On the Applicability of Rules to Automate Data Warehouse Logical Design

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 (mainly performance) requirements. Existing work in this area has mainly focused into aspects like data models, data structures specifically designed for DWs, and criteria for defining table partitions and indexes. This paper proposes a step forward on 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.

1 Introduction

A Data Warehouse (DW) is a database that stores information devoted to satisfy decision-making requests. DWs have features that distinguish them from transactional databases [9][17]. Firstly, DWs are mainly populated through batch processes [6] that implement data integration, quality improvement, and data structure transformations. In addition, the DW has to support complex queries (grouping, aggregates, crossing of data) instead of short read-write transactions (which characterises OLTP). Furthermore, end users analyse DW information through a multidimensional perspective using OLAP tools [10][16].

Adopting the terminology of [4][1], we distinguish three different design phases based in the concepts managed in each one: c*onceptual level* manages concepts that are close to the way users perceive data, *logical level* deals with concepts related to a certain kind of DBMS (e.g. relational, object oriented,) but are understandable by end users, and *physical level* depends on the specific DBMS and describes how data is actually stored. In this paper we deal with logical design of DWs.

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, various data structures

have been proposed in order to optimise queries [17]. These structures are the well-known star schema, snowflake, and constellations [17][3][19].

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 [12][7][3][14]. We believe that existing work lack in some main aspects related to the generation and management of mappings to source databases, and 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 discusses related work. Section 3 presents a global overview of the proposition and an example. Section 4 describes the framework components, including the design guidelines, the mappings between the DW conceptual schema and the source schema and the rule-based mechanism. Finally, Section 5 points out conclusions and future work.

2 Related Work

There are several proposals concerning to the automation of some tasks of DW relational design [12][7][3][14][19][5].

Some of them mainly focus on conceptual design [12][7][3][14] and deal with the generation of relational schemas following fixed design strategies. In [7], the MD conceptual model is presented (despite authors calling it a logical model) and two algorithms are proposed in order to show that conceptual schemas can be translated to relational or multidimensional logical models. But the work does not focus on a logical design methodology. In [12] a methodological framework, based in the DF conceptual model is proposed. Starting from a conceptual model, they propose to gener-

ate relational or multidimensional schemas. Despite not suggesting any particular model, the star schema is taken as example. The translation process is left to the designer, but interesting strategies and cost models are presented. Other proposals [3][14] also generate fixed star schemas.

As the proposals do not aim at generating complex DW structures, they do not offer flexibility to apply different design strategies. Furthermore, they do not manage complex mappings, between the generated relational schemas and the source ones, though some of them derive the conceptual schema from the source schema.

Other proposals do not take a conceptual schema as input to the logical design task [19][5][17]. In [19], the logical schema is built from an Entity-Relationship (E/R) schema of the source database. Several logical models, as star, snowflake and star-cluster are studied. [17] presents logical design techniques and several examples.

Other works related to automated DW design mainly focus in DW conceptual design and not logical design, like [15][23] which generate the conceptual schema from an E/R schema of the source database.

3 The DW Logical Design Environment

Our underlying design methodology for DW logical design takes as input the source database and a conceptual multidimensional schema ([12][7][15][23][8]). Then, 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 according to the refined conceptual schema and the source database.

The conceptual schema is refined by adding *design guidelines* that cover non-functional requirements. As an example of *guideline*, the designer indicates that he wants to fragment historical data or states the desired degree of normalization. Guidelines constitute a flexible way to express design strategies and properties of the DW.

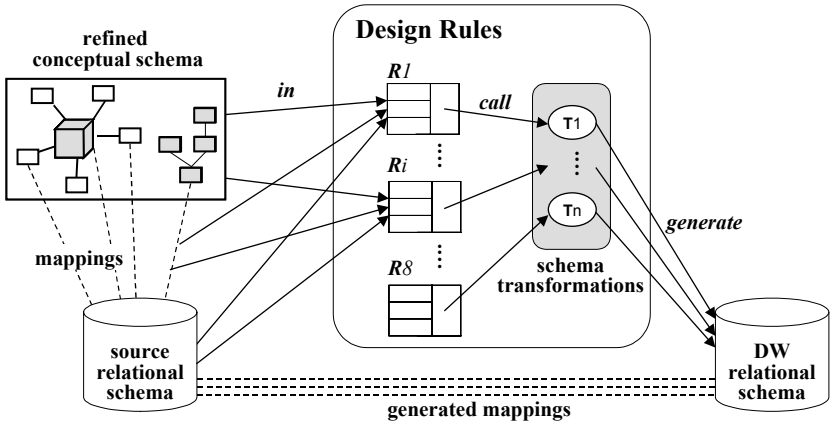

Fig. 1. DW logical design environment. The refined conceptual schema, the source database and the mappings between them (left side) are the input to the automatic rule-based mechanism that builds the DW relational schema (right side) applying schema transformations.

The following task consists of mapping the refined conceptual schema to the source database indicating how to calculate each conceptual structure from the source.

Finally, the DW relational schema should be generated. To do this, we propose a rule-based mechanism in which the DW relational schema is generated by successive application of transformations to the source schema [18]. Each rule determines which transformation must be applied according to the design conditions given by the refined conceptual schema, the source database and the mappings between them, i.e., when certain design condition is fulfilled, the rule applies certain transformation.

The proposed environment provides the infrastructure to carry on the specified process, and consists of: a refined conceptual schema, source and DW relational schemas, a set of rules, schema mappings and a set of schema transformations. Fig. 1 shows the environment.

Due to space constraints, we omit the formal definitions of guidelines and mappings, which can be consulted in [21][22], but sections 3.2 and 3.3 informally present some concepts that will be referenced in the remaining of the paper.

3.1 Motivation Example

Consider a simple example of a company that brings phone support to its customers and wants to analyze the amount of time spent in call attentions.

In the conceptual design phase, the DW analyst has identified two dimensions: *customers* and *dates*. The *customers* dimension has four levels: *state, city, customer* and *department*, organized in two hierarchies. The *dates* dimension has two levels: *year* and *month*. The designer also identified a fact table: *support* that crosses *customers* and *dates* dimensions and takes the call *duration* as measure. Fig. 2 sketches the conceptual schema using CMDM notation [8].

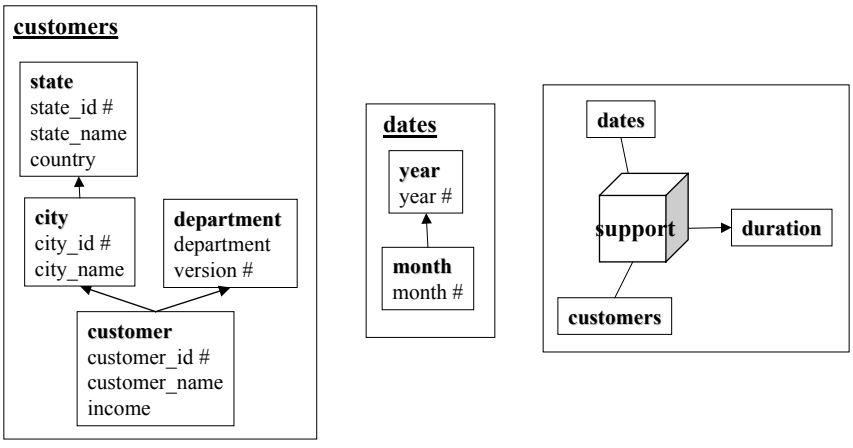

Fig. 2. Conceptual schema. Dimension representation consists of levels in hierarchies, which are stated as boxes with their names in bold followed by the attributes (called items). The items followed by a sharp (#) identify the level. Arrows between levels represent dimension hierarchies. Fact tables are represented through dimension crossings, which are stated as boxes. Measures are distinguished with arrows.

The source database has three master tables: *customers*, *cities* and *states*, a table with the customers' incomes and a table that registers customer calls. The underlined attributes represent primary keys. Foreign keys are also defined (the trivial ones).

- Customers (customer-code, name, address, telephone, city, department, category, registration-date)
- Cities (city-code, state, name, inhabitants)
- States (state-code, name, region)
- Incomes (customer-code, company-code, income)
- Calls (customer-code, date, hour, request-type, operator, duration)

The logical design process starts from the conceptual schema and the source databases. The first task consists in the refinement of the conceptual schema by adding non-functional requirements through *guidelines*.

Guidelines enable to state the desired degrees of normalization [19] or fragmentation [13] in a high level way. Furthermore, *guidelines* enable to define which aggregations (called cubes [8]) are going to be implemented, obtaining derived fact tables. We can also indicate the degree of fragmentation of fact tables [20], in order to manage tables with fewer records and improve query performance, for example, storing a table with data of the last two years and another with historical data. Finally, guidelines can be explicitly defined by the designer or derived using other specific techniques, for example [19][13].

The second design task is to relate the refined conceptual schema with the source database. For example, we can indicate that the items of the *city* level are calculated respectively from the *city-code* and *city-name* attributes of the *Cities* table and the *month* and *year* items of the *dates* dimension are obtained from the *date* attribute of the *Calls* table performing a calculation. This is done through the *schema mappings.*

Finally, the DW schema has to be generated. In order to do this manually, the designer takes a sequence of decisions that transform the resulting DW schema in each step. 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 the *customers* dimension, we have to store attributes corresponding to all the dimension items in the dimension table. This implies joining *Customers, Cities* and *States* tables and projecting the desired attributes. But if we want to normalize the dimension this transformation is not needed. Furthermore, if an item maps to a complex calculated expression several transformations are needed to calculate an attribute for it, for example, for calculating the *income* item as the sum of all customer job's incomes, stored in the *Incomes* table (sum(Incomes.income)).

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 this last task by using a set of rules. The rules check the conditions that hold and determine which transformations are to be applied. The rule conditions consider the conceptual schema, the source database, mappings between them, and additional design guidelines.

3.2 Design Guidelines

Through the guidelines, the designer defines the design style for the DW (snowflakes, stars, or intermediate strategies [19]) and indicates performance and storage requirements. As an example of guideline, we sketch the definition of one:

Example of Guideline: ***Vertical Fragmentation of Dimensions.*** Through this guideline, the designer specifies the level of normalization he wants to obtain in relational structures for each dimension. In particular, he may be interested in a star schema, denormalizing all the dimensions, or conversely he may prefer a snowflake schema, normalizing all the dimensions [19]. This decision can be made globally, regarding all the dimensions, or specifically for each dimension. An intermediate strategy is still possible by indicating the levels to be stored in the same table, allowing more flexibility for the designer.
To sum up, to specify this guideline, the designer must indicate which levels of each dimension he wishes to store together. Each set of levels is called a *fragment*. To do so, he corresponds each dimension with a set of fragments.

3.3 Mappings between the Conceptual Schema and the Source Database

The use of equivalence correspondences or mappings between different schemas is widely used in database design. In DWs design, other types of correspondences are needed to represent calculations and functions on the source database structures. In our proposal, the mappings indicate where the different conceptual schema objects are in the logical schema of the source database. They are functions that associate each conceptual schema item with a mapping expression.

Mapping Expressions. A mapping expression is an expression built using attributes of source tables. It can be: (i) an *attribute* of a source table, (ii) a *calculation* that involve several attributes from a tuple, (iii) an *aggregation* that involves several attributes of several tuples, or (iv) an *external* value like a constant, a time stamp or version digits. As examples of expressions we have: Customers.city, Year (Customers.registration-date), Sum (Incomes.income), and "Uruguay", respectively.

Mapping Functions. Given a set of items (Its), a mapping function associates a mapping expression to each item of the set. We also define the macro Mapping(Its) as the set of all possible mapping functions for a given item set.

$$Mappings(Its) \equiv \{ f / f : Its \rightarrow MapExpressions \}$$

Mapping functions are used in two contexts: in order to map dimension fragments with source tables, and to map cubes with source tables. In both contexts, each fragment or cube item is associated with a mapping expression.

4 The Rule-based Mechanism

Different DW design methodologies [13][17][3][19][12][2] divide the global design problem in smaller design sub-problems 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). At 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 operation. 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. In source metadata, there should be a join condition defined between both tables.
R2	Higher level names	Renames 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 calculation or aggregation.
R4	Materialize external expression *	Adds an attribute to a table to materialize an external expression, such as a constant, a time stamp or version digits. It is used when an item maps to an external expression.
R5	Eliminate not-used data	Deletes not-mapped attributes grouping by the other attributes and applying a roll-up function to measures.
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, so as to agree the key of a fragment or cube with the attribute 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 which are successively applied to the source schema. We use the set of predefined transformations presented in [18], 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. Fig. 3 sketches the context of a rule.

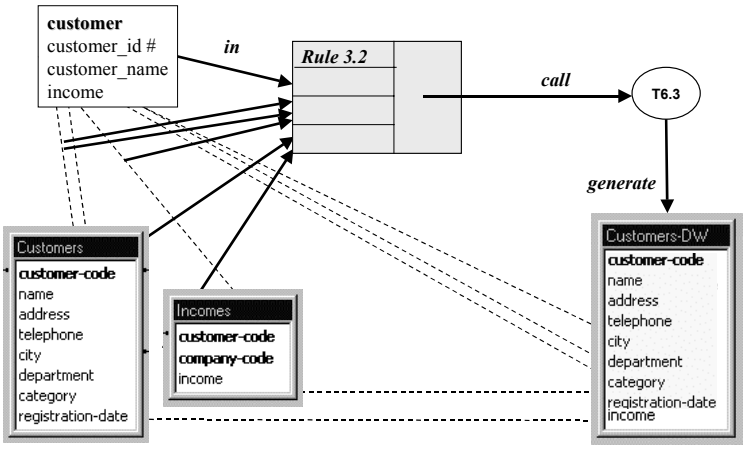

Fig. 3. Context of application of a rule. Refined conceptual schema structures, source tables and mappings are the input of the rule. If conditions are verified, the rule calls a transformation, which generates a new table transforming the input ones. Mappings to the conceptual schema are updated, and a transformation trace from the source schema is kept.

We show the specification of the rule R3.2 (MaterializeAggregate) of the family R3. The goal of this rule is to materialize an aggregation. The conditions to apply the rule are: (i) the mapping function of a cube or fragment corresponds an item to an *aggregation* expression, and (ii) the *attribute* and *calculation* expressions (corresponding to other items) reference only one table. The effect of this last condition is to simplify the application of the corresponding transformation, enforcing to apply join operations before (The previous application of *merge* rule assures that this condition can be satisfied). The rule invokes the transformation T6.3.

For example, consider a fragment (*customer* level) of the *Customers* dimension that maps the *income* item to the aggregation: *SUM(Incomes.income)*, and the other items to attributes of the *Customers* table. Fig 3 shows the application of the rule. It evaluates the conditions and applies a schema transformation (T6.3), which builds a new table: *Customers-DW* with all the attributes of the *Customers* table and a new attribute named *income*. The rule application also updates the mapping function to reference the attributes of the new table and keep the mapping between source tables (*Customers* and *Incomes*) and the generated table.

4.1 Rule Specification.

Rules are specified in 5 sections: *Description, Target Structures, Input State, Conditions* and *Output State*. The *Description* section is a natural language description of the rule behaviour. The *Target Structures* section enumerates the refined conceptual schema structures that are input to the rule. The *Input State* section enumerates the rule input state consisting of tables and 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:

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

Table 2 shows the specification of the rule R3.2 of family R3.

Table 2. Specification of a rule. As Target structures we have a fragment or cube and one of its items. As Input we have the mapping function of the structure and two tables referenced in the mapping function. The first table is referenced by *attribute* and *calculation* expressions, and the second one is referenced by the aggregation expression (The AttributeExpr, CalculationExpr and AggregationExpr functions return all the *attribute*, *calculation* and *aggregation* expressions corresponded by the mapping function). The Conditions have been explained previously. The Output table is the result of applying the transformation T6.3 and the Output mapping function is the result of updating mapping expressions in order to reference the new table.

RULE R3.2 – AGGREGATE CALCULATE

Description:

Given an object from the refined conceptual schema, one of its items, its mapping function (that corresponds the item to an aggregate expression) and two tables (referenced in the mapping function), the rule generates a new table adding the calculated attribute.

Target Structures:

- $S \in$ Fragments $\cup$ Cubes *// a conceptual structure: either fragment or cube*
- $I \in$ Items(S) *// an item of S*

Input State:

- **Mappings:** $F \in$ Mappings(Items(S)) *// the mapping function for S*
- **Tables:** $T_1 \in$ ReferencedTables (AttributeExpr(F) $\cup$ CalculationExpr(F))
  // a table referenced in all attribute and calculation expressions
  $T_2 \in$ ReferencedTables (F(I)) *// a table referenced in the mapping expressions of the item*

Conditions:

- F(I) $\in$ AggregationExpr(F) *// the expression for the item is an aggregation*
- #ReferencedTables (AttributeExpr(F) $\cup$ CalculationExpr(F)) = 1
  // all attribute and calculation expressions reference to one table

Output State:

- **Tables:**
  - T' = Transformation T6.3 ({T_1, T_2}, I = F(I), Link (T_1, T_2))
    // arguments are the source tables, the calculation function for the item and the join function between the tables.
  - UpdateTable (T', T) *// Update table metadata*
- **Mappings:**
  - F' / F'(I) = T'.I $\wedge$ $\forall$ J$\in$Items(S), J$\neq$I **.** (F'(J) = UpdateReference(F,T,T'))
    // The mapping function will correspond the given item with an attribute expression and update the mapping function to reference the new table

4.2 Rule Execution.

The proposed rules do not include a fixed execution order. Therefore, they need an algorithm that orders their application.

We propose an algorithm that is based on existing methodologies and some practical experiences. The algorithm consists of 15 steps. The first 6 steps build dimension tables. Then, steps 7 to 12 build fact tables, steps 13 and 14 perform the aggregates and finally step 15 carries on horizontal fragmentations. Each step is associated to the application of one or more rules.

Due to space limitations we present the global structure of the algorithm and the details of one of its steps (Table 3). The complete specification can be found in [22].

Algorithm Structure:

Phase 1: Build dimension tables for each fragment
- Step 1: Denormalize tables referenced in mapping functions (R1)
- Step 2: Rename attributes referenced in mapping functions (R2)
- Step 3: Materialize calculations (R3, R4)
- Step 4: Data Filter following mapping function conditions (R8)
- Step 5: Delete attributes non-referenced in mapping functions (R5)
- Step 6: Update primary keys of tables referenced in mapping functions (R7)

Phase 2: Build fact tables for each cube
- Step 7: Denormalize tables referenced in mapping functions (R1)
- Step 8: Rename attributes referenced in mapping functions (R2)
- Step 9: Materialize calculations (R3, R4)
- Step 10: Data Filter following mapping function conditions (R8)
- Step 11: Delete attributes non-referenced in mapping functions and apply measure roll-up (R5)
- Step 12: Update primary keys of tables referenced in mapping functions (R7)

Phase 3: Build aggregates
- Step 13: Build an auxiliary table with drill-ups attributes (if it does not exist) (R1, R2, R3, R5, R6)
- Step 14: Build aggregates tables (R5, R6)

Phase 4: Build horizontal fragmentations
- Step 15: Data Filter following fragmentations conditions (R8)

5 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 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 to apply existing design techniques into a unique place allowing for enhanced productivity and simplicity. Moreover, new design techniques can be integrated as new rules or transformations.

This framework has been prototyped [21][11]. 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. All this work has been implemented in a CASE environment for DW design.

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.

Table 3. Specification of a step of the algorithm

STEP 3: MATERIALIZE CALCULATIONS

For each fragment of the refined conceptual schema, apply: rule R3.1 to each calculated expression, rule R3.2 to each aggregate expression, and rule R4 to each extern expression.

For each S ∈ Fragments *// for each fragment S*
  F = FragmentMapping(S) *// F is the mapping function of the fragment*
  {T} = ReferencedTables (AttributeExpr(F) ∪ CalculationExpr(F))
    *// T is the table referenced by attribute and calculation expressions. There is only a referenced table because of previous execution of step 1 */*
  For each I ∈ Items(S) *// for each item of fragment S*
    If F(I) ∈ CalculationExpr(F) *// F corresponds I to a calculation expression*
      Apply Rule3.1: $S, I \left| \frac{<F,T>}{<F',T'>} \right.$
    If F(I) ∈ AggregationExpr(F) *// F corresponds I to an aggregation expression*
      Apply Rule3.2: $S, I \left| \frac{<F,T>}{<F',T'>} \right.$
    If F(I) ∈ ExternalExpr(F) *// F corresponds I to an external expression*
      Apply Rule4: $S, I \left| \frac{<F,T>}{<F',T'>} \right.$
    T = T'
    F = F' *// update loop variables*
  Next
  FragmentMapping(S) = F *// update the mapping function for the fragment*
Next

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 Boehnlein, M.; Ulbrich-vom Ende, A.:”Deriving the Initial Data Warehouse Structures from the Conceptual Data Models of the Underlying Operational Information Systems". DOLAP’99, USA, 1999.

6 Bouzeghoub, M.; Fabret, F.; Matulovic-Broqué, M.: “*Modeling Data Warehouse Refreshment Process as a Workflow Application*”. DMDW’99, Germany, 1999.

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

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

9 Chaudhuri, S.; Dayal, U.: "*An overview of Data Warehousing and OLAP technology*". SIGMOD Record, 26(1), 1997.

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

11 Garbusi, P.; Piedrabuena, F.; Vázquez, G.: “Diseño e Implementación de una Herramienta de ayuda en el Diseño de un Data Warehouse Relacional”. Undergraduate project. Universidad de la República, Uruguay, 2000.

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

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

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

15 Hüsemann, B.; Lechtenbörger, J.; Vossen, G.:"*Conceptual Data Warehouse Design*". DMDW’00, Sweden, 2000.

16 Kenan Technologies:"An Introduction to Multidimensional Databases”. White Paper, Kenan Technologies, 1996.

17 Kimball, R.:"The Datawarehouse Toolkit". John Wiley & Son, Inc., 1996.

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

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

20 Ozsu, M.T. Valduriez, P.: “Principles of Distributed Database Systems”. Prentice-Hall Int. Editors. 1991.

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

22 Peralta, V.: “Towards the Automation of Data Warehouse Logical Design: a Rule-Based Approach”. Technical Report. Universidad de la República, Uruguay. 2003.

23 Phipps, C.; Davis, K.: "Automating data warehouse conceptual schema design and evaluation". DMDW'02, Canada, 2002.