Evolution of Data Warehouses' Optimization: A Workload Perspective

Cécile Favre, Fadila Bentayeb, and Omar Boussaid

University of Lyon (ERIC-Lyon 2) - Campus Porte des Alpes
5 av. Pierre Mendès-France, 69676 Bron Cedex, France
{cfavre|bentayeb}@eric.univ-lyon2.fr, omar.boussaid@univ-lyon2.fr
http://eric.univ-lyon2.fr

Abstract. Data warehouse (DW) evolution usually means evolution of its model. However, a decision support system is composed of the DW and of several other components, such as optimization structures like indices or materialized views. Thus, dealing with the DW evolution also implies dealing with the maintenance of these structures. However, propagating evolution to these structures thereby maintaining the coherence with the evolutions on the DW is not always enough. In some cases propagation is not sufficient and redeployment of optimization strategies may be required. Selection of optimization strategies is mainly based on workload, corresponding to user queries. In this paper, we propose to make the workload evolve in response to DW schema evolution. The objective is to avoid waiting for a new workload from the updated DW model. We propose to maintain existing queries coherent and create new queries to deal with probable future analysis needs.

Keywords: Data warehouse, Model, Evolution, Optimization strategy, Workload, Query rewriting.

1 Introduction

Data warehouses (DW) support decision making and analysis tasks by providing consolidated views. These consolidated views are built from the multi-dimensional model, based on heterogeneous data sources, of the DW. The multi-dimensional model being the most important component of a decisional architecture, the key point for the success of this type of architecture is the design of this model according to two factors: available data sources and analysis needs. As business environment evolves, several changes in the content and structure of the underlying data sources may occur. In addition to these changes, analysis needs may also evolve, requiring an adaptation of the existing multi-dimensional model. Thus the DW model has to be updated.

DWs containing large volume of data, answering queries efficiently required efficient access methods and query processing techniques. One issue is to use redundant structures such as views and indices. Indeed, among the techniques adopted in relational implementations of DWs to improve query performance,

I.Y. Song, J. Eder, and T.M. Nguyen (Eds.): DaWaK 2007, LNCS 4654, pp. 13–22, 2007.

view materialization and indexing are presumably the most effective ones [1]. One of the most important issues in DW 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 [2]. In this perspective, the workload should correspond to a set of users' queries. The most recent approaches syntactically analyze the workload to enumerate relevant candidate (indices or views) [3]. For instance, in [4], the authors propose a framework for materialized views selection that exploits a data mining technique (clustering), in order to determine clusters of similar queries. They also propose a view merging algorithm that builds a set of candidate views, as well as a greedy process for selecting a set of views to materialize. This selection is based on cost models that evaluate the cost of accessing data using views and the cost of storing these views.

The workload is supposed to represent the users' demand on the DW. In the literature, most of the proposed approaches rely on the existence of a reference workload that represents the target for the optimization [5]. However, in [6], the authors 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.

In views selection area, various works concern their dynamical selection [7,8,9]. The dynamic aspect is an answer to workload evolution. However, workload evolution can also be considered from different points of views. In [9], the authors consider that the view selection must be performed at regular maintenance intervals and is based on an observed or expected change in query probabilities. In [7,8], the authors 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.

In this paper, we want to define a different type of workload evolution. Indeed, we have to consider that the DW model update induces impacts on the queries composing the workload. Our key idea consists in providing a solution to make the workload queries evolve. The objective is to maintain the queries coherent with the DW model evolution and to create new queries expressing forthcoming analysis needs if required. Since the processing time for answering queries is crucial, it is interesting to adopt a pro-active behaviour. That is we propose to make the workload evolve in response to the model evolution. In this paper, we present our preliminary work concerning this issue.

The remainder of this paper is organized as follows. First, we detail the problem of query evolution in Section 2. Then, we present the DW schema evolutions and their impacts on the workload queries in Section 3. In Section 4, we focus on our approach to adapt queries of an existing workload, as a support for

optimization strategy evolution. Next, we present an example of our approach applied to a simplified case study in Section 5. We finally conclude and provide future research directions in Section 6.

2 The Query Evolution Problem

The problem of query evolution in DWs has been addressed in an indirect way. We evoked in the introduction 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 in [10], 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 [11]. Traditional database modeling 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, in [12], the authors first introduce and sketch a graph-based model that captures relations, views, constraints and queries. Then, in [13], the authors 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 paper, 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.

To achieve our workload evolution objective, we first define a typology of changes applied to the model (more precisely we focus on schema evolution) and their impacts on the workload. In this context, we tend to follow two goals: (1) maintaining the coherence of existing queries according to schema updates; (2) defining new queries when the schema evolution induces new analysis possibilities.

3 Data Warehouse Schema Evolution and Consequences

In this paper, we focus on the DW schema changes. We do not deal with data updating.

In a relational context, schema changes can occur on two levels: table or attribute. In the DW context, we cannot deal only with this concept. We have to introduce the DW semantic. Indeed, we have to consider that updates on a dimension table and on a fact table do not have the same consequences.

In Figure 1, we represent these changes and their impact on queries. We define what "concept" is modified, what kind of modification it is and the consequence on the workload. We consider three consequences: (1) updating queries; (2) deleting queries; (3) creating queries.

Role in the model	Type	Operation	Query Updating	Query Deleting	Query Creating
Dimension and level	Table	Creating			YES
Dimension and level	Table	Deleting	YES	YES	
Dimension and level	Table	Updating (Renaming)	YES		
Fact	Table	Updating (Renaming)	YES		
Measure	Attribute	Creating			
Measure	Attribute	Deletion	YES	YES	
Measure	Attribute	Updating (Renaming)	YES		
Dimension descriptor	Attribute	Creating			YES
Dimension descriptor	Attribute	Deleting	YES	YES	
Dimension descriptor	Attribute	Updating (Renaming)	YES		

Fig. 1. Schema evolution possibilities and consequences on the workload 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 DW model evolve in an important way, so that the previous queries expressing analysis needs are no longer coherent or we are not able to define future analysis needs due to a large number of possibilities.

Concerning the query updating, it consists in propagating the change on the syntax of the implied queries. More precisely, the syntax of the queries' clauses have to be rewritten when they use a concept that has changed. For the creation

of a level, several possible queries can be created. However, we suppose in this paper that we only create one query that integrates the created level.

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.

Furthermore, when a measure is created, we propose no consequence on the workload. Indeed, these changes do not induce coherence problems on existing queries. Thus it implies neither query updating nor query deletion. Moreover, since the workload allows for performance study, we do not create additional queries. Indeed, choosing one or another measure does not have any impact on the query performance.

4 Workload Evolution: Our Approach

In this section, we detail various aspects of our proposal. First, we provide the general architecture of our approach. Then, we detail the algorithm for the workload evolution. Finally, we provide some explanations about the implementation.

4.1 General Architecture

Our proposal of workload evolution is placed in a global architecture (Figure 2). According to the use of a DW, an initial workload is defined, containing the users' queries. A selection of the optimization strategy can be carried out, on the basis of this workload. When the administrator makes the DW model evolve, the queries can become incoherent, i.e. they can no longer be executed on the updated DW.

Taking into account both the initial workload and the changes applied on the DW model, a workload evolution process is applied. This allows for the creation

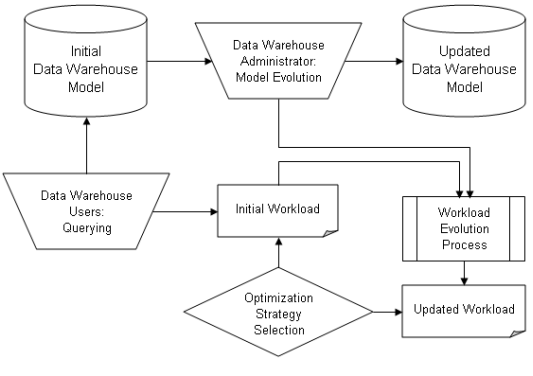

Fig. 2. Architecture for workload updating

of an updated workload that contains queries which can be executed on the updated DW. This workload contains not only queries that have been updated but also new queries if the DW changes induce new analysis possibilities. Note that this framework can be used for each DW model evolution. In the remainder of this paper we focus on schema (i.e. structure) evolution.

4.2 Algorithm

To achieve the workload evolution, we propose an algorithm which considers the DW schema evolutions and the initial workload as the input parameters (Algorithm 1).

First of all, concerning the rewriting process after the renaming of a concept, it consists in analyzing each clause of each query in the workload (SELECT, FROM, WHERE, GROUP BY) to find the old names and to replace them by the new ones.

Concerning the deletion of a concept, if a rewriting process fails, the corresponding query is deleted. The rewriting process can fail if the query no longer makes sense. For instance, if a dimension is deleted in the model and the existing query is an analysis according this dimension and no other dimension, we have to delete this one. Note that, as mentioned in the algorithm, we do not consider the deletion of measure even if it is the only one.

Concerning the creation of a concept (dimension table or attribute), we define a decisional query according to this new attribute or table. In such a query, we apply the operator AVG by default on a measure of the fact table. In the case of a table creation (resp. attribute), we can deal with the primary key of the new table (resp. this attribute) as a group by element.

4.3 Implementation

We implemented our approach according to a client/server architecture, with the Oracle 10g DBMS interfaced with PHP scripts. We suppose that a tool is used by the administrator to manage the DW model, and this tool provides the various changes that are carried out on the model. These changes 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.

Furthermore, we exploit the DW meta-model which is stored in a relational way. For instance, it allows for determining what are the dimension hierarchies, the links between tables in order to write new queries. This meta-model contains the semantic induced by the DW model, particularly for the dimension hierarchies. Indeed, a dimension hierarchy corresponds to a set of semantical relations between values.

The queries of the workload are prepared to be transformed into a relational representation. More precisely, a table gathers the ID of the query, the SELECT

Algorithm 1. Workload Evolution Process

Require: workload W, set of DW schema evolutions E
Ensure: updated workload W'
Copying the queries of W into W'
for all evolution $e \in E$ **do**
  if e="renaming dimension, level table" OR "renaming fact table" OR "renaming measure" OR "renaming dimension descriptor" **then**
    Rewriting the implied decisional queries according to the new names
  end if
  if e="creating granularity level, dimension" OR "creating a dimension descriptor" **then**
    Searching for relational link from the fact table to the concerned level table
    Writing a decisional query according to the new level: (created attribute or table key in the SELECT clause)
    Adding the query to the workload W'
  end if
  if e="deleting granularity level or dimension" **then**
    Searching for another level in the same dimension
    Rewriting the implied decisional queries according to the replaced level or deleting the queries if rewriting is not possible
  end if
  if e="deleting a measure" **then**
    Searching for another measure in the same fact table
    {We suppose that there is another measure, on the contrary, it is a fact table deletion and not only a measure deletion }
    Rewriting the implied decisional queries according to the replaced measure
  end if
  if e="deleting a dimension descriptor" **then**
    Searching for another dimension descriptor in the same level (table key by default)
    {Rewriting the implied decisional queries according to the replaced attribute}
  end if
end for
return workload W'

clause, the FROM clause, the WHERE clause, the GROUP-BY clause. The changes are carried out on this table, before computing the workload file.

5 Example

To illustrate our approach, we use a case study defined by Le Crédit Lyonnais french bank[1]. The DW model concerns 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). The dimension AGENCY comprises a hierarchy with the level COMMERCIAL_DIRECTION. Each commercial direction corresponds to a group of agencies.

According to the use of the DW, 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 DW model, the analysis process allows to aggregate data by using (1) aggregation operators such as SUM or AVG; and (2) GROUP BY clauses. Here, we consider a simple example with a small workload comprising five queries that analyze the sum or the average of NBI according to various dimensions at various granularity levels (Figure 4).

[1] Collaboration with LCL-Le Crédit Lyonnais (Rhône-Alpes Auvergne).

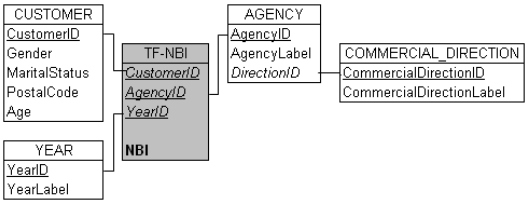

Fig. 3. Initial DW model for the NBI analysis

Q1: **SELECT** AgencyLabel, YearLabel, AVG (NBI) **FROM** AGENCY, YEAR, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND YEAR.YearID=TF-NBI.YearID **GROUP BY** AgencyLabel, 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** CommercialDirectionLabel, YearLabel;

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

Q4: **SELECT** CommercialDirectionLabel, YearLabel, AVG (NBI) **FROM** AGENCY, COMMERCIAL_DIRECTION, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND COMMERCIAL_DIRECTION.CommercialDirectionID=AGENCY.DirectionID AND YearLabel='2000' **GROUP BY** CommercialDirectionLabel, YearLabel;

Q5: **SELECT** AgencyLabel, SUM (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;

Fig. 4. Initial Workload for the NBI analysis

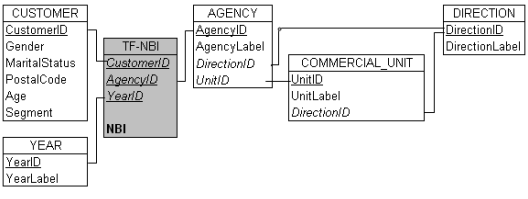

Fig. 5. Updated DW model for the NBI analysis

Due to evolutions of data sources and analysis needs, the following changes are applied on the DW model, resulting an updated model (Figure 5):

- adding the attribute Segment in the CUSTOMER dimension;
- renaming the COMMERCIAL_DIRECTION level to DIRECTION;
- renaming the CommercialDirectionID and the CommercialDirectionLabel attributes respectively to DirectionID and DirectionLabel;
- inserting the COMMERCIAL_UNIT level between AGENCY and DIRECTION levels.

After the schema evolution, our algorithm is applied, and an updated workload is provided (Figure 6). We can note that queries' syntax is updated to remain coherent with the DW model. Indeed, the schema's updates have been propagated to the required queries. Moreover, two queries have been added, to take into account potential analysis needs which concern the newly created COMMERCIAL_UNIT level and the newly created Segment attribute.

Q1: **SELECT** AgencyLabel, YearLabel, AVG (NBI) **FROM** AGENCY, YEAR, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND YEAR.YearID=TF-NBI.YearID **GROUP BY** AgencyLabel, YearLabel;

Q 2: **SELECT** DirectionLabel, YearLabel, AVG (NBI) **FROM** AGENCY, DIRECTION, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND DIRECTION.DirectionI D=AGENCY.DirectionID **GROUPBY** DirectionLabel, YearLabel;

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

Q4: **SELECT** DirectionLabel, YearLabel, SUM (NBI) **FROM** AGENCY, DIRECTION, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND DIRECTION.DirectionI D=AGENCY.DirectionID AND YearLabel='2000'**GROUP B Y** DirectionLabel, YearLabel;

Q5: **SELECT** AgencyLabel, AVG (NBI) **FROM** AGENCY, DIRECTION, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND DIRECTION.DirectionID=AGENCY.DirectionID AND YearLabel='2000' AND DirectionLabel='Lyon' **GROUP BY** AgencyLabel;

Q6: **SELECT** UnitLabel, AVG (NBI) **FROM** AGENCY, COMMERCIAL_UNIT, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND COMMERCIAL_UNIT.UnitID=AGENCY.UnitID **GROUP BY** UnitLabel;

Q7: **SELECT** Segment, AVG (NBI) **FROM** CUSTOMER, TF-NBI WHERE CUSTOMER.CustomerID=TF-NBI.CustomerID AND **GROUP BY** Segment;

Fig. 6. Updated Workload for the NBI analysis

6 Conclusion

In this paper, we presented a preliminary work to address the problem of query evolution in DWs. We focused particularly on the problem of workload evolution to support optimization strategy evolution. We proposed a global approach to achieve this task and an algorithm. We presented a simplified example based on a case study to illustrate our approach. The main advantage of our approach is to support the administrator by a pro-active method. Firstly it ensures that the existing queries remain coherent according to the schema evolution. Secondly it provides new queries to take into account possible analysis needs.

There are still many aspects to be explored. First of all, we want to develop a tool based on our proposal that helps the DW administrator in the evolution process. In this case, the workload evolution process can be extended to propose several consequences to be applied in a semi-automatic way and to create more complex queries by involving the administrator. For instance, for the deletion of a level which is between two others, the administrator can choose to rewrite the queries according to the higher level, or the lower level or both. Furthermore, in addition to the schema evolutions, we have to integrate the evolution of data in our approach. For instance, we can consider value evolution of a foreign key. This type of changes induces update on WHERE clauses of the queries. Moreover, we have to carry out a performance study in order to precisely determine in which context our approach is more interesting (instead of a manual evolution made by the administrator himself). We may study parameters such as the number of queries in the workload, the importance of changes in the DW. Finally, we want to integrate the evolution aspect in a benchmark. Indeed a benchmark allows for deciding of optimization strategies before the real use of the DW. And it can help to evaluate the impact of a change and to decide whether the change has to be applied or not. This issue requires to equip benchmarks with evolution operators in order to make the benchmark evolve.

Acknowledgments. The authors would like to thank Mohamed Bekhouche, Lorène Ducommun and Salim Guergouri for their contribution to this work.

References

1. Rizzi, S., Saltarelli, E.: View Materialization vs. Indexing: Balancing Space Constraint in Data Warehouse Design. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 502–519. Springer, Heidelberg (2003)
2. Gallo, J.: Operations and maintenance in a data warehouse environment. DM Review Magazine (2002), http://www.dmreview.com/article_sub. cfm?articleId=6118
3. Agrawal, S., Chaudhuri, S., Narasayya, V.R.: Automated selection of materialized views and indexes in sql databases. In: XXVIth International Conference on Very Large Data Bases (VLDB 00), Cairo, Egypt, pp. 496–505. Morgan Kaufmann, San Francisco (2000)
4. Aouiche, K., Jouve, P., Darmont, J.: Clustering-Based Materialized View Selection in Data Warehouses. In: Manolopoulos, Y., Pokorný, J., Sellis, T. (eds.) ADBIS 2006. LNCS, vol. 4152, pp. 81–95. Springer, Heidelberg (2006)
5. Theodoratos, D., Bouzeghoub, M.: A General Framework for the View Selection Problem for Data Warehouse Design and Evolution. In: IIIrd ACM International Workshop on Data Warehousing and OLAP (DOLAP 00), Washington, Columbia, USA, pp. 1–8. ACM Press, New York, NY, USA (2000)
6. Golfarelli, M., Saltarelli, E.: The Workload You Have, the Workload You Would Like. In: VIth ACM International Workshop on Data Warehousing and OLAP (DOLAP 03), New Orleans, Louisiana, USA, pp. 79–85. ACM Press, New York, NY, USA (2003)
7. Kotidis, Y., Roussopoulos, N.: DynaMat: A Dynamic View Management System for Data Warehouses. SIGMOD Rec. 28(2), 371–382 (1999)
8. Theodoratos, D., Sellis, T.: Incremental Design of a Data Warehouse. Journal of Intelligent Information Systems 15(1), 7–27 (2000)
9. Lawrence, M., Rau-Chaplin, A.: Dynamic view selection for olap. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK 2006. LNCS, vol. 4081, pp. 33–44. Springer, Heidelberg (2006)
10. Bellahsene, Z.: Schema Evolution in Data Warehouses. Knowledge and Information Systems 4(3), 283–304 (2002)
11. Vassiliadis, P., Papastefanatos, G., Vassiliou, Y., Sellis, T.: Management of the Evolution of Database-Centric Information Systems. In: Ist International Workshop on Database Preservation (PresDB 07), Edinburgh, Scotland, UK (2007)
12. Papastefanatos, G., Kyzirakos, K., Vassiliadis, P., Vassiliou, Y.: Hecataeus: A Framework for Representing SQL Constructs as Graphs. In: XXth International Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD 05), in conjunction with the XVIIth International Conference on Advanced Information Systems Engineering (CAiSE 05), Oporto, Portugal (2005)
13. Papastefanatos, G., Vassiliadis, P., Vassiliou, Y.: Adaptive Query Formulation to Handle Database Evolution. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, Springer, Heidelberg (2006)