INFO-H-419: Data Warehouses

TPC-DI Benchmarking Test of Microsoft SQL Server

Group

Diogo Repas

Nicole Kovacs

Andres Espinal

Adam Broniewski

Professor

Esteban Zimányi

December 2021

Table of Contents

List of Tables

List of Figures

1 Introduction

This report provides and overview of benchmarking completed on Microsoft SQL Server following TPC-DI v.1.1.0 specification as a guide.

1.1 TPC-DI benchmarking Overview

The Transaction Performance Processing Council (TPC) is an organization that creates, maintains, and oversees benchmarking. The TPC – Decision Integration (TPC-DI) benchmark is aimed to benchmark the extraction, transformation, and loading data process from an On-Line Transaction Processing (OLTP) system and other data sources into a data warehouse.

The process of the TPC-DI benchmark consists of 2 main components:

1. Data warehouse preparation
2. Benchmark run

1.2 Microsoft SQL Server Overview

Microsoft SQL Server is a relational DBMS developed and marketed by Microsoft. It is built based on the SQL language standards and is tied to Transact-SQL (or T-SQL) which adds programming constructs on top of SQL (Anon., 2021).

2 METHODOLOGY

This section describes the technology, references and methodology used to complete the database benchmark.

2.1 TOOLS USED

The following tools were installed to complete the benchmark:

- SQL Server 2019 Express
- SQL Server Data Tools 2017 (Standalone along with Visual Studio)
- Materials and programs provided by TPC-DI (Anon., 2021)

The benchmark queries and logging were implemented using Microsoft SQL Server Integration Services (SSIS).

The timing results were plotted in a live Tableau dashboard[1] that collects the logging results automatically from the database.

Data was generated using the TPC-DI data generator at 4 scale factors (SF):

- SF 3
- SF 10
- SF 20
- SF 30

There were two research papers used as a general reference for the TPC-DI ETL process that provided support in identifying data quality issues. These papers were:

- Data Quality Problems in TPC-DI Based Data Integration Processes (Yang, et al., 2018)
- TPC-DI: The First Industry Benchmark for Data Integration (Poess, et al., 2014)

A git repository was also used as a reference for the data warehouse table creation. The repository used was:

- https://github.com/detobel36/tpc-di (Moreira, et al., n.d.)

This repository was reviewed and checked against the current version of TPC-DI specification to ensure correctness and consistency.

[1] https://public.tableau.com/app/profile/andresespinalh/viz/TPC-DIBenchmark/TPC-DSDashboard

2.2 Benchmark Steps Overview

The TPC-DI benchmark requires the following tests to be executed in one run:

1. Data Warehouse Initialization
2. Historical Load
3. Incremental Update 1
4. Incremental Update 2
5. Automated Audit

For the scope of the project, Incremental Update 1, 2 and the Automated Audit were not completed. Only the initialization and historical load were completed and are described in the following sections. Initialization of the database warehouse is not a timed part of the benchmark.

2.3 Datawarehouse Initialization

The main goal for this benchmark is to measure the time it takes for transforming the data provided in multiple formats and loading it correctly in a data warehouse. In this project's implementation, a staging area was implemented to load all the raw data in a database to facilitate the interface between SSIS and the files. This step is called data warehouse initialization and, by definition, it is not timed in the benchmark.

Initialization started with the creation of a staging area in the database. The staging area was called 'source'. An empty table was generated for each source file following the table definitions in section 3.2[2] of the specification. SSIS was used to support loading the raw files into the empty relational tables in the DBMS staging area. No transformations were completed at this stage of the load into the staging area. Moving all files from disparate file formats into the staging area completed the extract part of the process. The loading of raw data into the database staging area was not a timed event. No transformations of the data took place as the data was loaded into SQL tables.

All data that existed in '.txt' format, the tables were loaded following TPC-DI specification with no additional manipulation required. The tables were loaded using SSIS.

The Customer Management System table, 'CustomerMgmt.xml' required the creation of an XML schema file in '.xsd' format used to validate the '.xml' file rules and explain the file form. This file was generated by inputting a portion of the 'CustomerMgmt.xml' into an online

[2] TPC-DI_v1.1.0 Section 3.2: Table Definitions

schema generator to produce and was reviewed and validated for use in the data load. With the '.xsd' file, CustomerMgmt.xml was loaded using SSIS.

For the FINWIRE files, the import was completed using a python script to parse the file into three different files:

1. CMP
2. FIN
3. SEC

This follows the specification definitions in section 2.2.2.8. The python script used can be reviewed in Appendix B.

2.4 Historical Load

For the historical load we have implemented the transformations described in the subsections below. The work was divided into simple loads, which describe simple straightforward transformations and complex loads, where many transformations steps are needed.

The historical load was done in a specific order to ensure that the data (and its foreign keys) exist. The loads were done in the following order:

1. DimDate
2. DimTime
3. StatusType
4. TaxRate
5. TradeType
6. DimBroker
7. DimCompany
8. DimCustomer
9. DimAccount
10. DimSecurity
11. DimTrade
12. FactCashBalances
13. FactMarketHistory
14. FactWatches
15. Industry
16. Financial
17. Prospect

All the loads were done from the staging area in the “source” schema into the data warehouse area in the “dbo” schema (Figure 2-1).

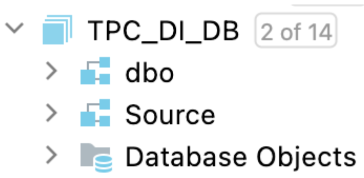


Figure 2-1: Database staging area "source" and data warehouse "dbo".

2.4.1 Simple loads

For simple loads, data is loaded from the staging area directly to the data warehouse using the existing table schema able to be load. These tables had no dependencies and were able to be inserted with a simple insert query (Table 2-1).

Table 2-1: Example of simple table inserts from staging area to data warehouse.

```
INSERT INTO dbo.DimDate
        (SK_DateID,
        . . .
        HolidayFlag)
SELECT
        SK_DateID,
      . . .
        HolidayFlag,
FROM Source.Date;
GO
```

The following tables were inserted this way:

- DimDate
- DimTime
- StatusType
- TaxRate
- TradeType

2.4.2 Complex loads

The remaining tables had a more complex approach that required some transformations and aggregations to fit the required data format. These transformations were completed with a combination of the following methods:

- CASE WHEN
- PARTITION BY

- JOIN
- UNION
- LEAD
- COALLESCE
- CONVERT
- Common table expressions (CTEs)

The tables that required this additional transformation include:

- DimBroker
- DimCompany
- DimCustomer
- DimAccount
- DimSecurity
- DimTrade
- FactCashBalances
- FactHoldings
- FactMarketHistory
- FactWatches
- Industry
- Financial
- Prospect

A repeating concept that was addressed was the slowly changing dimensions in several of the tables. The general approach to deal with any tables that had slow changing dimensions, such as customer account information updates or changes, was is described below.

In the case of **new rows** (such as new account, add account, etc.) the process was just to select the data, set the status to active and insert it using an INSERT SELECT statement.

In the case of **updates** (such as update account), we must refer to two different tables: the table with the update data (to_update) and the table with the old data to be updated (old_data). Now the process is to select the data from old_data, join it with the to_update data, and COALESCE the columns in an order such that, for each column it should use the to_update data if it is not null, and use the old_data otherwise.

In the case of **deletes**, the process was to select the data to be deleted, set the status column to "inactive" and insert the data using an INSERT SELECT statement.

Additionally **in all cases above**, we define the timestamp of the changes for the "from date" field, update the previous row (obtained by partitioning the full data by ID and ordered by action timestamp, and using the LEAD function to obtain the previous row) with the timestamp for the "to date" field, and change the is_current fields accordingly.

An overview of table specific implementation is provided below. Only tables with some interesting approaches are highlighted here.

2.4.2.1 DimBroker

Pulls the minimum DateValue to set it as Effective date

Table 2-2: Snippet of DimBroker SQL table to identify the EffectiveDate using MIN.

```sql
INSERT INTO dbo.DimBroker (IsCurrent, EffectiveDate, ...BatchID ...)
SELECT
   1 AS IsCurrent,
   (SELECT MIN(DateValue) FROM DimDate) as EffectiveDate,
   '9999-12-31' AS EndDate,
   1 as BatchID,
```

2.4.2.2 DimCompany

DimCompany makes use a combination of CASE and LEAD statement to determine which date to treat as the CurrentDate. It also uses a combination of COALESCE and LEAD to determine which row it should use as the EndDate

Table 2-3: DimCompany SQL script using CASE, LEAD, and COALESCE.

```sql
INSERT INTO dbo.DimCompany (IsCurrent, EffectiveDate, EndDate ...)
SELECT
   CASE WHEN LEAD( (SELECT TOP 1 BatchDate FROM Source.BatchDate) )
     OVER ( PARTITION BY CIK ORDER BY PTS ASC ) IS NULL THEN 1 ELSE 0 END AS
IsCurrent,
   (SELECT TOP 1 BatchDate FROM Source.BatchDate) as EffectiveDate,
   COALESCE( LEAD( (SELECT TOP 1 BatchDate FROM Source.BatchDate) )
     OVER ( PARTITION BY CIK ORDER BY PTS ASC ), '9999-12-31' ) AS EndDate
```

2.4.2.3 DimCustomer

DimCustomer uses a trim function on all string values to remove any leading spaces. There are several common table expressions created to make the combinations and processing easier to read through and understand. The COALESCE approach was used again on this table to join on first name, last name, and the two address lines. Tables where data is missing is still joined and with missing values. The implementation requires that a value is NULL on both tables being merged, it will not merge a table where it is NULL with one where the value in non-NULL (Table 2-4). This was not made explicit in TPC-DI benchmarking.

Table 2-4: DimCustomer SQL script joining on NULL values in each table.

```
COALESCE( UPPER( TRIM( NXML.C_F_NAME ) ), ' ' ) = COALESCE( UPPER( TRIM( P.FirstName ) ), ' ' )
-- Join on LastName if exists
AND COALESCE( UPPER( TRIM( NXML.C_L_NAME ) ), ' ' ) = COALESCE( UPPER( TRIM( P.LastName ) ), ' ' )
-- Join on AddressLine1 if exists
AND COALESCE( UPPER( TRIM( ADXML.C_ADLINE1 ) ), ' ' ) = COALESCE( UPPER( TRIM( P.AddressLine1 ) ),
' ' )
-- Join on AddressLine2 if exists
AND COALESCE( UPPER( TRIM( ADXML.C_ADLINE2 ) ), ' ' ) = COALESCE( UPPER( TRIM( P.AddressLine2 ) ),
' ' )
-- Join on PostalCode if exists
AND COALESCE( UPPER( TRIM( ADXML.C_ZIPCODE ) ), ' ' ) = COALESCE( UPPER( TRIM( P.PostalCode ) ), '
' )
```

Here we determine the customer status based on the ActionType during the insert (Table 2-5).

Table 2-5: DimCustomer SQL script to set customer status.

```
CustomersNew AS (
   SELECT *, 'ACTIVE' AS [Status] FROM Customers WHERE ActionType = 'NEW')
, CustomersUpd AS (
   SELECT * FROM Customers WHERE ActionType = 'UPDCUST')
, CustomersInactive AS (
   SELECT C_ID, ActionTS
   FROM [Source].[CustomerXML] CXML
      INNER JOIN [Source].[ActionXML] AXML
         ON CXML.Action_Id = AXML.Action_Id
   WHERE ActionType = 'INACT')
```

For any customers that are updated, the results of the subquery for updates will have a NULL value for any attribute that is not being updated with new information. A single update can have multiple changes, or there can be multiple updates that take place for every data load. The COALESCE function permits the first value in the query to be selected. If the first value in the COALESCE function is NULL (the potentially new update value), then the second value (the already existing value) will be returned by the function. A new row of data is created for every customer update that takes place.

For an update where the customer status is set to INACTIVE, the date does not need to be updated, only the status. The subquery will select the row with the latest timestamp (ActionTS) to update with the inactive status.

All of the activities described above are completed in a CTE called “CustomersFinal”. With all of the transformations complete, we can now insert all of the values in the data warehouse DimCustomer table. The final action taken during upload of each row is to use a CASE command with a PARTITION BY CustomerID to check which row of data is the most current. This row will receive a “1” in a column named “IsCurrent” and will have the EndDate set to 9999-12-31” (Table 2-6).

To compete select the correct row of data, Check to see if this row is the most current in the table. Create a IsCurrent row. Whenever the row stops being the latest value, make the IsCurrent value 0. For the most up to date value you use "1" as the indicator here. Lead looks at the row above me (which is the previous row in time based on the sort we do. If the value is NULL, we know that this is the most current value.)

Table 2-6: DimCustomer SQL identifying the most current customer row.

```
, CASE WHEN LEAD( ActionTS ) OVER ( PARTITION BY CustomerID ORDER BY ActionTS
ASC ) IS NULL THEN 1 ELSE 0 END AS IsCurrent
, 1 AS BatchID
, ActionTS AS EffectiveDate
, COALESCE( LEAD( ActionTS ) OVER ( PARTITION BY CustomerID ORDER BY ActionTS
ASC ), '9999-12-31 00:00:00' ) AS EndDate
```

2.4.2.4 DimTrade

The DimTrade table followed the TPC-DI requirements generally, however the TPC-DI documentation had an inconsistency in its restrictions. In section 3.2.10 of the TPC-DI specification, SK_CreateDateID and SK_CreateTimeID are specified as being NOT NULL (i.e. they should not accept NULL values). However in section 4.5.8.2, the specification outlines that columns should be set to NULL if a new DimTrade is being created. For our data warehouse, we chose to remain consistent with section 4.5.8.2 and changed our schema to allow NULL values into the DimTrade table for these attributes.

2.4.2.5 DimSecurity

DimSecurity is a table that is a dependency for several fact tables. There was a problem that impacted three tables with this dependency, FactMarketHistory, FactWatches, and DimTrade. There was no data being returned when trying to join these tables with the DimSecurity table. The issue was not able to be resolved and we decided to move forward as is to complete benchmarking. The table was commented out for the benchmark run.

2.4.2.6 FactHoldings

This table had the same inconsistency of NULL and NON-NULL value requirements from the TPC-DI spec as DimTrade. We followed the same approach and allowed NULL values. The spec was not explicit about which value to use for CurrentPrice, and we chose to use TradePrice, as it seemed to make the most sense.

2.4.2.7 FactMarketHistory

To calculate the 52-week and 52-week low, a CTE with 365-day partition with 3 self joins was used . The first SELECT statement returns our “main” table, the 2nd table that is joined is used to calculate the 52-week high, and the third table is used to calculate the 52-week low (Table 2-7).

Table 2-7: FactMarketHistory SQL using self-joins to identify 52-week high and low.

```
WITH DailyMarkets AS (
   SELECT DM1.*, MIN(DM2.DM_DATE) AS FiftyTwoWeekHighDate, MIN(DM3.DM_DATE) AS
FiftyTwoWeekLowDate
   FROM
   (
      SELECT
         DM_DATE,
         ...
         MAX(DM_HIGH) OVER(PARTITION BY DM_S_SYMB ORDER BY DM_DATE ROWS
BETWEEN 364 PRECEDING AND CURRENT ROW) AS FiftyTwoWeekHigh,
         MIN(DM_LOW) OVER(PARTITION BY DM_S_SYMB ORDER BY DM_DATE ROWS BETWEEN
364 PRECEDING AND CURRENT ROW) AS FiftyTwoWeekLow
      FROM Source.DailyMarket
   ) DM1

INNER JOIN Source.DailyMarket DM2
      ON DM2.DM_HIGH = DM1.FiftyTwoWeekHigh
      AND DM2.DM_DATE BETWEEN CONVERT(DATE, DATEADD(DAY, -364, DM1.DM_DATE))
      AND DM1.DM_DATE

INNER JOIN Source.DailyMarket DM3
      ON DM3.DM_LOW = DM1.FiftyTwoWeekLow
      AND DM3.DM_DATE BETWEEN CONVERT(DATE, DATEADD(DAY, -364, DM1.DM_DATE))
AND DM1.DM_DATE

GROUP BY DM1.DM_DATE, DM1.DM_S_SYMB, DM1.DM_CLOSE, DM1.DM_HIGH, DM1.DM_LOW,
DM1.DM_VOL, DM1.FiftyTwoWeekHigh, DM1.FiftyTwoWeekLow
),
```

There is another CTE used to sum the earnings per share by quarter for each company (Table 2-8).

Table 2-8: FactMarketHistory SQL to sum the earnings per share by quarter for each company.

```
FIN AS (
   SELECT
      CoNameOrCIK,
      SUM(CAST(EPS AS FLOAT)) OVER(PARTITION BY Quarter ORDER BY Year, Quarter
ROWS BETWEEN 4 PRECEDING AND CURRENT ROW) AS EPSSum
   FROM Source.FinwireFIN
),
```

With these CTEs, the final FactMarketHistory table can be created following TPC-DI requirements.

2.4.2.8 Financial

For this transformation, it was necessary to manually parse through the date (PTS) to convert into a format recognized by MS SQL Server (Table 2-9).

Table 2-9: Financial SQL parsing through the date column (PTS) to convert into MS SQL format.

```sql
SELECT CAST( CONCAT( SUBSTRING( PTS, 0, 5 ), '-', SUBSTRING( PTS, 5, 2 ) , '-
', SUBSTRING( PTS, 7, 2 ), ' ', SUBSTRING( PTS, 10, 2 ), ':', SUBSTRING( PTS,
12, 2 ), ':', SUBSTRING( PTS, 14, 2 ) ) AS DATETIME ) AS PTS
```

2.5 Final SSIS Workflow

The following screenshot illustrates the SSIS workflow used in the project.

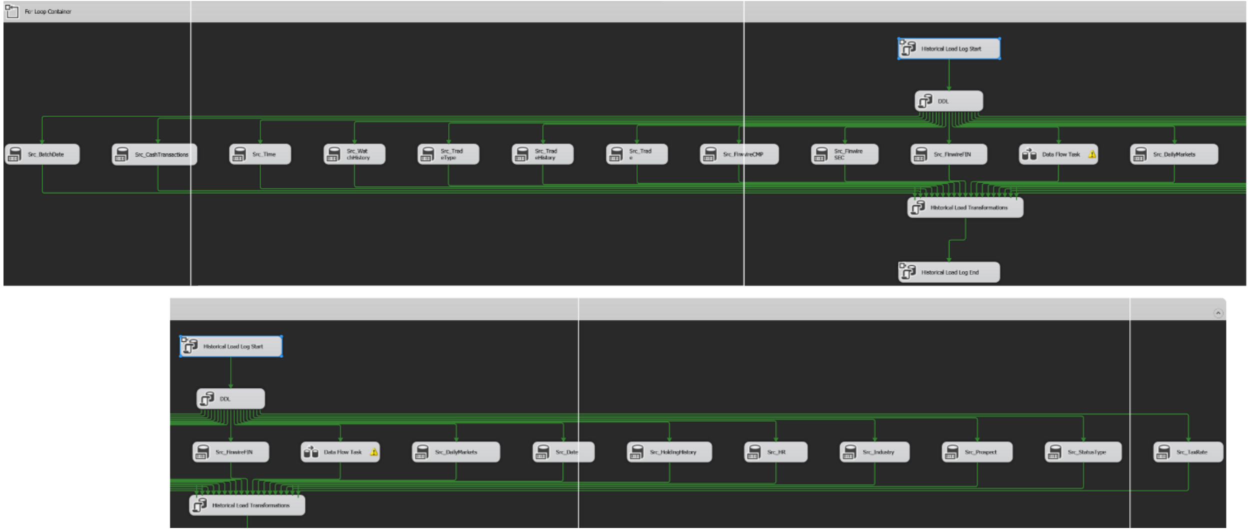


Figure 2-2: Diagram of the ETL process

3 Discussion

3.1 Classification of Data Quality Problems

Data quality issues during the ETL process can be grouped into major categories to help data engineers manage data quality during historical loads and updates. A proposal for data quality classification from Data Quality Problems in TPC-DI Based Data Integration Process (Yang, et al., 2018) includes the following:

1. Completeness (section 3.1.1)
2. Timeliness (section 3.1.2)
3. Consistency (section 3.1.3)
4. Operational Sequence (section 3.1.4)
5. Uniqueness (section 3.1.5)

3.1.1 Completeness

A lack of completeness can be seen anywhere our data has missing values. Missing values can be the result of a lack of direct rule enforcement, or due to the design of the database system. For example, a new customer may not need to include any information in "Address 2" or "Address 3". In this case, there is a missing value by design that is assumed to not create issues further along the data processing pipeline.

On the other hand, having a value like the FoundingDate missing from the DimCompany table can influence the results of a company's rating if the founding date forms an input to the formula. In this case, being able to resolve the missing value will result in improved business performance and decisions. Missing data issues like this need to be resolved or made clear to the end user(s) of the data to ensure appropriate decisions are being made.

3.1.2 Timeliness

During data import, certain records are compared against each other to see whether an existing record is already in the data warehouse. When completing this type of operation, the time at which the check takes place for an update relative to new data being loaded is important. There can be errors that result if a decision is made to speed up the database processing using caches and multiple threads. In this scenario, a record may be seen as not existing for an update, when in reality, the operation to insert the new record simply hasn't occurred yet. In this case, it is important to time the load of different dataset appropriately to avoid these data quality issues.

3.1.3 Consistency

Data format types need to be consistent between each other during data loads. For the "Financial" table for example, this required a manual parsing of the data to ensure the format matched requirements for MS SQL server. This type of data quality issue requires a regular maintenance and check to ensure that there are no changes to the date format of incoming tables, as the change will not be detected during parsing and there will be a significant loss in data load wherever changes take place.

There could also be issues with leading and lagging spaces due to a user interface issue. These spaces need to be removed for data processing. In our case, this was done by always using a TRIM statement for any strings that were being joined.

3.1.4 Operational Sequence

For some data loading or updating, there can be files that contain multiple types of records in the same tables. For example, the "CustomerMgmt.xml" file includes data for customers as well as accounts. When completing an update on data, there are multiple flags the ETL tool can look for to differentiate which item will be updated. In this case, a single customer can have multiple accounts. Anytime there is a new customer added, there will immediately be a new account as well. However, when an update is taking place, it can be either on the customer or the combination customer/account. This type of situation gives us the opportunity to decide if we want to carry out an operation by first identifying the purpose of the operation, and then the entity on which it is to be performed **or** first identify the entity, and then the purpose. Choosing the correct order of operations can significantly improve performance by quickly creating targeted subsets of data for the DBMS to search through.

3.1.5 Uniqueness

Sometimes the way data is represented in a table may not be unique. In the FINWIRE data, for example, the format of CoNameOrCIK attribute could be the company name or the company ID number. An additional check is needed in this to ensure there was consistency in the types of values that were being matched and joined. Not having these types of checks can result in errors occurring.

So how can we manage data quality with these different categories of issues in mind?

3.2 Managing Data Quality Issues

3.2.1 Completeness

For issues relating to completeness, business decisions and logic are needed to drive data requirements and how data should be handled. At a practical level, this requires a close collaboration between the backend data engineering roles and individuals driving the business. These decisions should be made not just at the level of whether or not some data is required or can be left NULL, but all how each attribute is interpreted. An attribute may have the same name or something very similar, but it's context may be derived from the table it originates from. This was the case when looking at the CurrentPrice attribute from fact holdings. Current price could be interpreted as current at the time of the purchase, load, or moment. In this case, given the context of the load, a decision was made on its meaning. These types of decisions must be validated on a regular basis between developers and the business team generating and consuming the data.

3.2.2 Timeliness

When working with parallel data operations, we can specify the order in which parallel operations can take place. For example, we can specify that all data creation steps must be complete before any reading, updating, or deleting takes place. These types of restrictions will be driven by the business scenario in which the ETL is taking place, and a rigorous testing by developers trying to break their own system. This will increase the confidence that the restrictions and rules in place are sufficient.

3.2.3 Consistency

Alignment in data format is critical. This would require seeking out and reviewing all incoming and transformed data to understand not only the syntactic data type but how it is pragmatically defined. A data dictionary can be used to help define and translate between different data sources. Here are some examples (Harish, 2021) where a pragmatic difference in definitions led to serious loss including:

- Mars Climate Orbiter Crash 1999 – Lockheed Martin (USA) used imperial units for propulsion measurements and NASA assumed the conversion to metric was already completed.
- Gimli Glider Fuel Shortage 1983 – plane runs out of fuel midflight due to incorrect amount of fuel used for 22,300 lb payload instead of 22,300 kg payload.

- Incorrect Patient Sedative - the Institute of Safe Medication Practices reported a patient received 0.5 grams of sedative instead of 0.5 grains (0.065 grams) due to misreading instructions.

3.2.4 Operational Sequence

To optimize load times, data that is combined into a single record prior to the ETL process can be reformatted into multiple tables in the staging area or in the initial phase of transformation to optimize database performance. Additionally, setting the loading order correctly is critical for tables that have dependencies.

3.2.5 Uniqueness

Regular expressions (RegEx) can be used to setup some logic behind how non-unique data types are handled, however this might sometimes not be enough if the data types are similar enough. Comprehensive logic and rules will need to be developed to ensure each data type is handled appropriately.

3.3 Test Performance Results

For the benchmarking of the MS SQL server using SSIS, a total of 16 benchmark tests were run at the 4 different scale factors. The total average run times are listed below:

- 3 SF: 57.67 seconds (average of 4 runs)
- 10 SF: 429.00 seconds ≅ 7 minutes (average of 4 tests)
- 20 SF: 1492.33 seconds ≅ 25 minutes (average of 4 tests)
- 30 SF: 3247.33 seconds ≅ 54 minutes (average of 4 tests)

A screenshot of the whole dashboard with the results is shown in Figure . The DBMS showed exponential behavior as the size of the data increased, which can be seen in the second plot (Execution Times Trend – DWH Historical Load).

Figure 3-1 shows a comparison between the initial size of data, the size of the data after transformations, and the execution time of the load. As the SF increases, we see an essentially linear increase in data size for the raw data, however the final output data can be seen to have a slightly exponential increase, with the execution time an even more pronounced exponential fit. Given the unions being completed during the transformation process, these results are not surprising.

To improve the ETL times, there can be some optimization completed on each query by understanding the query plan being used. Some manual indexing on key columns could be completed to aide the query execution as well. Most notably, as data size continues to scale, parallel threads should be implemented to process the data on multiple processors at the same time. Working with parallel process may require additional control measures to ensure unexpected data quality issues do not arise (as discussed in section 3.1.4).

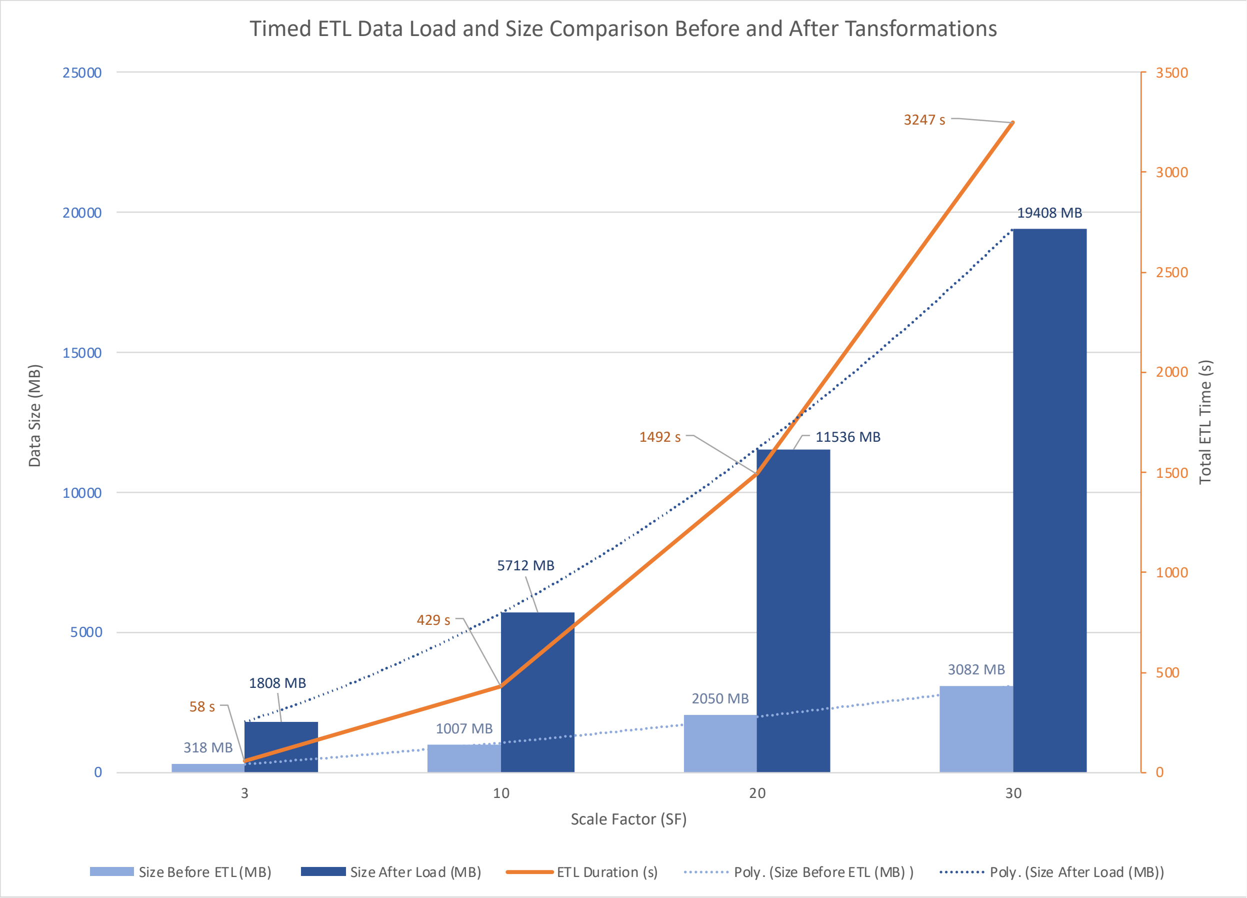
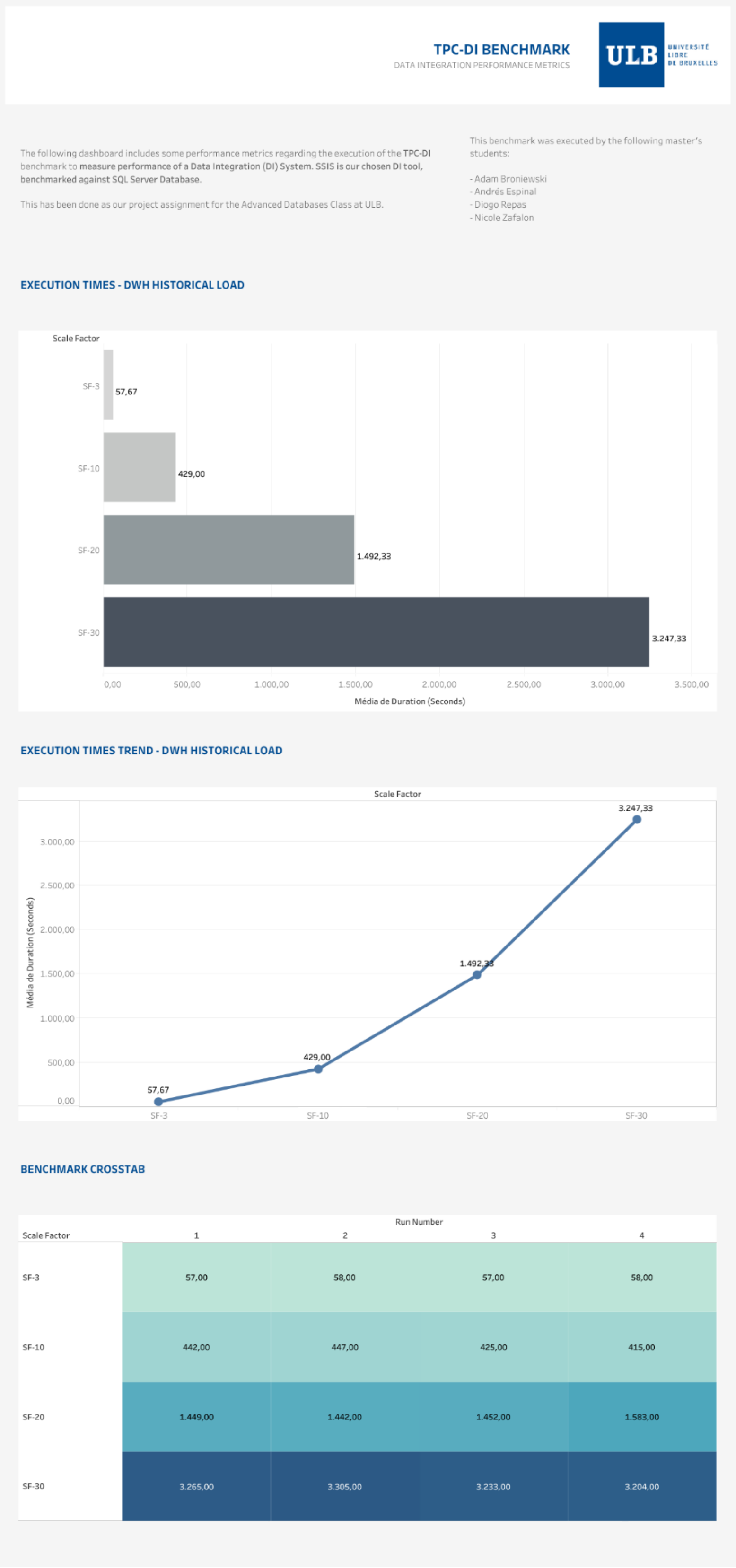


Figure 3-1: Execution time and ETL data size comparison before and after transformations.



TPC-DI BENCHMARK

DATA INTEGRATION PERFORMANCE METRICS

The following dashboard includes some performance metrics regarding the execution of the **TPC-DI** benchmark to **measure performance of a Data Integration (DI) System. SSIS is our chosen DI tool, benchmarked against SQL Server Database.**

This has been done as our project assignment for the Advanced Databases Class at ULB.

This benchmark was executed by the following master's students:

- Adam Broniewski
- Andrés Espinal
- Diogo Repas
- Nicole Zafalon

EXECUTION TIMES - DWH HISTORICAL LOAD

EXECUTION TIMES TREND - DWH HISTORICAL LOAD

BENCHMARK CROSSTAB

	Run Number			
Scale Factor	1	2	3	4
SF-3	57,00	58,00	57,00	58,00
SF-10	442,00	447,00	425,00	415,00
SF-20	1.449,00	1.442,00	1.452,00	1.583,00
SF-30	3.265,00	3.305,00	3.233,00	3.204,00

Figure 3-2: Screenshot of the results in the Tableau dashboard

4 Conclusion

This project report documented the approach taken to implement the TPC-DI process to benchmark the MS SQL Server DBMS using SSIS. The transformations were completed directly using SQL code, and the benchmark results showed an exponential increase in time required for each increase in scale factor.

An overview of data quality issues that are important to consider was also provided, along with a discussion of different ways in which data quality during the ETL process can be managed.

5 Works Cited

Anon., 2021. *TPC Download Current Specs/Sources.* [Online]
Available at: http://tpc.org/tpc_documents_current_versions/current_specifications5.asp

Anon., 2021. *What is SQL Server.* [Online]
Available at: https://www.sqlservertutorial.net/getting-started/what-is-sql-server/
[Accessed 15 11 2021].

Harish, A., 2021. *Simscale.* [Online]
Available at: https://www.simscale.com/blog/2017/12/nasa-mars-climate-orbiter-metric/
[Accessed 21 12 2021].

Moreira, G., Najafzade, N., Detobel, R. & Kashef, S., n.d. *GitHub - TPC-DI.* [Online]
Available at: https://github.com/detobel36/tpc-di/blob/master/createTables.sql
[Accessed 15 11 2021].

Poess, M., Rabl, T., Jacobsen, H.-A. & Caufield, B., 2014. *TPC-DI: The First Industry Benchmark for Data Integration.* Hangzhou, China, s.n.

Yang, Q., Helfert, M. & Ge, M., 2018. Data Quality Problems in TPC-DI Based Data Integration Processes. *ResearchGate.*

APPENDIX A Instructions for Replication

1. Generate files following TCP-DI instructions
2. Use python script to unpack FINWIRE files (Appendix B)
3. Load files into MS SQL database using SSIS
4. Move raw files into schema named “source” in SQL table format (Appendix C)
5. Transform and load all tables from “source” to “dbo” using SQL script in Appendix D.

APPENDIX B PYTHON SCRIPT FINWIRE FILE LOAD

```
import pandas as pd
import re
import os

# Helper functions

# Get a list of column ranges according to the type of schema
def get_col_spec_list( schema_df ):
    schema_df[ 'CharPosEnd' ] = schema_df[ 'Length' ].cumsum()
    schema_df[ 'CharPosStart' ] = ( schema_df.shift( periods = 1, fill_value
= 0 ) )[ 'CharPosEnd' ]
    schema_df.loc[ schema_df[ 'Field' ] == 'PTS', 'CharPosStart' ] = 0

    colspecs = [ ( CharPosStart, CharPosEnd ) for CharPosStart, CharPosEnd in
zip( schema_df[ 'CharPosStart' ], schema_df[ 'CharPosEnd' ] ) ]

    return colspecs

# Get a df that classifies each row in the file as each type of finwire file
def get_finwire_row_types( finwire_df ):
    finwire_type_df = pd.read_fwf( finwire_df, [( 15, 18 )], header = None )
    finwire_type_df.columns = [ 'Type' ]
    finwire_type_df[ 'RowID' ] = finwire_type_df.index
    return finwire_type_df

# Get a list of rows to exclude to keep rows the desired finwire type
def get_finwire_rows_to_exclude( finwire_df, finwire_type ):
    if( finwire_type == 'CMP' ):
        finwire_df = finwire_df[ finwire_df.Type != 'CMP' ]
    elif( finwire_type == 'SEC' ):
        finwire_df = finwire_df[ finwire_df.Type != 'SEC' ]
    elif( finwire_type == 'FIN' ):
        finwire_df = finwire_df[ finwire_df.Type != 'FIN' ]

    return [ RowID for RowID in finwire_df[ 'RowID' ] ]

# Get the finwire file parsed for a particular finwire type
def get_finwire_df_for_type( finwire_file_path, finwire_type, schema_df ):
    row_types_df = get_finwire_row_types( finwire_file_path )

    if( finwire_type == 'CMP' ):
        rows_excluded = get_finwire_rows_to_exclude( row_types_df, 'CMP' )
        df = pd.read_fwf( finwire_file_path, get_col_spec_list( schema_df ),
header = None, skiprows = rows_excluded )
    elif( finwire_type == 'SEC' ):
        rows_excluded = get_finwire_rows_to_exclude( row_types_df, 'SEC' )
        df = pd.read_fwf( finwire_file_path, get_col_spec_list( schema_df ),
header = None, skiprows = rows_excluded  )
    elif( finwire_type == 'FIN' ):
        rows_excluded = get_finwire_rows_to_exclude( row_types_df, 'FIN' )
        df = pd.read_fwf( finwire_file_path, get_col_spec_list( schema_df ),
header = None, skiprows = rows_excluded  )

    df.columns = [ col_name for col_name in schema_df[ 'Field' ] ]
    return df

# Scan a finwire file and return a dictionary with each of its finwire types
dataframes
def parse_finwire_data( finwire_file_path ):
    row_types_df = get_finwire_row_types( finwire_file_path )
    row_types = list( row_types_df[ 'Type' ].drop_duplicates() )
    finwire_dfs = { 'CMP': None, 'SEC': None, 'FIN': None }

    for row_type in row_types:
        if( row_type == 'CMP' ):
            finwire_dfs[ 'CMP' ] = get_finwire_df_for_type(
finwire_file_path, 'CMP', cmp_schema_df )
        elif( row_type == 'SEC' ):
            finwire_dfs[ 'SEC' ] = get_finwire_df_for_type(
finwire_file_path, 'SEC', sec_schema_df )
        elif( row_type == 'FIN' ):
            finwire_dfs[ 'FIN' ] = get_finwire_df_for_type(
finwire_file_path, 'FIN', fin_schema_df )

    return finwire_dfs

# Parse a list of finwire files and return each as a collection of finwire
type dataframes
def get_finwire_dfs( finwire_file_paths ):
    cmp_df = pd.DataFrame()
    sec_df = pd.DataFrame()
    fin_df = pd.DataFrame()

    for file_path in finwire_file_paths:
        finwire_df_dict = parse_finwire_data( file_path )

        for finwire_type in finwire_df_dict.keys():
            if( finwire_type == 'CMP' ):
                cmp_df = cmp_df.append( finwire_df_dict[ 'CMP' ] )
            elif( finwire_type == 'SEC' ):
                sec_df = sec_df.append( finwire_df_dict[ 'SEC' ] )
            elif( finwire_type == 'FIN' ):
                fin_df = fin_df.append( finwire_df_dict[ 'FIN' ] )

    return { 'CMP': cmp_df, 'SEC': sec_df, 'FIN': fin_df }


# Define file paths
root_path = r'E:\OneDrive - Université Libre de Bruxelles\Mes fichiers\419-
SQLServerBenchmark Project\TPC-DI'
finwire_source_files_path = root_path + r'\Data\SF20\Batch1'
finwire_schema_files_path = root_path + r'\Helpers\Datasets'

# Read the schema dataframes
cmp_schema_df = pd.read_csv( finwire_schema_files_path +
'\CMP_Records_Schema.csv' )
fin_schema_df = pd.read_csv( finwire_schema_files_path +
'\FIN_Records_Schema.csv' )
sec_schema_df = pd.read_csv( finwire_schema_files_path +
'\SEC_Records_Schema.csv' )
```

```
# Get the column specifications from the schema dataframes
cmp_colspecs = get_col_spec_list( cmp_schema_df )
fin_colspecs = get_col_spec_list( fin_schema_df )
sec_colspecs = get_col_spec_list( sec_schema_df )

# Search for all the finwire files
finwire_files = [ finwire_source_files_path + '\\' + file for file in
os.listdir( finwire_source_files_path ) if re.search( r'^FINWIRE......$',
file ) ]
g = get_finwire_dfs( finwire_files )

g[ 'CMP' ]['FoundingDate'] = g[ 'CMP'
]['FoundingDate'].astype(pd.Int64Dtype())

g[ 'CMP' ].to_csv( finwire_source_files_path + r'\{0}.csv'.format( 'CMP' ),
index=None )
g[ 'FIN' ].to_csv( finwire_source_files_path + r'\{0}.csv'.format( 'FIN' ),
index=None )
g[ 'SEC' ].to_csv( finwire_source_files_path + r'\{0}.csv'.format( 'SEC' ),
index=None )
```

APPENDIX C   STAGING AREA GENERATION

```
-- Original SQL Script pulled from TPC-DI Project completed by:
-- Gonçalo Moreira, Nazrin Najafzade, Rémy Detobel, Shafagh Kashef
-- https://github.com/detobel36/tpc-di/blob/master/createTables.sql

-- Code has been reviewed and checked against TPC-DI_SPEC_v1.1.0.pdf

CREATE TABLE DimBroker  (
    SK_BrokerID  INTEGER NOT NULL IDENTITY (1,1) PRIMARY KEY,
    BrokerID  INTEGER NOT NULL,
    ManagerID  INTEGER,
    FirstName       CHAR(50) NOT NULL,
    LastName       CHAR(50) NOT NULL,
    MiddleInitial       CHAR(1),
    Branch       CHAR(50),
    Office       CHAR(50),
    Phone       CHAR(14),
    IsCurrent BIT NOT NULL,
    BatchID INTEGER NOT NULL,
    EffectiveDate date NOT NULL,
    EndDate date NOT NULL
);



CREATE TABLE DimCompany (
    SK_CompanyID INTEGER NOT NULL IDENTITY (1,1) PRIMARY KEY,
    CompanyID INTEGER NOT NULL,
    Status CHAR(10) Not NULL,
    Name CHAR(60) Not NULL,
    Industry CHAR(50) Not NULL,
    SPrating CHAR(4),
    isLowGrade BIT,
    CEO CHAR(100) Not NULL,
    AddressLine1 CHAR(80),
    AddressLine2 CHAR(80),
    PostalCode CHAR(12) Not NULL,
    City CHAR(25) Not NULL,
    StateProv CHAR(20) Not NULL,
    Country CHAR(24),
    Description CHAR(150) Not NULL,
    FoundingDate DATE,
    IsCurrent BIT Not NULL,
    BatchID numeric(5) Not NULL,
    EffectiveDate DATE Not NULL,
    EndDate DATE Not NULL
);

CREATE TABLE DimCustomer  (
    SK_CustomerID  INTEGER NOT NULL IDENTITY (1,1) PRIMARY KEY,
    CustomerID INTEGER NOT NULL,
    TaxID CHAR(20) NOT NULL,
    Status CHAR(10) NOT NULL,
    LastName CHAR(30) NOT NULL,
    FirstName CHAR(30) NOT NULL,
    MiddleInitial CHAR(1),
    Gender CHAR(1),
    Tier Integer,
    DOB date NOT NULL,
    AddressLine1  varchar(80) NOT NULL,
    AddressLine2  varchar(80),
    PostalCode    char(12) NOT NULL,
    City   char(25) NOT NULL,
    StateProv     char(20) NOT NULL,
    Country       char(24),
    Phone1 char(30),
    Phone2 char(30),
    Phone3 char(30),
    Email1 char(50),
    Email2 char(50),
    NationalTaxRateDesc  varchar(50),
    NationalTaxRate      numeric(6,5),
    LocalTaxRateDesc     varchar(50),
    LocalTaxRate  numeric(6,5),
    AgencyID      char(30),
    CreditRating integer,
    NetWorth      numeric(10),
    MarketingNameplate varchar(100),
    IsCurrent BIT NOT NULL,
    BatchID INTEGER NOT NULL,
    EffectiveDate date NOT NULL,
    EndDate date NOT NULL
);


CREATE TABLE DimAccount  (
    SK_AccountID  INTEGER NOT NULL IDENTITY (1,1) PRIMARY KEY,
    AccountID  INTEGER NOT NULL,
    SK_BrokerID  INTEGER NOT NULL REFERENCES DimBroker (SK_BrokerID),
    SK_CustomerID  INTEGER NOT NULL REFERENCES DimCustomer (SK_CustomerID),
    Status       CHAR(10) NOT NULL,
    AccountDesc       varchar(50),
    TaxStatus  INTEGER NOT NULL CHECK (TaxStatus = 0 OR TaxStatus = 1 OR
TaxStatus = 2),
    IsCurrent BIT NOT NULL,
    BatchID INTEGER NOT NULL,
    EffectiveDate date NOT NULL,
    EndDate date NOT NULL
);
CREATE TABLE DimDate (
    SK_DateID INTEGER Not NULL PRIMARY KEY,
    DateValue DATE Not NULL,
    DateDesc CHAR(20) Not NULL,
    CalendarYearID numeric(4) Not NULL,
    CalendarYearDesc CHAR(20) Not NULL,
    CalendarQtrID numeric(5) Not NULL,
    CalendarQtrDesc CHAR(20) Not NULL,
    CalendarMonthID numeric(6) Not NULL,
    CalendarMonthDesc CHAR(20) Not NULL,
    CalendarWeekID numeric(6) Not NULL,
    CalendarWeekDesc CHAR(20) Not NULL,
    DayOfWeeknumeric numeric(1) Not NULL,
    DayOfWeekDesc CHAR(10) Not NULL,
    FiscalYearID numeric(4) Not NULL,
    FiscalYearDesc CHAR(20) Not NULL,
    FiscalQtrID numeric(5) Not NULL,
    FiscalQtrDesc CHAR(20) Not NULL,
    HolidayFlag BIT
);


CREATE TABLE DimSecurity(
    SK_SecurityID INTEGER Not NULL IDENTITY (1,1) PRIMARY KEY,
    Symbol CHAR(15) Not NULL,
    Issue CHAR(6) Not NULL,
    Status CHAR(10) Not NULL,
    Name CHAR(70) Not NULL,
    ExchangeID CHAR(6) Not NULL,
    SK_CompanyID INTEGER Not NULL REFERENCES DimCompany (SK_CompanyID),
    SharesOutstanding INTEGER Not NULL,
    FirstTrade DATE Not NULL,
    FirstTradeOnExchange DATE Not NULL,
    Dividend numeric(2) Not NULL,
    IsCurrent BIT Not NULL,
    BatchID numeric(5) Not NULL,
    EffectiveDate DATE Not NULL,
    EndDate DATE Not NULL
);

CREATE TABLE DimTime (
    SK_TimeID INTEGER Not NULL PRIMARY KEY,
    TimeValue TIME Not NULL,
    HourID numeric(2) Not NULL,
    HourDesc CHAR(20) Not NULL,
    MinuteID numeric(2) Not NULL,
    MinuteDesc CHAR(20) Not NULL,
    SecondID numeric(2) Not NULL,
    SecondDesc CHAR(20) Not NULL,
    MarketHoursFlag BIT,
    OfficeHoursFlag BIT
);

CREATE TABLE DimTrade (
    TradeID INTEGER Not NULL,
    SK_BrokerID INTEGER REFERENCES DimBroker (SK_BrokerID),
    SK_CreateDateID INTEGER Not NULL REFERENCES DimDate (SK_DateID),
    SK_CreateTimeID INTEGER Not NULL REFERENCES DimTime (SK_TimeID),
    SK_CloseDateID INTEGER REFERENCES DimDate (SK_DateID),
    SK_CloseTimeID INTEGER REFERENCES DimTime (SK_TimeID),
    Status CHAR(10) Not NULL,
    DT_Type CHAR(12) Not NULL,
    CashFlag BIT Not NULL,
    SK_SecurityID INTEGER Not NULL REFERENCES DimSecurity (SK_SecurityID),
    SK_CompanyID INTEGER Not NULL REFERENCES DimCompany (SK_CompanyID),
    Quantity numeric(6,0) Not NULL,
    BidPrice numeric(8,2) Not NULL,
    SK_CustomerID INTEGER Not NULL REFERENCES DimCustomer (SK_CustomerID),
    SK_AccountID INTEGER Not NULL REFERENCES DimAccount (SK_AccountID),
    ExecutedBy CHAR(64) Not NULL,
    TradePrice numeric(8,2),
    Fee numeric(10,2),
    Commission numeric(10,2),
    Tax numeric(10,2),
    BatchID numeric(5) Not Null
);

CREATE TABLE DImessages (
    MessageDateAndTime TIMESTAMP Not NULL,
    BatchID numeric(5) Not NULL,
    MessageSource CHAR(30),
    MessageText CHAR(50) Not NULL,
    MessageType CHAR(12) Not NULL,
    MessageData CHAR(100)
);

CREATE TABLE FactCashBalances (
    SK_CustomerID INTEGER Not Null REFERENCES DimCustomer (SK_CustomerID),
    SK_AccountID INTEGER Not Null REFERENCES DimAccount (SK_AccountID),
    SK_DateID INTEGER Not Null REFERENCES DimDate (SK_DateID),
    Cash numeric(15,2) Not Null,
    BatchID numeric(5)
);

CREATE TABLE FactHoldings (
    TradeID INTEGER Not NULL,
    CurrentTradeID INTEGER Not Null,
    SK_CustomerID INTEGER Not NULL REFERENCES DimCustomer (SK_CustomerID),
    SK_AccountID INTEGER Not NULL REFERENCES DimAccount (SK_AccountID),
    SK_SecurityID INTEGER Not NULL REFERENCES DimSecurity (SK_SecurityID),
    SK_CompanyID INTEGER Not NULL REFERENCES DimCompany (SK_CompanyID),
    SK_DateID INTEGER Not NULL REFERENCES DimDate (SK_DateID),
    SK_TimeID INTEGER Not NULL REFERENCES DimTime (SK_TimeID),
    CurrentPrice INTEGER CHECK (CurrentPrice > 0) ,
    CurrentHolding numeric(6) Not NULL,
    BatchID numeric(5)
);

CREATE TABLE FactMarketHistory (
    SK_SecurityID INTEGER Not Null REFERENCES DimSecurity (SK_SecurityID),
    SK_CompanyID INTEGER Not Null REFERENCES DimCompany (SK_CompanyID),
    SK_DateID INTEGER Not Null REFERENCES DimDate (SK_DateID),
    PERatio numeric(10,2),
    Yield numeric(5,2) Not Null,
    FiftyTwoWeekHigh numeric(8,2) Not Null,
    SK_FiftyTwoWeekHighDate INTEGER Not Null,
    FiftyTwoWeekLow numeric(8,2) Not Null,
    SK_FiftyTwoWeekLowDate INTEGER Not Null,
    ClosePrice numeric(8,2) Not Null,
    DayHigh numeric(8,2) Not Null,
    DayLow numeric(8,2) Not Null,
    Volume numeric(12) Not Null,
    BatchID numeric(5)
);

CREATE TABLE FactWatches (
    SK_CustomerID INTEGER Not NULL REFERENCES DimCustomer (SK_CustomerID),
    SK_SecurityID INTEGER Not NULL REFERENCES DimSecurity (SK_SecurityID),
    SK_DateID_DatePlaced INTEGER Not NULL REFERENCES DimDate (SK_DateID),
    SK_DateID_DateRemoved INTEGER REFERENCES DimDate (SK_DateID),
    BatchID numeric(5) Not Null
);

CREATE TABLE Industry (
    IN_ID CHAR(2) Not NULL,
    IN_NAME CHAR(50) Not NULL,
    IN_SC_ID CHAR(4) Not NULL
);

CREATE TABLE Financial (
    SK_CompanyID INTEGER Not NULL REFERENCES DimCompany (SK_CompanyID),
    FI_YEAR numeric(4) Not NULL,
    FI_QTR numeric(1) Not NULL,
    FI_QTR_START_DATE DATE Not NULL,
    FI_REVENUE numeric(15,2) Not NULL,
    FI_NET_EARN numeric(15,2) Not NULL,
    FI_BASIC_EPS numeric(10,2) Not NULL,
    FI_DILUT_EPS numeric(10,2) Not NULL,
    FI_MARGIN numeric(10,2) Not NULL,
    FI_INVENTORY numeric(15,2) Not NULL,
    FI_ASSETS numeric(15,2) Not NULL,
    FI_LIABILITY numeric(15,2) Not NULL,
    FI_OUT_BASIC numeric(12) Not NULL,
    FI_OUT_DILUT numeric(12) Not NULL
);

CREATE TABLE Prospect (
    AgencyID CHAR(30) NOT NULL UNIQUE,
    SK_RecordDateID INTEGER NOT NULL,
    SK_UpdateDateID INTEGER NOT NULL REFERENCES DimDate (SK_DateID),
    BatchID numeric(5) NOT NULL,
    IsCustomer BIT NOT NULL,
    LastName CHAR(30) NOT NULL,
    FirstName CHAR(30) NOT NULL,
    MiddleInitial CHAR(1),
    Gender CHAR(1),
    AddressLine1 CHAR(80),
    AddressLine2 CHAR(80),
    PostalCode CHAR(12),
    City CHAR(25) NOT NULL,
    State CHAR(20) NOT NULL,
    Country CHAR(24),
    Phone CHAR(30),
    Income numeric(9),
    numberCars numeric(2),
    numberChildren numeric(2),
    MaritalStatus CHAR(1),
    Age numeric(3),
    CreditRating numeric(4),
    OwnOrRentFlag CHAR(1),
    Employer CHAR(30),
    numberCreditCards numeric(2),
    NetWorth numeric(12),
    MarketingNameplate CHAR(100)
);

CREATE TABLE StatusType (
    ST_ID CHAR(4) Not NULL,
    ST_NAME CHAR(10) Not NULL
);

CREATE TABLE TaxRate (
    TX_ID CHAR(4) Not NULL,
    TX_NAME CHAR(50) Not NULL,
    TX_RATE numeric(6,5) Not NULL
);

CREATE TABLE TradeType (
    TT_ID CHAR(3) Not NULL,
    TT_NAME CHAR(12) Not NULL,
    TT_IS_SELL numeric(1) Not NULL,
    TT_IS_MRKT numeric(1) Not NULL
);

CREATE TABLE AuditTable (
    DataSet CHAR(20) Not Null,
    BatchID numeric(5),
    AT_Date DATE,
    AT_Attribute CHAR(50),
    AT_Value numeric(15),
    DValue numeric(15,5)
);

CREATE INDEX PIndex ON DimTrade (TradeID);
```

APPENDIX D   HISTORICAL LOAD SQL SCRIPT

```
-- DimDate
INSERT INTO dbo.DimDate
    (SK_DateID,
    DateValue,
    DateDesc,
    CalendarYearID,
    CalendarYearDesc,
    CalendarQtrID,
    CalendarQtrDesc,
    CalendarMonthID,
    CalendarMonthDesc,
    CalendarWeekID,
    CalendarWeekDesc,
    DayOfWeeknumeric,
    DayOfWeekDesc,
    FiscalYearID,
    FiscalYearDesc,
    FiscalQtrID,
    FiscalQtrDesc,
    HolidayFlag)
SELECT
    SK_DateID,
    DateValue,
    DateDesc,
    CalendarYearID,
    CalendarYearDesc,
    CalendarQtrID,
    CalendarQtrDesc,
    CalendarMonthID,
    CalendarMonthDesc,
    CalendarWeekID,
    CalendarWeekDesc,
    DayOfWeekNum,
    DayOfWeekDesc,
    FiscalYearID,
    FiscalYearDesc,
    FiscalQtrID,
    FiscalQtrDesc,
    HolidayFlag
FROM Source.Date;
GO

-- DimTime
INSERT INTO dbo.DimTime ( SK_TimeID, TimeValue, HourID, HourDesc, MinuteID,
MinuteDesc, SecondID, SecondDesc, MarketHoursFlag, OfficeHoursFlag )
SELECT  SK_TimeID, TimeValue, HourID, HourDesc, MinuteID, MinuteDesc,
SecondID, SecondDesc, MarketHoursFlag, OfficeHoursFlag
FROM    Source.Time
GO

-- StatusType
INSERT INTO dbo.StatusType ( ST_ID, ST_NAME )
SELECT  ST_ID, ST_NAME
```

```
FROM    Source.StatusType

GO

-- TaxRate
INSERT INTO dbo.TaxRate ( TX_ID, TX_NAME, TX_RATE )
SELECT  TX_ID, TX_NAME, TX_RATE
FROM    Source.TaxRate

GO

-- TradeType
INSERT INTO dbo.TradeType ( TT_ID, TT_NAME, TT_IS_SELL, TT_IS_MRKT )
SELECT  TT_ID, TT_NAME, TT_IS_SELL, TT_IS_MRKT
FROM    Source.TradeType
GO

-- DimBroker
INSERT INTO dbo.DimBroker (IsCurrent, EffectiveDate, EndDate, BatchID,
BrokerID, ManagerID, FirstName, LastName, MiddleInitial, Branch, Office,
Phone)
SELECT
   1 AS IsCurrent,
   (SELECT MIN(DateValue) FROM DimDate) as EffectiveDate,
   '9999-12-31' AS EndDate,
   1 as BatchID,
   EmployeeID as BrokerID,
   ManagerID as ManagerID,
   EmployeeFirstName as FirstName,
   EmployeeLastName as LastName,
   EmployeeMI as MiddleInitial,
   EmployeeBranch as Branch,
   EmployeeOffice as Office,
   EmployeePhone as Phone
FROM Source.HR
WHERE EmployeeJobCode = 314

GO

-- DimCompany
INSERT INTO dbo.DimCompany (IsCurrent, EffectiveDate, EndDate, BatchID,
CompanyID, Name, SPrating, CEO, Description, FoundingDate, AddressLine1,
AddressLine2, PostalCode, City, StateProv, Country, Status, Industry,
IsLowGrade)
SELECT
   CASE WHEN LEAD( (SELECT TOP 1 BatchDate FROM Source.BatchDate) ) OVER (
PARTITION BY CIK ORDER BY PTS ASC ) IS NULL THEN 1 ELSE 0 END AS IsCurrent,
   (SELECT TOP 1 BatchDate FROM Source.BatchDate) as EffectiveDate,
   COALESCE( LEAD( (SELECT TOP 1 BatchDate FROM Source.BatchDate) ) OVER (
PARTITION BY CIK ORDER BY PTS ASC ), '9999-12-31' ) AS EndDate,
   1 as BatchID,
   CIK as CompanyID,
   CompanyName as Name,
   SPrating as SPRating,
   CEOname as CEO,
   Description,
   FoundingDate,
   AddrLine1 as AddressLine1,
   AddrLine2 as AddressLine2,
   PostalCode,
   City,
   StateProvince as State_Prov,
   Country,
   S.ST_NAME as Status,
   I.IN_NAME as Industry,
   (CASE WHEN SPrating LIKE 'A%' OR SPrating LIKE 'BBB%' THEN 0 ELSE 1 END)
as IsLowGrade
FROM Source.FinwireCMP CMP, Source.StatusType S, Source.Industry I
WHERE CMP.Status = S.ST_ID
AND CMP.IndustryID = I.IN_ID

GO

-- DimCustomer
   WITH Customers_Preproc AS (
      SELECT CXML.C_ID AS CustomerID
         , TRIM( CXML.C_TAX_ID ) AS TaxID
         , TRIM( UPPER( CASE WHEN CXML.C_GNDR NOT IN ( 'm', 'f' ) OR
CXML.C_GNDR IS  NULL THEN 'u' ELSE CXML.C_GNDR END ) ) AS Gender
         , CXML.C_TIER AS Tier
         , CXML.C_DOB AS DOB
         , TRIM( CCIN.C_PRIM_EMAIL ) AS Email1
         , TRIM( CCIN.C_ALT_EMAIL ) AS Email2
         , TRIM( NXML.C_F_NAME ) AS FirstName
         , TRIM( NXML.C_M_NAME ) AS MiddleInitial
         , TRIM( NXML.C_L_NAME ) AS LastName
         , TRIM( ADXML.C_ADLINE1 ) AS AddressLine1
         , TRIM( ADXML.C_ADLINE2 ) AS AddressLine2
         , TRIM( ADXML.C_ZIPCODE ) AS PostalCode
         , TRIM( ADXML.C_CITY ) AS City
         , TRIM( ADXML.C_STATE_PROV ) AS StateProv
         , TRIM( ADXML.C_CTRY ) AS Country
         , CASE
            WHEN CP1XML.C_CTRY_CODE IS NOT NULL AND CP1XML.C_AREA_CODE IS NOT
NULL AND CP1XML.C_LOCAL IS NOT NULL
               THEN '+' + CP1XML.C_CTRY_CODE + ' (' + CP1XML.C_AREA_CODE + ')
' + CP1XML.C_LOCAL
            WHEN CP1XML.C_CTRY_CODE IS NULL AND ( CP1XML.C_AREA_CODE IS NOT
NULL AND CP1XML.C_LOCAL IS NOT NULL )
               THEN '(' + CP1XML.C_AREA_CODE + ') ' + CP1XML.C_LOCAL
            WHEN ( CP1XML.C_CTRY_CODE IS NULL AND CP1XML.C_AREA_CODE IS NULL
) AND CP1XML.C_LOCAL IS NOT NULL
               THEN CP1XML.C_LOCAL
         END AS Phone1_V1
         , CASE
            WHEN CP2XML.C_CTRY_CODE IS NOT NULL AND CP2XML.C_AREA_CODE IS NOT
NULL AND CP2XML.C_LOCAL IS NOT NULL
               THEN '+' + CP2XML.C_CTRY_CODE + ' (' + CP2XML.C_AREA_CODE + ')
' + CP2XML.C_LOCAL
            WHEN CP2XML.C_CTRY_CODE IS NULL AND ( CP2XML.C_AREA_CODE IS NOT
NULL AND CP2XML.C_LOCAL IS NOT NULL )
               THEN '(' + CP2XML.C_AREA_CODE + ') ' + CP2XML.C_LOCAL
            WHEN ( CP2XML.C_CTRY_CODE IS NULL AND CP2XML.C_AREA_CODE IS NULL
) AND CP2XML.C_LOCAL IS NOT NULL
```

```
               THEN CP2XML.C_LOCAL
         END AS Phone2_V1
         , CASE
            WHEN CP3XML.C_CTRY_CODE IS NOT NULL AND CP3XML.C_AREA_CODE IS NOT
NULL AND CP3XML.C_LOCAL IS NOT NULL
               THEN '+' + CP3XML.C_CTRY_CODE + ' (' + CP3XML.C_AREA_CODE + ')
' + CP3XML.C_LOCAL
            WHEN CP3XML.C_CTRY_CODE IS NULL AND ( CP3XML.C_AREA_CODE IS NOT
NULL AND CP3XML.C_LOCAL IS NOT NULL )
               THEN '(' + CP3XML.C_AREA_CODE + ') ' + CP3XML.C_LOCAL
            WHEN ( CP3XML.C_CTRY_CODE IS NULL AND CP3XML.C_AREA_CODE IS NULL
) AND CP3XML.C_LOCAL IS NOT NULL
               THEN CP3XML.C_LOCAL
         END AS Phone3_V1
         , TRIM( TR.TX_NAME ) AS NationalTaxRateDesc
         , TR.TX_RATE AS NationalTaxRate
         , TRIM( TR2.TX_NAME ) AS LocalTaxRateDesc
         , TR2.TX_RATE AS LocalTaxRate
         , CP1XML.C_EXT AS C_EXT1
         , CP2XML.C_EXT AS C_EXT2
         , CP3XML.C_EXT AS C_EXT3
         , TRIM( P.AgencyID ) AS AgencyID
         , P.CreditRating
         , P.NetWorth
         , CASE
            WHEN P.NetWorth > 1000000 OR P.Income > 200000 THEN 'HighValue'
            WHEN P.NumberChildren > 3 OR P.NumberCreditCards > 5 THEN
'Expenses'
            WHEN P.Age > 45 THEN 'Boomer'
            WHEN P.Income < 50000 OR P.CreditRating < 600 OR P.NetWorth <
100000 THEN 'MoneyAlert'
            WHEN P.NumberCars > 3 OR P.NumberCreditCards > 7 THEN 'Spender'
            WHEN P.Age < 25 AND P.NetWorth > 1000000 THEN 'Inherited'
         END AS MarketingNameplate
         , AXML.ActionType
         , AXML.ActionTS
      FROM [Source].[CustomerXML] CXML
         LEFT JOIN [Source].[ActionXML] AXML
            ON CXML.Action_Id = AXML.Action_Id
         LEFT JOIN [Source].[NameXML] NXML
            ON CXML.Customer_Id = NXML.Customer_Id
         LEFT JOIN [Source].[ContactInfoXML] CCIN
            ON CXML.Customer_Id = CCIN.Customer_Id
         LEFT JOIN [Source].[AddressXML] ADXML
            ON CXML.Customer_Id = ADXML.Customer_Id
         LEFT JOIN [Source].[TaxInfoXML] TXML
            ON CXML.Customer_Id = TXML.Customer_Id
         LEFT JOIN [Source].[TaxRate] TR
            ON TXML.C_NAT_TX_ID = TR.TX_ID
         LEFT JOIN [Source].[TaxRate] TR2
            ON TXML.C_LCL_TX_ID = TR2.TX_ID
         LEFT JOIN [Source].[C_PHONE_1_XML] CP1XML
            ON CCIN.ContactInfo_Id = CP1XML.ContactInfo_Id
         LEFT JOIN [Source].[C_PHONE_2_XML] CP2XML
            ON CCIN.ContactInfo_Id = CP2XML.ContactInfo_Id
         LEFT JOIN [Source].[C_PHONE_3_XML] CP3XML
            ON CCIN.ContactInfo_Id = CP3XML.ContactInfo_Id
         LEFT JOIN [Source].[Prospect] P
            ON
               -- Join on FirstName if exists
               COALESCE( UPPER( TRIM( NXML.C_F_NAME ) ), ' ' ) = COALESCE(
UPPER( TRIM( P.FirstName ) ), ' ' )
               -- Join on LastName if exists
               AND COALESCE( UPPER( TRIM( NXML.C_L_NAME ) ), ' ' ) =
COALESCE( UPPER( TRIM( P.LastName ) ), ' ' )
               -- Join on AddressLine1 if exists
               AND COALESCE( UPPER( TRIM( ADXML.C_ADLINE1 ) ), ' ' ) =
COALESCE( UPPER( TRIM( P.AddressLine1 ) ), ' ' )
               -- Join on AddressLine2 if exists
               AND COALESCE( UPPER( TRIM( ADXML.C_ADLINE2 ) ), ' ' ) =
COALESCE( UPPER( TRIM( P.AddressLine2 ) ), ' ' )
               -- Join on PostalCode if exists
               AND COALESCE( UPPER( TRIM( ADXML.C_ZIPCODE ) ), ' ' ) =
COALESCE( UPPER( TRIM( P.PostalCode ) ), ' ' )
   )

   , Customers AS (
      SELECT *
         , CASE WHEN C_EXT1 IS NOT NULL THEN Phone1_V1 + C_EXT1 ELSE
Phone1_V1 END Phone1
         , CASE WHEN C_EXT2 IS NOT NULL THEN Phone2_V1 + C_EXT2 ELSE
Phone2_V1 END Phone2
         , CASE WHEN C_EXT3 IS NOT NULL THEN Phone3_V1 + C_EXT3 ELSE
Phone3_V1 END Phone3
      FROM Customers_Preproc
   )
      -- These are the three cases. Take big table and subsect it into NEW,
UPDATED, and INACTIVE
   , CustomersNew AS (
      SELECT *, 'ACTIVE' AS [Status] FROM Customers WHERE ActionType = 'NEW'
   )

   , CustomersUpd AS (
      SELECT * FROM Customers WHERE ActionType = 'UPDCUST'
   )

   , CustomersInactive AS (
      SELECT C_ID, ActionTS
      FROM [Source].[CustomerXML] CXML
         INNER JOIN [Source].[ActionXML] AXML
            ON CXML.Action_Id = AXML.Action_Id
      WHERE ActionType = 'INACT'
   )
   , CustomersNewAndUpd AS (
      SELECT CustomerID
            , TaxID
            , 'ACTIVE' AS [Status]
            , LastName
            , FirstName
            , MiddleInitial
            , Gender
            , Tier
            , DOB
            , AddressLine1
            , AddressLine2
            , PostalCode
            , City
            , StateProv
            , Country
            , Phone1
            , Phone2
            , Phone3
            , Email1
            , Email2
            , NationalTaxRateDesc
            , NationalTaxRate
            , LocalTaxRateDesc
            , LocalTaxRate
            , AgencyID
            , CreditRating
            , NetWorth
            , MarketingNameplate
            , ActionTS
            , ActionType
            --, AS EffectiveDate
            --, AS EndDate
      FROM CustomersNew
      UNION
      -- UPDCUST
      SELECT  NC.CustomerID
            , COALESCE( UC.TaxID, NC.TaxID ) AS TaxID
            , NC.[Status] AS [Status]
            , COALESCE( UC.LastName, NC.LastName ) AS LastName
            , COALESCE( UC.FirstName, NC.FirstName ) AS FirstName
            , COALESCE( UC.MiddleInitial, NC.MiddleInitial ) AS MiddleInitial
            , COALESCE( UC.Gender, NC.Gender ) AS Gender
            , COALESCE( UC.Tier, NC.Tier ) AS Tier
            , COALESCE( UC.DOB, NC.DOB ) AS DOB
            , COALESCE( UC.AddressLine1, NC.AddressLine1 ) AS AddressLine1
            , COALESCE( UC.AddressLine2, NC.AddressLine2 ) AS AddressLine2
            , COALESCE( UC.PostalCode, NC.PostalCode ) AS PostalCode
            , COALESCE( UC.City, NC.City ) AS City
            , COALESCE( UC.StateProv, NC.StateProv ) AS StateProv
            , COALESCE( UC.Country, NC.Country ) AS Country
            , COALESCE( UC.Phone1, NC.Phone1 ) AS Phone1
            , COALESCE( UC.Phone2, NC.Phone2 ) AS Phone2
            , COALESCE( UC.Phone3, NC.Phone3 ) AS Phone3
            , COALESCE( UC.Email1, NC.Email1 ) AS Email1
            , COALESCE( UC.Email2, NC.Email2 ) AS Email2
            , COALESCE( UC.NationalTaxRateDesc, NC.NationalTaxRateDesc ) AS
NationalTaxRateDesc
            , COALESCE( UC.NationalTaxRate, NC.NationalTaxRate ) AS
NationalTaxRate
            , COALESCE( UC.LocalTaxRateDesc, NC.LocalTaxRateDesc ) AS
LocalTaxRateDesc
            , COALESCE( UC.LocalTaxRate, NC.LocalTaxRate ) AS LocalTaxRate
            , COALESCE( UC.AgencyID, NC.AgencyID ) AS AgencyID
            , COALESCE( UC.CreditRating, NC.CreditRating ) AS CreditRating
            , COALESCE( UC.NetWorth, NC.NetWorth ) AS NetWorth
            , COALESCE( UC.MarketingNameplate, NC.MarketingNameplate ) AS
MarketingNameplate
```

```
            , UC.ActionTS
            , UC.ActionType
      FROM CustomersNew NC
         INNER JOIN CustomersUpd UC
            ON NC.CustomerID = UC.CustomerID
   )

   , CustomersFinal AS (
      -- NEW and UPDCUST
      SELECT *
      FROM CustomersNewAndUpd
      UNION
      -- INACT
      SELECT CNU.CustomerID
               , CNU.TaxID
               , 'INACTIVE' AS [Status]
               , CNU.LastName
               , CNU.FirstName
               , CNU.MiddleInitial
               , CNU.Gender
               , CNU.Tier
               , CNU.DOB
               , CNU.AddressLine1
               , CNU.AddressLine2
               , CNU.PostalCode
               , CNU.City
               , CNU.StateProv
               , CNU.Country
               , CNU.Phone1
               , CNU.Phone2
               , CNU.Phone3
               , CNU.Email1
               , CNU.Email2
               , CNU.NationalTaxRateDesc
               , CNU.NationalTaxRate
               , CNU.LocalTaxRateDesc
               , CNU.LocalTaxRate
               , CNU.AgencyID
               , CNU.CreditRating
               , CNU.NetWorth
               , CNU.MarketingNameplate
               , CI.ActionTS
               , 'INACT' AS ActionType
      FROM CustomersNewAndUpd CNU
         INNER JOIN CustomersInactive CI
            ON CNU.CustomerID = CI.C_ID
         INNER JOIN (
            SELECT CustomerID, MAX( ActionTS ) ActionTSLatestCustomer
            FROM CustomersNewAndUpd
            GROUP BY CustomerID
         ) LC
         ON CNU.CustomerID = LC.CustomerID AND CNU.ActionTS =
LC.ActionTSLatestCustomer
   )

   INSERT INTO dbo.DimCustomer
   SELECT CustomerID
```

```
         , TaxID
         , [Status]
         , LastName
         , FirstName
         , MiddleInitial
         , Gender
         , Tier
         , DOB
         , AddressLine1
         , AddressLine2
         , PostalCode
         , City
         , StateProv
         , Country
         , Phone1
         , Phone2
         , Phone3
         , Email1
         , Email2
         , NationalTaxRateDesc
         , NationalTaxRate
         , LocalTaxRateDesc
         , LocalTaxRate
         , AgencyID
         , CreditRating
         , NetWorth
         , MarketingNameplate
         , CASE WHEN LEAD( ActionTS ) OVER ( PARTITION BY CustomerID ORDER BY
ActionTS ASC ) IS NULL THEN 1 ELSE 0 END AS IsCurrent
         , 1 AS BatchID
         , ActionTS AS EffectiveDate
         , COALESCE( LEAD( ActionTS ) OVER ( PARTITION BY CustomerID ORDER BY
ActionTS ASC ), '9999-12-31 00:00:00' ) AS EndDate
   FROM CustomersFinal
   ORDER BY CustomerID, ActionTS ASC
GO

-- DimAccount
WITH Accounts AS (
   SELECT Acc.CA_ID AS AccountID
      , Br.SK_BrokerID AS SK_BrokerID
      , DimC.SK_CustomerID AS SK_CustomerID
      , Acc.Customer_Id AS Customer_Id
      , Acc.CA_NAME AS AccountDesc
      , Acc.CA_TAX_ST AS TaxStatus
      , ActionType
      , ActionTS
   FROM Source.CustomerXML C
      INNER JOIN Source.ActionXML Act
         ON C.Action_Id = Act.Action_Id
      INNER JOIN Source.AccountXML Acc
         ON C.Customer_Id = Acc.Customer_Id
      INNER JOIN dbo.DimBroker Br
         ON Acc.CA_B_ID = Br.BrokerID
      INNER JOIN dbo.DimCustomer DimC
         ON C.C_ID = DimC.CustomerID
   WHERE Act.ActionTS >= DimC.EffectiveDate
```

```
     AND Act.ActionTS <= DimC.EndDate
)

, AccountsNewAndAddAcct AS (
   SELECT *, 'ACTIVE' AS [Status]
   FROM Accounts
   WHERE ActionType IN ('NEW', 'ADDACCT')
)

, AccountsUpd AS (
   SELECT * FROM Accounts WHERE ActionType = 'UPDACCT'
)

, AccountsCloseAcct AS (
   SELECT Acc.CA_ID AS AccountID
      , Act.ActionTS AS ActionTS
   FROM Source.AccountXML Acc
      INNER JOIN Source.CustomerXML C
         ON C.Customer_Id = Acc.Customer_Id
      INNER JOIN Source.ActionXML Act
         ON C.Action_Id = Act.Action_Id
   WHERE ActionType = 'CLOSEACCT'
)

, AccountsNewAndAddAcctAndUpd AS (
   SELECT AccountID
      , SK_BrokerID
      , SK_CustomerID
      , 'ACTIVE' AS [Status]
      , AccountDesc
      , TaxStatus
      , ActionTS
      , ActionType
   FROM AccountsNewAndAddAcct
   UNION
   SELECT UpdAcct.AccountID
      , COALESCE( UpdAcct.SK_BrokerID, NewAcct.SK_BrokerID ) AS SK_BrokerID
      , COALESCE( UpdAcct.SK_CustomerID, NewAcct.SK_CustomerID ) AS
SK_CustomerID
      , NewAcct.[Status] AS [Status]
      , COALESCE ( UpdAcct.AccountDesc, NewAcct.AccountDesc ) AS AccountDesc
      , COALESCE ( UpdAcct.TaxStatus, NewAcct.TaxStatus) AS TaxStatus
      , UpdAcct.ActionTS
      , UpdAcct.ActionType
   FROM AccountsNewAndAddAcct NewAcct
      INNER JOIN AccountsUpd UpdAcct
         ON NewAcct.AccountID = UpdAcct.AccountID
)

/*, AccountsUpdCust AS (
/* When ./@ActionType is �UPDCUST�
 For each account held by the customer being updated, perform an update to:
 Set SK_CustomerID to the associated customer�s DimCustomer current record
after it has
been updated. */
      select *
   FROM Source.CustomerXML C
      INNER JOIN Source.ActionXML Act
         ON C.Action_Id = Act.Action_Id WHERE ActionType = 'UPDCUST'
)*/

, AccountsFinal AS (
   -- NEW, ADDACCT and UPDACCT
   SELECT *
   FROM AccountsNewAndAddAcctAndUpd
   UNION
   -- CLOSEACCT
   SELECT AcctNewUpd.AccountID
      , SK_BrokerID
      , SK_CustomerID
      , 'INACTIVE' AS [Status]
      , AccountDesc
      , TaxStatus
      , AcctNewUpd.ActionTS
      , 'CLOSEACCT' AS ActionType
   FROM AccountsNewAndAddAcctAndUpd AcctNewUpd
      INNER JOIN AccountsCloseAcct AcctClose
         ON AcctNewUpd.AccountID = AcctClose.AccountID
      INNER JOIN (
         SELECT AccountID, MAX( ActionTS ) AS ActionTSLatestAccount
         FROM AccountsNewAndAddAcctAndUpd
         GROUP BY AccountID
      ) LastAcct
         ON AcctNewUpd.AccountID = LastAcct.AccountID
         AND AcctNewUpd.ActionTS = LastAcct.ActionTSLatestAccount
)

INSERT INTO dbo.DimAccount
SELECT AccountID
      , SK_BrokerID
      , SK_CustomerID
      , [Status]
      , AccountDesc
      , TaxStatus
      , CASE WHEN LEAD( ActionTS ) OVER ( PARTITION BY AccountID ORDER BY
ActionTS ASC ) IS NULL THEN 1 ELSE 0 END AS IsCurrent
      , 1 AS BatchID
      , ActionTS AS EffectiveDate
      , COALESCE( LEAD( ActionTS ) OVER ( PARTITION BY AccountID ORDER BY
ActionTS ASC ), '9999-12-31 00:00:00' ) AS EndDate
   --   , ActionType
FROM AccountsFinal
ORDER BY AccountID, ActionTS ASC
GO

-- DimSecurity
WITH SecurityFinal AS (
   SELECT F.Symbol AS Symbol
      , F.IssueType AS Issue
      , ST.ST_NAME AS [Status]
      , F.[Name] AS [Name]
      , F.ExID AS ExchangeID
      , DimCo.SK_CompanyID AS SK_CompanyID
      , F.ShOut AS SharesOutstanding
      , F.FirstTradeDate AS FirstTrade
      , F.FirstTradeExchg AS FirstTradeOnExchange
      , F.Dividend AS Dividend
      -- , IsCurrent
      --, 1 as BatchID
      , CONVERT(DATETIME,STUFF(STUFF(STUFF(STUFF(REPLACE(F.PTS,'-','
'),5,0,'-'),8,0,'-'),14,0,':'),17,0,':'),120) AS EffectiveDate
      -- , EndDate
   FROM Source.FinwireSEC F
   INNER JOIN DimCompany DimCo
      ON (CASE
         WHEN ISNUMERIC(F.CoNameOrCIK) = 1 THEN CAST(DimCo.CompanyID AS
VARCHAR) --TODO: improve query because it is joining varchars
         ELSE DimCo.[Name]
      END) = F.CoNameOrCIK
   INNER JOIN StatusType ST
      ON F.[Status] = ST.ST_ID
   WHERE F.RecType = 'SEC'
   -- AND F.PTS >= EffectiveDate --YYYYMMDD-HHMMSS
   -- AND F.PTS < EndDate
      AND CONVERT(DATETIME,STUFF(STUFF(STUFF(STUFF(REPLACE(F.PTS,'-','
'),5,0,'-'),8,0,'-'),14,0,':'),17,0,':'),120) >= EffectiveDate
      AND CONVERT(DATETIME,STUFF(STUFF(STUFF(STUFF(REPLACE(F.PTS,'-','
'),5,0,'-'),8,0,'-'),14,0,':'),17,0,':'),120) < EndDate
)

INSERT INTO dbo.DimSecurity(Symbol, Issue, [Status], [Name], ExchangeID,
SK_CompanyID, SharesOutstanding, FirstTrade, FirstTradeOnExchange, Dividend,
IsCurrent, BatchID, EffectiveDate, EndDate)
SELECT Symbol
      , Issue
      , [Status]
      , [Name]
      , ExchangeID
      , SK_CompanyID
      , SharesOutstanding
      , FirstTrade
      , FirstTradeOnExchange
      , Dividend
      , CASE WHEN LEAD( EffectiveDate ) OVER ( PARTITION BY Symbol ORDER BY
EffectiveDate ASC ) IS NULL THEN 1 ELSE 0 END AS IsCurrent
      , 1 AS BatchID
      , EffectiveDate
      , COALESCE( LEAD( EffectiveDate ) OVER ( PARTITION BY Symbol ORDER BY
EffectiveDate ASC ), '9999-12-31 00:00:00' ) AS EndDate
FROM SecurityFinal
ORDER BY Symbol, EffectiveDate
GO


-- DimTrade
WITH DimTradeStaging AS (
   SELECT T.T_ID AS TradeID
      , 1 AS SK_BrokerID -- FIX SK
      , CASE
         WHEN TH.TH_ST_ID = 'SBMT' AND T.T_TT_ID IN ( 'TMB', 'TMS' ) OR
```

```
TH.TH_ST_ID = 'PNDG' THEN TH.TH_DTS
         WHEN TH.TH_ST_ID IN ( 'CMPT', 'CNCL' ) THEN NULL
      END AS SK_CreateDateID
      , CASE
         WHEN TH.TH_ST_ID = 'SBMT' AND T.T_TT_ID IN ( 'TMB', 'TMS' ) OR
TH.TH_ST_ID = 'PNDG' THEN TH.TH_DTS
         WHEN TH.TH_ST_ID IN ( 'CMPT', 'CNCL' ) THEN NULL
      END AS SK_CreateTimeID
      , CASE
         WHEN TH.TH_ST_ID = 'SBMT' AND T.T_TT_ID IN ( 'TMB', 'TMS' ) OR
TH.TH_ST_ID = 'PNDG' THEN NULL
         WHEN TH.TH_ST_ID IN ( 'CMPT', 'CNCL' ) THEN TH.TH_DTS
      END AS SK_CloseDateID
      , CASE
         WHEN TH.TH_ST_ID = 'SBMT' AND T.T_TT_ID IN ( 'TMB', 'TMS' ) OR
TH.TH_ST_ID = 'PNDG' THEN NULL
         WHEN TH.TH_ST_ID IN ( 'CMPT', 'CNCL' ) THEN TH.TH_DTS
      END AS SK_CloseTimeID
      , ST.ST_NAME AS [Status]
      , TT.TT_NAME AS DT_Type
      , T.T_IS_CASH AS CashFlag
      , 1 AS SK_SecurityID -- FIX SK
      , 1 AS SK_CompanyID  -- FIX SK
      , T.T_QTY AS Quantity
      , T.T_BID_PRICE AS BidPrice
      , 1 AS SK_CustomerID  -- FIX SK
      , 1 AS SK_AccountID -- FIX SK
      , T.T_EXEC_NAME AS ExecutedBy
      , T.T_TRADE_PRICE AS TradePrice
      , T.T_CHRG AS Fee
      , T.T_COMM AS Commission
      , T.T_TAX AS Tax
      , 1 AS BatchID
   FROM [Source].[Trade] T
      INNER JOIN [Source].[TradeHistory] TH
         ON T.T_ID = TH.TH_T_ID
      INNER JOIN [Source].[StatusType] ST
         ON T.T_ST_ID = ST.ST_ID
      INNER JOIN [Source].[TradeType] TT
         ON T.T_TT_ID = TT.TT_ID
      --INNER JOIN dbo.DimSecurity DS
      -- ON T.T_S_SYMB = DS.Symbol
         --AND ON ( TH.TH_DTS BETWEEN DS.EffectiveDate AND DS.EndDate )
)

INSERT INTO dbo.DimTrade
SELECT TradeID
   , SK_BrokerID
   , ( SELECT SK_DateID FROM dbo.DimDate WHERE DateValue = CAST(
SK_CreateDateID AS DATE ) ) AS SK_CreateDateID
   , ( SELECT SK_TimeID FROM dbo.DimTime WHERE TimeValue = CAST(
SK_CreateTimeID AS TIME ) ) AS SK_CreateTimeID
   , ( SELECT SK_DateID FROM dbo.DimDate WHERE DateValue = CAST(
SK_CloseDateID AS DATE ) ) AS SK_CloseDateID
   , ( SELECT SK_TimeID FROM dbo.DimTime WHERE TimeValue = CAST(
SK_CloseTimeID  AS TIME ) ) AS SK_CloseTimeID
   ,[Status]
   , DT_Type
   , CashFlag
   , SK_SecurityID
   , SK_CompanyID
   , Quantity
   , BidPrice
   , SK_CustomerID
   , SK_AccountID
   , ExecutedBy
   , TradePrice
   , Fee
   , Commission
   , Tax
   , BatchID
FROM DimTradeStaging

-- FactCashBalances
INSERT INTO FactCashBalances(SK_CustomerID, SK_AccountID, SK_DateID, BatchID,
Cash)
SELECT
   DA.SK_CustomerID AS SK_CustomerID,
   DA.SK_AccountID AS SK_AccountID,
   DD.SK_DateID AS SK_DateID,
   1 AS BatchID,
   SUM(CT_AMT) AS Cash
FROM Source.CashTransaction CT, DimAccount DA, DimDate DD
WHERE CT.CT_CA_ID = DA.AccountID
   AND CONVERT(DATE, CT_DTS) BETWEEN DA.EffectiveDate AND DA.EndDate
   AND CONVERT(DATE, CT_DTS) = DD.DateValue
GROUP BY
   DA.SK_CustomerID,
   DA.SK_AccountID,
   DD.SK_DateID,
   CONVERT(DATE, CT_DTS)

-- FactHoldings
INSERT INTO FactHoldings(SK_CustomerID, SK_AccountID, SK_SecurityID,
SK_CompanyID, CurrentPrice, SK_DateID, SK_TimeID, TradeID, CurrentTradeID,
CurrentHolding, BatchID)
SELECT
   DT.SK_CustomerID,
   DT.SK_AccountID,
   DT.SK_SecurityID,
   DT.SK_CompanyID,
   DT.TradePrice AS CurrentPrice
   SK_CloseDateID AS SK_DateID,
   SK_CloseTimeID AS SK_TimeID,
   HH.HH_H_T_ID AS TradeID,
   HH.HH_T_ID AS CurrentTradeID,
   HH.HH_AFTER_QTY AS CurrentHolding,
   1 AS BatchID
FROM Source.HoldingHistory HH, DimTrade DT
WHERE HH.HH_T_ID = DT.TradeID
GO
```

```
-- TODO: DImessages
-- FactMarketHistory
WITH DailyMarkets AS (
   SELECT DM1.*, MIN(DM2.DM_DATE) AS FiftyTwoWeekHighDate, MIN(DM3.DM_DATE) 
AS FiftyTwoWeekLowDate
   FROM
   (
      SELECT
         DM_DATE,
         DM_S_SYMB,
         DM_CLOSE,
         DM_HIGH,
         DM_LOW,
         DM_VOL,
         MAX(DM_HIGH) OVER(PARTITION BY DM_S_SYMB ORDER BY DM_DATE ROWS 
BETWEEN 364 PRECEDING AND CURRENT ROW) AS FiftyTwoWeekHigh,
         MIN(DM_LOW) OVER(PARTITION BY DM_S_SYMB ORDER BY DM_DATE ROWS 
BETWEEN 364 PRECEDING AND CURRENT ROW) AS FiftyTwoWeekLow
      FROM Source.DailyMarket
   ) DM1
   INNER JOIN Source.DailyMarket DM2
      ON DM2.DM_HIGH = DM1.FiftyTwoWeekHigh
      AND DM2.DM_DATE BETWEEN CONVERT(DATE, DATEADD(DAY, -364, DM1.DM_DATE)) 
AND DM1.DM_DATE
   INNER JOIN Source.DailyMarket DM3
      ON DM3.DM_LOW = DM1.FiftyTwoWeekLow
      AND DM3.DM_DATE BETWEEN CONVERT(DATE, DATEADD(DAY, -364, DM1.DM_DATE)) 
AND DM1.DM_DATE
   GROUP BY DM1.DM_DATE, DM1.DM_S_SYMB, DM1.DM_CLOSE, DM1.DM_HIGH, 
DM1.DM_LOW, DM1.DM_VOL, DM1.FiftyTwoWeekHigh, DM1.FiftyTwoWeekLow
),
FIN AS (
   SELECT
      CoNameOrCIK,
      SUM(CAST(EPS AS FLOAT)) OVER(PARTITION BY Quarter ORDER BY Year, 
Quarter ROWS BETWEEN 4 PRECEDING AND CURRENT ROW) AS EPSSum
   FROM Source.FinwireFIN
),

CompanyEarnings AS (
   SELECT DISTINCT SK_CompanyID, EPSSum
   FROM DimCompany DC, FIN
   WHERE ISNUMERIC(FIN.CoNameOrCIK) = 1
   AND DC.CompanyID = CAST(FIN.CoNameOrCIK AS INT)
   UNION
   SELECT DISTINCT SK_CompanyID, EPSSum
   FROM DimCompany DC,  FIN
   WHERE ISNUMERIC(FIN.CoNameOrCIK) = 0
   AND DC.Name = FIN.CoNameOrCIK
)
--INSERT INTO FactMarketHistory(ClosePrice, DayHigh, DayLow, Volume, 
SK_SecurityID, SK_CompanyID, SK_DateID, FiftyTwoWeekHigh, 
SK_FiftyTwoWeekHighDate, FiftyTwoWeekLow, SK_FiftyTwoWeekLowDate, PERatio, 
Yield, BatchID)
SELECT
   DM.DM_CLOSE AS ClosePrice,
   DM.DM_HIGH AS DayHigh,
   DM.DM_LOW AS DayLow,
   DM.DM_VOL AS Volume,
   --DS.SK_SecurityID,
   --DS.SK_CompanyID,
   DD1.SK_DateID AS SK_DateID,
   DM.FiftyTwoWeekHigh,
   DD2.SK_DateID AS SK_FiftyTwoWeekHighDate,
   DM.FiftyTwoWeekLow,
   DD3.SK_DateID AS SK_FiftyTwoWeekLowDate,
   DM.DM_CLOSE / CE.EPSSum AS PERatio,
   --dividend / DM_CLOSE * 100 AS Yield,
   1 AS BatchID
FROM DailyMarkets DM, /*DimSecurity DS,*/ CompanyEarnings CE, DimDate DD1,
DimDate DD2, DimDate DD3
WHERE /*DM.DM_S_SYMB = DS.Symbol
AND DM.DM_DATE BETWEEN DS.EffectiveDate AND DS.EndDate
AND DS.SK_CompanyID = CE.SK_CompanyID
AND */DM.DM_DATE = DD1.DateValue
AND DM.FiftyTwoWeekHighDate = DD2.DateValue
AND DM.FiftyTwoWeekLowDate = DD3.DateValue

GO


-- FactWatches

INSERT INTO FactWatches(SK_CustomerID, SK_SecurityID, SK_DateID_DatePlaced,
SK_DateID_DateRemoved, BatchID)
SELECT
   DC.SK_CustomerID,
   DS.SK_SecurityID,
   DD.SK_DateID AS SK_DateID_DatePlaced,
   NULL AS SK_DateID_DateRemoved,
   1 AS BatchID
FROM Source.WatchHistory WH, ( SELECT SK_CustomerID, CustomerID,
EffectiveDate, EndDate FROM DimCustomer WHERE IsCurrent = 1 ) AS DC,
DimSecurity DS, DimDate DD
WHERE WH.W_ACTION = 'ACTV'
AND WH.W_C_ID = DC.CustomerID
AND WH.W_DTS BETWEEN DC.EffectiveDate AND DC.EndDate
AND WH.W_S_SYMB = DS.Symbol
AND WH.W_DTS BETWEEN DS.EffectiveDate AND DS.EndDate
AND WH.W_DTS = DD.DateValue

UPDATE FactWatches
   SET SK_DateID_DateRemoved = DD.SK_DateID
FROM Source.WatchHistory WH, DimCustomer DC, DimSecurity DS, DimDate DD
WHERE WH.W_ACTION = 'CNCL'
AND WH.W_C_ID = DC.CustomerID
AND WH.W_DTS BETWEEN DC.EffectiveDate AND DC.EndDate
AND WH.W_S_SYMB = DS.Symbol
AND WH.W_DTS BETWEEN DS.EffectiveDate AND DS.EndDate
AND WH.W_DTS = DD.DateValue

GO
```

```
-- Industry
INSERT INTO dbo.Industry ( IN_ID, IN_NAME, IN_SC_ID )
SELECT  IN_ID, IN_NAME, IN_SC_ID
FROM    Source.Industry
GO

-- Financial
   WITH DimFinancialStaging AS (
      SELECT CAST( CONCAT( SUBSTRING( PTS, 0, 5 ), '-', SUBSTRING( PTS, 5, 2
) , '-', SUBSTRING( PTS, 7, 2 ), ' ', SUBSTRING( PTS, 10, 2 ), ':',
SUBSTRING( PTS, 12, 2 ), ':', SUBSTRING( PTS, 14, 2 ) ) AS DATETIME ) AS PTS
         , CASE WHEN ISNUMERIC( CoNameOrCIK ) = 1 THEN CAST( CoNameOrCIK AS
INT ) ELSE NULL END CIK
         , CASE WHEN ISNUMERIC( CoNameOrCIK ) = 0 THEN CoNameOrCIK ELSE NULL
END CoName
         , Year AS FI_YEAR
         , Quarter AS FI_QTR
         , QtrStartDate AS FI_QTR_START_DATE
         , Revenue AS FI_REVENUE
         , Earnings AS FI_NET_EARN
         , EPS AS FI_BASIC_EPS
         , DilutedEPS AS FI_DILUT_EPS
         , Margin AS FI_MARGIN
         , Inventory AS FI_INVENTORY
         , Assets AS FI_ASSETS
         , Liabilities AS FI_LIABILITY
         , ShOut AS FI_OUT_BASIC
         , DilutedShOut AS FI_OUT_DILUT
      FROM [Source].[FinwireFIN] F
   )

   INSERT INTO dbo.Financial
   SELECT (
         SELECT DC.SK_CompanyID
         FROM dbo.DimCompany DC
         WHERE ( DS.CIK = DC.CompanyID OR DS.CoName = DC.Name )
         --AND DC.EffectiveDate <= DS.PTS
         --AND DS.PTS < DC.EndDate
         AND DC.IsCurrent = 1
      ) AS SK_CompanyID
      , FI_YEAR
      , FI_QTR
      , FI_QTR_START_DATE
      , FI_REVENUE
      , FI_NET_EARN
      , FI_BASIC_EPS
      , FI_DILUT_EPS
      , FI_MARGIN
      , FI_INVENTORY
      , FI_ASSETS
      , FI_LIABILITY
      , FI_OUT_BASIC
      , FI_OUT_DILUT
   FROM DimFinancialStaging DS
GO

-- Prospect
```

```
   INSERT INTO dbo.Prospect
   SELECT AgencyID
      , ( SELECT SK_DateID
         FROM dbo.DimDate
         WHERE DateValue = (SELECT BatchDate FROM [Source].[BatchDate]) ) AS
SK_RecordDateID
      , ( SELECT SK_DateID
         FROM dbo.DimDate
         WHERE DateValue = (SELECT BatchDate FROM [Source].[BatchDate]) ) AS
SK_UpdateDateID
      , 1 AS BatchID
      , (
         SELECT COUNT(*)
         FROM dbo.DimCustomer DC
         WHERE UPPER( DC.FirstName ) = UPPER( P.FirstName )
            AND UPPER( DC.LastName ) = UPPER( P.LastName )
            AND UPPER( DC.AddressLine1 ) = UPPER( P.AddressLine1 )
            AND UPPER( DC.AddressLine2 ) = UPPER( P.AddressLine2 )
            AND UPPER( DC.PostalCode ) = UPPER( P.PostalCode )
            AND DC.Status = 'ACTIVE'
      ) AS IsCustomer
      , LastName
      , FirstName
      , MiddleInitial
      , Gender
      , AddressLine1
      , AddressLine2
      , PostalCode
      , City
      , State
      , Country
      , Phone
      , Income
      , numberCars
      , numberChildren
      , MaritalStatus
      , Age
      , CreditRating
      , OwnOrRentFlag
      , Employer
      , numberCreditCards
      , NetWorth
      , CASE
         WHEN NetWorth > 1000000 OR Income > 200000 THEN 'HighValue'
         WHEN NumberChildren > 3 OR NumberCreditCards > 5 THEN 'Expenses'
         WHEN Age > 45 THEN 'Boomer'
         WHEN Income < 50000 OR CreditRating < 600 OR NetWorth < 100000 THEN
'MoneyAlert'
         WHEN NumberCars > 3 OR NumberCreditCards > 7 THEN 'Spender'
         WHEN Age < 25 AND NetWorth > 1000000 THEN 'Inherited'
      END AS MarketingNameplate
   FROM [Source].[Prospect] P
GO
```