Proceedings of 12th International Workshop on Database and Expert Systems Applications (DEXA 2001), ISBN 0-7695-1230-5, pp 128-132

Design of a Data Warehouse over Object-Oriented and Dynamically Evolving Data Sources

Bodgan Czejdo[1], Johann Eder[2],
Tadeusz Morzy[3], Robert Wrembel[3]
[1] *Department of Mathematics and Computer Science, Loyola University*
St. Charles Ave., New Orleans, LA 70118
czejdo@loyno.edu
[2] *Institut für Informatik-Systeme, Universität Klagenfurt*
Universitätsstr. 65, A-9020 Klagenfurt, Austria
eder@ifi.uni-klu.ac.at
[3] *Institute of Computing Science, Poznań University of Technology*
Piotrowo 3A, 60-965 Poznań, Poland
morzy@put.poznan.pl, Robert.Wrembel@cs.put.poznan.pl

Abstract

In this paper we present some of the results achieved while realizing an international research project aiming at the design and development of an Object-Relational Data Warehousing System (ORDAWA). The most important goals of the project are as follows: the development of techniques for the integration and consolidation of different external data sources in an object-relational data warehouse, the construction and maintenance of materialized relational as well as object-oriented views, and the development of techniques allowing to modify the schema of a data warehouse when data sources change their schemas.

1. Introduction

Organizational decision support systems require a comprehensive view of all aspects of an enterprise. Information collected in an enterprise are often of different data format and complexity (e.g. relational, object-relational, and object-oriented databases, on-line multimedia data stores, Web pages, spreadsheets, flat files). Therefore, the ability to integrate information from different, autonomous and heterogeneous external data sources (EDS) is crucial for today's businesses.

One of the basic approaches to integrate distributed external data sources and to provide integrated information to users [9, 2] is the *data warehousing* approach (or materialized views approach). The data warehousing approach offers: (1) high query performance, since all data is retrieved from a single location without the necessity of accessing remote external data sources which may be expensive and time consuming (due to the network delays), (2) higher availability of data since it can operate even when EDSs become temporarily unavailable, (3) logical and physical data independence between EDSs and global applications (at least partially) since global queries posed against the data warehouse are not visible outside the data warehouse. On the other hand, local processing at local sources is not affected by global applications. Due to its properties, the data warehousing approach has been found to be an extremely useful technology for information integration for large and medium companies requiring high query performance and high data availability [9, 2, 6].

Research in the data warehousing area focused on design issues, data maintenance strategies and query optimization in connection with relational view materialization and implementation issues. However, two important research issues have received relatively little attention so far: integration of object-oriented and/or object-relational databases with data warehouses and evolution of a data warehouse schema resulting from schema changes in underlying external data sources.

The more and more frequent need to create, store, and manage data of a complex structure and behavior leads to the make use of object-relational or object-oriented databases. These complex data, similarly as relational, will be the subject of the integration and analysis in a data warehouse. To this end, the data model of such a data warehouse should support complex types in order to reflect the complexity of source information. Moreover, object data use the methods and these methods should also be available in the integrated system, in order to retain the functionality and information as well as to be able to process the information. Therefore, recently some research papers suggest to use an object-relational or object-oriented data model (cf. [1]) as a common model for integrating heterogeneous data sources.

The second problem is related to dynamicity of external data sources. The data warehouse integrates autonomous and heterogeneous EDSs. The main feature of EDSs that makes the integration process difficult is autonomy of EDSs. Autonomy means that the external

data sources were designed and developed independently, and that they preserve the autonomous control over their data. An important consequence of the autonomy of EDSs is that they may evolve in time independently of each other and that they may change their contents (i.e. their data) as well as their schemas, without being controlled from the data warehouse level. When EDSs change their contents, these changes have to be propagated to the data warehouse to ensure that the data warehouse is kept up to date. However, the content of a data warehouse may become obsolete also because local schemas of EDSs have changed. Therefore, there is evidently a need to analyze and study the propagation of EDS schema changes to data warehouse systems.

In this paper, we present and briefly discuss some results obtained within the scope of the international research project, called *ORDAWA*, aiming at the design and development of an Object-Relational Data Warehousing System. The results concern: (1) the integration of object-oriented data in a data warehouse and (2) schema evolution in a data warehouse.

This paper is organized as follows. Section 2 briefly discusses the goals of the *ORDAWA* project and presents a data warehouse architecture. Section 3 discusses our approach to the design and maintenance of materialized object-oriented views. Section 4 discusses the impact of external data source schema changes on the data warehouse schema, the data warehouse content, and the middleware. Section 5 summarizes and concludes the paper.

2. *ORDAWA* Project

2.1. Goals of the Project

An international research project aiming at the design and development of an Object-Relational Data Warehousing System - *ORDAWA* was set up a year ago [4]. The project is conducted in co-operation of the Institute for Informatics-Systems at Klagenfurt University, the Institute of Computing Science at Poznań University of Technology, and the Department of Mathematics and Computer Science at Loyola University.

The most important goals of the *ORDAWA* project are as follows:

- the development of techniques for the integration and consolidation of different external data sources in an object-relational data warehouse;
- the construction and maintenance of materialized relational as well as object-oriented views;
- the development of techniques allowing to modify the schema of a data warehouse when data sources change their structures.

The results achieved so far concern: (1) the construction and maintenance of materialized object-oriented views [10, 7, 11, 12] and (2) the design of a data warehouse with dynamically changing data sources [3].

2.2. Operational Data Store

In our approach we use the architecture of a data warehouse with an operational data store, cf. [6]. An operational data store is a subject-oriented set of data coming from one or more data sources. The purpose to build an ODS is to separate long lasting, time consuming processing, e.g. On-Line Analytical Processing queries, from data sources. Since OLAP queries operate on data in an ODS, they do not interfere with the processing in data sources. Furthermore, an ODS may be designed and tuned especially for a particular pattern of processing, whereas the underlying data sources may be designed and tuned for other kind of processing, e.g. OLTP.

The need to bring into a warehouse the data of various complex formats implies that rich data model must be used in an operational data store. To this end, an object-oriented data model seems to be very promising [1, 5, 4].

Object-oriented views are important mechanisms providing an integrated access to data of complex structure and behavior. We argue that object-oriented features, especially methods, can bring a new powerful mechanism in data transformation, integration, and analysis in an object-relational data warehouse. Due to the high expressiveness of an object-oriented data model, object-oriented views seem to be well suited for the transformation and integration of many different data sources that use different data models. In our approach an ODS is modeled as the set of materialized object-oriented views.

2.3. Requirements for Object-Oriented Views

For object-oriented views that are used to ease the process of complex data warehousing, we identified the set of the following basic features that such views have to support.

Feature 1: View complexity. A view should be defined as a schema of an arbitrary complex structure and behavior, as data to be accessed by views may be complex.

Feature 2: View materialization. A view should provide means for materialization of its instances and consistency maintenance of such materialized data, due to efficiency reasons.

Feature 3: View separation from a source database. This feature means that referencing from a view the source database is not allowed. The main reason for the separation is as follows. A view schema that is used for data integration in an object-relational data warehousing system has to make available the means to integrate and locally materialize data coming from remote databases in

order to reduce access time to data and to eliminate the problem of temporary unavailability of data sources.

3. Designing and Implementing Materialized Object-Oriented Views

Within the area of object-oriented views application in data warehousing process we developed the concept of an object-oriented view that support the three features mentioned in Section 2.3.

3.1. The Concept of View Schema

In our approach, called *View Schema Approach* (VSA), an *object-oriented view* is defined as a *view schema* of an arbitrary complex structure and behavior, composed of view classes. Each *view class* is derived from one or more classes in a database schema. A view class is derived by an OQL-like command defining its structure, behavior, and set of instances [11, 12]. View classes in a view schema are connected by inheritance and aggregation relationships. Several view schemas may be defined in one database and each of them is uniquely identified by its name.

In order to check whether a view schema was designed correctly, we proposed two kinds of consistency criteria, namely: *closure* and *well formed inheritance* (cf. [12]).

3.2. View Schema Materialization and Maintenance - Structural Parts

Similarly as materialized relational views, a materialized object-oriented view schema has to be kept up-to-date with the content of a source database. Three following techniques for keeping a materialized view schema up-to-date were developed within the *View Schema Approach*, namely:

- deferred on commit incremental refreshing;
- deferred on demand incremental refreshing;
- deferred on demand complete refreshing.

Refreshing given view schema VS_i means that all the materialized instances of view classes in VS_i are refreshed. However, the incremental refreshing mode developed in our *View Schema Approach* is applicable to a limited set of view classes (for details refer to [12]).

In order to incrementally propagate the modifications from base to view objects we have developed additional data structures, called *Class Mapping Structure* (***CMS***), *Object Mapping Structure* (***OMS***), and *Log*, cf. [11, 12].

Class Mapping Structure. The structure is used to store derivation links between base class C_i and view class V_i derived from C_i. ***CMS*** is used while recording the changes made to base objects whether the modification needs to be propagated to materialized view objects.

Object Mapping Structure. The update of base object o_i should be propagated only to those view objects that were derived from o_i. Therefore, one important issue that must be solved is the identification of these materialized view objects that are affected by the update of o_i. To this end, ***OMS*** is used. The system uses the content of a given ***OMS*** also while creating complex view objects (cf. [12]).

Log. Modifications made to base objects are recorded in a data structure called *Log*. *Log* is associated with each base class from which a view class has been derived. It is created automatically by the system when creating a view class. The content of logs are used while refreshing a view schema.

3.3. View Schema Materialization and Maintenance - Behavioral Parts

By default, methods defined in a view class are computed each time they are invoked. When the computation of a method result takes long time it may be reasonable to store the result persistently in an ODS, i.e. materialize it. After materializing method m_i, the result of the first invocation of m_i for view object vo_i is stored persistently. Each subsequent invocation of m_i for the same object vo_i uses the already materialized value.

Methods may have various numbers of input arguments, that can be of different types. Methods that have input arguments are not good candidates for the materialization. However, in the *View Schema Approach* a method with input arguments can be materialized and maintained within acceptable time overhead provided that: (1) the method has few input arguments and (2) each of the arguments has a narrow, discrete domain.

We have proposed a novel technique of method materialization, called *hierarchical materialization*. When hierarchical materialization is applied to method m_i, then the result of *mi* is stored persistently and additionally, the results of methods called from m_i are also stored persistently. Hierarchical materialization may be useful only for those methods that call other methods and the computation of those called methods is costly.

The maintenance of materialized methods can take the advantage of hierarchical materialization. When a view object vo_i, used to materialize the result of method m_i, is updated or deleted, then m_i has to be recomputed. This recomputation can use unaffected intermediate materialized results, thus reducing the recomputation time overhead.

In order to materialize methods in a view class and maintain the materialized results, three additional data structures have been created in an operational data store. These structures, which are described below, are called *View Methods*, *Materialized Method Results Structure*, and *Graph of Method Calls* (cf. [7, 12]).

3.4. View Schema Approach Prototype

The theoretical foundation concerning a materialized object-oriented view have been incorporated into a prototype software, called the *View Schema Approach Prototype* (*VSAP*). The prototype has been implemented in C++ on top of the *Oracle8i* DBMS.

The functionality that has been implemented and is supported by the *View Schema Approach Prototype* is the following: (1) the creation and management of several view schemas, (2) the creation and management of view classes in a given view schema, (3) the materialization of view schemas and the maintenance of their consistency, and (4) checking the consistency of a view schema.

4. Management of Changes in a Data Warehouse Schema

As we mentioned already in the introduction, EDSs may evolve in time independently of each other and they may change their contents (i.e. their data) as well as their schemas, without being controlled from the data warehouse level. The local changes visible at the integration level can be categorized as: (1) content changes such as insert/delete/update tuples, and (2) schema changes such as add/delete/modify attribute or add/drop table. Most of the research done so far with respect to the data warehouse refreshment has focused on the issue of how to provide transactional incremental data warehouse refresh under content changes of EDSs. However, schema changes of EDSs during their life cycles are very common and inevitable as reported by different studies [8].

Integrating data from evolving EDSs raises new challenges in the maintenance and evolution of data warehouse systems. These challenges can be classified into three groups [8]:

- modifying data warehouse schema according to data source schema changes;
- modifying middleware level according to data source schema changes;
- modifying data warehouse content according to data source schema changes.

4.1. Modifying data warehouse schema according to EDS schema changes

Usually, a data warehouse schema is defined in terms of EDS exported schemas (the data warehouse defined as a set of materialized views over EDS exported schemas). Therefore, changes in underlying EDS schemas will result in an invalid definition of the data warehouse schema. They may change also the functionality of the data warehouse. There are two possible solutions to cope with EDS schema changes; either ensure the correct propagation of these changes to the data warehouse definition, or appropriately isolate the data warehouse from these changes. Isolation can be accomplished by the middleware level. However, the second approach is applicable for a limited period of time only, since evolution of EDS schemas will lead to inconsistency between EDSs and their descriptions at the data warehouse level. It may restrict the usage of existing client queries and reports (removal of some attributes, attribute meaning changes), or some important data will not be available for data warehouse clients (addition of new attributes, changes of attribute meanings). Another important aspect of the adaptation of the data warehouse system to EDS schema changes concerns the metadata repository. Evolution of EDS schemas must be propagated also to the metadata repository.

4.2. Modifying middleware level according to EDS schema changes

New solutions and algorithms are necessary to provide dynamic adaptation of middleware to EDS schema changes. The first problem is how to detect EDS schema changes. Even, if a data source is a fully-fledged DBMS using the replication option, it does not mean that it notifies the middleware level of the data warehouse system about these kind of changes. In general, we may distinguish two classes of EDS, depending on the level of cooperation between the EDS and a higher level of data warehouse system: cooperating sources and non-cooperating sources. The first type of a source is able to notify the higher levels of a data warehouse system about all changes that are made at the source level. The second type of the data source provides no cooperation functionality or a very limited one. However, for both types of data sources, it is necessary to adapt middleware to evolving data sources in such a way that wrappers are able to preserve their capabilities to translate all updates/queries types through wrappers. It is also obvious that other functions at the middleware level (data cleansing, transformation, etc.) should be adapted to data source changes.

4.3. Modifying data warehouse content according to EDS schema changes

It is clear from above that if we decided to modify a data warehouse schema according to EDS schema changes, these changes should be accompanied by corresponding adjustments of the data warehouse content. The data warehouse content adjustments can be done in two different ways. Once the definition and conceptual data warehouse schema have changed, the original data warehouse schema and corresponding data are archived and replaced by an altered data warehouse schema. The

data warehouse server has to be reloaded with new data. This solution leads to a "multiversion" data warehouse system and raises new challenges in the maintenance and evolution of such data warehousing systems. The second approach is based on adjustment of data warehouse content according to changes in the data warehouse schema caused by EDS changes. For example, adding a new attribute to an existing relation in the data warehouse will require data for this new attribute for all previously created tuples. Moreover, it also may be necessary to adjust some of the aggregates derived from basic relations.

However, the main problem is how to adapt the data warehouse content to EDS schema changes, which change meaning of attributes or meaning of some aggregates. It is easy to notice that this problem is closely related to the more general problem of how to provide the logical independence between the conceptual level of the data warehouse system and external level (i.e. existing front-end applications). Is it possible to use existing applications with respect to the altered data warehouse schema and content to produce correct answers? From this point of view, we may classify front-end applications into: applications that do not require any changes, applications that would run correctly after some changes, and applications that are no longer valid.

If EDS schema changes did not affect the part of the data warehouse schema that is of interest to an application, then the application does not require any modification. Sometimes, an application may be easily modified according to the altered data warehouse schema resulting from EDS changes. However, some changes to the data warehouse schema may result in invalid application since information necessary to run the application is not available in the data warehouse.

5. Summary

In this paper, we presented and briefly discussed some results obtained within the scope of the international research project *ORDAWA*, aiming at the design and development of an Object-Relational Data Warehousing System. The results concern two problems of data warehousing, namely: (1) integration of heterogeneous data in a data warehouse and (2) schema evolution in a data warehouse. With regard to the first problem we have developed: (1) the concept of an object-oriented view, defined as a view schema, (2) consistency criteria used for the verification of a view schema correctness, (3) view schema materialization and maintenance techniques, (4) methods materialization and maintenance technique in a view which, to the best of our knowledge, is the only such work done so far; we have proposed a novel method materialization technique, called *hierarchical materialization*, (5) the implementation of a prototype system supporting the construction, materialization, and maintenance of view schemas.

With regard to the second problem we presented the issue of a data warehouse system evolution resulting from the changes in underlying external data sources (EDS). We discussed how EDS schema changes affect (1) data warehouse schema, (2) middleware level, and (3) data warehouse content.

Further research issues will concern the development of schema and data versioning techniques in a data warehouse as well as the design of a query language applicable in a multiversion data warehouse.

6. References

[1] Bukhres O. A., Elmagarmid A. (eds.): Object-Oriented Multidatabase Systems: A Solution for Advanced Applications, Prentice Hall, 1996

[2] Chaudhuri S., Dayal U.: An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 26, 1997

[3] Czejdo B., Messa K., Morzy T., Putonti C.: Design of Data Warehouses with Dynamically Changing Data Sources. In Proc. of the Southern Conference on Computing. October, 2000

[4] J.Eder, H.Frank, T.Morzy, R.Wrembel, M.Zakrzewicz, Designing an Object-Relational Database System: Project ORDAWA. Proc. of challenges of the ADBIS-DASFAA'00 Conference, Czech Republic, 2000

[5] Fankhauser P., Gardarin G., Lopez M., Munoz J., Tomasic A.: Experiences in Federated Databases: From IRO–DB to MIRO–Web. Porc. of the VLDB Conference, USA, 1998, pp. 655-658

[6] Jarke M., Lenzerini M., Vassiliou Y., Vassiliadis P.: Fundamentals of Data Warehouses. Springer-Verlag, 2000, ISBN 3-540-65365-1

[7] Morzy T., Wrembel R., Koszlajda T.: Hierarchical materialisation of method results in object-oriented views. Proc. of the ADBIS-DASFAA'00 Conference, Czech Republic, 2000, LNCS 1874, Springer-Verlag, pp. 200-214

[8] Rudensteiner E., Koeller A., and Zhang X.: Maintaining Data Warehouses over Changing Information Sources. Communications of the ACM, vol. 43, No. 6, 2000

[9] Widom J.: Research Problems in Data Warehousing, Proc. of the 4th Int. Conference on Information and Knowledge Management (CIKM), 1995, pp. 25-30

[10] Wrembel R.: On a formal model of an object-oriented database with views supporting data materialisation. Proc. (of short papers) of the ADBIS'99 Conference, Slovenia, 1999, pp. 109-116

[11] Wrembel R.: On Materialising Object-Oriented Views. In Barzdins J., Caplinskas A. (eds.): Databases and Information Systems. Kluwer Academic Publishers, 2001, ISBN 0-7923-6823-1, pp. 15-28

[12] Wrembel R.: The Construction and Maintenance of Mateiralised Object-Oriented Views in Data Warehousing Systems. PhD thesis, Poznań University of Technology, Institute of Computing Sicence, Poznań, Poland, 2001