DIMENSION HIERARCHIES UPDATES IN DATA WAREHOUSES
A User-driven Approach

Cécile Favre, Fadila Bentayeb, Omar Boussaid
ERIC Laboratory, University of Lyon, 5 av. Pierre Mendès-France, 69676 Bron Cedex, France
cecile.favre@univ-lyon2.fr, bentayeb@eric.univ-lyon2.fr, omar.boussaid@univ-lyon2.fr

Keywords: Data warehouse model evolution, dimension hierarchies updates, users' knowledge, aggregation rules.

Abstract: We designed a data warehouse for the French bank LCL meeting users' needs regarding marketing operations decision. However, the nature of the work of users implies that their requirements are often changing. In this paper, we propose an original and global approach to achieve a user-driven model evolution that provides answers to personalized analysis needs. We developed a prototype called WEDrik (data Warehouse Evolution Driven by Knowledge) within the Oracle 10g DBMS and applied our approach on banking data of LCL.

1 INTRODUCTION

In the context of a collaboration with the French bank LCL-Le Crédit Lyonnais (LCL), the objective was to develop the decision support system for marketing operations. To achieve this objective, data of interests coming from heterogeneous sources have been extracted, filtered, merged, and stored in an integrating database, called data warehouse. Due to the role of the data warehouse in the daily business work of an enterprise like LCL, the requirements for the design and the implementation are dynamic and subjective. Therefore, data warehouse model design is a continuous process which has to reflect the changing environment, i.e. the data warehouse model must evolve over the time in reaction to the enterprise's changes.

Indeed, data sources are often autonomous and generally have an existence and purpose beyond that of supporting the data warehouse itself. An important consequence of the autonomy of data sources is the fact that those sources may change without being controlled from the data warehouse administrator. Therefore, the data warehouse must be adapted to any change which occurs in the underlying data sources, e.g. changes of the schemas. Beside these changes on the source level, the users often change their requirements. Indeed, the nature of their work implies that their requirements are personal and usually evolve, thus they do not reach a final state. Moreover these requirements can depend on their own knowledge.

Our key idea consists then in considering a specific users' knowledge, which can provide new aggregated data. We represent this users' knowledge under the form of aggregation rules. In a previous work (Favre et al., 2006), we proposed a data warehouse formal model based on these aggregation rules to allow a schema evolution. In this paper, we extend our previous work and propose an original and complete data warehouse model evolution approach driven by emergent users' analysis needs to deal not only with the schema evolution but also with the required evolution for the data.

To perform this model evolution, we define algorithms to update dimension hierarchies by creating new granularity levels and to load the required data exploiting users' knowledge. Hence, our approach provides a real time evolution of the analysis possibilities of the data warehouse to cope with personalized analysis needs. To validate our approach, we developed a prototype named WEDriK (data Warehouse Evolution Driven by Knowledge), within the Oracle 10g Database Management System (DBMS), and applied our approach on data of the French bank LCL.

The remainder of this paper is organized as follows. First, we detail our global evolution approach in Section 2. We present implementation elements in Section 3, detailing the necessary algorithms. In Section 4, we show, through a running example ex-

tracted from the LCL bank case study, how to use our approach for analysis purposes. Then, we discuss the state of the art regarding model evolution in data warehouses in Section 5. We finally conclude and provide future research directions in Section 6.

2 A USER-DRIVEN APPROACH

In this section, we present our user-driven global approach for updating dimension hierarchies to integrate new analysis possibilities into an existing data warehouse (Figure 1). The first phase of our approach is the *acquisition* of the users' knowledge under the form of rules. Then, in the *integration* phase, these rules are transformed into a mapping table within the DBMS. Then the *evolution* phase allows to create a new granularity level. Finally, the *analysis* phase could be carried out on the updated data warehouse model. It is an incremental evolution process, since each time new analysis needs may appear and find an answer. We detail in this section the first three phases.

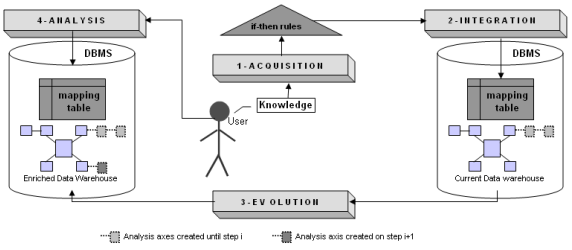

Figure 1: Data warehouse user-driven evolution process.

2.1 Acquisition phase

In our approach, we consider a specific users' knowledge, which determines the new aggregated data to integrate into the underlying data warehouse. More precisely, this knowledge defines how to aggregate data from a given level to another one to be created. It is represented in the form of "if-then" rules.

To achieve the knowledge acquisition phase, our key idea is to define an aggregation meta-rule to represent the structure of the aggregation link. The meta-rule allows the user to define: the granularity level which will be created, the attributes characterizing this new level, the granularity level on which is based the level to be created and finally the attributes of the existing level used to build the aggregation link.

Let EL be the existing level, $\{EA_i, i=1..m\}$ be the set of m existing attributes of EL on which conditions are expressed, GL be the generated level and $\{GA_j, j=1..n\}$ be the set of n generated attributes of GL. The meta-rule is as follows:

if ConditionOn(EL,$\{EA_i\}$) *then* GenerateValue(GL,$\{GA_j\}$)

Thus, the *if*-clause contains conditions on the attributes of the existing level. The *then*-clause contains the definition of the values of the generated attributes, which characterize the new level to be created. The user instanciates the above meta-rule to define the instances of the new aggregated level and the aggregation link with the instances of the existing level.

In the literature, different types of aggregation links can compose the dimension hierarchies to model real situations (Pedersen et al., 2001). In this paper we deal with classical links: an instance in a lower level corresponds to exactly one instance in the higher level, and each instance in the lower level is represented in the higher level. Thus, aggregation rules define a partition of the instances in the lower level, and each class of this partition is associated to one instance of the created level. To check that the aggregation rules satisfy this definition, we deal with the relational representation of aggregation rules which is a mapping table. The transformation is achieved during the integration phase.

2.2 Integration phase

The integration phase consists in transforming "if-then" rules into mapping tables and storing them into the DBMS, by means of relational structures.

For a given set of rules which defines a new granularity level, we associate one mapping table. To achieve this mapping process, we exploit firstly the aggregation meta-rule for building the structure of the mapping table, and secondly we use the aggregation rules to insert the required values in the created mapping table. Since we are in a relational context, each granularity level corresponds to one relational table.

Let ET (resp. GT) be the existing (resp. generated) table, corresponding to EL (resp. GL) in a logical context. Let $\{EA_i, i=1..m\}$ be the set of m attributes of ET on which conditions are expressed and $\{GA_j, j=1..n\}$ be the set of n generated attributes of GT. The mapping table MT_GT built with the meta-rule corresponds to the following relation:

$MT\_GT(EA_1, ..., EA_i, ..., EA_m, GA_1, ..., GA_j, ...GA_n)$

The aggregation rules allow to insert in this mapping table the conditions on the attributes EA_i, and the values of the generated attributes GA_j.

In parallel, we define a mapping meta-table to gather information on the different mapping tables. This meta-table includes the following attributes: Mapping_Table_ID and Mapping_Table_Name correspond respectively to the identifier and the name of the mapping table; Attribute_Name and Table_Name denote the attribute implied in the evolution process and its ta-

ble; Attribute_Type provides the role of this attribute. More precisely, this last attribute has two modalities: 'conditioned' if it appears in the *if*-clause and 'generated' if it is in the *then*-clause. Note that even if an attribute is a "generated" attribute, it can be used as a "conditioned" one for the construction of another level.

2.3 Evolution phase

The data warehouse model evolution consists in updating dimension hierarchies of the current data warehouse model by creating new granularity levels. The created level can be added at the end of a hierarchy or inserted between two existing ones, we thus speak of adding and inserting a level, respectively. In the two cases, the rules have to determine the aggregation link between the lower and the created levels. In the second case, it is also necessary to determine the aggregation link between the inserted level and the existing higher level in an automatic way. The user can insert a level between two existing ones only if it is possible to semantically aggregate data from the inserted level to the existing higher level. For each type of dimension hierarchy updates, we propose, in the following section, an algorithm which creates a new relational table and its necessary link(s) with existing tables.

3 IMPLEMENTATION

To achieve the data warehouse model updating, we developed a prototype named WEDriK (data Warehouse Evolution Driven by Knowledge) within the Oracle 10g DBMS. It exploits different algorithms whose sequence is represented in the Figure 2.

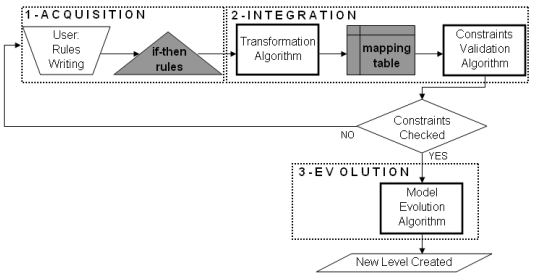

Figure 2: Algorithms sequence.

The starting point of the algorithms sequence is a set of rules expressed by a user to create a granularity level. These rules are transformed under the form of a mapping table (Algorithm 1). Then we check the validity of the rules by testing the content of the mapping table (Algorithm 2). If the rules are not valid, the user has to modify them. This process is repeated until the rules become valid. If they are valid, the model evolution is carried out (Algorithm 3).

Transformation algorithm. For one meta-rule MR and its associated set of aggregation rules R, we build one mapping table MT_GT. The structure of the mapping table is designed with the help of the meta-rule, and the content of the mapping table is defined with the set of aggregation rules. Moreover, the necessary information are inserted in the meta-table *MAPPING_META_TABLE* (Algorithm 1).

Algorithm 1 Pseudo-code for transforming aggregation rules into a mapping table

Require: (1) the meta-rule MR: *if* ConditionOn(EL,$\{EA_i\}$) *then* GenerateValue(GL, $\{GA_j\}$) where EL is the existing level, $\{EA_i, i = 1..m\}$ is the set of m attributes of EL, GL is the level to be created, and $\{GA_j, j = 1..n\}$ is the set of n generated attributes of GL; (2) the set of aggregation rules R; (3) the mapping meta-table *MAPPING_META_TABLE*
Ensure: the mapping table MT_GT
{Creation of the MT_GT's structure}
1: CREATE TABLE MT_GT($\{EA_i\}$,$\{GA_j\}$)
{Values insertion in MT_GT}
2: **for all** rule of R **do**
3: INSERT INTO MT_GT VALUES (ConditionOn(EL,$\{EA_i\}$) ,GenerateValue(GL, $\{GA_j\}$))
4: **end for**
{Values insertion in *MAPPING_META_TABLE*}
5: **for all** EA_i of $\{EA_i\}$ **do**
6: INSERT INTO *MAPPING_META_TABLE* VALUES(MT_GT, EL, EA_i, 'conditioned')
7: **end for**
8: **for all** GA_j of $\{GA_j\}$ **do**
9: INSERT INTO *MAPPING_META_TABLE* VALUES(MT_GT, GL, GA_i, 'generated')
10: **end for**
11: **return** MT_GT

Constraints checking algorithm. For each tuple of MT_GT, we write the corresponding query to build a view which contains a set of instances of the table ET. First, we check that the intersection of all views considered by pairs is empty. Second, we check that the union of the instances of the whole of views corresponds to the whole of instances of the table ET (Algorithm 2).

Model evolution algorithm. The model evolution consists in the creation of the table GT and the definition of the required links, according to the evolution type (Algorithm 3). For an addition or an insertion, it thus consists in creating the table GT and inserting values by exploiting the mapping table MT_GT, and

Algorithm 2 Pseudo-code for checking constraints

Require: existing table ET, mapping table MT_GT
Ensure: Intersection_constraint_checked and Union_constraint_checked two booleans
{Views creating}
1: **for all** tuple t of MT_GT **do**
2: CREATE VIEW v_t AS SELECT * FROM ET where Conjunction($t(\{EA\})$)
3: **end for**
{Checking the intersection of views considered by pair is empty}
4: **for** x = 1 to t_{max} **do**
5: I=SELECT * FROM v_x INTERSECT SELECT * FROM v_x+1
6: **if** $I \neq \emptyset$ **then**
7: Intersection_constraint_checked=false
8: **end if**
9: **end for**
{Checking the union of views corresponds to the table ET}
10: U=SELECT * FROM v_1
11: **for** x = 2 to t_{max} **do**
12: U=U UNION SELECT * FROM v_x
13: **end for**
14: **if** U $\neq$ SELECT * FROM ET **then**
15: Union_constraint_checked=false
16: **end if**
17: **return** Intersection_constraint_checked
18: **return** Union_constraint_checked

then in linking the new table GT with the existing one ET. For the insertion, we just use the mapping table MT_GT, which only contains the link between the lower table ET and the generated table GT. Then we have to automatically establish the aggregation link between GT and $ET2$, by inferring it according to the one that exists between ET and $ET2$.

Algorithm 3 Pseudo-code for hierarchy dimension updating

Require: *evol_type* the type of evolution (inserting or adding a level) , existing lower table ET, existing higher table $ET2$, mapping table MT_GT, $\{EA_i$, i=1..$m\}$ the set of m attributes of MT_GT which contain the conditions on attributes , $\{GA_j$, j=1..$n\}$ the set of n generated attributes of MT_GT which contain the values of the generated attributes, the key attribute GA_{key} of GT, the attribute B linking ET with $ET2$
Ensure: generated table GT
{Creating the new table GT and making the aggregation link between ET and GT}
1: CREATE TABLE GT (GA_{key}, $\{GA_j\}$);
2: ALTER TABLE ET ADD (GA_{key});
3: **for all** (tuple t of MT_GT) **do**
4: INSERT INTO GT VALUES (GA_{key},$\{GA_j\}$)
5: UPDATE ET SET GA_{key} WHERE $\{EA_i\}$
6: **end for**
{If it is a level insertion: linking GT with $ET2$}
7: **if** evol_type="inserting" **then**
8: ALTER TABLE GT ADD (B);
9: UPDATE GT SET B=(SELECT DISTINCT B FROM ET WHERE $ET.GA_{key}$=$GT.GA_{key}$);
10: **end if**
11: **return** GT

4 A RUNNING EXAMPLE

To illustrate our approach, we use a case study defined by the French bank LCL. LCL is a large company, where the data warehouse users have different points of view. Thus, they need specific analyses, which depend on their own knowledge and their own objectives. We applied our approach on data about the annual Net Banking Income (NBI). The NBI is the profit obtained from the management of customers account. It is a measure observed according to several dimensions: CUSTOMER, AGENCY and YEAR (Figure 3).

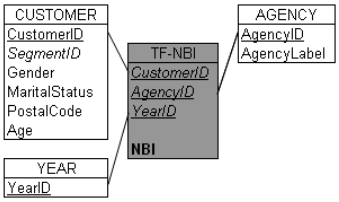

Figure 3: Data warehouse model for the NBI analysis.

To illustrate the usage of WEDriK, let us focus on a simplified running example. Let us take the case of the person in charge of student products. He knows that there is three types of agencies: "student" for agencies which gather only student accounts, "foreigner" for agencies whose customers do not leave in France, and "classical" for agencies without any particularity. He needs to obtain NBI analysis according to the agency type. The existing data warehouse could not provide such an analysis. What we seek to do is showing step by step how to achieve this analysis with our approach.

Let us consider the samples of the tables AGENCY and TF-NBI (Figure 4).

AGENCY		
AgencyID	*AgencyLabel*	...
01000	LYON REPUBLIQUE	...
01029	LYON GERLAND	...
01903	LYON III UNIVERSITE	...
01905	LYON LA DOUA	...
01929	AGENCE INTERNATIONALE	...
02256	CLERMONT LAFAYETTE (Etud)	...
02600	GRENOBLE	...
03730	ANNONAY	...

TF-NBI			
AgencyID	*CustomerID*	*YearID*	*NBI (€)*
01000	1	2006	2000
01903	2	2006	1000
01000	3	2006	4000
03730	4	2006	2000
01929	5	2006	5000
01000	6	2006	2000
01029	7	2006	1000
02600	8	2006	1000
01905	9	2006	2000
01929	10	2006	3000

Figure 4: Samples of tables.

1-Acquisition phase. The *acquisition phase* allows the user to define the following aggregation meta-rule in order to specify the structure of the aggregation link for the agency type:
if ConditionOn(AGENCY,{AgencyID})
then GenerateValue(AGENCY_TYPE,{AgencyTypeLabel})
He instances the meta-rule to define the different agency types:
(R1) *if* AgencyID$\in$ {'01903','01905','02256'} *then* AgencyTypeLabel='student'

(R2) *if* AgencyID='01929'*then* AgencyTypeLabel='foreigner'
(R3) *if* AgencyID∈{'01903','01905','02256','01929'}*then* AgencyTypeLabel='classical'

2-Integration phase. The *integration phase* exploits the meta-rule and the different aggregation rules to generate the mapping table MT_AGENCY_TYPE (Figure 5). The information concerning the mapping table is inserted in MAPPING_META_TABLE (Figure 6).

MT_AGENCY_TYPE	
AgencyID	*AgencyTypeLabel*
IN ('01903', '01905', '02256')	student
= '01929'	foreigner
NOT IN ('01903', '01905', '02256', '01929')	classical

Figure 5: Mapping table for the AGENCY_TYPE level.

MAPPING_META_TABLE			
Mapping_Table_Name	*Attribute_Name*	*Attribute_Table*	*Attribute_Type*
MT_AGENCY_TYPE	AgencyID	AGENCY	conditioned
MT_AGENCY_TYPE	AgencyTypeLabel	AGENCY_TYPE	generated

Figure 6: Mapping meta-table.

3-Evolution phase. Then the *evolution phase* allows to create the AGENCY_TYPE table and to update the AGENCY table (Figure 7). More precisely, the MAPPING_META_TABLE allows to create the structure of the AGENCY_TYPE table and a primary key is automatically added. The MT_AGENCY_TYPE mapping table allows to insert the required values. Furthermore, the AGENCY table is updated to be linked with the AGENCY_TYPE table, with the addition of the AgencyTypeID attribute.

AGENCY_TYPE	
AgencyTypeID	*AgencyTypeLabel*
1	student
2	foreigner
3	classical

AGENCY		
AgencyID	*...*	*AgencyTypeID*
01000	...	3
01029	...	3
01903	...	1
01905	...	1
01929	...	2
02256	...	1
02600	...	3
03730	...	3

Figure 7: Created AGENCY_TYPE table and updated AGENCY table.

4-Analysis phase. Finally, the *analysis phase* allows to exploit the model with new analysis axes. Usually, in an OLAP environment, queries require the computation of aggregates over various dimension levels. Indeed, given a data warehouse model, the analysis process allows to summarize data by using (1) aggregation operators such as SUM and (2) GROUP BY clauses. In our case, the user wants to know the sum of NBI according to the agency types he has defined. The corresponding query and the results are provided in Figure 8.

```
SELECT AgencyTypeLabel, SUM(NBI)
FROM TF-NBI, AGENCY, AGENCY_TYPE
WHERE TF-NBI.AgencyID=AGENCY.AgencyID
AND AGENCY.AgencyTypeID=AGENCY_TYPE.AgencyTypeID
GROUP BY AgencyTypeLabel ;
```

NBI ANALYSIS	
AgencyTypeLabel	*Sum(NBI)*
student	3 000 €
foreigner	8 000 €
classical	12 000 €

Figure 8: Query and results for NBI analysis.

5 RELATED WORK

When designing a data warehouse model, involving users is crucial (Kimball, 1996). What about the data warehouse model evolution ? According to the literature, model evolution in data warehouses can take the form of model updating or temporal modeling.

Concerning model updating, only one model is supported and the trace of the evolutions is not preserved. In this case, the data historization can be corrupted and analysis could be erroneous. We distinguish three types of approach. The first approach consists in providing evolution operators that allow an evolution of the model (Hurtado et al., 1999; Blaschka et al., 1999). The second approach consists in updating the model, focusing on enriching dimension hierarchies. For instance, some authors propose to enrich dimension hierarchies with new granularity levels by exploiting semantic relations provided by WordNet[1] (Mazón and Trujillo, 2006). The third approach is based on the hypothesis that a data warehouse is a set of materialized views. Then, when a change occurs in a data source, it is necessary to maintain views by propagating this change (Bellahsene, 2002).

Contrary to model updating, temporal modeling makes it possible to keep track of the evolutions, by using temporal validity labels. These labels are associated with either dimension instances (Bliujute et al., 1998), or aggregation links (Mendelzon and Vaisman, 2000), or versions (Eder and Koncilia, 2001; Body et al., 2002). Versioning is the subject today of many work. Versions are also used to provide an answer to "what-if analysis" by creating versions to simulate a situation (Bébel et al., 2004). Different work are then interested in analyzing data throughout versions, in order to achieve the first objective of data warehousing: analyzing data in the course of time (Morzy and Wrembel, 2004; Golfarelli et al., 2006). In these works, an extension to a traditional SQL language is required to take into account the particularities of the approaches for analysis or data loading.

Both of these approaches do not directly involve users in the data warehouse evolution process, and

[1] http://wordnet.princeton.edu/

thus constitute solutions rather for a data sources evolution than for users' analysis needs evolution. Thus these solutions make the data warehouse model evolve, without taking into account new analysis needs driven by users' knowledge, and thus without providing an answer to personalized analysis needs. However, the personalization of the use of a data warehouse becomes crucial. Works in this domain are particularly focused on the selection of data to be visualized, based on users' preferences (Bellatreche et al., 2005). In opposition of this approach, we add supplement data to extend and personalize the analysis possibilities of the data warehouse. Moreover our approach updates the data warehouse model without inducing erroneous analyses; and it does not require extensions for analysis.

6 CONCLUSION AND FUTURE WORK

We aimed in this paper at providing an original user-driven approach for the data warehouse model evolution. Our key idea is to involve users in the dimension hierarchies updating to provide an answer to their personalized analysis needs based on their own knowledge. To achieve this process, we propose a method to acquire users' knowledge and the required algorithms. We developed a prototype named WEDriK within the Oracle 10g DBMS and we applied our approach on the LCL case study.

To the best of our knowledge, the idea of user-defined evolution of dimensions (based on analysis needs) is novel; there are still many aspects to be explored. First of all, we intend to study the performance of our approach in terms of storage space, response time and algorithms complexity. Moreover, we have to study how individual users' needs evolve in time, and thus we have to consider different strategies of rules and mapping tables updating. Finally, we are also interested in investigating the joint evolution of the data sources and the analysis needs.

REFERENCES

Bébel, B., Eder, J., Koncilia, C., Morzy, T., and Wrembel, R. (2004). Creation and Management of Versions in Multiversion Data Warehouse. In *XIXth ACM Symposium on Applied Computing (SAC 04), Nicosia, Cyprus*, pages 717–723.

Bellahsene, Z. (2002). Schema Evolution in Data Warehouses. *Knowledge and Information Systems*, 4(3):283–304.

Bellatreche, L., Giacometti, A., Marcel, P., Mouloudi, H., and Laurent, D. (2005). A Personalization Framework for OLAP Queries. In *VIIIth ACM International Workshop on Data Warehousing and OLAP (DOLAP 05), Bremen, Germany*, pages 9–18.

Blaschka, M., Sapia, C., and Höfling, G. (1999). On Schema Evolution in Multidimensional Databases. In *Ist International Conference on Data Warehousing and Knowledge Discovery (DaWaK 99), Florence, Italy*.

Bliujute, R., Saltenis, S., Slivinskas, G., and Jensen, C. (1998). Systematic Change Management in Dimensional Data Warehousing. In *IIIrd International Baltic Workshop on Databases and Information Systems, Riga, Latvia*, pages 27–41.

Body, M., Miquel, M., Bédard, Y., and Tchounikine, A. (2002). A Multidimensional and Multiversion Structure for OLAP Applications. In *Vth ACM International Workshop on Data Warehousing and OLAP (DOLAP 02), McLean, Virginia, USA*, pages 1–6.

Eder, J. and Koncilia, C. (2001). Changes of Dimension Data in Temporal Data Warehouses. In *IIIrd International Conference on Data Warehousing and Knowledge Discovery (DaWaK 01)*, pages 284–293.

Favre, C., Bentayeb, F., and Boussaid, O. (2006). A Knowledge-driven Data Warehouse Model for Analysis Evolution. In *XIIIth International Conference on Concurrent Engineering: Research and Applications (CE 06), Antibes, France*.

Golfarelli, M., Lechtenborger, J., Rizzi, S., and Vossen, G. (2006). Schema Versioning in Data Warehouses: Enabling Cross-Version Querying via Schema Augmentation. *Data and Knowledge Engineering*, 59(2):435–459.

Hurtado, C. A., Mendelzon, A. O., and Vaisman, A. A. (1999). Maintaining Data Cubes under Dimension Updates. In *XVth International Conference on Data Engineering (ICDE 99), Sydney, Australia*, pages 346–355.

Kimball, R. (1996). *The Data Warehouse Toolkit*. John Wiley & Sons.

Mazón, J.-N. and Trujillo, J. (2006). Enriching Data Warehouse Dimension Hierarchies by Using Semantic Relations. In *XXIIIrd British National Conference on Databases (BNCOD 2006), Belfast, Northern Ireland*, volume 4042 of *LNCS*.

Mendelzon, A. O. and Vaisman, A. A. (2000). Temporal Queries in OLAP. In *XXVIth International Conference on Very Large Data Bases (VLDB 00), Cairo, Egypt*, pages 242–253.

Morzy, T. and Wrembel, R. (2004). On Querying Versions of Multiversion Data Warehouse. In *VIIth ACM International Workshop on Data Warehousing and OLAP (DOLAP 04), Washington, Columbia, USA*, pages 92–101.

Pedersen, T. B., Jensen, C. S., and Dyreson, C. E. (2001). A Foundation for Capturing and Querying Complex Multidimensional Data. *Information Systems*, 26(5):383–423.