Towards the Automation of Data Warehouse Logical Design: a Rule-Based Approach

Verónika Peralta, Alvaro Illarze, Raúl Ruggia
Instituto de Computación, Universidad de la República. Uruguay.
vperalta@fing.edu.uy, illarze@adinet.com.uy, ruggia@fing.edu.uy

Abstract. Data Warehouse logical design involves the definition of structures that enable an efficient access to information. The designer builds relational or multidimensional structures taking into account a conceptual schema representing the information requirements, the source databases, and non-functional requirements. Existing work in this area has mainly focused on aspects like data models, data structures specifically designed for DWs, and criteria for defining table partitions and indexes. This paper proposes a step forward to the automation of DW relational design through a rule-based mechanism, which automatically generates the DW schema by applying existing DW design knowledge. The proposed rules embed design strategies which are triggered by conditions on requirements and source databases, and perform the schema generation through the application of predefined DW design oriented transformations.

Keywords. Data Warehouse Design, Rule based design.

1 Introduction

DW logical design processes and techniques differ from the commonly applied on the OLTP database design. DW considers as input not only the conceptual schema but also the source databases and is strongly oriented to queries instead of transactions. Conceptual schemas are commonly specified by means of conceptual multidimensional models [1]. Concerning the logical relational schema [4][1], various data structures have been proposed in order to optimise queries [11]. These structures are the well-known star schema, snowflake, and constellations [11][3][13].

An automated DW logical design mechanism should consider as input: a conceptual schema, non-functional requirements (e.g. performance) and mappings between the conceptual schema and the source database. In addition, it should be flexible enough to enable the application of different design strategies. Furthermore, the mappings to the source databases should be stored in order to keep track of the exact expressions that define the required information items in terms of the source data items. Mappings are also useful to program the DW loading and refreshment.

This paper addresses DW logical (relational) design issues and proposes a rule-based mechanism to automate the generation of the DW relational schema, following the previously described principles.

Some proposals to generate DW relational schemas from DW conceptual schemas are presented in [8][5][3][10]. We believe that existing work lack in flexibility to apply different design strategies that enable to build complex DW structures (for example, explicit structures for historical data management, dimension versioning, or calculated data).

The here proposed mechanism is based on rules that embed logical design strategies and take into account three constructions: (1) a *multidimensional conceptual schema*, (2) *mappings*, which specify correspondences between the elements of the conceptual and source schemas, and (3) *guidelines*, which provide information on DW logical design criteria and state non functional requirements. The rules are triggered by conditions in these constructions, and perform the schema generation through the application of predefined DW design oriented transformations.

The main contribution of this paper is the specification of an open rule-based mechanism that applies design strategies, builds complex DW structures and keeps mappings with the source database.

The remaining sections of this paper are organized as follows: Section 2 presents an overview of the environment and section 3 describes the rule-based mechanism. Finally, Section 4 points out conclusions and future work.

2 The DW Logical Design Environment

Our underlying design methodology for DW logical design takes as input the source database and a conceptual multidimensional schema. The design process consists of three main tasks: (i) refine the conceptual schema adding non functional requirements and obtaining a "*refined conceptual schema*", (ii) map the refined conceptual schema to the source database, and (iii) generate the relational DW schema from the refined conceptual schema and the source database.

The conceptual schema is refined by adding *design guidelines* that cover non-functional requirements. A guideline may be – for example – an indication on how to fragment historical data [9] or to state the desired degree of normalization[13]. Guidelines constitute a flexible way to express design strategies and properties of the DW [14].

The following task consists in defining mappings between the refined conceptual schema and the source database, indicating how to calculate each multidimensional structure from the source database. Mappings are functions that associate each conceptual schema attribute (called item) with a mapping expression that can be an attribute of a source table (direct expression), a calculation that involve several attributes (calculated expression), an aggregation (aggregated expression) or an external value like a constant, a time stamp or version digits (external expression) [14].

Finally, the DW relational schema should be generated. In order to manually design the DW, the designer takes a sequence of decisions that transform a relational schema in each step, starting from the source schema. In doing so, he applies several transformations. Different transformations can be applied depending on the different input information and the design strategies.

For example, concerning the degree of normalization, there are several alternatives to define dimension and fact tables. If we want to denormalize a dimension, we have to store attributes corresponding to all the dimension items in the dimension table. This implies joining several tables and projecting the desired attributes. But if we want to normalize the dimension this transformation may not be needed. Analogous decisions can be taken for selecting which aggregates to materialize, fragmenting fact tables, filtering data, versioning or removing additional attributes.

We propose to automate most of the latter task by using a set of rules. The rules check which conditions are verified and determine which transformations have to be applied. The rule conditions involve the conceptual schema, the source database, mappings between them, and the design strategies expressed in the guidelines.

3 The Rule-based Mechanism

Different DW design methodologies [9][11][3][13][8][2] divide the global design problem in smaller design subproblems and propose algorithms (or simply sequences of steps) to order and solve them.

The here proposed mechanism makes use of rules to solve those concrete smaller problems (e.g. related with key generalization). In each design step, different rules are applied.

The proposed rules intend to automate the sequence of design decisions that trigger the application of schema transformations and consequently the generation of the DW schema. Alike the designer behaviour, rules take into account the conceptual and source schemas as well as the design guidelines and mappings, and then perform a schema design transformation.

Table 1 shows a table with the proposed set of rules.

Table 1. Design rules. The rules marked with * represent families of rules, oriented to solve the same type of problem with different design strategies.

	Rule	**Description**
R1	Merge	Combines two tables generating a new one. It is used to denormalize when a mapping function involves attributes from more than one table.
R2	Higher Level Names	Renames direct mapped attributes, using the item names. It is used when an item maps an attribute but the names are different.
R3	Materialize Calculation *	Adds an attribute to a table to materialize a calculation. It is used when an item maps to a calculated or aggregate expression.
R4	Materialize External Expression *	Adds an attribute to a table to materialize an external expression, such a constant, a time stamp or version digits. It is used when an item maps to an external expression.
R5	Eliminate Non Used Data	Deletes not mapped attributes grouping by the other attributes and applying a roll-up function to measures. It is used to delete not mapped attributes.
R6	Roll-Up *	Reduces the level of detail of a fact table, performing a roll-up of a dimension. It is used when a cube is expressed as an aggregate of another.
R7	Update Key	Changes a table primary key, to agree the key of a fragment or cube that it maps.
R8	Data Constraint	Deletes from a table the tuples that don't verify a condition. It is used to impose mapping constraints and to fragment a cube horizontally.

In order to generate the DW schema, the design rules invoke predefined schema transformations and the DW schema is generated by successive application of transformations to the source schema. We use the set of predefined transformations presented in [12], which take a sub-schema as input and generate another sub-schema as output, as well as an outline of the transformation of the corresponding instance. These schema transformations perform operations such as: table fragmentation, table merge, calculation, versioning, or aggregation.

Figure 1 sketches the context of a rule.

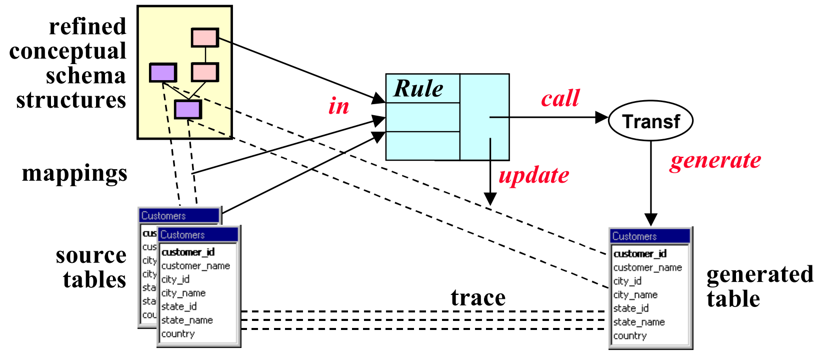

Figure 1. Context of a rule. Refined conceptual schema structures, source tables and mappings are the input of the rule. If conditions are verified, the rule calls a transformation, which generates a new table transforming the input ones. Mappings to the conceptual schema are updated, and a transformation trace from the source schema is kept.

Example. Consider a *customers* dimension with three levels: *state, city* and *customer* in which items map to two tables *Customers* and *Incomes*. Figure 2a shows the dimension (using CMDM notation [6]) and the source tables. Mappings are represented as arrows. The *income* item is defined as the sum of the *income* attribute of *Incomes* table.

We show the application of the rule R3.2 (MaterializeAggregate) of the family R3. The goal of this rule is to materialize a calculation. The conditions to apply the rule are: (i) the mapping function of a conceptual structure corresponds an item to an *aggregate* expression, and (ii) the *direct* expressions (corresponding to other items) reference only one table. The last condition assures that denormalization has been made before (The previous application of *merge* rule assures that this condition can be satisfied). The rule invokes the transformation T6.3.

The rule evaluates the conditions and applies the schema transformation DD-Adding N-N, which builds a new table with a calculated attribute (as *SUM(Incomes.income))*. The rule application also updates the mapping function to reference the attributes of the new table. Figure 2b shows the generated table and updated mapping function. □

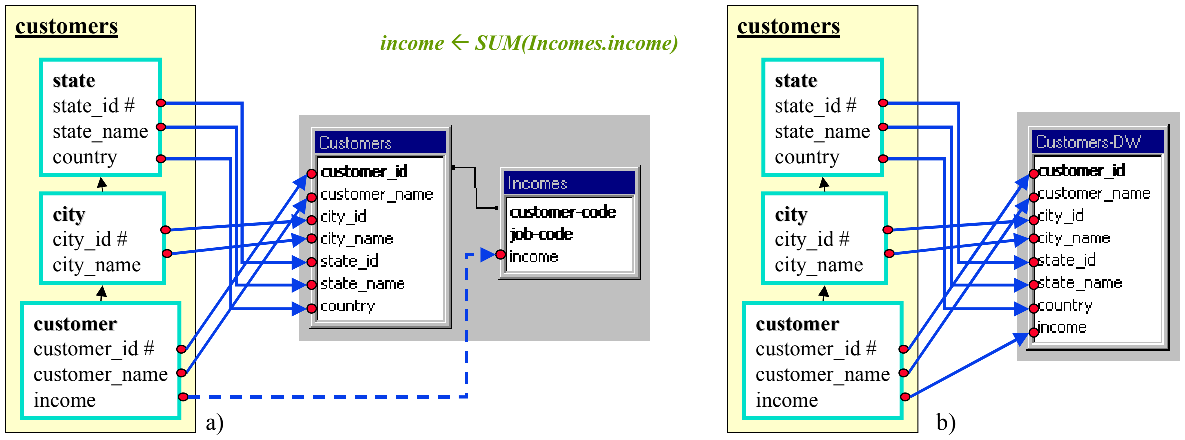

Figure 2. Application of a rule: Schemas and mappings a) before and, b) after applying the rule.

Rules are specified in 5 sections: *Name, Target, Input State, Conditions* and *Output State*. The *Name* section gives an idea of the rule behaviour. The *Target* section enumerates the refined conceptual schema structures that are going to be translated to the relational model. The *Input State* section enumerates the rule input state consisting of the mapping function of the target structures and tables referenced by the mapping functions. The *Conditions* section is a set of predicates that must be fulfilled before applying the rule. Finally, the *Output State* section is the rule output state, consisting of the tables obtained by applying transformations to the input tables, and mapping functions updated to reflect the transformation.

The following scheme sketches a rule specification.

RuleName:	Target Structures	Input State <mapping, tables>	conditions
		Output State <mapping, tables>	

Example. Rule 3.2 of previous example can be sketched as follows (see [14] for formal specification).

Aggregate Calculate:	S	<F, {T_1, T_2} >	All direct expressions refer to T_1
	I ∈Items(S)	<F', T'} >	F(I) is an aggregate expression that refer to T_2

The output state is calculated as follows:

$T' = \text{Transformation T6.3} (\{T_1, T_2\}, I = F(I), \text{Join-Condition} (T_1, T_2))$
$F' / F'(I) = T'.I \wedge \forall\ J \in \text{Items}(S), J \neq I\ .\ (F'(J) = \text{UpdateReference}(F,T,T'))$

The proposed rules do not include a fixed execution order, so an algorithm that orders their application is needed. The algorithm states in which order will the different design problems be solved. We propose an algorithm that is based on existing methodologies and some practical experiences. It first builds dimension tables, then builds fact tables, performs the aggregates and finally carries on horizontal fragmentations. Each algorithm step is associated to the application of one or more rules. The algorithm can be consulted in [14].

4 Conclusions

This paper presents a rule-based mechanism to automate the construction of DW relational schemas. The kernel of the mechanism consists of a set of design rules that decide the application of the suitable transformations to the source schema in order to solve different design problems. The overall framework integrates relevant elements in DW design: mappings between source and DW conceptual schemas, design guidelines that refine the conceptual schema, schema transformations which generate the target DW schema, and design rules that embed design strategies and call the schema transformations.

The proposed framework is a step forward to the automation of DW logical design. It provides an open and extensible environment that enables the application of different existing design techniques. Moreover, new design techniques can be integrated as new rules or transformations.

This framework has been prototyped [14] in a CASE environment for DW design. The system applies the rules and automatically generates the logical schema by executing an algorithm that considers the most frequent design problems suggested in existing methodologies and complemented with practical experiences. The prototype also includes a user interface for the definition of guidelines and mappings, an environment for the interactive application of the rules and the schema transformations and a graphical editor for the CMDM conceptual model.

In the near future we intend to enhance the capabilities of our framework, integrating further design techniques and extending it to cover the design of loading and refreshment tasks using the information provided by the mappings and the design trace of the transformations.

References

[1] Abello, A.; Samos, J.; Saltor, F.: "A Data Warehouse Multidimensional Data Models Clasification". Technical Report LSI-2000-6. Dept. Lenguages y Sistemas Informáticos, Universidad de Granada, 2000.

[2] Adamson, C.; Venerable, M.: “Data Warehouse Design Solutions”. J. Wiley & Sons, Inc.1998.

[3] Ballard, C.; Herreman, D.; Schau, D.; Bell, R.; Kim, E.; Valncic, A.: “Data Modeling Techniques for Data Warehousing”. SG24-2238-00. IBM Red Book. ISBN number 0738402451. 1998.

[4] Batini, C.; Ceri, S.; Navathe, S.: “Conceptual Database Design- an Entity Relationship Approach”. Benjamin Cummings, 1992.

[5] Cabibbo, L. Torlone, R.:"A Logical Approach to Multidimensional Databases", EDBT'98, Spain, 1998.

[6] Carpani, F. Ruggia, R.: “An Integrity Constraints Language for a Conceptual Multidimensional Data Model”. SEKE’01, Argentina, 2001.

[7] Codd, E.F.; Codd, S.B.; Salley, C.T.: "*Providing OLAP (on-line analytical processing) to user- analysts: An IT mandate*". Technical report, 1993.

[8] Golfarelli, M. Rizzi, S.: ”Methodological Framework for Data Warehouse Design", DOLAP’98, USA, 1998.

[9] Golfarelli, M. Maio, D. Rizzi, S.:”Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases”. DAWAK’00, UK, 2000.

[10] Hahn, K.; Sapia, C.; Blaschka, M.: ”*Automatically Generating OLAP Schemata from Conceptual Graphical Models*", DOLAP’00, USA, 2000.

[11] Kimball, R.:"The Datawarehouse Toolkit". John Wiley & Son, Inc., 1996.

[12] Marotta, A. Ruggia, R.: “Data Warehouse Design: A schema-transformation approach”. SCCC’2002. Chile. 2002.

[13] Moody, D.; Kortnik, M.: “From Enterprise Models to Dimensionals Models: A Methodology for Data Warehouse and Data Mart Design”. DMDW’00, Sweden, 2000.

[14] Peralta, V.: “Diseño lógico de Data Warehouses a partir de Esquemas Conceptuales Multidimensionales”. Master Thesis. Universidad de la República, Uruguay. 2001.