

1. Introduction

The process of combining data from different sources into a unified view is called Data Integration (DI) [1]. Several phases are involved in the DI process. Among them we find cleansing, extracting, transforming and loading data. The former three are ETL, a rather relevant concept in data warehousing in which data is taken from the source system and then loaded into the warehouse. This is done in order to obtain useful and consistent information for analytics from multiple data sources [2].

The present report includes the description and details of the TPC-DI benchmark utilisation with the tool Talend Open Studio for Data Integration. MS SQL Server is the technology used as a Data Warehouse (DW). Firstly, a brief introduction to the technology fundamentals will be provided. A description of the features and capabilities of both MS SQL Server and Talend, as well as an explanation of the TPC-DI benchmark. After this, the details of how to actually carry out the benchmark with Talend are included.

2. Technology fundamentals

2.1. MS SQL Server

Microsoft SQL Server is a database platform for large-scale online transaction processing (OLTP), DW, and a BI platform for data integration, analysis, and reporting solutions [3]. Its column-wise storage brings several advantages for OLAP and Business Intelligence (BI) tools query processing. Hanson, Larson and Price [4] describe the following as some of the customer benefits from using SQL Server 2012:

- Faster data exploration leading to better business decisions.
- Lower skill and time requirements; designing indexes, materialized views and summary tables requires time and a high level of database expertise.
- Reduced need to maintain a separate copy of the data on an OLAP server.
- Faster data ingestion due to reduced index and aggregate maintenance.
- Reduced need to move to a scale-out solution.
- Lower disk space, CPU, and power requirements.
- Overall lower costs.

Microsoft's web page also provides a pictogram with several benefits the SQL Server version 2017 has to offer [5]. Please refer to image below for one related to query performance.

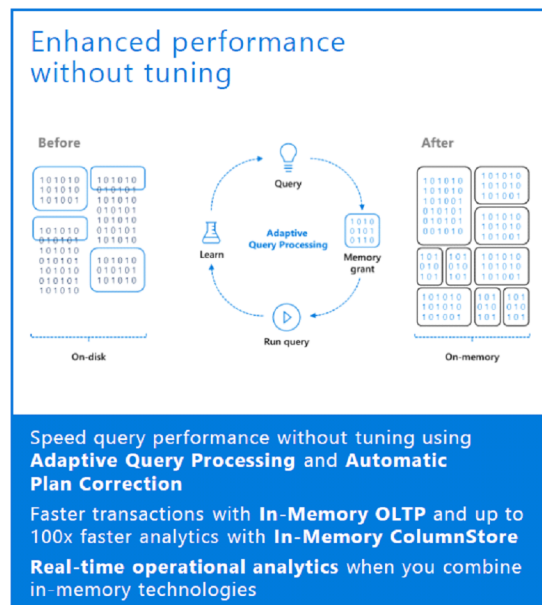


Figure 1: SQL Server adaptive query processing. Source (Microsoft.com)

It utilizes Tabular Data Stream (TDS) as its application layer protocol to handle client-server requests for database servers. It provides the following services [6]:

- Authentication and channel encryption negotiation.
- Specification of requests in SQL (including Bulk Insert).
- Invocation of a stored procedure, also known as a Remote Procedure Call (RPC).
- Returning of data.

- Transaction Manager Requests.

2.2. Talend Open Studio for Data Integration

As its name suggests, Talend Open Studio for Data Integration is an open-source tool for performing DI. It provides a graphical environment for devising and implementing custom integrations between systems [7]. The main feature of this technology is the number of built-in connectors it includes (more than 600), which aim to make possible to define complicated integration processes.

It can be seen as a Java code generator, thus being suitable for both developers and non-developers. Nevertheless, having Java knowledge can be useful when using Talend [8].

There is a handful of use cases in which Talend can be a very suitable tool. Among them we find [7]:

- Migrating data from one database to another: This is a rather common scenario when it comes to implementing new systems or upgrading existing ones. Thanks to the numerous database connectors and actions, this task can be done with Talend.
- Exchanging files between systems: This task requires oftenly transformations of some kind. Through its ability to handle different formats, Talend is a suitable tool for this kind of end-to-end file exchange process.
- Synchronizing data: A company can have the same data in multiple repositories. Talend can be used to maintain this data synchronization across different systems.
- ETL: As stated above, ETL is an essential element of a DW or BI system. Talend provides the users with different actions and connectors for extracting the data from operational systems, transforming it and loading that data into a DW system.

2.3. TPC-DI

As part of the benchmark toolset that the Transaction Processing Performance Council provides, we can find TPC-DI, a benchmark for assessing the DI capabilities of a given tool. This benchmark includes a model, which represents an extract from an OLTP system, together with other auxiliary data sources to be loaded into a DW [9]. The operations needed, the source and destination of schemas, and the implementation rules have been devised taking into account nowadays' DI requirements. Thus, the main features that this benchmark will assess, among others, are [10]:

- The ability of manipulating and loading large amounts of data
- Transformations support (v.g.: error checking, surrogate key lookups, aggregations, conversions, etc.)
- Historical and incremental loading of the DW.
- Multiple formats support.

In order to test a system regarding its DI capabilities, four main steps have to be done. Firstly, the environment has to be prepared for the benchmark assessing; the data is generated using

the programmes included in TPCDI, the DW is created, and the DI tool is prepared and configured. None of these tasks is timed regarding the benchmark. Secondly, the Historical Load phase is carried out. The empty tables are populated in a certain order, and once done this, the Validation Query is executed in order to assess the correctness of the DW. This phase has to be timed in order to be used for the benchmark metric. Thirdly, the Incremental Update phase is performed. Two incremental updates are done, as this assess repeatability. After them, the Validation Query is run again. The execution times are taken into consideration for the benchmark metric. Finally, the Automated Audit phase is carried out. Extensive tests on the DW are done, and a report of the results is generated. This phase is not timed, but all tests must pass for the run to be valid.

3. Implementation details

3.1. TPC-DI setup

The first phase of the TPC-DI benchmark involves preparing it. In order to do this, besides creating the DW and setting up the DI tool, the data has to be generated. In order to do this, we will download the TPC-DI files from the TPC webpage and run the data generator with the following command:

```
java -jar DIGen.jar
```

Some options can be added. Regarding the scale factor, it is set with the flag “-sf”, with a default value of 5, and with a meaning of hundreds of megabytes generated (i.e.: the default size of the data is around 500 mb).

Regarding the dependencies of this tool, we need to have Java 7 installed in our system. This java file will generate three different batches for our DI process. These files have different formats (.txt, .csv, .xml) and they are used for populating the tables of the DW. Besides, a report for the DIGen tool will also be generated, as can be seen in figure 2.

```
TPC-DI Data Generation Report
=====

Start Time: 2019-11-13T14:10:19+0100
End Time: 2019-11-13T14:11:11+0100
DIGen Version: 1.1.0
Scale Factor: 5
AuditTotalRecordsSummaryWriter - TotalRecords for Batch1: 7804509
AuditTotalRecordsSummaryWriter - TotalRecords for Batch2: 33380
AuditTotalRecordsSummaryWriter - TotalRecords for Batch3: 33455
AuditTotalRecordsSummaryWriter - TotalRecords all Batches: 7871344 187842.31 records/second

Command options used: -sf 5
PDGF Version: PDGF v2.5_#1343_b4177
Java version: Oracle Corporation 1.8.0_231
```

Figure 2: Report generated by the DIGen tool

3.2. Environment setup

3.2.1. Talend Open Studio for DI

As stated in the previous section, another important part in the preparation phase is setting up the DI tool. In order to do this, Talend Open Studio for DI can be downloaded in the Talend web page downloads section ([link](#)). Being an open-source tool, there are free versions available for both Windows and Mac. Regarding the installation in windows, it is important not to choose “Program Files” as the directory to install Talend, as this can lead us to problems in the future. Another important issue is that the System Under Test in which Talend will be installed needs to have the Java Runtime Environment installed, as it is Java-based, and, in the end, it is a Java code generator.

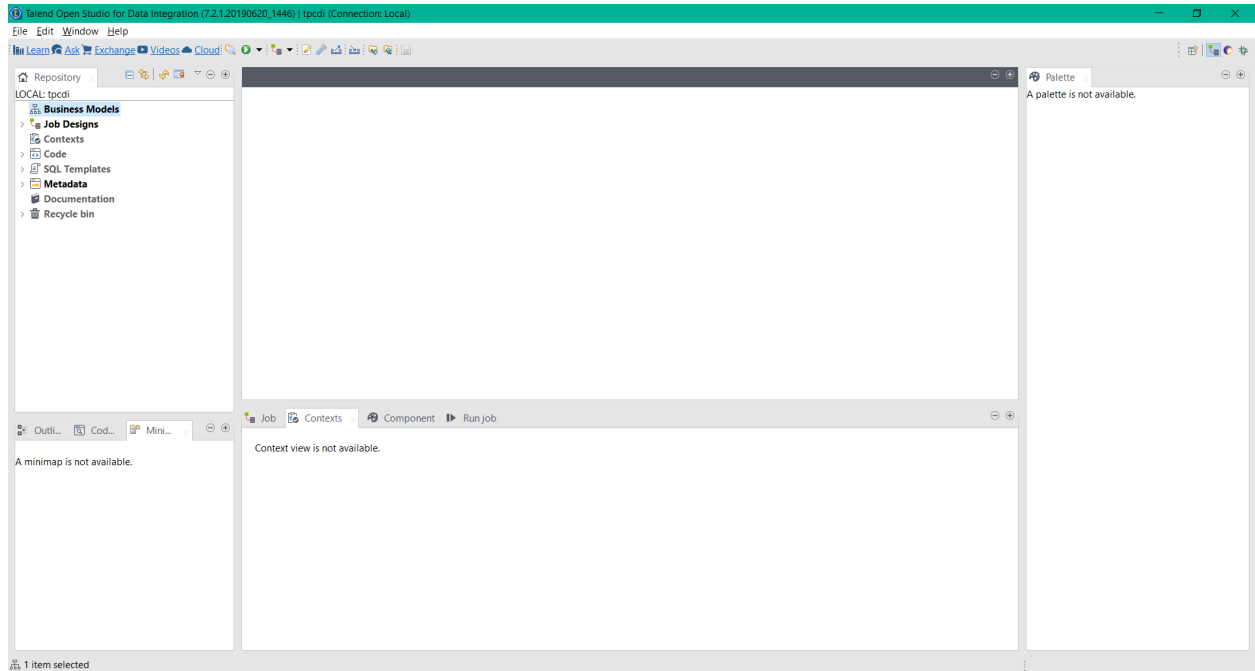


Figure 3: Talend Open Studio for Data Integration interface

Once installed, we will create a new project and we will have an interface as the one in figure 3. Now the database connection has to be created by assigning an account and credentials within the DB Connections interface inside Talend, in order to be able to do the inserts from the files generated in the previous section. As these steps turned out to be somehow complicated, we will include a small walkthrough for this issue for creating a new MS SQL Server connection. The steps that have to be done for this are:

- Under metadata section in the left menu, right click in db connections and select create connection (Figure 4)

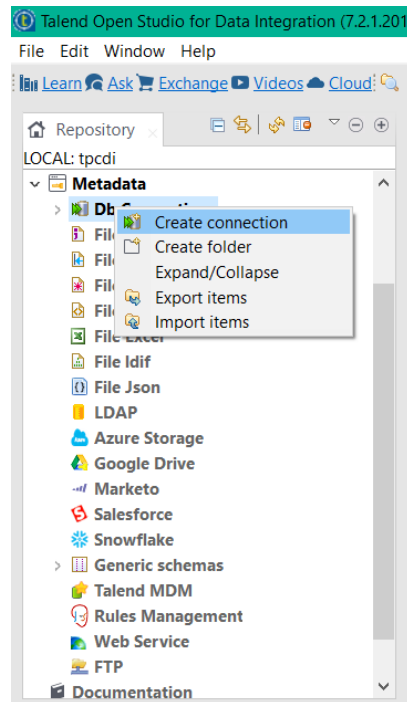


Figure 4: Talend menu for creating a new DB connection

- Give it a name (and, optionally, a purpose and a description)
- In the next window, the details from the database have to be set (Figure 5). The user has to be set to "sa".

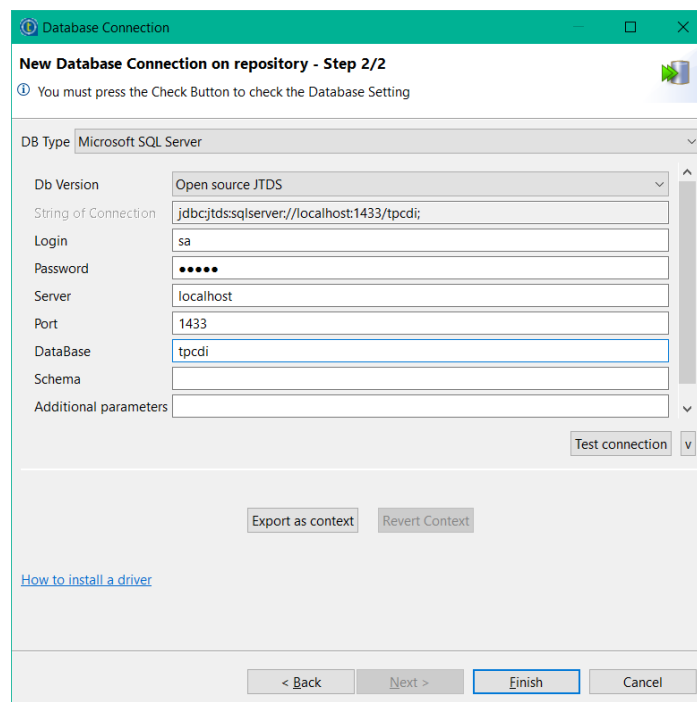


Figure 5: Step in the new DB connection wizard

Once done this, we will have to do some configurations in MS SQL Server as well. Firstly, we will go to SQL Server configuration manager, and under the SQL Server Network configuration, right click on the TCP/IP section, and in the IP addresses tab, set the TCP port to 1433 (Figure 6).

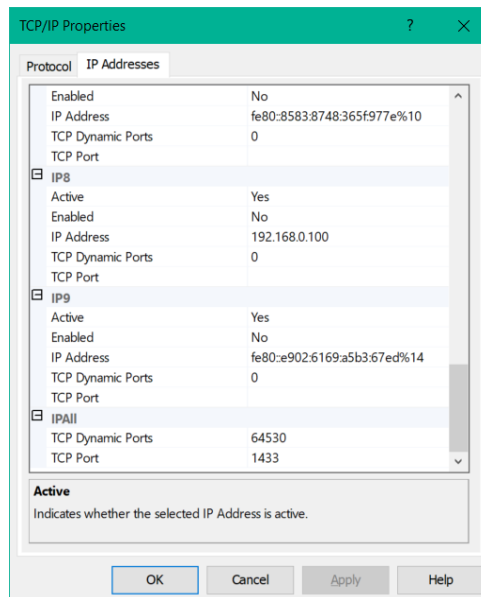


Figure 6: SQL Server TCP/IP settings

Finally, in order for these changes to take effect, we must restart the MS SQL Server service in the “Services” menu from Windows (Figure 7).

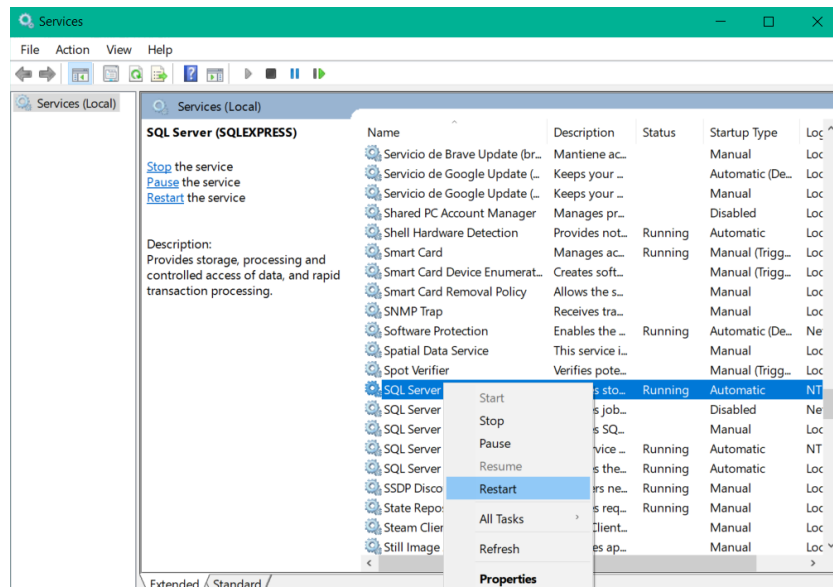


Figure 7: Services window where to restart SQL Service

3.3. Talend Components

Talend provides a series of components for processing and managing the whole ETL process. Detailed documentation on each of these components can be found at the Talend Open Studio for Data Integration Help Center ([link](#)).

Several of them were utilized within this project. Next you'll find a brief description of the most utilized for reference.

3.3.1. tMap

One of the most utilized components is tMap. It allows transformation and mapping of single or multiple sources to single or multiple destinations. You could think of it as a mapping structure that provides an interface for linking inputs with outputs, while enabling transformation features within the map itself. Following is the component's logo and mapping interface (figures have been extracted from the Talend Help Center website).

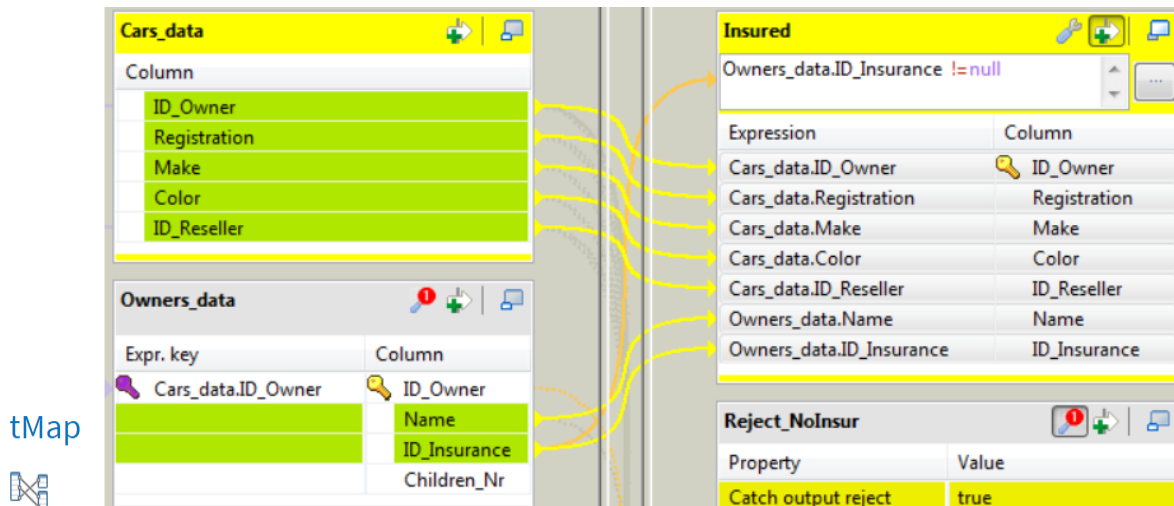


Figure 8: tMap interface example

Within the interface you're able to JOIN different inputs, generate variables that can be utilized for the output fields, set conditions for the outputs, and much more. It can be considered as the main component when working with Talend Data Integration.

3.3.2. tFileInput

There are several types of file input components depending on the file type (Delimited, XML, DB, etc...). While working with any of these components, the first and most important step is to define the schema/structure of the incoming data. Such schema will be utilized in the following components and determines the different data types. An example of the schema interface in any of the input components is as follows:

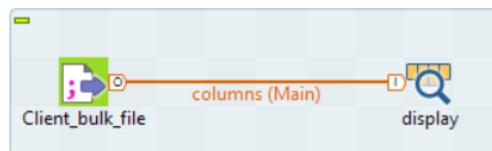
Schema of tFileInputDelimited_1									
tFileInputDelimited_1									
Column	K...	Type	<input checked="" type="checkbox"/> N..	Date Pa...	Len...	Pre...	D...	Co...	
TX_ID	<input type="checkbox"/>	Stri...	<input type="checkbox"/>						
TX_NAME	<input type="checkbox"/>	Stri...	<input type="checkbox"/>						
TX_RATE	<input type="checkbox"/>	float	<input type="checkbox"/>						

Figure 9: FileInputDelimited schema for TaxRate

Within the schema each field must have its Type, Date Pattern (if a Date type is selected), whether it can have NULL values or not (flag) and optionally you can set the Length, Precision, Default Values, and Comments. Furthermore, a Key field must be selected if eventually an output to a database is intended to be used, as it will provide the primary key over to which search for in the database table. We'll describe the tFileInputDelimited, tDBInput, and tFileInputXML.

3.3.2.1. tFileInputDelimited

tFileInputDelimited is the component utilized for importing any kind of file structure that utilizes different symbols for delimiting columns and rows (CSV, TXT, etc...). Below images represent the component logo as well as the properties.



Client_bulk_file(tFileInputDelimited_1)

Basic settings	Property Type	Repository	DELIM:Owners
Advanced settings	"When the input source is a stream or a zip file, footer and random shouldn't be bigger than 0."		
Dynamic settings	File name/Stream	"C:/Input/owners.csv" *	
View	Row Separator	"\n"	Field Separator "," *
Documentation	<input type="checkbox"/> CSV options		
Validation Rules	Header	1	Footer 0 Limit *
	Schema	Repository	DELIM:Owners - metadata *
	<input type="checkbox"/> Skip empty rows <input type="checkbox"/> Uncompress as zip file <input type="checkbox"/> Die on error		


Figure 10: tFileInputDelimited settings

tDB2Input

3.3.4.

This component is
Users need to pro
fields within the file

tFileInputXML



"DimCustomer" (tDBInput 4) (Microsoft SQL Server)

Basic settings

Database: Microsoft SQL Server

☐ Use an existing connection

Property Type: Repository

JDBC Provider: Open source JTDS

Host: localhost

Port: 1433

Schema: dbo

Database: TPCDI_Talend

Username: sa

Password: *****

Schema: Built-In

Table Name: DimCustomer

Query Type: Built-In

Query: `SELECT dbo.DimCustomer.SK_CustomerID,
dbo.DimCustomer.CustomerID,
dbo.DimCustomer.TaxID.`

Figure 12: tFileInputXML setting for DimAccount.

tFileInputXML

AccountMgmt(tFileInputXML_1)

Basic settings

Advanced settings

Dynamic settings

View

Documentation

Property Type

Built-In

Schema

Built-In

Edit schema

File name/Stream

context.staging_area_dir + "CustomerMgmt.xml"

Loop XPath query

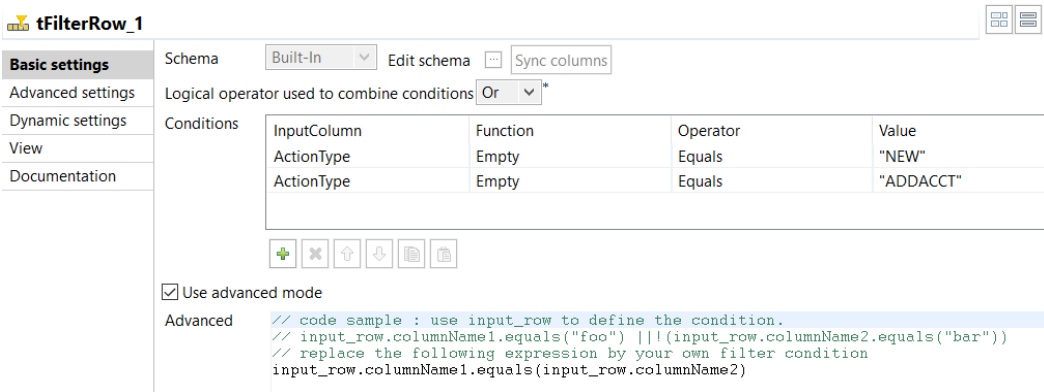
/TPCDt:Actions/TPCDt:Action

Mapping

Column	XPath query	<input type="checkbox"/> Get Nodes
AccountID	"Customer/Account/@CA_ID"	<input type="checkbox"/>
TaxStatus	"Customer/Account/@CA_TAX_ST"	<input type="checkbox"/>
AccountDesc	"Customer/Account/CA_NAME"	<input type="checkbox"/>
CustomerID	"Customer/@C_ID"	<input type="checkbox"/>
BrokerID	"Customer/Account/CA_B_ID"	<input type="checkbox"/>
ActionType	"@ActionType"	<input type="checkbox"/>
ActionTS	"@ActionTS"	<input type="checkbox"/>
Account	"Customer/Account"	<input type="checkbox"/>

3.3.5. tFilterRow

Such component allows users to filter incoming data on a specific condition. Conditions can be set with AND/OR and on multiple fields. If advanced mode enable, user can input JAVA code for filtering rather than using the interface. Example of such component provided below.



The screenshot shows the configuration window for the **tFilterRow_1** component. It features a sidebar with tabs: **Basic settings**, **Advanced settings**, **Dynamic settings**, **View**, and **Documentation**. The **Basic settings** tab is active, showing a **Schema** dropdown set to **Built-In**, with **Edit schema** and **Sync columns** buttons. Below this, the **Logical operator used to combine conditions** is set to **Or**. A **Conditions** table is displayed with two entries:

InputColumn	Function	Operator	Value
ActionType	Empty	Equals	"NEW"
ActionType	Empty	Equals	"ADDACCT"

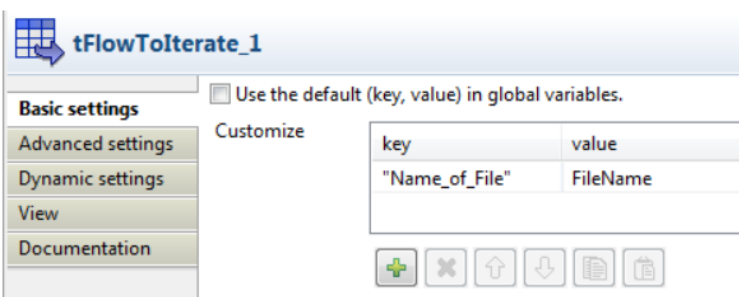
Below the table are icons for adding, deleting, and moving conditions. The **Use advanced mode** checkbox is checked. The **Advanced** section contains a code sample:

```
// code sample : use input_row to define the condition.
// input_row.columnName1.equals("foo") || (input_row.columnName2.equals("bar"))
// replace the following expression by your own filter condition
input_row.columnName1.equals(input_row.columnName2)
```

Figure 13: tFilterRow setting for DimAccoun.ActionType field

3.3.6. tFlowTolterate

This component generates a data flow table out of an input to iterate over each of the rows individually rather than processing them as a batch/bulk. Furthermore, it generates a list of global variables out of the input schema, which can be used in the subsequent components. A good example is defining a global variable for one of the IDs incoming, which you're able to use later as a condition for querying a database and retrieving its most up-to-date status or so. Example of the component settings and icon below.



The screenshot shows the configuration window for the **tFlowTolterate_1** component. It features a sidebar with tabs: **Basic settings**, **Advanced settings**, **Dynamic settings**, **View**, and **Documentation**. The **Basic settings** tab is active, showing a checkbox for **Use the default (key, value) in global variables.** Below this, the **Customize** section contains a table:

key	value
"Name_of_File"	FileName

Below the table are icons for adding, deleting, and moving customizations.

Figure 14: tFlowTolterate example settings

3.3.7. tFixedFlowInput

This component works together with the tFlowTolterate and generates a fixed flow from internal variables. Incoming data from the tFlowTolterate will be mapped to global variables defined within this component and process the data table generated. Such definition of the global variables need to be defined within the components settings, referencing the input flow name. Example displayed below.

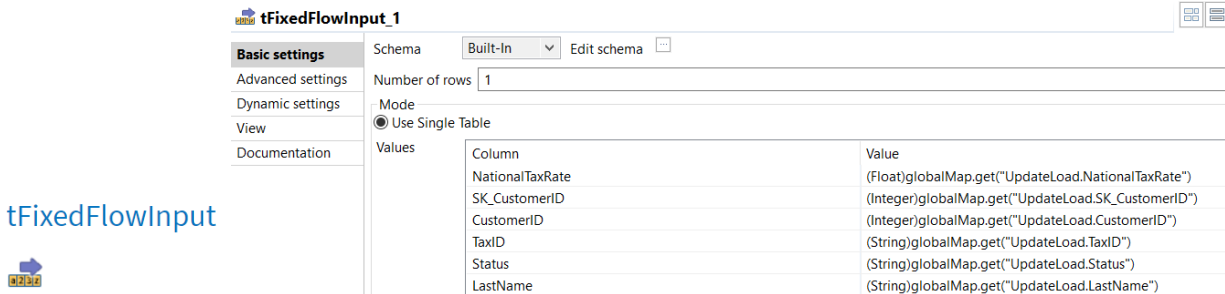


Figure 15: tFixedFlowInput setting for DimCustomerHistoricLoad job

3.3.8. tFileList

The tFileList component allows the user to fetch several files and iterate feed them to other components to iterate over them. In order to retrieve and load the files, it is done by giving a path where those files are (and, optionally, apply a mask to discard files that are not wanted). In our case, and as is shown in Figure 16, this component has been used for all the jobs in which the FINWIRE files are involved. The mask used is a regular expression that filters files with a name of the form "FINWIRE", a year (four digits), "Q" and another digit.

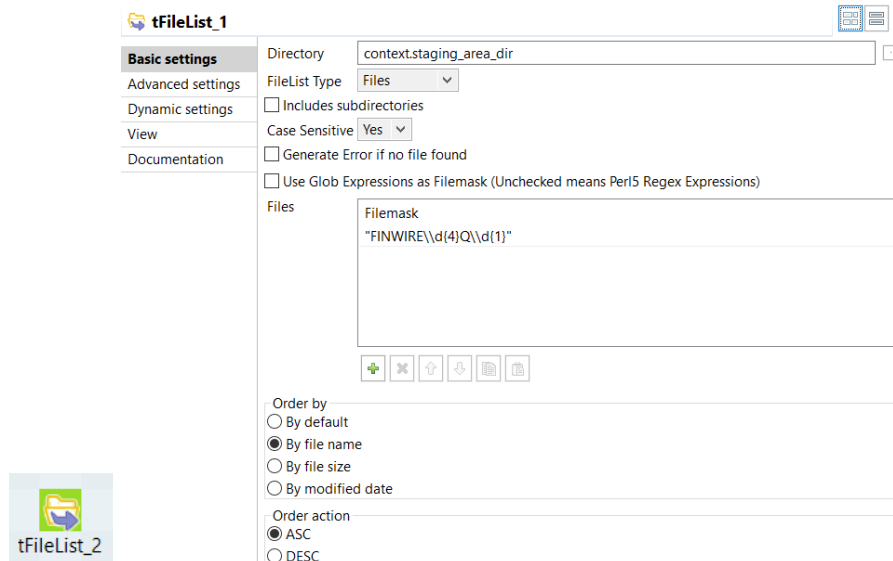
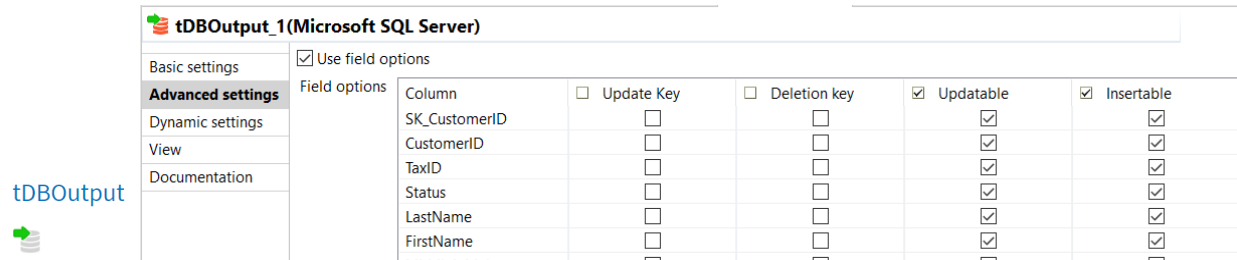


Figure P16: Detail of the tFileList component

3.3.9. tDBOutput

This component allows data loading into a specific database table. Input schema will be mapped to the DB schema (need to be identical) and fields will be Inserted/Updated depending on the setting set for the component. It also allows customization in the scenario where not all of the fields should be inserted or updated, but rather just a couple of them. A key must be defined as well so that the component knows what's the primary key to look for in the DB table. This is a generic DB output component in which user can specify the DB type corresponding. There are several others utilized for specific DB types (MySQL, MSSQL, Oracle, etc...).



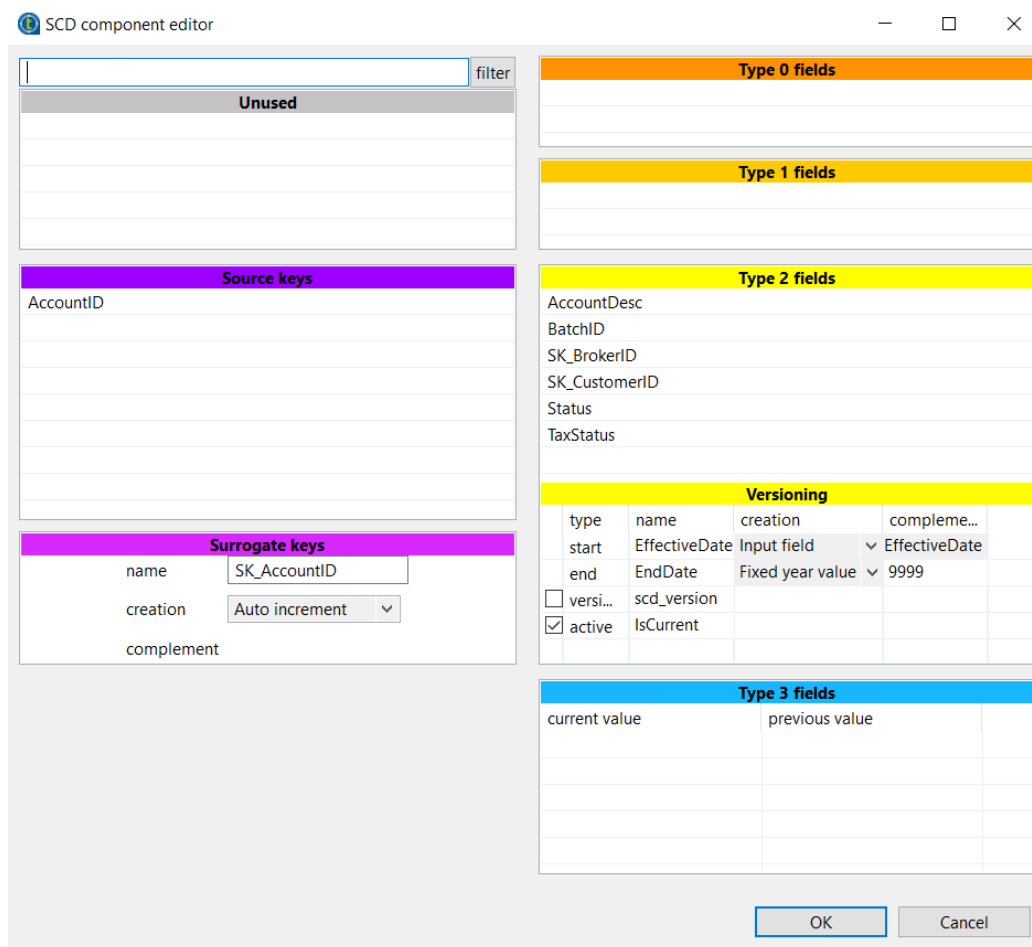
The image shows the 'tDBOutput_1(Microsoft SQL Server)' configuration window. The 'Advanced settings' tab is selected, showing 'Field options'. A table lists columns and their properties:

Column	Update Key	Deletion key	Updatable	Insertable
SK_CustomerID	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CustomerID	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
TaxID	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Status	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
LastName	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
FirstName	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 17: tDBOutput settings in DimCustomerHistoricLoad job

3.3.10. tDBSCD

Very similar to the tDBOutput, this table outputs data into a specific database table. The difference is that this component is utilized for tracking changes, which allow for historic records to be inserted and updated at the same time. It includes an interface called SCD editor, which provides several fields for defining active/end dates, version management, status, surrogate key and fields to update. More information on the Type # fields described on the Talend Help Center. Below an example of such an interface is displayed.



The image shows the 'SCD component editor' window. It contains several sections for configuring the SCD component:

- Unused:** A list of unused fields.
- Source keys:** A list of source keys, currently containing 'AccountID'.
- Surrogate keys:** A section for defining surrogate keys with fields for name, creation, and complement.
- Type 0 fields:** A section for defining Type 0 fields.
- Type 1 fields:** A section for defining Type 1 fields.
- Type 2 fields:** A section for defining Type 2 fields, currently containing 'AccountDesc', 'BatchID', 'SK_BrokerID', 'SK_CustomerID', 'Status', and 'TaxStatus'.
- Versioning:** A section for defining versioning fields with a table:

type	name	creation	compleme...
start	EffectiveDate	Input field	EffectiveDate
end	EndDate	Fixed year value	9999
versi...	scd_version		
active	IsCurrent		

- Type 3 fields:** A section for defining Type 3 fields with columns for 'current value' and 'previous value'.

Figure 18: SCD editor for tDBSCD component in DimCustomerHistoriLoad job

3.4. Import File Structure

The TPC-DI framework provides a set of files that are used for the extraction phase of the ETL process. Such set of files come in different formats (XML, CSV, TXT, etc...) and thus, require a specific type of transformation for the tool processing. The documentation provides metadata regarding the type of data each of the files has, as well as the dependency between each other when required. An example of the data format and type is provided below.

Base Type	Input formatting
BOOLEAN	"0" for False and "1" for True.
CHAR(n)	Character string of up to n single-byte characters.
DATE	Formatted as "YYYY-MM-DD", were YYYY is the year, MM is the month number and DD is the day number. MM and DD will have leading zeroes when appropriate.
DATETIME	Formatted as "YYYY-MM-DD HH:MM:SS", were YYYY is the year, MM is the month number, DD is the day number, HH is the hour, MM is the minute, and SS is the second in 24-hour format. Each part of the value will have leading zeroes when appropriate.
NUM(m[,n])	Unsigned numeric value with at most m total Digits, of which up to n Digits are to the right (after) the decimal point. The length does not exceed m+1 characters, including the decimal point.
SNUM(m[,n])	Signed numeric value with an optional "+" or "-" followed by at most m total Digits, of which up to n Digits are to the right (after) the decimal point. The length does not exceed m+2 characters, including the sign and decimal point. If there is no sign, the value is positive.

Figure 19: Data type definitions from source data files

Besides such data type definitions, there are a couple other metadata structures defined for specific tables. Such provide more detailed understanding on what those fields represent for the table. Table below provides those field definitions.

Meta Type	Base Type	Usage / Restrictions
BALANCE_T	SNUM(12,2)	Aggregate account and transaction related values such as account balances, total commissions, etc.
		account balances, total commissions, etc.
CDC_FLAG_T	CHAR(1)	"I", "U" or "D" for insert, update or delete
CDC_DSN_T	NUM(12)	Database Sequence Number, a monotonically increasing value
IDENT_T	NUM(11)	Numeric identifiers
S_COUNT_T	NUM(12)	Aggregate count of shares
S_PRICE_T	SNUM(8,2)	Share prices
S_QTY_T	NUM(6)	Quantity of shares for an individual trade
TRADE_T	NUM(15)	Trade identifiers
VALUE_T	SNUM(10,2)	Non-aggregated transaction and security related values such as cost, dividend, etc.

Figure 20: Meta-type definitions from source data files

3.5. Jobs

Within the Github repository, you're able to find two folder structures containing all the jobs regarding the data processing with Talend, as well as the SQL scripts utilized for generating the DW ([link](#)).

The screenshot shows the Github repository interface for 'TPC-DI Benchmark - Talend Open Studio for DI and MS SQL Server'. At the top, there are navigation tabs: Code, Issues (0), Pull requests (0), Actions, Projects (0), Wiki, Security, and Insights. Below the repository name, statistics are shown: 24 commits, 1 branch, 0 packages, 0 releases, and 2 contributors. A bar contains buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a green 'Clone or download' button. The commit history table shows:

Commit	Message	Time
eseuteo	Added jobs for FactWatches and for FactMarketHistory	Latest commit 8004c04 1 minute ago
process	Added jobs for FactWatches and for FactMarketHistory	1 minute ago
sql-server	changed dividends from int to numeric(10,2)	11 days ago
README.md	Initial commit	last month

Figure 21: Project Github repository

The “process” folder contains the jobs for all reference tables, dimensions and fact tables. Each of the folders has 3 files generated by Talend that need to be placed on the corresponding workspace folder in order for importing them into the tool.

This screenshot shows the 'process' folder within the 'TPCDI-talend-MSSQL' repository. The breadcrumb path is 'Branch: master / TPCDI-talend-MSSQL / process /'. The file list includes:

File	Type	Time
..		
DimAccount	Dimensions	11 days ago
DimBroker	Dimensions	11 days ago
DimCustomerHistoricNew	Dimensions	11 days ago
DimCustomerHistoricUpd	Dimensions	11 days ago
DimDate	Dimensions	11 days ago

Figure 22: Jobs Github location

This screenshot shows the files inside the 'DimAccount' folder. The breadcrumb path is 'Branch: master / TPCDI-talend-MSSQL / process / DimAccount /'. The file list includes:

File	Type	Time
jhuete	Dimensions	Latest commit ea2b293 11 days ago
..		
DimAccount_v3_0.1.item	Dimensions	11 days ago
DimAccount_v3_0.1.properties	Dimensions	11 days ago
DimAccount_v3_0.1.screenshot	Dimensions	11 days ago

Figure 23: Talend project files for DimAccount

In the following section, the details regarding how Talend was configured and what jobs were implemented in order to perform the DI tasks are provided. The jobs are divided into three sections, following the DW schema available in the TPC-DI specifications (Figure 24).

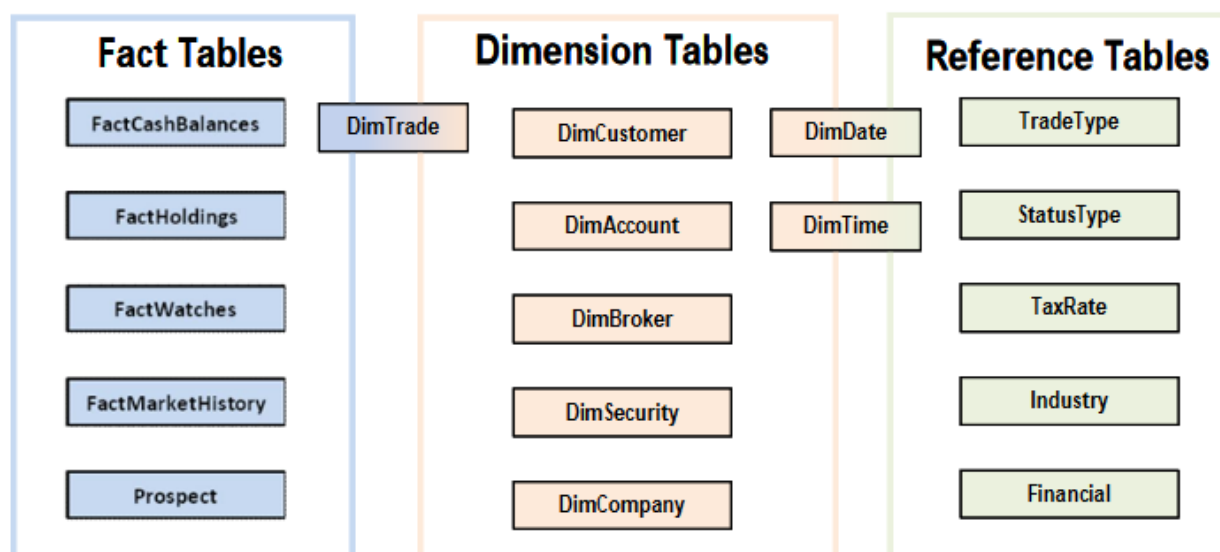


Figure 24: DW graphical depiction provided by the TPC-DI specifications document

3.5.1. Reference tables

First set of tables generated within the benchmark are “Reference Tables”. They are utilized within the dimensions to extract specific values for data references. Rather than managing changes of those values within every dimension, the reference tables allow users to handle maintenance within those specific tables and updates to the subsequent ones will be automatic. As can be expected, there are no incremental updates for any of the Reference Tables. Next the design for generating such tables is presented.

3.5.1.1. TradeType

The job corresponding to this table is the one called “loadTradeType”. As can be seen in Figure 25, it is rather straightforward, being composed by just one tFileInputDelimited which reads the TradeType.txt file in Batch1 folder, a tMap component and a tDBOutput component.

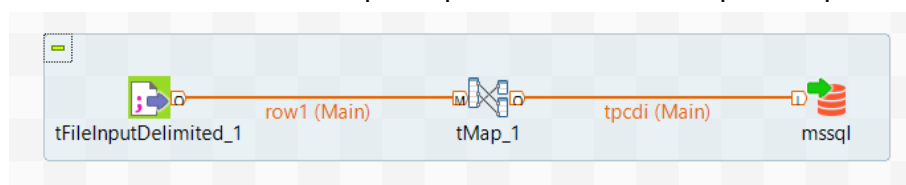


Figure 25: loadTradeType job in Talend

3.5.1.2. StatusType

Similarly to the previous job, loadStatusType is also simple and has the same components. The input file in this case is StatusType.txt in Batch1 directory. Figure 26 shows the tMap component for this job.

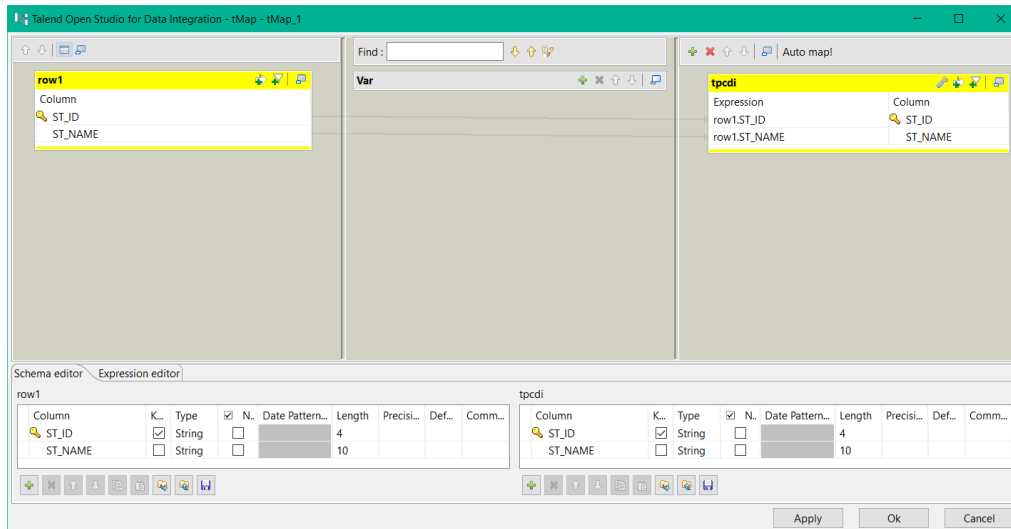


Figure 26: tMap component for loadStatusType job

3.5.1.3. TaxRate

Similarly to the previous jobs, TaxRate is rather straightforward, and no additional details will be included in the report.

3.5.1.4. Industry

Similarly to the previous jobs, loadIndustry is rather straightforward, and no additional details will be included in the report.

3.5.1.5. Financial

Prior to any work done regarding this job, it is necessary to create the job corresponding to DimComany table and populate it, as loadFinancial utilizes this table for lookup. The loadFinancial is the most complicated job from those corresponding to reference tables. Figures 27 and 28 show the two main subjobs that compose loadFinancial. As can be seen, the tFileList component iterates over the FINWIRE files, then the rows with the "FIN" flag are selected with the tFilterRow_1 component.

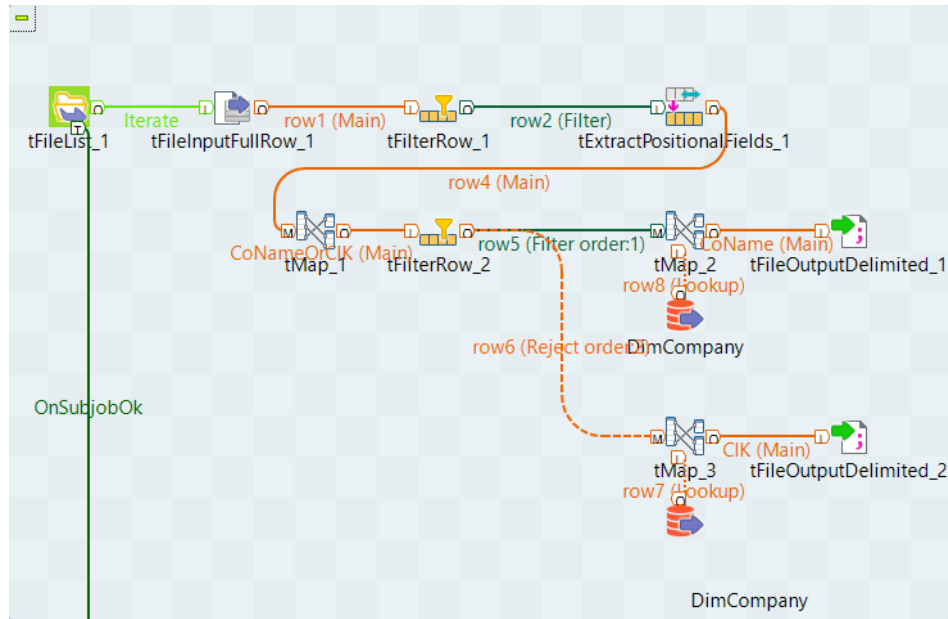


Figure 27: loadFinancial job (1/2)

After this, the different fields are extracted with tExtractPositionalFields. Then, the tMap_1 component generates the CIK or the CompanyName fields in the manner shown in Figure 29.

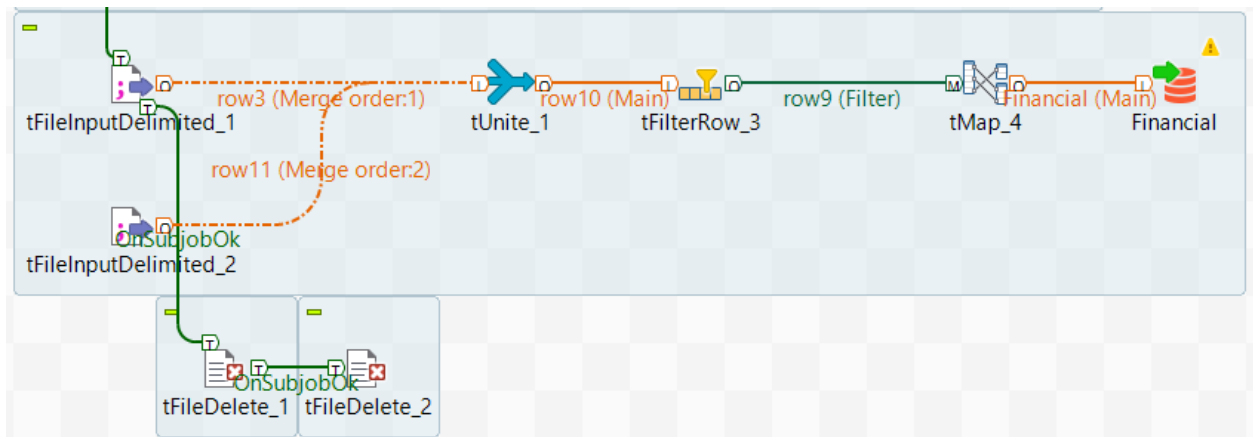


Figure 28: loadFinancial job (2/2)

Afterwards, depending on the value for the CIK field (null or not), the Financial rows will be stored in two different auxiliary files. tMap_2 details are shown in figure 30. As can be seen, the Company surrogate key is obtained by looking up the name of the company (in the other tMap component, the CIK of the company is used for this purpose).

Var		
Expression	Type	Variable
Mathematical.NUM(row4.CoNameOrCIK) == 1 ? Mathematical.INT(row4.CoNameOrCIK) : null	Integer	<input checked="" type="checkbox"/> CIK
Mathematical.NUM(row4.CoNameOrCIK) == 0 ? row4.CoNameOrCIK.trim() : null	String	<input checked="" type="checkbox"/> CoName

Figure 29: tMap_1 detail in which CIK and CoName are generated depending on the value of CoNameOrCIK field

Figure 28 shows the second part of the loadFinancial job. The files generated in the first part are now merged and stored into the Financial table. In the end, the auxiliary files created in the first part of the job are deleted.

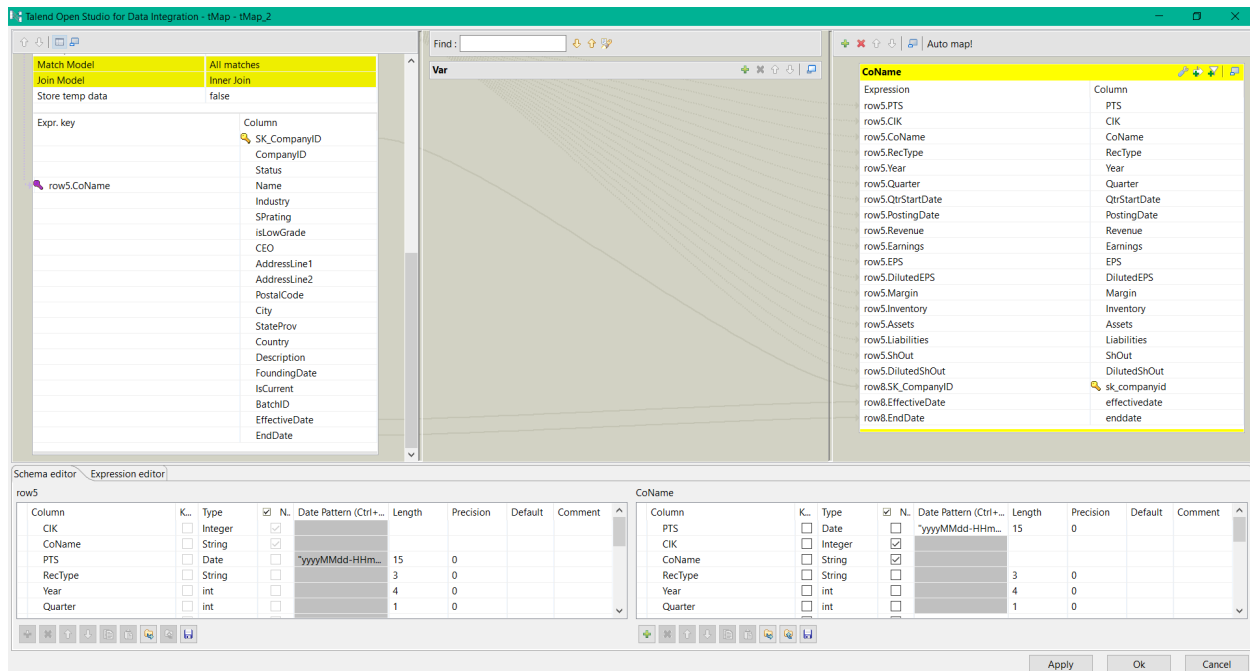


Figure 30: Details regarding the mapping component (tMap_2) for loadFinancial, including a lookup to the DimCompany table.

3.5.2. Dimension tables

3.5.2.1. DimCustomer

Data to populate such dimension is extracted from the CustomerMgmt.xml file. Within it there's several transactions related to the customer:

ActionType	Description
NEW	A new customer. A new customer is always created with 1 or more new accounts.
ADDACCT	One or more new accounts for an existing customer. This does not require any change to DimCustomer.
UPDACCT	Updates to the information in one or more existing accounts. No change to DimCustomer is required.
UPDCUST	One or more updates to existing customer's information. Only the identifying data and updated property values are supplied in the Source Data .
CLOSEACCT	Close one or more existing accounts. This does not require any change to DimCustomer.
INACT	Make an existing customer and that customer's currently active accounts inactive. No specific account information is supplied in the Source Data .

Figure 31: CustomerMgmt.xml action types

These action types define what type of output will each of the rows generate and how it needs to be handled. Important to note that the action types are not only related to the DimCustomer dimension, but also to DimAccount. All transactions must be processed in sequential order so that the information is up-to-date. For processing such dimension, two Talend jobs were generated.

3.5.2.1.1. DimCustomer_HistoricLoad

This job generates all the new customer records within the DimCustomer table in the DW. It specifically processes those records with the ActionType = "NEW". Diagram for such dimension is shown below.

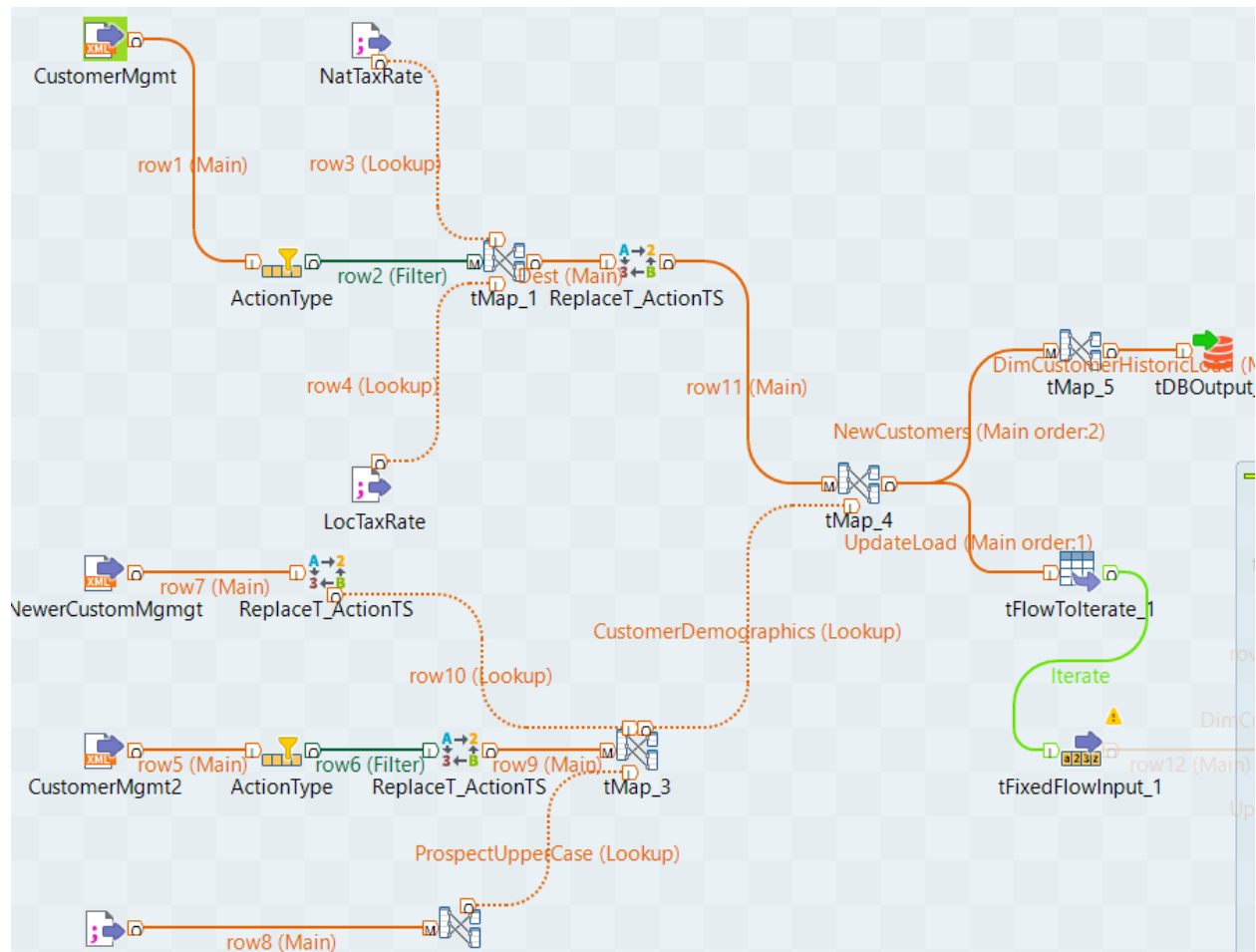


Figure 32: DimCustomer job diagram

This dimension also required information regarding the Customer Demographics, which needs to be extracted from the Prospect.csv file and joined by certain conditions with the CustomerMgmt.xml file. Information extracted from the Prospect file is presented on Figure 33.

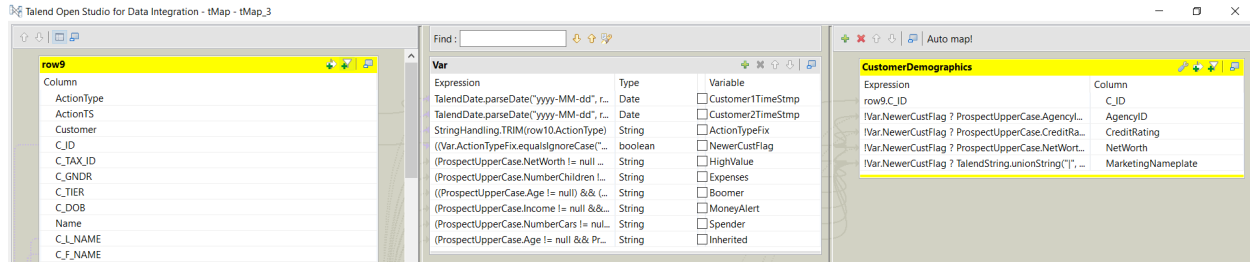


Figure 33: Customer Demographics tMap on DimCustomerHistoricLoad job

AgencyID, CreditRating, NetWorth, and MarketingNamePlate are generated and joined with the corresponding customer record based on the first name, last name, address1, address2 and postal code. Eventually, the records are uploaded into the DimCustomer table in the database with a tDBOutput component.

3.5.2.1.2. DimCustomer_UpdateLoad

This job works in the same way as the previous one, except it only processes the records with ActionType="UPDUCST"/"INACT". Such records are loaded into the database using a tDBSCD component for keeping the historic records and updating them. While processing the updates, both the DimCustomer and DimAccount tables are queried to retrieve the current values of those specific customers/accounts. Diagram of such job is presented below...

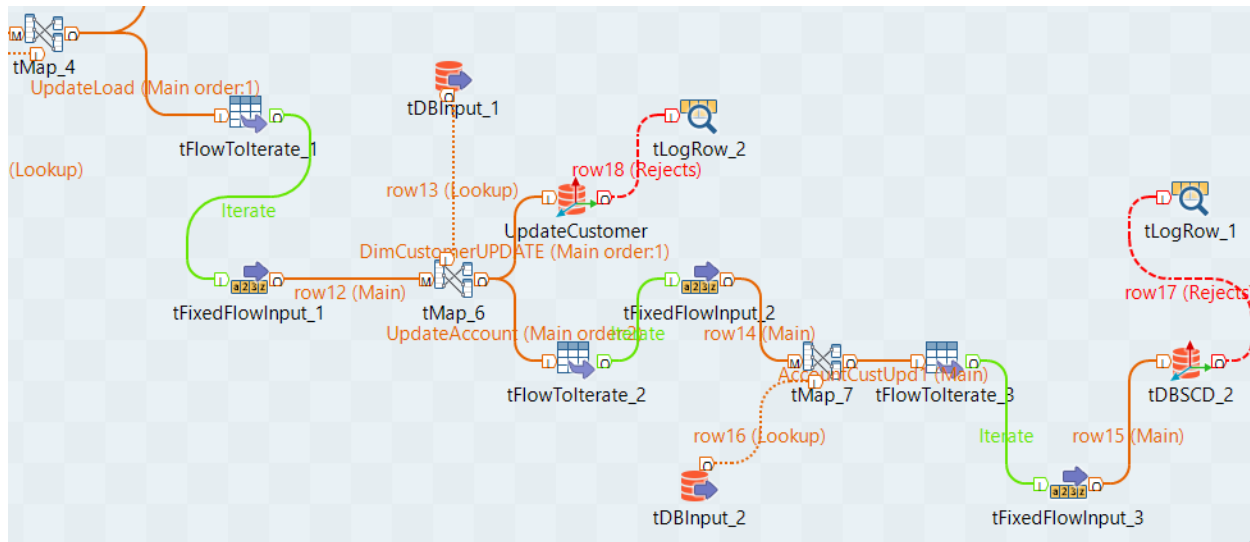


Figure 34: DimCustomer_Updated Load job diagram

While an update to a customer is done, the corresponding accounts referencing that customer need to be updated as well with the new SK_CustomerID value. Thus, two tDBSCD components had to be created for processing the same flow of records. Two tFlowTolterate and tFixedFlowInputs were used as well to generate the global variables used in the queries for the tDBInputs of both DimCustomer and DimAccount.

For the field EndDate, as the Talend SCD component does not allow us to set the specific date '31-12-9999', this is changed by means of a trigger in the database. The same procedure is done also for the tables DimAccount, DimSecurity, and DimCompany.

3.5.2.2. DimAccount

This dimension is populated from the CustomerMgmt.xml file, but referencing ActionType="NEW"/"ADDACT"/"UPACCT"/"CLOSEACT". Every time a new customer is created, the corresponding account is generated as well. Because of the accounts generated refer to a customer, the DimCustmer_HistoricLoad is required to be executed previous to the DimAccount job execution. Diagram of the job displayed below...

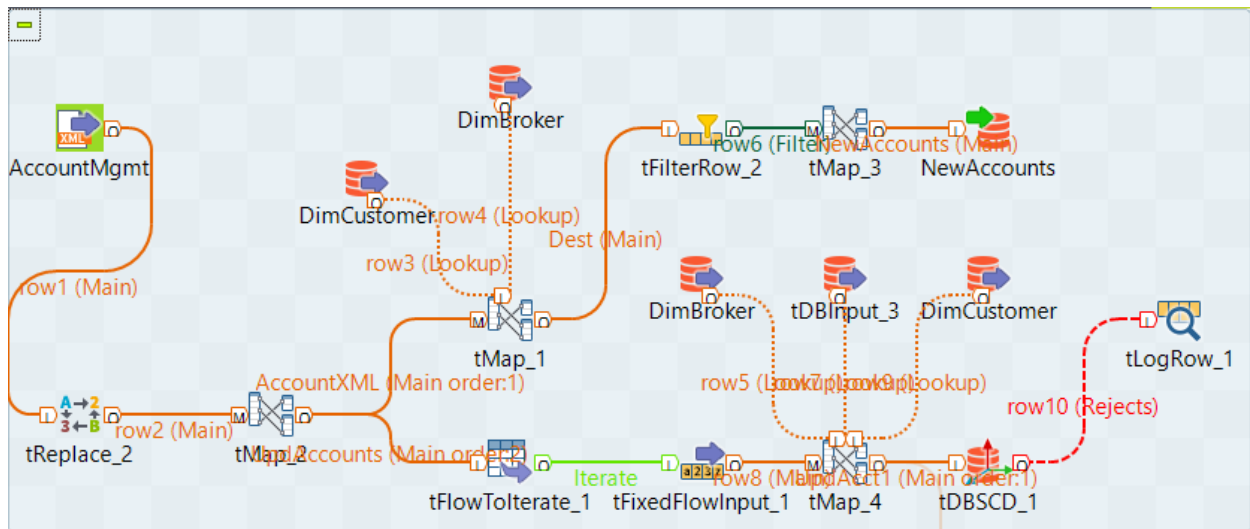


Figure 35: DimAccount job diagram

Within the mapping, surrogate keys for the corresponding broker and customer are pulled and mapped by the active CustomerID and BrokerID respectively. This dimension is also a historic tracking dimension, thus a tDBSCD component is used for handling the outputs, as well as a tDBOutput for the new accounts. Mapping of such dimensions is displayed below...

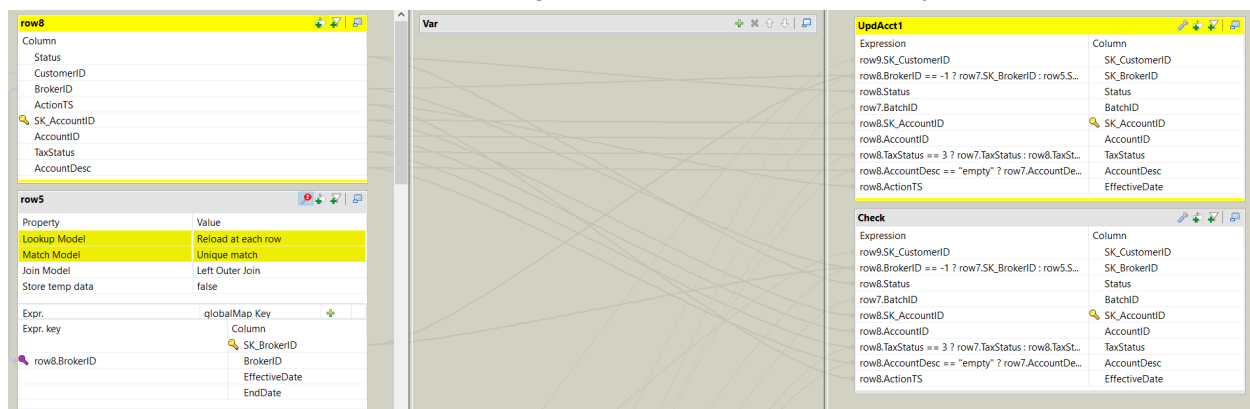


Figure 36: DimAccount job mapping on tMap

3.5.2.3. DimBroker

Data for this dimension is extracted from the HR.csv file. It also requires that the DimDate is generated previously so that the DateValue reference can be extracted. Diagram for this job presented below...

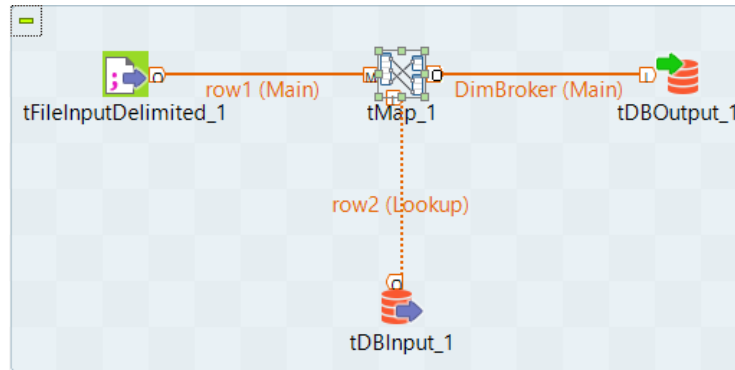


Figure 37: DimBroker job mapping on tMap

3.5.2.4. DimDate

This is the base date dimensions utilized for references within several other dimensions. It's generated from the Date.txt file and contains dates from 1950-01-01 until 2020-12-31 (YYYY-MM-DD). Design of this job is pretty straightforward and no additional details will be shown in this report.

3.5.2.5. DimTime

Similar to DimTrade, this dimension represents the times utilized and referenced by other dimensions. Data is extracted from the Time.txt file and values present are from 00:00:00 till 23:59:59 (HH:MM:SS). Design of this job is pretty straightforward and no additional details will be shown in this report.

3.5.2.6. DimTrade

Data for this dimension is extracted from Trade.txt and TradeHistory.txt files. Such files store the trades generated, as well as the historic updates for each one. This is not a historic dimension, thus the updates over the trades are done over the same records and no surrogate keys are required or generated. It requires several other dimensions to be generated, as it works similar to a fact table (DimSecurity, DimAccount, StatusType, TradeType, DimDate, and DimTime). Because of this dependency and the need to reference previous trades generated and already located in the database, the flow design of the job is processed on a one-by-one basis. Diagram of such dimension presented below...

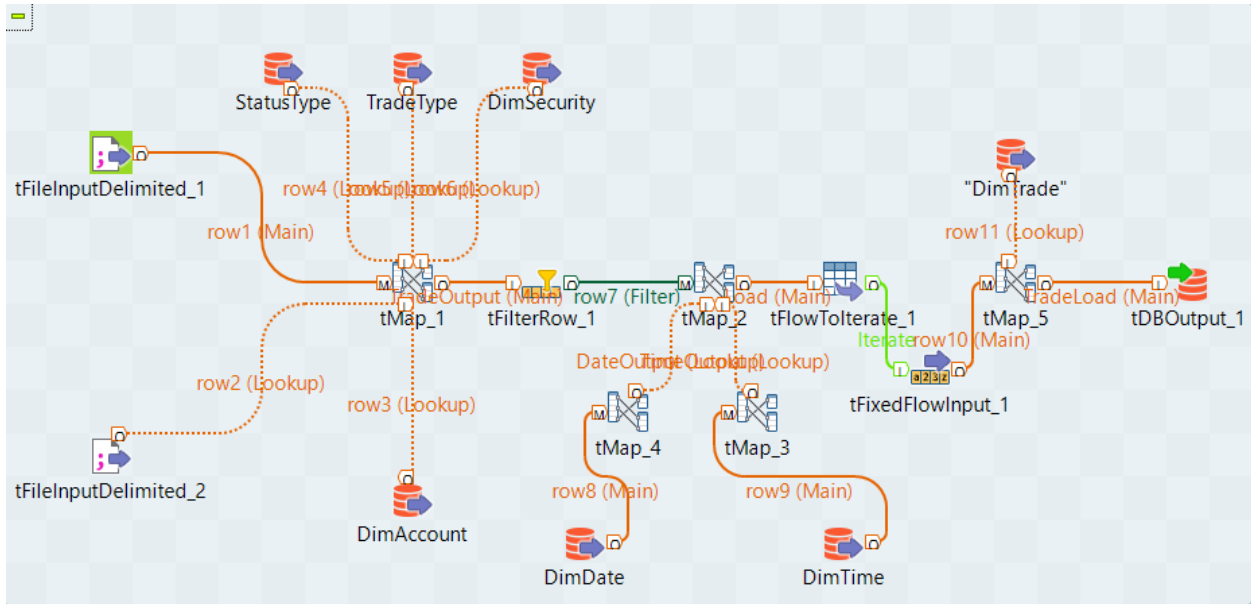


Figure 38: DimTrade job diagram

After mapping the trades with their corresponding account, status, trade type, company, and security references, it then looks for the active trade and updates its status, as well as the close date/time. Map utilized by the dimension is presented on the image below...

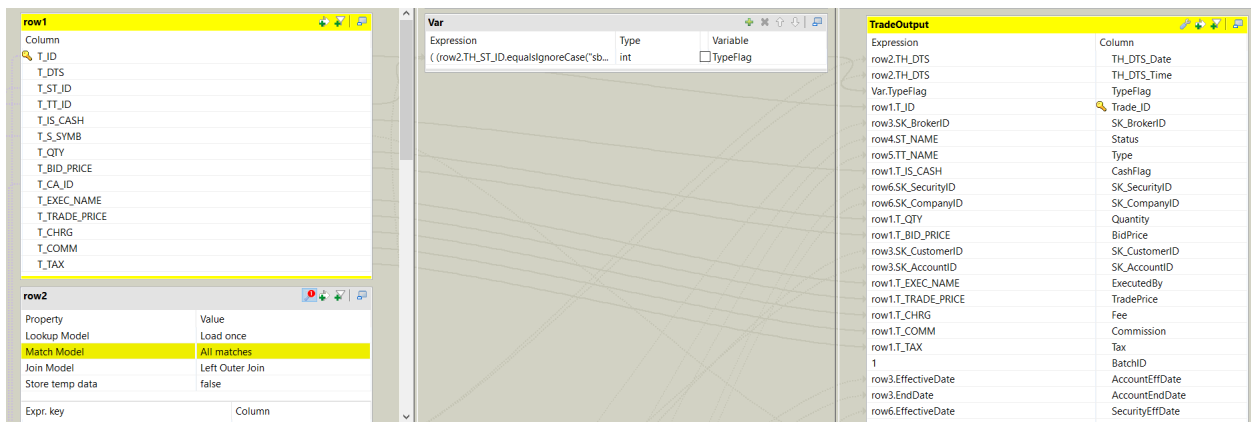


Figure 39: tMap for DimTrade job

Execution of such dimension took a lot of time due to the amount of records of both the Trade and TradeHistory file. End of execution was not achieved after more than 18hours of continuous running time.

3.5.2.7. DimSecurity

Data for the DimSecurity table is available in the FINWIRE files. As shown in Figure 40, after loading the rows and filtering the ones corresponding to Security, they are filtered regarding the length of the CoNameOrCIK field and inserted in an analogous way to the DimSecurity table.

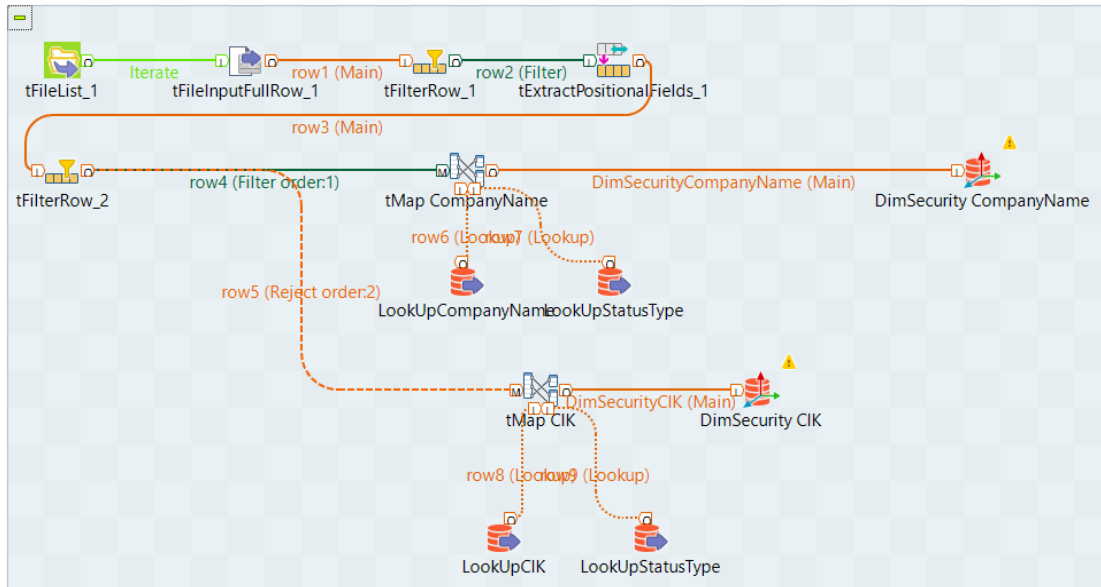


Figure 40: Job corresponding to DimSecurity table

In order to insert data to the DimSecurity table, an SCD component is used (Figure 41). Talend does not allow to fix a custom year value, and the date 01-01-9999 has to be put as the value for the EndDate of active rows. Finally, regarding incremental updates, no changes need to be done in this job.

Versioning			
type	name	creation	compleme...
start	EffectiveDate	Input field	EffectiveDate
end	EndDate	Fixed year value	9999
<input type="checkbox"/> versi...	scd_version		
<input checked="" type="checkbox"/> active	IsCurrent		

Figure 41: Details regarding the tDBSCD component for DimSecurity

3.5.2.8. DimCompany

Similarly to DimSecurity, data for populating DimCompany is obtained from FINWIRE files. After processing them, the rows are filtered. In case they do not have a proper SPRating, a message is stored in the DImessages table. In both cases they are stored in DimCompany table through the SCD components (Figure 42).

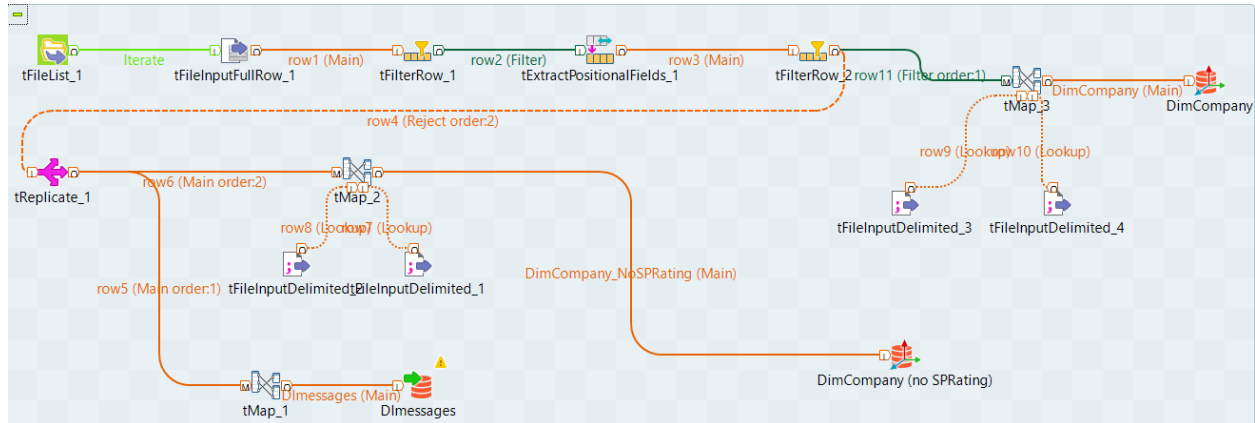


Figure 42: Job corresponding to DimCompany table

Regarding incremental updates, and as is stated in the TPC-DI benchmark specifications, no changes need to be done in DimCompany.

3.5.3. Fact tables

3.5.3.1. FactCashBalances

Data for this fact is extracted from the CashTransaction.txt file. Transactions are processed and the cash balance is aggregated to keep the historic balance over time. Such fact requires reference to DimAccount and DimDate to store the values of the account's transactions and the dates they occurred. Diagram for such transaction presented below...

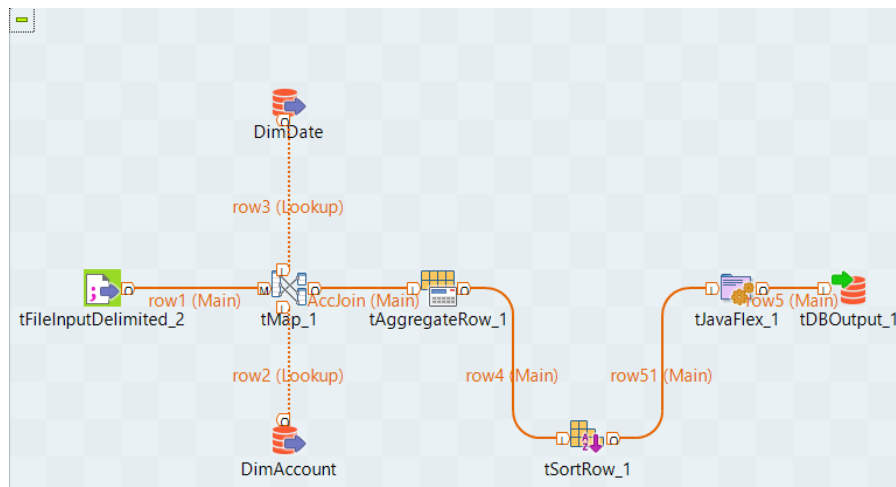


Figure 43: FactCashBalances job diagram

3.5.3.2. FactHoldings

Figure 44 contains the Talend job for populating the FactHoldings table. It is a rather simple job, which includes a lookup to the DimTrade table. Regarding the jobs corresponding to the incremental updates, they are similar, and the elements that are different belong to the tMap component.

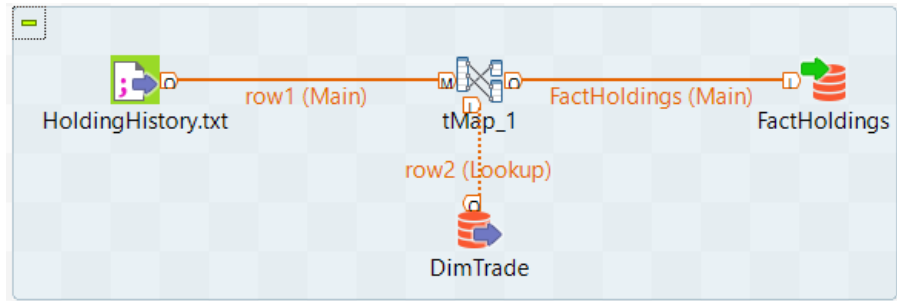


Figure 44: Job corresponding to FactHoldings table

3.5.3.3. FactWatches

The job corresponding to the FactWatches table is available in Figure 45. As can be seen, three different lookups are necessary in this case.

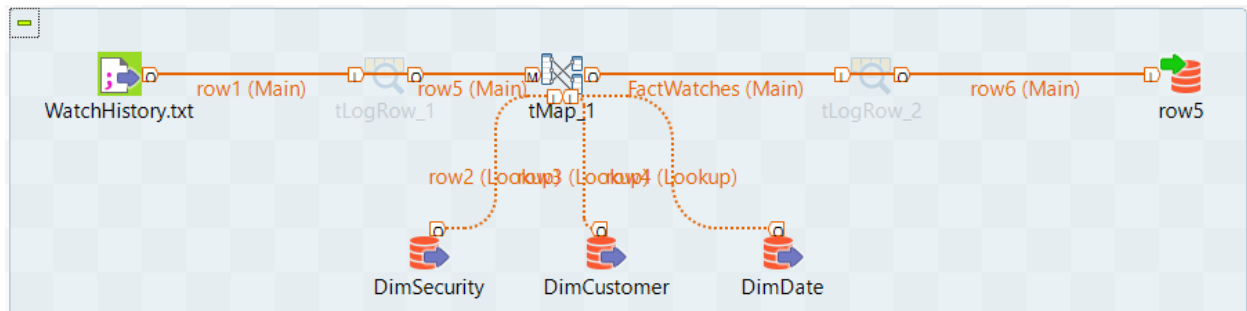


Figure 45: Talend job for FactWatches table population

3.5.3.4. FactMarketHistory

FactMarketHistory job depiction is provided in Figure 46. As it shows, there are a handful of lookups to database tables. Besides, an aggregation component was needed in order to obtain the maximum and minimum value for Daily Market highs and lows, respectively. Regarding including a lookup to the eps value from the last 4 quarters in the FINWIRE files, we could not finally implement that part of the job, and therefore the correctness of the job for FactMarketHistory could not be ensured.

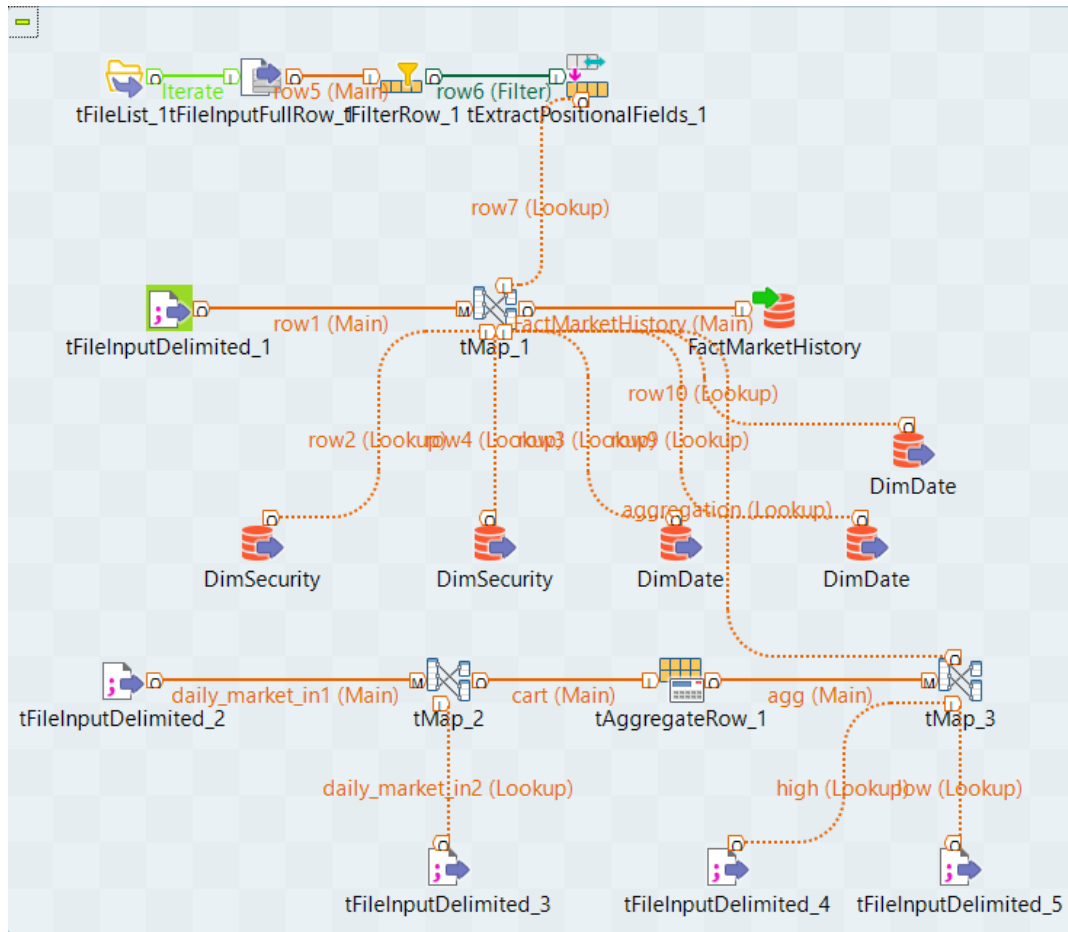


Figure 46: Job corresponding to FactMarketHistory table

3.5.3.5. Prospect

Data required for this fact table is extracted from the Prospect.csv file. It requires the DimCustomer to be executed to reference the corresponding data of the customer and generate their records. It verifies whether there's an active customer for that record generated or not to set the IsCustomer flag correspondingly. Diagram for such job presented below...

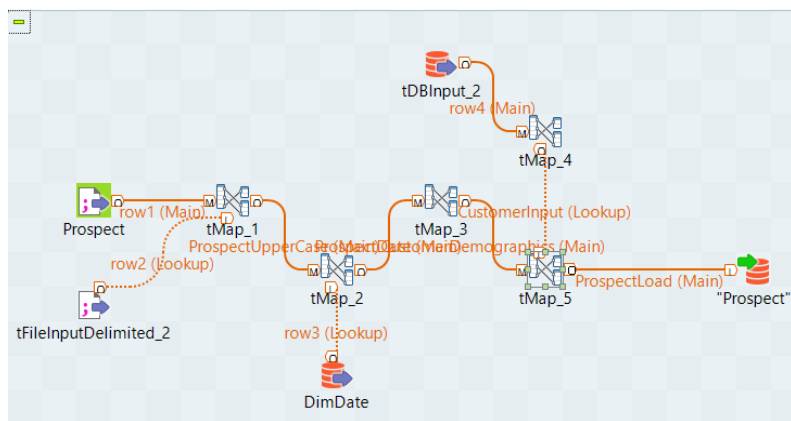


Figure 47: FactProspect job diagram

4. Results and conclusions

In the following section, the results obtained, mainly regarding the running times of the jobs implemented, will be shown, as well as some conclusions on the suitability of the Talend Open Studio for DI tool.

4.1. Benchmark results

Table 1 shows the time taken to populate the Reference tables.

Table	Time elapsed on population	Rows inserted
TradeType	875 milliseconds	5
StatusType	859 milliseconds	6
TaxRate	779 milliseconds	320
Industry	875 milliseconds	102
Financial	23358 milliseconds	180622

Table 1: Results for jobs that populate Reference tables

Table 2 shows the time taken to populate the Dimension tables.

Table	Time elapsed on population	Rows inserted
DimCustomer	2257081milliseconds	10810
DimAccount	5197220 milliseconds	21560
DimBroker	1654 milliseconds	25000
DimDate	2914 milliseconds	25933
DimTime	9082 milliseconds	86400
DimTrade	*Did not finish running	*Did not finish running
DimSecurity	90709 milliseconds	4000
DimCompany	103149 milliseconds	2500

Table 2: Results for jobs that populate Dimension tables

Table 3 shows the time taken to populate the Fact tables.

Table	Time elapsed on population	Rows inserted
FactCashBalances	15135 milliseconds	549361
FactHoldings	*Could not be run	*Could not be run
FactWatches	68530 milliseconds	1345805
FactHistoryMarket	*Could not be run	*Could not be run
Prospect	4546 milliseconds	24970

Table 3: Results for jobs that populate Reference tables

*DimTrade was not able to complete its execution, thus not providing the data necessary to run FactHolding.

4.2. Conclusions

Talend Open Studio for Data Integration is a well positioned as one of the leaders in the market for ETL, as stated by Gartner in their 2019 Gartner Magic Quadrant for Data Integration Tools [11]. With a list of different components that enable the processing of different source files into a variety of databases, it's one of the obvious options to utilize when thinking about ETL.

Even though its capabilities are resourceful and provides a lot of documentation regarding how each of the components, and the tool itself work, there's a few scenarios in which the tool is not as robust as others to process data as needed. One of those that has been encountered is when historic tracking records have to be compared to new records and some type of aggregation or such has to be done. Those types of cases where the data is processed in a single flow one-by-one requires more processing time and may take a significant amount of time depending on the size of the incoming data.

Regardless, it's a powerful tool that provides a very user-friendly interface to generate JAR files without having to be an expert in JAVA itself.

5. References

- [1] Lenzerini, Maurizio. "Data integration: A theoretical perspective." *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2002.
- [2] <https://www.talend.com/resources/what-is-data-integration/>
- [3] Perks, Ionut, Kulkarni, Xu, and Yang (June, 2019). Reference Architecture: Microsoft SQL Server. Retrieved October 27, 2019 from <https://lenovopress.com/lp1075.pdf>
- [4] Hanson, Larson and Price (2012). Columnar Storage in SQL Server 2012. Retrieved October 27, 2019 from <https://pdfs.semanticscholar.org/1d56/861b1c4736dc170ce62439cf2cf05533c560.pdf>
- [5] Microsoft Corporation (2017). SQL Server 2017 Datasheet. Retrieved October 28, 2019 from http://download.microsoft.com/download/F/9/A/F9A1B5AA-D57C-4B4D-9C3E-715B800B0419/SQL_Server_2017_Datasheet.pdf
- [6] Microsoft Corporation (n.d.) [MS-TDS]: Tabular Data Stream Protocol. Retrieved October 28 from https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-tds/b46a581a-39de-4745-b076-ec4dbb7d13ec
- [7] Bowen, Jonathan. *Getting Started with Talend Open Studio for Data Integration*. Packt Publishing Ltd, 2012.
- [8] Barton, Rick. *Talend Open Studio Cookbook*. Packt Publishing Ltd, 2013.
- [9] Poess, Meikel, et al. "TPC-DI: the first industry benchmark for data integration." *Proceedings of the VLDB Endowment* 7.13 (2014): 1367-1378.
- [10] http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-di_v1.1.0.pdf
- [11] Gartner (20/08/2019). *2019 Gartner Magic Quadrant for Data Integration Tools*. Retrieved the 19th of September of 2019 from: <https://b2bsalescafe.files.wordpress.com/2019/09/gartner-magic-quadrant-for-data-integration-tools-august-2019.pdf>