Data Warehouse Design: A schema-transformation approach

Adriana Marotta, Raul Ruggia
Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Montevideo, Uruguay
adriana@fing.edu.uy

Abstract

This paper addresses DW design problems, with the goal of improving the DW logical design process. Some of the existing work in transformation oriented methodologies for DW design construct the DW starting from an Entity-Relationship model of the source database, and arrive to a conceptual or high-level-logical dimensional model of the DW. We propose a mechanism for obtaining the DW logical schema through pre-defined transformations applied to the source logical schema, which can be used as a complement to the existing DW design methodologies. The transformations allow a refined logical design of the DW and provide a trace of the design and a mapping between the source and DW logical structures.

1. Introduction

A Data Warehouse (DW for short) is a Database that stores information in order to satisfy decision-making requests. This kind of Database has the following particular features. It contains data that is the result of transformations, quality improvement, and integration of data that comes from operational bases, also including indicators that give it additional value. The DWs have to support complex queries (summarisation, aggregates, crossing of data), however its maintenance does not suppose transactional load. These features cause the design techniques and the used strategies to be different from the traditional ones.

In this paper we address DW design issues. Our goal is to improve the logical design process by providing a transformation-oriented mechanism for constructing complex DW schema structures, which leaves a mapping between source logical schema and DW logical schema.

Database design methodologies based on *transformations* are those where the desired schema is constructed through successive application of transformations to sub-schemas. Such schema transformations may be applied from conceptual to logical schemas or from logical to logical schemas. We think this is an interesting approach for the case of DWs, which have a dependence on the source databases and may be constructed starting from them.

Some of the existing work in transformation oriented methodologies for DW design [6], [7], [2], construct the DW starting from an Entity-Relationship model of the source database, and arrive to a conceptual or high-level-logical dimensional model of the DW. In [6] they describe a method where the first step involves classifying entities of the ER-enterprise model in a number of categories, the second step involves identifying hierarchies that exist in the model and the final step involves collapsing these hierarchies and aggregating transaction data. In [7] they propose a method that starts applying transformations to the ER-enterprise model until they obtain a representation of the corporate dimensions, then, basing mainly in the gathered user requirements, they design the dimensional model. The methodology proposed in [2] starts from an ER-enterprise model. It re-structures this model, transforming it until they obtain a conceptual schema in MD (a conceptual model they define). Then, they provide a method to pass from this model to a logical dimensional model (in particular the star schema in the relational model).

We believe that some main aspects related to DW design are not covered by these proposals: the generation of mappings between source and DW logical structures, and the construction of complex DW structures. The former is necessary for solving the problems of data loading, source schema evolution, and error detecting [5]. With the latter we refer, for example, to explicit structures for historical data management, dimension versioning, calculated data, key generalizations. In [8] logical design techniques for this kind of problems are presented by means of examples.

In our work we propose a mechanism for DW logical design that intends to provide: (1) design traceability, (2) a mapping between source logical schema and DW logical schema, and (3) facilities for designing complex DW structures. It is supposed to be used as a complement to the previously commented existing methodologies. The mechanism is based on a set of pre-defined schema transformations that enables to build the DW logical schema from the source logical schema. The mechanism includes, for the utilisation of the proposed transformation set, two types of guidelines: *consistency rules,* which are applied in order to ensure the consistency of the obtained schema, and *design strategies*, which provide different solutions to typical DW design problems, by application of the transformations.

The proposed mechanism may be applied in different DW design scenarios. We following describe which may be a typical one. The design process consists of two steps: (1) apply one of the existing methodologies for designing the DW schema starting from the source conceptual schema, (2) having the DW schema designed in the previous step as the target schema, build it through application of transformations to the logical source schema, and (3) apply other necessary transformations so that the DW schema is refined (or optimised) according to the requirements. Figure 1 presents the DW design process showing the alternative paths that can be followed.

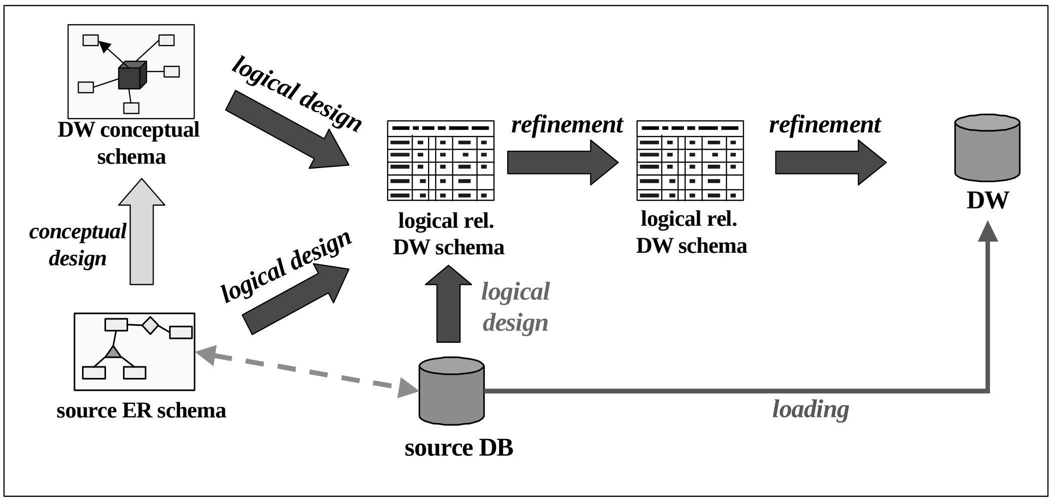

Figure 1: DW design process

The remainder of paper is organized as follows. Section 2 presents the proposed environment for DW design from the logical source schema, Section 3 presents a simple example that illustrates the design process and Section 4 presents the conclusions.

2. The proposed DW design environment

In this section we present the environment that supports the process of DW logical design, starting from the source logical schema. The DW schema is generated by application of transformations to the source schema and to the intermediate sub-schemas that are generated through the process, i.e. during the design the transformations are composed to obtain the target schema. Figure 2 shows the basic architecture of the transformation.

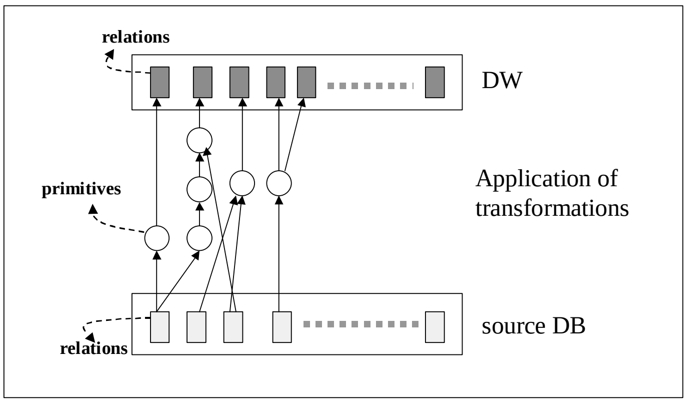

Figure 2: Architecture of the transformation

The transformations take as input a relational sub-schema and their output is another relational sub-schema. They also give as output an outline of the corresponding instance transformation. Some of them are grouped into families. Transformations belonging to the same family correspond to different alternatives or different design styles for solving the same problem.

2.1. Basic Definitions

The underlying model for the proposed transformations is the Relational Model. However, we classify the relational elements (relations and attributes) into different sets, according to dimensional concepts (e.g.: dimension relations, measure relations, descriptive attributes, measure attributes). This classification enables the transformations to perform a more refined treatment of the different situations in DW design.

The Sets defined over the Relational Model

Relation[1] sets:

Rel	- Set of all the relations (any kind of relation).
Rel_D	- Set of "dimension" relations. These are the relations that represent descriptive information about real world subjects.
Rel_C	- Set of "crossing" relations. These are the relations that represent relationships or combinations among the elements of a group of dimensions. Usually, they contain attributes that represent measures for the combinations.
Rel_M	- Set of "measure" relations. These are the crossing relations that have one or more measure attributes.
Rel_J	- Set of "hierarchy" relations. These are the dimension relations that contain a set of attributes that constitute a hierarchy.
$Rel_H(R)$	- Set of "history" relations. These are the relations that have historical information that correspond to information in relation R.

These sets verify: $Rel_M \subset Rel_C$, $Rel_J \subset Rel_D$, $\forall$ R $Rel_H(R) \subset (Rel_D \cup Rel_C)$

Attribute sets:

$Att(R)$	– Set of all attributes of relation R.
$Att_M(R)$	– Set of measure attributes of relation R.
$Att_D(R)$	– Set of descriptive attributes of relation R.
$Att_C(R)$	– Set of derived (calculated) attributes of relation R.
Att_J	– Set of sets of attributes that represent a hierarchy.
$Att_K(R)$	– Set of sets of attributes that are key in relation R.
$Att_{FK}(R)$	- Set of sets of attributes that are foreign key in relation R.
$Att_{FK}(R_1,R_2)$	– Set of attributes that are foreign key in relation R_1 with respect to relation R_2.

These sets verify: - $Att_M(R) \cup Att_D(R) \cup Att_C(R) = Att(R)$

[1] In this work, we use the word relation as a synonym of relation schema.

- $\forall$ X / X $\in$ Att_J, X $\subset$ $\cup_{R \in Rel}$ $Att_D(R)$, $Att_{FK}(R)$ = {e / e = $Att_{FK}(R, R_i)$}, i=1..n, where n is the number of relations with respect to which R has a foreign key.
- $\forall$ A / A $\in$ X and X $\in$ $(Att_K(R) \cup Att_{FK}(R))$, A $\in$ $Att_D(R)$
- If X $\in$ $Att_K(R)$ and Y $\in$ $Att_{FK}(R)$, it may be: X $\cap$ Y $\neq \varnothing$

We also define a set of Schema Invariants. This is a set of properties that must be satisfied by a relational DW schema for being consistent. They concern: referential integrity, hierarchies between attributes, the relationship between a relation and its corresponding history relation, and the dependencies between attributes of a measure relation. Due to space limitations we present only one of the invariants (for the whole set refer to [9]).

If a measure relation has an attribute that also belongs to a dimension relation, then it must have a foreign key relative to this relation.

Let R_D, R_M / $R_D \in Rel_D$ $\wedge$ $R_M \in Rel_M$
    if $\exists$ A / A $\in$ $Att(R_D)$ $\wedge$ A $\in$ $Att(R_M)$ $\Rightarrow$ $\exists$ X / X = $Att_{FK}(R_M, R_D)$

2.2. The transformations

We start presenting some example-transformations in order to illustrate their usefulness.

Many relations in operational systems do not maintain a temporal notion. For example, stock relations use to have the current stock data, updating it with each product movement. However, in DWs most relations need to include a temporal element so that they can maintain historical information. For this purpose, there is a transformation called **Temporalization** that adds an element of time to the set of attributes of a relation.

In operational systems, usually, data is calculated from other data at the moment of the queries, in spite of the complexity of some calculation functions, in order to prevent any kind of redundancy. For example, the product prices expressed in dollars are calculated from the product prices expressed in some other currency and a table containing the dollar values. In a DW system, sometimes it is convenient to maintain this kind of data calculated, for performance reasons. We have a group of transformations, which name is **DD-Adding**, that add to a relation an attribute that is derived from others.

Figure 3 shows a table containing the whole set of transformations proposed. The transformation names marked with a "*" symbol correspond to transformation families.

Transformation		Description
T1	**Identity**	Given a relation, it generates another that is exactly the same as the source one.
T2	**Data Filter**	Given a relation, it generates another one where only some attributes are preserved. Its goal is to eliminate purely operational attributes.
T3	**Temporalization**	It adds an element of time to the set of attributes of a relation.
T4	**Key Generalization ***	They generalize the primary key of a dimension relation, so that more than one tuple representing the same real-world subject, can be stored.
T5	**Foreign Key Update**	A foreign key and its references can be changed in a relation. This is useful when primary keys are modified.
T6	**DD-Adding ***	They add to a relation an attribute that is derived from others.
T7	**Attribute Adding**	It adds attributes to a dimension relation. It is useful for maintaining more than one version of an attribute in the same tuple.
T8	**Hierarchy Roll Up**	It does the roll up by one of the attributes of a relation following a hierarchy. Besides, it can generate another hierarchy relation with the corresponding grain.
T9	**Aggregate Generation**	Given a measure relation, it generates another measure relation, where data is resumed by a given set of

		attributes.
T10	**Data Array Creation**	Given a relation that contains a measure attribute and an attribute that represents a pre-determined set of values, it generates a relation with a data array structure.
T11	**Partition by Stability***	They partition a relation in order to organize its history data storage. Vertical Partition or Horizontal Partition can be applied.
T12	**Hierarchy Generation ***	They generate hierarchy relations, having as input relations that include a hierarchy or a part of one.
T13	**Minidimension Break off**	It eliminates a set of attributes from a dimension relation, constructing a new relation with them.
T14	**New Dimension Crossing**	It allows materialising a dimension data crossing in a new relation.

Figure 3: The transformations

We have not addressed the demonstration of the completeness of the set of transformations. However we believe that the usefulness of this set is supported by the following facts: (i) it was designed basing in general DW design techniques existing in the bibliography [8,10,11,12,13], (ii) it has been tried on many different study cases, and (iii) the proposed *design strategies* show how a wide spectrum of DW design problems can be solved through application of the transformations.

Figure 4 shows the specification of one of the transformations (whole set can be found in [9]).

T9 – Aggregate Generation

Description:

Given a measure relation, the transformation generates another measure relation, where data is summarised (or grouped) by a given set of attributes.

Input:

- source schema: $R(A_1,...,A_n) \in Rel_M$
- Z, set of attributes / card(Z)= k (measures)
- $\{e_1,...,e_k\}$, aggregate expressions
- $Y / Y \subset (Att_D(R) \cup Att_M(R))$, attributes to be removed
- source instance: r

Resulting schema:

$R'(A'_1,...,A'_m) \in Rel_M / \{A'_1,...,A'_m\} = \{A_1,...,A_n\} - Y \cup Z$

Generated instance:

```
r′ = select ({A′1,...,A′m} – Z) ∪ {e1,...,ek}
     from R
     group by {A′1,...,A′m} – Z
```

Figure 4: Specification of transformation T9

2.3. Consistency Rules and Design Strategies

Basing on the Invariants, we define some rules that should be applied always, when a DW schema is being constructed through application of the transformations. The rules consider the different cases of inconsistencies that can be generated by application of the transformations and state the actions that must be performed to correct them. The following is one of these rules (the rest can be found in [9]):

R1 – Foreign key updates

R1.1 – ON APPLICATION OF: *Temporalization* (adding the time attribute to the key) or *Key Generalization* to R, where X = old key and Y = new key
APPLY: *Foreign Key Update* to all R_i / $Att_{FK}(R_i,R)$ = X, obtaining $Att_{FK}(R_i,R)$ = Y

Strategies for application of the transformations were designed taking into account some typical problems of Data Warehousing. The strategies proposed address design problems relative to: dimension versioning, versioning of N:1 relationships between dimensions, data summarisation and data crossing, hierarchies' management, and derived data. We following show part of one of the proposed strategies (refer to [9] for the whole):

```
S1 -   Dimension versioning

A usual problem DW designers have to face is how to manage dimension
versioning. This refers to how dimension information must be structured when
its history needs to be maintained. The idea is to maintain versions of each
real-world subject information.

Several alternatives are provided:

S1)  Apply Temporalization  (T3), such that the time attribute belongs to the
     key of the relation.
S2)  Generalise the key of the dimension relation through one of the
     transformations of Key Generalization family (T4). The two options are:
   2.1)  Apply Version Digits (T4.1), so that version digits are added to the
         key.
   2.2)  Apply Key Extension (T4.2). In this case new attributes of the
         relation are included in the key.
.........................
```

3. Presenting the mechanism through an example

In this section we present an example that shows when and how the schema transformations can be applied.

3.1. Introduction

This is a simplified case of a product distribution company who wants to construct a DW. The most important requirements are related to: (i) sales evolution by product families and geographic regions, (ii) product cost analysis, (iii) market analysis (by types of customers), and (iv) geographic distribution of the sales.

The source database schema is shown in Figure 5.

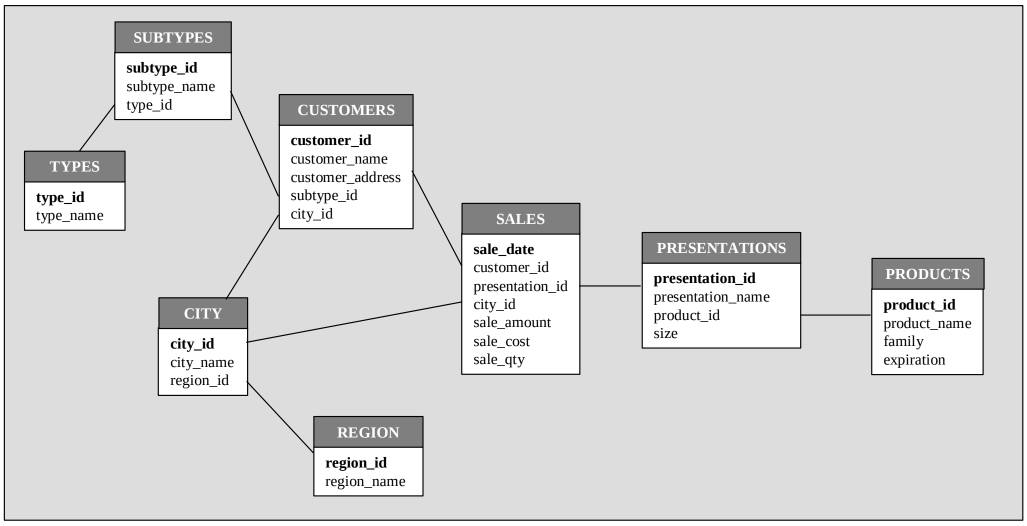

Figure 5: The source database schema

We suppose that, following one of the existing DW design methodologies [6,7,2], we arrive to the design presented in Figure 6. It is a star schema[2], where the dimensions are Time, Customers_DW, Products_DW, and Geography, and the fact table is Sales_DW, where sale_amount, sale_cost and sale_qty are the measures.

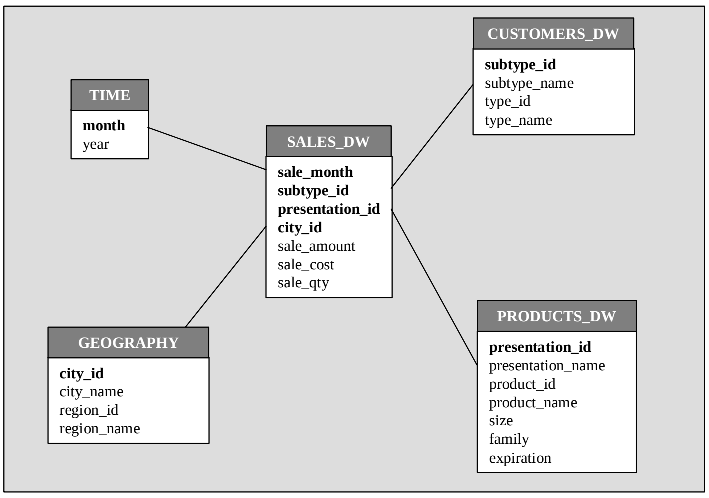

Figure 6: The target DW schema

3.2. Generation of the DW schema through schema transformations

Now, we apply the transformations to the source schema generating the desired DW relational schema.

First, we de-normalise the relations that correspond to the dimensions.

```
We apply T6.2 DD-Adding-1N to Presentations and Products, obtaining:
PRODUCTS_DW (presentation_id, presentation_name, product_id,
              product_name, size, family, expiration)

We apply T6.2 to Customers, Subtypes and Types, obtaining:
CUSTOMERS_DW_01 (customer_id, customer_name, customer_address,
                  subtype_id, city_id, subtype_name, type_id, type_name)

We apply T6.2 to relations City and Region, obtaining:
GEOGRAPHY (city_id, city_name, region_id, region_name)
```

CUSTOMERS_DW_01 has some attributes that are not relevant for this case.

```
We apply T2 Data Filter to relation Customers_DW_01, obtaining:
CUSTOMERS_DW_02 (customer_id, subtype_id, subtype_name, type_id,
                  type_name)
```

For the *Time* dimension we obtain the *date* attribute from the *Sales* relation. Then we calculate the attributes *month* and *year* from the *date*.

```
We apply T12.1 Hierarchy Generation to Sales, obtaining: TIME_01 (date)
and we apply twice T6.1 DD_Adding-11 to TIME_01 for adding atts month and
year:
TIME_02 (date, month)
TIME_03 (date, month, year)
```

We generate the fact table (measure relation) *Sales* with the desired granularity, which is *subtype* for *Customer* dimension and *month* for *Time* dimension.

```
We apply T8 Hierarchy Roll-Up to Sales and Customers_DW_02, obtaining:
```

[2] Star Schema is defined in [Kim96]

```
SALES_DW_01  (sale_date,  subtype_id,  presentation_id,  city_id,
              sale_amount, sale_cost, sale_qty)
and
CUSTOMERS_DW (subtype_id, subtype_name, type_id, type_name)
```

```
We apply T8 to Sales_DW_01 and Time_03, obtaining:
SALES_DW (sale_month, subtype_id, presentation_id, city_id, sale_amount,
         sale_cost, sale_qty)
and
TIME (month, year)
```

Through the applied transformations we generated the desired schema, showed in Figure 6.

Now we will refine the design. One of our requirements is about products' history. For query performance reasons, we decide to maintain this history data in a separate relation.

```
We apply T11.2 Horizontal Partition to Products_DW, obtaining:
PRODUCTS_DW_HIS_01 (presentation_id, presentation_name, product_id,
                      product_name, size, family, expiration)
```

```
We apply T3 Temporalization to Products_DW_His_01, obtaining:
PRODUCTS_DW_HIS (presentation_id, change_date, presentation_name,
                  product_id, product_name, size, family, expiration)
```

Finally, also for performance reasons, we want to add to *Geography* relation a calculated attribute *cust_qty*, which represents the quantity of customers that belongs to each city.

```
We apply T6.3 DD_Adding-NN to Geography and Customers, obtaining:
GEOGRAPHY_CUST (city_id, city_name, region_id, region_name, cust_qty)
```

The final DW schema is shown in Figure 7.

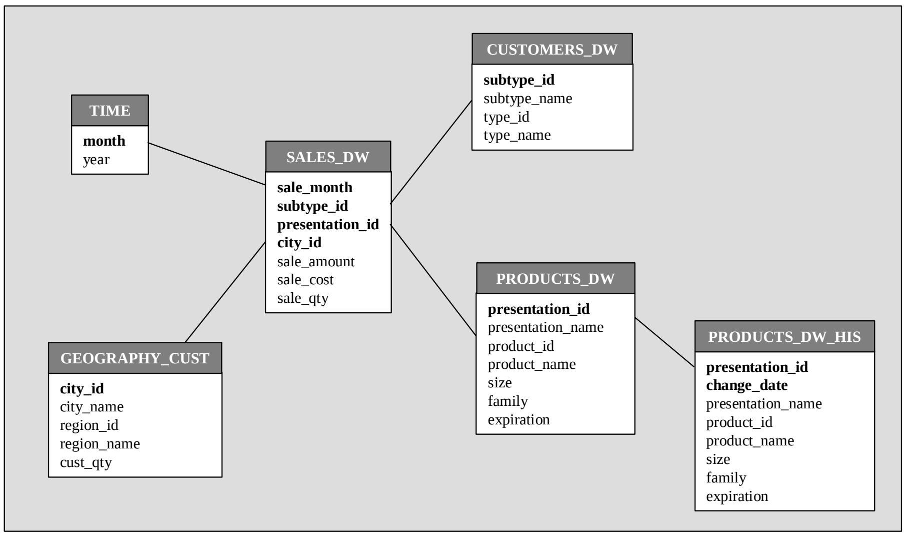

Figure 7: The final DW schema

3.3. The generated trace

The applied transformations generate a trace of the design, which is shown in Figure 8.

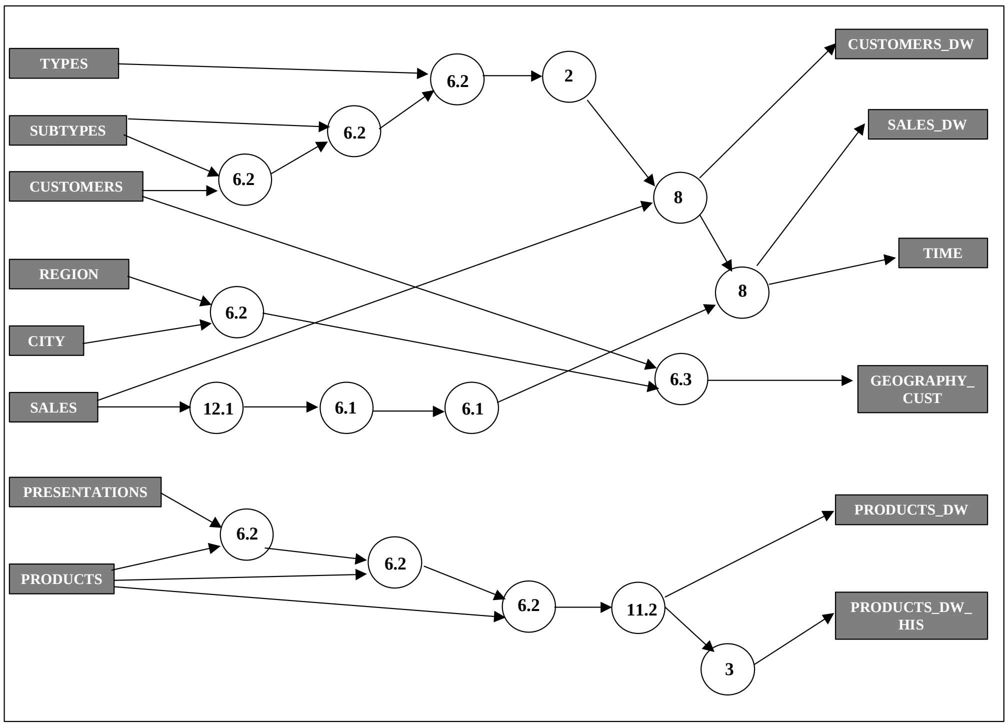

Figure 8: The generated trace

For each element of the final schema (relation or attribute) there is a transformation trace that can be seen as the path that was followed for obtaining this element starting from a source element. This trace provides the information about the sequences of primitives that were applied to the element, as well as the sub-schemas and other arguments the primitives received in each case.

We specify the trace of the design through a set of expressions in the form of function applications [9] and also through a graphic representation (Figure 8). The latter is a directed acyclic graph *G(T),* which main goal is to show a global perspective of the process, while facilitating the comprehension and localisation of each element's trace.

4. Conclusion

This paper presents a set of schema transformations as well as some strategies and rules for their practical application. These pre-defined transformations enable to design a DW from a relational source schema acting as design building blocks that embed DW design knowledge in their semantics.

The proposed transformations are methodologically neutral, and may be applied in different scenarios with other design strategies.

The use of these transformations enables to obtain a DW schema as well as to keep the design trace and mapping between the source schema structures and the DW schema ones. In addition, the proposed transformations enable to easily obtain complex DW structures.

Design traceability, which allows documentation of the design and can be useful for design process reuse, is obtained through the transformation application trace. This trace is of great importance also for DW management, since it provides the mapping between source and DW schema elements. This mapping is necessary at least for solving the problems of data loading processes, source schema evolution, and error detecting. In [14] we present a proposal for DW evolution in the context of Web Warehouses where the DW is designed through the transformations and the trace is used for propagating source evolution.

We have implemented the transformations in a DW design tool that enables the designer to apply them through a graphical interface [15,16]. The tool also counts with the consistency rules, which are triggered when inconsistencies are generated in the DW schema, and with schema evolution facilities.

We are currently working on DW evolution generated by source schema evolution, and on including integration functionalities to the set of transformations.

References

[1] L. Cabbibo, R. Torlone. *A Logical Approach to Multidimensional Databases.* EDBT' 98. Springer-Verlag, 1998.

[2] M. Golfarelli, S. Rizzi. *A Methodological Framework for Data Warehouse Design.* DOLAP, ACM 1998.

[3] C. Sapia, M. Blaschka, G. Hofling, B. Dinter. *Extending the E/R Model for the Multidimensional Paradigm.* LNCS Vol 1552, Springer-Verlag, 1999.

[4] F. Carpani, R. Ruggia. *An Integrity Constraints Language for a Conceptual Multidimensional Data Model.* XIII International Conference on Software Engineering & Knowledge Engineering. SEKE'01. Bs. As. Argentina.

[5] Mokrane Bouzeghoub, Zoubida Kedad. *A quality-based framework for physical data warehouse design.* Proc. CAISE '00 Workshop on Design and Management of Data Warehouses (DMDW '00).

[6] D. L. Moody, M. A. R. Kortink. *From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design.* Proc. CAISE '00 Workshop on Design and Management of Data Warehouses (DMDW '00).

[7] Chuck Ballard. *Data Modeling Techniques for Data Warehousing.* SG24-2238-00. IBM Red Book. ISBN number 0738402451.

[8] R. Kimball. *The Data Warehouse Toolkit.* J. Wiley & Sons, Inc. 1996

[9] A. Marotta. *Data Warehouse Design and Maintenance through Schema Transformations.* Master Thesis - 2000. Instituto de Computación, Facultad de Ingeniería, Universidad de la República. Montevideo – Uruguay.

[10] R. Kimball. *The Data Warehouse Lifecycle Toolkit.* J. Wiley & Sons, Inc. 1998

[11] R. Kimball. *Data Warehouse Architect.* Column in DBMS online magazine. 1997

[12] C. Adamson, M. Venerable. *Data Warehouse Design Solutions.* J. Wiley & Sons, Inc. 1998

[13] L. Silverston, W. H. Inmon, K. Graziano. *The Data Model Resource Book.* J. Wiley & Sons, Inc. 1997.

[14] A. Marotta, R. Motz, R. Ruggia. *Managing Source Schema Evolution in Web Warehouses.* International Workshop on Information Integration on the Web, WIIW '2001.

[15] P. Garbusi, F. Piedrabuena, G. Vazquez. *Design and Implementation of a schema transformation based DW design tool.* Graduate Project of the Engineering Faculty – Montevideo – Uruguay. 2000.

[16] A. Alcarraz, M. Ayala, P. Gatto. *Design and Implementation of a Data Warehouse evolution tool.* Graduate Project of the Engineering Faculty – Montevideo – Uruguay. 2001.