Chapter II

Dynamic Workload for Schema Evolution in Data Warehouses: a Performance Issue

Fadila Bentayeb, University of Lyon – ERIC Laboratory - France
Cécile Favre, University of Lyon – ERIC Laboratory - France
Omar Boussaid, University of Lyon – ERIC Laboratory - France

Abstract

A data warehouse allows the integration of heterogeneous data sources for identified analysis purposes. The data warehouse schema is designed according to the available data sources and the users' analysis requirements. In order to provide an answer to new individual analysis needs, we previously proposed, in recent work, a solution for on-line analysis personalization. We based our solution on a user-driven approach for data warehouse schema evolution which consists in creating new hierarchy levels in OLAP (On-Line Analytical Processing) dimensions.

One of the main objectives of OLAP, as the meaning of the acronym refers, is the performance during the analysis process. Since data warehouses contain a large volume of data, answering decision queries efficiently requires particular access methods. The main issue is to use redundant optimization structures such as views and indices. This implies to select an appropriate set of materialized views and indices, which minimizes total query response time, given a limited storage space. A judicious choice in this selection must be cost-driven and based on a workload which represents a set of users’ queries on the data warehouse.

In this chapter, we address the issues related to the workload’s evolution and maintenance in data warehouse systems in response to new requirements modeling resulting from users’ personalized analysis needs. The main issue is to avoid the workload generation from scratch. Hence, we propose a workload management system which helps the administrator to maintain and adapt dynamically the workload according to changes arising on the data warehouse schema. To achieve this maintenance, we propose two types of workload updates: (1) maintaining existing queries consistent with respect to the new data warehouse schema and (2) creating new queries based on the new dimension hierarchy levels. Our system helps the administrator in adopting a pro-active behaviour in the management of the data warehouse performance. In order to validate our workload management system, we address the implementation issues of our proposed prototype. This latter has been developed within client/server architecture with a web client interfaced with the Oracle 10g DataBase Management System.

Keywords: Data warehouse, schema evolution, optimization strategy, dynamic workload, personalization, dimension hierarchy updates, decision-support queries, OLAP.
Introduction

Research in data warehousing and OLAP (On-Line Analytical Processing) has produced important technologies for the design, management and use of information systems for decision support. Nevertheless, despite the maturity of these technologies, there are new data and analysis needs in companies. These needs not only demand more storage capacity, but also new methods, models, techniques or architectures. Some of the hot topics in data warehouses include schema evolution, versioning and OLAP operators for dimension updates.

To be an effective support of OLAP analysis, data warehouses store subject-oriented, integrated, time-variant and non-volatile collection of data coming from heterogeneous data sources in response to users' analysis needs. In a data warehouse, data are organized in a multidimensional way. The objective is then to analyse facts through measures, according to dimensions which can be divided into hierarchies, representing different granularity levels of information. The granularity levels of each dimension are fixed during the design step of the data warehouse system. After deployment, these dimensions remain static because schema evolution is poorly supported in current OLAP models.

To design a data warehouse, several approaches exist in the literature. Inmon, W.H. (2002) argues that the data warehouse environment is data-driven, in comparison to classical systems which are requirements-driven. Anahory, S., & Murray, D. (1997) propose a catalogue for conducting users interviews in order to collect end-user requirements while Kimball, R. (1996) and Kimball, R. et al. (1998) state that the main step of the design process in data warehousing is based on a business process to model. User requirements describe the tasks that the users must be able to accomplish with the help of the data warehouse system. They are often collected in the design step of the data warehouse. Thus some new requirements are not satisfied and some trends are not explored. Indeed, data sources and requirements are often changing so that it is very important that the data warehouse schema evolves according to these changes. In business environment, several changes in the content and structure of the underlying data sources may occur and individual analysts' needs can emerge and grow in time. Thus, a data warehouse schema cannot be designed in one step since it evolves over the time.

To consider this problem, two categories of research emerged: (1) temporal multidimensional data models that manage and keep the evolutions history by time-stamping relations over hierarchy levels proposed by Morzy, T., & Wrembel, R. (2004), Morzy, T., & Wrembel, R. (2003), Mendelzon, A.O., & Vaisman, A.A. (2000) and Bliujute, R. et al. (1998); and (2) extending the multidimensional algebra with a set of schema evolution operators proposed by Pourabbas, E., & Rafanelli, M. (2000), Hurtado, C.A. et al. (1999) and Blaschka, M. et al. (1999).

In the context of data warehouse evolution, we proposed in a previous work a new data warehouse architecture which takes into account personalized user's analyses independently of the data sources. More precisely, in Bentayeb, F. et al. (2008) and Favre, C. et al. (2007a) we designed a user-driven data warehouse model in which schema evolves according to new user's analysis needs. In this case, new dimension hierarchies are then created based on personalized user's analyses in an interactive way. The user becomes then one of the key points of the data warehouse incremental evolution process. To validate our model, we followed a relational approach and implemented our data warehouse model inside Oracle DataBase Management System (DBMS). Furthermore, we also applied our approach on banking data of the French bank Le Crédit Lyonnais1 (LCL).

In the other hand, when designing a data warehouse, choosing its architecture is crucial. We can find in the literature three classical main types of data warehouse models, namely star, snowflake, and constellation schemas presented by Inmon, W.H. (2002) and Kimball, R., & Ross, M. (1996) and implemented in several environments such as ROLAP (Relational OLAP), MOLAP (Multidimensional OLAP) or HOLAP (Hybrid OLAP). Moreover, in the case of data warehouse schema evolution, other data modelling possibilities exist such as data warehouse versions that are also based on the above classical data warehouse models.

Hence, the choice of data warehouse architecture is not neutral: it always has advantages and drawbacks and greatly influences the response time of decision-support queries. Once the architecture is selected, various

---

1Collaboration with the Management of Rhône-Alpes Auvergne Exploitation of LCL-Le Crédit Lyonnais within the framework of an Industrial Convention of Formation by Research (CIFRE)

Copyright © 2008, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
optimization techniques such as indexing or materializing views further influence querying and refreshing performance. This is especially critical when performing tasks, such as computing data cubes or performing data mining.

Generally, to evaluate the efficiency of performance optimization techniques, (such as index and materialized view techniques), one should design a data warehouse benchmark. This latter may be defined as a database model and a workload model (set of queries to execute on the database). Different goals can be achieved by using a benchmark: (1) compare the performance of various systems in a given set of experimental conditions (users); (2) evaluate the impact of architectural choices or optimization techniques on the performance of one given system (system designers). In a data warehouse benchmark, the workload may be subdivided into: (1) decision-support queries, mostly OLAP queries such as cube, roll-up, drill down and slice and dice; (2) queries used during the ETL (Extract Transform Load) process.

In the context of the data warehouse evolution approach, the workload of the benchmark should follow the evolution of the data warehouse schema. In Favre, C., et al. (2007b), we proposed a comprehensive automatic approach to reflect the data warehouse schema evolutions on the workload. Then, in this chapter, we propose a workload management system for controlling the execution of queries used to select optimization strategies, based on data warehouse schema evolution model. Our objective is to define a dynamic workload. Thus, we design a workload as a generic query model defined by a grammar, which introduces much-needed analytical capabilities to relational database querying. This increases the ability to perform dynamic, analytic SQL queries. So, given an initial workload, our objective is to update the workload with respect to the new data warehouse schema in an incremental way. The workload update consists then in: (1) creating new decision-support queries based on new hierarchy levels in dimensions, representing users' personalized analyses; (2) maintaining the existing queries consistent with regard to the new data warehouse schema. It allows the administrator to adopt a pro-active behaviour in the data warehouse performance management.

The remainder of this chapter is organized as follows. The following section is devoted to the related work on data warehouse evolution, including schema evolution, performance evaluation and query evolution. Then, we present our general framework for schema evolution in data warehouses. After that, we detail our approach to help the data warehouse administrator to update an existing workload, as a support for optimization dynamic strategy. Next, we present an example of our approach applied to a real simplified case study of LCL French bank, before providing implementation issues. We finally conclude and provide future research directions.

## Related Work

In this section, we present researches that are most closely related to our work, namely: (1) data warehouses' model evolution, (2) performance evaluation and (3) query evolution.

### Data warehouses' model evolution

We can distinguish in the literature two types of approaches regarding model evolution in data warehouses: model updating and temporal modelling.

The first approach consists in transforming the data warehouse schema and proposes to enrich data warehouses with adapted evolution operators that allow schema updates, such as proposed by Blaschka, M. (1999), Blaschka, M. et al. (1999), Hurtado, C.A. et al. (1999a) and Hurtado, C.A. et al. (1999b). In Hurtado, C.A. et al. (1999a) and Hurtado, C.A. et al. (1999b) dimensions are modeled as an acyclic graph, where nodes represent dimension attributes and arcs represent hierarchical links between these attributes. Evolution operators are then defined under the form of algebraic functions, which are able to modify the graph structure. Blaschka, M. (1999) and Blaschka, M. et al. (1999) propose elementary operators for schema evolution, which can be combined to define more complex operators. We note that these two methods support only one schema and do not keep the history of evolutions.
On the contrary, the second approach keeps track of the schema evolution, by using temporal validity labels. These labels are affixed either on dimension instances such as proposed by Bliujute, R. et al. (1998), or on aggregation links such as proposed by Mendelzon, A.O., & Vaisman, A.A. (2000), or on schema versions such as proposed by Morzy, T., & Wrembel, R. (2004), Bebel, B. et al. (2004) and Body, M. et al. (2002). Let us detail these different methods. Mendelzon, A.O., & Vaisman, A.A. (2000) propose a temporal multidimensional model and a query language supporting it (namely TOLAP). Dimension elements are timestamped at the schema or instance level (or both) in order to keep track of the updates that occur. The regular star schema treats all dimensions, one of them being the Time dimension, equally and assumes them to be independent. Bliujute, R. et al. (1998) propose a temporal star schema, where time is not a separate, independent dimension table, but is a dimension of all tables and is represented via one or more time-valued attributes in all tables. Another promising approach to handling changes in data warehouse structure and content is based on a multiversion data warehouse such as proposed by Morzy, T., & Wrembel, R. (2004), Bebel, B. et al. (2004) and Body, M. et al. (2002). In such a data warehouse, each version describes a schema and data at a certain period of time. In order to appropriately analyse multiversion data, an extension to traditional SQL language is required.

Both of these approaches do not directly involve users in the data warehouse evolution process, and thus constitute a solution rather for data sources evolution than for users' analysis needs evolution. Indeed, once the data warehouse is created, users can only carry out analyses provided by the model. Thus these solutions make the data warehouse schema evolve, ignoring new analysis needs driven by users' knowledge. A personalization process is then required in order to provide answers to users' emergent individual needs. Furthermore, the personalization of the data warehouse becomes crucial for expanding its use to the most of users. It is a research perspective, which emerges in the data warehouse community. Works in this domain are particularly focused on the visualization of data, based on users' preferences modelling and exploitation with the profile concept such as proposed by Bellatrèche, L. et al. (2005). It consists in refining queries to show a part of data, which meets user's preferences.

To achieve analyses personalization, whatever it is, the data warehouse should be more flexible. Works aimed at bringing flexibility within the data warehouse use mostly rule-based languages. Firstly, by defining the data warehouse schema in a flexible way, two approaches are possible: using rules either to express the analysis needs such as proposed by Kim, H.J. et al. (2003) or the knowledge about data warehouse construction such as proposed by Peralta, V. et al. (2003). Secondly, the data warehouse administrator could use rules to define some integrity constraints in order to ensure the consistency of the data and the analyses as a consequence. For Carpani, F., & Ruggia, R. (2001) and Ghozzi, F. et al. (2003), the expressed integrity constraints use the data semantic to manage data inconsistency within the analysis. Thirdly, in order to make the analysis more flexible, a rule-based language has been developed by Espil, M.M., & Vaisman, A.A. (2001) to manage exceptions during the aggregation process. This language allows to intentionally expressing redefinitions of aggregation hierarchies. Thus, rule-based languages allow flexibility within data warehouses in different works. We want to introduce such flexibility in the analysis evolution process. However, the flexibility of the analysis evolution depends on the flexibility of the schema evolution, in particular dimension hierarchies updates, as we present thereafter.

### Performance evaluation

One of the most important issues in data warehouse physical design is to select an appropriate set of materialized views and indices, which minimizes total query response time, given a limited storage space. A judicious choice in this selection must be cost-driven and influenced by the workload experienced by the system. Indeed, it is crucial to adapt the performance of the system according to its use as Gallo, J. (2002) says. In this perspective, the workload should correspond to a set of users' queries. The most recent approaches syntactically analyse the workload to enumerate relevant candidates (indices or views) such as proposed by Agrawal, S. et al. (2000). The selection is based on cost models that evaluate the cost of accessing data using optimization structures (views, indices) and the cost of storing and maintaining these structures.
The workload is supposed to represent the users’ demand on the data warehouse. In the literature, most of the proposed approaches rely on the existence of a reference workload that represents the target for the optimization such as Theodoratos, D., & Sellis, T. (2000). However, Golfarelli, M., & Saltarelli, E. (2003) argue that real workloads are much larger than those that can be handled by these techniques and thus view materialization and indexing success still depends on the experience of the designer. Thus, they propose an approach to build a clustered workload that is representative of the original one and that can be handled by views and indices selection algorithms.

**Query evolution**

**Workload evolution.** In views selection area, various works concern their dynamical selection such as those proposed by Kotidis, Y., & Roussopoulos, N. (1999), Theodoratos, D., & Sellis, T. (2000) and Lawrence, M., & Rau-Chaplin, A. (2006). The dynamic aspect is an answer to workload evolution. However, workload evolution can also be considered from different points of views. Lawrence, M., & Rau-Chaplin, A. (2006) consider that the view selection must be performed at regular maintenance intervals and is based on an observed or expected change in query probabilities. Kotidis, Y., & Roussopoulos, N. (1999), Theodoratos, D., & Sellis, T. (2000) suppose that some queries are added to the initial workload. Thus these works assume that previous queries are always correct. However, we affirm that it is not always the case. That is why we interested in the problem of query evolution.

**The query evolution problem.** The problem of query evolution in data warehouses has been addressed in an indirect way. We evoked briefly the problem of materialized view maintenance. However, we have to consider the duality of views, which are both sets of tuples according to their definition of extension and queries according to their definition of intention. Indeed, a view corresponds to the result of a query. Thus the question of view evolution can be treated as the query evolution problem. This perspective is followed by Bellahsène, Z. (2002), where the impact of data sources schema changes is examined to each of the clauses of the view query (structural view maintenance).

However, in data warehousing domain, the issue of query evolution has not yet been addressed as a problem in itself. This point is important because queries are not only used to define views; for instance, in a decisional architecture, queries are also used in reporting (predefined queries) or to test the performance of the system when they form a workload.

The problem of query maintenance has been evoked in the database field. Some authors think that model management is important for the entire environment of a database. Indeed, queries maintenance should also involve the surrounding applications and not only be restricted to the internals of a database such as explained by Vassiliadis, P. et al. (2007). Traditional database modelling techniques do not consider that a database supports a large variety of applications and provides tools such as reports, forms. A small change like the deletion of an attribute in this database might impact the full range of applications around the system: queries and data entry forms can be invalidated; application programs accessing this attribute might crash.

Thus, Papastefanatos, G. et al. (2005) first introduce and sketch a graph-based model that captures relations, views, constraints and queries. Then, Papastefanatos, G. et al. (2006) extend their previous work by formulating a set of rules that allow the identification of the impact of changes to database relations, attributes and constraints. They propose a semi-automated way to respond to these changes. The impact of the changes involves the software built around the database, mainly queries, stored procedures, triggers. This impact corresponds to an annotation on the graph, requiring a tedious work for the administrator.

In this chapter, we focus on the query evolution aspect in data warehousing context. More particularly, we focus on the performance optimization objective through the evolution of the workload. As compared to previous presented work, we do not consider annotations for each considered model. We propose a general approach to make any workload evolve. These evolutions are made possible by using the specificity of multidimensional model that encapsulates semantic concepts such as dimension, dimension hierarchy. Indeed, these concepts induce roles that are recognizable in any model.
General Framework for Schema Evolution in Data Warehouses

The issue we consider in this section is how to provide features to support schema evolution of data warehouses. In earlier work, namely in Bentayeb, F. et al. (2008), we investigated the use of data warehouse schema evolution to provide an answer to changes and extensions of business requirements. More precisely, our approach supports schema changes of data warehouses independently of the data sources. These changes should not be propagated to the data sources since they are autonomous. Indeed, they represent new users’ needs in terms of OLAP analyses. Thus, we presented an original approach for users’ personalized analyses in data warehouses, following a schema updating approach. In this section, we present the main aspects of our approach that constitutes our framework for schema evolution. Firstly, we present the framework of the personalization process. Then, we detail the metamodel used to achieve the schema evolution management.

Illustrative example

To illustrate our approach, we use a simplified example of the LCL French bank, and more particularly an extract of the data warehouse concerning 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 three dimensions: CUSTOMER, AGENCY and YEAR (Figure 1).

Figure 1: Data warehouse schema for the NBI analysis

The presented schema is an answer to the global analysis needs on NBI. However, LCL is a large company, where the data warehouse users work in various departments, and thus have different points of view. Then, they need specific analyses, which depend on their own knowledge and their own objectives.

Let us take now the case of the man in charge of student products. He knows that there are 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. Thus he knows the various types of agencies and the corresponding agency identifiers. However, this knowledge is not included in the existing data warehouse. It therefore cannot be used to carry out an analysis about student dedicated agencies.

Our idea consists then in considering this specific users' knowledge, which can provide new aggregated data, to generate a new analysis axis by dynamically creating a new granularity level: the agency type.

Data warehouse schema updates for analysis personalization

To achieve analyses personalization, we propose a global approach that allows the integration of new analysis needs into the existing data warehouse. This personalization corresponds to a user-driven data warehouse schema evolution (Figure 2).

Our approach can be represented as a global architecture. The first step is the acquisition of the users’ knowledge under the form of “if-then” rules. Then, in the second step, these rules are integrated within the DBMS, which implements the current data warehouse. This integration consists in transforming “if-then” rules into mapping tables. Then the evolution step is achieved by creating a new granularity level inside an existing dimension hierarchy or defining a new one. Finally, the analysis is carried out on the updated data.
It is an incremental evolution process, since each time new analysis needs may occur. In the following, we detail the first three steps of our architecture.

**Figure 2: Data warehouse user-driven evolution process**

Users’ knowledge acquisition. In our approach, we consider a specific users' knowledge, which determines the new aggregated data to integrate into the underlying data warehouse. More precisely, our idea consists in involving users in the data warehouse schema evolution, to create new granularity levels according to their own knowledge. Thus, we define the concept of “Aggregation Data Knowledge” (ADK), which defines how to aggregate data from a given level to another one to be created.

The ADK is represented in the form of “if-then” rules. These rules have the advantage of being very intelligible for users since they model information explicitly such as explained by Holland, J.H. et al. (1986), and are well adapted to define the aggregation link between two successive granularity levels. The *if*-clause contains conditions on the attributes of the lower level. The *then*-clause contains the definition of a higher level, and more precisely the values of the attributes, which characterize the new level to create.

For instance, let us assume that the person in charge of students' products needs to define the level *AGENCY TYPE* from the lower level *AGENCY*, since he knows that there is three types of agencies: some agencies host only student accounts, some others deal with foreigners, and other agencies are said to be classical. The following rules define the level *AGENCY TYPE*, by defining the values of the attributes *AgencyTypeLabel* and *AgencyTypeCode*, which characterize this level. The conditions expressed in the *if*-clause concern the attribute *Agency_ID* of the level *AGENCY*.

(R1) *if* \( \text{AgencyID} \in \{`01903´, `01905´, `02256´\} \)

*then* \( \text{AgencyTypeCode} = `STU´ \) and \( \text{AgencyTypeLabel} = `\text{student}´ \)

(R2) *if* \( \text{AgencyID} = `01929´ \)

*then* \( \text{AgencyTypeCode} = `\text{FOR}´ \) and \( \text{AgencyTypeLabel} = `\text{foreigner}´ \)

(R3) *if* \( \text{AgencyID} \notin \{`01903´, `01905´, `02256´, `01929´\} \)

*then* \( \text{AgencyTypeCode} = `\text{CLA}´ \) and \( \text{AgencyTypeLabel} = `\text{classical}´ \)

**Knowledge integration.** After the acquisition of the ADK under the form of “if-then” rules, these rules are integrated within the DBMS, which implements the current data warehouse. This integration 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 define one new granularity level, we associate one mapping table, which contains the conditions expressed on attribute(s) of the lower level, the corresponding value(s) for the attribute(s) in the created level, and some others useful information.

**Evolution.** The data warehouse schema evolution consists in updating the current schema by creating a new granularity level in a dimension hierarchy. 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. But, 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.

A user makes the choice to 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 instance, let us suppose the AGENCY dimension schema of Figure 3(a). In this case, it is possible to aggregate data according to Commercial Unit that is a set of agencies, according to their geographical localization. Let us suppose that we create a new level called Agencies Group, which is also a set of agencies, according to their geographical localization, but smaller than a commercial unit. Agencies Group level could be inserted between the AGENCY and Commercial Unit levels, where a commercial unit can be seen as a set of Agencies Groups (Figure 3(b)). Now, let us suppose that we add a new level that aggregates data according to the “size” of agencies: small, medium and large. It is semantically impossible to create an aggregation link between the Agency Size and the existing level Commercial Unit. In this case, the user cannot insert the created level between Agency and Commercial Unit, but he can create it to define a new hierarchy (Figure 3(c)).

**Figure 3: AGENCY dimension schemas**

With this user-centered architecture, users can create new analysis possibilities by creating new dimension hierarchy levels. Since this creation impacts the model, these analysis possibilities are shared with the whole of users. Note also, that the user can delete a level that he/she has created before.

**Metamodel for data warehouse model management**

To manage the schema update of a data warehouse according to our approach, we propose the UML model presented in Figure 4.
Figure 4: Data warehouse metamodel

A data warehouse is defined as a set of tables. The `DATA WAREHOUSE` and `TABLE` classes are linked with a composition link. These tables could be dimension or fact tables. Thus a generalization link connects `FACT` and `DIMENSION` classes to the `TABLE` class.

Each dimension table has a `DIMENSION PRIMARY ID` and various `DIMENSION ATTRIBUTES`. Moreover, fact tables present one or more `MEASURES` and a `PRIMARY ID`, which is an aggregation of foreign keys referencing `DIMENSION PRIMARY ID`.

There is an association between the `LEVEL` class and the `DIMENSION` class called `HIERARCHY`, to represent the fact that each dimension table corresponds to one or more granularity levels that constitute a dimension hierarchy.

The metamodel we propose allows ensuring genericity capabilities to our approach. Indeed it can generate every evolving data warehouse model.

Note that the main aspect of this metamodel to ensure the personalization process is the `LEVEL` class. Indeed, this class allows the user to add new levels representing the new analysis needs.

Workload Evolution: Our Approach

General architecture

To support the workload adaptation after changes in data warehouse schema, we propose the global architecture depicted in Figure 5. According to an initial data warehouse schema, a first workload is defined, containing the users' queries. A selection of optimization strategy can be carried out, based on this workload. When the data warehouse model evolves, according to the personalization process in our case, the workload has to be maintained.
Taking into account both the initial workload and the changes applied on the data warehouse schema, a workload evolution process is applied. This allows for the creation of an updated workload that contains queries which can be executed on the updated data warehouse. This workload contains not only queries that have been updated but also new queries if the data warehouse changes induce new analysis possibilities.

**Data warehouse schema evolution: changes and consequences**

In this section, we discuss changes arising in the data warehouse schema independently of the data sources, following the ROLAP (Relational OLAP) approach.

In a relational context, schema changes can occur at two levels: table and attribute. Besides classical concepts describing relational databases (table, attribute, foreign key, etc.), in data warehouses other specific concepts have to be considered (fact, dimension, level, measure, etc.). Indeed, changes occurred in a fact table or in a dimension table have different impacts.

In Table 1, we represent a typology of the main changes applied to the data warehouse schema and what their possible impacts on the workload are. We define what “concept” is modified, what kind of modification it is and its consequence on the workload. We consider three types of consequences: (1) updating queries; (2) deleting queries; (3) creating queries.

First, we note that we do not deal with the creation or the deletion of a fact table. In this case, there is an impact on dimension tables also. We think that these changes make the data warehouse schema evolve in an important way, so that the previous queries expressing analysis needs are no longer coherent or we are not able to define analysis needs due to a large number of possibilities.

Query updating consists in propagating the change on the syntax of the implied queries. More precisely, the syntax of the queries' clauses has to be rewritten when they use a concept that has changed. For the creation of a level, several possible queries can be created.

For the deleting operation, we can observe two consequences: propagation and deletion. This means that if rewriting (propagation) is not possible, the query is deleted.

Thereafter, we focus more precisely on query deleting and query creating that are required in the personalized context we defined previously.
Table 1. Typology of required changes for workload maintenance

<table>
<thead>
<tr>
<th>Role in the DW model</th>
<th>Type</th>
<th>Operation</th>
<th>Query Updating</th>
<th>Query Deleting</th>
<th>Query Creating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>Table</td>
<td>Creating</td>
<td></td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>Dimension and level</td>
<td>Table</td>
<td>Deleting</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Dimension and level</td>
<td>Table</td>
<td>Updating (Renaming)</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fact</td>
<td>Table</td>
<td>Updating (Renaming)</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measure</td>
<td>Attribute</td>
<td>Creating</td>
<td></td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>Measure</td>
<td>Attribute</td>
<td>Deleting</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Measure</td>
<td>Attribute</td>
<td>Updating (Renaming)</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimension descriptor</td>
<td>Attribute</td>
<td>Creating</td>
<td></td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>Dimension descriptor</td>
<td>Attribute</td>
<td>Deleting</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Dimension descriptor</td>
<td>Attribute</td>
<td>Updating (Renaming)</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Workload maintenance

In our context of personalization, we mainly investigated two types of workload maintenance: (1) query deleting and (2) query creating. Indeed, it is an answer to the fact that a user can add a granularity level or delete one (when he/she created it).

Query deleting. When a user deletes a level (that he/she created), we have to detect queries based on this level in order to delete them. Thus we define an evolution detection context in order to achieve the detection. We must be able to capture the concepts used in each query and to detect which query is concerned by the deletion of a concept.

In a relational context, decision support queries are based on clauses that exploit tables and attributes. Thus, we propose to represent this detection context by two matrices: one “queries-tables” matrix and one “queries-attributes” matrix. The “queries-tables” matrix has the various workload queries as lines and the various data warehouse tables as columns. The presence of a table in a clause of the query is marked by a “1” and the absence by a “0”. The “queries-attributes” matrix has the various workload queries as lines and the various attributes as “columns”. The use of an attribute in one of the query clauses is represented by a “1” and the absence by a “0”.

Note that this context can also be used for an evolution of attributes themselves. In our case, we say that we just delete queries that use the level that will be deleted. Thus we have just to consider the “queries-tables” matrix since a level corresponds to a table in the relational context. Then, when a level is deleted, we have just to find the queries where there is “1” for the table to be deleted.

Query creating. When a user creates a granularity level, the idea is to guide the administrator to create new queries according to this level. Concerning the definition of new queries exploiting the new analysis.
possibilities, we use the grammar that we defined in Darmont, J. et al. (2007). This grammar is a subset of the SQL-99 standard, which introduces much-needed analytical capabilities to relational database querying (Figure 6). Moreover, the idea is to exploit the metamodel defined previously, in order to guide the query creation.

**Figure 6: Decision-support queries grammar**

```sql
Query ::= [Select] [From] [Attribute Clause] [Aggregate Clause] [Group by Clause] [Having Clause] [Where Clause] [Condition Clause] [Operand Clause] [Join Clause] [Table Clause] [Alias] [Logical operator] [Comparison operator] [Attribute value] [Attribute value list]
```

**Case Study**

To illustrate our approach, let us consider the case study of the LCL French bank. Let us recall that the data warehouse concerns data about the annual Net Banking Income (NBI). Let us consider the following schema (Figure 7). In this schema, the NBI is analysed according to three dimensions: CUSTOMER, AGENCY and YEAR. The dimension AGENCY comprises a hierarchy with the level COMMERCIAL_DIRECTION that has been created by a user.

**Figure 7: Initial data warehouse schema for the NBI analysis**

According to the use of the data warehouse, a workload is defined from users’ queries. Usually, in an OLAP environment, queries require the computation of aggregates over various dimension levels. Indeed, given a data warehouse, the analysis process allows to aggregate data by using (1) aggregation operators such as SUM or AVG; and (2) GROUP BY clauses (GROUP BY CUBE, GROUP BY ROLLUP also). Here, we consider a
simple example with an extract of a workload comprising five queries that analyse the sum or the average of NBI according to various dimensions at various granularity levels (Figure 8).

Figure 8: Initial workload for the NBI analysis

Q1: SELECT Gender, Agency Label, YearLabel, AVG(NBI) FROM AGENCY, YEAR, TF-NBI WHERE AGENCY AgencyID=TF-NBI AgencyID AND YEAR YearID=TF-NBI YearID GROUP BY CUBE (Gender, Agency Label, YearLabel);

Q2: SELECT CommercialDirectionLabel, YearLabel, AVG(NBI) FROM AGENCY, COMMERCIAL_DIRECTION, TF-NBI WHERE AGENCY AgencyID=TF-NBI AgencyID AND COMMERCIAL_DIRECTION CommercialDirectionID=AGENCY DirectionID GROUP BY CUBE (CommercialDirectionLabel, YearLabel);

Q3: SELECT MaritalStatus, YearLabel, SUM(NBI) FROM CUSTOMER, TF-NBI WHERE CUSTOMER CustomerID=TF-NBI CustomerID GROUP BY CUBE (MaritalStatus, YearLabel);

Q4: SELECT CommercialDirectionLabel, AgencyLabel, SUM(NBI) FROM AGENCY, COMMERCIAL_DIRECTION, TF-NBI WHERE AGENCY AgencyID=TF-NBI AgencyID AND COMMERCIAL_DIRECTION CommercialDirectionID=AGENCY DirectionID AND YearLabel=2000 GROUP BY ROLLUP (CommercialDirectionLabel, AgencyLabel);

Q5: SELECT AgencyLabel, AVG(NBI) FROM AGENCY, COMMERCIAL_DIRECTION, TF-NBI WHERE AGENCY AgencyID=TF-NBI AgencyID AND COMMERCIAL_DIRECTION CommercialDirectionID=AGENCY DirectionID AND YearLabel=2000 AND CommercialDirectionLabel=Lyon GROUP BY AgencyLabel

The evolution detection context for this workload extraction is represented with the queries-tables matrix (Figure 9) and with the queries-attributes matrix (Figure 10).

Figure 9: Queries-tables matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>q2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>q3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>q4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>q5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 10: Queries-attributes matrix

<table>
<thead>
<tr>
<th></th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
<th>a5</th>
<th>a6</th>
<th>a7</th>
<th>a8</th>
<th>a9</th>
<th>a10</th>
<th>a11</th>
<th>a12</th>
<th>a13</th>
<th>a14</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Due to the personalization process, the following changes are applied on the data warehouse, resulting an updated schema (Figure 11):

- creation of the COMMERCIAL_UNIT level;
- deletion of the COMMERCIAL_DIRECTION level.
After the schema evolution, our approach is applied and an updated workload is provided (Figure 12). Firstly, the queries-tables matrix is used to detect that queries 2, 4 and 5 used the `COMMERCIAL_DIRECTION` level which has been deleted. Thus, these queries are deleted. Then the administrator is guided to create new queries on the `COMMERCIAL_UNIT` created level. In our example, he creates three queries (Q6, Q7 and Q8) with various clauses (GROUP BY, GROUP BY CUBE, GROUP BY ROLLUP) that exploit the new level.

Note that once the workload has been modified, the two matrices are also updated to represent properly the workload.

**Implementation Issues**

To validate our workload management system, we developed a prototype. We implemented our approach according to client/server architecture, with the Oracle 10g DBMS interfaced with PHP scripts in a platform named WEDriK (data Warehouse Evolution Driven by Knowledge).

The personalization process can be achieved by applying different algorithms, whose result is an evolution of the data warehouse schema. The sequence of these algorithms is represented in the Figure 13. The starting point of the algorithms sequence is a set of rules expressed by a user to create a granularity level. First an algorithm is used to check the validity of the rules. If the rules are not correct, the user has to modify them. This process is repeated until the rules are correct. If they are correct, the addition at the end of a hierarchy or the insertion between two existing successive levels of a new level is made according to the addition or the insertion algorithms respectively, by generating the corresponding queries. These algorithms are presented in Bentayeb, F. et al. (2008).
Once the data warehouse schema has evolved, we can consider the performance issue in the evolving context. Thus we are currently adding a module named WEPerf (data Warehouse Evolution Performance) to the WEDriK platform.

To achieve the whole process, we exploit the data warehouse metamodel presented previously, which is stored in a relational way. For instance, it allows for determining what the dimension hierarchies are, the links between tables in order to write new queries. This metamodel contains the semantic induced by the data warehouse model, particularly for the dimension hierarchies.

In one hand, we implemented a procedure that takes as input a workload and the data warehouse schema. This procedure builds the two matrices that represent the evolution detection context. These matrices are implemented under the form of relational tables.

In the other hand, changes on the data warehouse schema are stored into two relational tables: one for changes on tables, and one for changes on attributes. These tables present some useful information such as the evolution type, the name of the object that has changed. When changes occur, the two matrices are exploited to detect which ones of the queries are impacted. The workload is then modified according the required evolutions.

**Conclusion**

To consider an analysis personalization process in data warehouses, we proposed to involve users in data warehouse schema evolution. The idea was to integrate their knowledge to create new granularity levels. Thus, considering this evolving context and the OLAP objectives, what about performances? Indeed, to evaluate the efficiency of performance optimization techniques (such as index and materialized view techniques), we usually use a workload which is a set of representative (in terms of users’ analysis needs) queries, consistent with the data warehouse. With the data warehouse schema evolution, queries may be not representative anymore and the consistency can be corrupted. Thus, in this chapter, we designed a workload management system for data warehouse schema evolution model. Our workload model is dynamic, maintained incrementally and dynamically when data warehouse schema evolves, particularly in response to analysis personalization. The main advantage of our approach is to support the administrator task by providing him with a pro-active method to manage the data warehouse performances. Firstly it ensures that the existing queries remain consistent according to the data warehouse schema. Secondly it adds new queries to the existing workload based on new hierarchy levels in order to analyse performances on the whole data warehouse.

This work opens several perspectives. First of all, we plan to extend the possibilities of the workload maintenance. We want to consider not only structure evolution but also data evolution, such as evoked by Rizzi, S., & Golfarelli, M. (2006), since this type of changes may induce changes in queries. More
particularly, changes may require updates on WHERE clauses of the queries. For instance, we can consider value evolution of a foreign key.

In addition, we can investigate the possibilities of a semi-automatic process for creating new queries. For instance, it would be interesting to create automatically queries considering some parameters given by the administrator (number of queries to be created, constraints on the join numbers, etc.).

Moreover, we have to apply our proposal on other data warehouses more voluminous. Thus, we will be able to study the performance in a pro-active way since it is the final aim of our proposal. For instance, it may be interesting to study the impact of parameters such as the number of queries in the workload, the importance of changes in the data warehouse on our approach. And we have to determine the context in which the workload evolution has a real impact on the optimization strategy by testing algorithms that select optimization structures.

Acknowledgement

We would like to thank the anonymous referees for their helpful suggestions and comments on the previous draft of this chapter.

REFERENCES


Lawrence, M., & Rau-Chaplin, A. (2006) Dynamic View Selection for OLAP. In *Proceedings of 8th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 06)*, Volume 4081 of LNCS, Krakow, Poland (pp. 33-44).

Mendelzon, A.O., & Vaisman, A.A. (2000). Temporal Queries in OLAP. In *Proceedings of 26th International Conference on Very Large Data Bases (VLDB 00)*, Cairo, Egypt (pp. 242-253).


Rizzi, S., & Golfarelli, M. (2006) What Time Is It in the Data Warehouse ?. In *Proceedings of 8th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 06)*, Volume 4081 of LNCS, Krakow, Poland (pp. 134-144).
