UNIVERSITÉ LIBRE DE BRUXELLES

ÉCOLE POLYTECHNIQUE DE BRUXELLES

A TCP-DI benchmark implementation using Oracle

INFOH419 - DATA WAREHOUSES

Fall 2022

Authors:
Ivanović, Nikola,
Lorencio Abril, Jose Antonio,
Yusupov, Sayyor,
Živković, Bogdana

Professor: **Zimányi, Esteban**

Contents

List of Figures

Listings

Abstract

In this project we have reproduced the TPC-DI Benchmark specification to execute in an Oracle environment. More precisely, we have used Oracle XE, the free version of Oracle SQL systems.

The present report starts reviewing the ETL process and a summary of the specification of TPC-DI is provided in Chapter 1. In Chapter 2 the implementations details on how we have taken the specification from theory to practice are explained, with emphasis in the different tests performed in the benchmark, whose results are analyzed in Chapter 3. In this chapter we also compare our results with those obtained in [AVZ20].

All the explanations provided in this report refer to the github repository created for the project, which can be accessed with the following link https://github.com/Lorenc1o/TPC-DI_Oracle.

Chapter 1

Data Integration

In this chapter, we are going to introduce the key concept for which the TPC-DI benchmark is designed: the ETL process. We also present a summary of the TPC-DI specification, trying to represent its main characteristics, phases and objectives.

1.1 The ETL process

Data Integration or, equivalently, the **Extraction, Transformation and Load (ETL) process** refer[1] to the process of extracting data from internal and external sources of an organization, transforming these data, and loading them into a data warehouse.

The ETL process is critical in the data warehouse scheme, and in all the steps of its life cycle: since its design to its maintenance, updating and even redesign to cope with new restrictions and data. The DI should aim to define how all data from different sources are combined into a single one, solving possible conflicts of meaning and shape of the data, as well as minimizing the errors entering the data warehouse.

As can be inferred, the design of the ETL processes must be done before the creation of the data warehouse, because these processes define the scope of the data warehouse and its limits, as they are defining the precise form of the data that will be stored inside it. In addition, one needs to think not only in the population of the data warehouse for the first time, but also how it will be updated when new data is available. This is an important observation, because even if we are able to define the ETL processes in such a way that we handle all possible errors and divergent cases that can arise today, the next month a new type of error that did not exist before can appear, and our ETL processes would need to be consequently modified to be able to tackle it.

1.2 TPC-DI Benchmark

The TPC-DI benchmark[2] is a performance test of tools that perform the DI process of a system. The benchmark **workload** manipulates a defined volume of data, preparing it for use in a Data Warehouse. The **model** includes data representing an extract from an OLTP system, which needs to transformed along with data from different sources and loaded into a Data Warehouse.

[1] Definition from [VZ22].

[2] For a completely detailed explanation refer to [TPC14].

This way, the benchmark tests the behavior of system components associated to the DI process. These components are characterized by:

- The manipulation and loading of large volumes of data.
- A mixture of transformation applied to the data.
- Historical loading, as well as incremental updates.
- Consistency requirements.
- Multiple data sources with different formats.
- Multiple data tables with varied data types, attributes and relationships.

On the other hand, the **operations** are modeled as follows:

- **Source data** is automatically generated in flat files.
- Transformation of the data begins with the **System Under Test (SUT)** reading the Source data.
- The **transformations** validate the Source data and properly structure the data for loading into a DWH.
- The process **concludes** when all Source data has been transformed and is available at the DWH.

The **performance metric** of the benchmark is a throughput measure, the **Number of Source data rows processed per second**.

1.2.1 Business and Application environment

The data represents a sample data retrieved from retail brokerage. Five types of data sources are required for the benchmark.

First, there are multiple tables in the OLTP system. They contain information about brokers, trade deals, customers etc. For each table, there are Changed Data Capture extracts that list the changes made to the table. They are used in the Incremental Updates Phase. On the other hand, the full dump of the original tables are made for Historical Load phase.

The HR database contains one table with an information on employees. It is used in the Historical Load and does not contain CDC files for Incremental Updates.

The Prospects file contains information on potential and existing customers with information on their names, demographic data and addresses. The DI process should determine the changes that were made to this file. It is updated daily with a new extract.

In addition, Historical Load Phase uses two other external sources of information. Data on companies and securities comes from Financial Newswire Service (FINWIRE). The file contains three types of records that will be used to fill the tables DimCompany, DimSecurity and Financial during the Historical Load. Also, data with information on new and updated customers and accounts was retrieved from Customer Management System. This data is in the form of .xml file. It is used to fill the tables DimAccount and DimCustomer during Historical Load.

Finally, a collection of reference data is used once during Historical Load.

1.2.2 Summary of operations

It is usually the case that the DI process has two variants. One of them is performing of a historical load, when the DWH is initially created or needs to be refilled with historical data (e.g. a restructuring of the schema). The other variant is adding newly generated data to the data warehouse.

Phases of operation

Preparation phase

Some initial operations are required but their execution times are not used for the benchmark. These include:

- Generating the data and putting it to the Staging Area. The data is generated with an automated generator provided by the Benchmark. The size of the data (scale factor) needs to be specified and as a result, a folder is created, containing three directories *Batch[k]*, where $k = 1, 2, 3$.
- Creating the databases and the tables of the Data Warehouse.

Historical Load phase

Historical Load is done either when the Data Warehouse is initially created or when the Data Warehouse is regenerated using historical records. Transformation rules specific to Historical Load are specified in the benchmark.

This phase uses files from *Batch1* directory of the generated data.

Incremental Update phase

Incremental Updates also require specific transformations. For example, while for Historical Load the data can be loaded in the primary key order, data for Incremental Update phase is loaded in the order the changes to the data were made. Two Incremental Updates are made to confirm repeatability. After each Incremental Update, Validation Query collects information about the execution to use it later for evaluation.

This phase uses CDC extracts from the OLTP system, as mentioned earlier. They contain a column "CDC_FLAG" which contains one of the three characters I, U or D to indicate whether the row was inserted, updated or deleted since the last version. A row can be changed multiple times during a day. The extract also contains column "CDC_DSN" which is used to order the rows by time of the changes. It is specific to each data source.

The tables that are modified during the Incremental Updates, contain not only primary but also surrogate keys. Primary keys are constant but surrogate keys are updated for each recorded change.

This phase uses files from *Batch2* and *Batch3* directories for executing the first and second incremental updates.

Automated Audit phase

This phase is done after all other phases. It runs queries to check for the validity of the resulting data stored in the Data Warehouse. In the end, it generates a report with results of all the tests. Each of

the tests should be passed for the run to be considered valid.

1.2.3 Transformations

In this benchmark, transformation refers to any operation that needs to be done to prepare and load data in the Data Warehouse. In a nutshell, the transformation to be done can be summed up as:

- Convert the string-like data to the correct datatype compatible with the SUT.
- Compute surrogate keys.
- Merge fields or split fields.
- Check data for errors.
- Detect changes in dimension tables and apply the changes to the corresponding records.
- Detect changes in fact data.

Data Dependencies

In the DWH, there exist dependencies between tables, so there is an ordering to the processing of the tables. When a dependent table column refers to a column in a source table, any row in the source table that would change the result of the dependent row must be processed before the latter.

1.3 DIGen

DIGen is the automatic data generator provided by the TPC-DI benchmark. It is used to create Source Data and the audit information and it is the only acceptable way to generate them.

DIGen also produces statistics about the data generation process. This file is used to calculate the metric and for auditing and contains:

- General information about the data generation process.
- Options used.
- Rows counts for each batch.

1.4 Execution Rules & Metrics

1.4.1 Execution phases and measurements

- Preparation phase: it includes the one time tasks needed to prepare the SUT for execution of the benchmark.

  Source data is prepared in the staging area for the benchmark using the DIGen tool. A scale factor needs to be chosen.

  Also, the DWH needs to be prepared, creating all tables necessary for the execution.

- Benchmark Run: a benchmark run consists of:

1. Initialization phase: provides a SUT prepared to be benchmarked. Not timed.
2. Historical load phase: populates the DWH performing the required transformations, using data from *Batch1*. This phase is timed.
3. Incremental update 1 phase: an incremental insertion of data from *Batch2* directory is performed. It is also timed.
4. Incremental update 2 phase: an incremental insertion of data from *Batch3* directory is performed. It is also timed.
5. Automated audit phase: this phase begins immediately upon completion of the last incremental update phase. It must be executed to confirm the validity of the results.

1.4.2 Calculating throughput

The **historical load throughput** is defined using the elapsed time, in seconds, for the historical load:

$$E_H = CT_1 - CT_0,$$

where CT_i is a timestamp taken after completion of phase i. If the total number of rows of source data is R_H, the throughput is defined as

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

The **incremental update throughput** is defined similarly by

$$T_{I1} = \frac{R_{I1}}{\max(E_{I1}, 1800)}$$

and

$$T_{I2} = \frac{R_{I2}}{\max(E_{I2}, 1800)}.$$

1.4.3 Primary metrics

The **performance metric** is a combined metric using the three throughput values:

$$TPC\_DI\_RPS = trunc(geoMean(T_H, \min(T_{I1}, T_{I2}).$$

1.4.4 Price/Performance metric

This metric is defined as

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

Here, $ is the total 3-year pricing. We have not taken this measure into account.

1.5 Slowly Changing Dimensions

As explained in [AVZ20], **slowly changing dimensions (SCD)** are used in a Data Warehouse to keep the history of changes that occur in data sources. The TPC-DI benchmark focuses on what are called SDCs of type 2 with dependencies. This kind of dimension keeps track of the changes of the values by adding two temporal attributes: *StartDate* and *EndDate*, which indicate the date of insertion of the record in the database and the date when the tuple was deleted from the source table, respectively. Also, *'with dependencies'* means that the SCD table contains at least a surrogate key reference to another table, so when an update in the referenced dimension happens, the referencing table needs to be updated, too.

The dimensions of this type in the TPC-DI benchmark are *DimCustomer* and *DimAccount*, being the latter dependant of the former.

Chapter 2

Implementation Details

In this chapter, a thorough explanation of the implementation of the benchmark is provided, focusing on the underlying ideas, which are complemented by code snippets.

2.1 Setting up Oracle Database

Some members of our group used Windows OS and some Ubuntu OS so we had two types of setups for the project. The Ubuntu users set up a virtual machine using Oracle VM VirtualBox to run Oracle Linux 8.6. Windows users could directly install Oracle DB to their computers. All of the results in Chapter 3 were obtained using a computer running on Windows 64 bits, with CPU 11th Intel Core i7-11800H @ 2.30 GHz and 16 GB of RAM.

All of us used Oracle Database Express Edition (XE) 21c as the tool. It is a free version of Oracle Database. However, it had limitations of up to 12 GB of user data, up to 2 GB of database RAM and up to 2 CPU threads [Dat22].

2.2 Generating and Loading the Data

2.2.1 Data Generation

To generate the data, we have used the data generation tool provided by the TPC-DI developers: *DIGen.jar*.

2.2.2 Data transformation

Our implementation was inspired by a partial implementation of TPC-DI in MySQL [NI18].

To perform the transformations detailed in the specification of the benchmark, we have developed a Python program which makes use of both Python and OracleSQL to perform all transformations needed. In particular, command-line tool *SQL*Loader* was used to load tables with data from files, Oracle Database utility *SQL*Plus* was used to execute the queries and Python Library *python-oracledb* was used to connect to Oracle Database from Python.

For this project, only the transformations for the Historical Load were implemented, due to the time constraints.

The main class in this program is the class TPCDI_Loader, which is given the information needed to connect to the database and where to find the flat files of data generated before. Once it has this information, it creates all the tables in the database. Then, the different tables are processed one by one, following the order of their dependencies.

To create all the tables, an OracleSQL script have been defined, it can be seen in Appendix A. It has been made to adhere to the specification of the benchmark, adapting some parts to the OracleSQL syntax and restriction. For example, as OracleSQL does not implement a Boolean type, we have modeled boolean values as *char strings* with only two possible values *'true'* and *'false*.

Batch Date

To load the Batch Date, we simply read the batch date from *BatchDate.txt* and we enter into the table BatchDate the Batch number and the date.

DimDate

This table is loaded with the *sqlldr* tool, providing the *BatchDate.sql* file and the control file *DimDate.ctl*, which is shown in Listing 2.1.

```
LOAD DATA
INTO TABLE DimDate
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    SK_DateID,
    DateValue DATE "YYYY-MM-DD",
    DateDesc,
    CalendarYearID,
    CalendarYearDesc,
    CalendarQtrID,
    CalendarQtrDesc,
    CalendarMonthID,
    CalendarMonthDesc,
    CalendarWeekID,
    CalendarWeekDesc,
    DayOfWeeknumeric,
    DayOfWeekDesc,
    FiscalYearID,
    FiscalYearDesc,
    FiscalQtrID,
    FiscalQtrDesc,
    HolidayFlag
)
```

Listing 2.1: DimDate.ctl

DimTime

This table is treated very similarly to the previous one: we feed the *sqlldr* with *Time.txt* and *DimTime.ctl*, which can be seen in Listing 2.2.

```
LOAD DATA
INTO TABLE DimTime
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    SK_TimeID,
    TimeValue DATE "HH24:MI:SS",
```

```
    HourID,
    HourDesc,
    MinuteID,
    MinuteDesc,
    SecondID,
    SecondDesc,
    MarketHoursFlag,
    OfficeHoursFlag
)
```

Listing 2.2: DimTime.ctl

Industry

Once again, the *sqlldr* tool facilitates our work, just needing to provide the text and control files. From this point on, we will not show more control files in the main text, but the reader is referred to Appendix B. In this case, more precisely to Listing B.3.

StatusType

This table is analogous to the previous ones, we feed *sqlldr* with *StatusType.txt* and *StatusType.ctl*[1].

TaxRate

This table is analogous to the previous ones, we feed *sqlldr* with *TaxRate.txt* and *TaxRate.ctl*[2].

TradeType

This table is analogous to the previous ones, we feed *sqllsr* with *TradeType.txt* and *TradeType.ctl*[3].

DimCompany

This table is more complex than the previous ones. Our approach has been to divide its loading into two steps.

First, we populate a staging table, *S_Company* by means of the function *load_staging_finwire*. This function populates three staging tables: *S_Company*, *S_Financial* and *S_Security*. The idea is simple: we traverse all the *FINWIRE* files in the data directory, and for each of them, we traverse its lines, one by one. The lines can be one out of three types, which is indicated in the characters 15-17 of the line:

- CMP: the record is to be inserted into *DimCompany*.
- SEC: the record is to be inserted into *DimSecurity*.
- FIN: the record is to be inserted into *Financial*.

Now, depending on the type, the different fields are recovered and the insertion query into the staging database is composed. In the case of CMP records, additional checks needs to be made, as values can be missing and they need to be inputted as NULL into the database in this case. This does not happen with the other tables.

[1] See B.4.
[2] See B.5.
[3] See B.6.

An important concern is that DimCompany is a SDC, so newly inserted records can make it necessary to modify the previous ones, in terms of time validity: we have to set the column *IsActive* to 'false' and the column *EndDate* to *EffectiveDate* of the new record. This is done with a specific function for filling DimCompany.

After *load_staging_finwire* finishes populating the three staging tables, it's time to use these to fill the dimensions. For this task, we developed the function *load_target_dim_company*, which performs the following steps:

1. Load all records of *S_Company* into *DimCompany*, joining with previously loaded *Industry* and *StatusType* tables, and doing some checks on some of the values.

2. Creates a temporal table to perform the SDC changes in the database. This table is a copy of *DimCompany*, with a row number attribute that is useful to order the data by CompanyID and effective date.

3. Once we have this order, we update the records in *DimCompany*, comparing each record of each company with the next record for the same company. If there is such a 'next record', then the previous one is updated to deactivate it.

4. The temporal table is dropped.

This function also populates the *DImessages* table with messages generated from *DimCompany* tables, but this matter will be further detailed later.

Financial

Financial also relies on a staging table, *S_Financial*, which is populated as explained before. Once the staging table is filled, a specific function populates *Financial*, function *load_target_financial*.

This function is simpler than the previous one, and its functioning relies on the assumption that *load_target_company* has already been executed. It just joins *S_Financial* with *DimCompany*, checking the correctness of some values and making sure that the records are in the correct dates. Two queries of this type are used to address two different ways to join the values.

DimSecurity

DimSecurity is very similar to the last two tables. First, a staging table *S_Security* is populated with the function *load_staging_finwire*. Then it is populated by two queries treating two distinct cases to join the values of *S_Security*, *StatusType* and *DimCompany*. After this, as *DimSecurity* is a SDC, we do something similar as we did with *DimCompany*, we create a temporary copy of *DimSecurity* which is used to detect changes and perform them in the database.

DimBroker

Broker is also loaded in two parts. First, a staging table is populated using a control file[4]. Then, we execute simple query where only those employees matching the conditions specified in the specification are introduced in the table.

[4]See Listing B.7.

Prospect

Prospect load is similar to *Broker* load: again, a staging table is filled using a control file[5]. After this loading, some mesagges for the *DImessages* table are created. Then, two queries are performed to make the appropriate transformations into the data and introduce them into the table *Prospect*. For this purpose, the SQL function *get_marketing_template* is created and used to transform the data from the staging table properly.

Basically, the function classifies the marketing template using different ranges for net worth and income values.

Prospect needs to be updated after *DimCustomer* has been processed. With this objective, the function *update_prospect* was developed. It consists in a simple query that marks 'true' for those prospects for which there is a corresponding customer.

DimCustomer and DimAccount

The table *DimCustomer* is the most complex dimension table in the benchmark, together with *DimAccount*. As they are strongly related and these tables represent the two tables that model the SDC with dependencies tables that we mentioned in the Chapter 1, we are going to explain them together.

The data to fill this tables is first loaded into two staging tables, *S_Customer* and *S_Account*, which are processed at the same time, from the same file, similarly to what we did with *DimCompany*, *Financial* and *DimSecurity*. In this case, we have to parse a XML file, checking for all fields that are to be inserted into the staging tables.

Once the staging tables are filled, the tables start to be treated separately. A series of functions is executed as follows:

1. *load_new_customer*: fills *DimCustomer* table with those records from the staging table that are labeled as 'NEW'. The query joins *Prospect* and *S_Customer*.

2. *load_new_account*: fills *DimAccount* table with those records from the staging table that are labeled as 'NEW' or 'ADDACT'. The query joins *S_Account*, *DimBroker* and *DimCustomer*.

3. *create_trigger_UPD_customer_account*: creates a trigger which automatically updates those records in account that gets affected when the AccountID of a customer is modified.

4. *load_update_customer*: the values from *DimCustomer* table are updated with those records from *S_Customer* labeled as 'UPDCUST'. The basic idea of the query is getting the most recent valid value for each of the fields of each record. To achieve this, an individual query for each field is run. There are two types of queries: those which only need information from *S_Customer* and those which needs a join with the table *Prospect* to obtain the needed information.

5. *load_update_account*: similarly to the last function, here we are updating *DimAccount* table with those records from *S_Account* labeled as 'UPDACCT'.

6. *load_close_account*: in this case, those accounts which are labeled as 'CLOSEACCT' in the staging table are marked as closed in the *DimAccount* table.

7. *load_inact_customer*: this function marks set an EndDate for those customers that are labeled as 'INACT' in the staging table. They are also marked as not valid.

[5]See Listing B.8.

8. *load_inact_account*: this function marks set an EndDate for those accounts that are labeled as 'INACT' in the staging table. They are also marked as not valid.

DimTrade

A staging table, *S_Trade* is populated by means of a control file[6]. Then, the function *load_trade* executes a query that makes the pertinent transformations on the data. These transformations basically consist of inserting one value or another into *DimTrade* depending on the values held in *S_Trade*.

FactCashBalances

First, we make use of a control file[7] to fill a staging table. After that, a query joining *S_Cash_Balances*, *DimAccount* and *DimDate* properly populates the table *FactCashBalances*.

FactWatches

Similarly to *FactCashBalances*, this fact table is filled using a staging table populated by means of a control file[8] and a simple query which joins appropriately different tables. In this case, distinctions needs to be made using the values of the W_Action column in *S_Watches*.

FactHoldings

Again, a control file[9] and a properly done query makes the work.

FactMarketHistory

This table is the most complex one. First, a staging table with the facts is filled using a control file[10]. Then, the *FactMarketHistory* table needs to be populated. The problem at this point is that the query is really complex: it asks to get, for each record, the maximum and minimum value of the security close price for the past year, as well as the dates in which these values occurred.

To do this, we performed an analytic query, based on partitions and windows. As we will see in the results, this query is too complex and not very useful. We have not been able to develop a better solution for this problem. Therefore, we omitted this table from our execution.

The problem of this query is that we need to retrieve the dates when the highest and lowest values occurred. This involves several subqueries and multiple joins. We have not been able to come up with a fast implementation in OracleSQL, because it is not possible to return the values of the row in which we find the maximum value without using subqueries somehow. The fact that Oracle XE only allows for 2GB of RAM used makes it crash at some point. Our query can be seen in Listing 2.3, where one can see the problem: recovering the dates makes it unfeasible.

```
1 INSERT INTO FactMarketHistory (SK_SecurityID, SK_CompanyID, SK_DateID, PERatio, Yield,
    FiftyTwoWeekHigh,
2     SK_FiftyTwoWeekHighDate, FiftyTwoWeekLow, SK_FiftyTwoWeekLowDate, ClosePrice, DayHigh
    , DayLow, Volume, BatchID)
3 WITH MAXCLOSE AS (
4     SELECT DM_DATE, DM_S_SYMB, DM_CLOSE, MAX(DM_HIGH) OVER (PARTITION BY DM_S_SYMB ORDER
    BY DM_DATE ROWS BETWEEN 365 PRECEDING AND CURRENT ROW) AS MAXC
```

[6] See B.9.
[7] See B.10.
[8] See B.11.
[9] See B.12.
[10] See **??**.

```
    FROM S_Daily_Market DM),
  MAXDATE AS (
    SELECT DM.DM_S_SYMB AS SYMB, MIN(DM.DM_DATE) AS MAXD, MAX(MC.MAXC) AS MAXC, MC.
    DM_DATE AS BASEDATE
    FROM S_Daily_Market DM
    JOIN MAXCLOSE MC ON MC.DM_S_SYMB = DM.DM_S_SYMB
    WHERE DM.DM_DATE >= MC.DM_DATE - 365 AND DM.DM_DATE < MC.DM_DATE AND DM.DM_HIGH = MC.
    MAXC
    GROUP BY DM.DM_S_SYMB, MC.DM_DATE
  ),
  MINCLOSE AS (
    SELECT DM.DM_DATE, DM.DM_S_SYMB, MIN(DM.DM_LOW) OVER (PARTITION BY DM.DM_S_SYMB ORDER
     BY DM.DM_DATE ROWS BETWEEN 365 PRECEDING AND CURRENT ROW) AS MINC
    FROM S_Daily_Market DM),
  MINDATE AS (
  SELECT DM.DM_S_SYMB AS SYMB, MIN(DM.DM_DATE) AS MIND, MAX(MC.MINC) AS MINC, MC.DM_DATE
    AS BASEDATE
    FROM S_Daily_Market DM
    JOIN MINCLOSE MC ON MC.DM_S_SYMB = DM.DM_S_SYMB
    WHERE DM.DM_DATE >= MC.DM_DATE - 365 AND DM.DM_DATE < MC.DM_DATE AND DM.DM_LOW = MC.
    MINC
    GROUP BY DM.DM_S_SYMB, MC.DM_DATE
  ),
  EPS AS (
    SELECT DM.DM_DATE, DM.DM_S_SYMB, SUM(F.FI_BASIC_EPS) OVER (PARTITION BY DM.DM_S_SYMB
    ORDER BY DM.DM_DATE ROWS BETWEEN 130 PRECEDING AND CURRENT ROW) AS TOTAL_EPS
    FROM S_Daily_Market DM
    JOIN DimSecurity DS ON (DM.DM_S_SYMB = DS.Symbol)
    JOIN Financial F ON (F.SK_CompanyID = DS.SK_CompanyID))
SELECT  DS.SK_SecurityID,
    DS.SK_CompanyID,
    DD.SK_DateID,
      CASE
        WHEN EPS.TOTAL_EPS IS NULL THEN NULL
        ELSE DM.DM_CLOSE/EPS.TOTAL_EPS
      END,
      DS.DIVIDEND/DM.DM_CLOSE*100,
      MXC.MAXC,
      DMX.SK_DateID,
      MNC.MINC,
      DMN.SK_DateID,
      DM.DM_CLOSE,
      DM.DM_HIGH,
      DM.DM_LOW,
      DM.DM_VOL,
      1
FROM    S_Daily_Market DM INNER JOIN DimDate DD ON (TO_CHAR(DM.DM_DATE, 'YYYY-MM-DD') =
    TO_CHAR(DD.DateValue, 'YYYY-MM-DD'))
                          INNER JOIN DimSecurity DS ON (DM.DM_S_SYMB = DS.Symbol)
                          INNER JOIN MAXCLOSE MXC ON (DM.DM_S_SYMB = MXC.DM_S_SYMB AND
    MXC.DM_DATE = DM.DM_DATE)
                          INNER JOIN MAXDATE MXD ON (MXD.BASEDATE = DM.DM_DATE AND MXD.
    SYMB = DM.DM_S_SYMB)
                          INNER JOIN DimDate DMX ON (TO_CHAR(MXD.MAXD, 'YYYY-MM-DD') =
    TO_CHAR(DMX.DateValue, 'YYYY-MM-DD'))
                          INNER JOIN MINCLOSE MNC ON (DM.DM_S_SYMB = MNC.DM_S_SYMB AND
    MNC.DM_DATE = MNC.DM_DATE)
                          INNER JOIN MINDATE MND ON (MND.BASEDATE = DM.DM_DATE AND MND.
    SYMB = DM.DM_S_SYMB)
                          INNER JOIN DimDate DMN ON (TO_CHAR(MND.MIND, 'YYYY-MM-DD') =
    TO_CHAR(DMN.DateValue, 'YYYY-MM-DD'))
```

```
53                              INNER JOIN EPS ON (DM.DM_S_SYMB = EPS.DM_S_SYMB AND DM.DM_DATE
    = EPS.DM_DATE)
54 WHERE    DS.EffectiveDate <= DM.DM_DATE AND DM.DM_DATE < DS.EndDate;
```

Listing 2.3: FactMarketHistory load query.

DImessages

The table *DImessages* is used to store alert messages for records, statuses of different execution phases and validation messages.

Alert records are written when columns have values that do not make sense. For example, a record will be inserted DImessages if a row in table *DimCustomer* has value of column *Tier* that is not one of the valid values, namely 1, 2 or 3. Such invalid records were deliberately inserted by the system during the Data Generation phase.

DImessages also stores Validation Messages of *Batch Validation Queries* that are run after the Historical Load. They are part of the *Audit Phase* that is described in section 1.2.2. They include the records of the number of records in the resulting tables.

Chapter 3

Results and discussion

Below the results of executing Historical Load of TPC-DI are shown. We decided to run the Load with 4 scale factors, namely 3, 5, 7 and 9. For each scale factor, we saved the records of DIMessages and the execution time of the entire Historical Load.

Figure 3.1 below shows the execution times for each scale factor. The execution time increased at about the same rate, around 3.1 minutes for the next chosen scale factor. The increase from scale factor 5 to 7 was slightly higher, at 4.2 minutes.

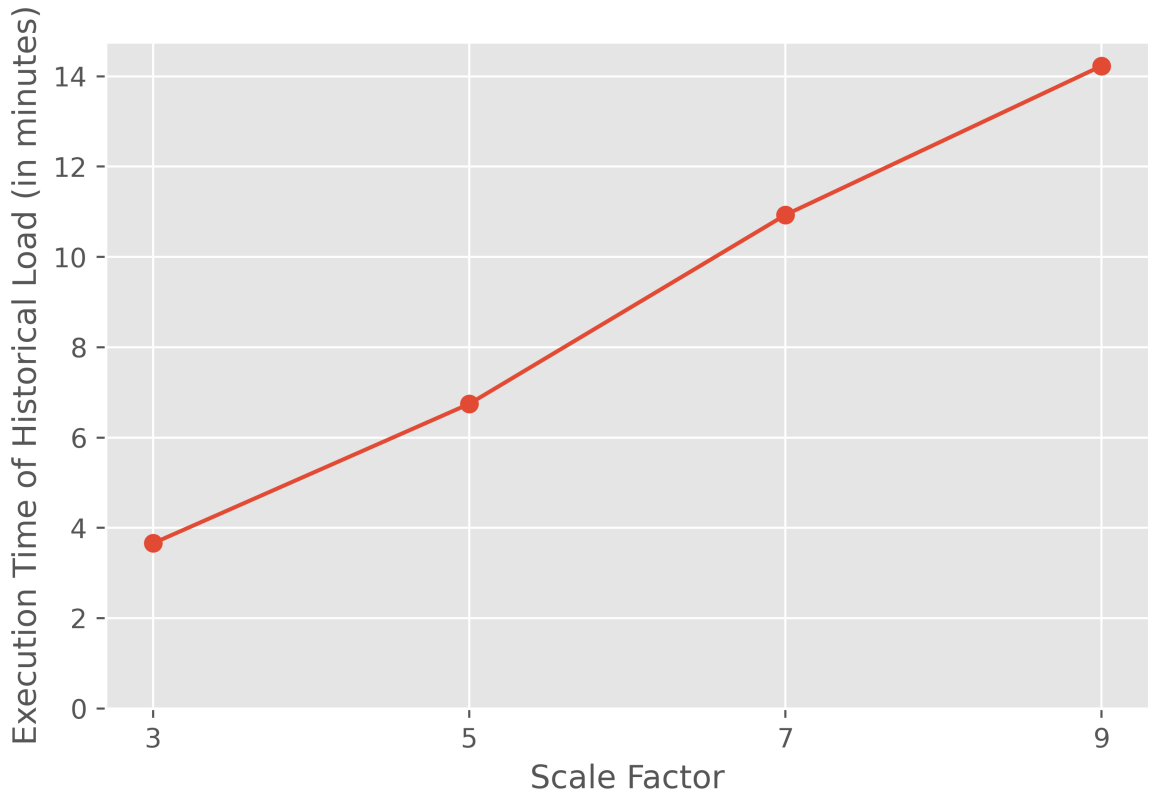

Figure 3.1: Execution Time of Historical Load by Scale Factor.

Figure 3.2 shows the number of records loaded for each scale factor. There were around 850,000 more

records for the next chosen scale factor. However, the rate of this increase decreased slightly. For example, there were around 1 million records more records in scale factor 5 compare to scale factor 3, while there were only 820,000 records more in scale factor 9 compared to scale factor 7.

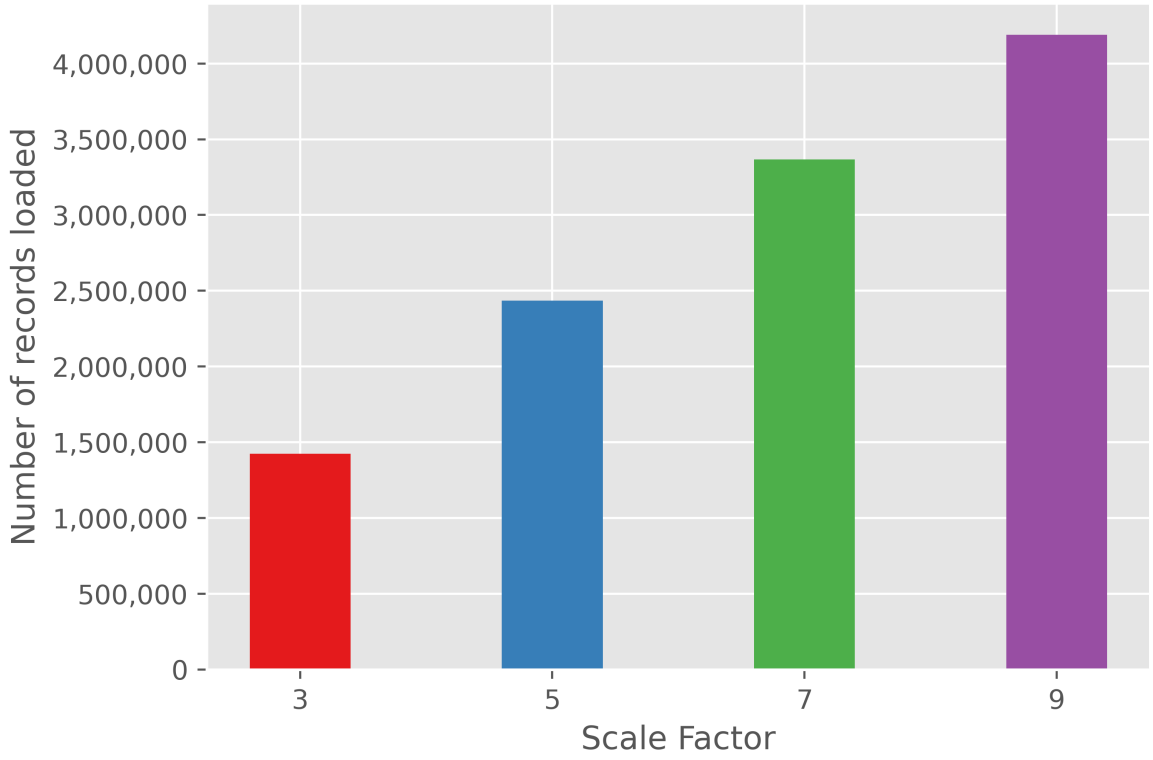

Figure 3.2: Cumulative Number of Records by Scale Factor.

You can see the number of records for each table in Figure 3.3. For all the scale factor, the Dimension Table DimFactCash and Fact Tables Balances, FactWatches, FactHoldings were the tables with the greatest number of records in that order, and accounted for around 91% of all the records. All the Reference Tables, including TradeType, StatusType, Industry and TaxRate, had a small number of records.

Figure 3.4 shows the number records with alerts which are recorded with DImessages, as mentioned previously. The proportion of alerted records are as expected about the same for all the scale factors, with 0.03% of all the records being alerted records of Table *DimCustomer* and around 0.0055% of all the records being alerted records of Table *DimTrade*. From this, we can assume that the records with alerts were not significantly greater than usual and our implementation inserted the records correctly but further assessment is needed.

Now, in Table 3.1 we are comparing the results obtained in [AVZ20] using PostgreSQL and our results using OracleSQL. The only scale factors shared are 3 and 5, and we can see how OracleSQL is performing faster, but we need to take into account that we are not loading FactMarketHistory. Nonetheless, in the article the authors show the time taken for each table to load for scale factor 3, so we can subtract the time taken by *FactMarketHistory* and make a more fair comparisons of the times. This is shown in Table 3.2, and we can see how disregarding this table makes OracleSQL a good option for this process. Of course, without that table, the implementation is useless, so there would be work to do in order to be able to load the table in a reasonable time. At the moment we don't know if this is even possible.

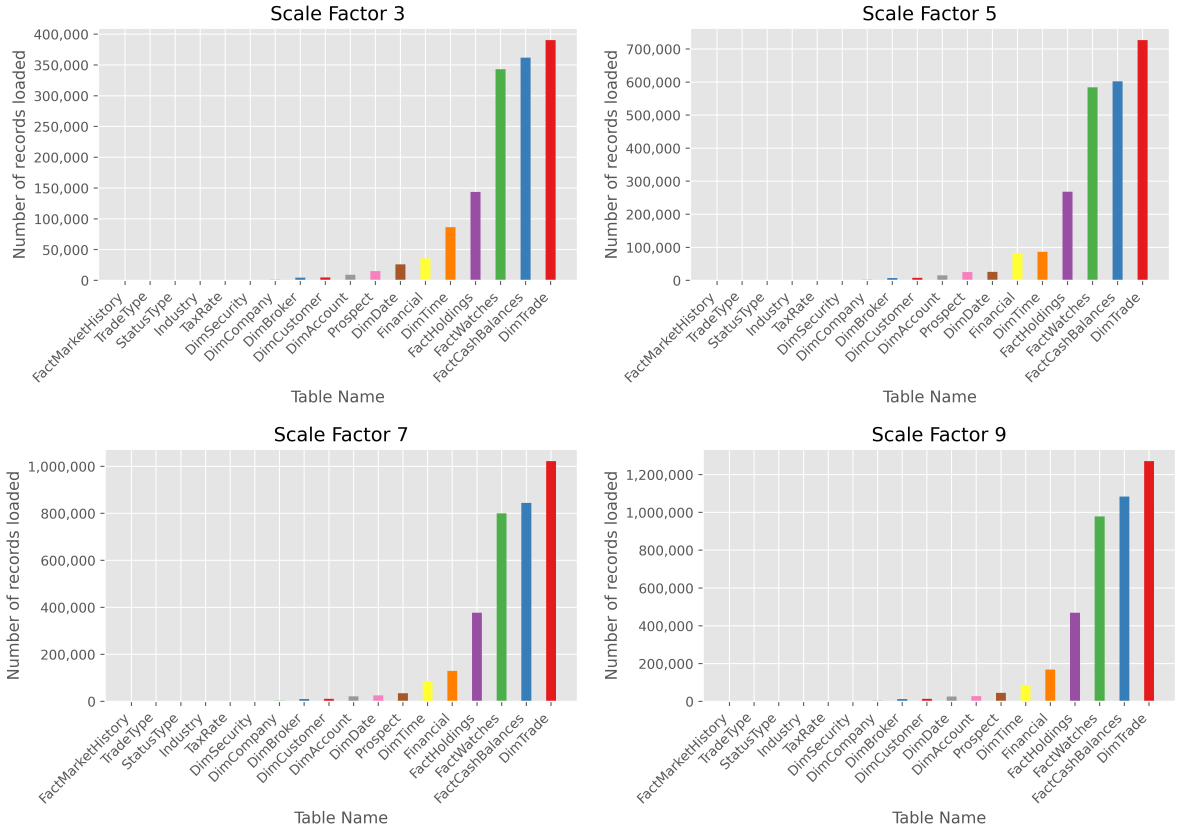

Figure 3.3: Number of Records by Table for each Scale Factor.

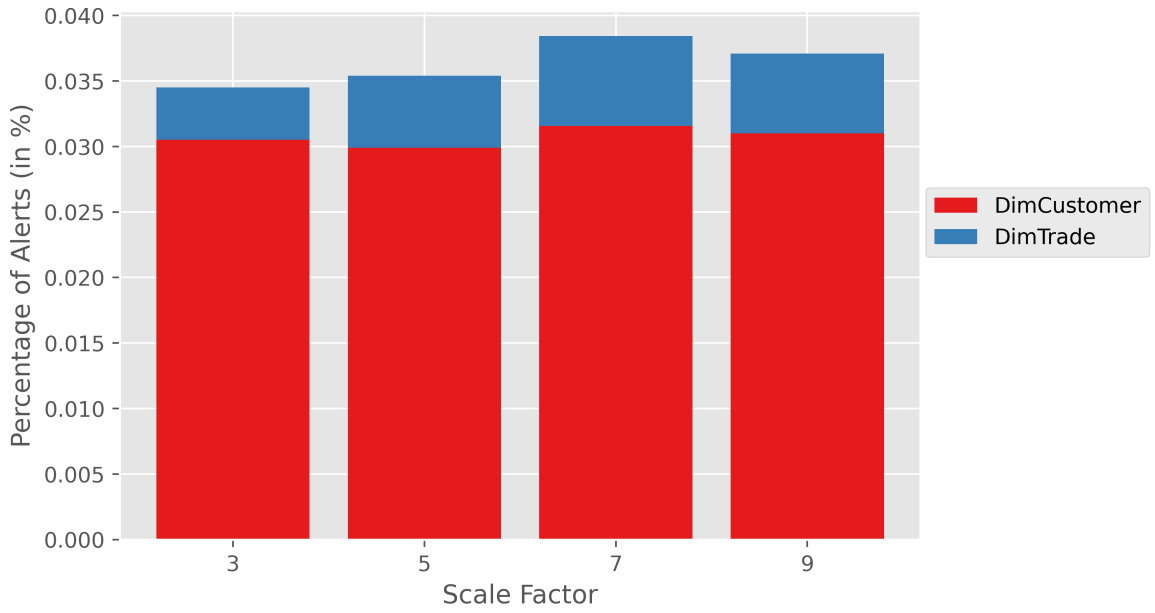

Figure 3.4: Proportion of Records with Alerts by Scale Factor

Technology	SF=3	SF=5
PostgreSQL	00:12:50	00:22:31
OracleSQL	00:03:40*	00:06:45*

Table 3.1: Comparison between PostgreSQL results ([AVZ20]) and OracleSQL.

*: Without the *FactMarketHistory* table.

SF=3	Time
PostgreSQL	00:10:20
OracleSQL	00:03:40

Table 3.2: Comparison between PostgreSQL results ([AVZ20]) and OracleSQL without *FactMarketHistory* table for scale factor 3.

Chapter 4

Conclusion

This project helped us experience first-hand Data Integration for a system. We understood that decision support systems can require input data from various sources with various formats, the amount of detail and constraints. The data needs to be processed in various ways for it to be loaded to a decision support system, often involving combinations of several data sources. It is important to think carefully about the strategy to take in order to create a healthy ETL pipeline and now we are aware of different tools and ideas that are useful for this purpose.

Also, we have experienced the importance of understanding well the operations and transformations used. For instance, we have not been able to come up with an efficient way to recover the values of a row associated with a maximum, which seems like an important operation to have implemented in the system, as it is usual to encounter requirements as those in *FactMarketHistory*.

Further Work

We understand that our implementation of TPC-DI is not complete without the following steps:

- Improving the queries for table *FactMarketHistory*.
- Executing the two phases of *Incremental Updates*.
- Implementing the rest of *Automated Audit Phase* after all the 3 phases are complete.

Bibliography

[TPC14] Transaction Processing Performance Council (TPC). *TPC BENCHMARK™ DI - Data Integration*. 1.1.0. TPC. Nov. 2014. URL: https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp.

[AVZ20] Judith Awiti, Alejandro A. Vaisman, and Esteban Zimányi. "Design and implementation of ETL processes using BPMN and relational algebra". In: *Data & Knowledge Engineering* 129 (2020), p. 101837. ISSN: 0169-023X. DOI: https://doi.org/10.1016/j.datak.2020.101837. URL: https://www.sciencedirect.com/science/article/pii/S0169023X19306111.

[Dat22] Oracle® Database. *Oracle Database Express Edition*. Oracle®. 2022. URL: https://www.oracle.com/be/database/technologies/appdev/xe.html.

[NI18] Dwi Prasetyo Adi Nugroho and Haydar Ali Ismail. *tpcdi-kit*. https://github.com/haydarai/tpcdi-kit. 2018.

[VZ22] Alejandro Vaisman and Esteban Zimányi. *Data Warehouse Systems*. Springer Berlin Heidelberg, 2022. DOI: 10.1007/978-3-662-65167-4.

Appendix A

Creation script

In Listing A.1 the script for creating all tables of the database is shown. This script creates the schema explained in TPC-DI documentation adapted to the OracleSQL syntax and restrictions.

```
CREATE TABLE BatchDate( BatchNumber NUMBER(3,0),
            BatchDate DATE NOT NULL
);

CREATE TABLE DimBroker  ( SK_BrokerID  NUMBER(11,0) GENERATED BY DEFAULT AS IDENTITY
    PRIMARY KEY,
              BrokerID  NUMBER(11,0) NOT NULL,
              ManagerID  NUMBER(11,0),
              FirstName       CHAR(50) NOT NULL,
              LastName       CHAR(50) NOT NULL,
              MiddleInitial       CHAR(1),
              Branch       CHAR(50),
              Office       CHAR(50),
              Phone       CHAR(14),
              IsCurrent CHAR(5) NOT NULL CHECK (IsCurrent = 'true' OR IsCurrent = 'false'
    ),
              BatchID NUMBER(5,0) NOT NULL,
              EffectiveDate date NOT NULL,
              EndDate date NOT NULL
);

CREATE TABLE DimCustomer  ( SK_CustomerID  NUMBER(11,0) GENERATED BY DEFAULT AS IDENTITY
    PRIMARY KEY,
              CustomerID NUMBER(11,0) 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 NUMBER(1,0),
              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),
```

```
37              Phone2  char(30),
38              Phone3  char(30),
39              Email1  char(50),
40              Email2  char(50),
41              NationalTaxRateDesc varchar(50),
42              NationalTaxRate NUMBER(6,5),
43              LocalTaxRateDesc  varchar(50),
44              LocalTaxRate  NUMBER(6,5),
45              AgencyID  char(30),
46              CreditRating NUMBER(5,0),
47              NetWorth  NUMBER(10),
48              MarketingNameplate varchar(100),
49              IsCurrent CHAR(5) NOT NULL CHECK (IsCurrent = 'true' OR IsCurrent = 'false'
    ),
50              BatchID NUMBER(5,0) NOT NULL,
51              EffectiveDate date NOT NULL,
52              EndDate date NOT NULL
53 );
54
55 CREATE TABLE DimAccount  ( SK_AccountID  NUMBER(11,0) GENERATED BY DEFAULT AS IDENTITY
    PRIMARY KEY,
56                          AccountID  NUMBER(11,0) NOT NULL,
57                          SK_BrokerID  NUMBER(11,0) NOT NULL REFERENCES DimBroker (
    SK_BrokerID),
58                          SK_CustomerID  NUMBER(11,0) NOT NULL REFERENCES DimCustomer (
    SK_CustomerID),
59                          Status       CHAR(10) NOT NULL,
60                          AccountDesc       varchar(50),
61                          TaxStatus NUMBER(1,0) NOT NULL CHECK (TaxStatus = 0 OR
    TaxStatus = 1 OR TaxStatus = 2),
62                          IsCurrent CHAR(5) NOT NULL CHECK (IsCurrent = 'true' OR
    IsCurrent = 'false'),
63                          BatchID NUMBER(5,0) NOT NULL,
64                          EffectiveDate date NOT NULL,
65                          EndDate date NOT NULL
66 );
67
68
69 CREATE TABLE DimCompany (   SK_CompanyID NUMBER(11,0) GENERATED BY DEFAULT AS IDENTITY
    PRIMARY KEY,
70              CompanyID NUMBER(11,0) NOT NULL,
71              Status CHAR(10) Not NULL,
72              Name CHAR(60) Not NULL,
73              Industry CHAR(50) Not NULL,
74              SPrating CHAR(4),
75              isLowGrade CHAR(5) NOT NULL CHECK (isLowGrade = 'true' OR isLowGrade = '
    false' or isLowGrade = 'NULL'),
76              CEO CHAR(100) Not NULL,
77              AddressLine1 CHAR(80),
78              AddressLine2 CHAR(80),
79              PostalCode CHAR(12) Not NULL,
80              City CHAR(25) Not NULL,
81              StateProv CHAR(20) Not NULL,
82              Country CHAR(24),
83              Description CHAR(150) Not NULL,
84              FoundingDate DATE,
85              IsCurrent CHAR(5) NOT NULL CHECK (IsCurrent = 'true' OR IsCurrent = 'false'
    ),
86              BatchID NUMBER(5,0) Not NULL,
87              EffectiveDate DATE Not NULL,
88              EndDate DATE Not NULL
```

```
);

CREATE TABLE DimDate (  SK_DateID NUMBER(11,0) PRIMARY KEY,
            DateValue DATE Not NULL,
            DateDesc CHAR(20) Not NULL,
            CalendarYearID NUMBER(4) Not NULL,
            CalendarYearDesc CHAR(20) Not NULL,
            CalendarQtrID NUMBER(5) Not NULL,
            CalendarQtrDesc CHAR(20) Not NULL,
            CalendarMonthID NUMBER(6) Not NULL,
            CalendarMonthDesc CHAR(20) Not NULL,
            CalendarWeekID NUMBER(6) Not NULL,
            CalendarWeekDesc CHAR(20) Not NULL,
            DayOfWeeknumeric NUMBER(1) Not NULL,
            DayOfWeekDesc CHAR(10) Not NULL,
            FiscalYearID NUMBER(4) Not NULL,
            FiscalYearDesc CHAR(20) Not NULL,
            FiscalQtrID NUMBER(5) Not NULL,
            FiscalQtrDesc CHAR(20) Not NULL,
            HolidayFlag CHAR(5) NOT NULL CHECK (HolidayFlag = 'true' OR HolidayFlag = '
    false')
);

CREATE TABLE DimSecurity( SK_SecurityID NUMBER(11,0) GENERATED BY DEFAULT AS IDENTITY
    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 NUMBER(11,0) Not NULL REFERENCES DimCompany (SK_CompanyID),
              SharesOutstanding NUMBER(12,0) Not NULL,
              FirstTrade DATE Not NULL,
              FirstTradeOnExchange DATE Not NULL,
              Dividend NUMBER(10,2) Not NULL,
              IsCurrent CHAR(5) NOT NULL check (IsCurrent = 'false' or IsCurrent = 'true'
    ),
              BatchID NUMBER(5) Not NULL,
              EffectiveDate DATE Not NULL,
              EndDate DATE Not NULL
);

CREATE TABLE DimTime ( SK_TimeID NUMBER(11,0) PRIMARY KEY,
            TimeValue DATE Not NULL,
            HourID NUMBER(2) Not NULL,
            HourDesc CHAR(20) Not NULL,
            MinuteID NUMBER(2) Not NULL,
            MinuteDesc CHAR(20) Not NULL,
            SecondID NUMBER(2) Not NULL,
            SecondDesc CHAR(20) Not NULL,
            MarketHoursFlag CHAR(5) NOT NULL check (MarketHoursFlag = 'false' or
    MarketHoursFlag = 'true'),
            OfficeHoursFlag CHAR(5) NOT NULL check (OfficeHoursFlag = 'false' or
    OfficeHoursFlag = 'true')
);

CREATE TABLE DimTrade ( TradeID NUMBER(11,0) Not NULL,
            SK_BrokerID NUMBER(11,0) REFERENCES DimBroker (SK_BrokerID),
            SK_CreateDateID NUMBER(11,0) REFERENCES DimDate (SK_DateID),
            SK_CreateTimeID NUMBER(11,0) REFERENCES DimTime (SK_TimeID),
            SK_CloseDateID NUMBER(11,0) REFERENCES DimDate (SK_DateID),
```

```
            SK_CloseTimeID NUMBER(11,0) REFERENCES DimTime (SK_TimeID),
            Status CHAR(10) Not NULL,
            Type CHAR(12) Not NULL,
            CashFlag CHAR(5) NOT NULL check (CashFlag = 'false' or CashFlag = 'true'),
            SK_SecurityID NUMBER(11,0) Not NULL REFERENCES DimSecurity (SK_SecurityID),
            SK_CompanyID NUMBER(11,0) Not NULL REFERENCES DimCompany (SK_CompanyID),
            Quantity NUMBER(6,0) Not NULL,
            BidPrice NUMBER(8,2) Not NULL,
            SK_CustomerID NUMBER(11,0) Not NULL REFERENCES DimCustomer (SK_CustomerID),
            SK_AccountID NUMBER(11,0) Not NULL REFERENCES DimAccount (SK_AccountID),
            ExecutedBy CHAR(64) Not NULL,
            TradePrice NUMBER(8,2),
            Fee NUMBER(10,2),
            Commission NUMBER(10,2),
            Tax NUMBER(10,2),
            BatchID NUMBER(5) Not Null
);

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

CREATE TABLE FactCashBalances ( SK_CustomerID NUMBER(11,0) Not Null REFERENCES
    DimCustomer (SK_CustomerID),
              SK_AccountID NUMBER(11,0) Not Null REFERENCES DimAccount (SK_AccountID),
              SK_DateID NUMBER(11,0) Not Null REFERENCES DimDate (SK_DateID),
              Cash NUMBER(15,2) Not Null,
              BatchID NUMBER(5)
);

CREATE TABLE FactHoldings ( TradeID NUMBER(11,0) Not Null,
              CurrentTradeID NUMBER(11,0) Not Null,
              SK_CustomerID NUMBER(11,0) Not NULL REFERENCES DimCustomer (SK_CustomerID),
              SK_AccountID NUMBER(11,0) Not NULL REFERENCES DimAccount (SK_AccountID),
              SK_SecurityID NUMBER(11,0) Not NULL REFERENCES DimSecurity (SK_SecurityID),
              SK_CompanyID NUMBER(11,0) Not NULL REFERENCES DimCompany (SK_CompanyID),
              SK_DateID NUMBER(11,0) Not NULL REFERENCES DimDate (SK_DateID),
              SK_TimeID NUMBER(11,0) Not NULL REFERENCES DimTime (SK_TimeID),
              CurrentPrice NUMBER(8,2) CHECK (CurrentPrice > 0) ,
              CurrentHolding NUMBER(6) Not NULL,
              BatchID NUMBER(5)
);

CREATE TABLE FactMarketHistory (    SK_SecurityID NUMBER(11,0) Not Null REFERENCES
    DimSecurity (SK_SecurityID),
                SK_CompanyID NUMBER(11,0) Not Null REFERENCES DimCompany (SK_CompanyID)
    ,
                SK_DateID NUMBER(11,0) Not Null REFERENCES DimDate (SK_DateID),
                PERatio NUMBER(10,2),
                Yield NUMBER(5,2) Not Null,
                FiftyTwoWeekHigh NUMBER(8,2) Not Null,
                SK_FiftyTwoWeek NUMBER(11,0) Not Null,
                FiftyTwoWeekLow NUMBER(8,2) Not Null,
                SK_FiftyTwoWeekL NUMBER(11,0) Not Null,
                ClosePrice NUMBER(8,2) Not Null,
                DayHigh NUMBER(8,2) Not Null,
                DayLow NUMBER(8,2) Not Null,
```

```
                    Volume NUMBER(12) Not Null,
                    BatchID NUMBER(5)
);

CREATE TABLE FactWatches ( SK_CustomerID NUMBER(11,0) Not NULL REFERENCES DimCustomer (SK_CustomerID),
                SK_SecurityID NUMBER(11,0) Not NULL REFERENCES DimSecurity (SK_SecurityID),
                SK_DateID_DatePlaced NUMBER(11,0) Not NULL REFERENCES DimDate (SK_DateID),
                SK_DateID_DateRemoved NUMBER(11,0) REFERENCES DimDate (SK_DateID),
                BatchID NUMBER(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 NUMBER(11,0) Not NULL REFERENCES DimCompany (SK_CompanyID),
              FI_YEAR NUMBER(4) Not NULL,
              FI_QTR NUMBER(1) Not NULL,
              FI_QTR_START_DATE DATE Not NULL,
              FI_REVENUE NUMBER(15,2) Not NULL,
              FI_NET_EARN NUMBER(15,2) Not NULL,
              FI_BASIC_EPS NUMBER(10,2) Not NULL,
              FI_DILUT_EPS NUMBER(10,2) Not NULL,
              FI_MARGIN NUMBER(10,2) Not NULL,
              FI_INVENTORY NUMBER(15,2) Not NULL,
              FI_ASSETS NUMBER(15,2) Not NULL,
              FI_LIABILITY NUMBER(15,2) Not NULL,
              FI_OUT_BASIC NUMBER(12) Not NULL,
              FI_OUT_DILUT NUMBER(12) Not NULL
);

CREATE TABLE Prospect ( AgencyID CHAR(30) NOT NULL UNIQUE,
              SK_RecordDateID NUMBER(11,0) NOT NULL,
              SK_UpdateDateID NUMBER(11,0) NOT NULL REFERENCES DimDate (SK_DateID),
              BatchID NUMBER(5) NOT NULL,
              IsCustomer CHAR(5) NOT NULL check (IsCustomer = 'false' or IsCustomer = 'true'),
              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 NUMBER(9),
              numericberCars NUMBER(2),
              numericberChildren NUMBER(2),
              MaritalStatus CHAR(1),
              Age NUMBER(3),
              CreditRating NUMBER(4),
              OwnOrRentFlag CHAR(1),
              Employer CHAR(30),
              numericberCreditCards NUMBER(2),
              NetWorth NUMBER(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 NUMBER(6,5) Not NULL
);

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


CREATE TABLE Audit_ ( DataSet CHAR(20) Not Null,
              BatchID NUMBER(5),
              Date_ DATE,
              Attribute CHAR(50) not null,
              Value NUMBER(15),
              DValue NUMBER(15,5)
);

-- staging tables

CREATE TABLE S_Company (
  PTS CHAR(15) NOT NULL,
  REC_TYPE CHAR(3) NOT NULL,
  COMPANY_NAME CHAR(60),
  CIK CHAR(10) NOT NULL,
  STATUS CHAR(4) NOT NULL,
  INDUSTRY_ID CHAR(2) NOT NULL,
  SP_RATING CHAR(4),
  FOUNDING_DATE CHAR(8),
  ADDR_LINE_1 CHAR(80),
  ADDR_LINE_2 CHAR(80),
  POSTAL_CODE CHAR(12),
  CITY CHAR(25),
  STATE_PROVINCE CHAR(20),
  COUNTRY CHAR(24),
  CEO_NAME CHAR(46),
  DESCRIPTION CHAR(150)
);

CREATE TABLE S_Security (
  PTS CHAR(15) NOT NULL,
  REC_TYPE CHAR(3) NOT NULL,
  SYMBOL CHAR(15) NOT NULL,
  ISSUE_TYPE CHAR(6) NOT NULL,
  STATUS CHAR(4) NOT NULL,
  NAME CHAR(70) NOT NULL,
  EX_ID CHAR(6) NOT NULL,
  SH_OUT CHAR(13) NOT NULL,
  FIRST_TRADE_DATE CHAR(8) NOT NULL,
  FIRST_TRADE_EXCHANGE CHAR(8) NOT NULL,
  DIVIDEN CHAR(12) NOT NULL,
  COMPANY_NAME_OR_CIK CHAR(60) NOT NULL
```

```
);

CREATE TABLE S_Financial(
  PTS CHAR(15),
  REC_TYPE CHAR(3),
  YEAR CHAR(4),
  QUARTER CHAR(1),
  QTR_START_DATE CHAR(8),
  POSTING_DATE CHAR(8),
  REVENUE CHAR(17),
  EARNINGS CHAR(17),
  EPS CHAR(12),
  DILUTED_EPS CHAR(12),
  MARGIN CHAR(12),
  INVENTORY CHAR(17),
  ASSETS CHAR(17),
  LIABILITIES CHAR(17),
  SH_OUT CHAR(13),
  DILUTED_SH_OUT CHAR(13),
  CO_NAME_OR_CIK CHAR(60)
);

CREATE TABLE S_Prospect(
    AGENCY_ID CHAR(30) NOT NULL,
    LAST_NAME CHAR(30) NOT NULL,
    FIRST_NAME CHAR(30) NOT NULL,
    MIDDLE_INITIAL CHAR(1),
    GENDER CHAR(1),
    ADDRESS_LINE_1 CHAR(80),
    ADDRESS_LINE_2 CHAR(80),
    POSTAL_CODE CHAR(12),
    CITY CHAR(25) NOT NULL,
    STATE CHAR(20) NOT NULL,
    COUNTRY CHAR(24),
    PHONE CHAR(30),
    INCOME NUMBER(9),
    NUMBER_CARS NUMBER(2),
    NUMBER_CHILDREM NUMBER(2),
    MARITAL_STATUS CHAR(1),
    AGE NUMBER(3),
    CREDIT_RATING NUMBER(4),
    OWN_OR_RENT_FLAG CHAR(1),
    EMPLOYER CHAR(30),
    NUMBER_CREDIT_CARDS NUMBER(2),
    NET_WORTH NUMBER(12)
);

CREATE TABLE S_Watches(
    W_C_ID INTEGER NOT NULL,
    W_S_SYMB CHAR(15) NOT NULL,
    W_DTS DATE NOT NULL,
    W_ACTION CHAR(4) NOT NULL
);

CREATE TABLE S_Cash_Balances(
    CT_CA_ID INTEGER NOT NULL,
    CT_DTS DATE NOT NULL,
    CT_AMT CHAR(20) NOT NULL,
    CT_NAME CHAR(100) NOT NULL
);

```

```
CREATE TABLE S_Broker(
    EmployeeID INTEGER NOT NULL,
    ManagerID INTEGER NOT NULL,
    EmployeeFirstName CHAR(30) NOT NULL,
    EmployeeLastName CHAR(30) NOT NULL,
    EmployeeMI CHAR(1),
    EmployeeJobCode NUMBER(3),
    EmployeeBranch CHAR(30),
    EmployeeOffice CHAR(10),
    EmployeePhone CHAR(14)
);

CREATE TABLE S_Account ( SK_AccountID  NUMBER(11,0),
            AccountID  NUMBER(11,0),
            BrokerID  NUMBER(11,0),
            CustomerID NUMBER(11,0),
            Status       CHAR(10) NOT NULL,
            AccountDesc       varchar(50),
            TaxStatus NUMBER(1,0),
            BatchID NUMBER(5,0),
            EffectiveDate date,
            EndDate date,
            ActionType CHAR(10)
);

CREATE TABLE S_Customer  (  ActionType CHAR(10),
                            CustomerID NUMBER(11,0),
              TaxID CHAR(20),
              Status CHAR(10),
              LastName CHAR(30),
              FirstName CHAR(30),
              MiddleInitial CHAR(1),
              Gender CHAR(1),
              Tier NUMBER(1,0),
              DOB date,
              AddressLine1  varchar(80),
              AddressLine2  varchar(80),
              PostalCode  char(12),
              City  char(25),
              StateProv char(20),
              Country char(24),
              Phone1  char(30),
              Phone2  char(30),
              Phone3  char(30),
              Email1  char(50),
              Email2  char(50),
              NationalTaxRateDesc varchar(50),
              NationalTaxRate NUMBER(6,5),
              LocalTaxRateDesc  varchar(50),
              LocalTaxRate  NUMBER(6,5),
              BatchID NUMBER(5,0),
              EffectiveDate date,
              EndDate date
);

CREATE TABLE S_Trade (
      cdc_flag CHAR(1),
      cdc_dsn NUMBER(12),
      t_id NUMBER(15),
      t_dts DATE,
      t_st_id CHAR(4),
```

```
444       t_tt_id CHAR(3),
445       t_is_cash CHAR(3),
446       t_s_symb CHAR(15) NOT NULL,
447       t_qty NUMBER(6) NOT NULL,
448       t_bid_price NUMBER(8),
449       t_ca_id NUMBER(11),
450       t_exec_name CHAR(49),
451       t_trade_price NUMBER(8),
452       t_chrg NUMBER(10),
453       t_comm NUMBER(10),
454       t_tax NUMBER(10)
455 );
456
457 CREATE TABLE S_Trade_History (
458     th_t_id NUMBER(15),
459     th_dts DATE,
460     th_st_id CHAR(4)
461 );
462
463 exit
```

Listing A.1: oracle-schema.sql

Appendix B

Control files

In this appendix, we show all control files, in the order of appearance in the text.

DimDate

See 2.2.2.

```
LOAD DATA
INTO TABLE DimDate
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    SK_DateID,
    DateValue DATE "YYYY-MM-DD",
    DateDesc,
    CalendarYearID,
    CalendarYearDesc,
    CalendarQtrID,
    CalendarQtrDesc,
    CalendarMonthID,
    CalendarMonthDesc,
    CalendarWeekID,
    CalendarWeekDesc,
    DayOfWeeknumeric,
    DayOfWeekDesc,
    FiscalYearID,
    FiscalYearDesc,
    FiscalQtrID,
    FiscalQtrDesc,
    HolidayFlag
)
```

Listing B.1: DimDate.ctl

DimTime

```
LOAD DATA
INTO TABLE DimTime
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    SK_TimeID,
    TimeValue DATE "HH24:MI:SS",
    HourID,
```

```
    HourDesc,
    MinuteID,
    MinuteDesc,
    SecondID,
    SecondDesc,
    MarketHoursFlag,
    OfficeHoursFlag
)
```

Listing B.2: DimTime.ctl

Industry

See 2.2.2.

```
LOAD DATA
INFILE
TRUNCATE
INTO TABLE Industry
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    IN_ID,
    IN_NAME,
    IN_SC_ID
)
```

Listing B.3: Industry.ctl

StatusType

See 2.2.2.

```
LOAD DATA
INFILE
TRUNCATE
INTO TABLE StatusType
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    ST_ID,
  ST_NAME
)
```

Listing B.4: StatusType.ctl

TaxRate

See 2.2.2.

```
LOAD DATA
INFILE
TRUNCATE
INTO TABLE TaxRate
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    TX_ID,
  TX_NAME,
  TX_RATE
)
```

Listing B.5: TaxRate.ctl

TradeType

See 2.2.2.

```
LOAD DATA
INFILE
TRUNCATE
INTO TABLE TradeType
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    TT_ID,
  TT_NAME,
  TT_IS_SELL,
  TT_IS_MRKT
)
```

Listing B.6: TradeType.ctl

Broker

See 2.2.2.

```
LOAD DATA
INTO TABLE S_Broker
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    EmployeeID,
    ManagerID,
    EmployeeFirstName,
    EmployeeLastName,
    EmployeeMI,
    EmployeeJobCode,
    EmployeeBranch,
    EmployeeOffice,
    EmployeePhone
)
```

Listing B.7: Broker.ctl

Prospect

See 2.2.2.

```
LOAD DATA
INTO TABLE S_Prospect
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    AGENCY_ID,
    LAST_NAME,
    FIRST_NAME,
    MIDDLE_INITIAL,
    GENDER,
    ADDRESS_LINE_1,
    ADDRESS_LINE_2,
    POSTAL_CODE,
    CITY,
    STATE,
    COUNTRY,
    PHONE,
    INCOME,
```

```
    NUMBER_CARS,
    NUMBER_CHILDREM,
    MARITAL_STATUS,
    AGE,
    CREDIT_RATING,
    OWN_OR_RENT_FLAG,
    EMPLOYER,
    NUMBER_CREDIT_CARDS,
    NET_WORTH
)
```

Listing B.8: Prospect.ctl

DimTrade

See 2.2.2.

```
LOAD DATA
INTO TABLE S_Trade
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    t_id,
    t_dts DATE "YYYY-MM-DD HH24:MI:SS",
    t_st_id,
    t_tt_id,
    t_is_cash,
    t_s_symb,
    t_qty,
    t_bid_price,
    t_ca_id,
    t_exec_name,
    t_trade_price,
    t_chrg,
    t_comm,
    t_tax
)
```

Listing B.9: Trade.ctl

FactCashBalances

See 2.2.2.

```
LOAD DATA
INTO TABLE S_Cash_Balances
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    CT_CA_ID,
    CT_DTS DATE "YYYY-MM-DD HH24:MI:SS",
    CT_AMT,
    CT_NAME
)
```

Listing B.10: CashBalances.ctl

FactWatches

See 2.2.2.

```
LOAD DATA
INTO TABLE S_Watches
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    W_C_ID,
    W_S_SYMB,
    W_DTS DATE "YYYY-MM-DD HH24:MI:SS",
    W_ACTION
)
```

Listing B.11: Watches.ctl

FactHoldings

See 2.2.2.

```
LOAD DATA
INTO TABLE S_Holdings
FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(
    HH_H_T_ID,
    HH_T_ID,
    HH_BEFORE_QTY,
    HH_AFTER_QTY
)
```

Listing B.12: Holdings.ctl