ULB UNIVERSITÉ LIBRE DE BRUXELLES

INFO-H-419 : Data Warehouses

TPC-DI Benchmark on Microsoft's SQL Server Database

Arina Gepalova

Marie Giot

You Xu

Tianheng Zhou

Professor

Esteban Zimányi

2022

Table des matières

1 Introduction

1.1 TPC-DI

TPC-DI is a benchmark for data integration. It combines and transforms data extracted from different sources, and loads it to a data warehouse. TPC-DI uses data integration of a factious Retail Brokerage Firm as a model.

1.2 SQL Server

SQL Server - is a Microsoft relational database management system. It supports standard SQL, and comes with Microsoft proprietary language - Transact-SQL.

1.3 SSIS

There are multiple components and services of SQL Server. One of them - SQL Server Integration Service - provides Extract-Transform and Load capabilities of the different types of data from one source to another.

2 Implementation

2.1 Scale Factors

Similar to the TPC-DS test, the benchmark is available and valid in running and testing based on various original data sizes, also called “scale factor.” Our benchmark process applied the data in a total of 4 exponentially incrementing scale factors from DIGen data generation. The data records and original data sizes are listed below for “Batch1,” which is the primary data source this project utilizes for the “historical load” phase of the benchmark test.

— SF 3 : 4539962 records / 293.6 MB

— SF 6 : 9443149 records / 577.7 MB

— SF 12 : 19242651 records / 1.19 GB

— SF 24 : 38842886 records / 2.41 GB

2.2 Preparation

2.2.1 Data Generation

The benchmark source data across four different scale factors are generated from the suggested “DIGen” data generation to compile the TPC-DI Benchmark rules. According to the official TPC-DI documentation, “DIGen is the data generator program the TPC provides for creating Source Data and audit information.” Therefore, this project applies DIGen version 1.10 from the re-compiled java program with no essential functionality modification on the local environment to ensure the compatibility of the generator on our benchmark local machine. The re-compiled program is available on GitHub for reference. All source data are created and stored initially on the local benchmark machine’s storage disk in text file format and copied into the “Staging Area” later, as specified below. This step is not part of the timed benchmark process.

2.2.2 Staging Area Creation

Before loading the data into the staging area, our tested data platform - SQL Server, requires a set of predefined table schemas to store the source data in the corresponding database. Since the data loading process should ensure the consistency and integrity of the original data format, the designated table schemas and table creation SQL command strictly follow the direction of Clause 2 of the documentation. In the staging area database, each table corresponds to each file described in Clause 2, except for FINWIRE data. For FINWIRE data, since it contains three types of combined information that have no column correspondence with each other, there are, in total, three staging area tables created for storing them instead of a single table (FINWIRE_CMP, FINWIRE_SEC, and FINWIRE_FIN). To separate the three tables, a python script containing parsing methods was used and this script can be found on the project's Github (the src folder). Each column in the table is represented on a field in the document. Furthermore, the data type of each column is assigned with the closest correspondence in the SQL Server regarding the "Type" in Clause 2. The SQL command for staging area creation is available in the GitHub repository (the sql folder). This step is not part of the timed benchmark process.

2.2.3 Data Warehouse Creation

Similar to the "Staging Area," the "Data Warehouse" database in the SQL Server also requires a set of predefined table schemas. Therefore, before the ETL process, this project follows the description in Clause 4 to create the empty tables that are ready for the loading process after the transformation process as part of the whole timed benchmark process. This is achieved by executing a set of SQL "create table" commands. In the data warehouse database, each table corresponds to each table described in Clause 3. Each column in the table is represented on a field in the document. And the data type of each column is assigned with the closest correspondence in the SQL Server in reference to the "Type" in Clause 3. The SQL command for data warehouse creation is available in the GitHub repository (the sql folder). This step is not part of the timed benchmark process.

For the data warehouse, we make some small changes in some tables to suit the historical load process, these can be found in section 2.3.1.

2.2.4 Data Integration Preparation

In this project, SSIS handles all the data transformation processes. To build an SSIS project, it is required to install an SSIS service extension to SQL Server and the corresponding Data Integration

extension to Visual Studio. The data integration project is only executable with the mentioned environment above in Visual Studio. This step is not part of the timed benchmark process.

2.2.5 Staging Area Loading

This project loads the source file into the staging area database in the SQL Server using SSIS with a Visual Studio project. For each table, this project applies a “data flow task.” Inside the data flow task, for most table loading tasks, there are three subtasks applied : “Flat File Source,” “Data Transformation,” and “OLE DB Destination.” A workflow sample is shown in Figure 2.1.

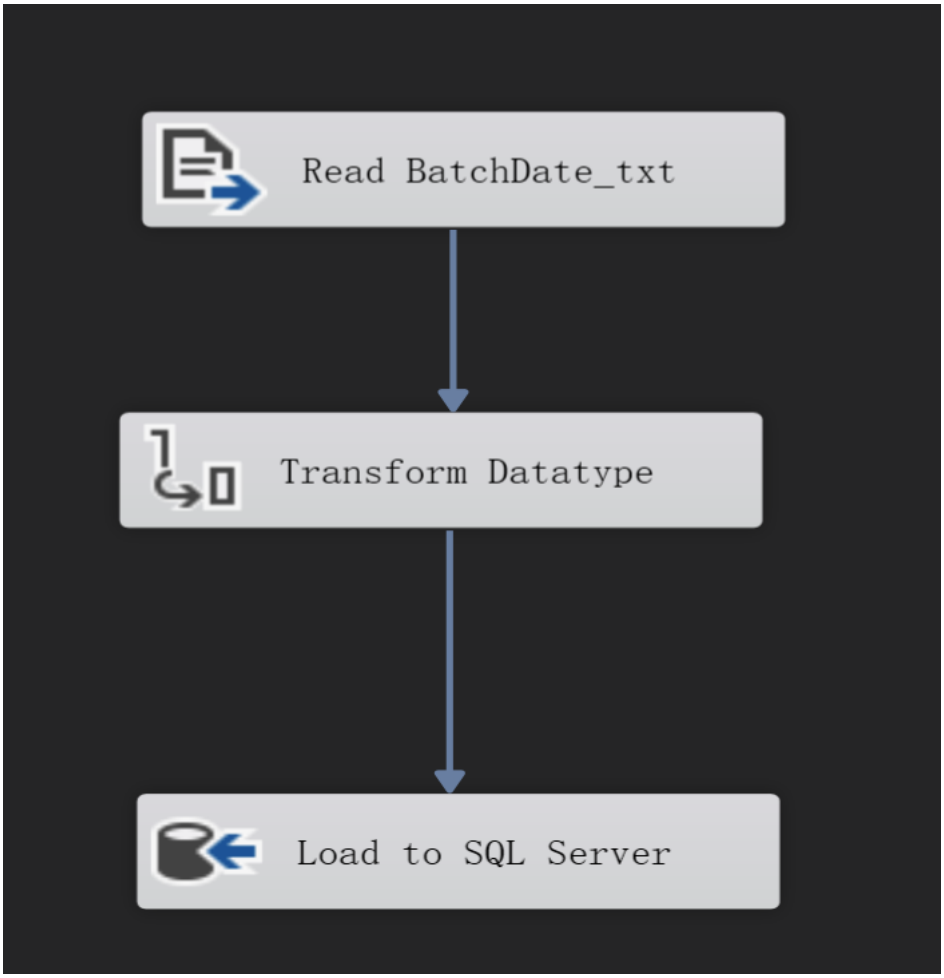

FIGURE 2.1 – Sample SSIS Data Flow of Copying Source File into Staging Area

During the “Flat File Source” task, SSIS will connect to a single source file in the local storage, either in text or CSV format. By specifying the delimiter and column names, it reads all the records in the file as a relational table. Then, during the “Data Transformation” task, SSIS transforms the data type of each column into the correct and SQL Server-supported datatype that corresponds to the staging area database. Finally, by completing the “OLE DB Destination” task, SSIS connects to the local database server and loads the selected columns into the corresponding table in the

staging area table. By completing these three subtasks, SSIS successfully copies the data from the source file into the staging area that is ready for the timed benchmark process later regarding transformation and loading into the data warehouse.

For table "CustomerMgmt", we first convert the "CustomerMgmt.xml" file into a CSV format using an XML to CSV converter. Then we will do the same thing as other flat files.

For tables “Time” and “Date,” to match the string values “true” and “false” in the original source data file to the boolean data column in the staging area, we apply the “Derive Column” task in the SSIS to ensure the datatype integrate, where “true” becomes 1 and “false” becomes 0.

We execute the “data flow task” for each table and file only once and one after another. This step is not part of the timed benchmark process.

The SQL Server Integration Service Project for this subsection is in the SSIS Project/Load Data to Stage DB folder on Github Repo.

2.3 Historical Load

For the Historical Load phase, we take advantage of SSIS to transform and load the data from the Staging area to the data warehouse. The SSIS data flow for loading each of the tables, either static, dimensional, or fact, in the data warehouse contains three sub-tasks : “OLE DB Source,” “Data Transformation,” and “OLE DB Destination.” A workflow sample is shown in Figure 2.2.

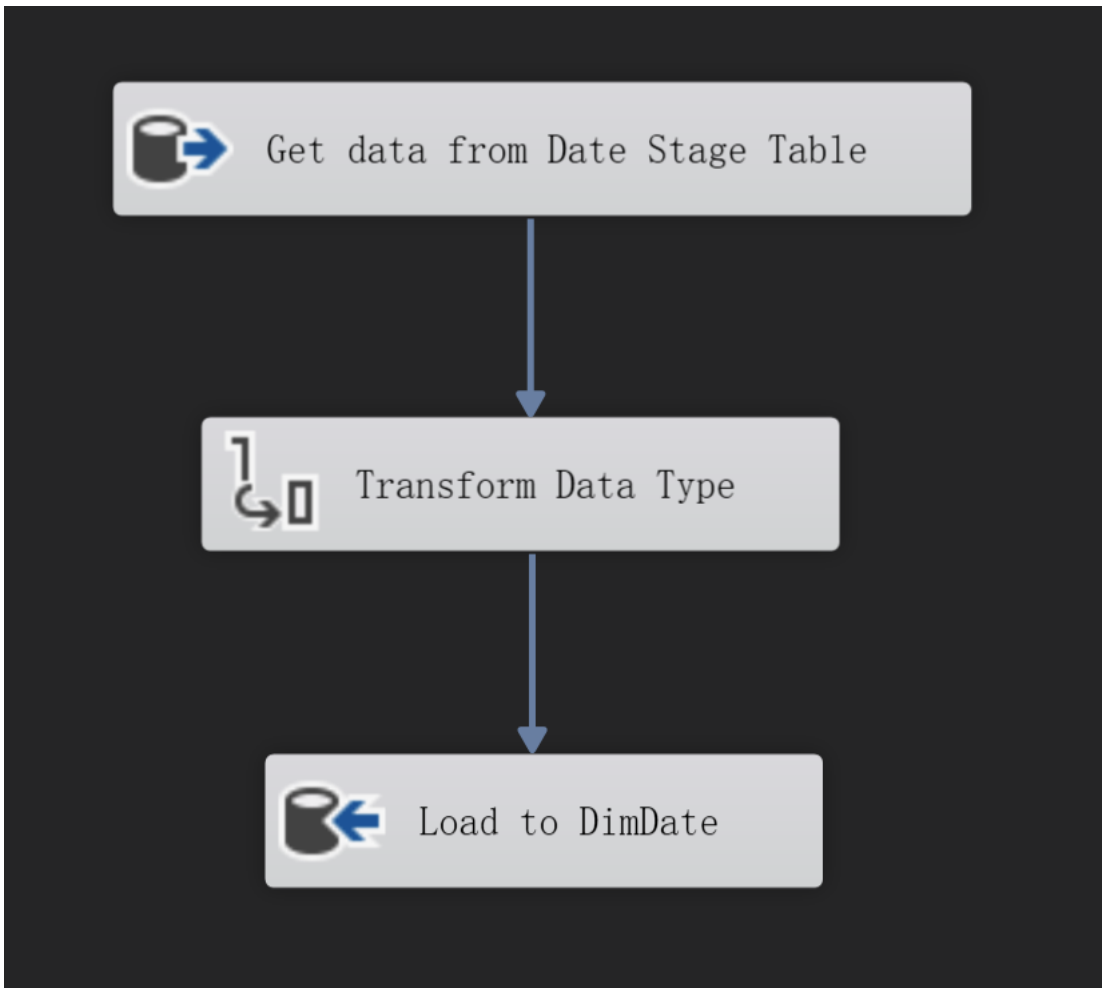

FIGURE 2.2 – Sample SSIS Data Flow of transforming and loading data from Staging Area to the Data Warehouse

In “OLE DB Source,” SSIS retrieves the transformed data by executing a “SELECT” clause SQL command on the staging area database. A brief introduction of each transformation SQL command is specified below. After this, a “Data Transformation” task helps to transform some of the data types of the columns to match the destination data warehouse table schema. Finally, the “OLE DB Destination” task will load the columns into the correct data warehouse table by specifying the columns.

It was necessary to follow a certain order to transform the tables.

1. The following tables are in the 1st batch :
   - DimTime
   - DimDate
   - DimBroker
   - DimCompany
   - DimCustomer

2. After these tables :
    - DimAccount
    - DimSecurity
3. DimTrade
4. The last tables in turn are :
    - FactCashBalance
    - FactHoldings
    - FactMarketHistory
    - FactWatches
    - Financial
    - Industry
    - Prospect
    - StatusType
    - TaxRate
    - TradeType
    - DIMessage
    - Audit

Figure 2.3 shows our workflow for executing the historical load in SSIS. The SSIS Project for historical load is in SSIS Project/Historical Load folder on Github Repo.

For neatness of this report, we won't talk about those SQL commands for historical load transformation. Those SQL files are all in sql/transformation folder on our Github Repo, and explanation to them is in the TPC-DI Specification.

FIGURE 2.3 – The Sequence of Control Flow in SSIS, Each Component in the Sequence is a Data Flow

2.3.1 Encountered Problems

Some inconsistencies have been met in the TPC-DI instructions. For example, in the table DimTrade, the columns SK_CloseDateID,SK_CloseTimeID, SK_CloseDateID and SK_CloseTimeID have a non-null constraint but in the instructions at point 4.5.8.2 of the pdf, it says that in some cases, those columns must take null values. In this case, it seemed logical to solve the problem by removing the non-null constraint because it made no sense to invent a new value to replace null.

In other cases, we had to change some columns type. For example, in DimTrade, the columns SK_CloseTimeID and SK_CreateTimeID were supposed to be times as written in section 4.5.8.2 of the TPC-DI instructions but were orginaly set as bigint. So we ended up changing the column type.

The other changes that has to be executed are mentioned in the sql/change_in_datawarehouse.sql on the GitHub with a "change" mention next to the lines that are concerned.

3 Results

The transformation and Loading Process run on the device with the following settings :

— **Computer** : HP ENVY x360 Convertible 15-bp1xx

— **Operating System** : Windows 10

— **RAM** : 20GB in total

— **CPU** : Eight Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz

— **Storage** : 500GB in total

— **Software** : SQL Server 15.0.2000.5, SQL Server Management Studio 15.0.18424.0, Visual Studio 2019 with SQL Server Integration Service 15.0.

For the "FactMarketHistory" table, it requires too many resources to completely transform and load the data from OLEDB source (even scale factor3 takes more than 3 hours), so we gave up running it in the middle. All the times in Table 3.1 are recorded as an average of three runs.

For the "Audit" table since we only did the Stage Creation, Data Warehouse Creation, Stage DB Loading, and Historical Load, we did not implement it.

Figure 3.1 and Figure 3.2 shows the heatmap of Historical Load time of different tables (scaled by log10 for a better color bar scale) and line plot of aggregated running time for different scale factors, respectively.

Time (seconds)	Scale Factor 3	Scale Factor 6	Scale Factor 12	Scale Factor 24
DimTime	0.922	1.547	1.435	4.687
DimDate	0.750	1.031	1.156	1.562
DimBroker	0.625	0.968	0.984	1.063
DimCompany	0.734	1.156	1.078	1.110
DimCustomer	2.812	8.312	18.297	30.282
DimAccount	1.187	3.031	5.171	6.438
DimSecurity	0.672	0.985	4.844	0.891
DimTrade	15.860	27.468	47.062	156.047
FactCashBalance	4.500	6.312	13.047	16.813
Financial	39.938	201.922	855.453	3828.141
Industry	0.672	0.984	0.937	0.688
Prospect	1.187	1.718	3.172	3.234
StatusType	0.797	0.891	0.906	0.703
TaxRate	0.954	0.844	0.954	0.687
TradeType	0.844	0.828	0.875	0.687
DImessages	1.672	1.937	2.547	4.359
FactHoldings	7.422	12.688	26.109	100.312
FactWatches	4.516	5.938	7.141	23.796
FactMarketHistory	> 10800 (3h)	-	-	-
Audit	Not Executed	Not Executed	Not Executed	Not Executed

TABLE 3.1 – The Time for Transformation and Loading

	Scale Factor 3	Scale Factor 6	Scale Factor 12	Scale Factor 24
Time (seconds)	86.064	278.56	991.168	4181.5

TABLE 3.2 – The total Historical Load Time for Different Scale Factors

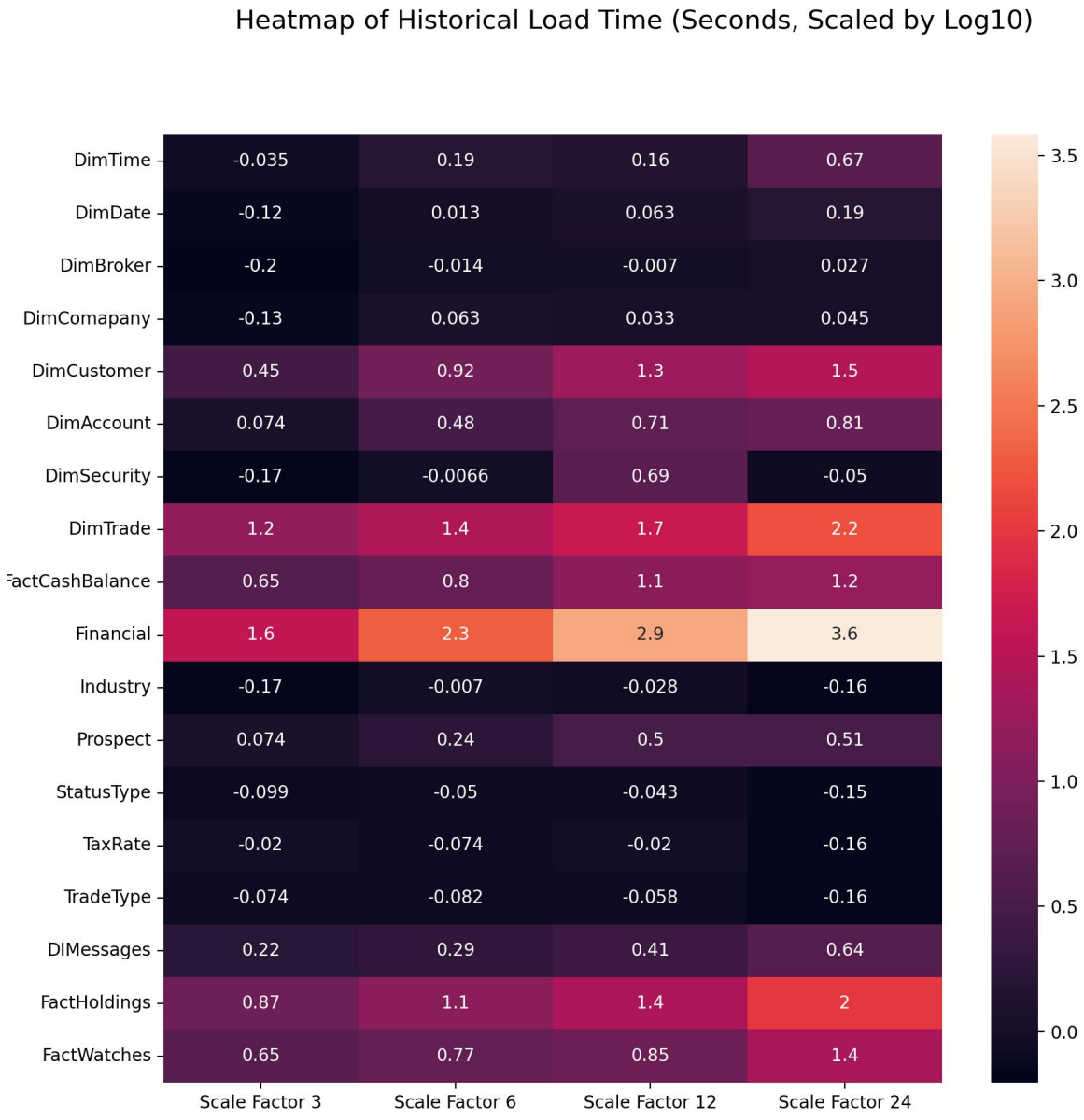

FIGURE 3.1 – The Heatmap of Table 3.1 - Historical Load Time (Scaled by Log10)

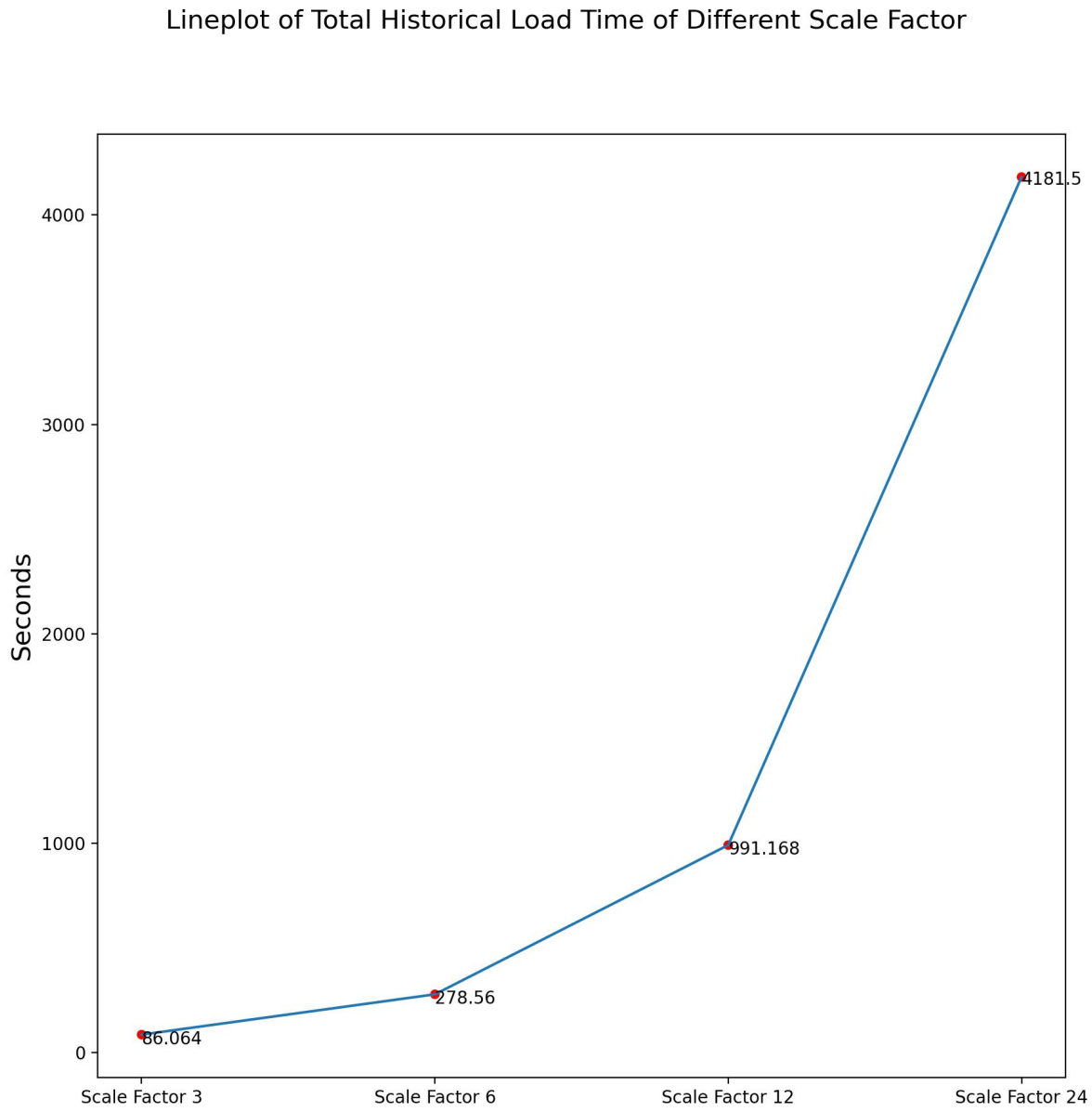

FIGURE 3.2 – The Total Historical Load Time for Different Scale Factors

4 Discussion and Limitation

From Figure 3.2, we can see that when the initial data doubles, the total historical loading time does not double, but grows with an exponential tendency. This is mainly due to our transformation operations that don't have a linear time complexity, like the joins for example.

Figure 3.1 visualizes Table 3.1 in an intuitive format. We can see that the Historical Load time of some tables in the data warehouse does not change much as the scale factor varies, such as Industry, StatusType, TaxRate, TradeType, etc. This is basically because those tables are static, as required by TPC-DI specification, which means they do not change when we load them from the stage area. As a result, the Historical Load time for them won't change much.

This is also interesting to note that some execution times even decrease with the scale factor, even if they are not static tables. For example, it took more time to complete DimCompany's script with the scale factor 6 than with the scale factors 12 and 24. This can probably be explained by the fact that this sql script doesn't contain many joins and those joins are only on relatively small tables compared to other sql transformation scripts.

While for some other tables, the Historical Load time varies as the scale factor changes, like Financial, DimTrade, DimCustomer, FactWatches, and FactHoldings. Their time also grows non-linearly as the transformation operation for them is not linear (like the union operation or the join operation).

For limitations, historical load to some tables takes too much time to run, like the FactMarketHistory. We also find some inconsistencies in our operations and TPC-DI specification, as illustrated in section 2.3.1. To improve the speed of the Historical Load, we can use indexes for the recurrent join statements. We also find that some tables whose data are never modified and only accept new addition of data, are used very often, like DimDate. We can use materialized views for those frequently used tables to reduce resource consumption. Also, Distributed or Parallel Processing may help speed up the process to a certain extent.

5 Conclusion

In this report, we introduce how to replicate TPC-DI Benchmark using Microsoft SQL Server and SQL Server Integration Service (SSIS). We briefly introduce them and the background first; Then we talk about how we implement the benchmark with different data-generating scale factors. First we create the Database for staging area and data warehouse, following the instruction in TPC-DI Specification. Then we prepare the SSIS and load multiple-format data into the staging area, and finally introduce the workflow of the Historical Load with visualized results of running times and discussions. We can conclude that the time taken to complete the transformation's operations of the ETL process increases in average exponentially with the scale factor of the data used.

6 Appendix

6.1 Appendix I : Replication Instructions

All SQL commands and SSIS Projects to replicate this project are in this open-sourced GitHub repository.

`https://github.com/xuyou1999/TPC-DI_DW_F2022`

Articles

[14] *TPC Benchmark™ DI - Standard Specification, Version 1.1.0* (nov. 2014).