TPC-DI Benchmark Implementation of Pentaho Data Integration (Kettle)

INFO-H-419 Data Warehousing

Group Project

Akash Malhotra

Haonan Jin

Yalei Li

TABLE OF CONTENT

1. Abstract

With the rise in importance of Data and its growing size, Data Integration has become an indispensable part of the smooth functioning of any organization of importance. In this report, we introduce Kettle, a popular open-source tool designed for this task. We evaluated Kettle and showed its working through the TPC-DI dataset. We presented our results and discussed the experience of working with this tool. Overall, our experience was positive and we would encourage the reader to try Kettle for his/her future Data Integration Projects.

2. Data Integration

Data Integration, formerly known as ETL (Extract Load Transform), is a process that is commonly used to combine many data sources and load them into a database. Usually, data is scattered across many files and sources. In the age of the internet, even more so, as evidenced by billions of web pages all over the internet, with dozens of encoding schemes being used. An organization may need to extract the relevant data from multiple sources both offline, and online as this data helps the organization become smarter and run smoothly. Besides this, they often need to migrate data from one system to another, for various reasons. Technology stacks change very fast and also needs for increasing data storage and management should be met in a fast and reliable way. In light of all this, an organization may need to perform ETL on the available data sources. There are many efforts to parse the data, both offline and online, to extract relevant information. Therefore, there are many good Data Integration tools in the market. For this project, we use Pentaho Data Integration, formerly known as Kettle to perform Data Integration on the TPC-DI Dataset.

3. TPC-DI

The **Transaction Processing Performance Council** or TPC is a non-profit organization that promotes civilized competition between data tools vendors. It often defines benchmarks to test the available transactional processing tools in the market. TPC-DI is an artificially generated dataset by the scripts carefully designed by TPC, which enforces certain standards on the data integration tools, which includes common functionalities that one may expect from a data integration tool. When starting a new project on Data-Integration, an organization often faces the question of which tool to use for the job. This decision is difficult owing to so many tools available in the market. Furthermore, different tools may be optimized for different platforms. To solve this dilemma, one can leverage the TPC-DI dataset to test and evaluate various available tools and select the best one for the job.

Figure 3.1 shows the data model of the TPC-DI benchmark. It represents a typical retail brokerage firm. The whole process is divided into data sources, the staging area which contains the files extracted from the data sources, the transformations which are applied to the data from the files, and the final loading of this transformed data to a data warehouse. The TPC-DI doesn't test the process of extracting files from the data sources to the staging area. Instead, it automatically generates these files using its generative scripts. The extraction into the Data Integration tool (Kettle, in our case), is then tested, followed by transformations and loading in the data warehouse.

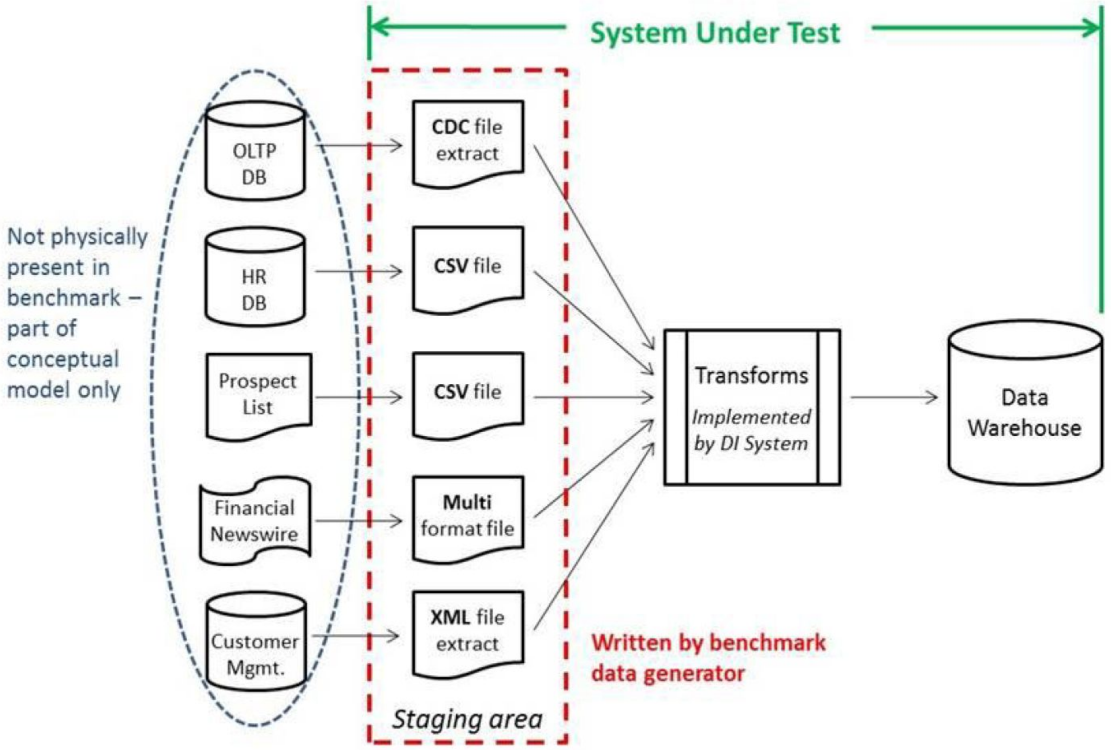
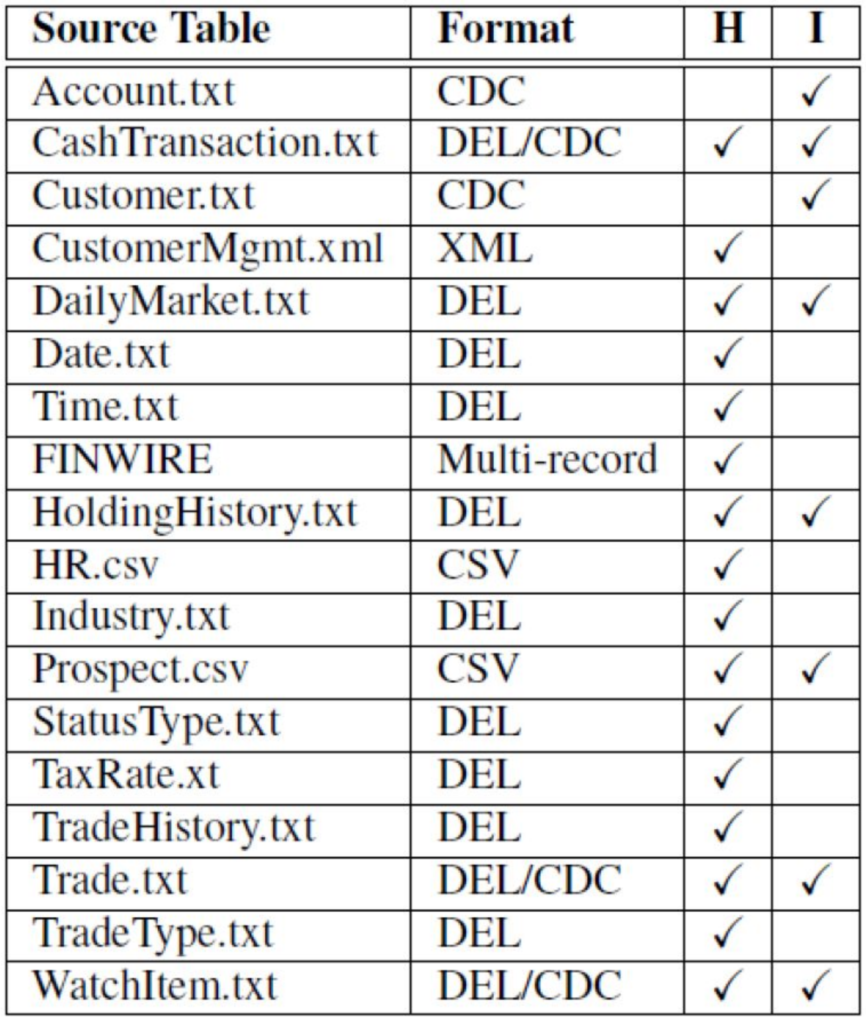


Figure 3.1 TPC-DI Data Mode

The term "transformations" in this benchmark includes everything that must be done to prepare and load data into the Data Warehouse. This can include:

- Conversion of data from character representations to data types compatible with the Data Warehouse specification.
- Lookups of business keys to obtain surrogate keys for the Data Warehouse.
- Merging or formatting multiple fields into one, or splitting one field into multiple.
- Checking data for errors or for adherence to business rules.
- Detecting changes in dimension data, and applying appropriate tracking mechanisms.
- (retaining history or overwriting).
- Detecting changes in fact data, and journaling updates to reflect current state.[1]

The generator script, based on Parallel Data Generation Framework (PDGF)[2], generates various files to emulate the real-world scenario of a typical retail brokerage firm. As can be seen from Figure 3.2 below, the files are quite diverse and consist of file extensions like CDC, DEL, XML, etc. Some of them pertain to historical load, and others pertain to incremental load. FINWIRE data consists of many files that represent the financial quarter years. We merged these files sequentially before applying the transformations.



Source Table	Format	H	I
Account.txt	CDC		✓
CashTransaction.txt	DEL/CDC	✓	✓
Customer.txt	CDC		✓
CustomerMgmt.xml	XML	✓	
DailyMarket.txt	DEL	✓	✓
Date.txt	DEL	✓	
Time.txt	DEL	✓	
FINWIRE	Multi-record	✓	
HoldingHistory.txt	DEL	✓	✓
HR.csv	CSV	✓	
Industry.txt	DEL	✓	
Prospect.csv	CSV	✓	✓
StatusType.txt	DEL	✓	
TaxRate.xt	DEL	✓	
TradeHistory.txt	DEL	✓	
Trade.txt	DEL/CDC	✓	✓
TradeType.txt	DEL	✓	
WatchItem.txt	DEL/CDC	✓	✓

Figure 3.2. Source Data Summary

Figure 3.3 shows the schema of the target data warehouse. Although the schema is defined, the data scale can be arbitrary depending on the user. For this report, we generated 3 GB of data.

The target schema has 6 Fact tables, 5 reference tables, 2 operational tables, and 8-dimensional tables, the strict sense of star schema. They are all interconnected using foreign keys.

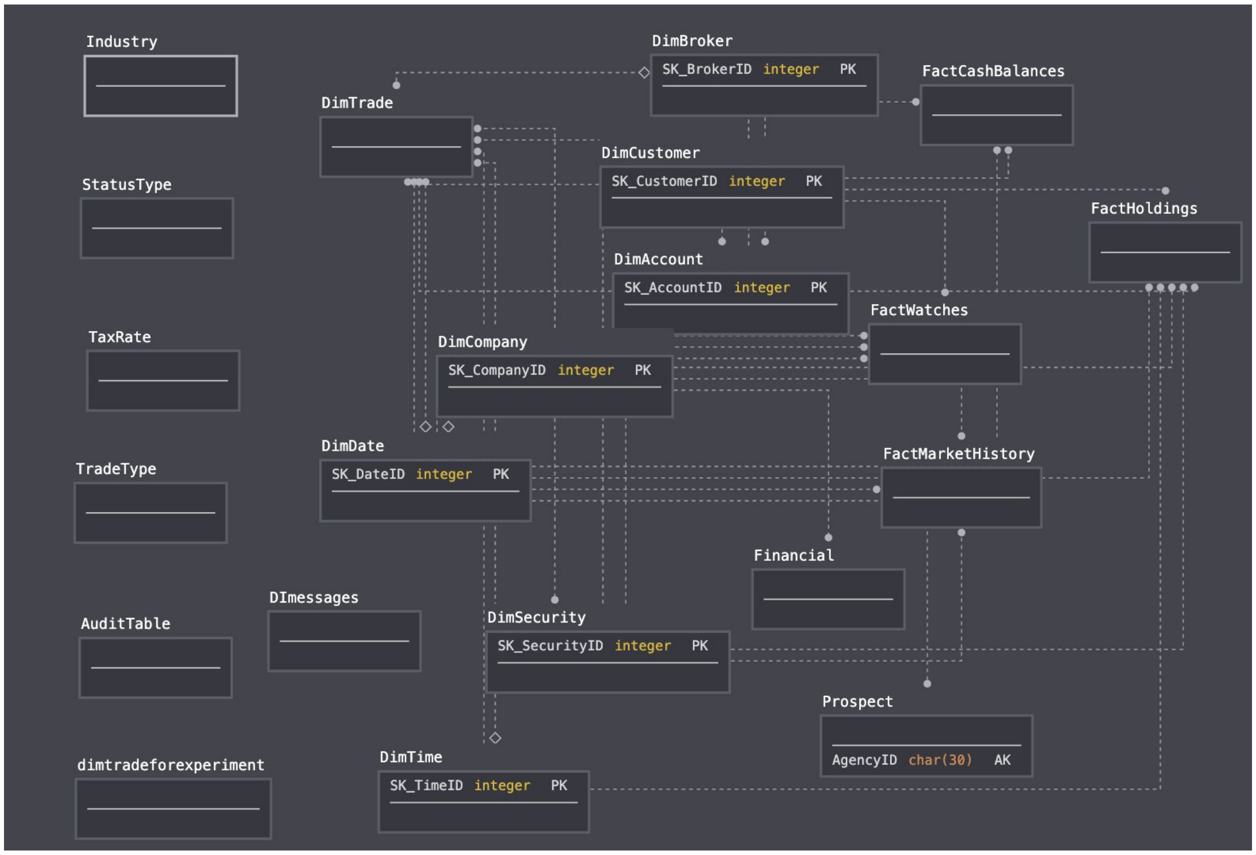


Figure 3.3 Diagrammatic view of Data warehouse schema

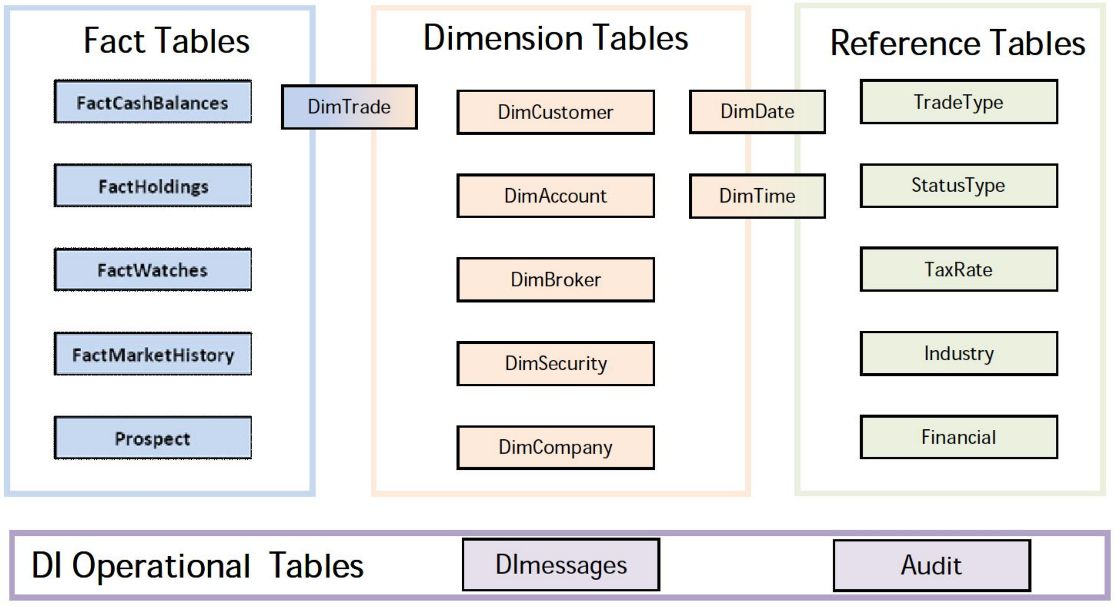


Figure 3.4 Pictorial overview of Data warehouse schema

In many real-world DI applications, there are two variants of the DI process. One variant performs a historical load, at times when the data warehouse is initially created or when it is recreated from historical records such as when the data warehouse schema is restructured.

The second variant performs incremental updates, representing a load of new data into an existing data warehouse. There are many different rates at which incremental updates may occur, from rarely to near real-time. Daily updates are common and are the model for the TPC-DI benchmark.[1]

4. Pentaho Data Integration (Kettle)

4.1. General Information

Kettle is the former name of Pentaho Data Integration (PDI) after its joining to the open-source business intelligence organization Pentaho in 2006. Kettle is one of the most popular ETL (Extract, Transform, Load) tools, which are often used to build data warehouses, but not limited to this field. Using ETL tools can complete data extraction from the target data source, and after a series of data transformation, the required data model is formed and can be loaded into the data warehouse. Besides, it is an open-source ETL tool developed and implemented using the Java language. It provides a wide collection of components for data extraction, transformation, and loading.

Kettle can be mainly used in the following scenarios: (1) Integrate data between different applications or databases; (2) Export the data from the database to a formatted file; (3) Load large quantities of data into the database; (4) Data cleaning; (5) Integrated application related projects are for use [4].

Kettle is a practical tool and accessible with great capability. It does not need to write code to implement any functions which can be simply achieved through graphic interface design. Kettle is designed based on metadata. Needless to say, it supports a broad range of input and output formats, including text files, data tables, and commercial/open-source database engines. In addition, Kettle's powerful conversion function is very convenient for manipulating data. Its two core components are Transformation and Job: (1) Transformation is configured to complete the data ETL work and; (2) Job defines a set of processes to complete the entire workflow.

4.2. PDI Structure

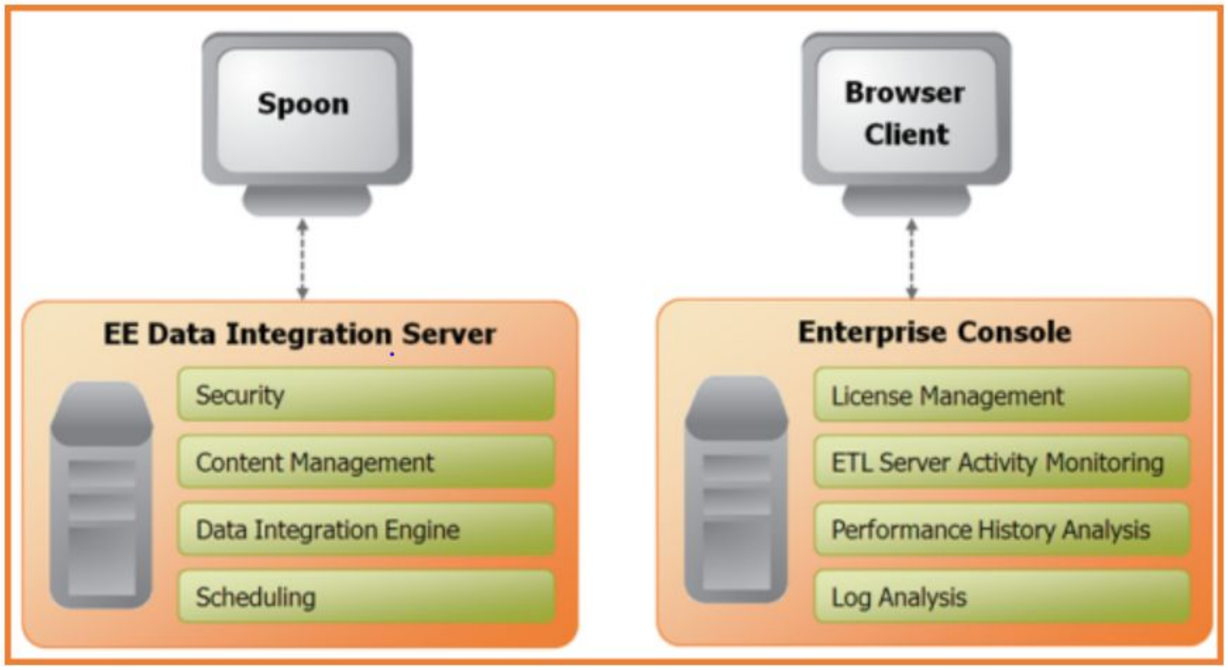


Figure 4.1. PDI Core Components[3]

Spoon is a tool for building ETL Jobs and Transformations. In Spoon, users can graphically design the transformations in a drag-and-drop manner, and invoke a dedicated data integration engine or cluster. Data Integration Server is a dedicated ETL server. Its main functions are:

Function	Description
Implementation	Perform ETL jobs or transformations through the Pentaho Data Integration engine
Safety	Manage users, roles, or integrated security
Content Management	Provides a centralized repository for managing ETL jobs and transformations. The repository contains historical versions of all content and features.
Timing	Provides services for managing the timing and monitoring of activities on the Data Integration Server in the spoon designer environment

Table 4.1. Functions and Description[3]

Pentaho Enterprise Console provides a small client for managing Pentaho Data Integration Enterprise Edition deployment, including enterprise version certificate management, monitoring, and control of remote Pentaho Data Dynamic performance of activities on the Integration server, analysis of registered jobs and conversions [3].

4.3. PDI Components

PDI mainly consists of four parts: Spoon, Pan, Kitchen, and Carte. Their main features are described in Table 4.2. For this report, we take advantage of the Spoon to perform most of the benchmark testing procedures.

name	**Description**
Spoon	A graphical interface for desktop applications for editing jobs and conversions.
Pan	A standalone command-line program for performing transformations and jobs edited by Spoon.
Kitchen	A standalone command-line program for executing jobs edited by Spoon.
Carte	Carte is a lightweight web container for building a dedicated, remote ETL Server.

Table 4.2. Name and Description[3]

4.4. Related Terms & Basic Concepts

The execution procedure of Kettle can be divided into two levels: Job and Transformation (a detailed conceptual model is shown in Figure 4.2.). The main difference between the two levels is the way data is passed and run.

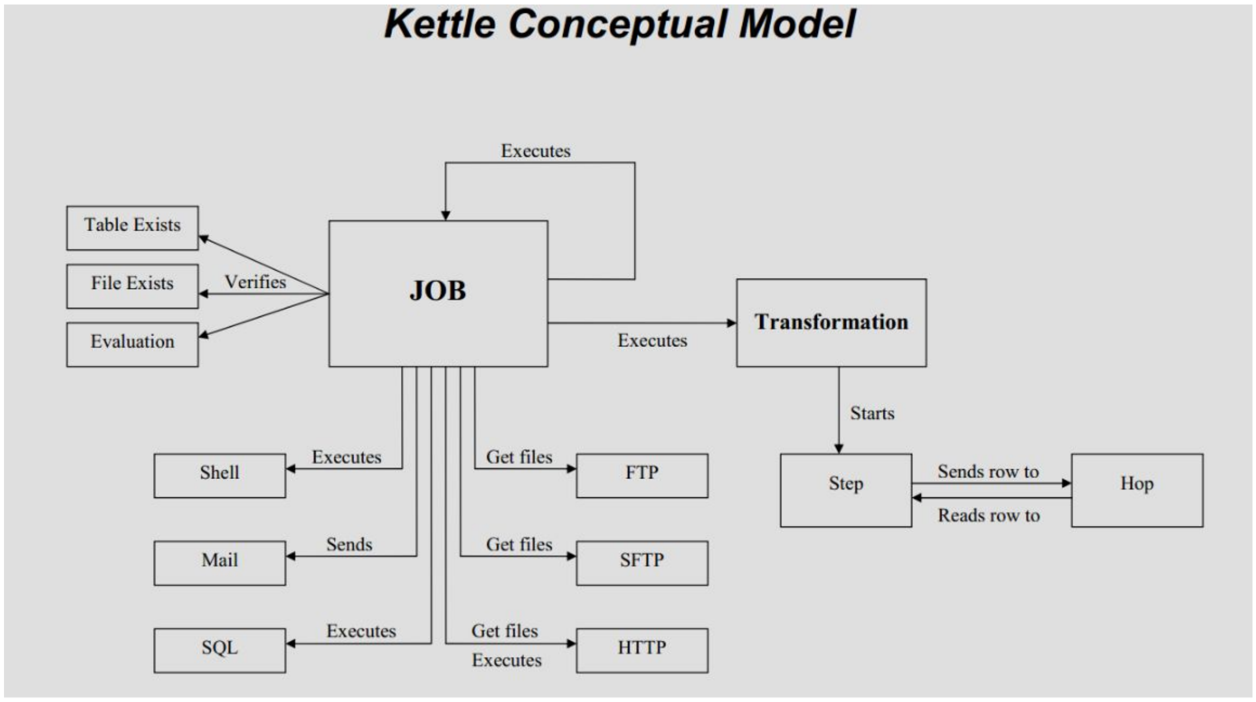


Figure 4.2. Conceptual model of PDI [3]

4.4.1. Step

Steps are the smallest operable components of Kettle and the building blocks of the transformation. For example, a text file input or a table output is a step. There are more than 140 step models in PDI, and are classified according to different functions, such as input class, output class, script class, etc. Each step is used to complete a specific function. By configuring and combining a set of steps, users can complete the task you need to perform.

In this report, it is worth mentioning that the Merge and Sort steps are extremely expensive to perform during the whole experiment. For instance, Figure 4.3 is the beginning of the DimTrade ETL process, which features a series of sorting and merging operations. During the actual execution, the CPU and memory usage surged to an extreme and resulted in several software failures (details can be found in Figure 4.4 and 4.5). We tuned down the configuration of the loading size in memory for each merge and join, and eventually managed to conduct the whole execution.

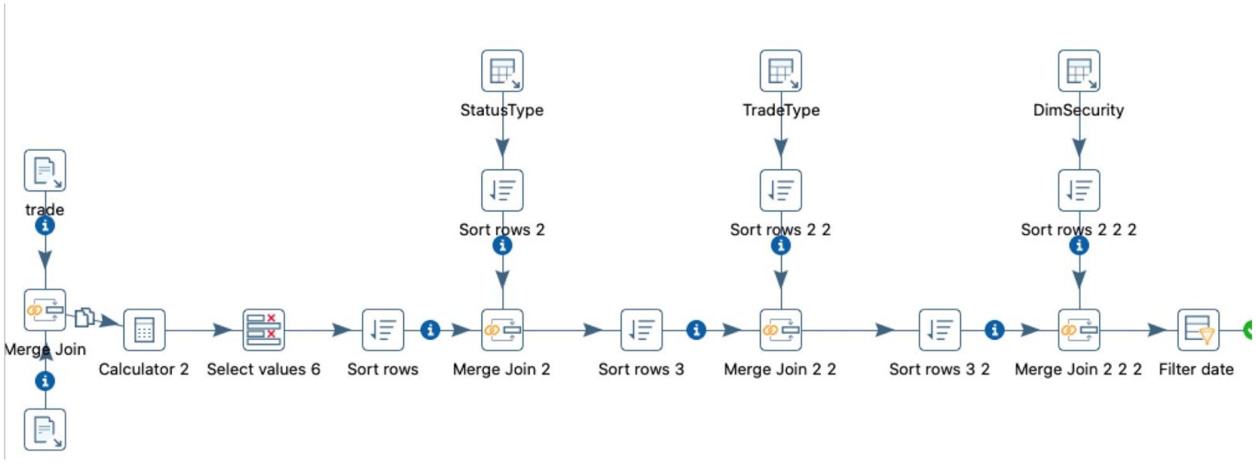
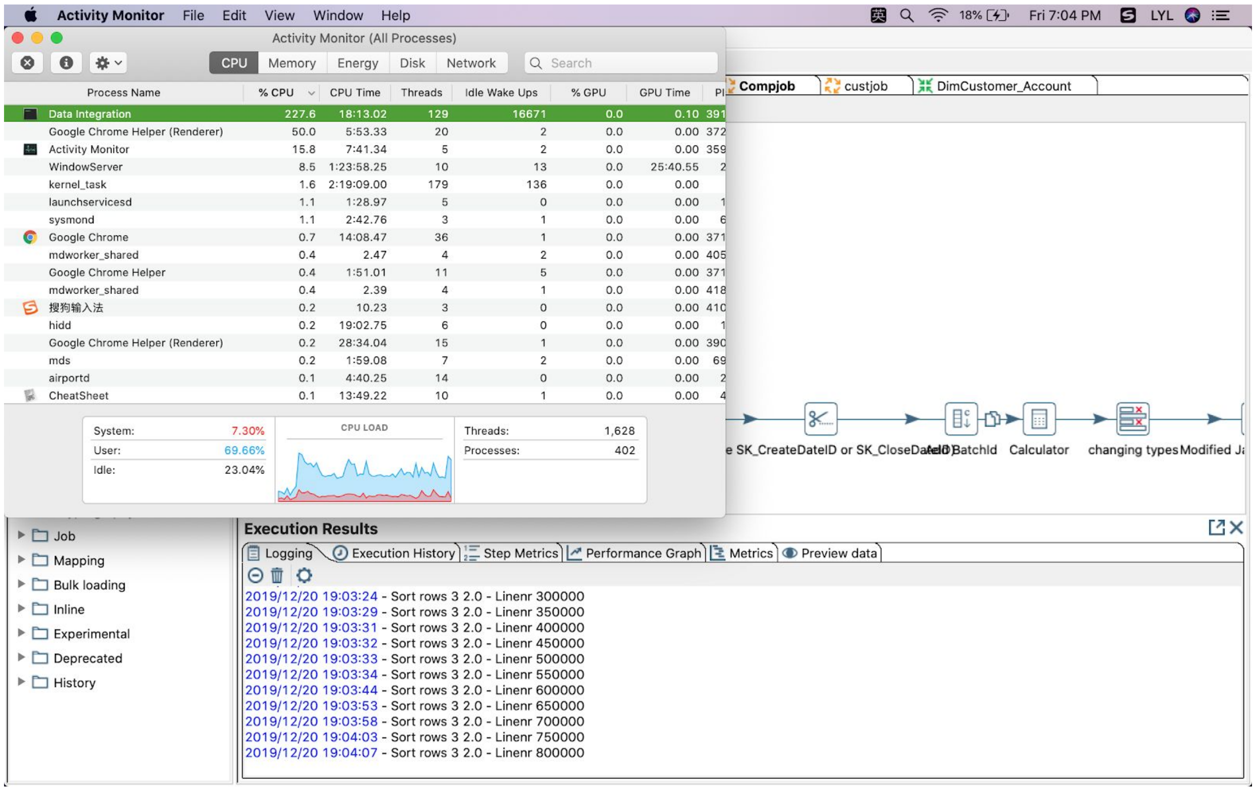


Figure 4.3. Partial Demonstration of DimTrade ETL Process


Activity Monitor File Edit View Window Help
Activity Monitor (All Processes)
CPU Memory Energy Disk Network Search
Compjob custjob DimCustomer_Account
SK_CreateDateID or SK_CloseDateID BatchId Calculator changing types Modified J
Job
Mapping
Bulk loading
Inline
Experimental
Deprecated
History
Execution Results
Logging Execution History Step Metrics Performance Graph Metrics Preview data
CPU LOAD

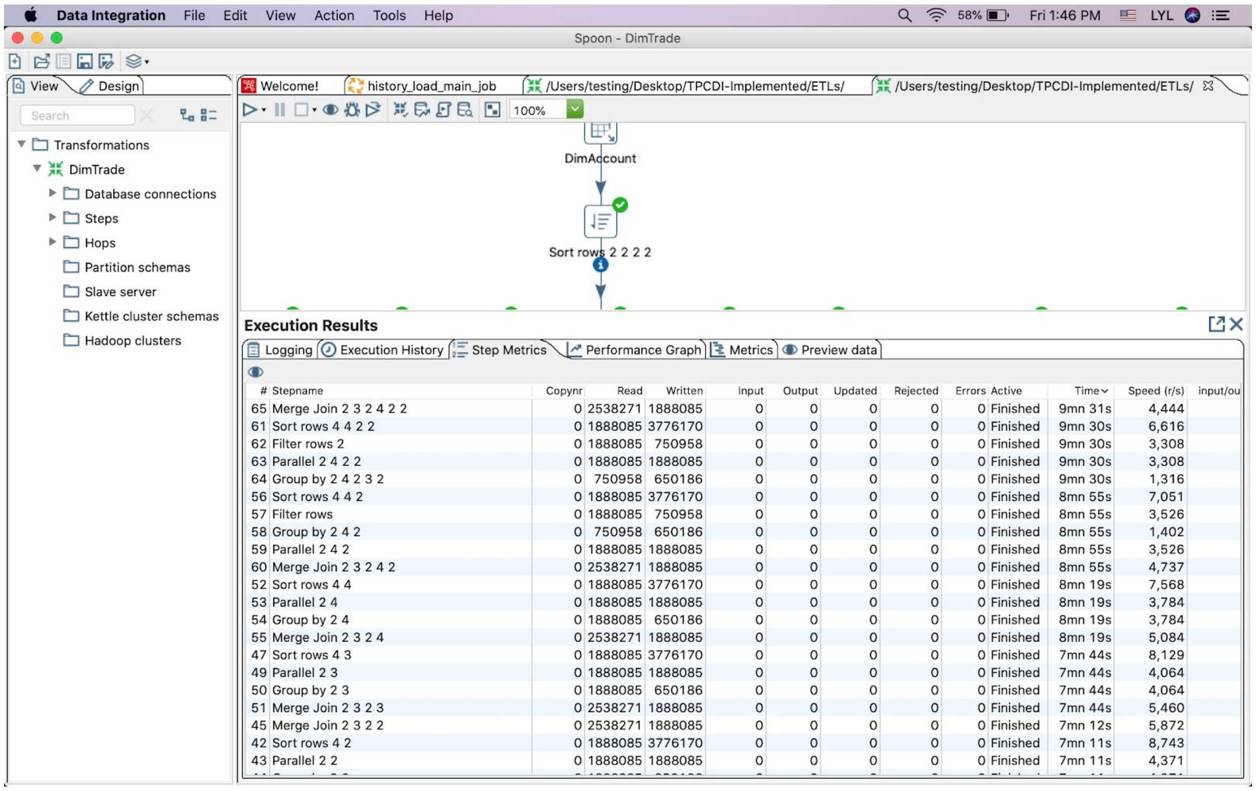
Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	% GPU	GPU Time	PI
Data Integration	227.6	18:13.02	129	16671	0.0	0.10	391
Google Chrome Helper (Renderer)	50.0	5:53.33	20	2	0.0	0.00	372
Activity Monitor	15.8	7:41.34	5	2	0.0	0.00	359
WindowServer	8.5	1:23:58.25	10	13	0.0	25:40.55	2
kernel_task	1.6	2:19:09.00	179	136	0.0	0.00	
launchservicesd	1.1	1:28.97	5	0	0.0	0.00	1
sysmond	1.1	2:42.76	3	1	0.0	0.00	6
Google Chrome	0.7	14:08.47	36	1	0.0	0.00	371
mdworker_shared	0.4	2.47	4	2	0.0	0.00	405
Google Chrome Helper	0.4	1:51.01	11	5	0.0	0.00	371
mdworker_shared	0.4	2.39	4	1	0.0	0.00	418
搜狗输入法	0.2	10.23	3	0	0.0	0.00	410
hidd	0.2	19:02.75	6	0	0.0	0.00	1
Google Chrome Helper (Renderer)	0.2	28:34.04	15	1	0.0	0.00	390
mds	0.2	1:59.08	7	2	0.0	0.00	69
airportd	0.1	4:40.25	14	0	0.0	0.00	2
CheatSheet	0.1	13:49.22	10	1	0.0	0.00	4

System:	7.30%
User:	69.66%
Idle:	23.04%

Threads:	1,628
Processes:	402

```
2019/12/20 19:03:24 - Sort rows 3 2.0 - Linenr 300000
2019/12/20 19:03:29 - Sort rows 3 2.0 - Linenr 350000
2019/12/20 19:03:31 - Sort rows 3 2.0 - Linenr 400000
2019/12/20 19:03:32 - Sort rows 3 2.0 - Linenr 450000
2019/12/20 19:03:33 - Sort rows 3 2.0 - Linenr 500000
2019/12/20 19:03:34 - Sort rows 3 2.0 - Linenr 550000
2019/12/20 19:03:44 - Sort rows 3 2.0 - Linenr 600000
2019/12/20 19:03:53 - Sort rows 3 2.0 - Linenr 650000
2019/12/20 19:03:58 - Sort rows 3 2.0 - Linenr 700000
2019/12/20 19:04:03 - Sort rows 3 2.0 - Linenr 750000
2019/12/20 19:04:07 - Sort rows 3 2.0 - Linenr 800000
```

Figure 4.4. Real-time CPU Usage for Sorting



#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)	input/ou
65	Merge Join 2 3 2 4 2 2	0	2538271	1888085	0	0	0	0	0	Finished	9mn 31s	4,444	
61	Sort rows 4 4 2 2	0	1888085	3776170	0	0	0	0	0	Finished	9mn 30s	6,616	
62	Filter rows 2	0	1888085	750958	0	0	0	0	0	Finished	9mn 30s	3,308	
63	Parallel 2 4 2 2	0	1888085	1888085	0	0	0	0	0	Finished	9mn 30s	3,308	
64	Group by 2 4 2 3 2	0	750958	650186	0	0	0	0	0	Finished	9mn 30s	1,316	
56	Sort rows 4 4 2	0	1888085	3776170	0	0	0	0	0	Finished	8mn 55s	7,051	
57	Filter rows	0	1888085	750958	0	0	0	0	0	Finished	8mn 55s	3,526	
58	Group by 2 4 2	0	750958	650186	0	0	0	0	0	Finished	8mn 55s	1,402	
59	Parallel 2 4 2	0	1888085	1888085	0	0	0	0	0	Finished	8mn 55s	3,526	
60	Merge Join 2 3 2 4 2	0	2538271	1888085	0	0	0	0	0	Finished	8mn 55s	4,737	
52	Sort rows 4 4	0	1888085	3776170	0	0	0	0	0	Finished	8mn 19s	7,568	
53	Parallel 2 4	0	1888085	1888085	0	0	0	0	0	Finished	8mn 19s	3,784	
54	Group by 2 4	0	1888085	650186	0	0	0	0	0	Finished	8mn 19s	3,784	
55	Merge Join 2 3 2 4	0	2538271	1888085	0	0	0	0	0	Finished	8mn 19s	5,084	
47	Sort rows 4 3	0	1888085	3776170	0	0	0	0	0	Finished	7mn 44s	8,129	
49	Parallel 2 3	0	1888085	1888085	0	0	0	0	0	Finished	7mn 44s	4,064	
50	Group by 2 3	0	1888085	650186	0	0	0	0	0	Finished	7mn 44s	4,064	
51	Merge Join 2 3 2 3	0	2538271	1888085	0	0	0	0	0	Finished	7mn 44s	5,460	
45	Merge Join 2 3 2 2	0	2538271	1888085	0	0	0	0	0	Finished	7mn 12s	5,872	
42	Sort rows 4 2	0	1888085	3776170	0	0	0	0	0	Finished	7mn 11s	8,743	
43	Parallel 2 2	0	1888085	1888085	0	0	0	0	0	Finished	7mn 11s	4,371	

Figure 4.5. Overall Merge & Sort Running Time

4.4.2. Hop

Hops (node connection) are channels of data that connect two steps and determine the direction of data flows so that metadata is passed from one step to another. In the job sample in figure 4.6, the connected steps seem to happen sequentially, but that is not the case. In practice, the connection between steps merely determines the incoming and outcoming data flow, while each step within the transformation follows its own execution thread and continuously receives and pushes data through hops.

Note: All steps are started and run synchronously, so the order of step initialization is unknown. Because we cannot set a variable in the first step and then use it in the next step.

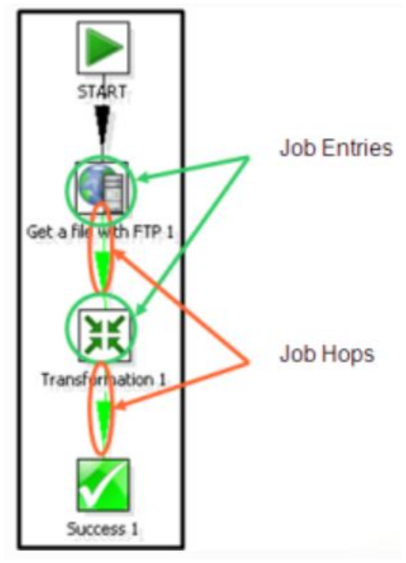


Figure 4.6. Example of a Kettle Job

A step can have multiple hops within a transformation, where the data flow is directed from one step to multiple steps. If data from one step is output to multiple steps, the data can be either copied or distributed [3].

In Spoon, a hop connects one transformation step or job entry to another. The direction of the data flow is indicated with an arrow on the graphical view pane. A hop can be enabled or disabled (for testing purposes for example). Figure 4.7 shows a hop of our testing TPC-DI dataset, which is between *Prospect 2* and *Incremental Updates DimCustomer*. Table 4.3 shows the description of various colors of hops.

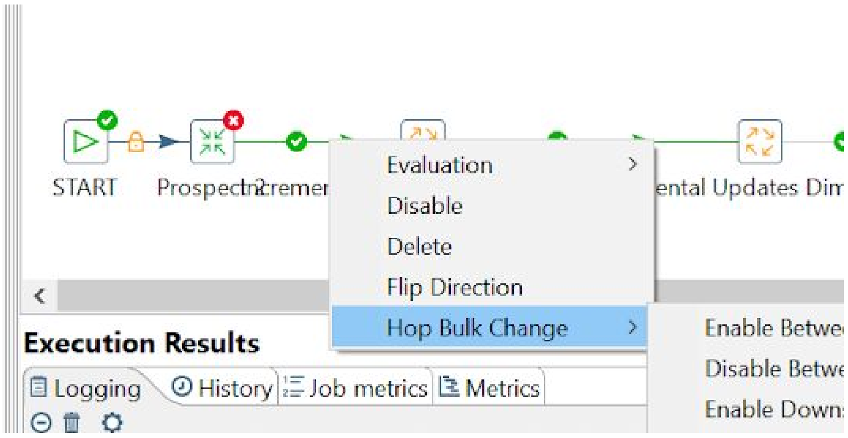


Figure 4.7. Hop of TPC-DI dataset

Hop color	Meaning

Yellow	The hop is used for carrying rows that caused errors in source step(s).
Red (Bold Dot line)	The hop is never used because no data will ever go there.
Red	The hop is disabled.
Orange (Dot line)	The hop has a named target step.
Magenta	Provides info for step, distributes rows
Green	Provides info for step, copies rows
Gray	Distribute rows: if multiple hops are leaving a step, rows of data will be evenly distributed to all target steps.
Blue	Copies rows: if multiple hops are leaving a step, all rows of data will be copied to all target steps.
Black	Candidate hop using middle button + drag

Table 4.3 Explanation of Hop ColourS[3]

4.4.3. Transformation

Transformation is a network of steps. The two main components of transformation are steps and hops (node connection). The extension of the converted file is .ktr. A transformation is essentially a data stream. Figure 4.8 is an example of a transformation that reads data from a text file, filters it, then sorts it, and finally loads the data into the database. In essence, transformation is a logical structure of a set of graphical data transformation configurations.[3]

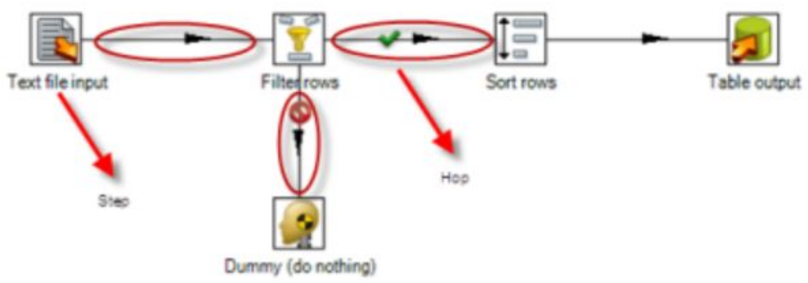


Figure 4.8 Example of Transformation

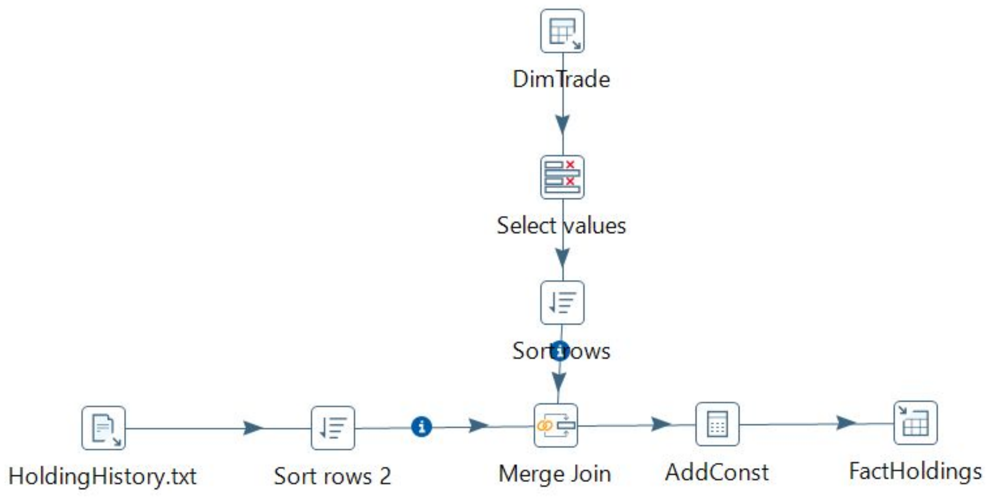


Figure 4.9 Transformation of TPC-DI FatcHoldingUpdate

Taking our experiment as an example, Figure 4.9 illustrates that FachHoldingUpdate transformation, with the flow of steps such as HoldingHistory.txt, Sort rows 2 and Select Value. To begin with, there are two parallel paths, path 1 is to get Holdinghistory dataset and sort rows; Meanwhile, path 2 is to implement the DimTrade, select vaLues, and sort rows. Next, based on the sort results, Merge Join can be implemented. Finally, addConst will be implemented and FachHoldings is to be updated.

4.4.4. Job

Jobs are modelled ETL activities based on the workflow, serving for coordinating data sources, execution processes, and related dependencies. A job combines functionality and physical processes together. It consists of node connections, entities, and settings. The job file extension is *.kjb*.

As shown in Figure 4.6, the typical job example includes obtaining files from FTP, checking the existence of a database table, performing a conversion, and sending an email to notify a conversion error. The result of the final work could be updating the data warehouse.

In this report, there are three major jobs: *history_load_main_job*, *Inc1_Load_Main_Job,* and *Inc2_Load_Main_Job* corresponded to Batch 1, Batch 2 and Batch 3 load respectively under TPC-DI schema. Figure 4.10 illustrates the detailed procedure of historical load*,* which has three various paths that will be implemented based on the figure schema. It includes jobs of lower levels and transformation units.

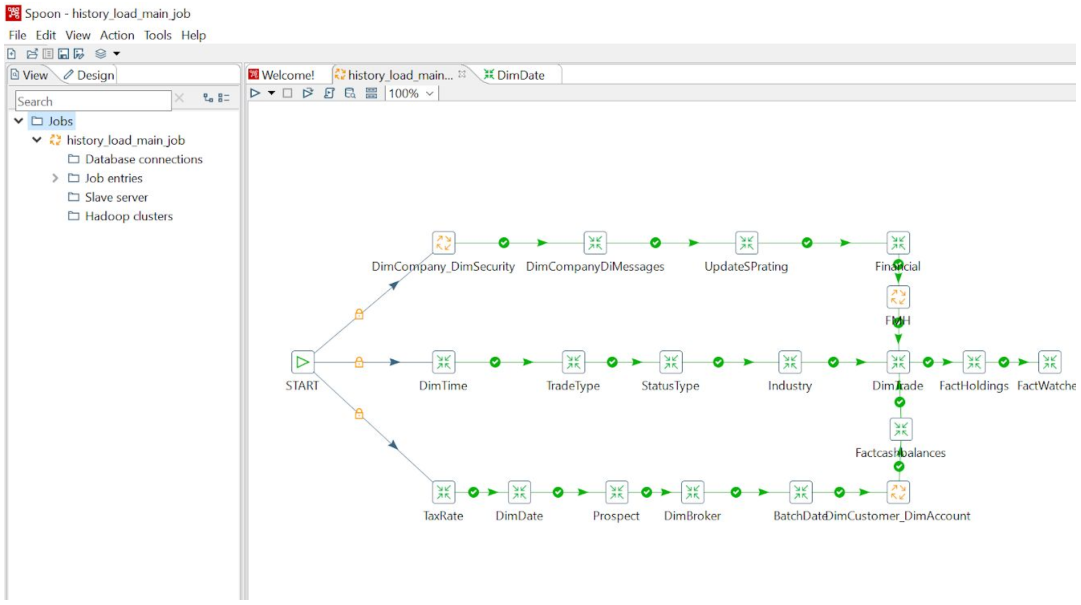


Figure 4.10 Designed Job of the Historical Load

4.5. Variable

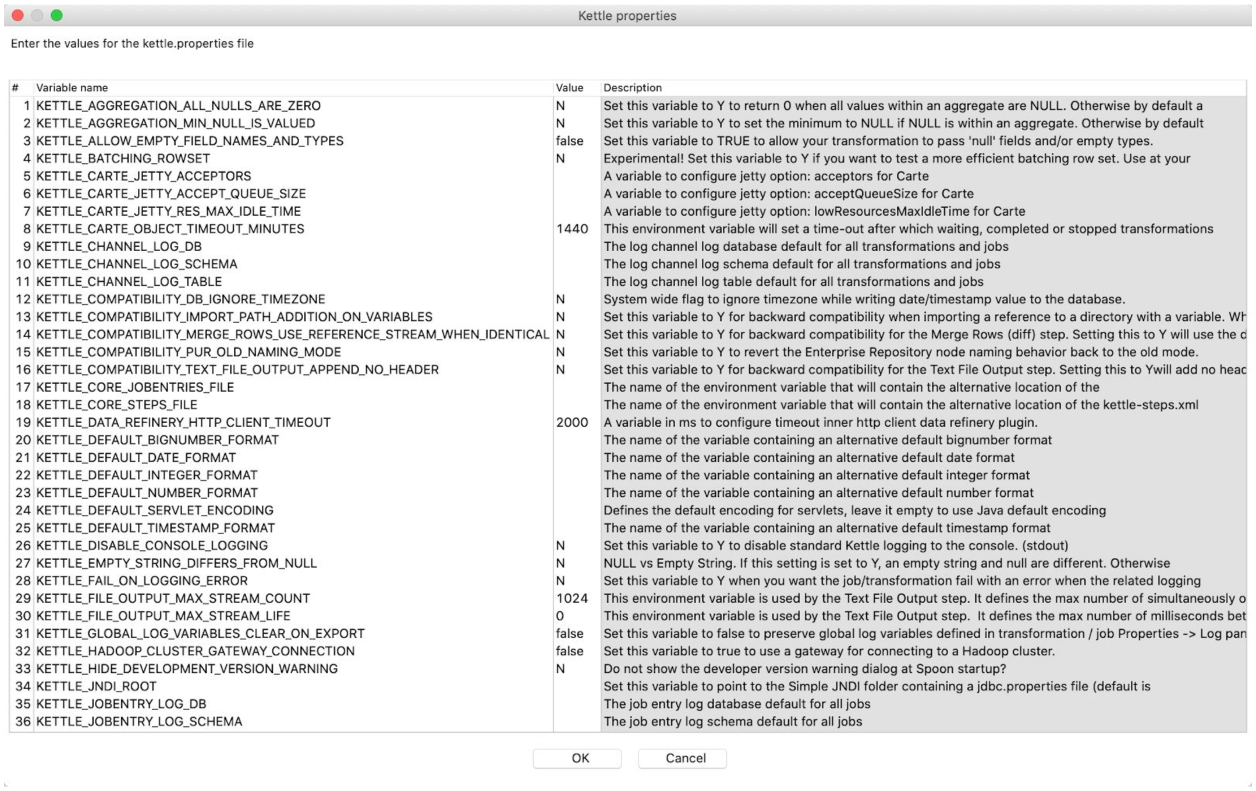
According to the scope, variables are divided into two categories: environment variables and kettle variables [5].

4.5.1. Environment Variables

Environment variables can be defined by the "set environment variable" dialogue box under the edit menu. The problem when using environment variables may be that they cannot be used dynamically. Conflicts may occur if two or more conversions using the same environment variable are performed in the same application server. Environment variables are visible in all applications using *JVM*. In this study, no environment variable is explicitly used.

4.5.2. Kettle variables

Kettle variables are used to store small amounts of information in a small dynamic range. Kettle variables are local to the kettle, and the scope can be a job or a transition where variables can be set or modified. Setting variables are used to configure the related tasks and their scopes, such as parent work, grandfather work, or rootwork. No specified variables are set for this benchmark testing, and our full kettle property is shown in Figure 4.11.



Kettle properties

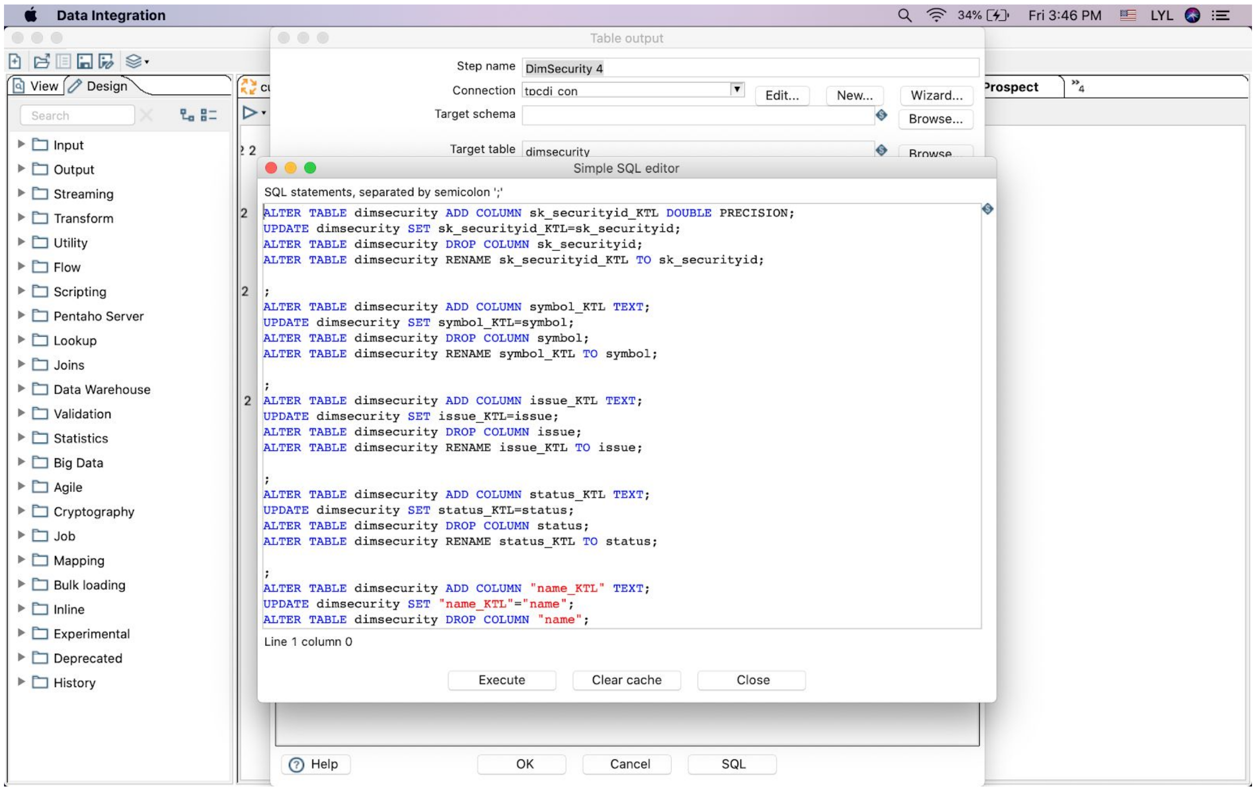
Enter the values for the kettle.properties file

#	Variable name	Value	Description
1	KETTLE_AGGREGATION_ALL_NULLS_ARE_ZERO	N	Set this variable to Y to return 0 when all values within an aggregate are NULL. Otherwise by default a
2	KETTLE_AGGREGATION_MIN_NULL_IS_VALUED	N	Set this variable to Y to set the minimum to NULL if NULL is within an aggregate. Otherwise by default
3	KETTLE_ALLOW_EMPTY_FIELD_NAMES_AND_TYPES	false	Set this variable to TRUE to allow your transformation to pass 'null' fields and/or empty types.
4	KETTLE_BATCHING_ROWSET	N	Experimental! Set this variable to Y if you want to test a more efficient batching row set. Use at your
5	KETTLE_CARTE_JETTY_ACCEPTORS		A variable to configure jetty option: acceptors for Carte
6	KETTLE_CARTE_JETTY_ACCEPT_QUEUE_SIZE		A variable to configure jetty option: acceptQueueSize for Carte
7	KETTLE_CARTE_JETTY_RES_MAX_IDLE_TIME		A variable to configure jetty option: lowResourcesMaxIdleTime for Carte
8	KETTLE_CARTE_OBJECT_TIMEOUT_MINUTES	1440	This environment variable will set a time-out after which waiting, completed or stopped transformations
9	KETTLE_CHANNEL_LOG_DB		The log channel log database default for all transformations and jobs
10	KETTLE_CHANNEL_LOG_SCHEMA		The log channel log schema default for all transformations and jobs
11	KETTLE_CHANNEL_LOG_TABLE		The log channel log table default for all transformations and jobs
12	KETTLE_COMPATIBILITY_DB_IGNORE_TIMEZONE	N	System wide flag to ignore timezone while writing date/timestamp value to the database.
13	KETTLE_COMPATIBILITY_IMPORT_PATH_ADDITION_ON_VARIABLES	N	Set this variable to Y for backward compatibility when importing a reference to a directory with a variable. Wh
14	KETTLE_COMPATIBILITY_MERGE_ROWS_USE_REFERENCE_STREAM_WHEN_IDENTICAL	N	Set this variable to Y for backward compatibility for the Merge Rows (diff) step. Setting this to Y will use the d
15	KETTLE_COMPATIBILITY_PUR_OLD_NAMING_MODE	N	Set this variable to Y to revert the Enterprise Repository node naming behavior back to the old mode.
16	KETTLE_COMPATIBILITY_TEXT_FILE_OUTPUT_APPEND_NO_HEADER	N	Set this variable to Y for backward compatibility for the Text File Output step. Setting this to Ywill add no heac
17	KETTLE_CORE_JOBENTRIES_FILE		The name of the environment variable that will contain the alternative location of the
18	KETTLE_CORE_STEPS_FILE		The name of the environment variable that will contain the alternative location of the kettle-steps.xml
19	KETTLE_DATA_REFINERY_HTTP_CLIENT_TIMEOUT	2000	A variable in ms to configure timeout inner http client data refinery plugin.
20	KETTLE_DEFAULT_BIGNUMBER_FORMAT		The name of the variable containing an alternative default bignumber format
21	KETTLE_DEFAULT_DATE_FORMAT		The name of the variable containing an alternative default date format
22	KETTLE_DEFAULT_INTEGER_FORMAT		The name of the variable containing an alternative default integer format
23	KETTLE_DEFAULT_NUMBER_FORMAT		The name of the variable containing an alternative default number format
24	KETTLE_DEFAULT_SERVLET_ENCODING		Defines the default encoding for servlets, leave it empty to use Java default encoding
25	KETTLE_DEFAULT_TIMESTAMP_FORMAT		The name of the variable containing an alternative default timestamp format
26	KETTLE_DISABLE_CONSOLE_LOGGING	N	Set this variable to Y to disable standard Kettle logging to the console. (stdout)
27	KETTLE_EMPTY_STRING_DIFFERS_FROM_NULL	N	NULL vs Empty String. If this setting is set to Y, an empty string and null are different. Otherwise
28	KETTLE_FAIL_ON_LOGGING_ERROR	N	Set this variable to Y when you want the job/transformation fail with an error when the related logging
29	KETTLE_FILE_OUTPUT_MAX_STREAM_COUNT	1024	This environment variable is used by the Text File Output step. It defines the max number of simultaneously o
30	KETTLE_FILE_OUTPUT_MAX_STREAM_LIFE	0	This environment variable is used by the Text File Output step. It defines the max number of milliseconds bet
31	KETTLE_GLOBAL_LOG_VARIABLES_CLEAR_ON_EXPORT	false	Set this variable to false to preserve global log variables defined in transformation / job Properties -> Log pan
32	KETTLE_HADOOP_CLUSTER_GATEWAY_CONNECTION	false	Set this variable to true to use a gateway for connecting to a Hadoop cluster.
33	KETTLE_HIDE_DEVELOPMENT_VERSION_WARNING	N	Do not show the developer version warning dialog at Spoon startup?
34	KETTLE_JNDI_ROOT		Set this variable to point to the Simple JNDI folder containing a jdbc.properties file (default is
35	KETTLE_JOBENTRY_LOG_DB		The job entry log database default for all jobs
36	KETTLE_JOBENTRY_LOG_SCHEMA		The job entry log schema default for all jobs

Figure 4.11. Kettle Variable Setting for this Report

4.6. Simple SQL Editor and Data Explorer

The Simple SQL Editor (Figure 4.12) is a practical tool for users to execute standard SQL commands for tasks such as creating tables, dropping indexes and modifying fields. The SQL Editor is used to preview and execute DDL (Data Definition Language) generated by Spoon such as "create/alter table, "create index," and "create sequence" SQL commands. For example, if you add a Table Output step to a transformation and click the SQL button at the bottom of the Table Input dialog, Spoon automatically generates the necessary DDL for the output step to function properly and present that to the end-user through the SQL Editor [3].



Step name: DimSecurity 4
Connection: tpcdi con
Target schema:
Target table: dimsecurity

SQL statements, separated by semicolon ';'

```sql
ALTER TABLE dimsecurity ADD COLUMN sk_securityid_KTL DOUBLE PRECISION;
UPDATE dimsecurity SET sk_securityid_KTL=sk_securityid;
ALTER TABLE dimsecurity DROP COLUMN sk_securityid;
ALTER TABLE dimsecurity RENAME sk_securityid_KTL TO sk_securityid;

;
ALTER TABLE dimsecurity ADD COLUMN symbol_KTL TEXT;
UPDATE dimsecurity SET symbol_KTL=symbol;
ALTER TABLE dimsecurity DROP COLUMN symbol;
ALTER TABLE dimsecurity RENAME symbol_KTL TO symbol;

;
ALTER TABLE dimsecurity ADD COLUMN issue_KTL TEXT;
UPDATE dimsecurity SET issue_KTL=issue;
ALTER TABLE dimsecurity DROP COLUMN issue;
ALTER TABLE dimsecurity RENAME issue_KTL TO issue;

;
ALTER TABLE dimsecurity ADD COLUMN status_KTL TEXT;
UPDATE dimsecurity SET status_KTL=status;
ALTER TABLE dimsecurity DROP COLUMN status;
ALTER TABLE dimsecurity RENAME status_KTL TO status;

;
ALTER TABLE dimsecurity ADD COLUMN "name_KTL" TEXT;
UPDATE dimsecurity SET "name_KTL"="name";
ALTER TABLE dimsecurity DROP COLUMN "name";
```

Line 1 column 0

Figure 4.12 SQL Editor for DimSecurity Transformation

The Database Explorer provides the ability to explore configured database connections. The Database Explorer also supports tables, views, and synonyms along with the catalogue and/or schema to which the table belongs.

Table 4.4 illustrates the feature and description of the data explorer.

Feature	**Description**
Generate DDL	Generates the DDL to create the selected table based on the current connection type
Generate DDL for other connection	Prompts the user for another connection, then generates the DDL to create the selected table based on the user-selected connection type.
Number of rows	Specifies the three-digit client number for the connection

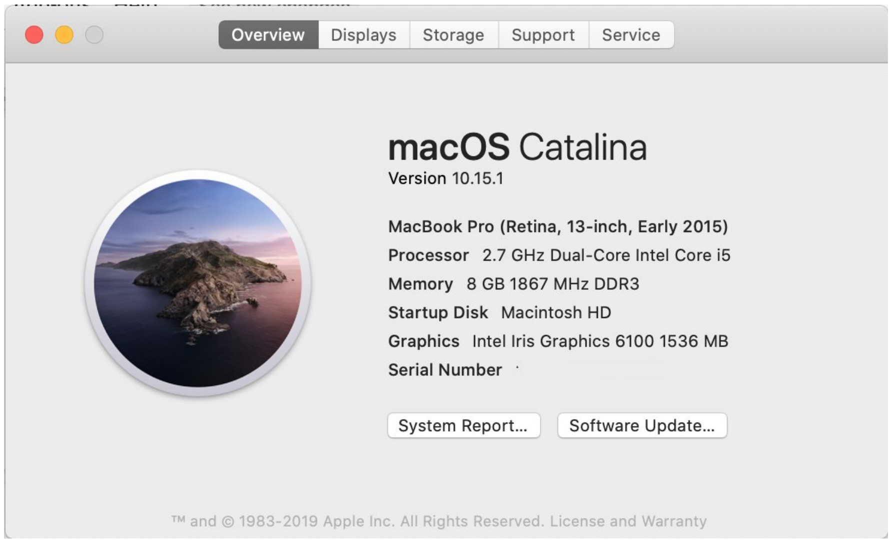
Open SQL for	Launches the Simple SQL Editor for the selected table
Preview the first 100 rows	Returns the first 100 rows from the selected table
Preview the first row	Prompts the user for the number of rows to return from the selected table
Show Layout	Displays a list of column names, data types, and so on from the selected table
Truncate table	Generates a TRUNCATE table statement for the current table.

Table 4.4 Data Explorer Features & Description[3]

5. Implementation

5.1. System Configuration

Given the user-friendly feature of Kettle UI, we perform the benchmark testing under a local OSX environment. The computer is equipped with 8GB memory, powered by a Dual-Core Intel Core i5 processor capable of 2.7 GHz processing speed, under macOS Catalina 64 bits operating system (details shown in Figure 5.1).



macOS Catalina
Version 10.15.1

MacBook Pro (Retina, 13-inch, Early 2015)
Processor 2.7 GHz Dual-Core Intel Core i5
Memory 8 GB 1867 MHz DDR3
Startup Disk Macintosh HD
Graphics Intel Iris Graphics 6100 1536 MB
Serial Number

™ and © 1983-2019 Apple Inc. All Rights Reserved. License and Warranty

Figure 5.1. Testing System Information

5.2. Data Preparation

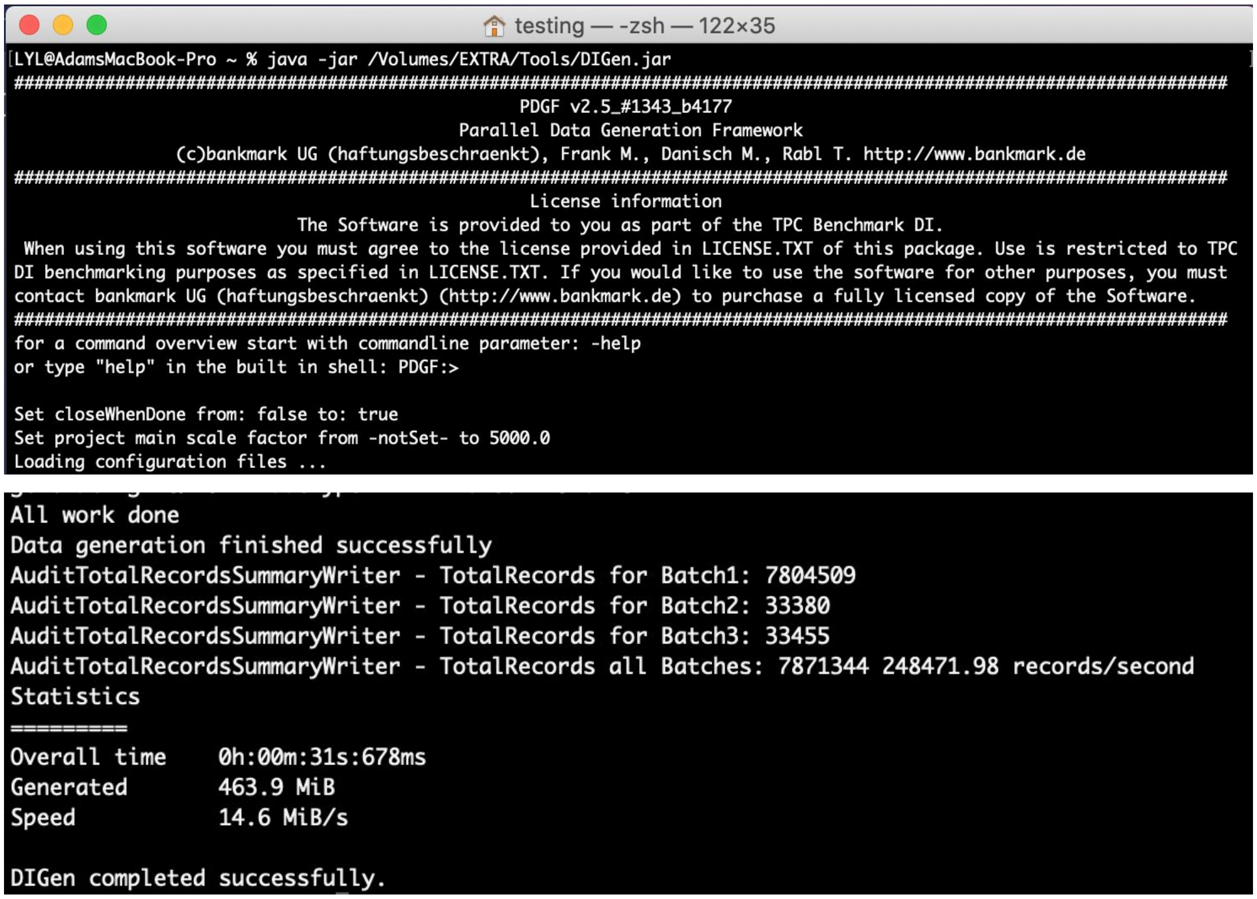
Source data is prepared by TPC with a data generator DIGen. It will directly generate all files and data required for a trial run. The prerequisites include Java SE 7+ and specified directory of folders. This report is conducted on DIGen Version 1.1.0 on OSX with Java 8. Below are the detailed steps of generating the data.

1. Download the TPCDI_Tools zip file at TPC- Current Specifications.

2. Unzip this file. A 'Tools' directory is created.

3. Change the 'PDGF' directory name to be all lower-case (i.e. 'pdgf') to meet the prerequisite.

4. Run DIGen with: $ java -jar DIGen.jar [options]. An output directory (-o) is defaulted as local and the default scale factory (-sf) is deployed (i.e. 3) (shown in Figure 5.2).



```
LYL@AdamsMacBook-Pro ~ % java -jar /Volumes/EXTRA/Tools/DIGen.jar
##############################################################################################################
                                            PDGF v2.5_#1343_b4177
                                        Parallel Data Generation Framework
                 (c)bankmark UG (haftungsbeschraenkt), Frank M., Danisch M., Rabl T. http://www.bankmark.de
##############################################################################################################
                                             License information
                              The Software is provided to you as part of the TPC Benchmark DI.
 When using this software you must agree to the license provided in LICENSE.TXT of this package. Use is restricted to TPC
DI benchmarking purposes as specified in LICENSE.TXT. If you would like to use the software for other purposes, you must
contact bankmark UG (haftungsbeschraenkt) (http://www.bankmark.de) to purchase a fully licensed copy of the Software.
##############################################################################################################
for a command overview start with commandline parameter: -help
or type "help" in the built in shell: PDGF:>

Set closeWhenDone from: false to: true
Set project main scale factor from -notSet- to 5000.0
Loading configuration files ...
```

```
All work done
Data generation finished successfully
AuditTotalRecordsSummaryWriter - TotalRecords for Batch1: 7804509
AuditTotalRecordsSummaryWriter - TotalRecords for Batch2: 33380
AuditTotalRecordsSummaryWriter - TotalRecords for Batch3: 33455
AuditTotalRecordsSummaryWriter - TotalRecords all Batches: 7871344 248471.98 records/second
Statistics
=========
Overall time    0h:00m:31s:678ms
Generated       463.9 MiB
Speed           14.6 MiB/s

DIGen completed successfully.
```

Figure 5.2. Data Generation Interface & Results

5.3. Data Warehouse Preparation

PostgreSQL (PSQL) is deployed as the data warehouse host to perform the benchmark. To ensure a “clean” SUT for the benchmark run, we set up PSQL from the very beginning. The following shows the data warehouse development in details.

1. Install PSQL with Brew.

```
brew install postgres -v
```

2. Initialize PSQL with the specified testing directory.

```
initdb /usr/local/var/postgres
```

3. Start PSQL.

```
pg_ctl -D /usr/local/var/postgres -l /usr/local/var/postgres/server.log start
```

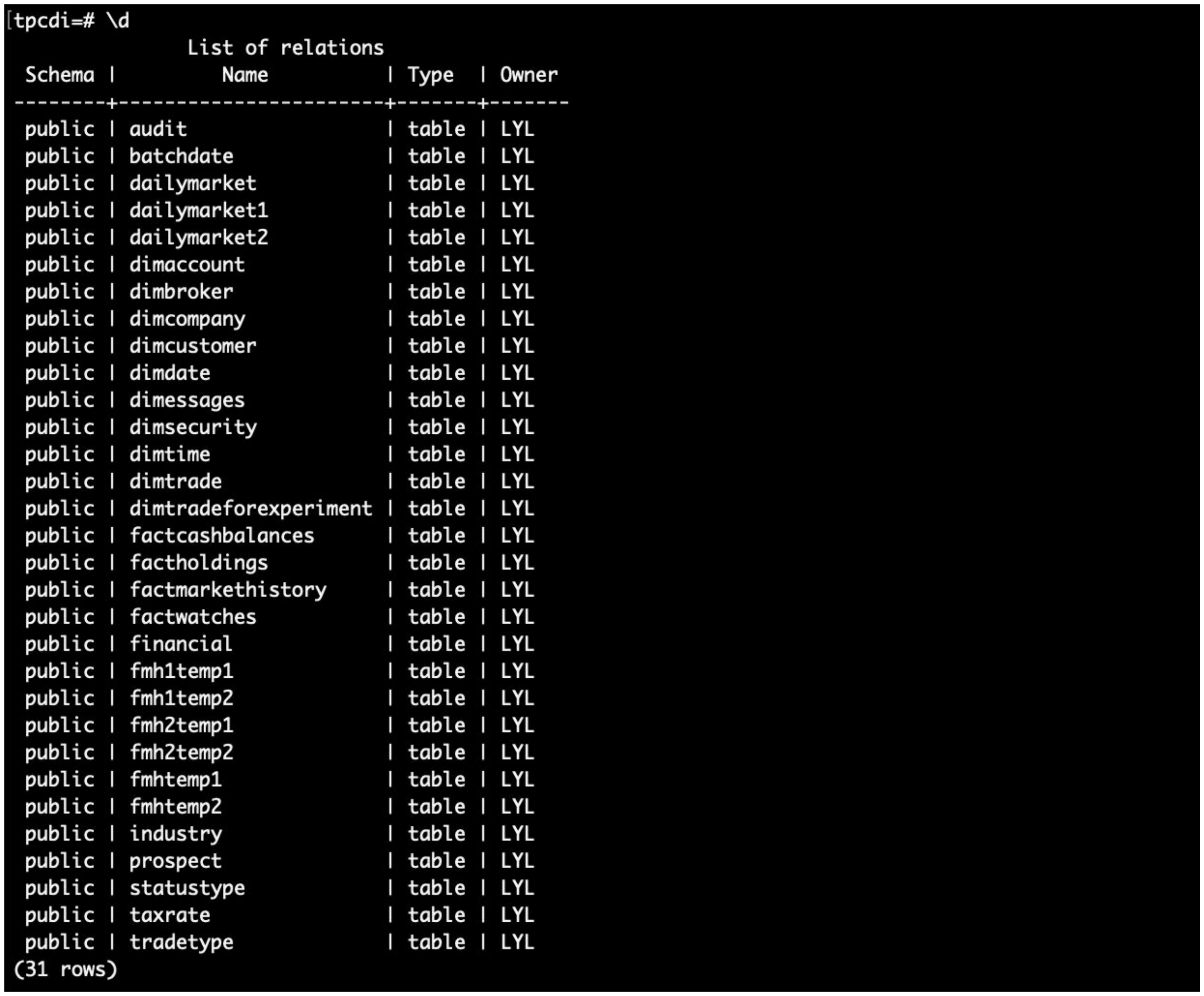
4. Create a testing database tpcdi.

```
createdb tpcdi
```

5. Access into tpcdi database, and load pre-established benchmark schema into tpcdi database.

```
psql tpcdi
tpcdi=# \i ~/TPCDI-postgre-schema.sql
```

6. Check the built table. Result is shown in Figure 5.3.



```
[tpcdi=# \d
                List of relations
 Schema |         Name          | Type  | Owner
--------+-----------------------+-------+-------
 public | audit                 | table | LYL
 public | batchdate             | table | LYL
 public | dailymarket           | table | LYL
 public | dailymarket1          | table | LYL
 public | dailymarket2          | table | LYL
 public | dimaccount            | table | LYL
 public | dimbroker             | table | LYL
 public | dimcompany            | table | LYL
 public | dimcustomer           | table | LYL
 public | dimdate               | table | LYL
 public | dimessages            | table | LYL
 public | dimsecurity           | table | LYL
 public | dimtime               | table | LYL
 public | dimtrade              | table | LYL
 public | dimtradeforexperiment | table | LYL
 public | factcashbalances      | table | LYL
 public | factholdings          | table | LYL
 public | factmarkethistory     | table | LYL
 public | factwatches           | table | LYL
 public | financial             | table | LYL
 public | fmh1temp1             | table | LYL
 public | fmh1temp2             | table | LYL
 public | fmh2temp1             | table | LYL
 public | fmh2temp2             | table | LYL
 public | fmhtemp1              | table | LYL
 public | fmhtemp2              | table | LYL
 public | industry              | table | LYL
 public | prospect              | table | LYL
 public | statustype            | table | LYL
 public | taxrate               | table | LYL
 public | tradetype             | table | LYL
(31 rows)
```

Figure 5.3. Established Hosting Data Warehouse in PostgreSQL

5.4. DI System Preparation

Kettle is developed under Java language, and therefore JDK is essential for the successful operation of Kettle. Here, our underlying JDK version is 1.8.0. We then download the latest version of Pantaho Kettle from the official website, and by unzip the file to the desired directory, we can easily use Kettle to conduct the test. In this report, we perform ETL development under the Spoon, the graphic interface program developed by Kettle. Spoon can be opened by (initial interface is shown in Figure 5.4):

```
$ sh spoon.sh
```

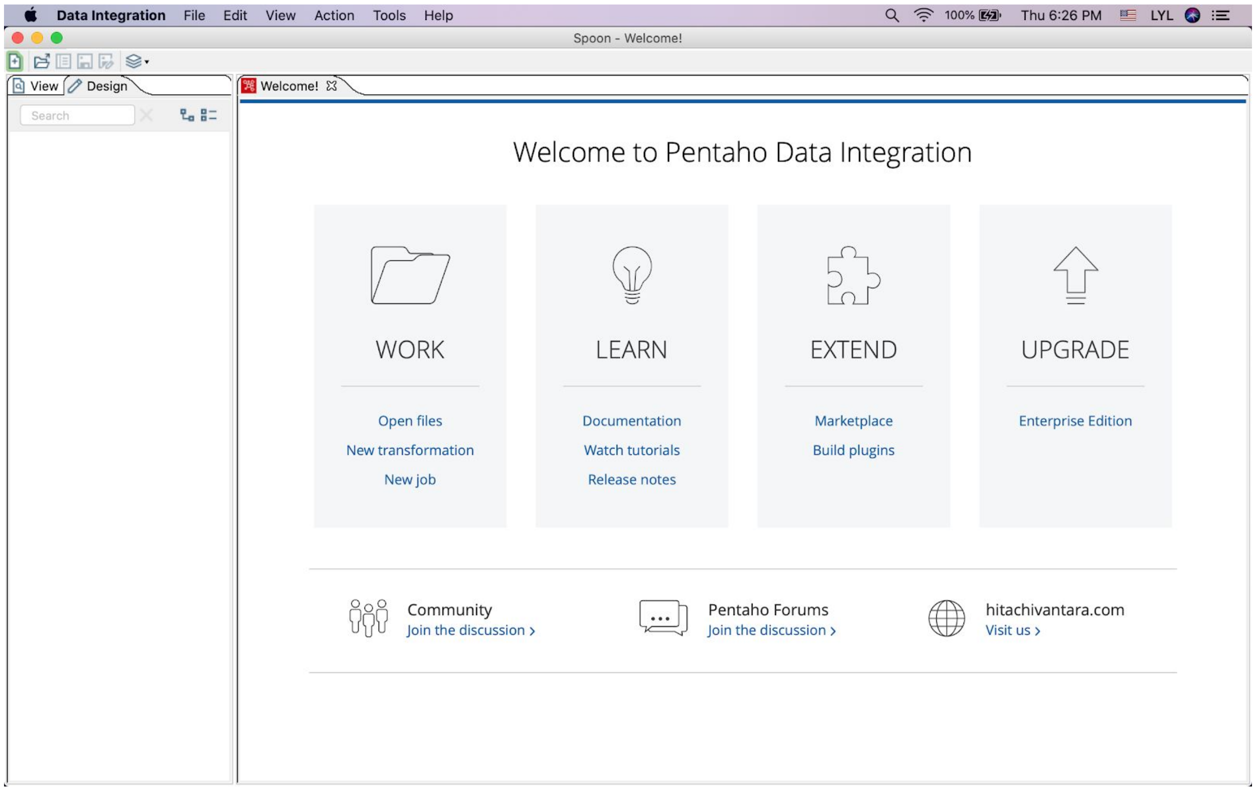


Figure 5.4. Initial Opening Interface of Kettle Spoon

5.5. Historical Load

The Historical Load includes different transformations than Incremental Updates. Destination tables are initially empty and being populated with new data, and the source files may have different ordering properties (an unload of the data from an OLTP table might be in primary key order, while a CDC extract would be in the order of the time that changes occurred).

In addition, there are sources of data that are different from the Incremental Updates. The Historical Load naturally uses a larger set of data than an Incremental Update. Following the Historical Load, the Validation Query collects certain information that will be used to check for correctness in the automated audit phase. [TPC-DI document]

With the abundant transformation units available in Kettle, the designed process of historical load in Kettle is shown in Figure 5.5. The successful running result is demonstrated in Figure 5.6 with detailed logging. What's worth mentioning is some simplified steps here. Due to the local computer testing limitation, the complicated transformation about the *DimAccount* and *DimSecurity* cannot be performed completely (*FINWIRE* & *CustomerMgmt.xml* cannot be performed wholly under the testing memory and CPU). Thus here we use rather simplified steps to load merely the basic data, which also implies some missing data in the *DImessages* table.

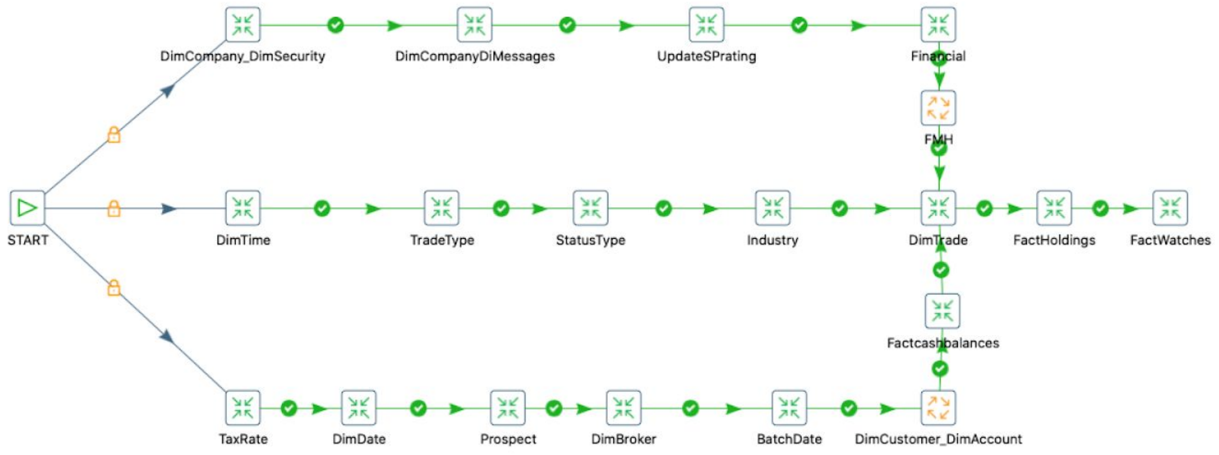
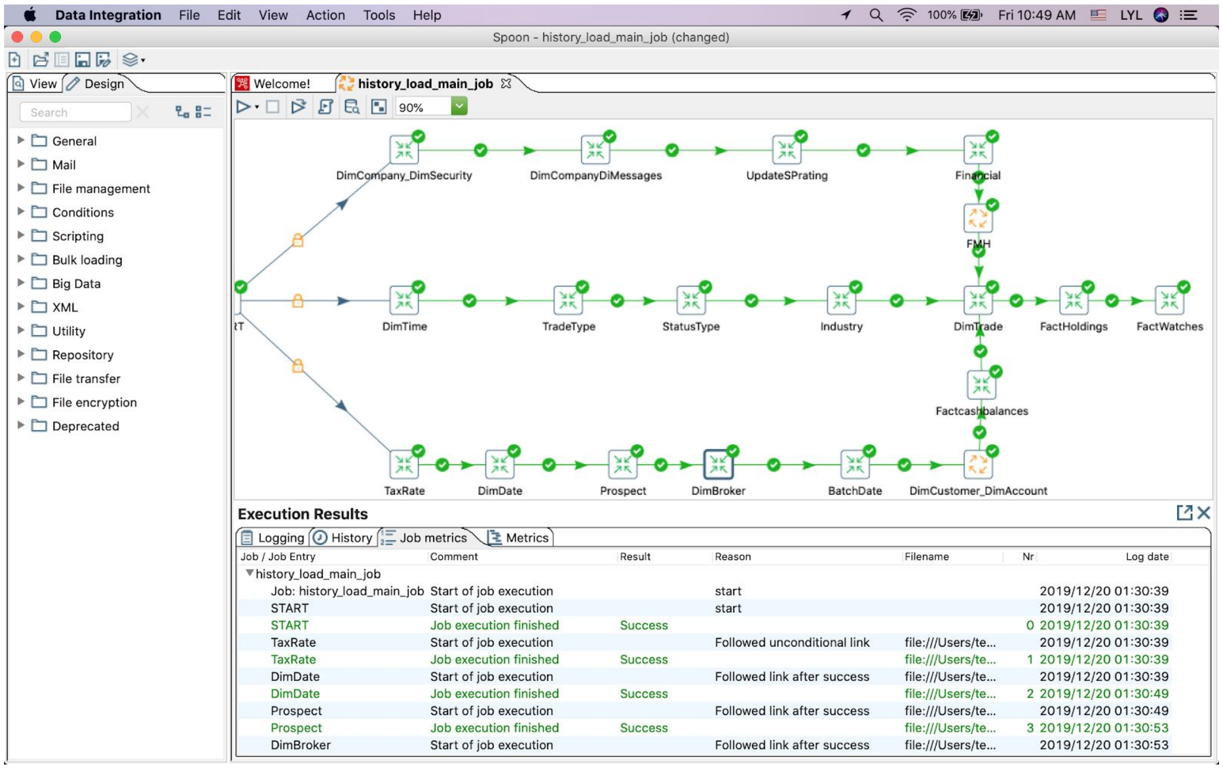


Figure 5.5. Historical Load ETL Process



Execution Results

Job / Job Entry	Comment	Result	Reason	Filename	Nr	Log date
history_load_main_job						
Job: history_load_main_job	Start of job execution		start			2019/12/20 01:30:39
START	Start of job execution		start			2019/12/20 01:30:39
START	Job execution finished	Success			0	2019/12/20 01:30:39
TaxRate	Start of job execution		Followed unconditional link	file:///Users/te...		2019/12/20 01:30:39
TaxRate	Job execution finished	Success		file:///Users/te...	1	2019/12/20 01:30:39
DimDate	Start of job execution		Followed link after success	file:///Users/te...		2019/12/20 01:30:39
DimDate	Job execution finished	Success		file:///Users/te...	2	2019/12/20 01:30:49
Prospect	Start of job execution		Followed link after success	file:///Users/te...		2019/12/20 01:30:49
Prospect	Job execution finished	Success		file:///Users/te...	3	2019/12/20 01:30:53
DimBroker	Start of job execution		Followed link after success	file:///Users/te...		2019/12/20 01:30:53

Figure 5.6. Successful Execution Result of Historical Load

According to the TPCDI official documentation, the validation query is also executed after the successful loading of the batch, and a phase completion record is written to the *DImessages* table. It is mainly used to generate information for the purpose of timing and validating the correctness of the batch loading.

5.6. Incremental updates

An Incremental Update includes different transformations than the Historical Load. The input files from the OLTP database are modeled as CDC extracts, which show the changes in the table data since the last extract. The *Prospect* file is a full data set (not CDC), so it is up to the DI application to determine what changes have occurred.

Two Incremental Update phases are required in a TPC-DI benchmark run. By requiring more than one Incremental Update, the benchmark ensures repeatability. Following each Incremental Update, the Validation Query collects certain information that will be used to check for correctness in the automated audit phase.[TPC-DI document]

The two incremental load processes are expressed in Figure 5.7 and 5.8 separately. Similarly, after the successful performance of each batch, the batch validation query is also executed.

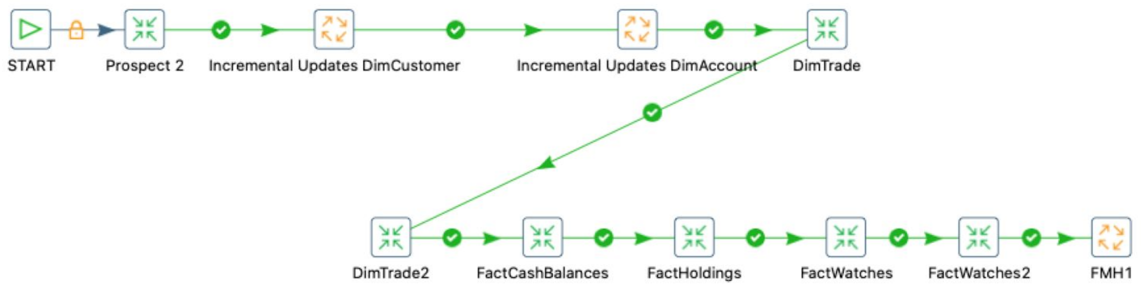


Figure 5.7. Incremental 1 ETL Process Design

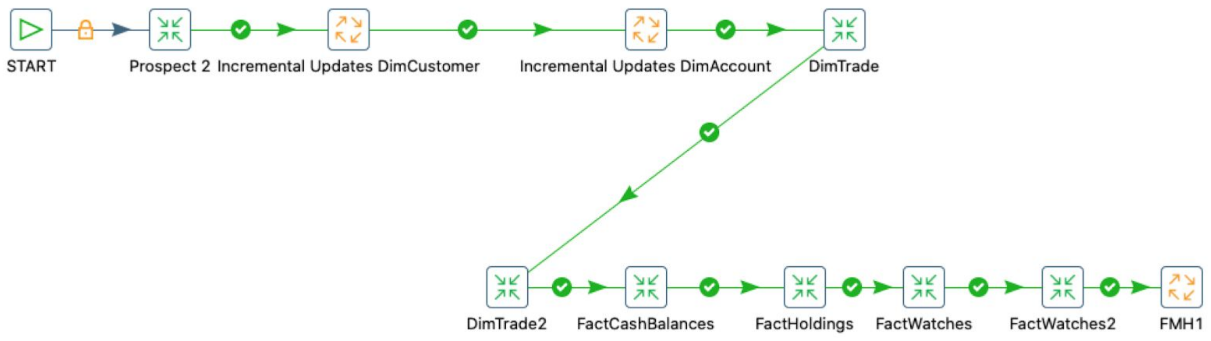
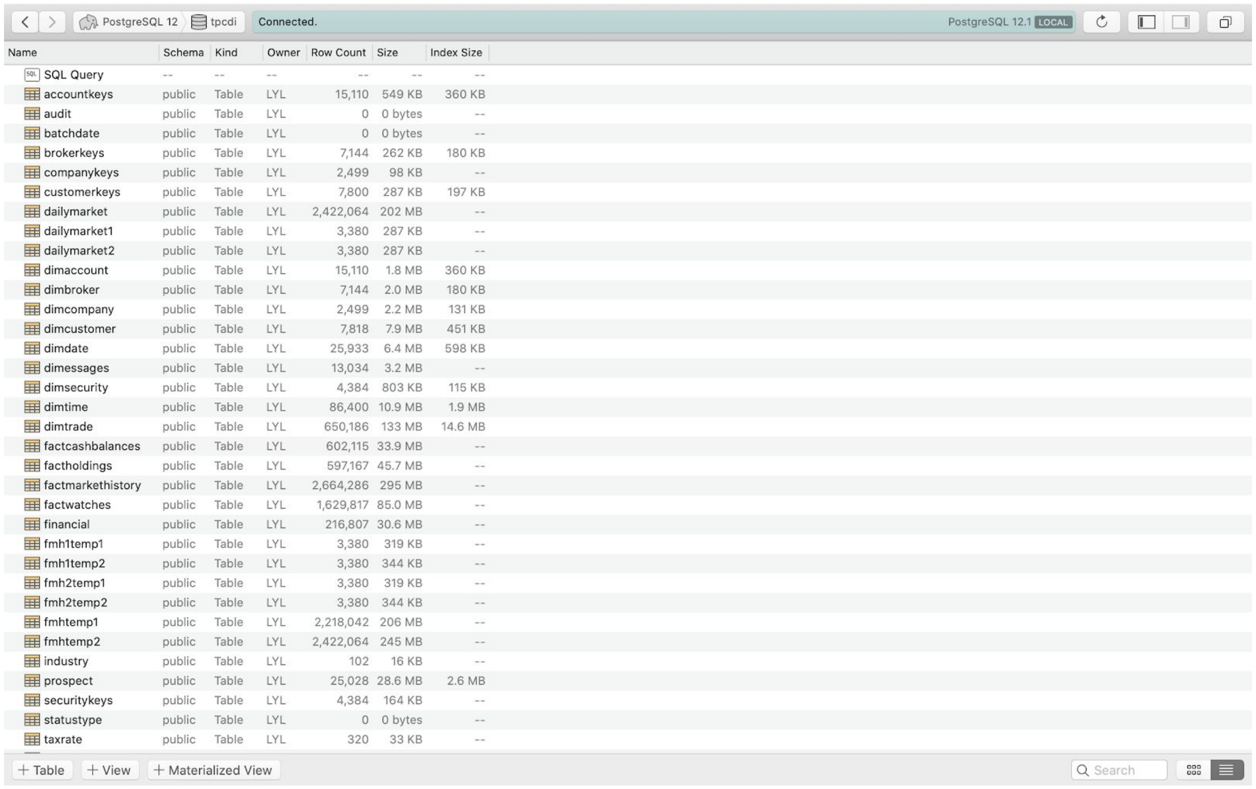


Figure 5.8. Incremental 2 ETL Process Design

After the successful ETL performances of all three batches, the data warehouse is loaded with data as presented in Figure 5.9.

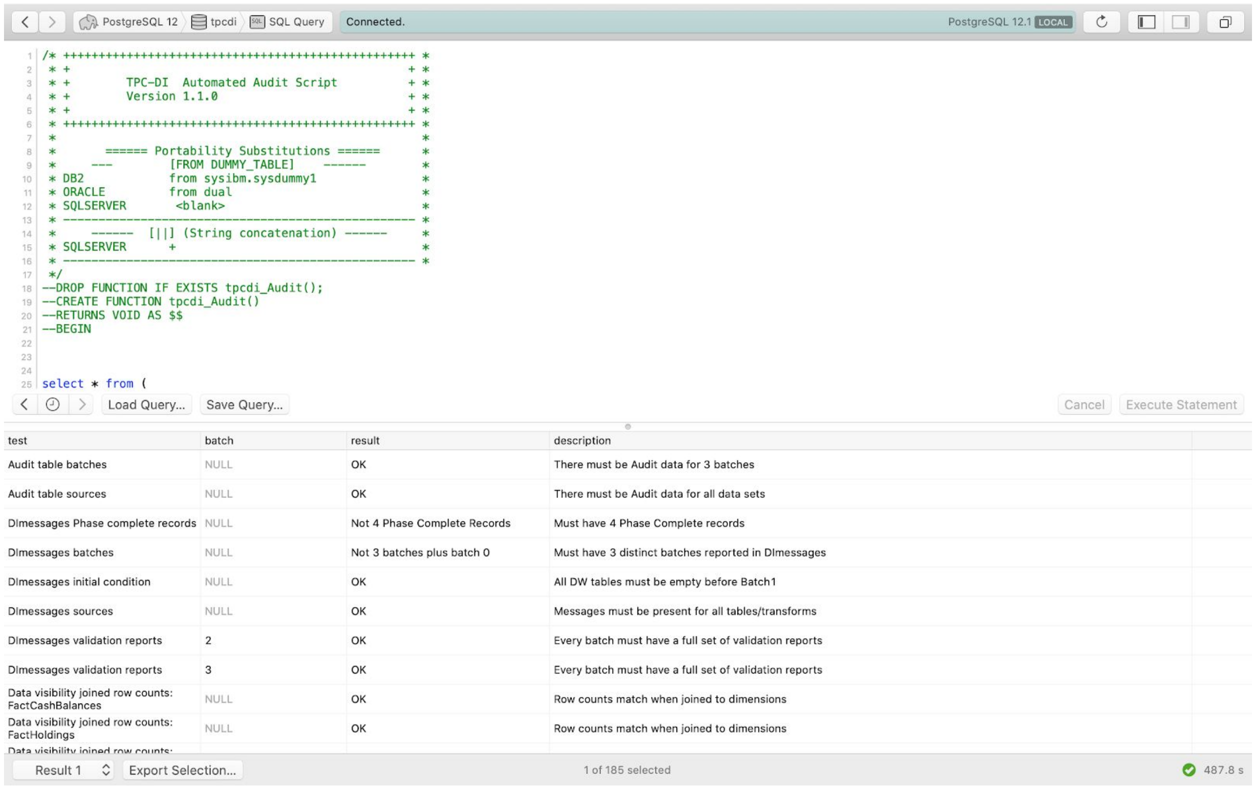


Name	Schema	Kind	Owner	Row Count	Size	Index Size
SQL Query	--	--	--	--	--	--
accountkeys	public	Table	LYL	15,110	549 KB	360 KB
audit	public	Table	LYL	0	0 bytes	--
batchdate	public	Table	LYL	0	0 bytes	--
brokerkeys	public	Table	LYL	7,144	262 KB	180 KB
companykeys	public	Table	LYL	2,499	98 KB	--
customerkeys	public	Table	LYL	7,800	287 KB	197 KB
dailymarket	public	Table	LYL	2,422,064	202 MB	--
dailymarket1	public	Table	LYL	3,380	287 KB	--
dailymarket2	public	Table	LYL	3,380	287 KB	--
dimaccount	public	Table	LYL	15,110	1.8 MB	360 KB
dimbroker	public	Table	LYL	7,144	2.0 MB	180 KB
dimcompany	public	Table	LYL	2,499	2.2 MB	131 KB
dimcustomer	public	Table	LYL	7,818	7.9 MB	451 KB
dimdate	public	Table	LYL	25,933	6.4 MB	598 KB
dimessages	public	Table	LYL	13,034	3.2 MB	--
dimsecurity	public	Table	LYL	4,384	803 KB	115 KB
dimtime	public	Table	LYL	86,400	10.9 MB	1.9 MB
dimtrade	public	Table	LYL	650,186	133 MB	14.6 MB
factcashbalances	public	Table	LYL	602,115	33.9 MB	--
factholdings	public	Table	LYL	597,167	45.7 MB	--
factmarkethistory	public	Table	LYL	2,664,286	295 MB	--
factwatches	public	Table	LYL	1,629,817	85.0 MB	--
financial	public	Table	LYL	216,807	30.6 MB	--
fmh1temp1	public	Table	LYL	3,380	319 KB	--
fmh1temp2	public	Table	LYL	3,380	344 KB	--
fmh2temp1	public	Table	LYL	3,380	319 KB	--
fmh2temp2	public	Table	LYL	3,380	344 KB	--
fmhtemp1	public	Table	LYL	2,218,042	206 MB	--
fmhtemp2	public	Table	LYL	2,422,064	245 MB	--
industry	public	Table	LYL	102	16 KB	--
prospect	public	Table	LYL	25,028	28.6 MB	2.6 MB
securitykeys	public	Table	LYL	4,384	164 KB	--
statustype	public	Table	LYL	0	0 bytes	--
taxrate	public	Table	LYL	320	33 KB	--

Figure 5.9. PostgreSQL Data Warehouse General Information after Loading

5.7. Automatic Auditing

After all of the other phases are complete, the automated audit queries the Data Warehouse to perform extensive tests on the resulting data and creates a simple report of the results (execution interface in PostgreSQL is presented in Figure 5.10). Referred to the official instruction, two more additional data visibility queries must be executed before the automated audit phase. All queries in this part are provided by the TPCDI official.



```
/* +++++++++++++++++++++++++++++++++++++++++++++++++ *
 * +                                               + *
 * +        TPC-DI  Automated Audit Script         + *
 * +        Version 1.1.0                          + *
 * +                                               + *
 * +++++++++++++++++++++++++++++++++++++++++++++++++ *
 *                                                   *
 *       ====== Portability Substitutions ======     *
 *     ---         [FROM DUMMY_TABLE]    ------      *
 * DB2             from sysibm.sysdummy1             *
 * ORACLE          from dual                         *
 * SQLSERVER        <blank>                          *
 * ------------------------------------------------- *
 *     ------  [||] (String concatenation) ------    *
 * SQLSERVER      +                                  *
 * ------------------------------------------------- *
 */
--DROP FUNCTION IF EXISTS tpcdi_Audit();
--CREATE FUNCTION tpcdi_Audit()
--RETURNS VOID AS $$
--BEGIN



select * from (
```

test	batch	result	description
Audit table batches	NULL	OK	There must be Audit data for 3 batches
Audit table sources	NULL	OK	There must be Audit data for all data sets
DImessages Phase complete records	NULL	Not 4 Phase Complete Records	Must have 4 Phase Complete records
DImessages batches	NULL	Not 3 batches plus batch 0	Must have 3 distinct batches reported in DImessages
DImessages initial condition	NULL	OK	All DW tables must be empty before Batch1
DImessages sources	NULL	OK	Messages must be present for all tables/transforms
DImessages validation reports	2	OK	Every batch must have a full set of validation reports
DImessages validation reports	3	OK	Every batch must have a full set of validation reports
Data visibility joined row counts: FactCashBalances	NULL	OK	Row counts match when joined to dimensions
Data visibility joined row counts: FactHoldings	NULL	OK	Row counts match when joined to dimensions
Data visibility joined row counts:			

Figure 5.10. Automated Audit Execution Interface in PostgreSQL

5.8 Challenges Faced

1. *Outofmemory* error: especially during sorting and merging. When the row size is too large, the data has to be loaded in the memory. 8 GB is not enough. Even when no transformation is running, Spoon takes about 4GB memory. And thus, on our laptop, there is not enough memory for the transformations to run. This can be seen by the frequent lags and freezes that happen when running the transformations in the kettle.

2. *DImessages*: This table serves as the DI operational alerting and monitoring functions containing periodical checks for information status and specified conditions, and batch completion, validation and visibility. As mentioned before, some changes are done in order to continuously loading data through our DI tool. For instance, while loading the DimAcc Table, the pre-designed ETL requires an exhaust validation for information status and format check, which is hard to perform in our testing environment. Therefore, we omit certain processes for *DImessage* table to complete the overall benchmarking and ensure the completion of the basic data warehouse schema.

6. Benchmark Results

6.1. Execution time

Execution time is one of the metrics upon which Data Integration tools are evaluated. Figure 6.1 shows the execution time for the 3 jobs that were executed for this project. Historical Load, as expected, took the most time (45:36 mins), followed by Incremental 1 load(7:13 mins) and Incremental 2 load(4:37 mins). Normally, historical load would take even longer, but in our case we could not load data to DIMessages table and had to reduce the number of rows almost for all the tables. This is because, if we tried loading all the rows, our computer would run out of memory and would show errors. Thus, we had to compromise.

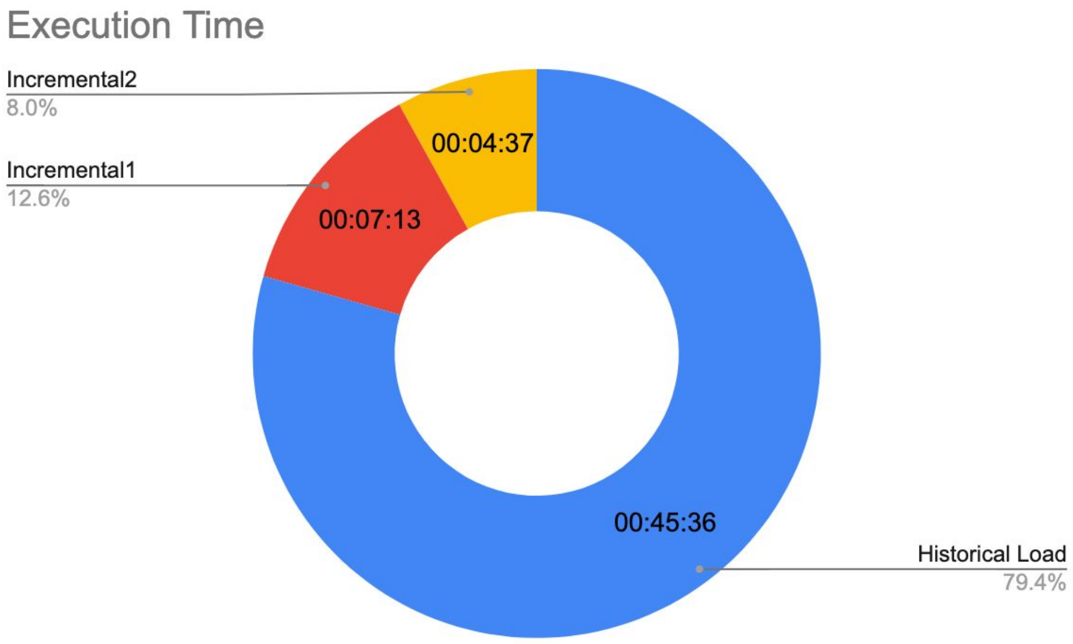


Figure 6.1 Execution Time for All Batch Loads

6.2. Audit Result

From the conceptual point of view, and as far as correctness is concerned, we succeeded in completing our task (audit result summary is shown in Figure 6.2). However, as can be seen in Auditing output in Appendix 1, we got many mismatch errors. Most of them are due to our memory limitations. We frequently encountered “OutOfMemory” exception while running our transformations, especially during sorting and merging (as explained in detail in 4.4.1). When the row size was too large, the data could not fit in the memory. And thus, on our PC, there is not enough memory for the transformations to run. Due to the time constraints, we could not partition the data to make sure that it fits in the memory.

There are few other errors we encountered like Bad value and not unique. More investigation is required to find out as to what caused them. Not unique pertains to error in Primary Key, but due to time constraints, we could not debug these errors.

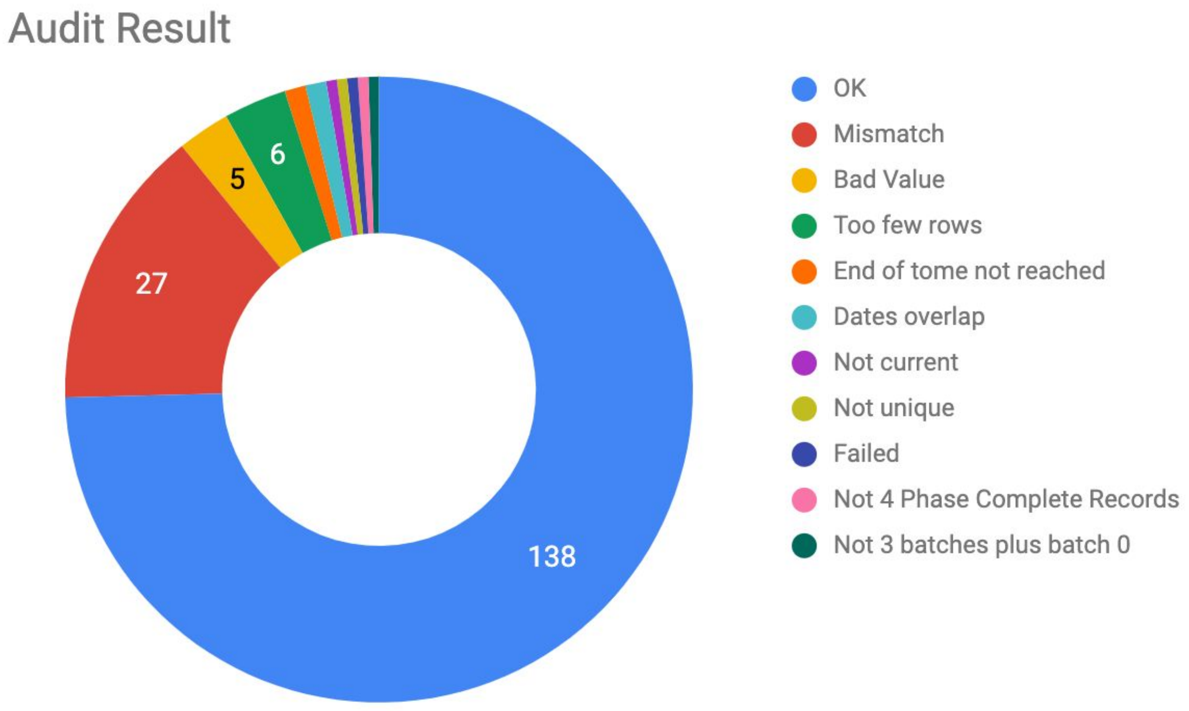


Figure 6.2. Audit Result Summary

7. Discussion

Our analysis of Kettle (Pentaho Data Integration), shows that it is an easy to use tool with a GUI that can be leveraged for making transformations and jobs in ETL easy to design and debug. However, having a GUI makes it RAM hungry which makes it difficult to run on a personal computer. We believe that Kettle would be best suitable to run on a High Performance Computer or a cluster, especially for big data. For a personal computer, given that data is not a lot, it would be best to use some scripting method like PETL (Python ETL).

However, Spoon GUI makes it quite easy for the developer to design and monitor processes. Kettle is quite charitable with the debugging logs and the developer can easily trace back to the source of the problem, should it arise during the design and run process.

8. Conclusion

In this report, we studied what it means to benchmark a tool in the data management world. With TPC, with its rich tools, giving us a way to evaluate the data management related tools in the market, it is easier than ever to evaluate and select the right tools for the task at hand. We introduced Kettle, a common open-source Data Integration Tool and benchmarked it using the TPC-DI dataset. We found that the tool is quite user-friendly and efficient when dealing with the data and for designing the ETL jobs. It also supports SQL and users can easily write jobs in the script should she feel like it. Kettle also has a rich online community so one can easily seek help if she gets stuck conducting an ETL project using Kettle.

References

1. TPC BENCHMARK ™ DI. (2015). *Transaction Processing Performance Council (TPC)*. Retrieved from http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-di_v1.1.0.pdf

2. Poess, M., Rabl, T., Jacobsen, H. A., & Caufield, B. (2014). TPC-DI: the first industry benchmark for data integration. *Proceedings of the VLDB Endowment, 7(*13), 1367-1378.

3. SQL Editor - Pentaho Data Integration - Pentaho Wiki. (2019). Retrieved from https://wiki.pentaho.com/display/EAI/.04+SQL+Editor

4. Pentaho Data Integration Introduction. (2019). Retrieved from https://help.pentaho.com/Documentation/6.0/0J0/0C0/000

5. Set Kettle Variables. (2019). Retrieved from https://help.pentaho.com/Documentation/7.1/0L0/0Y0/030/050/020/000/010/010

Appendices

Appendix 1: Audit Results for TPC-DI Benchmark Testing

test	batch	result	description
Audit table batches		OK	There must be Audit data for 3 batches
Audit table sources		OK	There must be Audit data for all data sets
DImessages Phase complete records		Not 4 Phase Complete Records	Must have 4 Phase Complete records
DImessages batches		Not 3 batches plus batch 0	Must have 3 distinct batches reported in DImessages
DImessages initial condition		OK	All DW tables must be empty before Batch1
DImessages sources		OK	Messages must be present for all tables/transforms
DImessages validation reports	2	OK	Every batch must have a full set of validation reports
DImessages validation reports	3	OK	Every batch must have a full set of validation reports
Data visibility joined row counts: FactCashBalances		OK	Row counts match when joined to dimensions
Data visibility joined row counts: FactHoldings		OK	Row counts match when joined to dimensions
Data visibility joined row counts: FactMarketHistory		OK	Row counts match when joined to dimensions
Data visibility joined row counts: FactWatches		OK	Row counts match when joined to dimensions
Data visibility row counts: DimAccount		OK	Row counts must be non-decreasing over time
Data visibility row counts: DimBroker		OK	Row counts must be non-decreasing over time
Data visibility row counts: DimCompany		OK	Row counts must be non-decreasing over time
Data visibility row counts: DimCustomer		OK	Row counts must be non-decreasing over time
Data visibility row counts: DimDate		OK	Row counts must be non-decreasing over time

Data visibility row counts: DimSecurity		OK	Row counts must be non-decreasing over time
Data visibility row counts: DimTime		OK	Row counts must be non-decreasing over time
Data visibility row counts: DimTrade		OK	Row counts must be non-decreasing over time
Data visibility row counts: FactCashBalances		OK	Row counts must be non-decreasing over time
Data visibility row counts: FactHoldings		OK	Row counts must be non-decreasing over time
Data visibility row counts: FactMarketHistory		OK	Row counts must be non-decreasing over time
Data visibility row counts: FactWatches		OK	Row counts must be non-decreasing over time
Data visibility row counts: Financial		OK	Row counts must be non-decreasing over time
Data visibility row counts: Industry		OK	Row counts must be non-decreasing over time
Data visibility row counts: Prospect		OK	Row counts must be non-decreasing over time
Data visibility row counts: StatusType		OK	Row counts must be non-decreasing over time
Data visibility row counts: TaxRate		OK	Row counts must be non-decreasing over time
Data visibility row counts: TradeType		OK	Row counts must be non-decreasing over time
DimAccount EffectiveDate	1	OK	All records from a batch have an EffectiveDate in the batch time window
DimAccount EffectiveDate	2	OK	All records from a batch have an EffectiveDate in the batch time window
DimAccount EffectiveDate	3	OK	All records from a batch have an EffectiveDate in the batch time window
DimAccount End of Time		OK	Every Account has one record with a date range reaching the end of time
DimAccount EndDate		OK	EndDate of one record matches EffectiveDate of another, or the end of time
DimAccount IsCurrent		OK	IsCurrent is 1 if EndDate is the end of time, else Iscurrent is 0
DimAccount Overlap		OK	Date ranges do not overlap for a given Account

DimAccount SK_BrokerID		OK	All SK_BrokerIDs match a broker record with a valid date range
DimAccount SK_CustomerID		OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
DimAccount Status		Bad value	All Status values are valid
DimAccount TaxStatus		OK	All TaxStatus values are valid
DimAccount batches		Mismatch	BatchID values must match Audit table
DimAccount consolidation		OK	No records become effective and end on the same day
DimAccount distinct keys		OK	All SKs are distinct
DimAccount inactive customers		OK	If a customer is inactive, the corresponding accounts must also have been inactive
DimAccount row count	1	Too few rows	Actual row count matches or exceeds Audit table minimum
DimAccount row count	2	Too few rows	Actual row count matches or exceeds Audit table minimum
DimAccount row count	3	Too few rows	Actual row count matches or exceeds Audit table minimum
DimBroker BatchID	1	OK	All rows report BatchID = 1
DimBroker EffectiveDate		OK	All rows have Batch1 BatchDate as EffectiveDate
DimBroker EndDate		OK	All rows have end of time as EndDate
DimBroker IsCurrent		OK	All rows have IsCurrent = 1
DimBroker distinct keys		OK	All SKs are distinct
DimBroker row count		OK	Actual row count matches Audit table
DimCompany Country		OK	All Country values are valid
DimCompany EffectiveDate	1	OK	All records from a batch have an EffectiveDate in the batch time window
DimCompany EffectiveDate	2	OK	All records from a batch have an EffectiveDate in the batch time window
DimCompany EffectiveDate	3	OK	All records from a batch have an EffectiveDate in the batch time window
DimCompany End of Time		End of tome not reached	Every company has one record with a date range reaching the end of time
DimCompany EndDate		OK	EndDate of one record matches EffectiveDate of another, or the end of time
DimCompany Industry		OK	Industry values are from the Industry table
DimCompany Overlap		Dates overlap	Date ranges do not overlap for a given company
DimCompany SPrating		OK	All SPrating values are valid
DimCompany Status		OK	All Status values are valid

DimCompany batches		OK	BatchID values must match Audit table
DimCompany consolidation		OK	No records become effective and end on the same day
DimCompany distinct keys		OK	All SKs are distinct
DimCompany distinct names		OK	Every company has a unique name
DimCompany row count	1	OK	Actual row count matches or exceeds Audit table minimum
DimCompany row count	2	OK	Actual row count matches or exceeds Audit table minimum
DimCompany row count	3	OK	Actual row count matches or exceeds Audit table minimum
DimCustomer EffectiveDate	1	OK	All records from a batch have an EffectiveDate in the batch time window
DimCustomer EffectiveDate	2	OK	All records from a batch have an EffectiveDate in the batch time window
DimCustomer EffectiveDate	3	OK	All records from a batch have an EffectiveDate in the batch time window
DimCustomer Email1		OK	Email1 values are properly formatted
DimCustomer Email2		OK	Email2 values are properly formatted
DimCustomer End of Time		OK	Every Customer has one record with a date range reaching the end of time
DimCustomer EndDate		OK	EndDate of one record matches EffectiveDate of another, or the end of time
DimCustomer Gender		Bad value	All Gender values are valid
DimCustomer IsCurrent		Not current	IsCurrent is 1 if EndDate is the end of time, else Iscurrent is 0
DimCustomer LocalTaxRate		OK	LocalTaxRateDesc and LocalTaxRate values are from TaxRate table
DimCustomer NationalTaxRate		OK	NationalTaxRateDesc and NationalTaxRate values are from TaxRate table
DimCustomer Overlap		OK	Date ranges do not overlap for a given Customer
DimCustomer Phone1		Mismatch	Phone1 values are properly formatted
DimCustomer Phone2		Mismatch	Phone2 values are properly formatted
DimCustomer Phone3		Mismatch	Phone3 values are properly formatted
DimCustomer Status		Bad value	All Status values are valid
DimCustomer TaxID		OK	TaxID values are properly formatted
DimCustomer age range alerts	1	OK	Count of age range alerts matches Audit table

DimCustomer age range alerts	2	Mismatch	Count of age range alerts matches Audit table
DimCustomer age range alerts	3	Mismatch	Count of age range alerts matches Audit table
DimCustomer batches		Mismatch	BatchID values must match Audit table
DimCustomer consolidation		OK	No records become effective and end on the same day
DimCustomer customer tier alerts	1	OK	Count of customer tier alerts matches Audit table
DimCustomer customer tier alerts	2	OK	Count of customer tier alerts matches Audit table
DimCustomer customer tier alerts	3	OK	Count of customer tier alerts matches Audit table
DimCustomer demographic fields		OK	For current customer records that match Prospect records, the demographic fields also match
DimCustomer distinct keys		OK	All SKs are distinct
DimCustomer inactive customers	1	Mismatch	Inactive customer count matches Audit table
DimCustomer inactive customers	2	Mismatch	Inactive customer count matches Audit table
DimCustomer inactive customers	3	Mismatch	Inactive customer count matches Audit table
DimCustomer row count	1	Too few rows	Actual row count matches or exceeds Audit table minimum
DimCustomer row count	2	Too few rows	Actual row count matches or exceeds Audit table minimum
DimCustomer row count	3	Too few rows	Actual row count matches or exceeds Audit table minimum
DimSecurity EffectiveDate	1	OK	All records from a batch have an EffectiveDate in the batch time window
DimSecurity EffectiveDate	2	OK	All records from a batch have an EffectiveDate in the batch time window
DimSecurity EffectiveDate	3	OK	All records from a batch have an EffectiveDate in the batch time window
DimSecurity End of Time		End of tome not reached	Every company has one record with a date range reaching the end of time

DimSecurity EndDate		OK	EndDate of one record matches EffectiveDate of another, or the end of time
DimSecurity ExchangeID		OK	All ExchangeID values are valid
DimSecurity IsCurrent		OK	IsCurrent is 1 if EndDate is the end of time, else Iscurrent is 0
DimSecurity Issue		OK	All Issue values are valid
DimSecurity Overlap		Dates overlap	Date ranges do not overlap for a given company
DimSecurity SK_CompanyID		OK	All SK_CompanyIDs match a DimCompany record with a valid date range
DimSecurity Status		Bad value	All Status values are valid
DimSecurity batches		OK	BatchID values must match Audit table
DimSecurity consolidation		OK	No records become effective and end on the same day
DimSecurity distinct keys		OK	All SKs are distinct
DimSecurity row count	1	OK	Actual row count matches or exceeds Audit table minimum
DimTrade SK_AccountID		OK	All SK_AccountIDs match a DimAccount record with a valid date range
DimTrade SK_BrokerID		OK	All SK_BrokerIDs match a DimBroker record with a valid date range
DimTrade SK_CompanyID		OK	All SK_CompanyIDs match a DimCompany record with a valid date range
DimTrade SK_CustomerID		OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
DimTrade SK_SecurityID		OK	All SK_SecurityIDs match a DimSecurity record with a valid date range
DimTrade Status		OK	All Trade Status values are valid
DimTrade Type		OK	All Trade Type values are valid
DimTrade batches		OK	BatchID values must match Audit table
DimTrade canceled trades		Mismatch	Actual row counts matches Audit table
DimTrade charge alerts		OK	Actual row counts matches Audit table
DimTrade commission alerts		OK	Actual row counts matches Audit table
DimTrade date check	2	OK	All SK_DateID values are in the correct batch time window
DimTrade date check	3	OK	All SK_DateID values are in the correct batch time window
DimTrade distinct keys		Not unique	All keys are distinct
DimTrade row count	1	Mismatch	Actual total matches Audit table

DimTrade row count	2	Mismatch	Actual total matches Audit table
DimTrade row count	3	Mismatch	Actual total matches Audit table
FactCashBalances SK_AccountID		OK	All SK_AccountIDs match a DimAccount record with a valid date range
FactCashBalances SK_CustomerID		OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
FactCashBalances SK_DateID	2	OK	All dates are within batch date range
FactCashBalances SK_DateID	3	OK	All dates are within batch date range
FactCashBalances batches		OK	BatchID values must match Audit table
FactHoldings CurrentTradeID		Failed	CurrentTradeID matches a DimTrade record with and Close Date and Time are values are used as the holdings date and time
FactHoldings SK_AccountID		OK	All SK_AccountIDs match a DimAccount record with a valid date range
FactHoldings SK_CompanyID		OK	All SK_CompanyIDs match a DimCompany record with a valid date range
FactHoldings SK_CustomerID		OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
FactHoldings SK_DateID	2	OK	All dates are within batch date range
FactHoldings SK_DateID	3	OK	All dates are within batch date range
FactHoldings SK_SecurityID		OK	All SK_SecurityIDs match a DimSecurity record with a valid date range
FactHoldings batches		OK	BatchID values must match Audit table
FactHoldings row count	1	Mismatch	Actual row count matches Audit table
FactHoldings row count	2	Mismatch	Actual row count matches Audit table
FactHoldings row count	3	Mismatch	Actual row count matches Audit table
FactMarketHistory SK_CompanyID		OK	All SK_CompanyIDs match a DimCompany record with a valid date range
FactMarketHistory SK_DateID	2	OK	All dates are within batch date range
FactMarketHistory SK_DateID	3	OK	All dates are within batch date range

FactMarketHistory SK_SecurityID		OK	All SK_SecurityIDs match a DimSecurity record with a valid date range
FactMarketHistory batches		OK	BatchID values must match Audit table
FactMarketHistory relative dates		OK	52-week-low <= day_low <= close_price <= day_high <= 52-week-high
FactMarketHistory row count	1	Mismatch	Actual row count matches Audit table
FactMarketHistory row count	2	Mismatch	Actual row count matches Audit table
FactMarketHistory row count	3	Mismatch	Actual row count matches Audit table
FactWatches SK_CustomerID		OK	All SK_CustomerIDs match a DimCustomer record with a valid date range
FactWatches SK_SecurityID		OK	All SK_SecurityIDs match a DimSecurity record with a valid date range
FactWatches active watches	1	Mismatch	Actual total matches Audit table
FactWatches active watches	2	Mismatch	Actual total matches Audit table
FactWatches active watches	3	Mismatch	Actual total matches Audit table
FactWatches batches		OK	BatchID values must match Audit table
FactWatches date check	2	OK	All SK_DateID_ values are in the correct batch time window
FactWatches date check	3	OK	All SK_DateID_ values are in the correct batch time window
FactWatches row count	1	Mismatch	Actual row count matches Audit table
FactWatches row count	2	Mismatch	Actual row count matches Audit table
FactWatches row count	3	Mismatch	Actual row count matches Audit table
Financial EPS		OK	Earnings calculations are valid
Financial FI_QTR		OK	All quarters are in (1, 2, 3, 4)
Financial FI_QTR_START_DATE		OK	All quarters start on correct date
Financial FI_YEAR		OK	All Years are within Batch1 range
Financial SK_CompanyID		OK	All SK_CompanyIDs match a DimCompany record
Financial row count		Mismatch	Actual row count matches Audit table

Prospect Country		OK	All Country values are valid
Prospect MarketingNameplate		Bad value	All MarketingNameplate values match the data
Prospect SK_RecordDateID	1	OK	All records from batch have SK_RecordDateID in or after the batch time window
Prospect SK_RecordDateID	2	OK	All records from batch have SK_RecordDateID in or after the batch time window
Prospect SK_RecordDateID	3	OK	All records from batch have SK_RecordDateID in or after the batch time window
Prospect SK_UpdateDateID		OK	SK_RecordDateID must be newer or same as SK_UpdateDateID
Prospect batches		OK	BatchID values must match Audit table