Université Libre de Bruxelles

INFO-H-419: Data Warehouses

Project Assignment

Part II: Implement the TPC-DI benchmark

Authors:
Gonçalo Moreira
Nazrin Najafzade
Rémy Detobel
Shafagh Kashef
Group: 4

Supervisor:
Prof. Esteban Zimányi

December 20, 2018

Contents

1 Introduction

Analyze the performance of a software is not easy. This is even more when it concern big task like ETL. Usually we use benchmark to compare the performance of different programs. This project will focus on TPC-DI benchmark and its application on "SQL Server Integration Services" abbreviated by "SSIS". There exist different tools to make ETL like Pentaho Data Integration, Talend Data Studio, SQL scrpts, ect.
This report describe the implementation of TPC-DI for SSIS.

2 TPC

TPC is an corporation which proposes different benchmarks. The aim is to use the same basis of comparison. There exist different benchmarks: TPC-DS (Decision Support Benchmark, which models the decision support functions of a retail product supplier), TPC-DI (Data Integration Support Benchmark, which models a typical ETL process that loads a data warehouse).

In the first part of this project, we worked with the TPC-DS and now, for the second part, we will focus on TPC-DI which is a benchmark for data integration to test ETL.

This benchmark tool is very useful for hardware and software vendors as it enables them to showcase the performance, price-performance, and efficiency of their systems in a competitive, fair and open way.

3 Objectives

The main goal of this project is to implement the TPC-DI benchmark which combines and transforms data extracted from an On-Line Transaction Processing (OTLP) system along with other sources of data, and loads it into a data warehouse.
Another goal is to learn and understand the process to make a full ETL implementation. Unfortunately, we quickly noticed that implement all the process take a lot of time. We have therefore decide to focus on Historical Load (see point 4).

4 Implementation

This point describe the different step to create an ETL process for TPC-DI benchmark. Note that the code of this implementation is published on Github[1].

Generally when a company set up a data warehouse database they have two steps: fill the database with existing information and then use ETL process to incrementally add information

[1] `https://github.com/detobel36/tpc-di`

to the destination database. Obviously TPC-DI is based on this structure. So there are two phases:

1. Historical load
2. Incremental updates

Like already mention, this document will only focus on historical load.

4.1 Initialize

Before the implementation of the ETL process it is important to prepare the environment. That is, source data but also destination tables.

The TPC-DI tool allows to generate the data source. These files are divided in different folders:

- *Batch1* contains all files used for the Historical Load.
- *Batch2* and *Batch3* contain all files used for Incremental Update.

The TPC-DI tool allow the generate these file with different scale factor (more information in the section result 5).
The generation of the destination database is based on the TPC-DI documentation. Indeed, this documentation explain the different operation that must be done on the data to fill the destination database. The structure and the type of columns are given. SSIS is a microsoft product and is thus fully compatible with Microsoft SQL Server. That is why the destination database was create on this type of database.

4.2 Historical Load

The goal of the historical load is to fill the table with all past information. Indeed the destination database must contain all information. So the first step is to save the evolution of all information and to save which information is currently correct.
All the historical information that have been generated by the TPC-DI Tool are located in the folder *Batch1*.

Here are the tables (associate with the main source file) that will be filled at the end of this step:

- StatusType (StatusType.txt)
- Industry (Industry.txt)
- DimBroker (HR.csv)
- DimTime (Time.txt),
- DimDate (Date.txt),

- TaxRate (TaxRate.txt),
- TradeType (TradeType.txt),
- DimAccount (CustomerMgmt.xml)
- DimCompany and Financial and DimSecurity (FINWIRE)
- DimCustomer (CustomerMgmt.xml)
- DimTrade and DImessages (Trade.txt, TradeHistory.txt),
- FactCashBalances (CashTransaction.txt),
- FactHoldings (HoldingHistory.txt),
- FactMarketHistory (DailyMarket.txt),
- FactWatches (WatchHistory.txt),
- Prospect (Prospect.csv)

IsCurrent and **EndDate** columns are for history tracking of dimension tables. They enable us to distinguish between the current record and the duration of validity of each previous record.

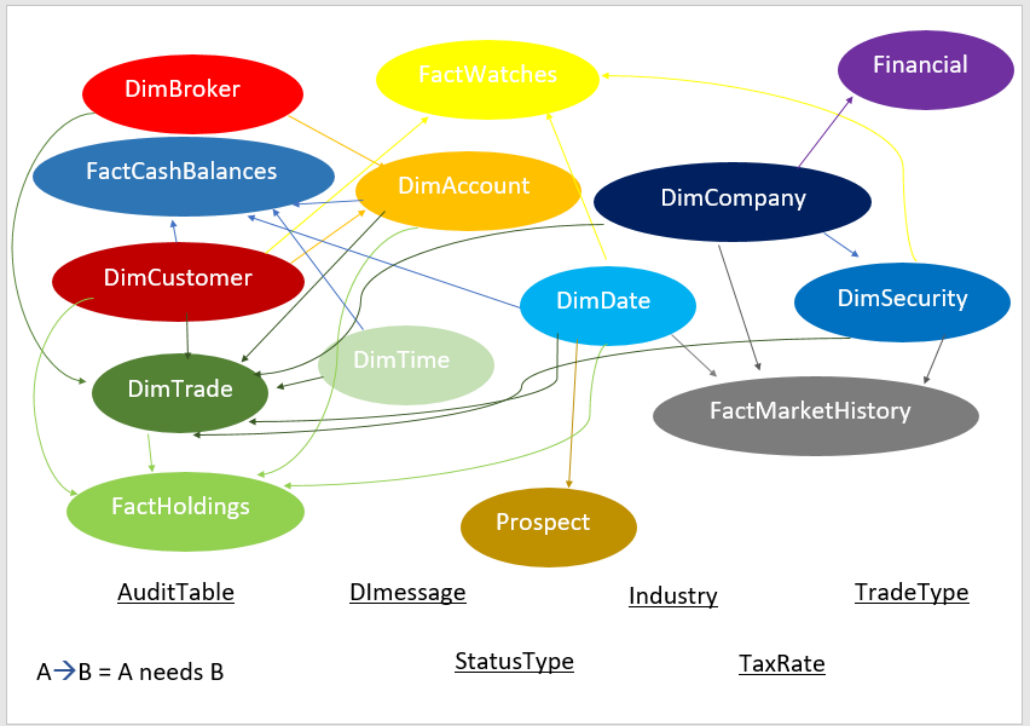

Figure 1: DI Operational Tables Relations

The TPC-DI documentation details the different step to create these tables. But the implementation is not so easy. For instant you need to implement *Reference Tables* first. For the other tables you need to check the dependencies (see figure 1). You will find below, an explanation of the implementation of all tables:

DimTime, DimDate, StatusType, TaxRate, Industry, TradeType

All the following tables are fixed. So the ETL process is very simple. Read the data from the file (respectively *Time.txt*, *Date.txt*, *StatusType.txt*, *TaxRate.txt*, *Industry.txt* and *TradeType.txt*), convert type if necessary and store the result in the destination database.

DimBroker

The table DimBroker contain some information of the "HR.csv" file. Indeed, only entries which have "EmployeeJobCode" set to "314" are saved. Notice that DimBroker table is an historical table, so there are column to know the time validity. But the source file doesn't have any date information. The effectiveDate is defined on the earlier date in the DimDate table.

DimCustomer

DimCustomer is one of the Dimension Tables and it is used to describe the business entity - Customer.

The *CustomerMgmt.xml* file represents data extracted from a Customer Management System (CMS). The CMS handles new and updated customer and account information. The data in the file is transactional in nature and uses a hierarchical structure to represent data relationships. This source provides the data for the DimAccount and DimCustomer tables in the Historical Load phase.

In *CustomerMgmt.xml* we have many element properties like Action Type, Customer identifier, Customer name, Address, Contact Info and Tax Info.

The structure of the XML document is described by an XML Schema (xsd file). To parse and validate the XML document, we used the formal schema given by the paper (reference) - *CustomerMgmt.xsd*. Note that we had some issues formatting the phone number.

DimAccount

DimAccount is based on *CustomerMgmt.xml* which is describe in DimCustomer. But DimAccount need also some information from other tables. Indeed, DimAccount need to have a link with the related broker. Thus the table need to have the related surrogate key. Same for the Customer.
Note that here the project focus only on the new account entry.

DimCompany, Financial and DimSecurity

These three tables are based on FINWIRE files. There exist one FINWIRE per quarter. So there are 4 files per year. If you have 51 years, you have 204 files. The name of FINWIRE files have the following structure: $FINWIRE\{year\}Q\{quarter\}$. Where quarter is just a number. A loop is necessary to process all the files.

All the row in FINWIRE begin with a date and a type: "FIN", "SEC" or "CMP". This is the only information that all lines have in common. Based on the type, it is possible to split (with "fixed" length) other column.

DimCompany is thus based on "CMP" type. To fill the table some link need to be done with StatusType (table) and the file *Industry.txt*.

DimSecurity is based on "SEC" type and need also some link. With StatusType table (like DimCompany) but also with DimCompany table.

Finally, **Financial** is process with "FIN" type and like DimSecurity, it make a link with DimCompany.

Note that DimCompany is a historical table. This means that we need to filter the join entry to be sure that the key was valid when the new record have been created (also based on his historical information).
Note also that normally DimSecurity and Financial must be update (during the Incremental part) but considering that the number of update on these tables is small, we can ignore these changes.

DimTrade

DimTrade is based on *TradeHistory.txt* and *Trade.txt* but some informations need to be catch on other tables.
For the time for instance, DimTrade use directly DimDate and DimTime id. Each row have also directly the status name (based on StatusType table) and not only the key. Same for trade type (based on TradeType table). The process fetch also the id of account (based on DimAccount) and security (based on DimSecurity).
This process is one of the biggest of the project. Indeed, they have a lot of table referencies.

FactCashBalances

FactCashBalances data is obtained from the data file *CashTransaction.txt*. This process is more complicated than the other. Indeed, we need to take into account the order of the transaction and make a cumulative sum. In SSIS this require to use script.
After this part, we need to make some link with DimAccount and DimDate table, to have the key. Note that, as usual, we need to check the date validity for historical tables (here DimAccount).

FactHoldings

Data for FactHoldings comes from the HoldingHistory.txt file and the DimTrade table. The quantity and price values reflect the holdings for a particular security after the most recent trade. The customer can have a positive or negative position (Quantity) as a result of a trade.

FactMarketHistory

FactMarketHistory is based on *DailyMarket.txt*. A difficult point of this table is to group the entry by date and security while at the same time keep the date of the maximum and the minimum for these two key.
To do that in SSIS, one solution is to split the flow in four: one for maximum, second for maximum and date, third for minimum and the last for minimum and date. We made a join to keep the relevant information.
The calculate of PERation and Yield is not so easy. A link, following by an aggregation must be done with the table "Financial".
The other part of the process is easy: link with DimSecurity and DimDate

FactWatches

Data for FactWatches comes from the WatchHistory.txt file. Surrogate keys must be obtained for the Customer, Security and Date dimension references. The date keys show the dates the watch was set and removed.

Prospect

Prospect table is based on *Prospect.csv* file which contains directly a lot of information that can be directly mapped to the destination table. The only thing to do is to check that a customer exist (or not) and make a link with DimDate.

4.3 Not implemented section

Some tables doesn't have sens to implement for this project. Indeed, here we focus only on historical part. The following table have not been implemented: DImessage, AuditTable, dimtradeForexperiment.

5 Results

Data Set Scalling is very pertinent for benchmark tools because of two reasons.

Firstly, for a benchmark to be relevant to real world problems, it needs to reflect data sizes used in real world systems.

Secondly, systems and data sets tend to grow rapidly over time. A benchmark with a static data set size will become obsolete within a few years due to the compute power used by real world applications. Hence, a benchmark needs to be able to adapt to different data sizes.

5.1 Configuration

- **Data tool:** Microsoft SQL Server Data Tools for Visual Studio 2017 (SSDT) Version 15.9.2
- **Database Management System:** Microsoft SQL Database.
- **Server:** localhost (use hostname .)
- **Database:** destination-di
- **Shared folder:** C:/Users/Public/Documents/TPC-DI/Batch1. All the files are store at the same place (which not user dependant). Indeed, we choose a "public" folder.

6 Conclusion

In this project tables of the TPC-DI historical load are created, which aims to describe the process of extracting and combining data from a variety of data source formats, transforming that data into a unified data model representation and loading it into a data store.

References

[1] Alejandro Vaisman Esteban Zimányi. Data warehouse systems: Design and implementation. In *Data Warehouse Systems: Design and Implementation*, pages 1–639. Springer Berlin Heidelberg, 2014, 2014.

[2] TPC. Tpc-di is a benchmark for data integration, 2001-2018. `http://www.tpc.org/tpcds`.

[3] bahadley. Tpc-di (transaction processing performance council - data integration, 2017. `https://github.com/bahadley/tpc-di`.

[4] AKartashoff. Tpc-di, 2018. `https://github.com/AKartashoff/TPCDI-PDI`.

[5] Esteban Zimanyi. Cours: Info-h-419: Data warehouses, 2018. `https://cs.ulb.ac.be/public/teaching/infoh419`.

[6] Meikel Poess, Tilmann Rabl, Hans-Arno Jacobsen, and Brian Caufield. Tpc-di: The first industry benchmark for data integration. *PVLDB*, 7(13):1367–1378, 2014.