT

### Challenges

In summary, ‘ETL\_LOG’ in MICRSOFT, ‘IBM AUDIT’ in DB2 and ‘IBM AUDIT’ in TERADATA main objective is to track data transfer throughout the entire online integration cycle starting from main source till it reaches Teradata successfully. Accordingly, all three tables should be in sync. All successful batches in Microsoft should exist in DB2 IBM AUDIT and in IBM AUDIT in TERADATA. But due to jobs failures and other problems, several issues arise during data transfer between the 3 tables causing un-synced tables. Some of these issues were having records transferred to IBM AUDIT TABLE but their status is ‘Failed’ in Microsoft and shouldn’t have been transferred, also finding records in TERA AUDIT TABLE that do not exist in DB2 IBM Audit and vice versa. Referential integrity problems also occurred like having records in TERA AUDIT that do not exist in the model while others exist in the model but were not logged in the audit table, and many other problems. Hence, an automated solution was needed to detect these integration issues to be able troubleshoot them which is the main objective of the Audit Maintenance sequence. It ensures an error free integration between MIC, IBM and Teradata.

## Audit Maintenance sequence

Audit maintenance sequence consists of 2 main parts explained below in further detail.

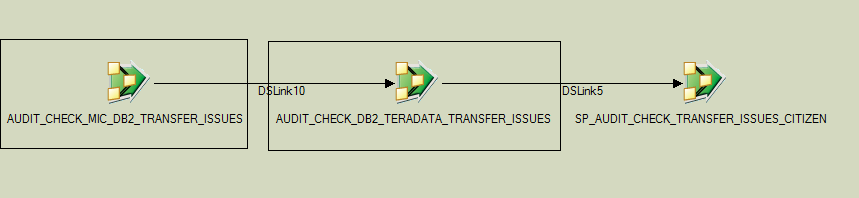
### Audit Maintenance Part 1

The first part first compares MICRSOFT ETL\_LOG, IBM DB2 AUDIT, then it compares between IBM DB2 AUDIT TERA AUDIT tables. It checks that there are no discrepancies between the three tables. Any Issue detected is stored in INTEGRATION.AUDIT\_MAINTENANCE table with the below schema:

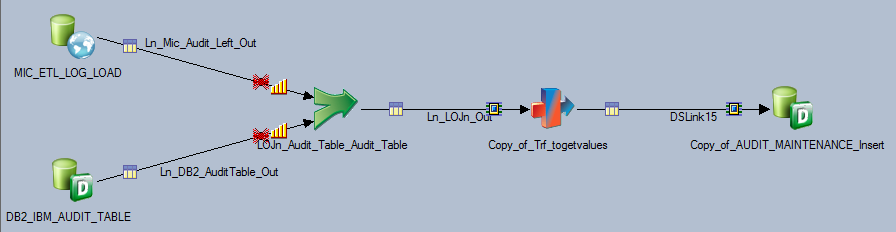
|  |  |  |
| --- | --- | --- |
| **Column Name** | **Column Type** | **Description** |
| BATCH\_ID | INTEGER | BATCH\_ID |
| LOAD\_ID | VARGRAPHIC(60) | LOAD\_ID |
| RECORD\_STATUS | VARGRAPHIC(100) | RECORD\_STATUS |
| DESCRIPTION | VARGRAPHIC(300) | DESCRIPTION |
| MIC\_GANAME | VARGRAPHIC(200) | MIC\_GANAME |
| MIC\_TARGETTABLE | VARGRAPHIC(200) | MIC\_TARGETTABLE |
| DB2\_TABLE\_NAME | VARGRAPHIC(200) | DB2\_TABLE\_NAME |
| TERA\_TABLE\_NAME | VARGRAPHIC(200) | TERA\_TABLE\_NAME |
| TABLE\_ID | INTEGER | TABLE\_ID |
| RESOLVED\_FLAG | INTEGER | RESOLVED\_FLAG |
| SOURCE\_FLAG | VARGRAPHIC(5) | SOURCE\_FLAG |
| ENTRY\_DATE | TIMESTAMP | ENTRY\_DATE |

The first part of the sequence is composed of 2 Jobs which are:

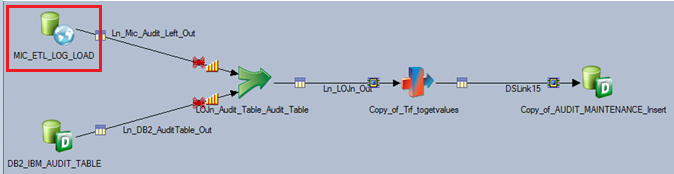
* AUDIT\_CHECK\_MIC\_DB2\_TRANSFER\_ISSUES.
* AUDIT\_CHECK\_DB2\_TERADATA\_TRANSFER\_ISSUES.

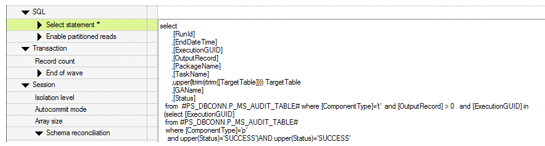


1. **AUDIT\_CHECK\_MIC\_DB2\_TRANSFER\_ISSUES Job**

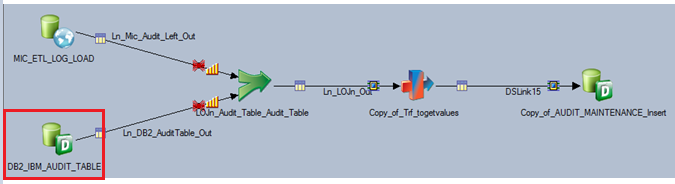


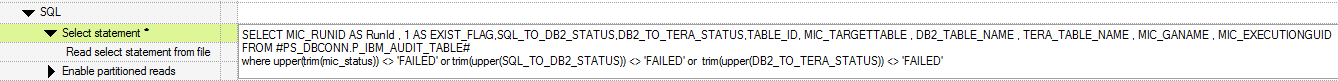
This job checks that all batches in MICRSOFT ETL\_LOG have successfully been moved to DB2 IBM AUDIT, it detects any transfer issues between the 2 tables and stores the result in AUDIT\_MAINTENANCE table in DB2. The job is described below in more detail.



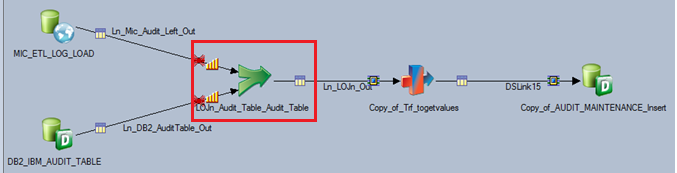


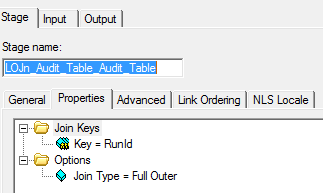
The first job selects from MIC ETL\_LOG table all successful batches that were loaded in MIC Tables which should exist in DB2 IBM\_AUDIT\_TABLE.



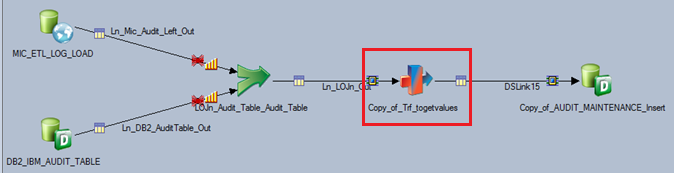


These jobs select batches in DB2 IBM\_AUDIT\_TABLE having successful SQL\_TO\_DB2\_STATUS indicating that these batches have been successfully moved from MIC to DB2.



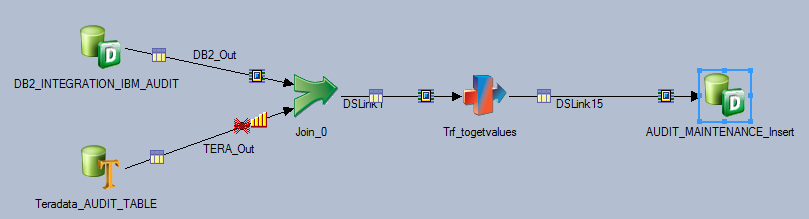


A full outer join on ‘RunId’ is done between successful batches selected from MIC ETL\_LOG table and from DB2 IBM\_AUDIT\_TABLE to point out any discrepancies between the 2 tables. All selected batches from MIC ETL\_LOG table should exist in DB2 IBM\_AUDIT\_TABLE.

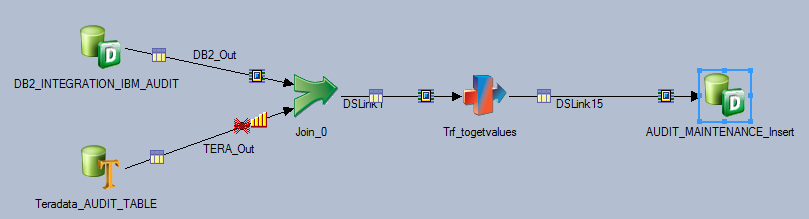


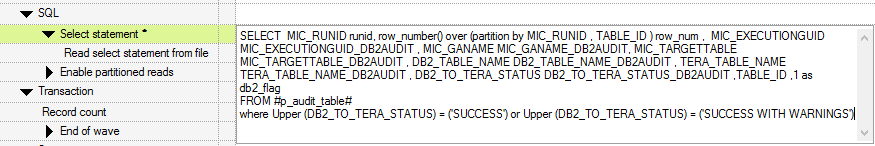
The transformer stage checks if there are run ids exist in Microsoft and not in db2 and vice versa. And finally, the result is stored in AUDIT\_MAINTENANCE table.

1. AUDIT\_CHECK\_ DB2\_TERADATA \_TRANSFER\_ISSUES Job

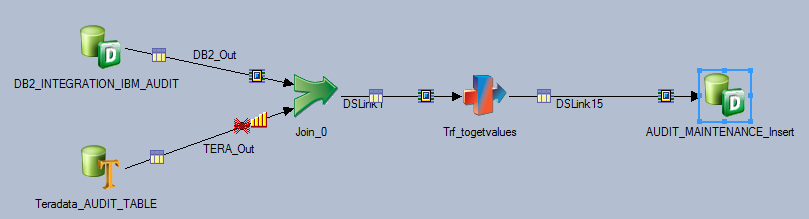


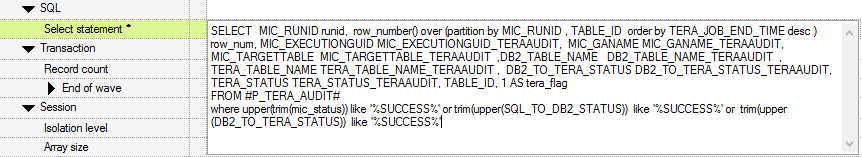
This job checks that all batches in DB2 IBM AUDIT have successfully been moved to TERA AUDIT, detects any transfer issues between the 2 tables and stores the result in AUDIT\_MAINTENANCE table in DB2. The job is described below in more details.



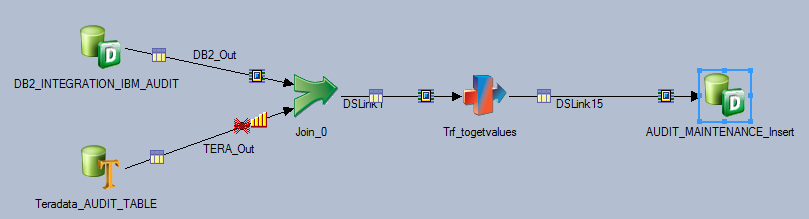


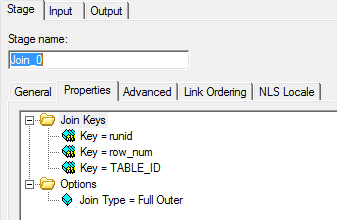
This Job first selects the batches from DB2 IBM AUDIT TABLE with DB2\_TO\_TERA\_STATUS = 'SUCCESS' or 'SUCCESS WITH WARNINGS', which represents all the batches that should be existing in TERA AUDIT TABLE. A row number is added to the result set partitioned by RUNID.



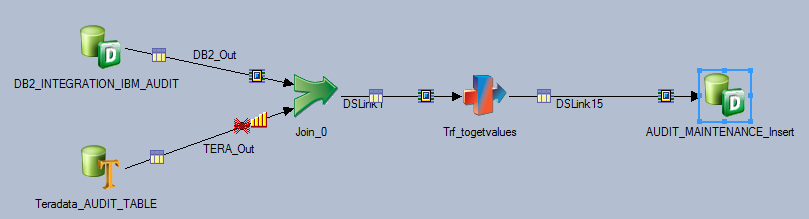


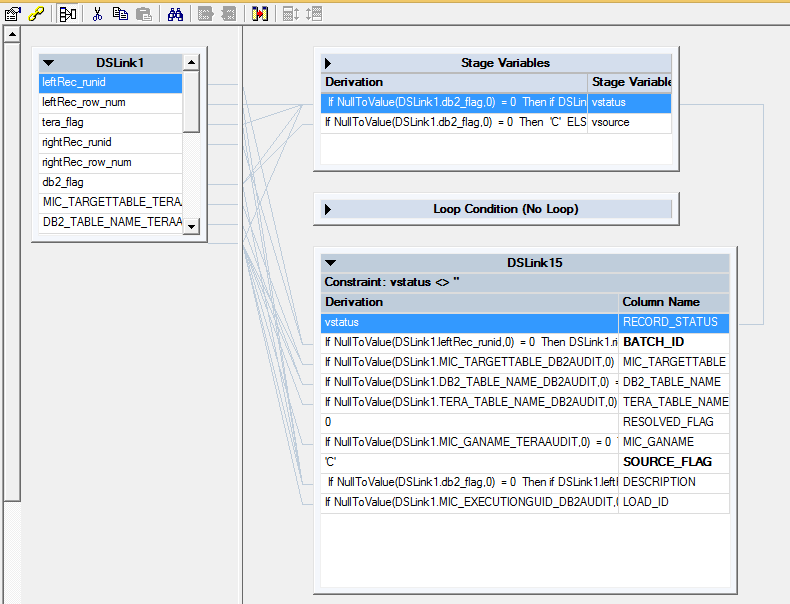
This stage selects all successful batches from TERA AUDIT TABLE with row number partitioned by RUNID.



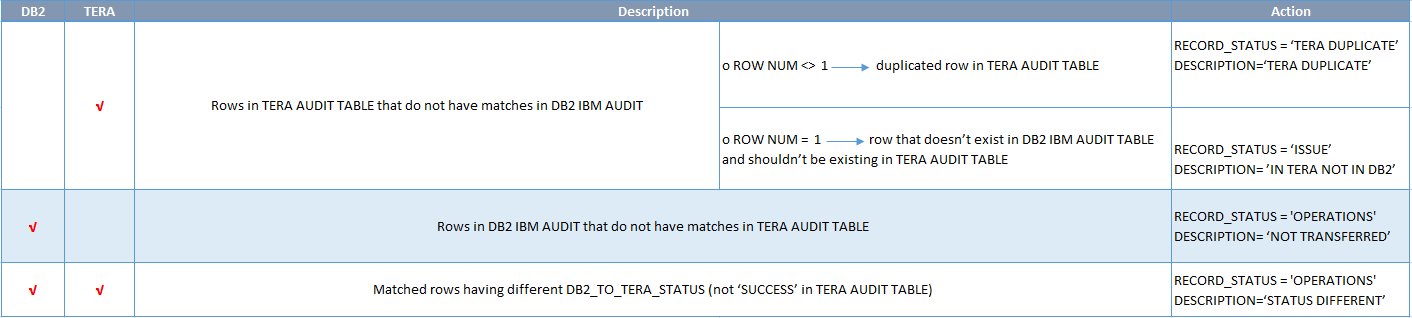


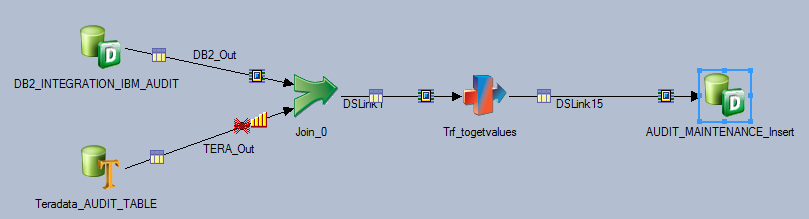
Then a Full Outer Join is done with TERA AUDIT TABLE on RUNID and ROW NUM and TABLE ID.





The result of the FULL JOIN points out different cases illustrated below, based on which the values for the columns ‘RECORD\_STATUS’ and ‘DESCRIPTION’ are identified: -





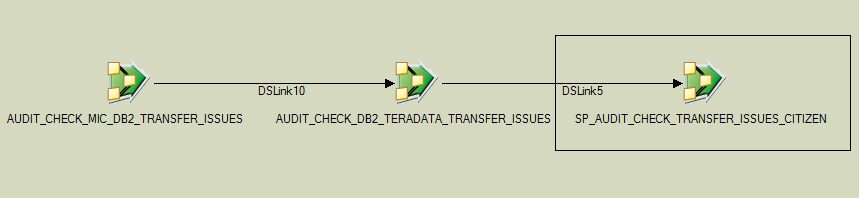
The output is inserted in INTEGRATION.AUDIT\_MAINTENANCE table in DB2 with structure described below: -

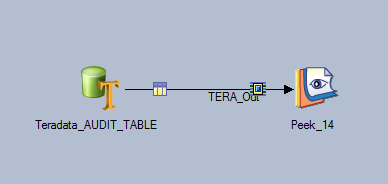
|  |  |  |
| --- | --- | --- |
|  | **Column Type** | **Description** |
| BATCH\_ID | INTEGER | BATCH\_ID |
| LOAD\_ID | VARGRAPHIC(60) | LOAD\_ID |
| \*\*RECORD\_STATUS | VARGRAPHIC(100) | RECORD\_STATUS |
| \*\*DESCRIPTION | VARGRAPHIC(300) | DESCRIPTION |
| MIC\_GANAME | VARGRAPHIC(200) | MIC\_GANAME |
| MIC\_TARGETTABLE | VARGRAPHIC(200) | MIC\_TARGETTABLE |
| DB2\_TABLE\_NAME | VARGRAPHIC(200) | DB2\_TABLE\_NAME |
| TERA\_TABLE\_NAME | VARGRAPHIC(200) | TERA\_TABLE\_NAME |
| TABLE\_ID | INTEGER | TABLE\_ID |
| RESOLVED\_FLAG | INTEGER | RESOLVED\_FLAG |
| SOURCE\_FLAG | VARGRAPHIC(5) | SOURCE\_FLAG |
| ENTRY\_DATE | TIMESTAMP | ENTRY\_DATE |

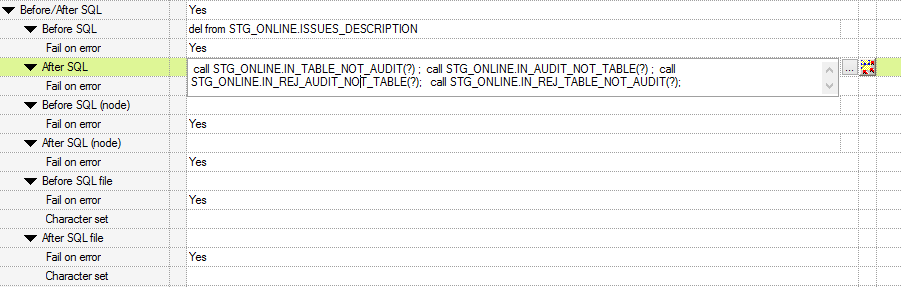
### Audit Maintenance Part 2

The second part ensures that all data logged in IBM\_AUDIT\_TABLE and IBM\_REJECTION\_AUDIT table in Teradata exists in the Model and vice versa.

‘SP\_AUDIT\_CHECK\_TRANSFER\_ISSUES’ runs 4 stored procedures.







The 4 procedures are: -

**IN\_TABLE\_NOT\_AUDIT**

**IN\_REJ\_TABLE\_NOT\_TABLE**

**IN\_AUDIT\_NOT\_TABLE**

**IN\_REJ\_AUDIT\_NOT\_TABLE**

**IN\_TABLE\_NOT\_AUDIT** ensures that all batch Ids existing in the model are logged in IBM\_AUDIT\_TABLE. Whereas **IN\_AUDIT\_NOT\_TABLE** ensures that all batch Ids logged in IBM\_AUDIT\_TABLE exist in the model.

**IN\_REJ\_TABLE\_NOT\_TABLE** ensures that all rejected batch Ids existing in the rejection table of each data source are logged in IBM\_REJECTION\_AUDIT, while **IN\_REJ\_AUDIT\_NOT\_TABLE** ensures that all batch Ids logged in **IBM\_REJECTION\_AUDIT** exists in rejection tables. Below is a detailed explanation of each of the above stored procedures.

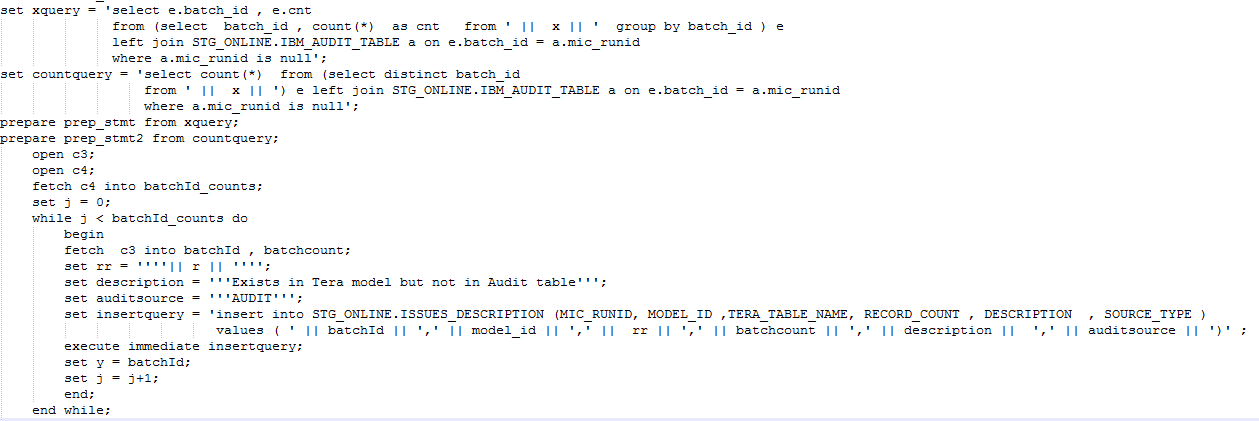
* + 1. **IN\_TABLE\_NOT\_AUDIT SP**



The procedure first joins **dbc.tables** ,which is a metadata table that contains data about all existing tables in the database, and **Table\_lookup** which contains the table names in each data source in DB2 with their corresponding name in Teradata. It also flags whether each table exists in Citizen or Economy database. It selects from TABLE\_LOOKUP tables that exist in the current database (‘C’ in case of citizen database, ‘E’ in case of Economy) in STG\_ONLINE schema as shown below.



Then loop through each table in the result set, get distinct batch ids and their count then left join with IBM\_AUDIT\_TABLE to identify Batch Ids that doesn’t exist in audit table. These batch ids are inserted in STG\_ONLINE.ISSUES\_DESCRIPTION table with description ‘**Exists in Tera model but not in Audit table’** and source **‘AUDIT’ (***representing IBM\_AUDIT\_TABLE*)



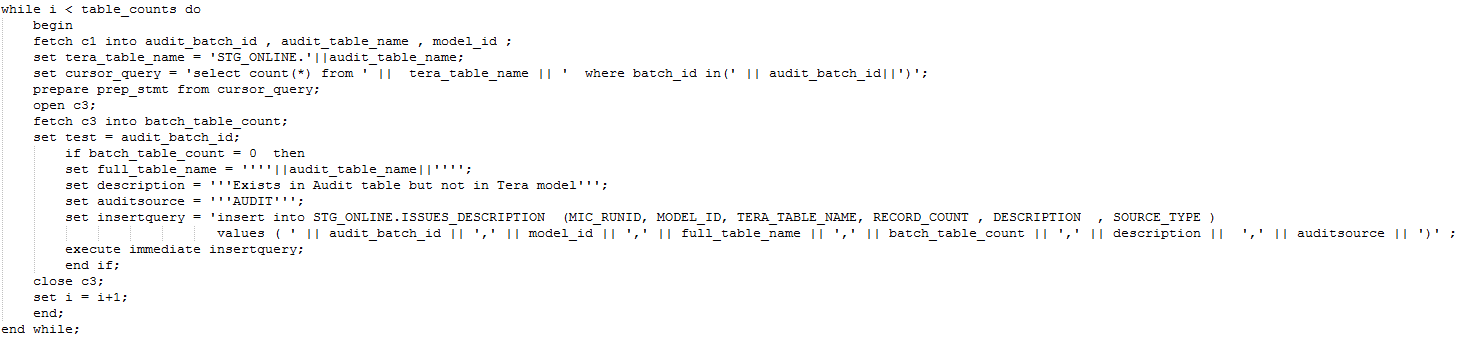
* + 1. **IN\_AUDIT\_NOT\_TABLE SP**



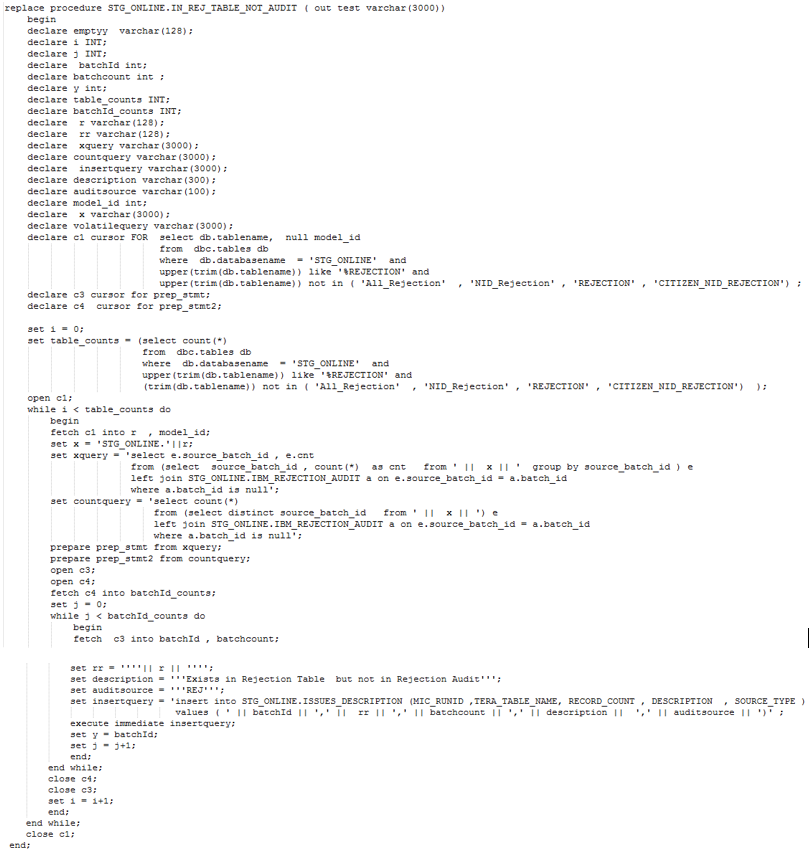
The procedure first joins the IBM\_AUDIT\_TABLE with TABLE\_LOOKUP to get only tables that exist in the current database (‘C’ in case of citizen database, ‘E’ in case of Economy).



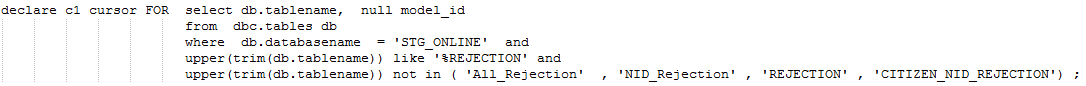
It selects batch id and table name and then loops through each table to ensure that the corresponding batch id exists in the table. If batch id doesn’t exist in the table, it is inserted in STG\_ONLINE.ISSUES\_DESCRIPTION table with description ‘**'Exists in Audit table but not in Tera Model’** andsource **‘AUDIT’ (***representing IBM\_AUDIT\_TABLE*)



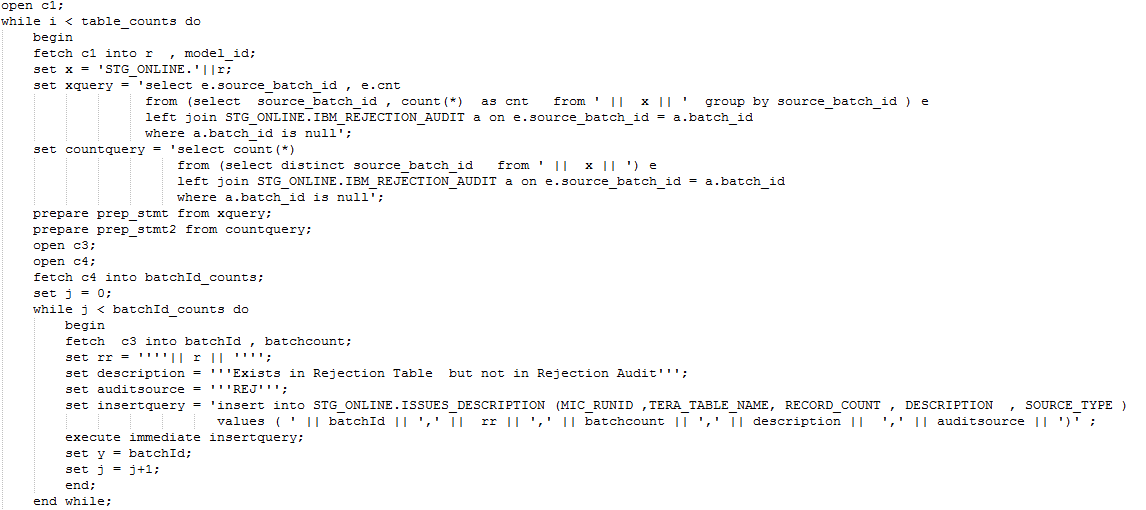
* + 1. **IN\_REJ\_TABLE\_NOT\_AUDIT SP**



The procedure first selects all rejection tables for each data source from dbc.tables excluding unused tables.



Then it loops through each table in the result set and selects distinct batch ids and then left joins with IBM\_REJECTION\_AUDIT to identify batch ids that do not exist in IBM\_REJECTION\_AUDIT. These Batch Ids are inserted in STG\_ONLINE.ISSUES\_DESCRIPTION table with description **'** **Exists in Rejection Table but not in Rejection Audit’** andsource **‘REJ’ (***representing IBM\_REJECTION\_AUDIT*)



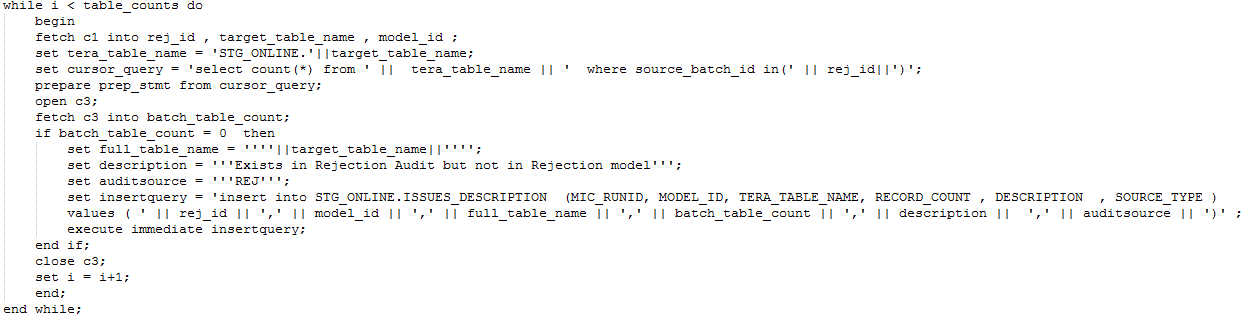
* + 1. **IN\_REJ\_AUDIT\_NOT\_TABLE SP**



The procedure first joins the IBM\_REJECTION\_AUDIT with TABLE\_LOOKUP to get only tables that exist in the current database (‘C’ in case of citizen database, ‘E’ in case of Economy).



It selects batch id and rejection table name from the result set and loops through each table to ensure that the corresponding batch id exists in the table. If batch id doesn’t exist in the table, it is inserted in STG\_ONLINE.ISSUES\_DESCRIPTION table with description **‘Exists in Rejection Audit but not in Rejection model’** andsource **‘REJ’ (***representing IBM\_REJECTION\_AUDIT*).



## Reusable Artifacts

### IBM Datastage Sequence



### Stored Procedures





# Audit Maintenance solution

Maintaining historical data in data warehouse is a critical task that needs to be handled securely to assure data credibility and auditability. To maintain such task, mainly in the master/transaction tables, a modification type column is added to these tables. This column indicates the type of change or the status of each record in the table, whether it’s the first presence of this record in the database which is flagged by an (I) insert modification type, or mark changes/updates happened on the record which is flagged by an (U) updated modification type, or the record no longer active/deleted from the database which is flagged by a (D) delete modification type. This modification type mechanism assures the presence of the latest up to date version of each record in each table, but due to some issues in data transfer, data loading and other types of issues this mechanism is violated somehow. The modification type maintenance solution discussed below is a database stored procedure that detects and logs such issues into a database table to be fixed later.

## Modification type maintenance solution description

The procedure runs on database server for each data source and insert its output into a table called **MODIFICATION\_TYPE\_ISSUES** that is checked later for solving the reported issues.

In each master/transaction table, it sorts each business key by the sequence it was received (from oldest to newest), in order to be able to track the operations done on that key every batch and ensure that it is a valid operation.

**It consists mainly of three main parts that detects below:**

### Unexpected modification types

In this part the procedure detects and logs the records that have unexpected modification types other than the confirmed types which are (I, D, U) like (1, 2, M ...etc) and insert them into **MODIFICATION\_TYPE\_ISSUES** table to be solved.

## 

### Duplicate modification type for the same batch

Each batch received should contain the latest image of each key. If there is any key duplication within the same batch, the procedure detects it and logs the full row

into **MODIFICATION\_TYPE\_ISSUES** table to be solved.

### Invalid modification type order

In this part the procedure detects and logs the records that have invalid modification type order, the right order for records insertion must be as follows:

Initial version of a record is marked with (I) then any update followed is either marked by (U) if the record is updated or marked by (D) if the record is deleted.

1. Once a record is marked by (D) the next version of it must be marked by (I) not (U).
2. The initial version of a record must not be marked by (U) or (D).

Once a record found violating this insertion criteria the procedure detects it and inserts it into **MODIFICATION\_TYPE\_ISSUES** table to be solved.

### Stored procedure snapshot

Graphical user interface, application

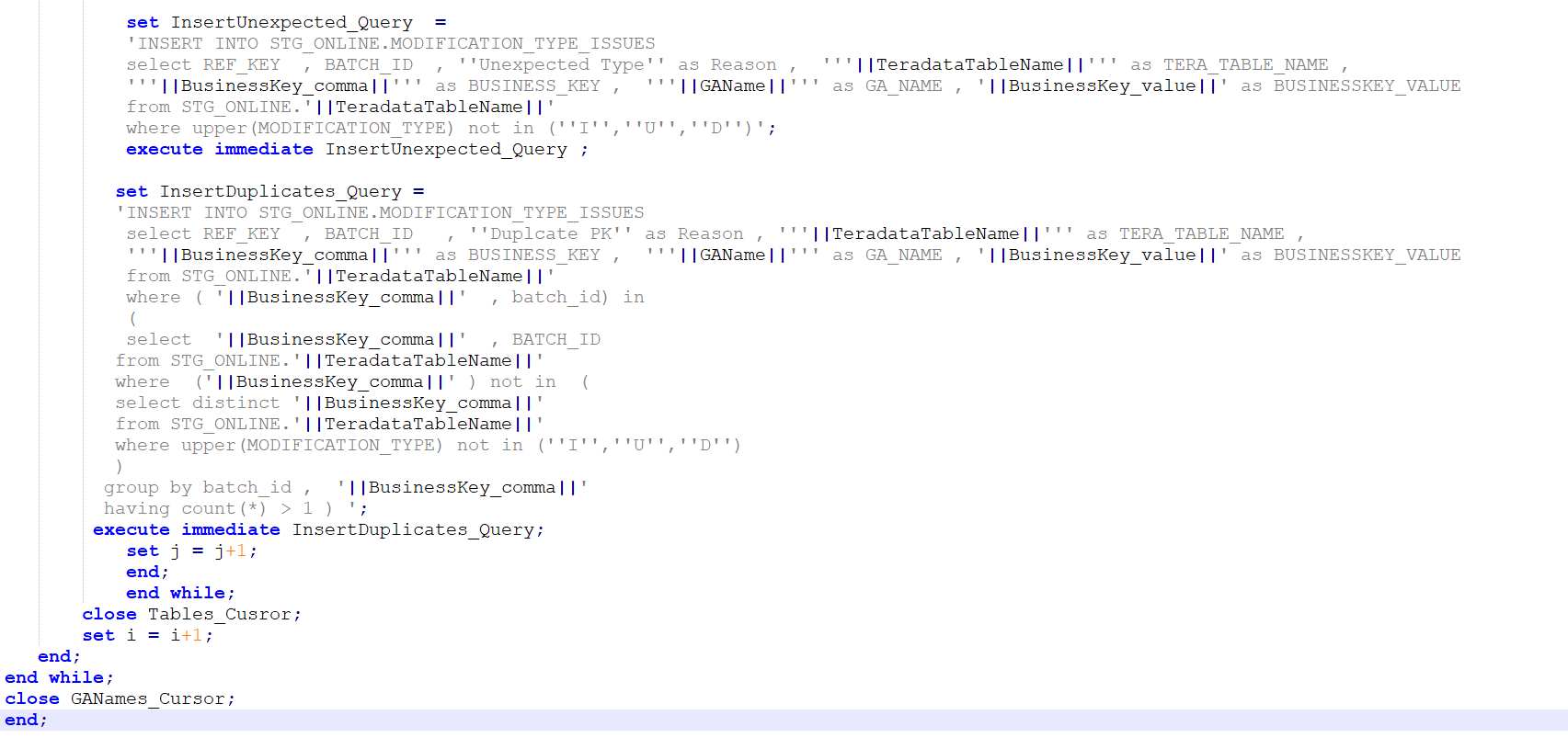
Description automatically generated



Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## Modification type maintenance solution running

The stored procedure takes the data source name as an input, loops on all its Master and Transaction tables which names are selected from **TABLE\_LOOKUP** (Table that contains the table names in each data source and their database names and their type whether they are master or transaction or lookup tables). The procedure detects all issues and then inserts the output into **MODIFICATION\_TYPE\_ISSUES** table.

So as a prerequisite, Table lookup must have the following:

* + TABLE\_TYPE should be equal ‘M’ master or ‘T’ transaction or ‘L’ lookup.
  + COMPOSITE BUSINESS\_KEYS should be comma separated.
  + GA NAME, TERADATA\_TABLE\_NAME, source\_db should not be NULL.

## TELECOM\_CONSOLIDATED Example

Below is an example on how to run the stored procedure on 'Telecom\_Consolidated' data source.

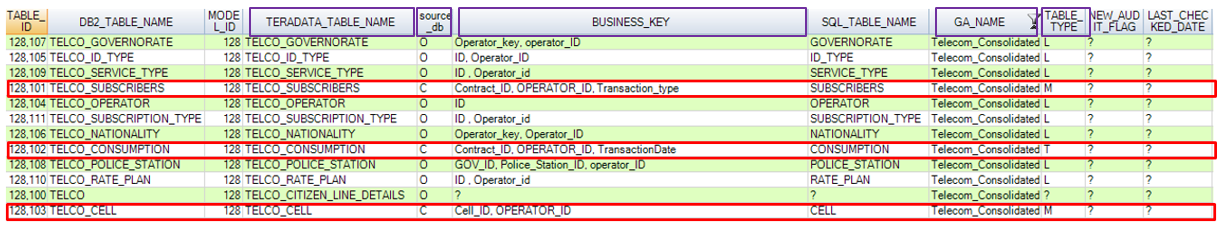
### Calling the stored procedure

To run the procedure on ‘TELECOM\_CONSOLIDATED’ data source use the following database command

*call STG\_ONLINE.MODIFICATION\_TYPE\_ORDER (‘TELECOM\_CONSOLIDATED’)*

### Detecting modification type issues

The procedure fetches **TABLE\_LOOKUP** for the master and transaction tables then loops on them which are (**TELCO\_SUBSCRIBERS**, **TELCO\_CONSUMPTION**, **TELCO\_CELL** tables)

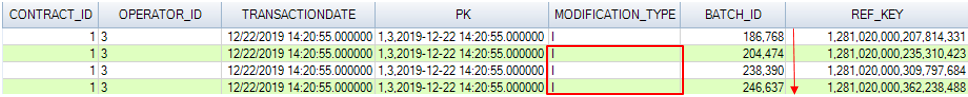


Taking **TELECO\_CONSUMPTION** Table as an example:

BUSINESS KEY**:** CONTRACT\_ID, OPEERATOR\_ID, TRANSACTION\_DATE

Business Key (CONTRACT\_ID = 1, OPERATOR\_ID = 3, TRANSACTION\_DATE = 12/22/2019) is sorted by the order it was received [oldest to newest] based on BATCH\_ID.

1. In the initial Batch (186,768) the Modification Type = I which is valid.
2. In the following Batches (204474, 238390, 246637) the Modification Type = I which is invalid.

****

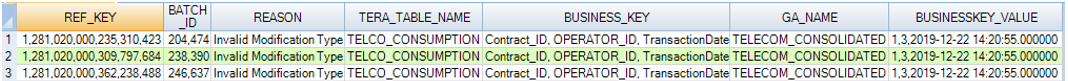
### Logging modification type issues

For each invalid record the BATCH\_ID, REF\_KEY, BUSNIESS\_KEY (comma separated)

are logged into **MODIFICATION\_TYPE\_ISSUES** table with different reasons depending on the types of issues discussed in section 2 of this document.

Taking **TELECO\_CONSUMPTION** Table as an example:

Each invalid record will be logged in **MODIFICATION\_TYPE\_ISSUES** with reason = ‘Invalid Modification Type’



## Reusable Artifacts:

