Automating Transformations in Data Vault Data Warehouse Loads

Mikko PUONTI [a], Timo RAITALAAKSO [a,b], Timo AHO [c] and Tommi MIKKONEN [b]

[a] *Solita PLC, Åkerlundinkatu 11, FI-33100 Tampere, Finland, e-mail: puonti@iki.fi, timo.raitalaakso@iki.fi*

[b] *Department of Pervasive Computing, Tampere University of Technology, PO BOX 553, FI-33101 Tampere, Finland, e-mail: tommi.mikkonen@tut.fi*

[c] *Yle, The Finnish Broadcasting Company, Box 97, FI-00024 Yleisradio, Finland, e-mail: timo.aho@yle.fi*

Abstract. Data warehousing is a process of integrating multiple data sources into one for, e.g., reporting purposes. An emerging modeling technique for this is the data vault method. The use of data vault creates many structurally similar data processing modifications in the transform phase of ETL work. Is it possible to automate the creation of transformations? Based on our study, the answer is mostly affirmative. Data vault modeling creates certain constraints to data warehouse entities. These model constraints and data vault table populating principles can be used to generate transformation code. Based on the original relational database model and data flow metadata we can gather populating principles. These can then be used to create general templates for each entity. Nevertheless, we need to note that the use of data flow metadata can be only partially automated and includes the only manual work phases in the process. In the end we can generate the actual transformation code automatically. In this paper, we carefully describe the creation of automation procedure and analyze the practical problems based on our experiences on PL/SQL proof of concept implementation. To the best of our knowledge, similar has not yet been described in the scientific literature.

Keywords. data vault, database modeling, ELT, ETL, code generation

1. Introduction

Data warehousing is a technique for integrating data from several source systems to enable reporting and finding dependencies in the data. Data merging from different data sources to a data warehouse is typically done in three phases: The data is *E*xtracted from source systems, *T*ransformed to the target structure, and *L*oaded to a data warehouse. The order of the last two stages may vary and we call the variants accordingly Extract-Load-Transform (ELT) or Extract-Transform-Load (ETL). These phases are typically done in all the data warehouse modeling types.

In the case of ELT, the two first functions form a way to bring source data as-is to a staging area in a data warehouse. In this case, extract and load functions use similar data structure, and the transformation is done only at the last stage in the process. The final transformation, the final phase, is actually about loading data from the initial staging area to data vault tables. Its implementation can be (and often is) an ETL system of its own, and an ETL tool is used manually to define the transformation. This is the case in data warehouses using data vault modeling technique [5]. Data vault is a modeling technique designed to meet the needs of enterprise data warehousing. In the original work, Linstedt gives the following definition for it: "The Data Vault is a detail oriented, historical tracking and uniquely linked set of normalized tables that support one or more functional areas of business."[5].

Modeling is an integral part of the work in data warehousing and there are different ways of doing this. In particular, data vault modeling has become a popular way in designing a data warehouse because of its flexibility. When using data vault as modeling technique there are several similar data transformations. When adding a new data source to a data warehouse, the data model needs to be updated. Information about attributes in the source system interface is used when modeling the data warehouse. In addition, the transformation from a data source to a data warehouse needs to be tuned manually.

In this paper, our research concentrates on this manual work—how it is possible to minimize or even omit the manual part? The goal is to reduce the amount manual work needed by using model information more intensively. Based on the data vault modeling technique, there are rules based on the structure and rules on how the data vault entities will be populated. Based on these principal rules, it is possible to automatically generate ETL code. This code generation needs information about model and data flow. To enable this, a data model of system's metadata is created for this code generation. However, the process of creating transformations based on the metadata of models is different than in a manual ETL process.

As a practical contribution, we have implemented a PL/SQL proof of concept code generator for transformations from a data source to a data warehouse using data vault modeling. In this paper, we describe the process of automatically generating the transformations based on the metadata of data models. In addition, we present our implementation and analyze the learnings and pitfalls based on it.

The rest of this paper is structured as follows. In Section 2, we introduce the background of the paper. In Section 3, we present the principles of populating the data vault entities. In Section 4, we introduce data model to automate transformation and what to automate with it. In Section 5 we provide a small example regarding data map usage. In Section 6, we analyze the overall work and our experiences more generally. In Section 7 possible new research topics. Finally, in Section 8 we draw conclusions.

2. Background and Related Work

Jovanovic and Bojicic present a platform independent metamodel to store the data warehouse model information [3]. They give several examples of doing the conversion from a logical data model to a data vault model.

Phipps and Davis automate the data warehouse conceptual schema design [9]. They divide data warehouse creation to five steps: pre-development activities, architecture selection, schema creation, warehouse population, and data warehouse maintenance. They focus on schema creation phase and automation of that, whereas we are focusing on the next phase, warehouse population.

El Akkaoui et al, propose a model-driven development framework for ETL processes [1]. That framework aims at automatic generation of ETL code for several vendor specific platforms. The vendor-independent model is defined using a platform independent design model of ETL based on business process model notation (BPMN4ETL), ETL processes are generated based on that model.

Pankov et al share the idea of using metadata as a starting point for automating data warehouse implementation [8]. They describe metadata model which is modelled with data vault principles. Based on that metadata they show possibilities regarding how to generate ETL processes. A limitation of this approach is that the ETL tool has to expose its functionality as an application programming interface.

Several case tools for database modeling are available, such as Oracle SQL Developer Data Modeler[1] and ER Studio[2]. They use their own internal models to store the model information. The core usage of these tools is to forward and reverse engineer database structures. It is also possible to draw process diagrams with these tools. Data flow may be derived from a process model, if the structure is tied to the processes. Attributes used in process information structures may be associated with table columns with these tools. This capability is hidden behind several steps in graphical user interfaces. No code generation capabilities are available based on this information. In this case, traditional ETL tools are used to develop the transformations.

Data vault modeling consists of three major components. A hub has only the business key columns for a specific entity. A link draws a many-to-many relationship between several hubs. A satellite holds descriptive information about the context. In addition, reference tables are related to data vault modeling. Nevertheless, according to Lindstedt [4, 102], reference data should be separated from other data vault tables.

Data vault modeling introduces flexibility by adding new entities to the model. Existing entities and structures will remain and new modeling will be added to the model. As for the tool support for doing this, while data warehouse software with automation features exist, most of the software systems that are suited for data vault modeling are licensed products. However, there are two

[1] http://www.oracle.com/technetwork/developer-tools/datamodeler/overview/
[2] http://www.embarcadero.com/products/er-studio

open source distributions: Quipu[3] and Optimal Data Engine[4]Both of these use Microsoft Visual Studio as the front-end for the development.

3. Principles for Automation

Data vault modeling yields certain rules that can be used in code generation. Firstly, data vault entities hub and link have a surrogate key which is a one column primary key. Secondly, a satellite primary key is constructed with the surrogate key and load time. In addition, satellites and links have foreign key references. In this section these rules are explained for each data vault entity: hub, link, satellite, and reference.

Common attributes to all data vault tables Every data vault table has metadata. This metadata is stored in attributes as listed in Table 1. Our naming convention is to start every data vault metadata attribute with prefix `DV_`.

Table 1. Data vault common attributes

`DV_ID`	Surrogate key
`DV_LOAD_TIME`	Timestamp when the row is inserted
`DV_SOURCE`	Metadata of the source system
`DV_LOAD_NAME`	Name of the transformation which inserted the row
`DV_RUN_ID`	Every load batch has a unique id

3.1. Principles for a Hub

Business keys are vital to locate the business data [7]. Natural business key is a unique identifier for business concept [2, Chapter 4]. A hub present the business key for the business concept. If the business uses the same business key in all steps of the business process, then all data would be linked via the hub. Unfortunately, this is not usually the case, since different systems use quite often sequence numbers for their business keys. A hub is a mapping from a business key to a surrogate key. Surrogate key is one column primary key.

A business key may consist of several columns. A hub stores all unique business key values and creates a unique key constraint for business key attributes.

3.2. Principles for a Link

A link is an associative entity between hubs. When two or more business keys interact, a link is created to represent this interaction. This link is a many-to-many relationship table between hub tables, storing hub surrogate key values. Link table attributes are the common attributes listed in Table 1, together with all associative hub surrogate key values, which are related to that particular link. A link can also be used to store hierarchy information of a hub. In this case,

[3]http://www.datawarehousemanagement.org
[4]https://github.com/OptimalBI/optimal-data-engine-mssql

the link has relations to only one hub table. Every link hub surrogate key has a foreign key reference to the related hub surrogate key attribute.

Data vault modeling allows creating more than one link between two or more hub tables. The link structure introduces flexibility in the modeling process. When there are changes in the real world, the corresponding modifications can be done in the data vault model simply by creating new links.

A link table referring to another link table should not be modeled. Such a link to link structure can be resolved by creating a link to have all hub surrogate keys from both links. Database foreign key information is used in code generation, which is the reason why all link to link relations is forbidden.

3.3. Principles for a Satellite Entity

A satellite is an entity which holds descriptive information of a hub or a link. It includes other attributes than those that comform a business key for that entity. The referring attribute to a hub or a link is a surrogate key.

We list all common satellite attributes in Table 2. Satellite tables have two additional columns compared to a hub or a link table. Datahash is special attribute, which is used for capturing changes in a satellite [6, Chapter 11.2.5]. The Datahash value is calculated from the concatenation of business key and all satellite data attributes. There might be a need in future to have multiple active values in a satellite, we added in every satellite record order for that purpose.

Table 2. Satellite table common attributes

`DV_ID`	Surrogate key
`DV_LOAD_TIME`	Timestamp when the row is inserted
`DV_SOURCE`	Metadata of the source system
`DV_LOAD_NAME`	Name of the transformation which inserted the row
`DV_RUN_ID`	Every load batch has a unique id
`DV_DATAHASH`	Datahash is computed of business keys and data attributes
`DV_RECORD_ORDER`	This is for multiple active satellite rows

Satellite information is subject to change over time [5]. A satellite will store the whole history of the descriptive data. To help querying the satellite we have created current view, which show the latest known data row for each surrogate key. When populating the satellite table the current view can be used. New row is added from staging table to satellite when the data in staging table differs from data in current view, the comparison is done by comparing the datahash attribute values in staging table and satellite current view.

There are modeling principles regarding how to split the satellite attributes to several satellite tables [2, Chapter 21], but how the split is done is not relevant for generating transformations to satellite tables.

Status satellite has a special meaning A status satellite tells the time intervals when the referenced row has been active. If the staging table is a full dump from a source system, the deleted rows can be recognized. If it is an incremental dump for a link, information about a driving hub should be available. The driving hub information is used to identify changed links.

3.4. Principles for a Reference

Reference data holds a list of code values with their descriptions. Typical items are units of measure, postal codes, currencies, and so on. The code can be in several places in a data vault, and it is not reasonable to copy the same descriptions to several places. A reference – a different kind of entity from other major data vault components – refers to other tables in the logical sense, but it will not have any foreign key references in the physical model. To mitigate new transformation for reference the reference implementation can be changed to view which is based on a hub and a satellite table. Then the reference transformation is splitted to hub and satellite transformations, there is no need to have different transformation rules for the reference.

3.5. Transformation

A transformation phase in ELT is in itself a kind of a transformation of its own, this time of type ETL. One transformation between staging and data vault table is extracting rows from the source table, transforming them, and then inserting the chosen columns of the rows to the target table. All of these steps may be done in a single insert into select statement.

A transformation is a data flow element at entity level. When a data flow is coming from a staging table to a data vault structure, each source entity - target entity constructs a transformation.

Attributes in a Transformation A transformation at the attribute level is constructed using mappings. For a normal attribute, such mapping is simply a relation from a source attribute to a target attribute for raw data vault transformations. There should not be any attribute level transformations in the mappings. No data cleansing is to be done in this phase of data warehouse population. Consequently, both source and target attributes in a mapping should have matching data types. The business keys in hubs are used for consolidation. This is why it is suggested to trim business key columns [6, Chapter 11]. This could lead to a need for technical hubs that store different presentations of the business keys. The original presentation might be stored in a satellite as a normal attribute. This way, dirty data in source systems could lead to a need for misusing multi-valued satellite implementation. For the hub transformations, the whole business key should be mapped. In other words, if the hub has several columns in its business key, there should be a mapping for all of its key columns in each transformation that will be populating the hub. The attributes are mapped from a source staging table columns to target data vault entity columns. It is equally important to map the access paths for columns used in foreign keys. Surrogate key columns are used in foreign keys in the data vault. To define a transformation for a satellite surrogate key, a data transformation from the same staging table defines which business key columns are used to look up or construct the surrogate key. The same rule applies to link referential hub surrogate key columns.

Defining Transformations for a Hierarchy Parent-child relationships in the model introduce a hierarchy link in the data vault. Examples of such are product and account hierarchies. In these cases, several transformations from the same staging table to the same hub exist, and at least one to a link that defines the hierarchy. Therefore, transformation naming needs more information than just source entity and target entity names.

4. Automate Transformation

An agile approach to develop transformations is by slicing the model. One way to divide work to get smaller deliverable is to develop by source system dataset.

Current ETL software have lack of parallel development. The development design happens in the same server and developers have to communicate that they do not change the same ETL object at the same time. Development with automating transformations can be separated. Each developer has their own design environment. A developer creates the implementation of transformations and the implementation is shared among other developers via version control system.

4.1. Information Model for Automated Transformation – Data Map

For automating transformations, we need information of source structure and target structure. The power of the data model in Figure 1 is that it represents model information and mapping between different models. Automating the transformations is based on the structure (model) and data flow information.

Structure information is in Figure 1 with green color. An entity is in a relational database table or view, but it can be something else if the data warehouse is not in a relational database. An attribute is column information, each entity has several attributes. Reference is foreign key relations at attribute level.

Data flow information between entities is a transformation. In the model, a transformation has a source entity and a target entity. A mapping is transformation information at attribute level. A mapping information is entered for each attribute for a target entity. Figure 1 shows the data flow information at violet color.

Populating the Structure The data model may be populated from a existing relational database metadata. While populating an entity and querying the table information from a database metadata also the reference and attribute level information may be queried. Based on the found information the model may be populated. An entity may become from a table or a view. A staging entity may be implemented as a view. The rules of the data vault structure may be used to populate the structure part. The entity type may be decided based on the rules. Also if there is a satellite in the model there has to be a hub or a link that the satellite belongs to. Similarly if there is a link there should be at least two hubs in the model that the link is referring to. So if the model is populated from a database metadata the model population may be started from satellites. Traverse through foreign key definitions and populate the entity, attribute and reference information on the way.

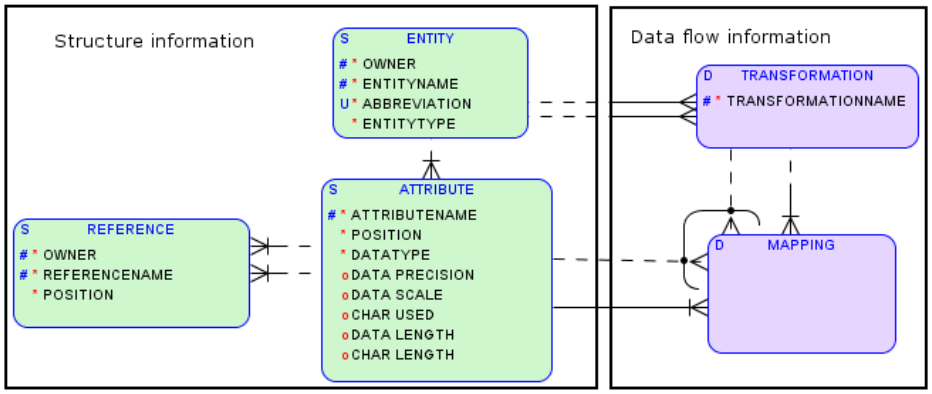

Figure 1. The data map model

Populating the Data Flow The transformation level is chosen first. This means choosing from which staging entity to a data vault entity there will be a transformation. Similar rules like populating the structure part need to be used. If there is a transformation from a staging table to a satellite, there has to be a transformation from the same staging table to a dependent hub or link. The same rule applies to a link transformations. All dependent hubs should have a transformation from the same staging table that is transforming the link. Mapping at an attribute level may be suggested based on the similar naming of the attributes. Another way to suggest the mapping information is using the attribute order of a source entity. Also similar data types should be checked to avoid implicit data type conversions. After the model is enriched with several mappings the mapping table constructs a dictionary, which is used to suggests the mappings in other transformations.

4.2. Generating Transformations

We have created a data map data model capable of automating transformation generation. Data map enables more than just transformation generation.

Data map model can be used as a source to generate create table clauses. It can be used as a primary definition place to define a data warehouse data model. This would support data first development strategy. We have currently chosen the model first development strategy where the data warehouse model is drawn first with a suitable tool for the job. Those tools support the features of the environments that the data vault model will be installed to.

As the data vault is the place to store a history of data changes the latest knowledge of the data may be published through views that publish a latest known rows, we call these views current view. Current views is possible to generate based on structure information and rule that the latest known row will be displayed. The history is stored mainly in satellites. For a satellite the current view publishes the latest rows for each surrogate key. These satellite current views are used in data map insert views generated in the next steps. Also a relationship may be changed

in source systems. So there is a need to make links disapear in certain points in time. This is done with a status satellite for a link. In link current view a link rows that are not marked as deleted in the latest status satellite are published. A link without a status satellite may be published as is as a link current view. The view is created for future changes in the model. A link status satellite may be created afterwards for the link. The view may be changed to take the status satellite into account. By creating such a link current view the interface to the data vault for user stays similar even when future changes appear. Point in time views are somewhat similar to current views. A point in time table is a query assistant table [4, Chapter 7] . They publish the transaction time aspect history of the tables. A point in time view is showing the hub surrogate key and satellite transformation times related to that hub for every snapshot date chosen to point in time view. This generation needs only the structure part of the data map.

Also basic publishing views may be generated based on the structure. A link publishing view could join the used hubs and publish the used business keys in a view. A current satellite may be published together with the business keys in the related hub. Also supernova views may be generated [10]. Supernova modeling technique generates views on top data vault modeled structure and reveals data to the reporting layer.

A ETL transformation could be done by simply with a single insert into clause. That could be generated from data map. We have chosen to split the single SQL statement into a view and a function. These insert views and functions are generated for each ETL transformation. The split is done for better testability.

The insert view publishes the new rows from the staging table compared with the latest rows in the data vault. The insert view is using earlier generated current views for satellites. For hubs and links the comparison is made straight between the staging entity and target table. The insert view publishes distinct rows from the source.

The insert functions include the insert into select clauses. Insert is into the target table and select part is using the generated insert views. The function is returning the number of inserted rows. Inside the generated function it is possible to do error handling code. Unique key constraint violation can be handled in the functions if there happens to become several runs of the ETL transformations in parallel.

How a data vault 2.0 style surrogate key hash is generated may be interpreted from the data map. Several staging tables may be used to populate the same hub. The used order of the columns in hash generation should be taken from the order of the target hub attribute ordering. The access path to the ordering of the staging table columns goes through the mapping table. Data hash generation is based similarly to the target table column ordering than the surrogate key hash generation.

4.3. Developing with Automated Transformation

Here is a list of steps when developing with automated transformation.

1. Model and create the new data warehouse entities with case tool
2. Populate structure information from database metadata

3. Populate data flow information
4. Generate objects based structure and data flow information
5. Testing in development environment
6. Create delivery scripts from development to other environments like test and production

First step is to make the data warehouse data model changes with a case tool. When the data model is modeled, it will be created with ddl-statements to a relational database. The end result of the modeling will be ddl-statements and model information in the relational database.

Structure is populated from database metadata of all entities which structure information is needed in the implementation. Structure population use structure rules and populate all dependant entities. When populating link or satellite structure information check is the dependent objects structure information populated and populate those if needed. In Listing 1 the structure population is on row 1.

Listing 1 Pseudocode commands for automating transformations

```
1  populate_entity('<insert entity here>');
2  populate_transformation('<source table>','<target table>');
3  create_current_views();
4  create_insert_views_and_functions();
```

Data flow information population needs manual acceptance. Data flow information is populated at entity and attribute level. Entity level data flow information is a transformation and it can be given with command like in Listing 1 at row 2. transformation information need information on source table and target table. Mapping information population can be suggested based on either attribute names or position of the attributes in a source table.

Automating work is generating the deliverable without human work. After the structure and data flow information is in the data map model all transformations can be generated. In generation one command generates all objects, compared with manual work where each object has to done manually. Generation can also be done one by one by specifying what an object should generate.

In parallel development testing has to be part of the development. Test cases are stored in version control and after each development cycle a developer will execute the test cases. We stress the importance of local testing in parallel development.

After testing in the local development is passed then the implementation is locally ready. In parallel development there has to be version control in use and release management practices. Create the delivery script from local environment by following the release management practices.

5. Data Map Usage Example

Next, we provide a small example, where data from several source systems is stored in a data vault data warehouse. There are three different source systems –

customer relations management (CRM), supply chain management (SCM), and product life cycle management (PLM) systems. The data model of this example is given in Figure 2. Tables are classified by using colors; white tables are staging tables, and blue, red, and yellow links represent data vault tables, links, and satellites, respectively.

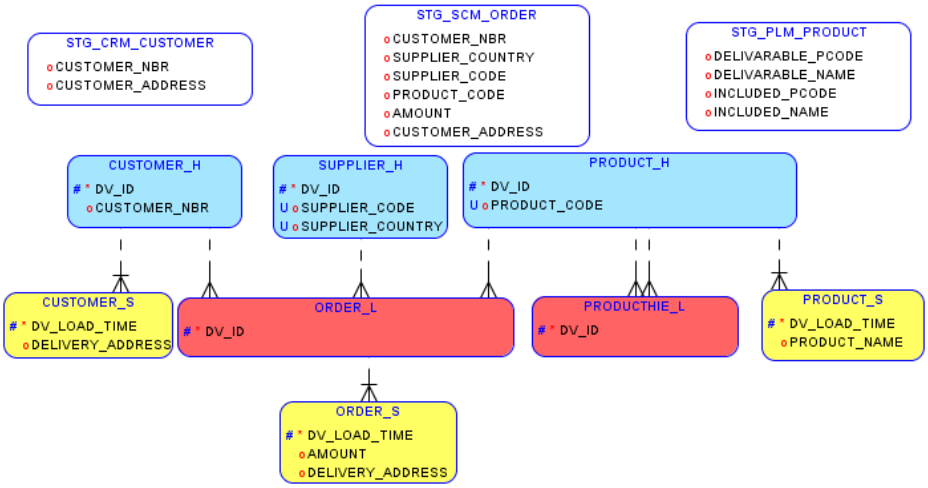

Figure 2. Customer - Supplier - Product example

CRM Populate Structure The source is `STG_CRM_CUSTOMER` and targets are `CUSTOMER_H` and `CUSTOMER_S`. We populate the structure information with the following commands. After the structure information population we create the current view for the satellite.

```
1 /* populate structure */
2 populate_entity('STG_CRM_CUSTOMER');
3 populate_entity('CUSTOMER_S');
4 /* create current views for satellites */
5 create_curr_views;
```

CRM Populate Data Flow Populating data flow information for CRM can be done with one command.

```
1 populate_transformation('STG_CRM_CUSTOMER', 'CUSTOMER_S');
```

Based on the model rules, transformations from `STG_CRM_CUSTOMER` to `CUSTOMER_S` and `CUSTOMER_H` are created. In addition, the mapping for surrogate key attribute is populated for these transformations.

For transformation attribute level population, data map suggests population code. Based on similar naming, the `CUSTOMER_H.CUSTOMER_NBR` is sug-

gested correctly. CUSTOMER_ADDRESS needs to be chosen to be mapped to CUSTOMER_S.DELIVERY_ADDRESS. Missing attribute mappings for populated entity level transformations may be queried from the data map.

Code generation for insert views and functions is possible after the data flow is populated into the data map. Insert views and functions are generated for both transformations from STG_CRM_CUSTOMER to CUSTOMER_S and CUSTOMER_H.

PLM Populate Structure To populate PLM entities use the following commands.

```
1 populate_entity('STG_PLM_PRODUCT');
2 populate_entity('PRODUCT_S');
3 populate_entity('PRODUCTHIE_L');
```

The dependant PRODUCT_H is populated on row 2 based on model rules when PRODUCT_S is populated. The link PRODUCTHIE_L population is satisfied with the earlier population of the hub because the row 2 populate the hub. If the order in populating script would be different between PRODUCT_S and PRODUCTHIE_L the hub PRODUCT_H would be populated in that entity which is executed first.

Based on the populated data map structure part, a PRODUCT_SC current view for the satellite may be generated.

PLM Populate Data Flow Transformations from STG_PLM_PRODUCT to PRODUCT_H, PRODUCT_S and PRODUCTHIERARCHY_L will be generated. There will be two transformations to both PRODUCT_H and PRODUCT_S. One pair for staging table included and deliverable columns.

```
1 populate_transformation('STG_PLM_PRODUCT','PRODUCTHIE_L');
2 populate_transformation('STG_PLM_PRODUCT','PRODUCT_S');
3 populate_transformation('STG_PLM_PRODUCT','PRODUCT_S');
```

Row 1 generates two transformations to PRODUCT_H and links the surrogate key mapping for PRODUCTHIE_L attribute to foreign key reference.

In row 2 there already is a hub transformation for PRODUCT_H from the same STG_PLM_PRODUCT table. The surrogate key reference mapping cannot be done automatically. From data map, it is possible to create suggestions that there are two candidate hub transformations available for this satellite transformation. The user may choose the hub transformation that is used in deliverable part of the transformation. Based on that the user maps the attribute level mapping from DELIVARABLE_NAME to PRODUCT_NAME.

Row 3 will generate a satellite transformation for the included_name mapping. Also here the surrogate key mapping needs to be chosen and the INCLUDED_NAME to PRODUCT_NAME need to be mapped at attribute level.

The mapping of data map table is now populated for STG_PLM_PRODUCT source entity transformations, which is visualized in Table 3.

SCM Populate Structure SCM entities are populated with commands.

```
1 populate_entity('STG_SCM_ORDER');
2 populate_entity('ORDER_S');
```

Table 3. Mapping table contents for STG_PLM_PRODUCT source entity transformations

TR	TARGETENTITY	TARGETATTRIB	SOURCEATTRIB	DVIDTR
PPD4	PRODUCT_H	PRODUCT_CODE	DELIVARABLE_PCODE	
PPD5	PRODUCT_H	PRODUCT_CODE	INCLUDED_PCODE	
PPL3	PRODUCTHIE_L	DV_ID_PROD_DELI		PPD4
PPL3	PRODUCTHIE_L	DV_ID_PROD_INCL		PPD5
PPS6	PRODUCT_S	DV_ID		PPD4
PPS6	PRODUCT_S	PRODUCT_NAME	DELIVARABLE_NAME	
PPS7	PRODUCT_S	DV_ID		PPD5
PPS7	PRODUCT_S	PRODUCT_NAME	INCLUDED_NAME	

Entity `ORDER_L` is populated based on the model rules. If the previous CRM and SCM parts are already in the data map model, the `CUSTOMER_H` and `PRODUCT_H` entities are populated already into the model. If the SCM part is developed privately without first populating other parts, also `CUSTOMER_H` and `PRODUCT_H` entities are populated. This way it is possible to develop transformations in parallel. Now, current views for the link and the satellite are ready to be generated.

SCM Populate Data Flow Transformations from `STG_SCM_ORDER` to `CUSTOMER_H, SUPPLIER_H, PRODUCT_H, ORDER_L` and `ORDER_S` will be created with one command.

```
1  populate_transformation('STG_SCM_ORDER', 'ORDER_S');
```

All other attributes except `CUSTOMER_ADDRESS` match with the naming so data map suggest the mappings. If the CRM part is already stored in the mapping table, also the mapping from `STG_SCM_ORDER.CUSTOMER_ADDRESS` to `ORDER_S.DELIVERY_ADDRESS` may be suggested. This is based on the previous usage of similar naming in `STG_CRM_CUSTOMER` to `CUSTOMER_S` transformation. Artifacts for all five transformations are now ready to be generated.
`SUPPLIER_H` has two column business key. If the hash primary keys are used, the column ordering is taken from the order of the hub column order. Even thou the ordering of the columns is different in the staging table.

6. Discussion

In this paper we show how to generate code for ELT transformations from a staging table to a data vault table. This is enabled by the presented data map, resulting from the introduced model for storing structure and dataflow information. The power of the data map is in using both structure and data flow information instead of relying to only one of them. It is possible to make also changes by using data map, although this is not its initial purpose. Instead, there are various case tools available for managing the structural information. The tools also enable structural changes in a flexible fashion. Moreover, such tools can also store data flow information. Nevertheless, the data flow information is only presented graphically, and it is not possible to extract out of the tool easily in a similar way as with our solution data model.

In addition to the above tools, there are ETL tools that are intended for transformations. Such ETL tools include information similar to our data map.

These ETL tools often introduce a vendor lock-in situation, and work done using one tool is not modifiable with other tools. Consequently, internal models cannot be exported from currently available tools. Nevertheless, there are some ways to tackle the vendor lock-in. For example, [1] provides BPMN4ETL design model for ETL processes which allows transporting between several technologies. In BPMN4ETL a ETL process is designed manually, we are creating transformations based on data vault principles. In theory, BPMN4ETL could be generated based on data map model information.

Even if data map could be developed in data first fashion, we chose to use model first approach. Model first approach create data warehouse structure which is more timeless than data model created based on source systems models. These strategies have different ways regarding how the data map is populated. The structure information in data maps enables generating statements that create tables for a data warehouse database model, which is very convenient when develop in data first fashion.

It is worth noting that the staging table columns may include null values, which are then populated to hubs and satellites. Initially, all values are unknown as there are no rows in the target model. In case of satellite tables, null values override possible previously known values, whereas in hubs null values should actually be stored in a data vault. There is also a common usage pattern to replace nulls with some fixed characters, like -1 for an example. However, this character is then not available for any other use. We have chosen to allow null values as a business key and tie that to a hash binary that is not produced from actual data. In this case, the hash of null value is defined to be also null.

As described earlier, it is possible to generate the transformations by using both the existing data warehouse data and data flow information. At the moment, we generate the following items based on data map data:

- data warehouse create table statements,
- current views for satellites,
- current views for links,
- insert views for all data vault tables and
- transformation functions which use insert view and have error handling for parallelizing the ETL loads.

In addition, we can also generate other interesting structures from on the data map. Most of the ETL tools can read XML formatted transformation specifications. We can generate the specifications with data map information in XML format. This way we can actually execute the transformations with ready ETL tools.

Data flow information includes data lineage information which can be presented graphically with different tools, like DOT[5] language.

ETL has been historically based on batch load operations. Nevertheless, there is an ongoing trend towards online processing, which requires streaming the data. In this case, we do not have separate extract or loading phases in ETL, but the whole process is about transformation. Consequently, the data map approach is readily available for streaming transformation generation as well.

[5] http://www.graphviz.org/doc/info/lang.html

Finally, it is important to consider the tradeoff between manual work and work needed for automating the tasks. Obviously, the automation effort should be placed on development steps that require a lot of manual work. Nevertheless, automation also introduces other benefits. The resulting code systematically similar for all generated objects. The coded objects are therefore often easier to maintain, even if manual work is required.

7. Future Work

Generating data transformations out of a data vault will be the next long-term goal of this work. The first step is to implement an inverse transformation of a staging interface, so the data vault model may be published with a similar interface as is used when storing the data.

As described and discussed in this article, transforming data to a data vault structure splits the entities to smaller groups of columns. This way a single executable transformation takes place between a staging table and a data vault table. Mappings between source and target columns are stored in data map data model, with information of access paths to resolve surrogate key values in foreign keys. The introduced data map data model in this article supports the table splitting transformation. When getting data out of the data vault, the entities are put together. Such join transformations would consist of several data vault tables as the input. The data map model described in this paper cannot store such information. The model needs some minor changes to accomplish these requirements.

8. Conclusion

In data warehousing, data vault modeling is widely used methodology. The data is divided into four different entities, which are hub, link, satellite, and reference. A data warehouse that is modeled using data vault modeling technique can be flexibly modified if the source systems change. As a cost of the flexibility, we have significantly increased the amount of data load operations and transformations, which requires reducing the amount of manual work involved in these phases.

Fortunately, the manual work for the data transformation development can be mitigated. We note that data vault methodology gives certain principles regarding how the different entities should be populated. By using these principles, the metadata of data models, and data flow information it is possible to semi-automatically generate the actual data transformations. To support this, in this paper we have introduced a metadata model which allows the transformations code generation.

In our approach, the amount of needed manual work is reduced but not completely eliminated. Nevertheless, even for the manual work phase, with data flow information we can give good proposals based on the data map metadata. As there are several exceptions which are easy for human to recognize to instruct to a machine, the proposed data flow information needs at times to be accepted by a human.

Using data model information in generating data transformations introduces several advantages. By reducing manual work, we also reduce the possibilities for human errors. Transformations can also be generated as large sets, whereas manually implemented transformations have to be done one by one. Furthermore, transformations generated from data map are readily available for commercial production environments.

Acknowledgements

The authors would like to express deep gratitude to Janne Pirkkanen for his encourage to think differently, he was the person who have inaugurate the systematic way of doing data warehousing.

The work was financially supported by TEKES (Finnish Funding Agency for Innovation) DIGILE Need for Speed program. We would also like to thank Solita for the possibility to do this research.

References

[1] Zineb El Akkaoui, Esteban Zimànyi, Jose-Norberto Mazón, and Juan Trujillo. A model-driven framework for etl process development. In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*, pages 45–52. ACM, 2011.

[2] Hans Hultgren. *Modeling the Agile Data Warehouse with Data Vault*. New Hamilton, 2012.

[3] Vladan Jovanovic and Ivan Bojicic. Conceptual data vault model. In *SAIS Conference, Atlanta, Georgia: March*, volume 23, pages 1–6, 2012.

[4] Dan Lindstedt and Kent Graziano. *Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault*. CreateSpace, 2011.

[5] Dan Linstedt. Data vault series 1–data vault overview. *The Data Administration Newsletter*, 2002.

[6] Dan Linstedt and Michael Olschimke. *Building a Scalable Data Warehouse with Data Vault 2.0: Implementation Guide for Microsoft SQL Server 2014*. Morgan Kaufmann, 2015.

[7] Dan Linstedt, Kent Graziano, and Hans Hultgren. The new business supermodel, the business of data vault modeling. *Lulu. com*, 2008.

[8] Ivan Pankov, Pavel Saratchev, Filip Filchev, and Paul Fatacean. Towards a generic metadata warehouse.

[9] Cassandra Phipps and Karen C Davis. Automating data warehouse conceptual schema design and evaluation. In *DMDW*, volume 2, pages 2–2. Citeseer, 2002.

[10] Rick F. van der Lans. *Data Vault and Data Virtualization: Double Agility*, March 2015 (accessed 4 january 2016). URL `https://www.cisco.com/web/services/enterprise-it-services/data-virtualization/documents/whitepaper-cisco-datavaul.pdf`.