ECOLE POLYTECHNIQUE DE BRUXELLES

Année d'étude

Data Warehouse

TCP-DI Benchmarking using Pentaho Data Integration

Sony Shrestha (000583110)

Aayush Paudel (000583200)

MD Kamrul Islam (000583487)

Shofiyyah Nadhiroh (000583951)

Professeur

Esteban Zimanyi

2023

Contents

List of Figures

1 Introduction

1.1 ETL

ETL is a data integration process that involves extracting raw data from diverse sources, transforming it to meet analytical needs through cleansing and restructuring, and loading it into a centralized repository or data warehouse for analysis and reporting[Wik23]. The three major steps involved in ETL are:

Extract (E)

In this step, data is extracted from various source systems, which include databases, flat files, APIs, and other data repositories. The goal is to gather raw data from different sources to be used for analysis or reporting. The extracted data may be structured (E.g. Relational Databases) or unstructured (E.g. Log Files, Documents).

Transform (T)

Once the data is extracted, it undergoes a transformation process. Transformation involves cleaning, structuring, and enriching the data to make it suitable for analysis or reporting. Transformation tasks may include data cleansing (removing errors and inconsistencies), data normalization, data aggregation, and the application of business rules. The goal is to ensure that the data is accurate, consistent, and conforms to the desired structure.

Load (L)

After the data has been extracted and transformed, it is loaded into a target data repository or data warehouse. The data warehouse is a centralized storage system optimized for analytical processing. Loading can be done in batches or in real-time, depending on the requirements of the

organization. The loaded data is typically organized in a way that facilitates efficient querying and reporting.

1.2 TPC-DI Benchmark

The TPC-DI benchmark is designed as a way of evaluating performance of tools involved in the Data Integration (DI) process. The benchmark workload focuses on manipulating a specified volume of data to prepare it for utilization in a Data Warehouse. The model incorporates data sourced from an OLTP system, requiring transformation alongside data from diverse sources for loading it into the Data Warehouse.
This benchmark aims to assess the behavior of system components related to the DI process, characterized by:

1. Manipulation and loading of substantial data volumes
2. Application of diverse transformations to the data
3. Historical loading and incremental updates
4. Adherence to consistency requirements
5. Handling multiple data sources with varying formats
6. Managing multiple data tables with diverse data types, attributes, and relationships

Operations within the benchmark are modeled as follows:

1. Source data is generated automatically in flat files.
2. The transformation process commences with the System Under Test (SUT) reading the source data.
3. Transformations validate the source data and structure it appropriately for loading into the Data Warehouse.
4. The process concludes when all source data has been transformed and is available in the Data Warehouse.

The benchmark's performance metric is measured in throughput (ops/sec), specifically the Number of Source data rows processed per second.

1.2.1 Business and Application Environment

The data model for the TPC-DI benchmark represents a retail brokerage[Tea21]. Figure 1.1 illustrates the conceptual model of the brokerage DI system.

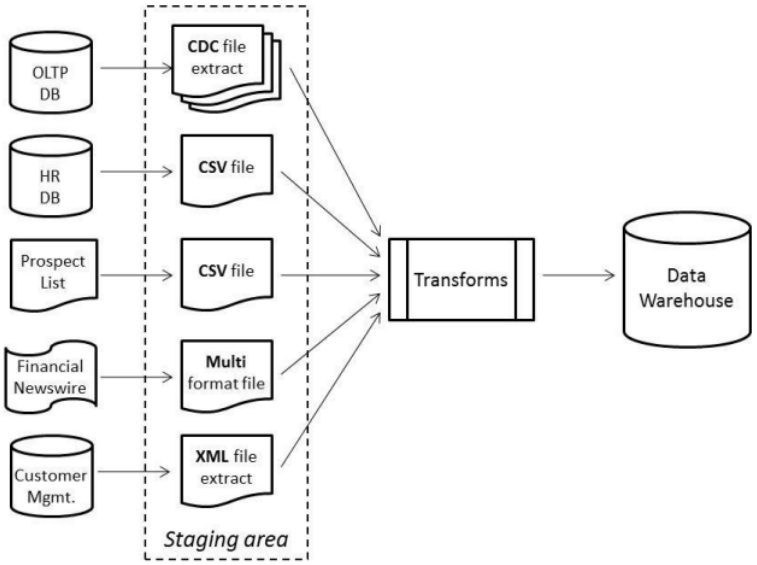


Figure 1.1: Conceptual Model of the brokerage DI system

The OLTP system comprises numerous tables containing information on customers, accounts, brokers, securities, trade details, account balances, market information, and more. These tables are extracted into a staging area, and the extracted data is represented as flat files. The extracts in the staging area take two forms: Changed Data Capture (CDC) extracts for Incremental Updates, showcasing changes to the tables since the last extract, and for the Historical Load, the extract is structured as a comprehensive dump of the entire tables.

The HR database features a single table containing employee data, which is extracted as a complete table and represented in the Staging Area as a comma-separated values (CSV) file.

The Prospects file contains names, addresses, and demographic data for potential customers, resembling the type of information a company could obtain from a syndicated data provider. This data is delivered in a comma-separated values (CSV) file format. The Data Integration (DI) process is responsible for recognizing and evaluating changes in this file since the last update.

During the Historical Load phase of the benchmark, two additional sources are utilized to supply information not directly accessible from the OLTP system. Financial details regarding companies and securities are sourced from a **Financial Newswire Service (FINWIRE)** with an extensive archival history. This data is presented in variable-format records within files stored in the Staging Area. Information related to customers and accounts is extracted from a **Customer Management System**. Historical CMS data is stored in the Staging Area in the form of an XML-formatted extract.

1.2.2 Summary of Operations

TPC-DI depicts an environment where data from different source systems has been extracted and stored in flat files within a staging area, serving as the starting point for the rest of the Data Integration (DI) process. It does not aim to replicate the extensive variety of data sources present in the marketplace; instead, it models abstracted data sources and assesses all systems engaged in the movement and transformation of data from the Staging Area to the Data Warehouse.

Phases of Operation

It is comprised of four different phases, namely, Initialization Phase, Historical Phase, Incremental Update Phase and Automated Audit Phase.

Initialization Phase

Certain initial operations are necessary, although their execution times are not considered for the benchmark. These operations encompass:

1. Utilizing an automated generator provided by the benchmark to generate and add data into the Staging Area. Scale Factor must be specified to let generator know volume of data that needs to be generated.

2. Creating database along with all the tables of Data Warehouse.

Historical Load Phase

The Historical Load Phase consists of transformations different from Incremental Updates, along with different and extensive data compared to later one. Transformation rules specific to Historical Load are provided in the benchmark specification. This phase uses files from **Batch1 directory** of the generated data.

Incremental Update Phase

Transformation rules specific to Incremental Update are specified in the benchmark specification. Unlike Historical Load, where data is loaded in the order of primary key, the Incremental Update phase loads data in the order of the changes made to it. Two Incremental Updates are performed for confirmation, and after each update, a Validation Query is executed to collect execution information for later evaluation.
This phase relies on **Changed Data Capture (CDC)** extracts from the OLTP system. The extracts

include a "CDC_FLAG" column with values I, U, or D, indicating whether a row was inserted, updated, or deleted since the last version. A row may undergo multiple changes in a day. The "CDC_DSN" column in the extract is used to order rows based on the time of changes, and it is specific to each data source.
Tables modified during Incremental Updates have both primary and surrogate keys. While primary keys remain constant, surrogate keys are updated for each recorded change. **Batch2 and Batch3 directories** provide the necessary files for executing the first and second incremental updates.

Automated Audit Phase

Conducted as the final step after all other phases, the Automated Audit phase executes queries to verify the integrity of the data stored in the Data Warehouse. Subsequently, it generates a comprehensive report presenting the outcomes of all conducted tests. The run is deemed valid only if all individual tests pass successfully.

1.2.3 Source Data Models

The OLTP database represents a database with transactional information about securities market trading and the entities involved, i.e. customers, accounts, brokers, securities, trade details, account balances, market information, and so on. Files used as input to the Historical Load are full extracts containing all the rows in the table and input to an Incremental Update are CDC extracts containing columns like "CDC_FLAG" and "CDC_DSN" at the beginning of each row. The CDC_FLAG is a single character I, U or D specifying whether the row has to be inserted, updated or deleted since the last change. For updates there is no indication as to what in the row has been changed. Rows that have not changed since the last extract will not appear in the CDC extract file at all. Files from the OLTP system include:

1. Account.txt (Incremental Update)
2. Customer.txt (Incremental Update)
3. Trade.txt (Historical Load and Incremental Update)
4. TradeHistory.txt (Historical Load)
5. CashTransaction.txt (Historical Load and Incremental Update)
6. HoldingHistory.txt (Historical Load and Incremental Update)

7. DailyMarket.txt (Historical Load and Incremental Update)

8. WatchItem.txt (Historical Load and Incremental Update)

The HR database is represented by a single extract file, HR.csv. This file encompasses details about the company's employees without Changed Data Capture (CDC).

The Prospect.csv file signifies data sourced from an external data provider, encompassing names, contact details, and demographic information about potential customers, including those who are already clients of the brokerage. The structure of this file is designed to represent a complete daily extract from the data provider, meaning there is no inherent indication within the data itself regarding what has changed since the previous extraction.

For each quarter of historical data, a **FINWIRE** file exists, featuring three distinct record types with fixed-width fields. These record types contribute data to the tables DimCompany, DimSecurity, and Financial during the Historical Load phase. Although updates to company and security data are typical in a data warehouse, the benchmark doesn't account for these changes in the Incremental Update phases due to their minimal impact on performance.

The CustomerMgmt.xml file embodies data extracted from a Customer Management System, which is used to populate the DimAccount and DimCustomer tables during the Historical Load phase.

Reference data is loaded only during the Historical Load and not modified again in the course of the benchmark. The list of reference files are provided below.

1. Date.txt

2. Time.txt

3. Industry.txt

4. StatusType.txt

5. TaxRate.txt

6. TradeType.txt

1.2.4 Destination Data Model

The destination of the TPC-DI workload is a data warehouse, with number of dimensions and facts table. **Dimension Tables** contain information about dates (in the DimDate table), times (DimTime), customers (DimCustomer), accounts (DimAccount), brokers (DimBroker), securities (DimSecurity), companies (DimCompany), and trades (DimTrade). The **Fact Tables** describe holdings

(FactHoldings), trades (DimTrade), cash balances (FactCashBalances), the market history (FactMarketHistory), and customer watches on securities (FactWatches). It is sometimes possible that certain table serves more than one role like The DimTrade table (both dimension and fact), depending on how it is being used.

1.2.5 Transformations

Transformations encompass all tasks necessary for the preparation and loading of data into the Data Warehouse. This includes:

1. Converting data from character representations to types compatible with the Data Warehouse specifications.
2. Performing lookups of business keys to acquire surrogate keys for the Data Warehouse.
3. Merging or formatting multiple fields into one or splitting one field into multiple.
4. Verifying data for errors or compliance with business rules.
5. Identifying changes in dimension data and implementing suitable tracking mechanisms, such as retaining historical records or overwriting.
6. Identifying changes in fact data and recording updates to reflect the current state.

1.3 DIGen

DIGen is a generator based on PDGF to create data sources and audit information for TPC-DI benchmark. It is required to be executed in a Java environment and PDGF needs to be placed in the same directory[Nam18]. It produces a data generation statistics file named **digen_report.txt** as part of the data generation process which is used to calculate the metric and for auditing. The file contains:

1. General information about the data generation process
2. Options used during the generation process
3. Row counts generated for each batch

1.4 Performance Metrics

1.4.1 Throughput

Throughput for each phase is calculated by dividing number of rows loaded by that phase by the total time needed to complete it .

The Throughput of Historical Load is given by:

$$T_H = \frac{R_H}{E_H}$$

where,

R_H = Number of rows loaded during Historical Phase

$E_H = CT_1 - CT_0$

CT_0 = Completion Time of Initialization Phase

CT_1 = Completion Time of Historical Load Phase

Completion Time of Initialization Phase can be extracted by using query given below.

```
SELECT
    MessageDateAndTime
FROM DImessages
WHERE BatchID = 0 AND
MessageType = 'PCR';
```

Completion Time of Historical Load Phase can be extracted by using query given below.

```
SELECT
    MessageDateAndTime
FROM DImessages
WHERE BatchID = 1 AND
MessageType = 'PCR';
```

The Throughput of Incremental Updates is given by

$$T_{I_1} = \frac{R_{I_1}}{MAX(E_{I_1}, 1800)}$$

$$T_{I_2} = \frac{R_{I_2}}{MAX(E_{I_2}, 1800)}$$

where,

R_{I1} = Number of rows loaded during first Incremental Update Phase

R_{I2} = Number of rows loaded during second Incremental Update Phasee

$E_{I1} = CT_2 - CT_1$

$E_{I2} = CT_3 - CT_2$

CT_1 = Completion Time of Historical Load Phase

CT_2 = Completion Time of first Incremental Update Phase

CT_3 = Completion Time of second Incremental Update Phase

Completion Time of Historical Load Phase (CT_1) can be extracted by using query given below.

```
SELECT
    MessageDateAndTime
FROM DImessages
WHERE BatchID = 1 AND
MessageType = 'PCR';
```

Completion Time of first Incremental Update Phase (CT_2) can be extracted by using query given below.

```
SELECT
    MessageDateAndTime
FROM DImessages
WHERE BatchID = 2 AND
MessageType = 'PCR';
```

Completion Time of second Incremental Update Phase (CT_3) can be extracted by using query given below.

```
SELECT
    MessageDateAndTime
FROM DImessages
WHERE BatchID = 2 AND
MessageType = 'PCR';
```

1.4.2 Performance metric

The Performance Metric is a combined metric using the three throughput values and is given by:

$$TPC\_DI\_RPS = \text{Trunc}\,(\text{GeoMean}(T_H, MIN(T_{I1}, T_{I2}))$$

where,

T_H = Throughput of the Historical Load

T_{I1} = Throughput of Incremental Update 1

T_{I2} = Throughput of Incremental Update 2

GeoMean() = Geometric Mean of the arguments

Min() = Argument with the least value

Trunc() = Whole number portion of the argument

1.4.3 Price/Performance metric

The Price/Performance is computed using the performance metric TPC_DI_RPS as:

$$\text{Price-per-TPC\_DI\_RPS} = \frac{\$}{\text{TPC\_DI\_RPS}}$$

where,

$ = Total 3-years pricing

TPC_DI_RPS = Combined Performance Metric

We have not considered this measure in our analysis.

1.4.4 Slowly Changing Dimensions

Slowly Changing Dimensions (SCD) play a crucial role in a Data Warehouse by preserving the historical changes in data sources. The TPC-DI benchmark specifically addresses SDCs of **Type 2** with dependencies. In this type of dimension, changes in values are tracked by incorporating two temporal attributes: **StartDate** and **EndDate**. These attributes signify the date of record insertion in the database and the date when the tuple was deleted from the source table, respectively. Additionally, 'with dependencies' implies that the SCD table includes at least a surrogate key reference to another table. Consequently, when an update occurs in the referenced dimension, the referencing table also requires updating.

2 Implementation Details

This section provides a comprehensive overview of the methodology employed for TPC-DI benchmarking using Pentaho Data Integration (PDI).

2.1 Setting up Oracle Database

We installed **Oracle Database 21c Enterprise Edition, Release 21.0.0.0.0 - Production** in our local machine. All of the results in this report were obtained using Windows 64 bits, AMD Ryzen 5 5600H with Radeon Graphics and 16 GB of RAM.

2.2 Setting up Pentaho Data Integration (PDI)

In order to perform ETL operations, we installed **Pentaho Data Integration 9.4** in our Windows Machine, which requires Java as a pre-requisite.

2.3 Generating Data

To generate the data, we have used the Data Generation Tool provided by the TPC-DI developers called **DIGen.jar**. The Java program to generate the source files is downloaded by filling out a form on the TPC-DI website and clicking a link in an email. The downloaded zip file needs to be unzipped and PDGF folder needs to be renamed to pdgf using following command.

```
unzip FBBFF443-FB65-4A82-8BFC-76FF65803462-TPC-DI-Tool.zip
mv Tools/PDGF Tools/pdgf
cd Tools
```

DIGen.jar file needs to be used in order to generate data for different scale factors.

```
java -jar .\DIGen.jar -sf 6 \
-o C:\ULB\Data_Warehouse\TPC_DI_TOOLKIT\Tools\output\sf6
```

The above command was executed with scale factors of 3, 4, 5, and 6, resulting in the generation of data stored in separate folders. Each directory consists of raw data in diverse formats such as CSV, TXT, XML etc.

2.4 Table Structure

A DDL script was generated based on the table definitions outlined in the TPC-DI Specification. This script includes the creation of 20 tables, encompassing Dimension, Fact, Reference, and DI Operational Tables. Figure **??** shows the pictorical overview of the Data Warehouse Tables.

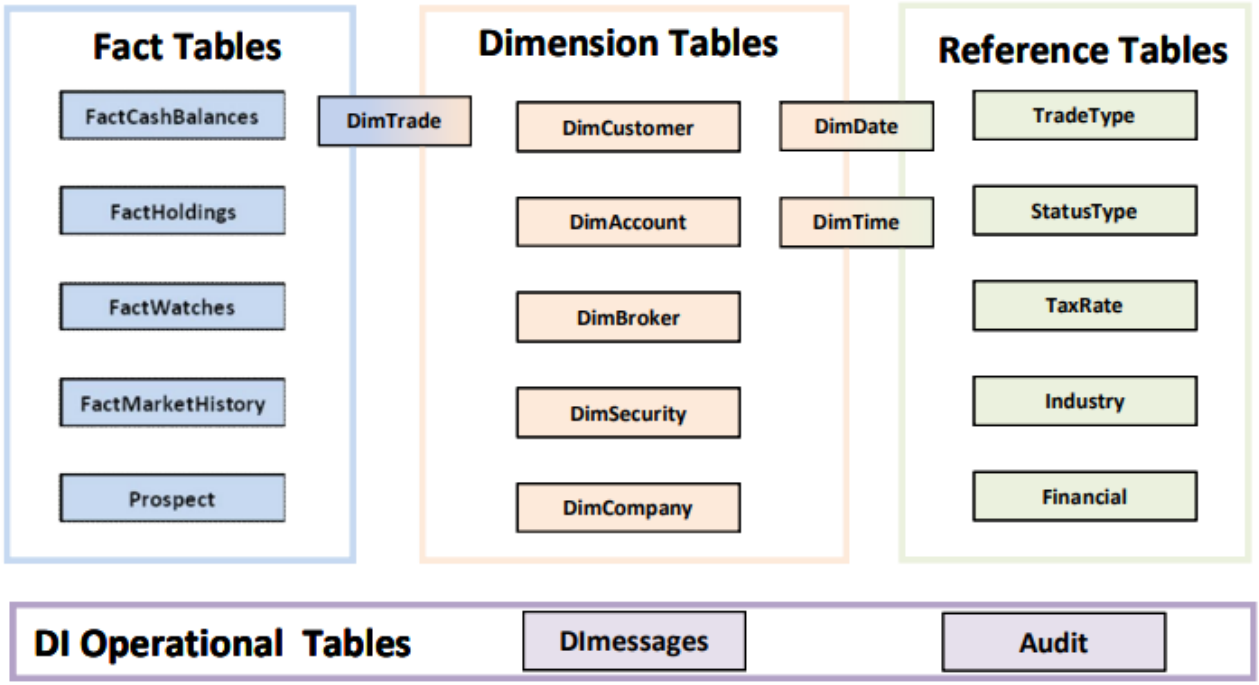


Figure 2.1: Pictorial Overview of the Data Warehouse Tables

2.5 Conceptual ETL Design

Figure 2.2 shows conceptual view of ETL that needs to be performed for TPC-DI benchmarking. It shows the sequence in which data must be loaded into database.

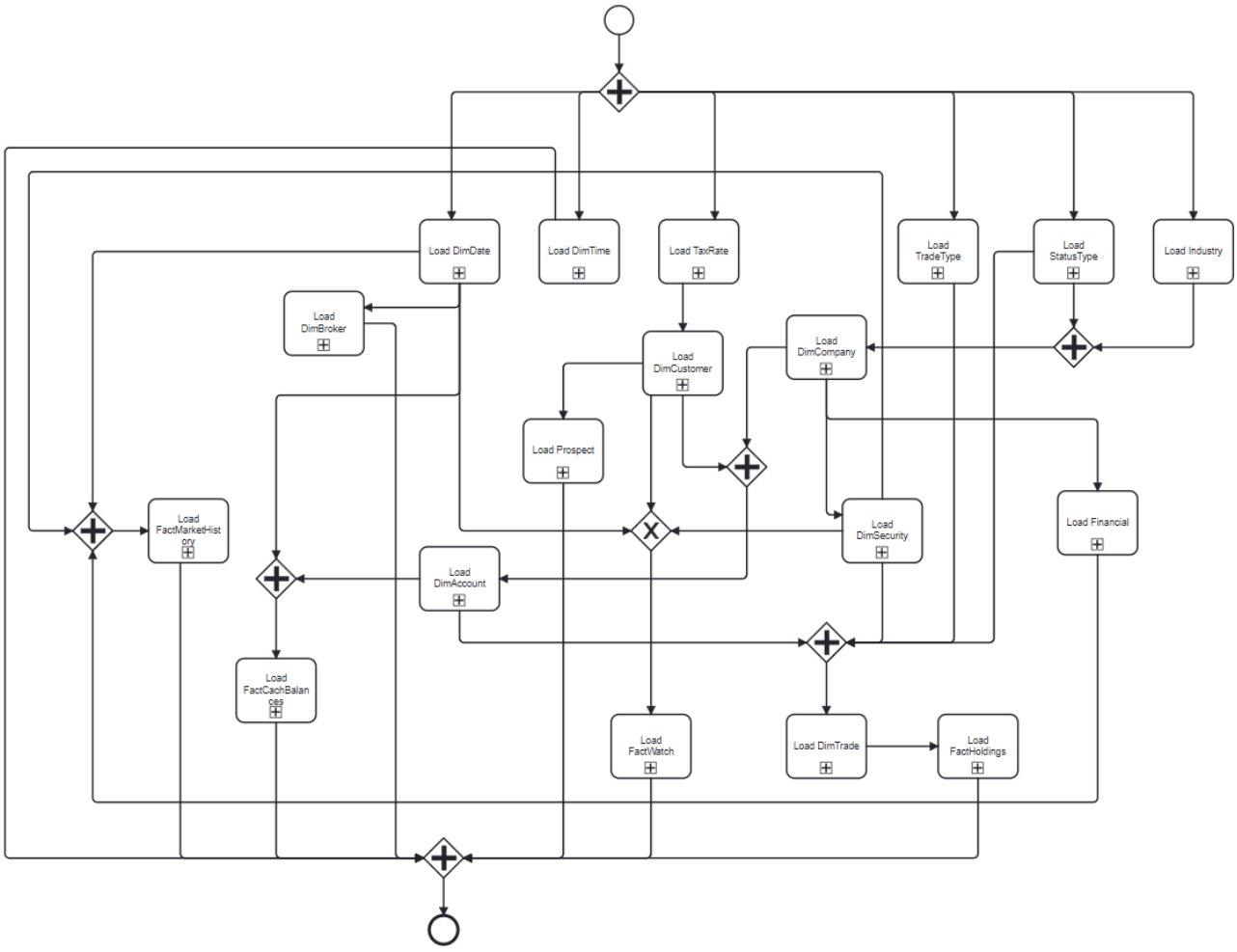


Figure 2.2: Conceptual ETL Design

2.6 Transformations for Historical Data Load

Pentaho Transformations were prepared in order to load historical data into the database. The transformation details were provided in the specification document under clause 4.5.

2.6.1 DimDate

The transformation details for loading historical data into the table DimDate is provided below.

1. **DimDate** table is populated from **Date.txt** file.
2. This is a static table and should only be loaded once without any further modifications.
3. Fields in Date.txt file are mapped to corresponding fields in the table, with no further modifications.

Figure 2.3 shows transformation prepared in order to load historical data into DimDate table.

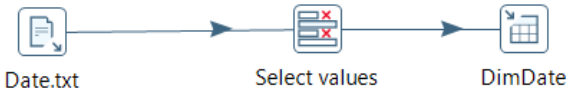


Figure 2.3: Historical DimDate Table Load

2.6.2 DimTime

The transformation details for loading historical data into the table DimTime is provided below.

1. **DimTime** table is populated from **Time.txt** file.
2. This is a static table and should only be loaded once without any further modifications.
3. Fields in Time.txt file are mapped to corresponding fields in the table, with no further modifications.

Figure 2.4 shows transformation prepared in order to load historical data into DimTime table.

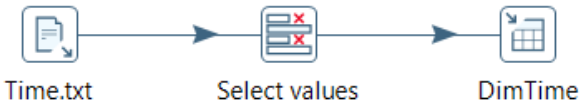


Figure 2.4: Historical DimTime Table Load

2.6.3 TaxRate

The transformation details for loading historical data into the table TaxRate is provided below.

1. **TaxRate** table is populated from **TaxRate.txt** file.
2. This is a static table and should only be loaded once without any further modifications.
3. Fields in TaxRate.txt file are mapped to corresponding fields in the table, with no further modifications.

Figure 2.5 shows transformation prepared in order to load historical data into TaxRate table.

Figure 2.5: Historical TaxRate Table Load

2.6.4 StatusType

The transformation details for loading historical data into the table StatusType is provided below.

1. **StatusType** table is populated from **StatusType.txt** file.
2. This is a static table and should only be loaded once without any further modifications.
3. Fields in StatusType.txt file are mapped to corresponding fields in the table, with no further modifications.

Figure 2.6 shows transformation prepared in order to load historical data into StatusType table.

Figure 2.6: Historical StatusType Table Load

2.6.5 TradeType

The transformation details for loading historical data into the table TradeType is provided below.

1. **TradeType** table is populated from **TradeType.txt** file.
2. This is a static table and should only be loaded once without any further modifications.
3. Fields in TradeType.txt file are mapped to corresponding fields in the table, with no further modifications.

Figure 2.7 shows transformation prepared in order to load historical data into TradeType table.

Figure 2.7: Historical TradeType Table Load

2.6.6 Industry

The transformation details for loading historical data into the table Industry is provided below.

1. **Industry** table is populated from **Industry.txt** file.
2. This is a static table and should only be loaded once without any further modifications.
3. Fields in the Industry.txt file are mapped to corresponding fields in the table, with no further modifications.

Figure 2.8 shows transformation prepared in order to load historical data into Industry table.

Figure 2.8: Historical Industry Table Load

2.6.7 DimBroker

The transformation details for loading historical data into the table DimBroker is provided below.

1. **DimBroker** table is populated from the **HR.txt** file.
2. All HR records having **EmployeeJobCode=314** are considered as Brokers.
3. EffectiveDate is set to the minimum date from the DimDate Table.
4. EndDate is set to **9999-12-31**.
5. BatchId is set to 1 for historical data load.

6. IsCurrent is set to 1.

7. Surrogate Key is generated for each record in the DimBroker table.

Figure 2.9 shows transformation prepared in order to load historical data into DimBroker table.

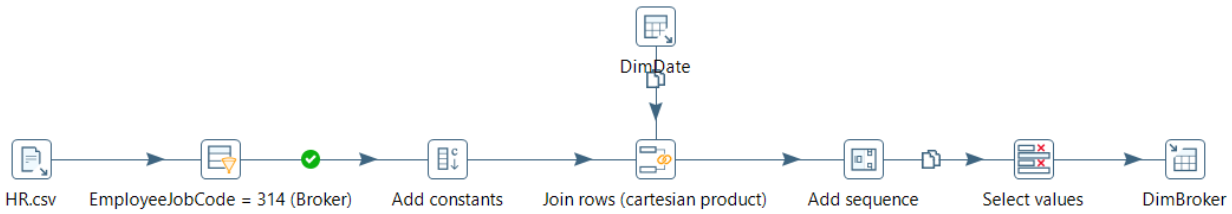


Figure 2.9: Historical DimBroker Table Load

2.6.8 DimCompany

The transformation details for loading historical data into the table DimCompany is provided below.

1. **FINWIRE** files provided with prefix FINWIREYYYYQQ in their file name, satisfying filtering criteria **RecType=CMP**, are used to load data into **DimCompany** Table.

2. StatusType and Industry tables are used to fetch Status and Industry Information respectively.

3. PTSDate is equivalent to EffectiveDate.

4. EndDate is set to 9999-12-31.

5. Surrogate Key is generated for each record in the DimCompany table.

6. BatchId is set to 1 for Historical Load.

7. If SPrating starts with 'A' or 'BBB', isLowGrade is set to False, True otherwise.

8. Empty string is populated as NULL.

9. AAA, AA[+/-], A[+/-], BBB[+/-], BB[+/-], B[+/-], CCC[+/-], CC, C, D are valid values for field SPRating. In case, value of SPRating is invalid, SPRating and isLowGrade are populated as NULL. Additionally, such entries are also tracked in DIMessages table with:

(a) MessageType = 'Alert'

(b) MessageText = 'Invalid SPRating'

(c) MessageSource = 'DimCompany'

(d) MessageData = 'C_ID =' + SK_COMPANYID + ',CO_SP_RATE = ' + SPrating

Figure 2.10 shows transformation prepared in order to load historical data into DimCompany table.

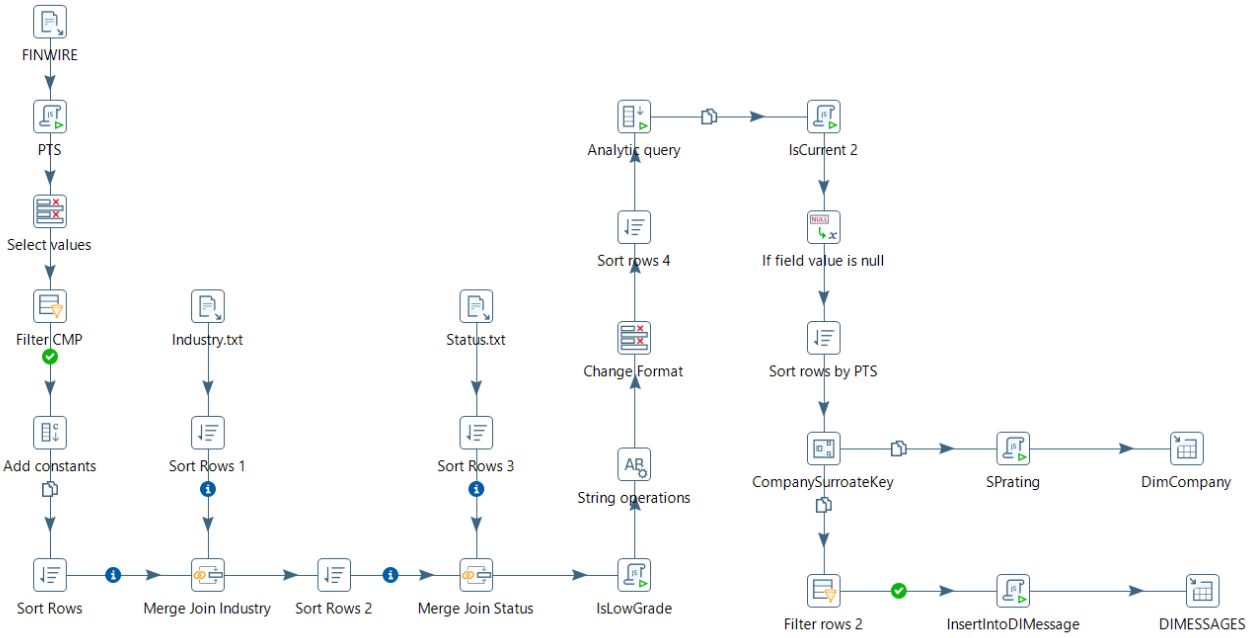


Figure 2.10: Historical DimCompany Table Load

2.6.9 Financial

The transformation details for loading historical data into the table Financial is provided below.

1. **FINWIRE** files provided with prefix FINWIREYYYYQQ in their file name, satisfying filtering criteria **RecType = FIN**, are used to load data into **Financial** Table.

2. Company information (SK_COMPNAYID) is obtained from DimCompany table by matching CoNameOrCIK with Name or CIKcode (depending on the characters found in CoNameOrCIK), where EffectiveDate <= PTS < EndDate.

Figure 2.11 shows transformation prepared in order to load historical data into Financial table.

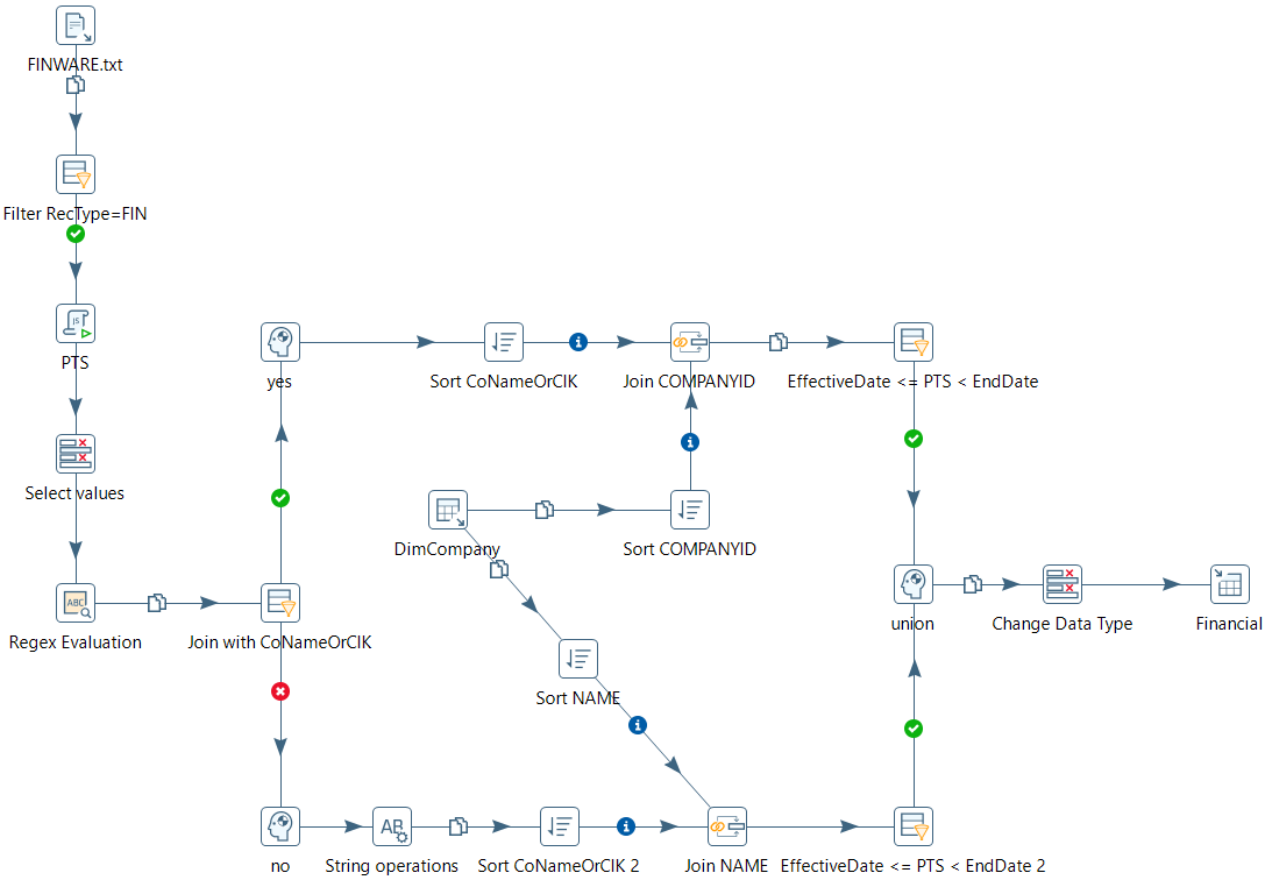


Figure 2.11: Historical Financial Table Load

2.6.10 DimSecurity

The transformation details for loading historical data into the table DimSecurity is provided below.

1. **FINWIRE** files provided with prefix FINWIREYYYYQQ in their file name, satisfying filtering criteria **RecType = SEC**, are used to load data into **DimSecurity** Table.

2. Company information (SK_COMPNAYID) is obtained from DimCompany table by matching CoNameOrCIK with Name or CIKcode (depending on the characters found in CoNameOrCIK), where EffectiveDate <= PTS < EndDate.

3. Status information is obtained from the StatusType table using ST_ID.

4. BatchID is set to 1 for historical load.

5. EffectiveDate is equivalent to PTS.

6. IsCurrent is set to 1 where ENDDATE = '9999-12-31 00:00:00', else 0.

While preparing the transformation for the DimSecurity table, we need to consider the following

points.

1. There is some complexity in "how" to handle the CONAMEORCIK. If it is numeric, it is CIK. If it is not numeric it is CONAME. Therefore, we need to handle joining DimCompany by determining whether it is numeric or not.

2. We also need to handle the case where DimCompany records updates in between the effective/end date for the Security record. So, we need to check if the DimCompany record end date is before the end date of the security record (also test the other way where the effective date of DimCompany is after the effective date of the security record). This is because the join where the date is between the DimCompany effective/end dates will match multiple records if the DimCompany record was updated during the duration of the Security record.

Figure 2.12 shows transformation prepared in order to load historical data into DimSecurity table.

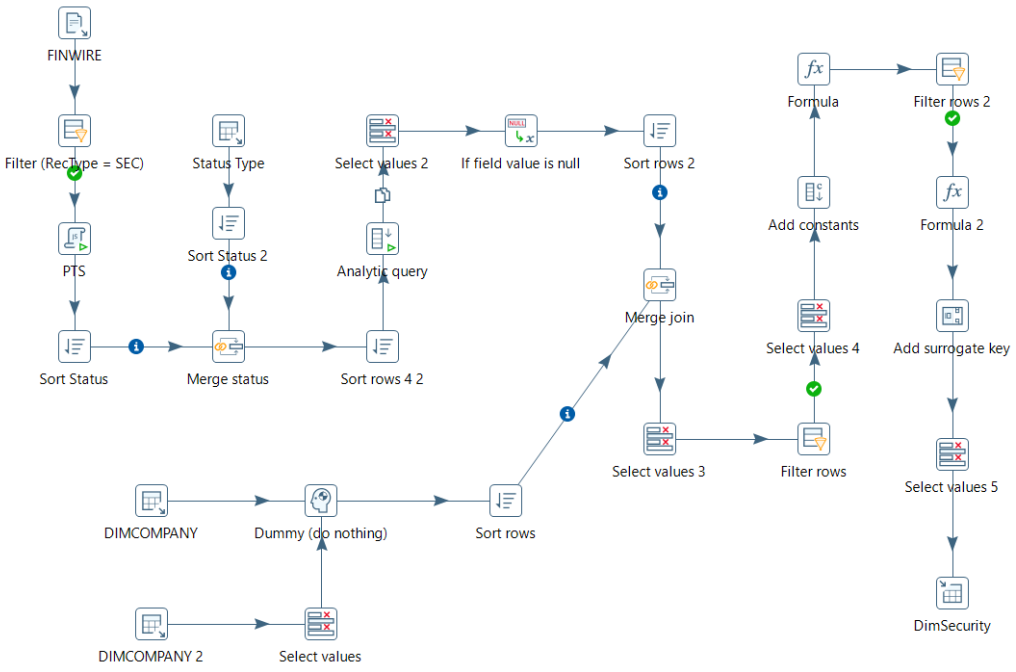


Figure 2.12: Historical DimSecurity Table Load

2.6.11 DimCustomer

The transformation details for loading historical data into the table DimCustomer is provided below.

1. **CustomerMgmt.xml** file is used to load data into **DimCustomer** Table.

2. Gender is uppercased and values other than M and F are populated as U.
3. Phone Number is transformed as:
   (a) PhoneNumber = '+C_CTRY_CODE' + '(C_AREA_CODE)' + 'C_LOCAL' + 'C_EXT', where only provided parts are added.
   (b) If C_CTRY_CODE is not null, display C_CTRY_CODE as +C_CTRY_CODE
   (c) If C_AREA_CODE is not null, display C_AREA_CODE as (C_AREA_CODE)
4. NationalTaxRateDesc, NationalTaxRate, LocalTaxRateDesc and LocalTaxRate are obtained from table TAX_RATE.
5. AgencyID, CreditRating, NetWorth, MarketingNameplate are obtained from Prospect File.
6. BatchId is set to 1 for historical load.
7. PTSDate is equivalent to EffectiveDate.
8. Surrogate Key is generated for each record.
9. EndDate is set to '9999-12-31 00:00:00'.
10. Latest Record for each customer is set to IsCurrent = 'Y'.
11. When ./@ActionType is 'NEW', New record has to be added in DIMCUSTOMER table
12. When ./@ActionType is 'UPDCUST', update is made to one or more existing customer's information.
13. When ./@ActionType is 'INACT', existing customer is marked as Inactive Customer.

While preparing the transformation for the DimCustomer table, we need to consider the following points.

1. Invalid Customer Tier values i.e. value not in (1,2,3) should be reported in DIMessages Table with:
   (a) MessageType = 'Alert'
   (b) MessageText = 'Invalid customer tier'
   (c) MessageSource = 'DimCustomer'

(d) MessageData = 'C_ID =' + SK_CUSTOMERID + ', C_TIER = ' + C_TIER

2. Invalid DOB i.e. if DOB < Batch Date – 100 years or DOB > Batch Date should be reported in DIMessages Table with:

   (a) MessageType = 'Alert'

   (b) MessageText = 'DOB out of range'

   (c) MessageSource = 'DimCustomer'

   (d) MessageData = 'C_ID =' + SK_CUSTOMERID + ', C_DOB = ' + C_DOB

Figure 2.13 shows transformation prepared in order to load historical data into DimCustomer table.

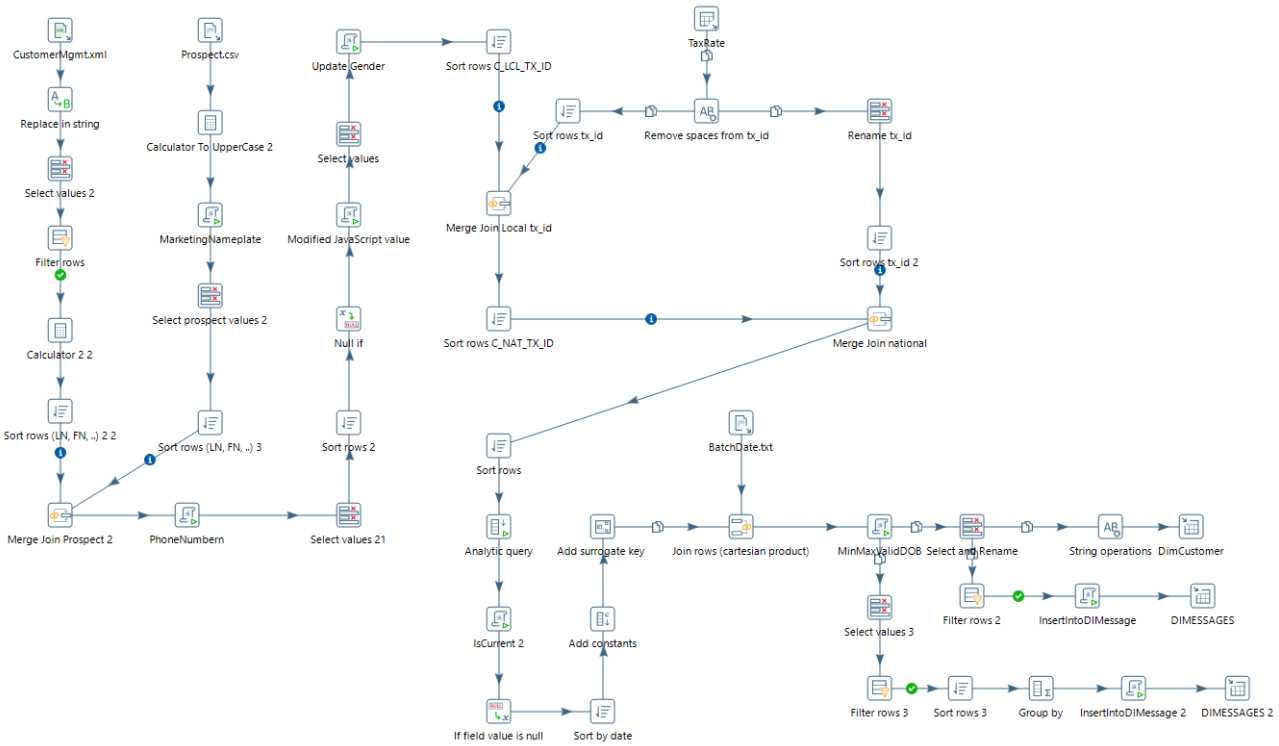


Figure 2.13: Historical DimCustomer Table Load

2.6.12 Prospect

The transformation details for loading historical data into the table Prospect is provided below.

1. **Prospect.csv** file is used to load data into **Prospect** Table.

2. AgencyID is a unique identifier for Prospect.

3. If the prospective customer record matches a current customer record in DimCustomer whose status is 'ACTIVE', then IsCustomer is set to 1, otherwise set to 0. A Prospect record is deemed to match a DimCustomer record if the FirstName, LastName, AddressLine1, AddressLine2 and PostalCode fields all match when upper-cased.

4. MarketingNameplate is set by concatenating zero or more tags with a "+" character between them if multiple tags apply to a given customer. For example, a prospect that qualifies for both the "Boomer" tag and the "Spender" tag would be assigned the MarketingNameplate value "Boomer+Spender". If multiple tags are used they must be in the order given below, and if no tags apply the nameplate is NULL.

5. The tags are defined as:

   (a) HighValue: NetWorth > 1000000 or Income > 200000

   (b) Expenses: NumberChildren > 3 or NumberCreditCards > 5

   (c) Boomer: Age > 45

   (d) MoneyAlert: Income < 50000 or CreditRating < 600 or NetWorth < 100000

   (e) Spender: NumberCars > 3 or NumberCreditCards > 7

   (f) Inherited: Age < 25 and NetWorth > 1000000

6. BatchId is set to 1 for historical load.

Figure 2.14 shows transformation prepared in order to load historical data into Prospect table.

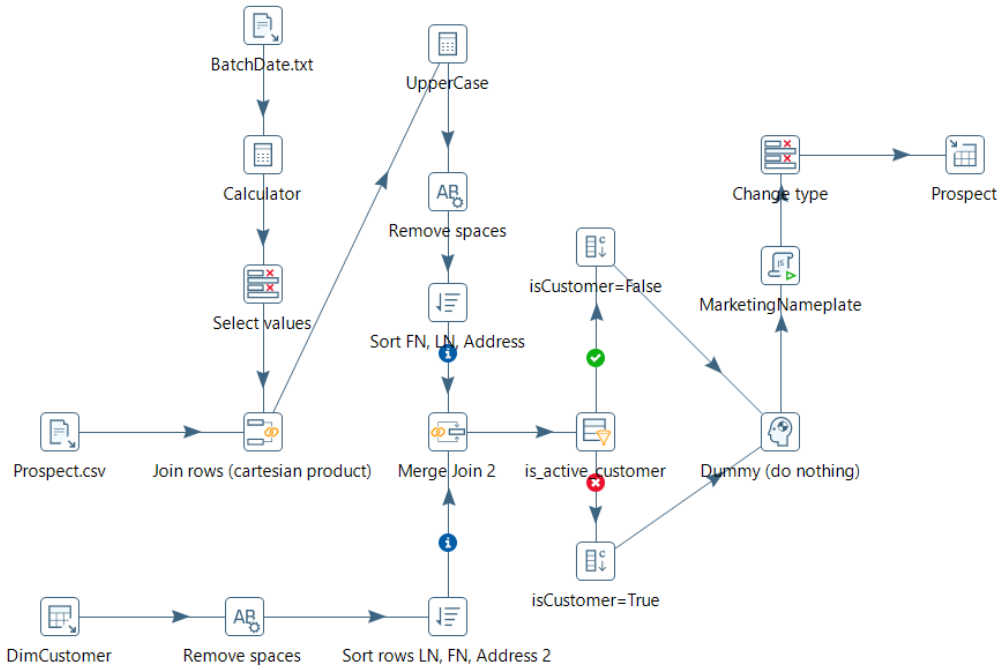


Figure 2.14: Historical Prospect Table Load

2.6.13 DimAccount

Every change in the customer should also be reflected in the account. The transformation details for loading historical data into the table DimAccount is provided below.

1. **CustomerMgmt.xml** file is used to load data into **DimAccount** Table.
2. Customer and Broker information is obtained from DimCustomer and DimBroker table respectively.
3. When ./@ActionType is 'NEW' or 'ADDACCT', new record is added into table.
4. When ./@ActionType is 'UPDACCT', a new entry is added into DIMACCOUNT with ISCURRENT = 1 and previous record with ISCURRENT = 1 for this account is updated to 0.
5. When ./@ActionType is 'CLOSEACCT', Status of account is set to Inactive.
6. When ./@ActionType is 'UPDCUST', For each account held by the customer being updated, update is performed to set SK_CustomerID to the associated customer's DimCustomer current record after it has been updated.
7. When ./@ActionType is 'INACT', Status is set to Inactive and for each account held by the customer being marked as inactive, update is performed to set SK_CustomerID to the associated customer's DimCustomer record after it has been marked 'INACTIVE'.

Figure 2.15 shows transformation prepared in order to load historical data into DimAccount table.

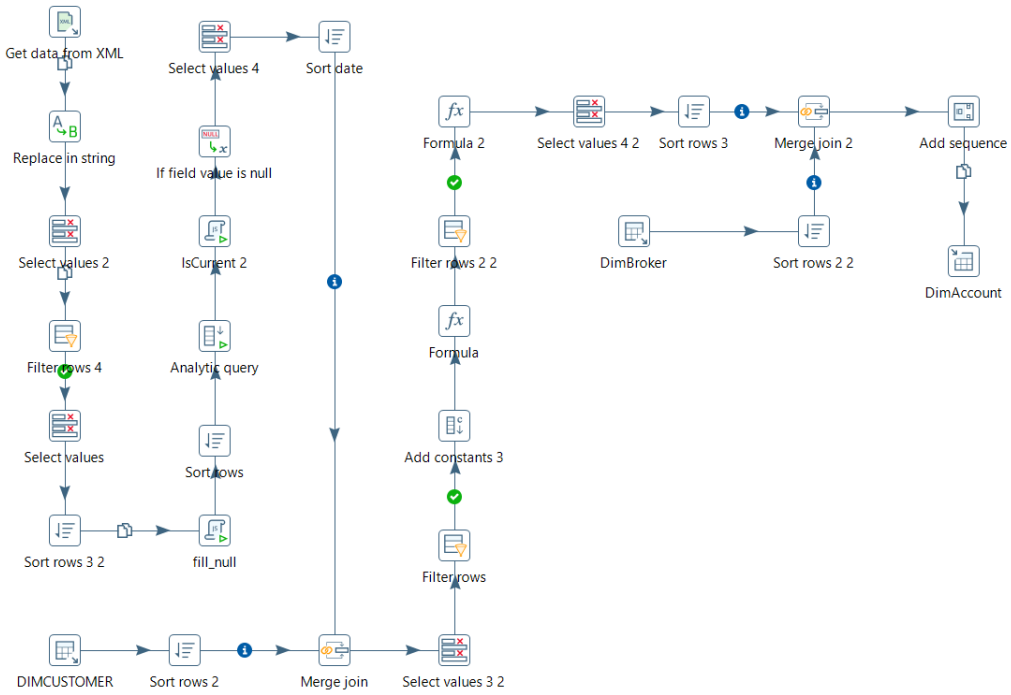


Figure 2.15: Historical DimAccount Table Load

2.6.14 FactCashBalances

The transformation details for loading historical data into the table FactCashBalances is provided below.

1. **CashTransaction.txt** file is used to load data into **FactCashBalances** table.
2. The DimAccount table is used to fetch Account details.
3. Cash Transactions are totalled on a daily level per account. Cash is calculated as the sum of the prior Cash amount for this account plus the sum of all CT_AMT values from all transactions in this account on this day. If there is no previous FactCashBalances record for the associated account, zero is used.

Figure 2.16 shows transformation prepared in order to load historical data into FactCashBalances table.

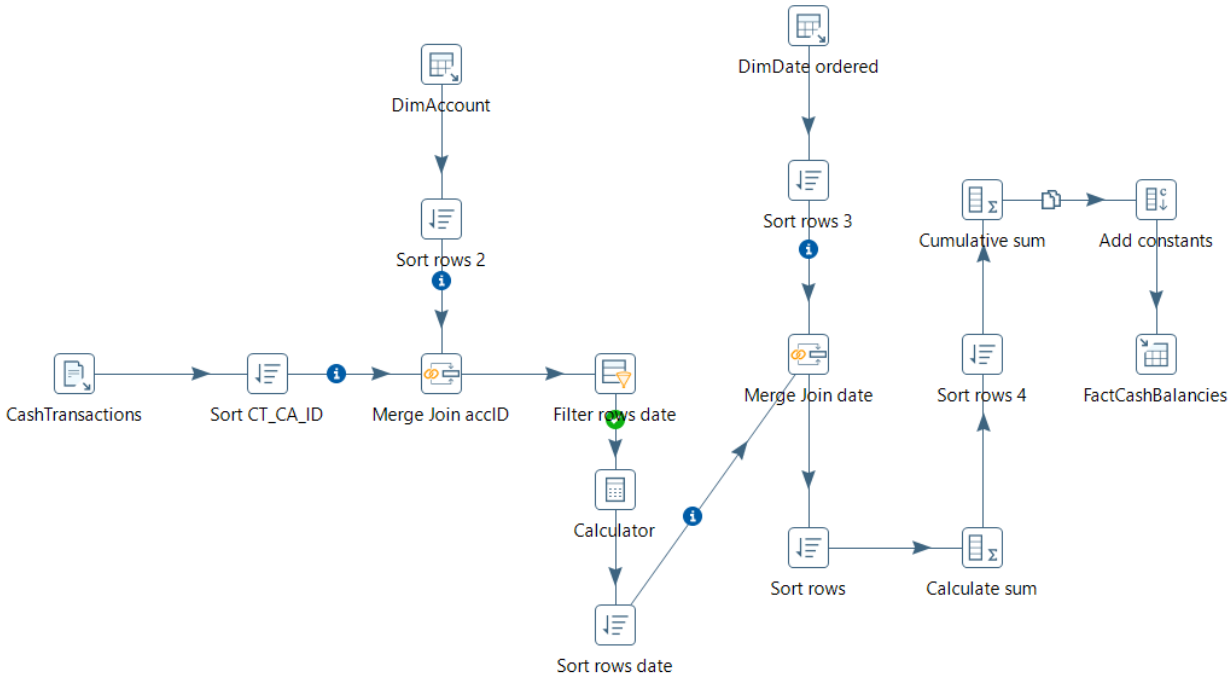


Figure 2.16: Historical FactCashBalances Table Load

2.6.15 FactWatches

The transformation details for loading historical data into the table FactWatches is provided below.

1. The **WatchHistory.txt** file is used to load data into the **FactWatches** table.
2. Customer and Security information are obtained from the DimCustomer and DimSecurity table respectively.
3. When W_ACTION= “ACTIV”, Records are inserted into the FactWatches Table with
   (a) SK_DateID_DatePlaced – set based on W_DTS
   (b) SK_DateID_DateRemoved – set to NULL
4. When W_ACTION = “CNCL”, No new record is inserted, instead, existing record is updated, i.e. SK_DateID_DateRemoved is set based on W_DTS.

Figure 2.17 shows transformation prepared in order to load historical data into FactWatches table.

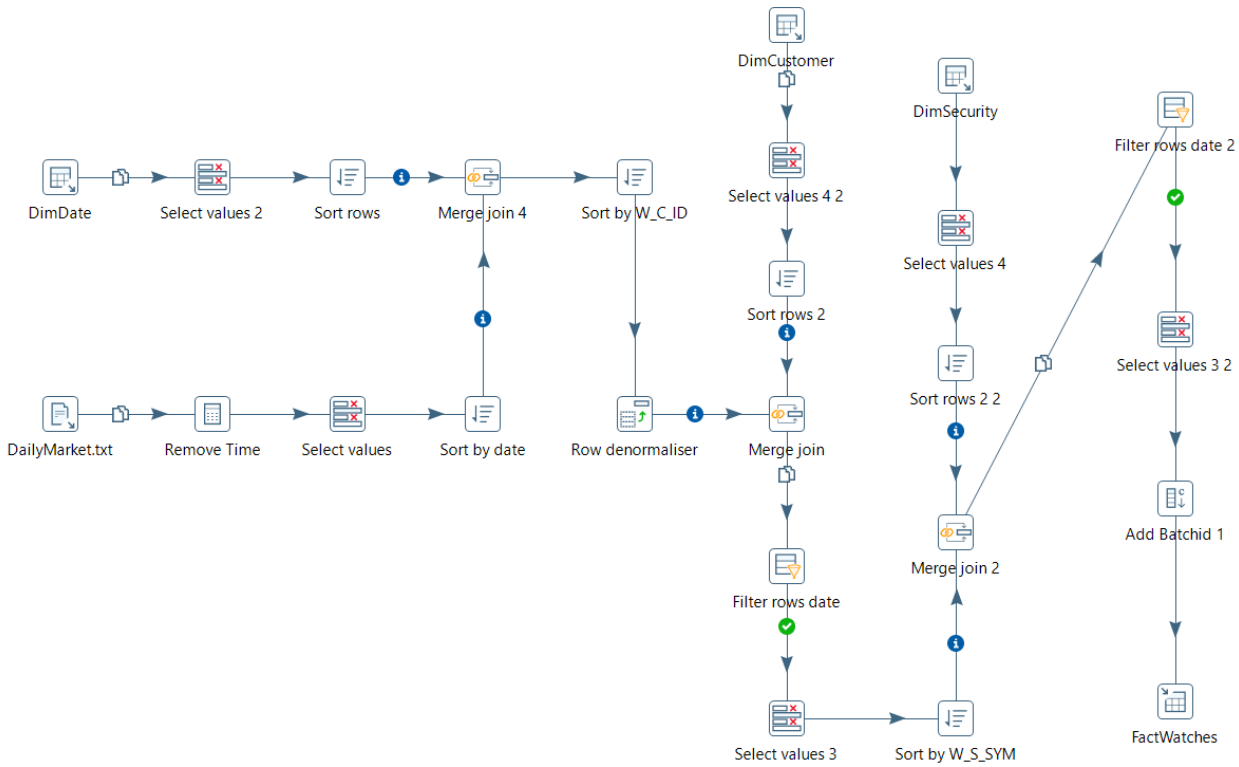


Figure 2.17: Historical FactWatches Table Load

2.6.16 FactHoldings

The transformation details for loading historical data into the table FactHoldings is provided below.

1. The **HoldingHistory.txt** file and the DimTrade table are used to load data into the **FactWatches** table.
2. SK_CustomerID, SK_AccountID, SK_SecurityID, SK_CompanyID and CurrentPrice are extracted from DimTrade Table using HH_T_ID field.
3. SK_DateID is set to the value of SK_CloseDateID.
4. SK_TimeID is set to the value of SK_CloseTimeID.
5. TradeId, CurrentTradeID, CurrentHolding are equivalent to HH_H_T_ID, HH_T_ID and HH_AFTER_QTY respectively.
6. BatchID is set to 1.

Figure 2.18 shows transformation prepared in order to load historical data into FactHoldings table.

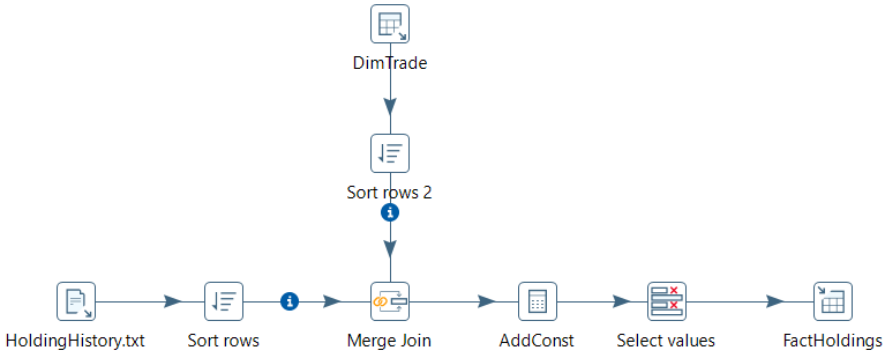


Figure 2.18: Historical FactHoldings Table Load

2.6.17 FactMarketHistory

The transformation details for loading historical data into the table FactMarketHistory is provided below.

1. The **DailyMarket.txt** file is used to load data into the **FactMarketHistory** table.
2. ClosePrice, DayHigh, DayLow, and Volume are copied from DM_CLOSE, DM_HIGH, DM_LOW, and DM_VOL respectively.
3. SK_SecurityID is obtained from DimSecurity table.
4. SK_CompanyID is obtained from DimSecurity table.
5. SK_DateID is obtained from DimDate by matching DM_DATE with DateValue.
6. FiftyTwoWeekHigh and SK_FiftyTwoWeekHighDate are determined by finding the highest price over the last year (approximately 52 weeks) for a given security.
7. FiftyTwoWeekLow and SK_FiftyTwoWeekLowDate are determined by finding the lowest price over the last year (approximately 52 weeks) for a given security.
8. PERatio is calculated by dividing DM_CLOSE by the sum of the company's quarterly earnings per share over the previous 4 quarters prior to DM_DATE.
9. Yield is calculated by dividing the security's dividend by DM_CLOSE, then multiplying by 100.
10. The dividend is obtained from DimSecurity by matching DM_S_SYMB with Symbol, where DM_DATE is in the range given by EffectiveDate and EndDate.
11. BatchID is set to 1.

Figure 2.19 shows transformation prepared in order to load historical data into FactMarketHistory table.

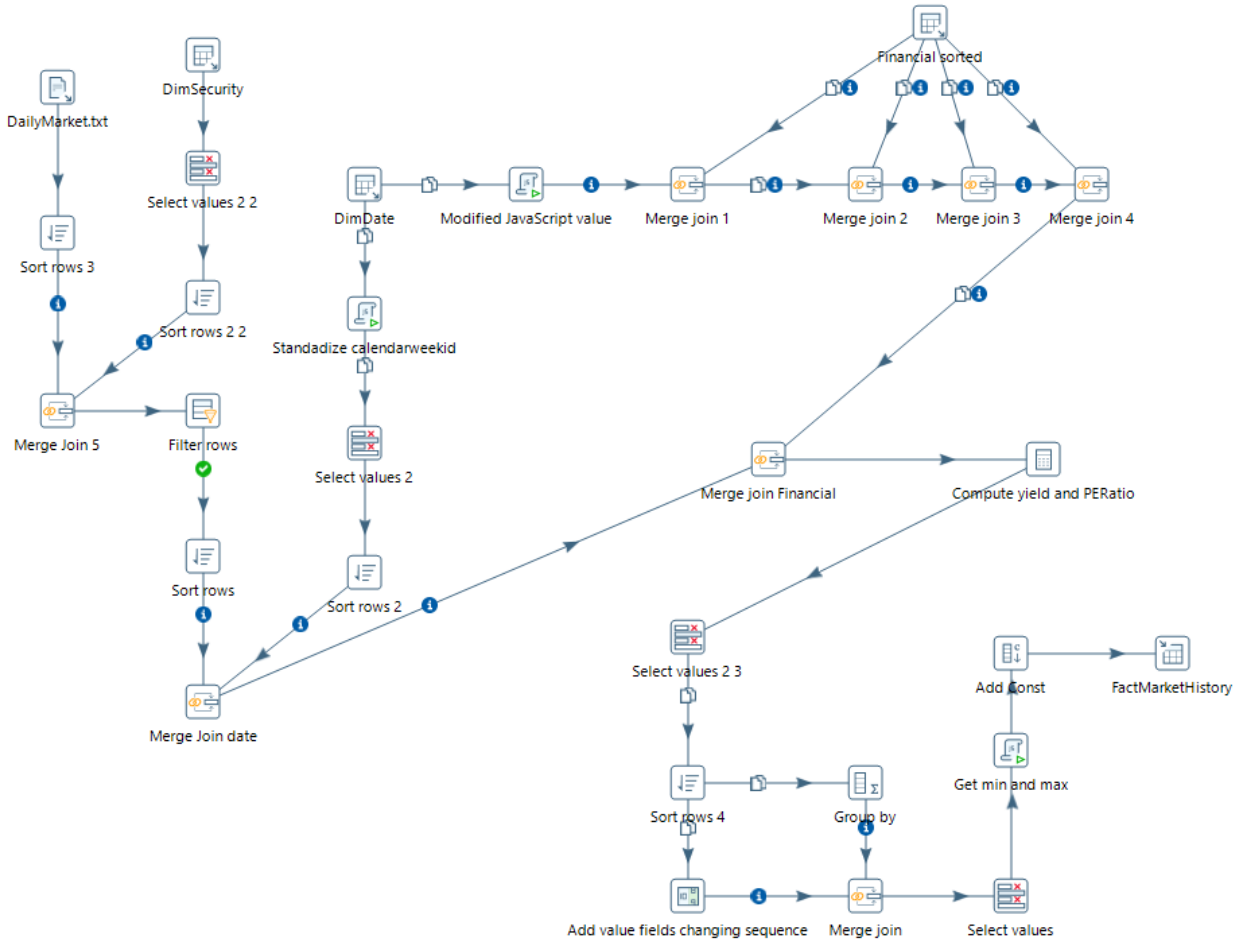


Figure 2.19: Historical FactMarketHistory Table Load

2.7 Transformations for Incremental Data Load

2.7.1 DimAccount

The transformation details for the incremental load of the table DimAccount is provided below.

1. The **Account.txt** file is used to load data into **DimAccount** table.
2. When CDC_FLAG is 'I', a new DimAccount record is inserted.
3. When CDC_FLAG is 'U', the updates to DimAccount are implemented in a history-tracking manner.
4. AccountID, AccountDesc and TaxStatus fields are equivalent to CA_ID, CA_NAME and CA_TAX_ST respectively.

5. SK_BrokerID and SK_CustomerID are set by obtaining surrogate keys by matching CA_B_ID with DimBroker and DimCustomer respectively.

6. Status is obtained from StatusType table.

7. BatchID is set to 2.

Figure 2.20 shows transformation prepared in order to load incremental data into DimAccount table.

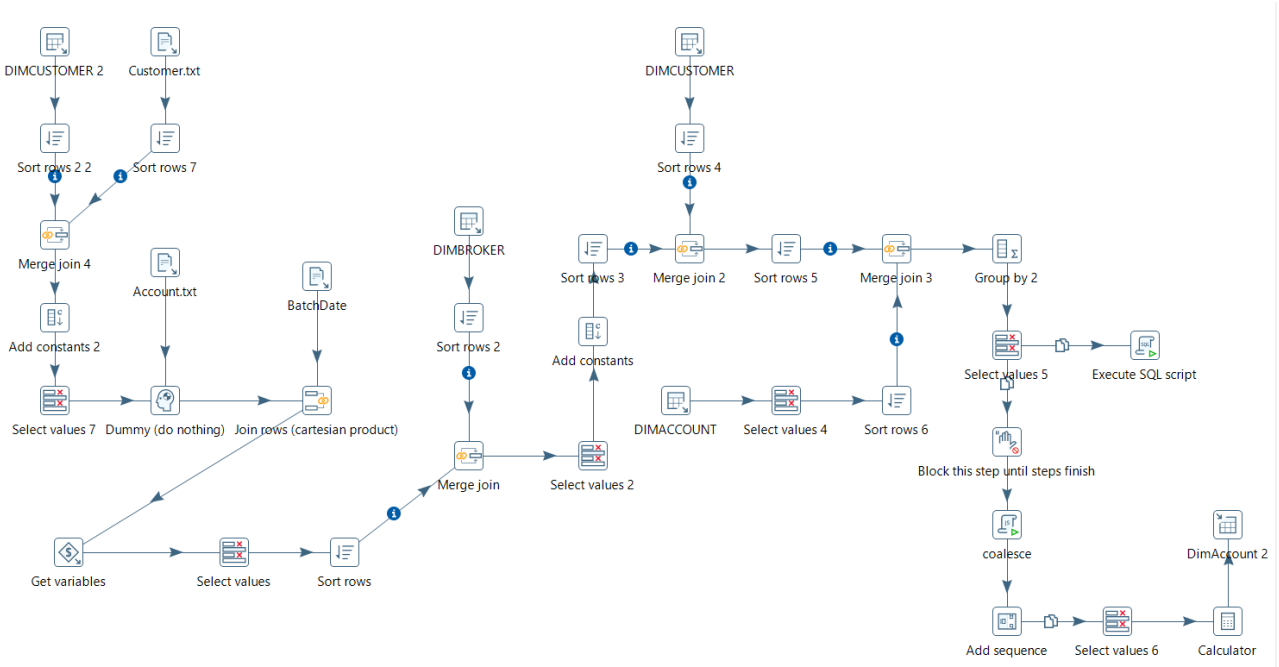


Figure 2.20: Incremental DimAccount Table Load

2.7.2 DimCustomer

The transformation details for the incremental load of the table DimCustomer is provided below.

1. The **Customer.txt** file is used to load data into **DimCustomer** table.

2. When CDC_FLAG is 'I', a new DimCustomer record is inserted.

3. When CDC_FLAG is 'U', the updates to DimCustomer are implemented in a history-tracking manner.

4. CustomerID, TaxID, LastName, FirstName, MiddleInitial, Tier, DOB, Email1 and Email2 are copied from C_ID, C_TAX_ID, C_L_NAME, C_F_NAME, C_M_NAME, C_TIER, C_DOB, C_EMAIL_1, C_EMAIL_2 respectively.

5. Gender is uppercased and values other than M and F are populated as U.
6. Phone Number is transformed as:
   (a) PhoneNumber = '+C_CTRY_CODE' + '(C_AREA_CODE)' + 'C_LOCAL' + 'C_EXT', where only provided parts are added.
   (b) If C_CTRY_CODE is not null, display C_CTRY_CODE as +C_CTRY_CODE
   (c) If C_AREA_CODE is not null, display C_AREA_CODE as (C_AREA_CODE)
7. AddressLine1, AddressLine2, PostalCode, City, StateProv, and Country are copied from C_ADLINE1, C_ADLINE2, C_ZIPCODE, C_CITY, C_STATE_PROV, and C_CTRY.
8. Status is obtained from StatusType table.
9. NationalTaxRateDesc, NationalTaxRate, LocalTaxRateDesc and LocalTaxRate are obtained from table TAX_RATE.
10. AgencyID, CreditRating, NetWorth, MarketingNameplate are obtained from Prospect File.
11. BatchId is set to 2.

While preparing the transformation for the DimCustomer table, we need to consider the following points.

1. Invalid Customer Tier values i.e. value not in (1,2,3) should be reported in DIMessages Table with:
   (a) MessageType = 'Alert'
   (b) MessageText = 'Invalid customer tier'
   (c) MessageSource = 'DimCustomer'
   (d) MessageData = 'C_ID =' + SK_CUSTOMERID + ', C_TIER = ' + C_TIER
2. Invalid DOB i.e. if DOB < Batch Date – 100 years or DOB > Batch Date should be reported in DIMessages Table with:
   (a) MessageType = 'Alert'
   (b) MessageText = 'DOB out of range'
   (c) MessageSource = 'DimCustomer'

(d) MessageData = 'C_ID =' + SK_CUSTOMERID + ', C_DOB = ' + C_DOB

Figure 2.21 shows transformation prepared in order to load incremental data into DimCustomer table.

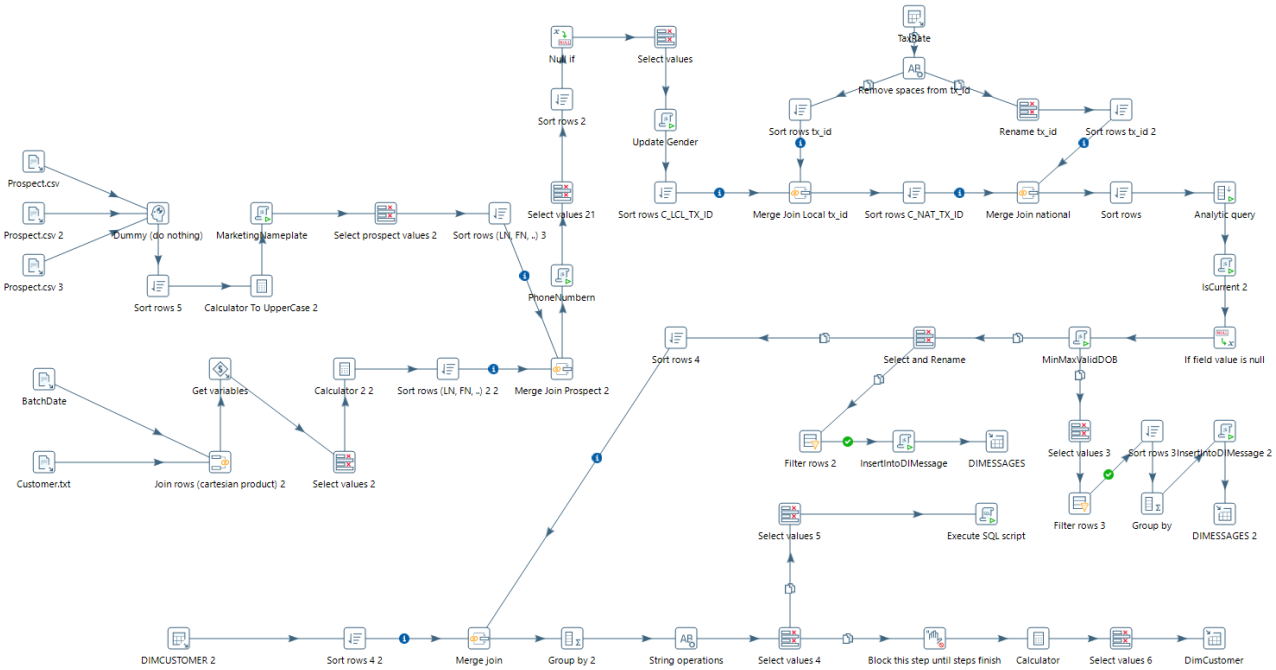


Figure 2.21: Incremental DimCustomer Table Load

2.7.3 DimTrade

The transformation details for incremental load of the table DimTrade is provided below.

1. The **Trade.txt** file is used to load data into **DimTrade** table.
2. When CDC_FLAG is 'I', a new DimTrade record is inserted
3. When CDC_FLAG is 'U', the existing DimTrade record whose TradeID matches the incoming T_ID is updated.
4. For new trade record, SK_CreateDateID and SK_CreateTimeID are set based on T_DTS, while SK_CloseDateID and SK_CloseTimeID are set to NULL.
5. If T_ST_ID is "CMP" or "CNCL", SK_CloseDateID and SK_CloseTimeID are set based on T_DTS.
6. Status and Type are obtained from StatusType and TradeType respectively.

7. SK_SecurityID and SK_CompanyID are obtained from DimSecurity Table.
8. SK_AccountID, SK_CustomerID, and SK_BrokerID are obtained from DimAccount Table.
9. BatchID is set to 2 and 3 for first and second incremental load respectively.

While preparing the transformation for the DimTrade table, we need to consider the following points.

1. Invalid trade commission should be reported in DIMessages Table with:
   (a) MessageType = 'Alert'
   (b) MessageText = 'Invalid trade commission'
   (c) MessageSource = 'DimTrade'
   (d) MessageData = 'T_ID =' + T_ID + ', T_COMM = ' + T_COMM
2. Invalid trade fee should be reported in DIMessages Table with:
   (a) MessageType = 'Alert'
   (b) MessageText = 'Invalid trade fee'
   (c) MessageSource = 'DimTrade'
   (d) MessageData = 'T_ID =' + T_ID + ', T_CHRG = ' + T_CHRG

Figure 2.22 shows transformation prepared in order to load incremental data into DimTrade table.

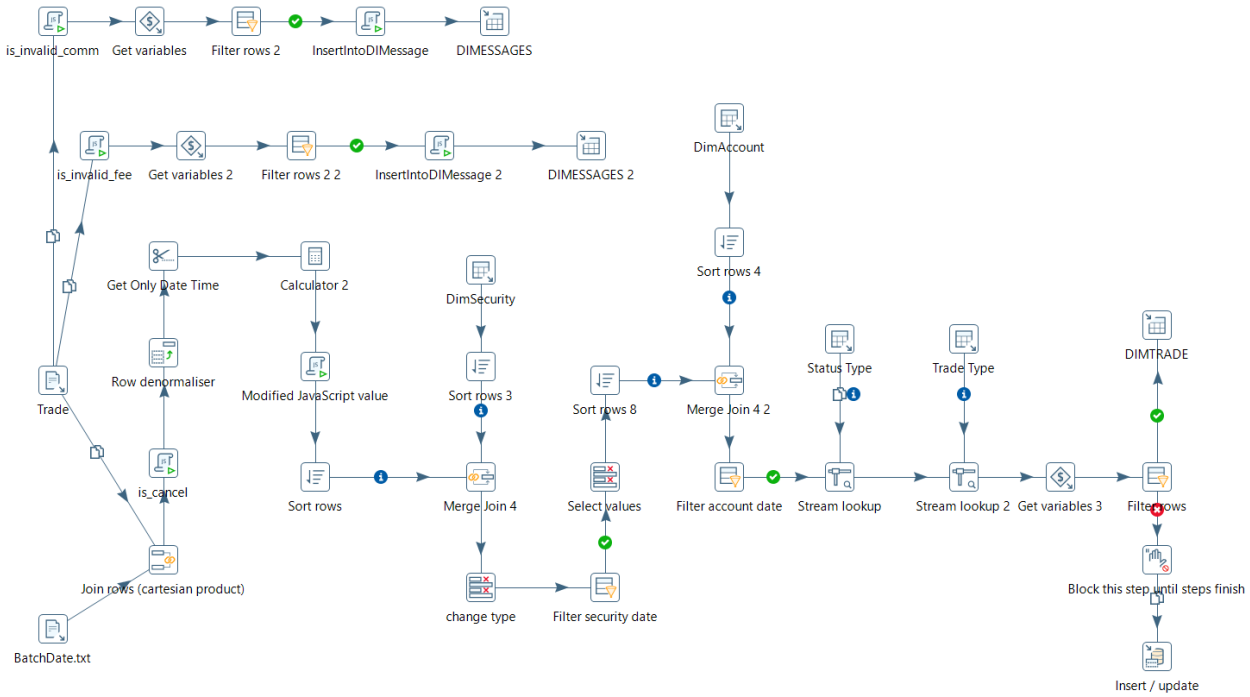


Figure 2.22: Incremental DimTrade Table Load

2.7.4 FactCashBalances

The transformation details for the table FactCashBalances is provided below.

1. The **CashTransaction.txt** file is used to load data into **FactCashBalances** table.
2. The DimAccount table is used to fetch Account details.
3. SK_DateID is set to the DimDate SK_DateID field that corresponds to the Batch Date.
4. Cash Transactions are totalled on a daily level per account. Cash is calculated as the sum of the prior Cash amount for this account plus the sum of all CT_AMT values from all transactions in this account on this day. If there is no previous FactCashBalances record for the associated account, zero is used.

Figure 2.23 shows transformation prepared in order to load incremental data into FactCashBalances table.

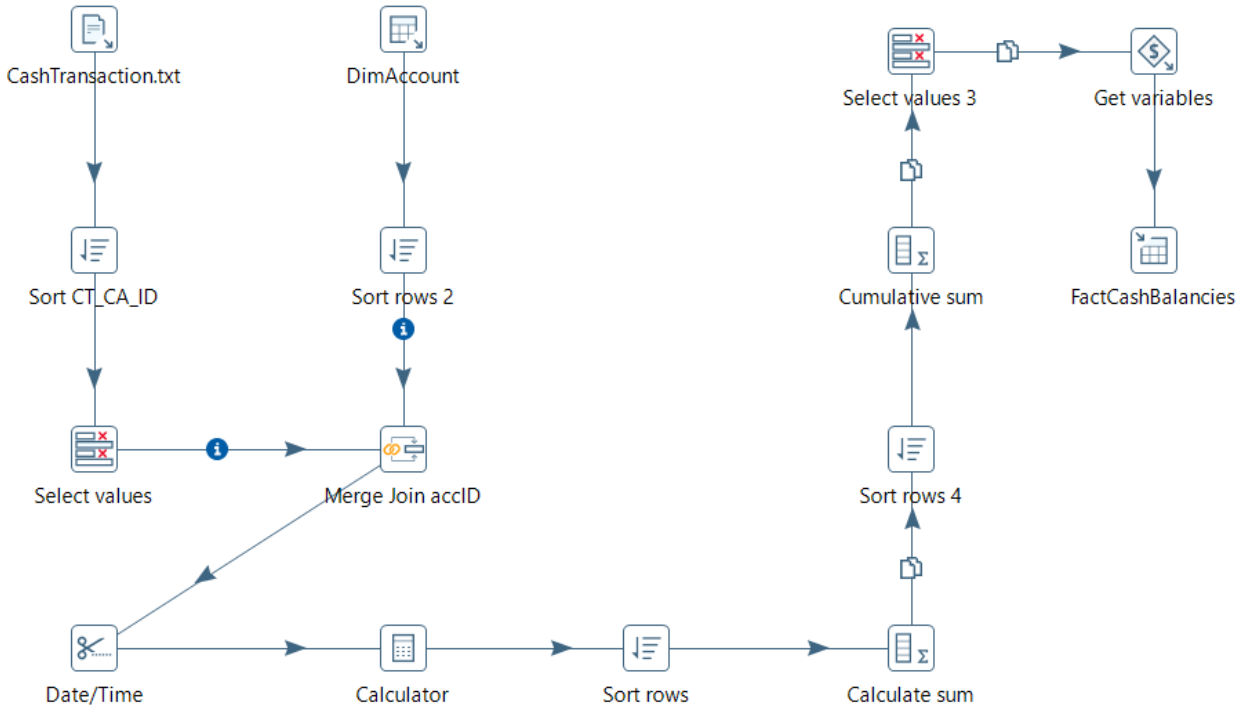


Figure 2.23: Incremental FactCashBalances Table Load

2.7.5 FactHoldings

The transformation details for the table FactHoldings is provided below.

- The **HoldingHistory.txt** file and the DimTrade table are used to load data into the **FactWatches** table.
- SK_CustomerID, SK_AccountID, SK_SecurityID, SK_CompanyID and CurrentPrice are extracted from DimTrade Table using HH_T_ID field.
- TradeId, CurrentTradeID, CurrentHolding are equivalent to HH_H_T_ID, HH_T_ID and HH_AFTER_QTY respectively.
- BatchID is set to 2 and 3 depending on whether first or second incremental load is performed respectively.

Figure 2.24 shows transformation prepared in order to load incremental data into FactHoldings table.

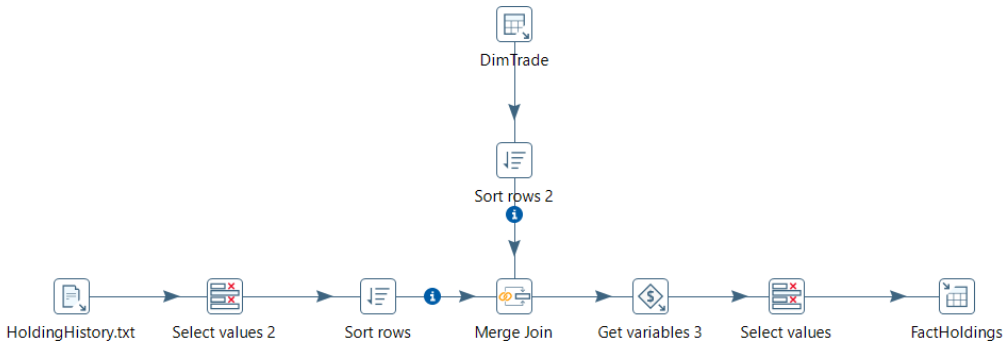


Figure 2.24: Incremental FactHoldings Table Load

2.7.6 FactMarketHistory

The transformation details for incremental load of the table FactMarketHistory is provided below.

1. The **DailyMarket.txt** file is used to load data into the **FactMarketHistory** table.
2. ClosePrice, DayHigh, DayLow, and Volume are copied from DM_CLOSE, DM_HIGH, DM_LOW, and DM_VOL respectively.
3. SK_SecurityID is obtained from DimSecurity table.
4. SK_CompanyID is obtained from DimSecurity table.
5. SK_DateID is obtained from DimDate by matching DM_DATE with DateValue.
6. FiftyTwoWeekHigh and SK_FiftyTwoWeekHighDate are determined by finding the highest price over the last year (approximately 52 weeks) for a given security.
7. FiftyTwoWeekLow and SK_FiftyTwoWeekLowDate are determined by finding the lowest price over the last year (approximately 52 weeks) for a given security.
8. PERatio is calculated by dividing DM_CLOSE by the sum of the company's quarterly earnings per share over the previous 4 quarters prior to DM_DATE.
9. Yield is calculated by dividing the security's dividend by DM_CLOSE, then multiplying by 100.
10. The dividend is obtained from DimSecurity by matching DM_S_SYMB with Symbol, where DM_DATE is in the range given by EffectiveDate and EndDate.

11. BatchID is set to 2 or 3 depending on whether first or second incremental load is being performed repectively.

12. If there are no earnings found for a company, such records are reported in the DImessages table with:

    (a) MessageType = 'Alert'

    (b) MessageText = 'No earnings for company'

    (c) MessageSource = 'FactMarketHistory'

    (d) MessageData = 'DM_S_SYMB =' + DM_S_SYMB

Figure 2.25 shows transformation prepared in order to load incremental data into FactMarketHistory table.

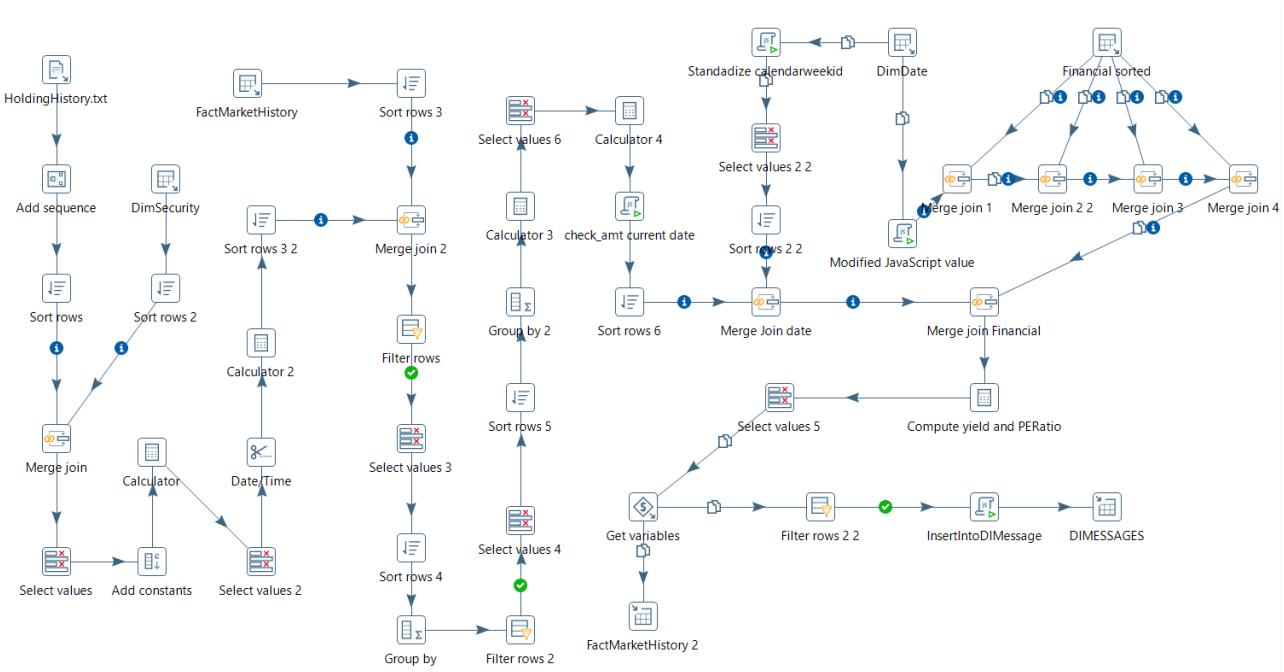


Figure 2.25: Incremental FactMarketHistory Table Load

2.7.7 FactWatches

The transformation details for incremental load of the table FactWatches is provided below.

1. The **WatchHistory.txt** file is used to load incremental data into **FactWatches** table.

2. When a security is added to a watch list, as indicated by W_ACTION = 'ACTV', SK_DateID_DatePlaced is set based on W_DTS and SK_DateID_DateRemoved is set to NULL.

3. When a security is removed from a watch list, as indicated by W_ACTION = 'CNCL', SK_DateID_DateRemoved is set based on W_DTS.

4. BatchID is set to 2 and 3 depending on whether first or second incremental load is performed respectively.

Figure 2.26 shows transformation prepared in order to load incremental data into FactWatches table.

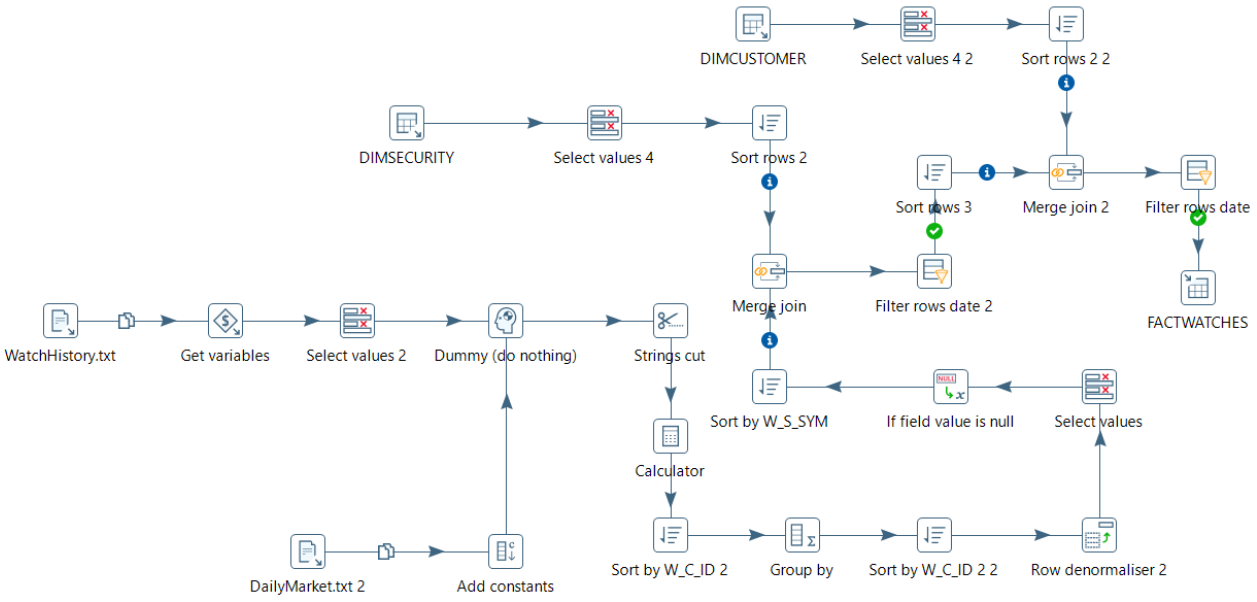


Figure 2.26: Incremental FactWatches Table Load

2.7.8 Prospect

The transformation details for the table Prospect is provided below.

1. The **Prospect.csv** file is used to load incremental data into **Prospect** table.

2. SK_RecordDateID is set to the DimDate SK_DateID field that corresponds to the Batch Date.

3. SK_UpdateDateID is set to the DimDate SK_DateID field that Batch Date if this is the first time this AgencyID value has appeared in the Prospect file or if this AgencyID value has appeared before and the values of any of the following fields are different from prior saved values for the same AgencyID value in the Prospects table: LastName, FirstName, MiddleInitial, Gender, AddressLine1, AddressLine2, PostalCode, City, State, Country, Phone, Income, NumberCars, NumberChildren, MaritalStatus, Age, CreditRating, OwnOrRentFlag,

Employer, NumberCreditCards, NetWorth. Otherwise, SK_UpdateDateID retains its prior saved value.

4. If the prospective customer record matches a current customer record in DimCustomer whose status is 'ACTIVE', then IsCustomer is set to 1, otherwise set to 0. A Prospect record is deemed to match a DimCustomer record if the FirstName, LastName, AddressLine1, AddressLine2 and PostalCode fields all match when upper-cased.

5. MarketingNameplate is set by concatenating zero or more tags with a "+" character between them if multiple tags apply to a given customer. For example, a prospect that qualifies for both the "Boomer" tag and the "Spender" tag would be assigned the MarketingNameplate value "Boomer+Spender". If multiple tags are used they must be in the order given below, and if no tags apply the nameplate is NULL.

6. The tags are defined as:

   (a) HighValue: NetWorth > 1000000 or Income > 200000

   (b) Expenses: NumberChildren > 3 or NumberCreditCards > 5

   (c) Boomer: Age > 45

   (d) MoneyAlert: Income < 50000 or CreditRating < 600 or NetWorth < 100000

   (e) Spender: NumberCars > 3 or NumberCreditCards > 7

   (f) Inherited: Age < 25 and NetWorth > 1000000

7. BatchID is set to 2 and 3 depending on whether first or second incremental load is performed respectively.

Figure 2.27 shows transformation prepared in order to load incremental data into Prospect table.

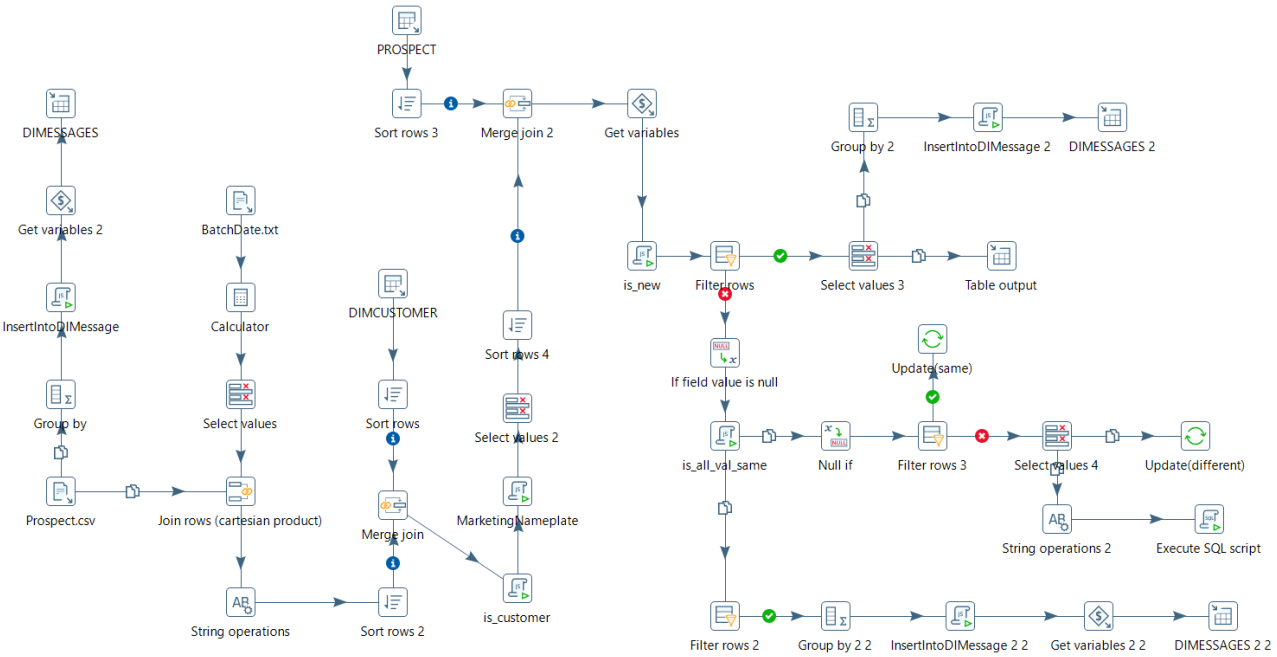


Figure 2.27: Incremental Prospect Table Load

2.7.9 Tables without incremental data

For following tables, changes do not occur during Incremental Updates.

1. DimBroker
2. DimCompany
3. DimDate
4. DimSecurity
5. Industry
6. Financial
7. StatusType
8. TaxRate
9. TradeType

2.8 Integration of Transformations

Once all the transformations required to load historical and incremental data into dimension and fact tables were prepared, jobe were created to organize them in proper order. Figure 2.28, 2.29

and 2.30 shows job prepared in order to load all historical and incremental tables.

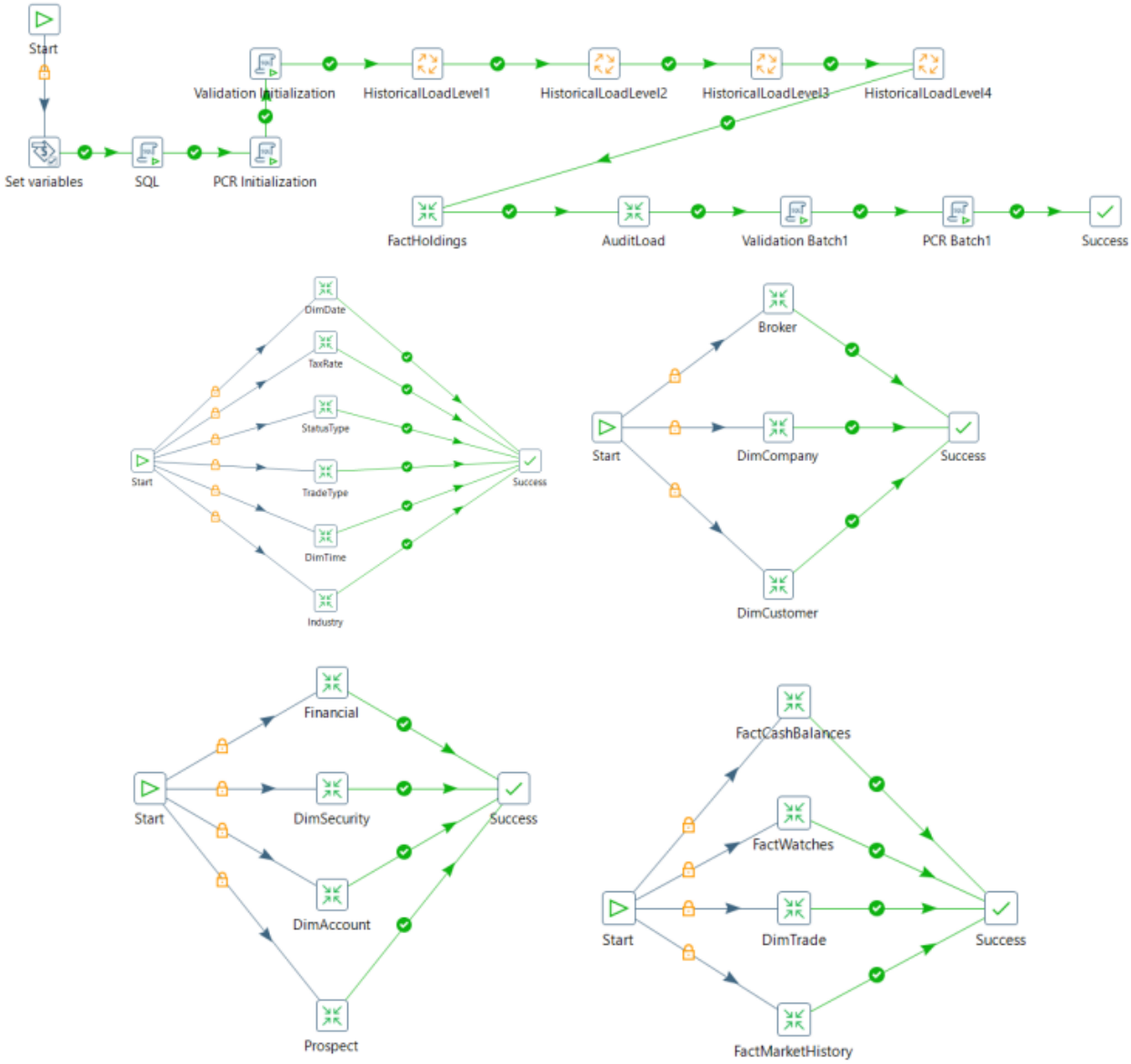


Figure 2.28: PDI Job for Historical Data Load

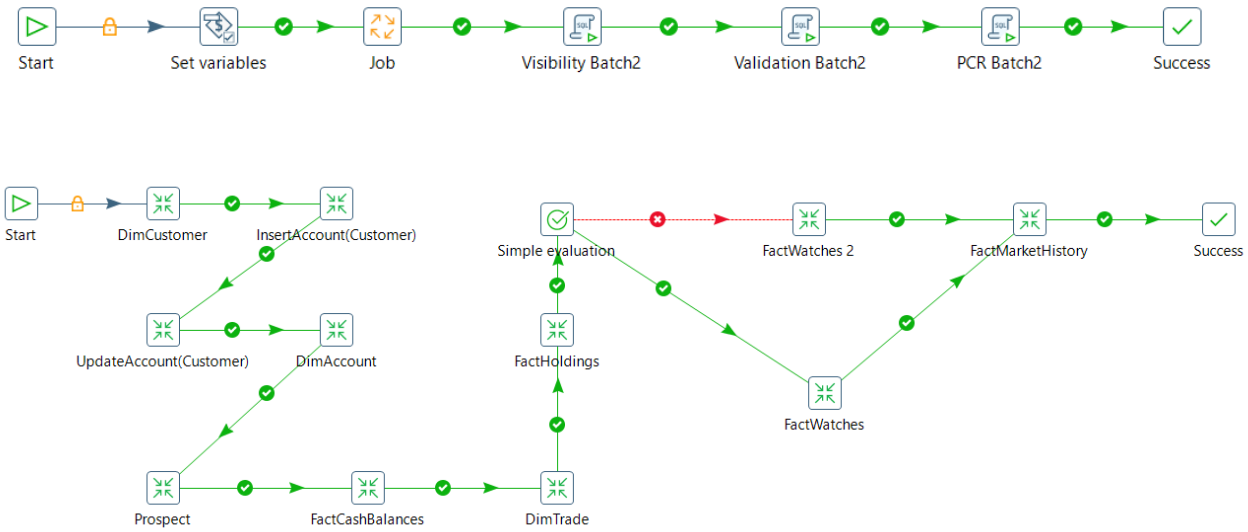


Figure 2.29: PDI Job for Incremental Data Load

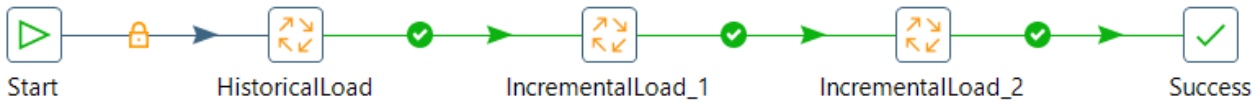


Figure 2.30: PDI Job for Data Load

2.9 Validation of Data Load

Validation of Historical and Incremental data loaded using jobs and transformations prepared in PDI is the must. The correctness of these data were tested using the SQL Script provided by Professor through mail.

Note: Slight modifications were made in Query to make it compatible with Oracle Syntax. You can find the modified query on the GitHub link provided here.

The summary of changes made in query is provided below.

1. Extra space was added in COALESCE statement

```
-- Original Query
SELECT
    'DimCustomer demographic fields',
    NULL,
    CASE WHEN (
        SELECT
            COUNT(*)
```

```
        FROM DIMCUSTOMER c
        JOIN PROSPECT p
        ON UPPER(c.FirstName || c.LastName || c.AddressLine1 ||
           COALESCE(c.AddressLine2,'') || c.PostalCode) = UPPER(p.
           FirstName || p.LastName || p.AddressLine1 || COALESCE(p
           .AddressLine2,'') || p.PostalCode)
        AND COALESCE(c.CreditRating,0) = COALESCE(p.CreditRating
           ,0)
        AND COALESCE(c.NetWorth,0) = COALESCE(p.NetWorth,0)
        AND COALESCE(c.MarketingNameplate, '') = COALESCE(p.
           MarketingNameplate,'')
        AND c.IsCurrent = '1'
    ) = (
        SELECT
            COUNT(*)
        FROM DIMCUSTOMER
        WHERE AgencyID IS NOT NULL
        AND IsCurrent = '1'
    )
THEN 'OK' ELSE 'Mismatch' END, 'For current customer records that
   match Prospect records, the demographic fields also match'
FROM dual;
```

```
-- Modified Query
SELECT
    'DimCustomer demographic fields',
    NULL,
    CASE WHEN (
        SELECT
            COUNT(*)
        FROM DIMCUSTOMER c
        JOIN PROSPECT p
        ON UPPER(c.FirstName || c.LastName || c.AddressLine1 ||
           COALESCE(c.AddressLine2,'') || c.PostalCode) = UPPER(p.
           FirstName || p.LastName || p.AddressLine1 || COALESCE(p
           .AddressLine2,'') || p.PostalCode)
```

```
        AND COALESCE(c.CreditRating,0) = COALESCE(p.CreditRating
            ,0)
        AND COALESCE(c.NetWorth,0) = COALESCE(p.NetWorth,0)
        AND COALESCE(c.MarketingNameplate, ' ')
        = COALESCE(p.MarketingNameplate,' ')
        AND c.IsCurrent = '1'
    ) = (
        SELECT
            COUNT(*)
        FROM DIMCUSTOMER
        WHERE AgencyID IS NOT NULL
        AND IsCurrent = '1'
    )
THEN 'OK' ELSE 'Mismatch' END, 'For current customer records that
   match Prospect records, the demographic fields also match'
FROM dual;
```

2. Add Truncate function in Date Comparison

```
-- Original Query
select 'FactCashBalances SK_CustomerID', NULL, case when
        (select count(*) from FactCashBalances) =
        (
                select
                        count(*)
                from FactCashBalances a
                join DimCustomer c
                on a.SK_CustomerID = c.SK_CustomerID and
                c.EffectiveDate <= (
                        select
                                DateValue
                        from DimDate
                        where SK_DateID = a.SK_DateID
                ) and (
                        select
                                DateValue
```

```
                        from DimDate
                        where SK_DateID = a.SK_DateID) <= c.
                            EndDate
                )
then 'OK' else 'Bad join' end,
'All SK_CustomerIDs match a DimCustomer record with a valid date
    range'
FROM dual;
```

```
-- Modified Query
select 'FactCashBalances SK_CustomerID', NULL, case when
        (select count(*) from FactCashBalances) =
        (
                select
                        count(*)
                from FactCashBalances a
                join DimCustomer c
                on a.SK_CustomerID = c.SK_CustomerID and
                 TRUNC(c.EffectiveDate) <= (
                        select
                                DateValue
                        from DimDate
                        where SK_DateID = a.SK_DateID
                ) and (
                        select
                                DateValue
                        from DimDate
                        where SK_DateID = a.SK_DateID) <=
                            TRUNC(c.EndDate)
                )
then 'OK' else 'Bad join' end,
'All SK_CustomerIDs match a DimCustomer record with a valid date
    range'
FROM dual;
```

2.10 Validation Result

All test cases passed, confirming the accuracy of the prepared transformations and jobs.

	TEST	BATCH	RESULT	DESCRIPTION
1	"AUDIT" table batches	[NULL]	OK	There must be "AUDIT" data for 3 batches
2	"AUDIT" table sources	[NULL]	OK	There must be "AUDIT" data for all data sets
3	Batch joined row count: FactCashBalances	1	OK	Row counts match when joined to dimensions
4	Batch joined row count: FactCashBalances	2	OK	Row counts match when joined to dimensions
5	Batch joined row count: FactCashBalances	3	OK	Row counts match when joined to dimensions
6	Batch joined row count: FactHoldings	1	OK	Row counts match when joined to dimensions
7	Batch joined row count: FactHoldings	2	OK	Row counts match when joined to dimensions
8	Batch joined row count: FactHoldings	3	OK	Row counts match when joined to dimensions
9	Batch joined row count: FactMarketHistory	1	OK	Row counts match when joined to dimensions
10	Batch joined row count: FactMarketHistory	2	OK	Row counts match when joined to dimensions
11	Batch joined row count: FactMarketHistory	3	OK	Row counts match when joined to dimensions
12	Batch joined row count: FactWatches	1	OK	Row counts match when joined to dimensions
13	Batch joined row count: FactWatches	2	OK	Row counts match when joined to dimensions
14	Batch joined row count: FactWatches	3	OK	Row counts match when joined to dimensions
15	Batch row count: DimAccount	1	OK	Row counts do not decrease between successive batches
16	Batch row count: DimAccount	2	OK	Row counts do not decrease between successive batches
17	Batch row count: DimAccount	3	OK	Row counts do not decrease between successive batches
18	Batch row count: DimBroker	1	OK	Row counts do not decrease between successive batches
19	Batch row count: DimBroker	2	OK	Row counts do not decrease between successive batches
20	Batch row count: DimBroker	3	OK	Row counts do not decrease between successive batches
21	Batch row count: DimCompany	1	OK	Row counts do not decrease between successive batches
22	Batch row count: DimCompany	2	OK	Row counts do not decrease between successive batches
23	Batch row count: DimCompany	3	OK	Row counts do not decrease between successive batches
24	Batch row count: DimCustomer	1	OK	Row counts do not decrease between successive batches
25	Batch row count: DimCustomer	2	OK	Row counts do not decrease between successive batches
26	Batch row count: DimCustomer	3	OK	Row counts do not decrease between successive batches
27	Batch row count: DimDate	1	OK	Row counts do not decrease between successive batches
28	Batch row count: DimDate	2	OK	Row counts do not decrease between successive batches
29	Batch row count: DimDate	3	OK	Row counts do not decrease between successive batches
30	Batch row count: DimSecurity	1	OK	Row counts do not decrease between successive batches

Figure 2.31: Test Cases Execution Result 1

	TEST	BATCH	RESULT	DESCRIPTION
31	Batch row count: DimSecurity	2	OK	Row counts do not decrease between successive batches
32	Batch row count: DimSecurity	3	OK	Row counts do not decrease between successive batches
33	Batch row count: DimTime	1	OK	Row counts do not decrease between successive batches
34	Batch row count: DimTime	2	OK	Row counts do not decrease between successive batches
35	Batch row count: DimTime	3	OK	Row counts do not decrease between successive batches
36	Batch row count: DimTrade	1	OK	Row counts do not decrease between successive batches
37	Batch row count: DimTrade	2	OK	Row counts do not decrease between successive batches
38	Batch row count: DimTrade	3	OK	Row counts do not decrease between successive batches
39	Batch row count: FactCashBalances	1	OK	Row counts do not decrease between successive batches
40	Batch row count: FactCashBalances	2	OK	Row counts do not decrease between successive batches
41	Batch row count: FactCashBalances	3	OK	Row counts do not decrease between successive batches
42	Batch row count: FactHoldings	1	OK	Row counts do not decrease between successive batches
43	Batch row count: FactHoldings	2	OK	Row counts do not decrease between successive batches
44	Batch row count: FactHoldings	3	OK	Row counts do not decrease between successive batches
45	Batch row count: FactMarketHistory	1	OK	Row counts do not decrease between successive batches
46	Batch row count: FactMarketHistory	2	OK	Row counts do not decrease between successive batches
47	Batch row count: FactMarketHistory	3	OK	Row counts do not decrease between successive batches
48	Batch row count: FactWatches	1	OK	Row counts do not decrease between successive batches
49	Batch row count: FactWatches	2	OK	Row counts do not decrease between successive batches
50	Batch row count: FactWatches	3	OK	Row counts do not decrease between successive batches
51	Batch row count: Financial	1	OK	Row counts do not decrease between successive batches
52	Batch row count: Financial	2	OK	Row counts do not decrease between successive batches
53	Batch row count: Financial	3	OK	Row counts do not decrease between successive batches
54	Batch row count: Industry	1	OK	Row counts do not decrease between successive batches
55	Batch row count: Industry	2	OK	Row counts do not decrease between successive batches
56	Batch row count: Industry	3	OK	Row counts do not decrease between successive batches
57	Batch row count: Prospect	1	OK	Row counts do not decrease between successive batches
58	Batch row count: Prospect	2	OK	Row counts do not decrease between successive batches
59	Batch row count: Prospect	3	OK	Row counts do not decrease between successive batches
60	Batch row count: StatusType	1	OK	Row counts do not decrease between successive batches

Figure 2.32: Test Cases Execution Result 2

	TEST	BATCH	RESULT	DESCRIPTION
61	Batch row count: StatusType	2	OK	Row counts do not decrease between successive batches
62	Batch row count: StatusType	3	OK	Row counts do not decrease between successive batches
63	Batch row count: TaxRate	1	OK	Row counts do not decrease between successive batches
64	Batch row count: TaxRate	2	OK	Row counts do not decrease between successive batches
65	Batch row count: TaxRate	3	OK	Row counts do not decrease between successive batches
66	Batch row count: TradeType	1	OK	Row counts do not decrease between successive batches
67	Batch row count: TradeType	2	OK	Row counts do not decrease between successive batches
68	Batch row count: TradeType	3	OK	Row counts do not decrease between successive batches
69	DImessages Phase complete records	[NULL]	OK	Must have 4 Phase Complete records
70	DImessages batches	[NULL]	OK	Must have 3 distinct batches reported in DImessages
71	DImessages initial condition	[NULL]	OK	All DW tables must be empty before Batch1
72	DImessages sources	[NULL]	OK	Messages must be present for all tables/transforms
73	DImessages validation reports	0	OK	Every batch must have a full set of validation reports
74	DImessages validation reports	1	OK	Every batch must have a full set of validation reports
75	DImessages validation reports	2	OK	Every batch must have a full set of validation reports
76	DImessages validation reports	3	OK	Every batch must have a full set of validation reports
77	Data visibility joined row counts: FactCashB	[NULL]	OK	Row counts match when joined to dimensions
78	Data visibility joined row counts: FactHoldi	[NULL]	OK	Row counts match when joined to dimensions
79	Data visibility joined row counts: FactMarke	[NULL]	OK	Row counts match when joined to dimensions
80	Data visibility joined row counts: FactWatch	[NULL]	OK	Row counts match when joined to dimensions
81	Data visibility row counts: DimAccount	[NULL]	OK	Row counts must be non-decreasing over time
82	Data visibility row counts: DimBroker	[NULL]	OK	Row counts must be non-decreasing over time
83	Data visibility row counts: DimCompany	[NULL]	OK	Row counts must be non-decreasing over time
84	Data visibility row counts: DimCustomer	[NULL]	OK	Row counts must be non-decreasing over time
85	Data visibility row counts: DimDate	[NULL]	OK	Row counts must be non-decreasing over time
86	Data visibility row counts: DimSecurity	[NULL]	OK	Row counts must be non-decreasing over time
87	Data visibility row counts: DimTime	[NULL]	OK	Row counts must be non-decreasing over time
88	Data visibility row counts: DimTrade	[NULL]	OK	Row counts must be non-decreasing over time
89	Data visibility row counts: FactCashBalance:	[NULL]	OK	Row counts must be non-decreasing over time
90	Data visibility row counts: FactHoldings	[NULL]	OK	Row counts must be non-decreasing over time

Figure 2.33: Test Cases Execution Result 3

	TEST	BATCH	RESULT	DESCRIPTION
91	Data visibility row counts: FactMarketHistor	[NULL]	OK	Row counts must be non-decreasing over time
92	Data visibility row counts: FactWatches	[NULL]	OK	Row counts must be non-decreasing over time
93	Data visibility row counts: Financial	[NULL]	OK	Row counts must be non-decreasing over time
94	Data visibility row counts: Industry	[NULL]	OK	Row counts must be non-decreasing over time
95	Data visibility row counts: Prospect	[NULL]	OK	Row counts must be non-decreasing over time
96	Data visibility row counts: StatusType	[NULL]	OK	Row counts must be non-decreasing over time
97	Data visibility row counts: TaxRate	[NULL]	OK	Row counts must be non-decreasing over time
98	Data visibility row counts: TradeType	[NULL]	OK	Row counts must be non-decreasing over time
99	DimAccount EffectiveDate	1	OK	All records from a batch have an EffectiveDate in the batch time window
100	DimAccount EffectiveDate	2	OK	All records from a batch have an EffectiveDate in the batch time window
101	DimAccount EffectiveDate	3	OK	All records from a batch have an EffectiveDate in the batch time window
102	DimAccount End of Time	[NULL]	OK	Every Account has one record with a date range reaching the end of time
103	DimAccount EndDate	[NULL]	OK	EndDate of one record matches EffectiveDate of another, or the end of time
104	DimAccount IsCurrent	[NULL]	OK	IsCurrent is 1 if EndDate is the end of time, else Iscurrent is 0
105	DimAccount Overlap	[NULL]	OK	Date ranges do not overlap for a given Account
106	DimAccount SK_BrokerID	[NULL]	OK	All SK_BrokerIDs match a broker record with a valid date range
107	DimAccount SK_CustomerID	[NULL]	OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
108	DimAccount Status	[NULL]	OK	All Status values are valid
109	DimAccount TaxStatus	[NULL]	OK	All TaxStatus values are valid
110	DimAccount batches	[NULL]	OK	BatchID values must match "AUDIT" table
111	DimAccount consolidation	[NULL]	OK	No records become effective and end on the same day
112	DimAccount distinct keys	[NULL]	OK	All SKs are distinct
113	DimAccount inactive customers	[NULL]	OK	If a customer is inactive, the corresponding accounts must also have been inactive
114	DimAccount row count	1	OK	Actual row count matches or exceeds "AUDIT" table minimum
115	DimAccount row count	2	OK	Actual row count matches or exceeds "AUDIT" table minimum
116	DimAccount row count	3	OK	Actual row count matches or exceeds "AUDIT" table minimum
117	DimBroker BatchID	1	OK	All rows report BatchID = 1
118	DimBroker EffectiveDate	[NULL]	OK	All rows have Batch1 BatchDate as EffectiveDate
119	DimBroker EndDate	[NULL]	OK	All rows have end of time as EndDate
120	DimBroker IsCurrent	[NULL]	OK	All rows have IsCurrent = 1

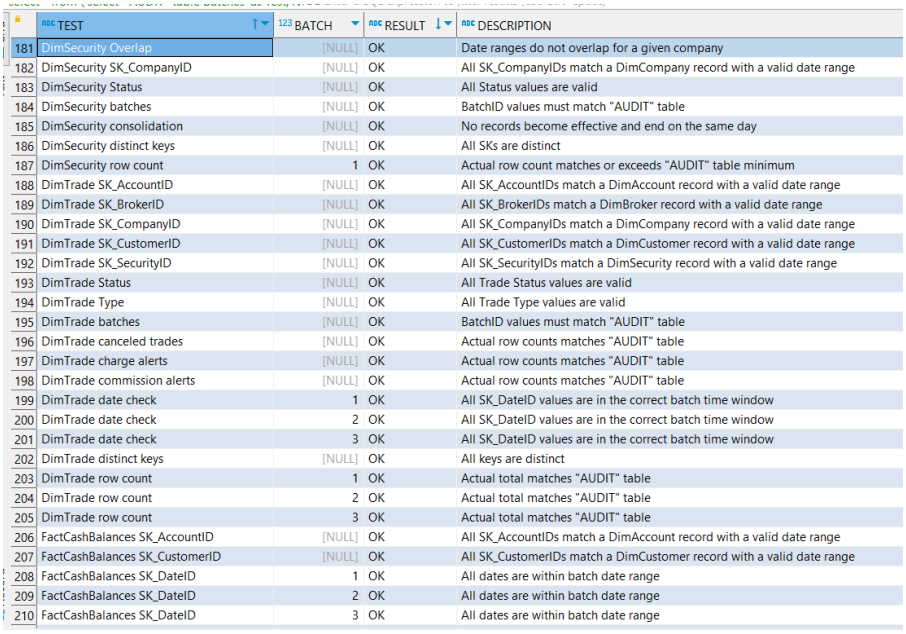
Figure 2.34: Test Cases Execution Result 4

	TEST	BATCH	RESULT	DESCRIPTION
121	DimBroker distinct keys	[NULL]	OK	All SKs are distinct
122	DimBroker row count	[NULL]	OK	Actual row count matches "AUDIT" table
123	DimCompany Country	[NULL]	OK	All Country values are valid
124	DimCompany EffectiveDate	1	OK	All records from a batch have an EffectiveDate in the batch time window
125	DimCompany EffectiveDate	2	OK	All records from a batch have an EffectiveDate in the batch time window
126	DimCompany EffectiveDate	3	OK	All records from a batch have an EffectiveDate in the batch time window
127	DimCompany End of Time	[NULL]	OK	Every company has one record with a date range reaching the end of time
128	DimCompany EndDate	[NULL]	OK	EndDate of one record matches EffectiveDate of another, or the end of time
129	DimCompany Industry	[NULL]	OK	Industry values are from the Industry table
130	DimCompany Overlap	[NULL]	OK	Date ranges do not overlap for a given company
131	DimCompany SPrating	[NULL]	OK	All SPrating values are valid
132	DimCompany Status	[NULL]	OK	All Status values are valid
133	DimCompany batches	[NULL]	OK	BatchID values must match "AUDIT" table
134	DimCompany consolidation	[NULL]	OK	No records become effective and end on the same day
135	DimCompany distinct keys	[NULL]	OK	All SKs are distinct
136	DimCompany distinct names	[NULL]	OK	Every company has a unique name
137	DimCompany row count	1	OK	Actual row count matches or exceeds "AUDIT" table minimum
138	DimCompany row count	2	OK	Actual row count matches or exceeds "AUDIT" table minimum
139	DimCompany row count	3	OK	Actual row count matches or exceeds "AUDIT" table minimum
140	DimCustomer EffectiveDate	1	OK	All records from a batch have an EffectiveDate in the batch time window
141	DimCustomer EffectiveDate	2	OK	All records from a batch have an EffectiveDate in the batch time window
142	DimCustomer EffectiveDate	3	OK	All records from a batch have an EffectiveDate in the batch time window
143	DimCustomer Email1	[NULL]	OK	Email1 values are properly formatted
144	DimCustomer Email2	[NULL]	OK	Email2 values are properly formatted
145	DimCustomer End of Time	[NULL]	OK	Every Customer has one record with a date range reaching the end of time
146	DimCustomer EndDate	[NULL]	OK	EndDate of one record matches EffectiveDate of another, or the end of time
147	DimCustomer Gender	[NULL]	OK	All Gender values are valid
148	DimCustomer IsCurrent	[NULL]	OK	IsCurrent is 1 if EndDate is the end of time, else Iscurrent is 0
149	DimCustomer LocalTaxRate	[NULL]	OK	LocalTaxRateDesc and LocalTaxRate values are from TaxRate table
150	DimCustomer NationalTaxRate	[NULL]	OK	NationalTaxRateDesc and NationalTaxRate values are from TaxRate table

Figure 2.35: Test Cases Execution Result 5

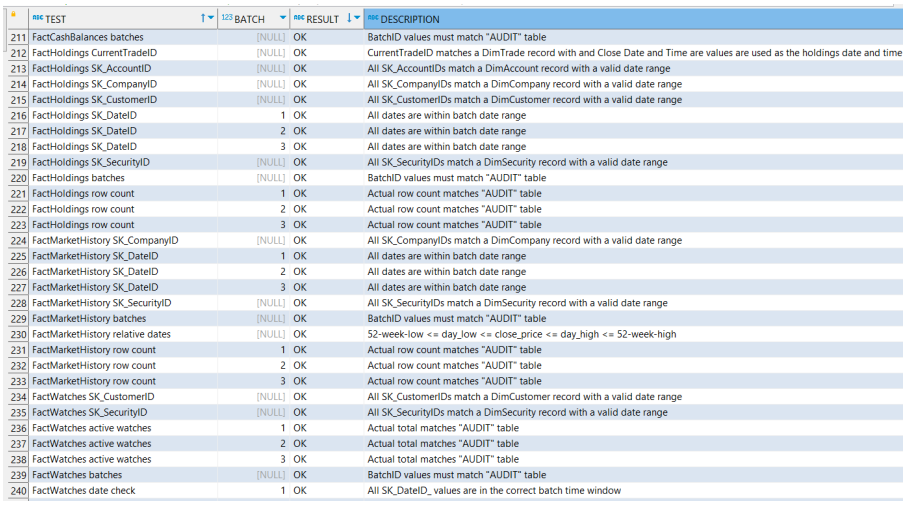
	TEST	BATCH	RESULT	DESCRIPTION
151	DimCustomer Overlap	[NULL]	OK	Date ranges do not overlap for a given Customer
152	DimCustomer Phone1	[NULL]	OK	Phone1 values are properly formatted
153	DimCustomer Phone2	[NULL]	OK	Phone2 values are properly formatted
154	DimCustomer Phone3	[NULL]	OK	Phone3 values are properly formatted
155	DimCustomer Status	[NULL]	OK	All Status values are valid
156	DimCustomer TaxID	[NULL]	OK	TaxID values are properly formatted
157	DimCustomer age range alerts	1	OK	Count of age range alerts matches "AUDIT" table
158	DimCustomer age range alerts	2	OK	Count of age range alerts matches "AUDIT" table
159	DimCustomer age range alerts	3	OK	Count of age range alerts matches "AUDIT" table
160	DimCustomer batches	[NULL]	OK	BatchID values must match "AUDIT" table
161	DimCustomer consolidation	[NULL]	OK	No records become effective and end on the same day
162	DimCustomer customer tier alerts	1	OK	Count of customer tier alerts matches "AUDIT" table
163	DimCustomer customer tier alerts	2	OK	Count of customer tier alerts matches "AUDIT" table
164	DimCustomer customer tier alerts	3	OK	Count of customer tier alerts matches "AUDIT" table
165	DimCustomer demographic fields	[NULL]	OK	For current customer records that match Prospect records, the demographic fields also match
166	DimCustomer distinct keys	[NULL]	OK	All SKs are distinct
167	DimCustomer inactive customers	1	OK	Inactive customer count matches "AUDIT" table
168	DimCustomer inactive customers	2	OK	Inactive customer count matches "AUDIT" table
169	DimCustomer inactive customers	3	OK	Inactive customer count matches "AUDIT" table
170	DimCustomer row count	1	OK	Actual row count matches or exceeds "AUDIT" table minimum
171	DimCustomer row count	2	OK	Actual row count matches or exceeds "AUDIT" table minimum
172	DimCustomer row count	3	OK	Actual row count matches or exceeds "AUDIT" table minimum
173	DimSecurity EffectiveDate	1	OK	All records from a batch have an EffectiveDate in the batch time window
174	DimSecurity EffectiveDate	2	OK	All records from a batch have an EffectiveDate in the batch time window
175	DimSecurity EffectiveDate	3	OK	All records from a batch have an EffectiveDate in the batch time window
176	DimSecurity End of Time	[NULL]	OK	Every company has one record with a date range reaching the end of time
177	DimSecurity EndDate	[NULL]	OK	EndDate of one record matches EffectiveDate of another, or the end of time
178	DimSecurity ExchangeID	[NULL]	OK	All ExchangeID values are valid
179	DimSecurity IsCurrent	[NULL]	OK	IsCurrent is 1 if EndDate is the end of time, else Iscurrent is 0
180	DimSecurity Issue	[NULL]	OK	All Issue values are valid

Figure 2.36: Test Cases Execution Result 6



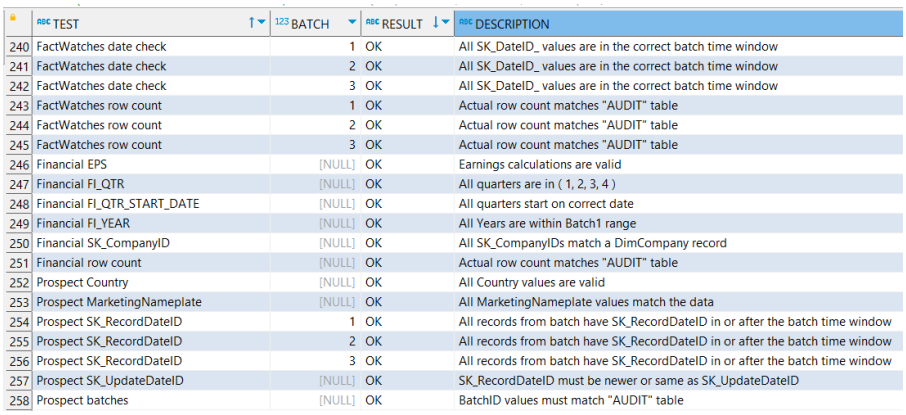
	TEST	BATCH	RESULT	DESCRIPTION
181	DimSecurity Overlap	[NULL]	OK	Date ranges do not overlap for a given company
182	DimSecurity SK_CompanyID	[NULL]	OK	All SK_CompanyIDs match a DimCompany record with a valid date range
183	DimSecurity Status	[NULL]	OK	All Status values are valid
184	DimSecurity batches	[NULL]	OK	BatchID values must match "AUDIT" table
185	DimSecurity consolidation	[NULL]	OK	No records become effective and end on the same day
186	DimSecurity distinct keys	[NULL]	OK	All SKs are distinct
187	DimSecurity row count	1	OK	Actual row count matches or exceeds "AUDIT" table minimum
188	DimTrade SK_AccountID	[NULL]	OK	All SK_AccountIDs match a DimAccount record with a valid date range
189	DimTrade SK_BrokerID	[NULL]	OK	All SK_BrokerIDs match a DimBroker record with a valid date range
190	DimTrade SK_CompanyID	[NULL]	OK	All SK_CompanyIDs match a DimCompany record with a valid date range
191	DimTrade SK_CustomerID	[NULL]	OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
192	DimTrade SK_SecurityID	[NULL]	OK	All SK_SecurityIDs match a DimSecurity record with a valid date range
193	DimTrade Status	[NULL]	OK	All Trade Status values are valid
194	DimTrade Type	[NULL]	OK	All Trade Type values are valid
195	DimTrade batches	[NULL]	OK	BatchID values must match "AUDIT" table
196	DimTrade canceled trades	[NULL]	OK	Actual row counts matches "AUDIT" table
197	DimTrade charge alerts	[NULL]	OK	Actual row counts matches "AUDIT" table
198	DimTrade commission alerts	[NULL]	OK	Actual row counts matches "AUDIT" table
199	DimTrade date check	1	OK	All SK_DateID values are in the correct batch time window
200	DimTrade date check	2	OK	All SK_DateID values are in the correct batch time window
201	DimTrade date check	3	OK	All SK_DateID values are in the correct batch time window
202	DimTrade distinct keys	[NULL]	OK	All keys are distinct
203	DimTrade row count	1	OK	Actual total matches "AUDIT" table
204	DimTrade row count	2	OK	Actual total matches "AUDIT" table
205	DimTrade row count	3	OK	Actual total matches "AUDIT" table
206	FactCashBalances SK_AccountID	[NULL]	OK	All SK_AccountIDs match a DimAccount record with a valid date range
207	FactCashBalances SK_CustomerID	[NULL]	OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
208	FactCashBalances SK_DateID	1	OK	All dates are within batch date range
209	FactCashBalances SK_DateID	2	OK	All dates are within batch date range
210	FactCashBalances SK_DateID	3	OK	All dates are within batch date range

Figure 2.37: Test Cases Execution Result 7



	TEST	BATCH	RESULT	DESCRIPTION
211	FactCashBalances batches	[NULL]	OK	BatchID values must match "AUDIT" table
212	FactHoldings CurrentTradeID	[NULL]	OK	CurrentTradeID matches a DimTrade record with and Close Date and Time are values are used as the holdings date and time
213	FactHoldings SK_AccountID	[NULL]	OK	All SK_AccountIDs match a DimAccount record with a valid date range
214	FactHoldings SK_CompanyID	[NULL]	OK	All SK_CompanyIDs match a DimCompany record with a valid date range
215	FactHoldings SK_CustomerID	[NULL]	OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
216	FactHoldings SK_DateID	1	OK	All dates are within batch date range
217	FactHoldings SK_DateID	2	OK	All dates are within batch date range
218	FactHoldings SK_DateID	3	OK	All dates are within batch date range
219	FactHoldings SK_SecurityID	[NULL]	OK	All SK_SecurityIDs match a DimSecurity record with a valid date range
220	FactHoldings batches	[NULL]	OK	BatchID values must match "AUDIT" table
221	FactHoldings row count	1	OK	Actual row count matches "AUDIT" table
222	FactHoldings row count	2	OK	Actual row count matches "AUDIT" table
223	FactHoldings row count	3	OK	Actual row count matches "AUDIT" table
224	FactMarketHistory SK_CompanyID	[NULL]	OK	All SK_CompanyIDs match a DimCompany record with a valid date range
225	FactMarketHistory SK_DateID	1	OK	All dates are within batch date range
226	FactMarketHistory SK_DateID	2	OK	All dates are within batch date range
227	FactMarketHistory SK_DateID	3	OK	All dates are within batch date range
228	FactMarketHistory SK_SecurityID	[NULL]	OK	All SK_SecurityIDs match a DimSecurity record with a valid date range
229	FactMarketHistory batches	[NULL]	OK	BatchID values must match "AUDIT" table
230	FactMarketHistory relative dates	[NULL]	OK	52-week-low <= day_low <= close_price <= day_high <= 52-week-high
231	FactMarketHistory row count	1	OK	Actual row count matches "AUDIT" table
232	FactMarketHistory row count	2	OK	Actual row count matches "AUDIT" table
233	FactMarketHistory row count	3	OK	Actual row count matches "AUDIT" table
234	FactWatches SK_CustomerID	[NULL]	OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
235	FactWatches SK_SecurityID	[NULL]	OK	All SK_SecurityIDs match a DimSecurity record with a valid date range
236	FactWatches active watches	1	OK	Actual total matches "AUDIT" table
237	FactWatches active watches	2	OK	Actual total matches "AUDIT" table
238	FactWatches active watches	3	OK	Actual total matches "AUDIT" table
239	FactWatches batches	[NULL]	OK	BatchID values must match "AUDIT" table
240	FactWatches date check	1	OK	All SK_DateID_ values are in the correct batch time window

Figure 2.38: Test Cases Execution Result 8



	TEST	BATCH	RESULT	DESCRIPTION
240	FactWatches date check	1	OK	All SK_DateID_ values are in the correct batch time window
241	FactWatches date check	2	OK	All SK_DateID_ values are in the correct batch time window
242	FactWatches date check	3	OK	All SK_DateID_ values are in the correct batch time window
243	FactWatches row count	1	OK	Actual row count matches "AUDIT" table
244	FactWatches row count	2	OK	Actual row count matches "AUDIT" table
245	FactWatches row count	3	OK	Actual row count matches "AUDIT" table
246	Financial EPS	[NULL]	OK	Earnings calculations are valid
247	Financial FI_QTR	[NULL]	OK	All quarters are in (1, 2, 3, 4)
248	Financial FI_QTR_START_DATE	[NULL]	OK	All quarters start on correct date
249	Financial FI_YEAR	[NULL]	OK	All Years are within Batch1 range
250	Financial SK_CompanyID	[NULL]	OK	All SK_CompanyIDs match a DimCompany record
251	Financial row count	[NULL]	OK	Actual row count matches "AUDIT" table
252	Prospect Country	[NULL]	OK	All Country values are valid
253	Prospect MarketingNameplate	[NULL]	OK	All MarketingNameplate values match the data
254	Prospect SK_RecordDateID	1	OK	All records from batch have SK_RecordDateID in or after the batch time window
255	Prospect SK_RecordDateID	2	OK	All records from batch have SK_RecordDateID in or after the batch time window
256	Prospect SK_RecordDateID	3	OK	All records from batch have SK_RecordDateID in or after the batch time window
257	Prospect SK_UpdateDateID	[NULL]	OK	SK_RecordDateID must be newer or same as SK_UpdateDateID
258	Prospect batches	[NULL]	OK	BatchID values must match "AUDIT" table

Figure 2.39: Test Cases Execution Result 9

3 Results

Benchmarking was conducted using data generated for the **Scale Factors of 3, 4, 5, and 6**. Total Time taken to load data (Load Time) along with Number of operations completed per second (Throughput) has been analyzed. We also did further analysis on the number of records that were loaded during each phase of the benchmarking.

3.1 Load Time

Load Time measures the amount of time needed to load data into database. The bar graph presented in Figure 3.1 illustrates the overall Load Time, including separate metrics for Historical Load and Incremental Updates across four different scale factors.

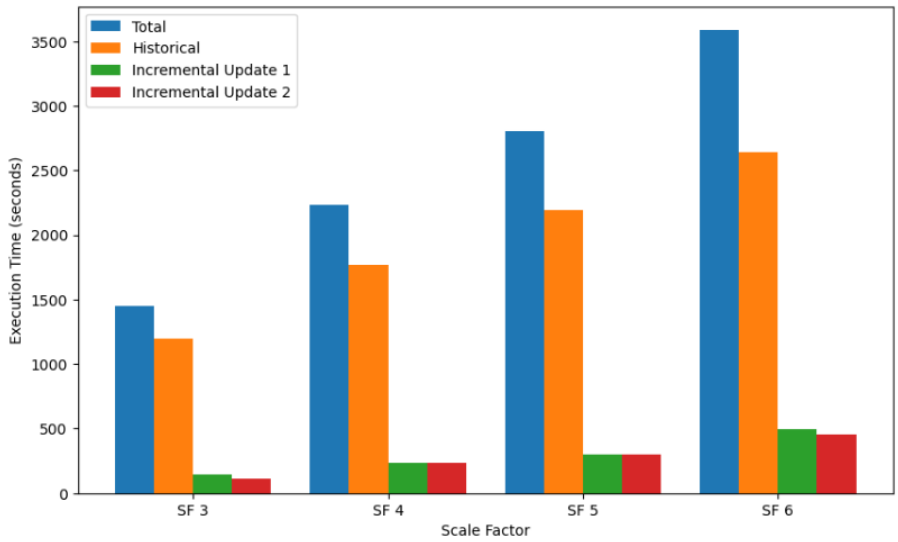


Figure 3.1: Execution Time for different Scale Factors

3.2 Throughput

Throughput, measured in ops/sec, is defined as the number of operations completed per seconds. The bar graph presented in Figure 3.2 illustrates the Throughput for Historical and Incremental Updates across four different scale factors.

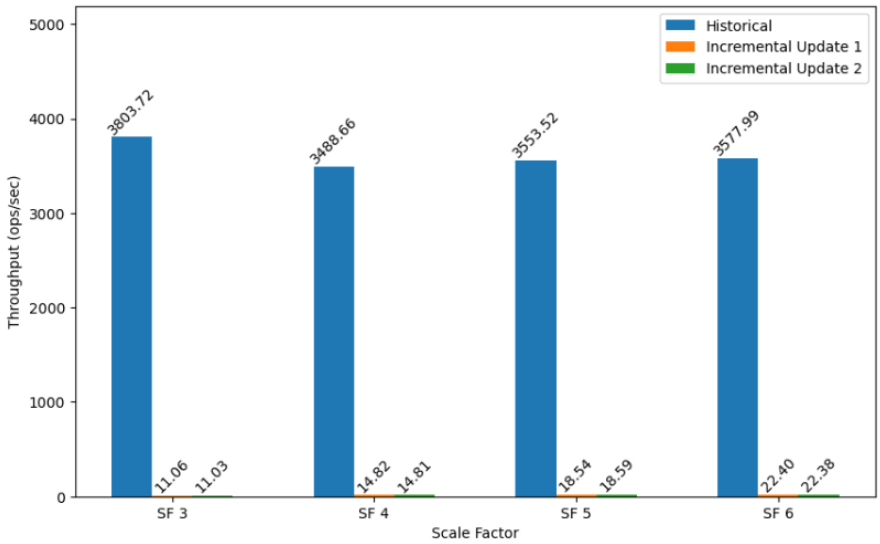


Figure 3.2: Throughput for different Scale Factors

3.3 Performance Metric

Performance Metric is a combined metric calculated using the three throughput values. The bar graph presented in Figure 3.3 illustrates the Performance Metric across four different scale factors.

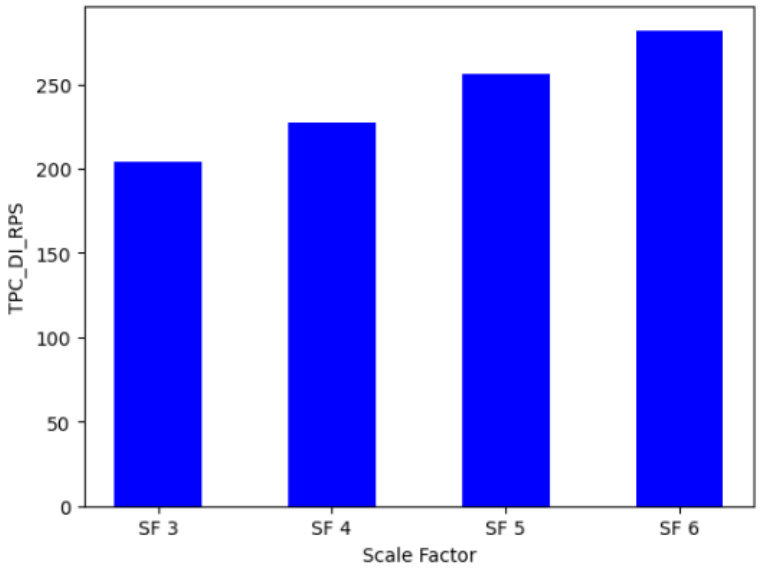


Figure 3.3: Performance Metric for different Scale Factors

3.4 Number of Records Loaded

Higher the scale factor, higher is the number of records generated by DIGen.

Note: For the scale factor of 3, data equivalent to approx. 300 MB is generated.

The bar graph presented in Figure 3.4 illustrates the total number of records loaded, including separate metrics for Number of Records loaded for Historical Load and Incremental Updates across four different scale factors.

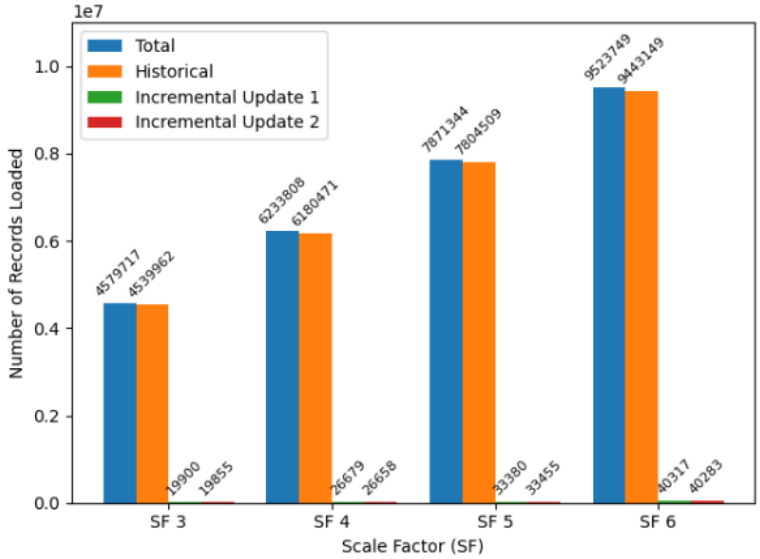


Figure 3.4: Number of Records Loaded

In addition to this, the bar graph presented in Figure 3.5, 3.6, 3.7 and 3.8 illustrates the total number of records loaded in each table for the sclae factor of 3, 4, 5 and 6 respectively.

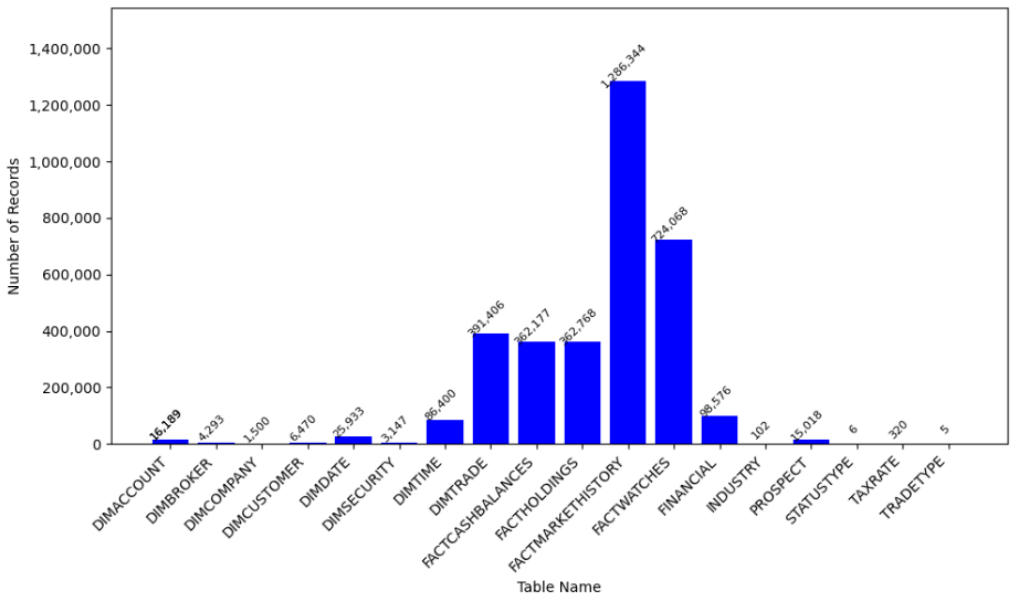


Figure 3.5: Tables with Record Count (SF3)

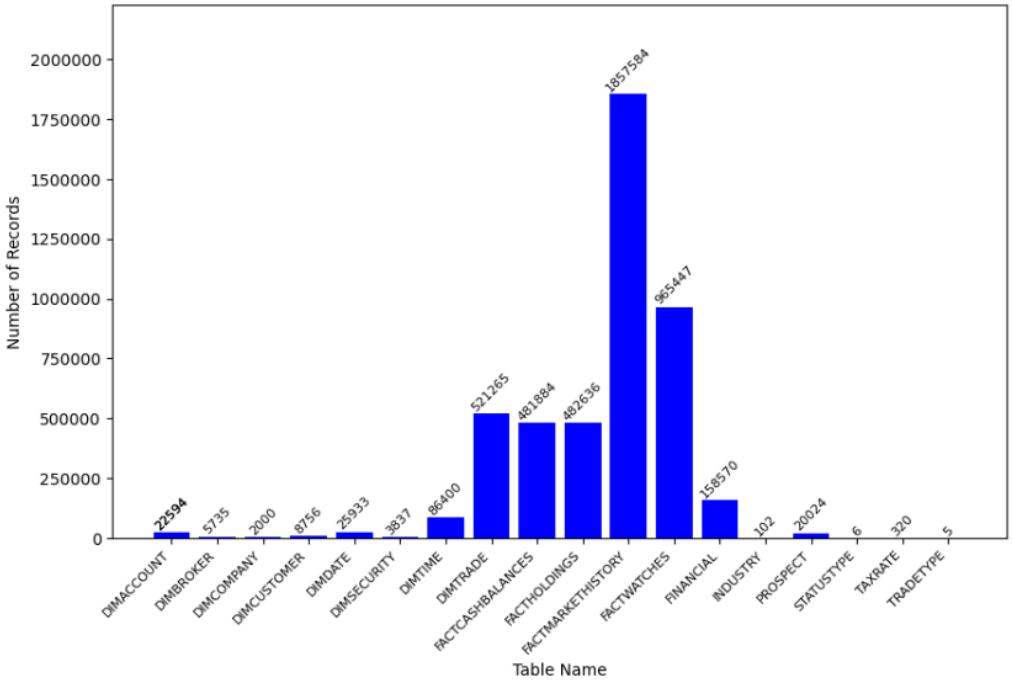


Figure 3.6: Tables with Record Count (SF4)

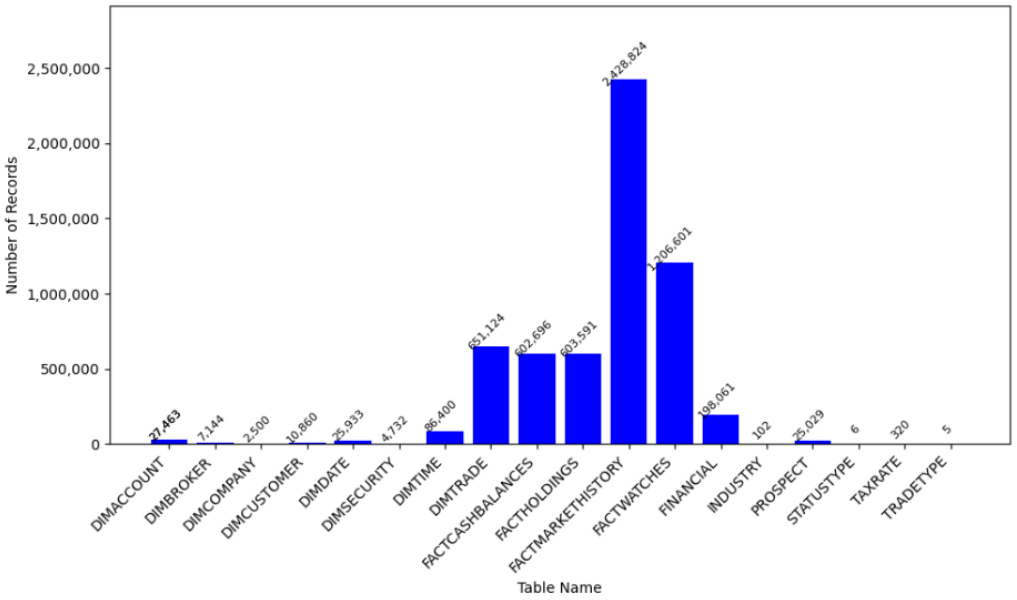


Figure 3.7: Tables with Record Count (SF5)

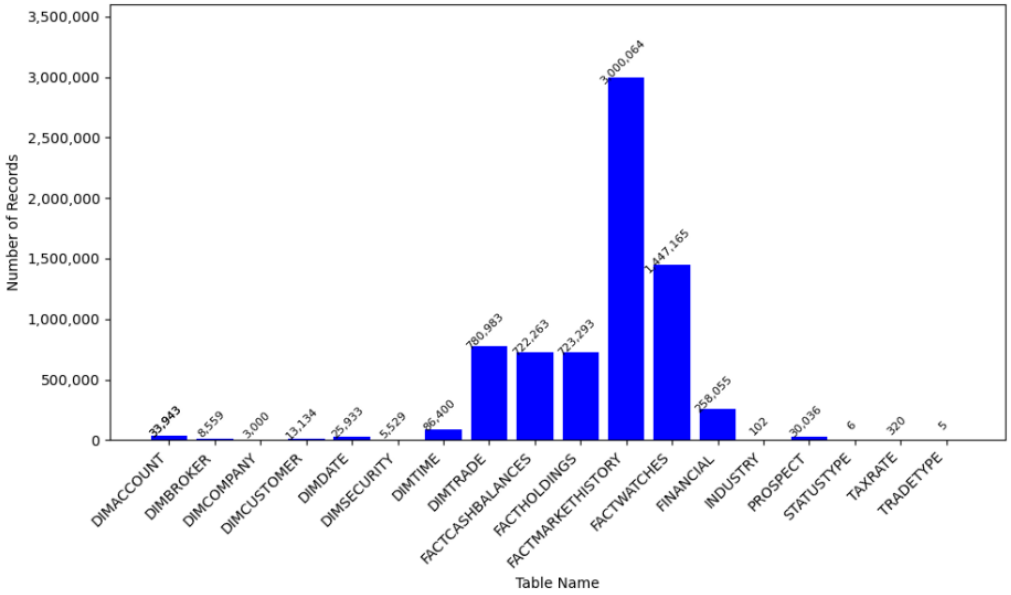


Figure 3.8: Tables with Record Count (SF6)

4 Conclusion

To sum up, TPC-DI is a benchmark designed by The TPC (Transaction Processing Performance Council) for the purpose of data integration. This benchmark, set within the context of a retail brokerage firm, aims to be representative of a wide range of use cases where data integration is crucial for creating a concise Decision Support System. The operational system of TPC-DI comprises an OLTP (On-Line Transaction Processing) database, an HR system, and a CRM system, with the assumption that externally acquired financial and marketing data are also available.

The TPC-DI specification outlines the requirements for data transformations which involve extracting data from a staging area, processing and cleansing it, and then loading it into a data warehouse with a schema consisting of facts and dimensions. The specification is intentionally presented in a relaxed, semi-formal manner to avoid tying the benchmark to specific models and technologies, allowing vendors to use their custom technologies and implementations. However, the lack of a structured description, such as SQL syntax, makes it challenging for researchers to use these processes for test cases and examples.

The purpose of this project was to create ETL processes that adhere to the TPC-DI specification using some technologies. Pentaho Data Integration (PDI) was employed to model the workflows, and Oracle as database for target Data Warehouse. The source files for the ETL workflows were generated by using DIGen. Historical Load, followed by two Incremental Loads were performed. In order to validate transformations prepared, steps provided in Audit Phase section of specification were carried out. All the test cases have been successfully validated, indicating the precision of the implemented transformations. In addition to this, we analyzed how the ETL tool was performing in terms of Load Time and Throughput across different scale factors.

References

[Nam18] Name(s), Author (2018). “Data Quality Problems in TPC-DI Based Data Integration Processes”. In: *Journal or Conference*. [Online; accessed 17-December-2023]. URL: https://doras.dcu.ie/22315/1/2018_Data_Quality_Problems_in_TPC-DI_Based_Data_Integration_Processes.pdf.

[Tea21] Team, TPC Benchmark™ Development (2021). “TPC-DI Benchmark”. In: *TPC Documents*. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-di_v1.1.0.pdf.

[Wik23] Wikipedia (2023). *Extract, Transform, Load*. [Online; accessed 17-December-2023]. URL: https://en.wikipedia.org/wiki/Extract,_transform,_load.