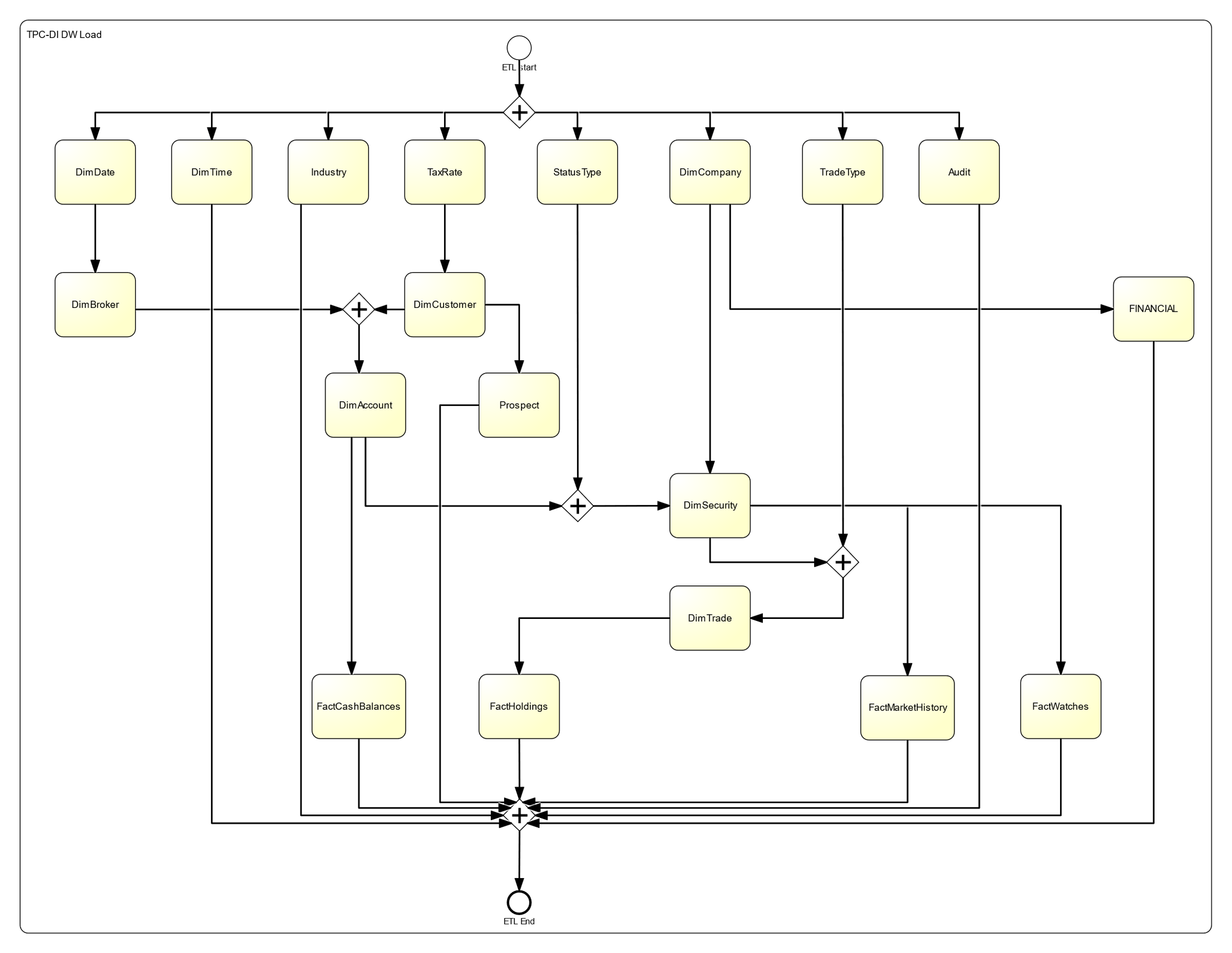
Load scheduling

According to the specification document of TPC-DI, they are dependencies between the load of the tables that define the final scheduling of the whole work. This scheduling is illustrated in the following BPMN diagram.

The parallel gateways imply that the processes that are after can run in parallel. On the other hand, when a process has an incoming edge from a parallel gateway, it means that every process before the gateway should have finished, in order to start this process. For instance, DimAccount requires DimBroker and DimCustomer as well as their predecessors (DimDate andTaxRate).

Table creation:

Every table that has EffectiveDate and EndDate fields is created with a UNIQUE INDEX on the Natural key and the EndDate. In this way, we are ensuring that we do not violate the Primary key constraints of the OLTP database.

File transformation:

CustomerMgmt.xml was transformed to csv using XSLT. The used XSLT template can be found in [Apendix B](#_Apendix_B).

FINWIRE files were merged into one FINWIRE\_MERGED.

Historical load description:

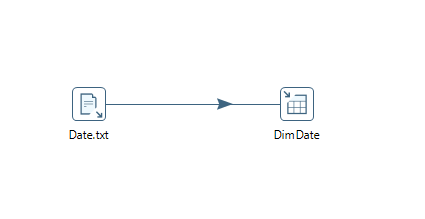
Static tables

Static tables are the ones that are initially loaded with data and they are never updated. The following tables belong to this category: DimDate, DimTime, Financial, Industry, statusType, taxRate, tradeType and DimBroker. The last one is structured as a history tracking dimension table. However, nothing in the

input file will provide any history of changes over time; it is simply a snapshot of the current

state of the HR data.

An example of this category is depicted in the below transformation (DimDate). It is the simplest transformation as the only needed action is to read the data from the source file Date.txt and load them to the target table.



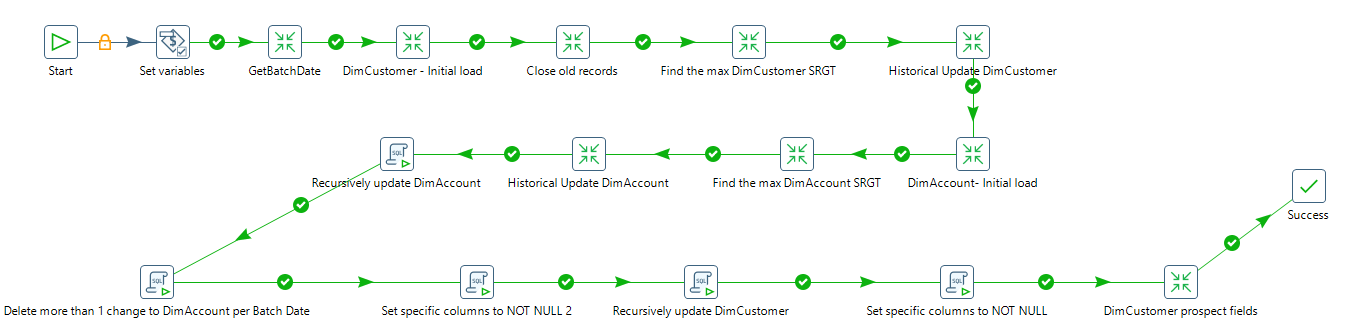
Slowly changing Dimensions

This category of tables are Dimension tables that change over time. Most of them are structured as history tracking dimension tables, which means that we can track the evolution of the business entities (for example customers) over the time, by using the IsCurrent, EffectiveDate and EndDate fields.

Some tables like DimCompany are populated with data only once and some other, like DimCustomer, have Incremental updates. However, in the historical file, it is possible to find multiple records for the same natural key (business key). This means that we need to keep track of the changes, as it is described in chapter 4.4.1 of the specification document.

Below we are presenting some tables that belong to this category.

**DimCustomer**

Both tables are filled with data from CustomerMgmt file. As it is mentioned, this file was transformed to csv and as a result the package uses this file as input. The below job is used to synchronize the load of DimCustomer and DimAccount:

For DimCustomer, we are using only the rows that have ActionType in (‘NEW’, ‘UPDCUST’, ‘INACT’). Firstly, we are inserting only the New customers to the target table. Then, for every customer that has at least one Update or Inactive row, we are closing all the old records. We are defining here that by saying “closing records” we mean that we set to the records that have EndDate = ‘9999-12-31’ and IsCurrent = ‘True’ the EffectiveDate of the new records and ‘False’, respectively. After that, we are calculating in memory the order of the new records. This means that if a customer has more than one updates, we are sorting the records by ActionDate. In this way, we are achieving to keep track of the changes during the time. We also set the EndDate of the update records of each customer that have a forthcoming update to the EffectiveDate of the next record and the IsCurrent to ‘False’, which means that this record will not be the active one for this customer. All this process is done by using the Analytical query component of Pentaho DI that allows us to Partition the dataset by CustomerID (C\_ID) and use the LEAD and LAG rows.

Below, there is an example of a Customer with 1 record with ActionType ‘NEW’, 2 records with ‘UPDCUST’ and 1 record with ‘INACT’:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sk\_customerid | customerid | taxid | status | lastname | firstname | … | marketingnameplate | iscurrent | batchid | effectivedate | enddate |
| 1934 | 21 | 494-50-2485 | ACTIVE | Neely | Herman | … | NULL | FALSE | 1 | 2007-07-15 | 2007-09-04 |
| 4760 | 21 | 494-50-2485 | ACTIVE | Neely | Herman | … | NULL | FALSE | 1 | 2007-09-04 | 2008-05-08 |
| 4761 | 21 | 494-50-2485 | ACTIVE | Neely | Herman | … | NULL | FALSE | 1 | 2008-05-08 | 2008-09-27 |
| 4762 | 21 | 494-50-2485 | INACTIVE | Neely | Herman | … | NULL | TRUE | 1 | 2008-09-27 | 9999-12-31 |

Another issue is that the update records of DimCustomer contain only the values of the updated field. The inserted rows should maintain the value from the previous rows for the field that are not updated and assign the new values for the updated fields. In our code we achieve this by initially inserting every update record with nulls to the non-updated fields and with the new value to the updated ones (we have created the table with nullable fields and in the end of the initial load we alter the fields to not nullable). After the table is fully loaded, we are executing the [sql query 1](#_Query_1) of the Apendix A. For every customer we are recursively update the records from the oldest to the earliest one, by setting to the null fields the value from the previous record.

Finally, we are updating DimCustomer to assign the right values to the prospect fields. This is done by getting all the records of DimCustomer and joining the rows with Prospect.csv source file on LastName, FistName, Address 1, Address 2 and Postcode.

**DimAccount**

The initial load of DimAccount is happening in the same Job with DimCustomer but after the Initial Load of DimCustomer because the first one has the Surrogate key of DimCustomer in its fields. As a result, a change to DimCustomer triggers 1 or more change(s) to DimAccount.

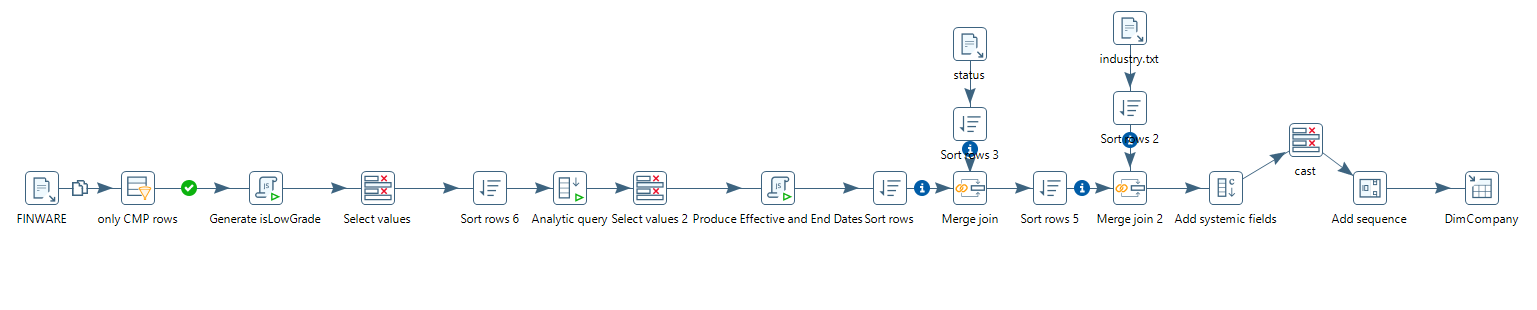
We are following the same strategy as DimCustomer. Firstly, we are inserting only the new accounts, then we close the old records and we are proceeding to the update phase. The difference here is that a row of DimAccount may be updated because of an update to the account fields (ActionType in ‘UPDACC’, ‘CLOSEACCT’) or to the customer that holds this account (ActionType in ‘UPDCUST’, ‘INACT’).

Regarding the execution path of the updates to the customer, we should first join with the DimCustomer table to get the old surrogate key that will be used later to join with the records of DimAccount that should be closed. After that, we union the two paths and we do similar process with the update of DimCustomer (Analytical query etc).

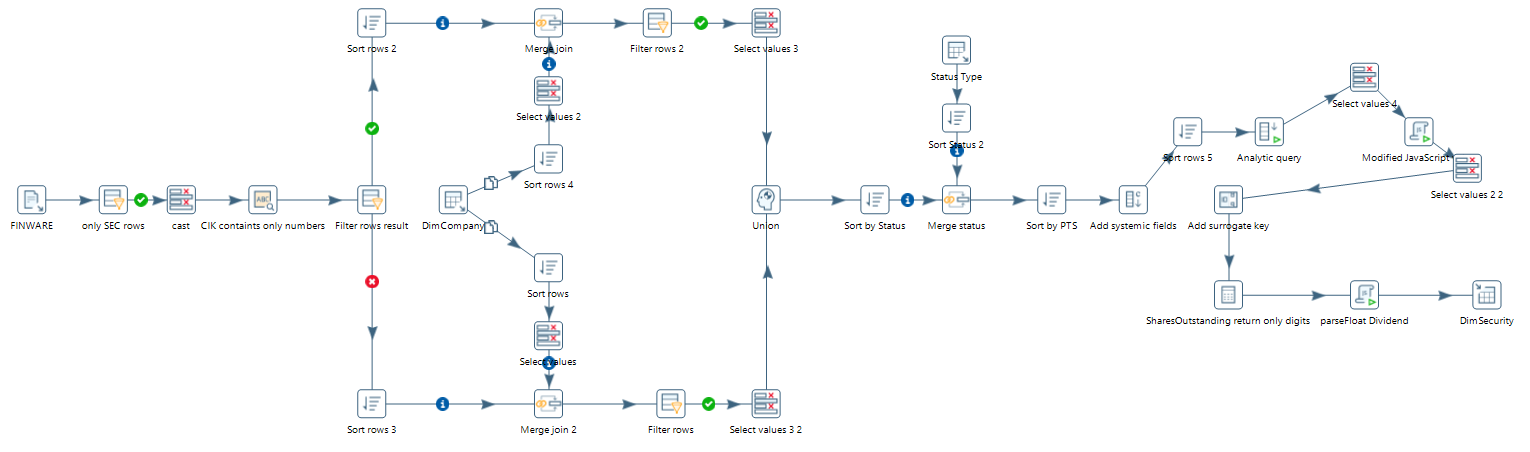
Finally, the view of the account of customer 21 is:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sk\_accountid | accountid | sk\_brokerid | sk\_customerid | status | … | iscurrent | batchid | effectivedate | enddate |
| 98 | 21 | 2261 | 1934 | ACTIVE | … | FALSE | 1 | 2007-07-15 | 2007-08-12 |
| 9248 | 21 | 2261 | 1934 | INACTIVE | … | FALSE | 1 | 2007-08-12 | 2007-09-04 |
| 9249 | 21 | 2261 | 4760 | INACTIVE | … | FALSE | 1 | 2007-09-04 | 2008-05-08 |
| 9250 | 21 | 2261 | 4761 | INACTIVE | … | FALSE | 1 | 2008-05-08 | 2008-09-27 |
| 9251 | 21 | 2261 | 4762 | INACTIVE | … | TRUE | 1 | 2008-09-27 | 9999-12-31 |
| 100 | 559 | 3785 | 4760 | ACTIVE | … | FALSE | 1 | 2008-02-19 | 2008-05-08 |
| 9252 | 559 | 3785 | 4761 | ACTIVE | … | FALSE | 1 | 2008-05-08 | 2008-09-27 |
| 9253 | 559 | 3785 | 4762 | INACTIVE | … | FALSE | 1 | 2008-09-27 | 2009-01-18 |
| 9255 | 559 | 1724 | 4762 | INACTIVE | … | TRUE | 1 | 2009-01-18 | 9999-12-31 |
| 99 | 1003 | 3208 | 4761 | ACTIVE | … | FALSE | 1 | 2008-08-18 | 2008-09-27 |
| 9246 | 1003 | 3208 | 4762 | INACTIVE | … | FALSE | 1 | 2008-09-27 | 2013-08-09 |
| 9254 | 1003 | 350 | 4762 | INACTIVE | … | FALSE | 1 | 2013-08-09 | 2015-06-15 |
| 9247 | 1003 | 350 | 4762 | INACTIVE | … | TRUE | 1 | 2015-06-15 | 9999-12-31 |

The last transform we need to do is to delete the rows that triggered more than 1 update during the same day. This is done by executing [query 2](#_Query_2).

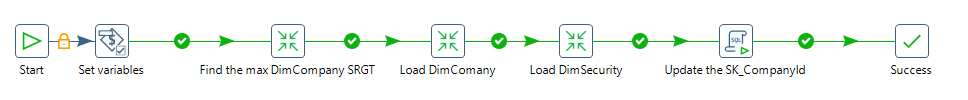
**DimCompany**

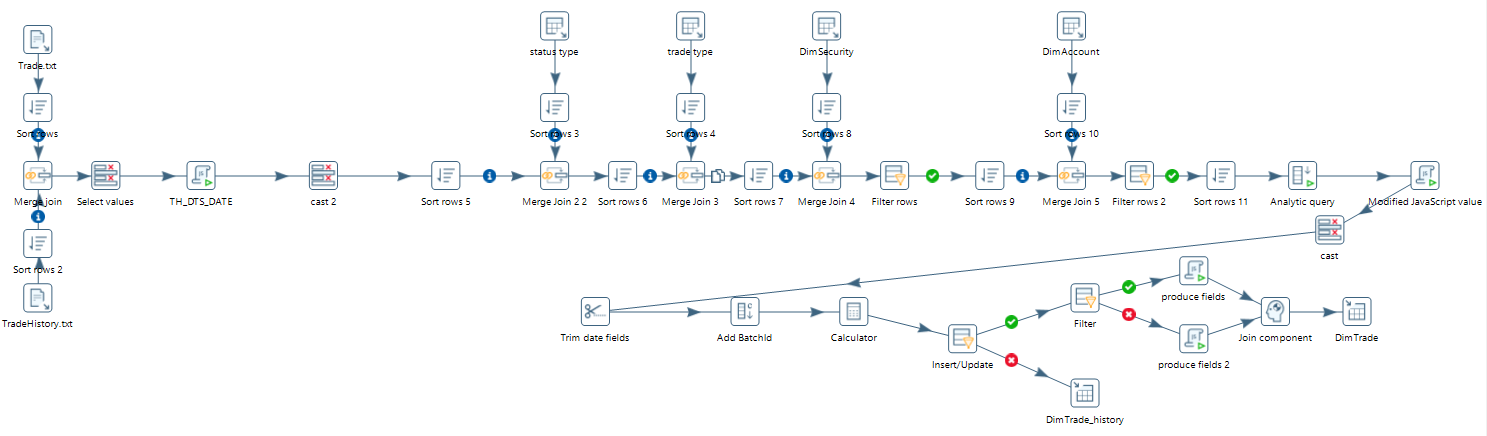
DimCompany table is filled with data from FINWIRE\_MERGED.txt file. We filter the rows and we keep only those that have RecType = ‘CMP’ and we are also importing status.txt and industry.txt to form some other fields. Analytical query is also used in this to manipulate the EffectiveDate and EndDate fields.

**DimSecurity**

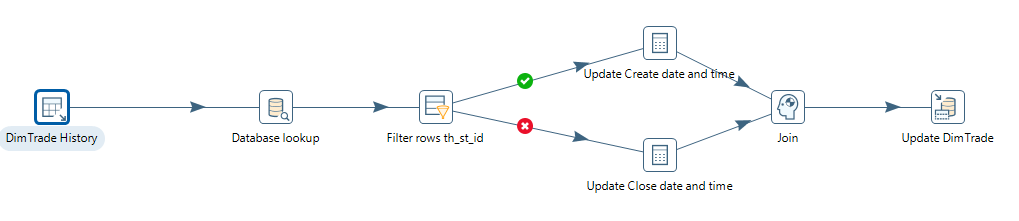
DimSecurity is filled right after DimCompany, by using the rows with RecType = ‘SEC’ from FINWIRE\_MERGED.txt file. The rows are joined with DimCompany and Status type in order to get some fields. Analytical query is also used in this to manipulate the EffectiveDate and EndDate fields. Finally, it is needed to run the [query 3](#_Query_3) in order to update the Surrogate keys of the companies that were updated in the DimCompany table.

Both tables are filled by loadDimcompany\_DimSecurity.kjb job which is depicted below:



**DimTrade**

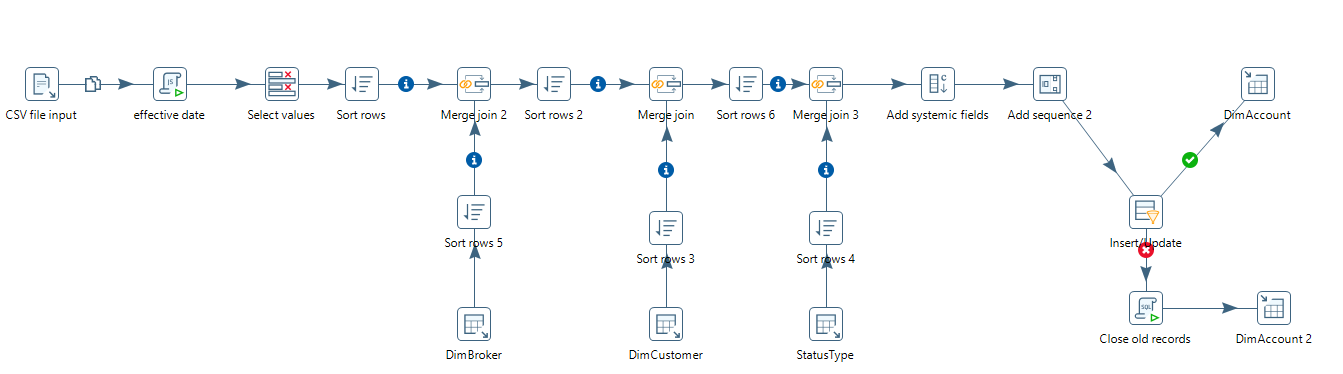
Dimtrade is a hybrid table, which means that it can be used either as a Fact or as a Dimensional table. It is filled by both Trade.txt and TradeHistory.txt files. Data from Status type, Trade type, DimSecurity and DimAccount are combined to form its fields. In addition, we are using Analytical query component to find the earliest record for each tradeid. This record will be inserted to DimTrade and all the other records will be inserted to DimTrade\_history table.

In the next step, we are fetching only the newest update records from DimTrade\_history, by running the [query 4](#_Query_4), and we are updating the records of DimTrade. In this way, we only keep the final view of DimTrade, without keeping track of its history.

Incremental updates

Incremental updates are those updates that happen periodically to some of the tables of the data warehouse (daily/monthly etc.). In TPC-DI, we have those updates to the following tables: DimCustomer, DimAccount, DimTrade and every Fact table.

In our case, the phase of incremental updates takes place during Batch2 and Batch3. These types of updates are easy to manipulate as the source files of Batch2 and Batch3 are deltas. This means that we do not have to identify the changes that happened to the OLTP database during the current Batch because there is the field CDC\_FLAG which indicates whether the record is an insert or an update. For every new record (insert) we import it to the data warehouse while for every old record (update) we first close the old record that exists in the data warehouse and then we insert the new one.

The transformations are slightly changed compared to those from the Initial load. Below, there is an example transformation (DimAccount).

We can see that we only need to have a filter in the end of the transformation (Insert/Update component) that defines whether the record needs just to be inserted to the database or an update (close of the existing record) has to be done first.

Apendix A

Query 1

with RECURSIVE temp as(

    select ROW\_NUMBER () OVER (

        PARTITION BY customerid

        ORDER BY effectivedate

        ) as rn,

        \*

    from dimcustomer

),new\_dimcustomer as(

    select \* from temp

    where rn = 1

    union

    select

        b.rn

        ,b.SK\_CustomerID

        ,b.customerid

        ,case when b.TaxID is null then a.TaxID else b.TaxID end as TaxID

        ,case when b.Status is null then a.Status else b.Status end as Status

        ,case when b.LastName is null then a.LastName else b.LastName end as LastName

        ,case when b.FirstName is null then a.FirstName else b.FirstName end as FirstName

        ,case when b.MiddleInitial is null then a.MiddleInitial else b.MiddleInitial en d as MiddleInitial

        ,case when b.Gender is null then a.Gender else b.Gender end as Gender

        ,case when b.Tier is null then a.Tier else b.Tier end as Tier

        ,case when b.DOB is null then a.DOB else b.DOB end as DOB

        ,case when b.AddressLine1 is null then a.AddressLine1 else b.AddressLine1 end as AddressLine1

        ,case when b.AddressLine2 is null then a.AddressLine2 else b.AddressLine2 end as AddressLine2

        ,case when b.PostalCode is null then a.PostalCode else b.PostalCode end as PostalCode

        ,case when b.City is null then a.City else b.City end as City

        ,case when b.StateProv is null then a.StateProv else b.StateProv end as StateProv

        ,case when b.Country is null then a.Country else b.Country end as Country

        ,case when b.Phone1 is null then a.Phone1 else b.Phone1 end as Phone1

        ,case when b.Phone2 is null then a.Phone2 else b.Phone2 end as Phone2

        ,case when b.Phone3 is null then a.Phone3 else b.Phone3 end as Phone3

        ,case when b.Email1 is null then a.Email1 else b.Email1 end as Email1

        ,case when b.Email2 is null then a.Email2 else b.Email2 end as Email2

        ,case when b.NationalTaxRateDesc is null then a.NationalTaxRateDesc else b.NationalTaxRateDesc end as NationalTaxRateDesc

        ,case when b.NationalTaxRate is null then a.NationalTaxRate else b.NationalTaxRate end as NationalTaxRate

        ,case when b.LocalTaxRateDesc is null then a.LocalTaxRateDesc else b.LocalTaxRateDesc end as LocalTaxRateDesc

        ,case when b.LocalTaxRate is null then a.LocalTaxRate else b.LocalTaxRate end as LocalTaxRate

        ,case when b.AgencyID is null then a.AgencyID else b.AgencyID end as AgencyID

        ,case when b.CreditRating is null then a.CreditRating else b.CreditRating end as CreditRating

        ,case when b.NetWorth is null then a.NetWorth else b.NetWorth end as NetWorth

        ,case when b.MarketingNameplate is null then a.MarketingNameplate else b.MarketingNameplate end as MarketingNameplate

        ,b.IsCurrent

        ,b.BatchID

        ,b.effectivedate

        ,b.enddate

    from temp b inner join new\_dimcustomer a on a.rn = b.rn-1 and a.customerid = b.customerid

)

update dimcustomer as old\_cust

set TaxID = new\_cust.TaxID  ,

Status = new\_cust.Status  ,

LastName = new\_cust.LastName  ,

FirstName = new\_cust.FirstName  ,

MiddleInitial = new\_cust.MiddleInitial  ,

Gender = new\_cust.Gender  ,

Tier = new\_cust.Tier  ,

DOB = new\_cust.DOB  ,

AddressLine1 = new\_cust.AddressLine1  ,

AddressLine2 = new\_cust.AddressLine2  ,

PostalCode = new\_cust.PostalCode  ,

City = new\_cust.City  ,

StateProv = new\_cust.StateProv  ,

Country = new\_cust.Country  ,

Phone1 = new\_cust.Phone1  ,

Phone2 = new\_cust.Phone2  ,

Phone3 = new\_cust.Phone3  ,

Email1 = new\_cust.Email1  ,

Email2 = new\_cust.Email2  ,

NationalTaxRateDesc = new\_cust.NationalTaxRateDesc  ,

NationalTaxRate = new\_cust.NationalTaxRate  ,

LocalTaxRateDesc     = new\_cust.LocalTaxRateDesc  ,

LocalTaxRate = new\_cust.LocalTaxRate  ,

AgencyID = new\_cust.AgencyID  ,

CreditRating  = new\_cust.CreditRating  ,

NetWorth = new\_cust.NetWorth  ,

MarketingNameplate = new\_cust.MarketingNameplate

from  new\_dimcustomer as new\_cust

where new\_cust.sk\_customerid = old\_cust.sk\_customerid

--order by new\_cust.customerid, new\_cust.effectivedate;

Query 2

--Delete the records for the accounts that changed more than once in a batch date

--Example: Accountid = 138,159 and 270

with tmp as(

select a.accountid, max(a.sk\_accountid) as sk\_accountid

,min(a.sk\_accountid) as sk\_accountid\_new, max(enddate) as enddate\_new

from dimaccount a

inner join(

    select accountid, effectivedate from dimaccount where

    accountid in

    (

        select accountid from dimaccount where effectivedate = enddate

    )

    group by accountid, effectivedate

    having count(1)>1

) b on a.accountid = b.accountid and a.effectivedate = b.effectivedate

group by a.accountid)

,deleted as(

    delete from dimaccount a

    using tmp b where a.sk\_accountid = b.sk\_accountid

    returning a.\*

)

update dimaccount a

set iscurrent = b.iscurrent

,enddate = b.enddate

from deleted b, tmp c

where c.sk\_accountid = b.sk\_accountid and a.sk\_accountid = c.sk\_accountid\_new;

--recreate the unique index

CREATE UNIQUE INDEX UI\_DimAccount ON DimAccount (AccountID, EndDate);

Query 3

with tmp as(

select max(sk\_securityid) as max\_srgt from dimsecurity

),

updated\_sec as(

update dimsecurity as b

set enddate = a.enddate,

iscurrent = 'False'

from

        (select \*

         ,case when LEAD(sk\_companyid,1) OVER ( PARTITION BY companyid ORDER BY effectivedate) is null then -1 else LEAD(sk\_companyid,1) OVER ( PARTITION BY companyid ORDER BY effectivedate) end as next\_sk\_companyid

         ,LEAD(effectivedate,1) OVER ( PARTITION BY companyid ORDER BY effectivedate) next\_effd

         ,LEAD(enddate,1) OVER ( PARTITION BY companyid ORDER BY effectivedate) next\_sk\_endd

         from dimcompany) as a

where a.sk\_companyid = b.sk\_companyid

and a.enddate <> '9999-12-31'

and b.effectivedate <= a.enddate and a.enddate < b.enddate

and b.enddate = '9999-12-31'

RETURNING

b.Symbol ,

b.Issue ,

b.Status ,

b.Name L,

b.ExchangeID ,

a.next\_sk\_companyid,

b.SharesOutstanding ,

b.FirstTrade ,

b.FirstTradeOnExchange ,

b.Dividend ,

cast('True' as boolean) ,

b.BatchID ,

a.EndDate as EFFECTIVEDATE,

cast('9999-12-31' as date)

)

insert into dimsecurity

select max\_srgt + ROW\_NUMBER() OVER (ORDER BY EFFECTIVEDATE)

,a.\* from updated\_sec a, tmp

Query 4

--Fetch the last record in order to update DimTrade table

select \*

from

(

    select rank() over (partition by tradeid order by date\_int desc, time\_int desc) as rank

    ,\*

    from dimtrade\_history

) a

where rank = 1

order by tradeid

Apendix B

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="text" encoding="UTF-8"/>

<xsl:template match="/Actions">

        <xsl:for-each select="Action/Customer">

            <xsl:value-of select="concat(../@ActionType, '|',../@ActionTS, '|'

                                        ,@C\_ID, '|',@C\_TAX\_ID, '|',@C\_GNDR, '|',@C\_TIER, '|',@C\_DOB, '|'

                                        , Name/C\_L\_NAME, '|', Name/C\_F\_NAME, '|', Name/C\_M\_NAME, '|'

                                        , Address/C\_ADLINE1, '|', Address/C\_ADLINE2, '|', Address/C\_ZIPCODE, '|' , Address/C\_CITY, '|', Address/C\_STATE\_PROV, '|', Address/C\_CTRY, '|'

                                        , ContactInfo/C\_PRIM\_EMAIL, '|', ContactInfo/C\_ALT\_EMAIL, '|'

                                        , ContactInfo/C\_PHONE\_1/C\_CTRY\_CODE, '|', ContactInfo/C\_PHONE\_1/C\_AREA\_CODE, '|', ContactInfo/C\_PHONE\_1/C\_LOCAL, '|', ContactInfo/C\_PHONE\_1/C\_EXT, '|'

                                        , ContactInfo/C\_PHONE\_2/C\_CTRY\_CODE, '|', ContactInfo/C\_PHONE\_2/C\_AREA\_CODE, '|', ContactInfo/C\_PHONE\_2/C\_LOCAL, '|', ContactInfo/C\_PHONE\_2/C\_EXT, '|'

                                        , ContactInfo/C\_PHONE\_3/C\_CTRY\_CODE, '|', ContactInfo/C\_PHONE\_3/C\_AREA\_CODE, '|', ContactInfo/C\_PHONE\_3/C\_LOCAL, '|', ContactInfo/C\_PHONE\_3/C\_EXT, '|'

                                        , TaxInfo/C\_LCL\_TX\_ID, '|', TaxInfo/C\_NAT\_TX\_ID, '|'

                                        , Account/@CA\_ID, '|', Account/@CA\_TAX\_ST, '|', Account/CA\_B\_ID, '|', Account/CA\_NAME

                                        )"/>

            <xsl:if test="position()!=last()">

                <xsl:text>

</xsl:text>

            </xsl:if>

        </xsl:for-each>

</xsl:template>

</xsl:stylesheet>