UNIVERSITY OF LATVIA

DARJA SOLODOVNIKOVA

DATA WAREHOUSE
SCHEMA EVOLUTION-ORIENTED
REPORTS DEFINITION AND EXECUTION TOOL

Summary of Doctoral Thesis

Rīga - 2010
UNIVERSITY OF LATVIA

DARJA SOLODOVNIKOVA

DATA WAREHOUSE
SCHEMA EVOLUTION-ORIENTED
REPORTS DEFINITION AND EXECUTION TOOL

Summary of Doctoral Thesis

Advisor:
Dr. sc. comp.
LAILA NIEDRĪTE

Riga - 2010
The Thesis was supported by the European Social Fund (ESF) Projects Nr. 2009/0216/1DP/1.1.1.2.0/09/APIA/VIAA/044 and 2009/0138/1DP/1.1.2.1.2/09/IPIA/VIAA/004

Advisor:

Dr. sc. comp. Laila Niedrīte
University of Latvia

Referees:

Professor, Dr. sc. comp. Jānis Bičevskis,
University of Latvia

Dr. sc. comp. Anita Jansone,
Liepaja University

Professor, Dr. sc. comp. Hele-Mai Haav,
Institute of Cybernetics at Tallinn University of Technology, Estonia

The defence of the thesis will take place in an open session of the Council for Promotion in Computer Science, University of Latvia, ________________________________, in the Institute of Mathematics and Computer Science, the University of Latvia (Rīga, Raiņa bulv. 29, room 413).

The thesis (collection of works) and its summary are available at the Library of the University of Latvia (Kalpaka bulv. 4, Rīga).

Head of the Council                                                         J.Bārzdinš
# Table of Contents

**GENERAL DESCRIPTION** 4

**RESEARCH MOTIVATION AND TOPICALITY** 4
**GOAL OF THE THESIS** 4
**MAIN RESULTS OF THE THESIS** 5
**PUBLICATION OF THE RESULTS OF THE THESIS AND PRESENTATIONS AT CONFERENCES** 5
**STRUCTURE OF THE THESIS** 6

**MAIN ELEMENTS OF A DATA WAREHOUSE** 7

**DATA WAREHOUSE EVOLUTION PROBLEMS** 8

**DEVELOPMENT OF E-LEARNING DATA MART** 12

**DATA WAREHOUSE EVOLUTION FRAMEWORK** 14

**THE FORMAL MODEL FOR MULTIVERSION DATA WAREHOUSE AND METADATA** 17

**DATA WAREHOUSE SCHEMA CHANGES** 23

**QUERY GENERATION ON DATA WAREHOUSE AND USAGE OF REPORTS** 28

**TECHNICAL IMPLEMENTATION OF THE REPORTING TOOL** 30

**EVALUATION OF THE PROPOSED APPROACH** 32

**CONCLUSION** 36

**REFERENCES** 40

**PUBLICATIONS OF THE AUTHOR IN INTERNATIONAL SCIENTIFIC EDITIONS AND CONFERENCE PROCEEDINGS** 40
**OTHER PUBLICATIONS, REPORTS, ARTICLES OF THE AUTHOR** 40
**OTHER AUTHORS’ PUBLICATIONS** 40
General Description

Research Motivation and Topicality

Data warehouse is a system that stores integrated, historical and aggregated information from multiple data sources that can change in the course of time. Business requirements of a data warehouse software also change quite often. Changes in data sources or business requirements can invalidate existing schemata of a data warehouse, data extraction, transformation and loading (ETL) processes and reports defined on the created data warehouse schema.

In the context of data warehouse development, four evolution problems can be defined, namely, maintenance, adjustment, adaptation and maintenance after adaptation. **Maintenance** means updating a data warehouse in accordance with changes in source data. **Adjustment** is updating of data warehouse content, according to changes in the data warehouse schema caused by modifications of the data warehouse business requirements. **Adaptation** means data warehouse alteration in accordance with changes in schemata of data sources. **Maintenance after adaptation** is data warehouse data updating according to new schemata of a data warehouse.

Data warehouse evolution problems arise in many projects, including a project, where the author of the thesis participated. In the project the e-learning data mart was developed. During the maintenance of the e-learning data mart, the developers had to deal with all four types of data warehouse evolution problems. To address the problems caused by changes, data warehouse and ETL processes had to be adapted manually. In addition, different versions of the data warehouse schema were created and it became necessary to provide users with the possibilities to generate and execute reports on several versions of the schema. The previously used online analytical processing (OLAP) tool was unable to provide these possibilities. Currently, data warehouse maintenance is supported in both commercial and open source data warehouse software. Commercial systems for data warehouse development and OLAP tools do not support neither automated processing of changes in data sources and data warehouse schemata, nor simultaneous operation with multiple versions of schemata.

In recent years, data warehouse maintenance and evolution were included in the topics of interest in several conferences on research in data warehousing and database fields, such as DOLAP, DaWaK, CAISE, ADBIS. Data warehouse evolution problems are addressed also in scientific publications [HMV99], [MV00], [Bla00] [KPR04] [Bel02] [RKZ00], [BMB +03], [SNP05], [GLR +06], [MW04], [WB05]. In 2009 the first workshop dedicated to data warehouse evolution problems within the framework of ADBIS conference. In the workshop the multiversion approach was discussed in particular, which assures that the approach chosen by the author of the thesis is perspective. The author of the thesis also gave the presentation of her approach in the workshop.

Goal of the Thesis

The goal of the doctoral thesis is to develop a concept of a multiversion data warehouse and reports definition and presentation tool, capable of executing reports on several data warehouse schema versions.
To achieve the goal, the following tasks have to be resolved:

1) Investigate the problems associated with data warehouse development and existing approaches for solving these problems.

2) Offer a data warehouse architecture to deal with the data warehouse evolution problems, which includes solutions for the automatic detection of changes in structure of data sources, for modification of data warehouse schemata and for automated adaptation of ETL processes in accordance with changes occurred.

3) Develop the support for a multiversion data warehouse, i.e. solution for physical storage of schema versions in a relational database, formal multiversion data warehouse model and metadata, which describe data warehouse schema versions, as well as a physical structure of data warehouse and semantics of data warehouse data.

4) Develop algorithms for processing of changes in data warehouse schemata.

5) Create multiversion data warehouse reporting tool, which allows to define, execute and adapt reports after different types of changes in data warehouse schemata based on metadata.

Main Results of the Thesis

The main results of the doctoral thesis are the following:

• The data warehouse evolution framework was offered. It is able to detect changes in data sources automatically, adapt a data warehouse schema and ETL processes in accordance with decisions of an administrator, handle direct changes to a data warehouse schema, support versions of a data warehouse schema in the development environment and in reports in the user environment.

• The metadata of data warehouse schema versions, which are based on the recognized metadata standard, and storage solution of schema versions in relational database were offered and developed.

• The reports definition and execution tool was developed. The tool is based on the proposed model of metadata of data warehouse schema versions and allows a user to create ad-hoc reports on multiple data warehouse schema versions easily and quickly and to analyze data in reports, displaying them in tables or graphs and performing OLAP operations.

• An SQL query construction algorithm was developed. Basing on user-defined reports specification in metadata, the algorithm generates queries to one or more versions of a data warehouse schema in accordance with a user's choice.

• The developed approach was tested for the schema versions of the e-learning data mart. The approach was evaluated and compared with the traditional approach, which uses a commercial reporting tool that does not support schema versions. The completeness of the data warehouse schema metadata was evaluated by the set of test cases that cover different changes in data warehouse schemata.

Publication of the Results of the Thesis and Presentations at Conferences

The results of the research conducted within the framework of the doctoral thesis are reflected in the publications, which are listed in the end of this summary:
• publications about e-learning data mart and adaptation of data warehouse schemata according to changes in data sources [SN05], [SN06],
• publication about the data warehouse evolution framework [Sol07a],
• publications about data warehouse schema versions, metadata and changes [Sol08a], [Sol08b],
• publications about reports definition and execution tool [Sol08c], [Sol09a], [SN10].

The results of the doctoral thesis were presented at the following conferences:
• ADBIS in years 2005 [SN05] and 2009 [Sol09a],
• Baltic DB&IS in years 2006 [SN06] and 2008 [Sol08c],
• SYRCoDIS in 2007 [Sol07a],
• ISD in years 2008 [Sol08b] and 2010 [SN10];
• [Sol07b] 65th LU scientific conference, Information Technology Section in 2007,
• [Sol07b] 66th LU scientific conference, Information Technology Section in 2008,
• [Sol07b] 67th LU scientific conference, Information Technology Section in 2009,
• [Sol07b] 68th LU scientific conference, Information Technology Section in 2010.

**Structure of the Thesis**

In the 1st chapter of the thesis, the research scope of the doctoral thesis is defined and the motivation of the choice of the topic is stated.

In the 2nd chapter of the thesis, basic elements of a data warehouse are defined, the representations of a data warehouse and OLAP operations, which are often used for analysis of data warehouse data, are described.

In the 3rd chapter of the thesis, the data warehouse evolution problems are classified. These problems are maintenance, adjustment, adaptation and maintenance after adaptation. Possible solutions of these problems are described for data warehouses, which are defined as sets of materialized views and as multidimensional models.

In the 4th chapter of the thesis, the e-learning data mart is described. The data model and history of the data mart are presented. The evolution problems, which occurred during the development of the e-learning data mart, and solutions used for them are illustrated. The author’s publication related to this chapter is [SN05].

In the 5th chapter of the thesis, the data warehouse framework, which was designed to address different types of data warehouse evolution problems, is proposed. Components of the framework and their operating principles are described. The author’s publications related to this chapter are [Sol07a], [SN06].

In the 6th chapter of the thesis, the formal model of multiversion data warehouse is defined and the description of the metadata models of data warehouse schemata and reports is given. These metadata were developed corresponding to the formal data warehouse model. The described metadata are the part of the data warehouse evolution framework and the operation of several components of the framework is directly based on these metadata. The reporting tool, which was developed within the framework of the thesis, defines and executes data warehouse reports, based
on the data warehouse schema and reporting metadata. The metadata models are provided that describe the data warehouse scheme at the physical, logical and semantic levels, as well as reports on data warehouse schemata. The author’s publications related to this chapter are [Sol08a] [Sol08b], [SN10].

In the 7th chapter of the thesis, procedures that are executed in cases of various changes are formally described. These procedures make changes in instance of the formal model of data warehouse defined in the 6th chapter. The changes, which should be made in the data warehouse schema metadata according to the proposed approach in case of data warehouse evolution, are presented. As a result of these changes a new data warehouse schema version is created. The author’s publications related to this chapter are [Sol08a] [Sol08b].

In the 8th chapter of the thesis, the query generation algorithm is presented. The algorithm builds an ad-hoc SQL queries on one or more versions of a data warehouse schema basing on the schema metadata and report specification. This algorithm is used in the query definition and display tool. The author’s publication related to this chapter is [Sol08c].

In the 9th chapter of the thesis, technical implementation details of the reporting tool are described, the requirements for reports definition and display, the tool architecture, technologies and tools used, and reports presentation metadata are given.

In the 10th chapter of the thesis, the evaluation of the proposed approach is described. The evaluation is based on the e-learning data mart project. The comparison is made of the proposed approach with the approach used in one of commercial reporting tools that does not support data warehouse schema versions. The changes in the database, actions that were conducted to reflect the schema versions of the e-learning data mart activity star schema in the metadata, the query generation process on two versions of the schema are described in more detail. The usage of the proposed approach in test cases from scientific papers on data warehouse evolution is described with the purpose to evaluate the correctness and completeness of the data warehouse schema metadata. The author’s publication related to this chapter is [Sol09a].

In the conclusions of the thesis, the main results, the contribution of the thesis and the conclusions about the developed approach are summarized. The possible future research directions and planned additions to the proposed approach are considered.

Main Elements of a Data Warehouse

The purpose of this chapter is to define a data warehouse schema and other concepts used in the thesis. Two data warehouse representations are described in the chapter. These are a multidimensional schema and a set of materialized views. The implementation of a data warehouse in relational database and online analytical processing operations are also discussed in the chapter.

Multidimensional Data Warehouse Schema

Typically, a multidimensional schema is used to model a data warehouse. The main object of a multidimensional schema is a measure, which usually contains numeric and additive data that describe certain facts [CD97], [MW04]. Each measure is dependent from several dimensions that characterize the context of the measure [CD97]. Several dimensions together clearly define a value of the measure. Each dimension is characterized by several attributes, which usually contain
descriptive and textual information. Dimensions contain hierarchies, which are used for data analysis. Hierarchies consist of ordered dimension levels, which in turn are made of dimension attributes [Ora03]. Hierarchy levels define the degree of aggregation. Each level in a hierarchy is connected with levels above and below it by classification relationship. Data values at the lowest level are aggregated in the data values at the highest level.

**Data Warehouse as a Set of Materialized Views**

In several references [AAS+97], [GM95], [ZGH+95], data warehouse is represented as a set of integrated materialized views on source systems. According to this approach, source systems are usually relational database systems independent of each other. Data warehouse users’ queries are executed on materialized views, without access to source databases. Very often data warehouse is represented as a set of select-project-join (SPJ) materialized views [QGM+96]. These are views that contain one projection, followed by one selection and a Cartesian product of base relations. In addition to SPJ views, data warehouse materialized views can also be defined with more complex structures, such as aggregation views, or a union or difference of several subqueries.

**Data warehouse data analysis - OLAP operations**

In order to support a multidimensional data analysis, OLAP operations are implemented in OLAP tools. The most popular OLAP operations [JLV+03], which are also supported in the solution proposed in the thesis, are roll-up, drill-down, slicing, selection, pivoting, ranking, definition of calculated attributes [CD97]. In addition to support of OLAP operations, there are some tools to help users build ad-hoc queries, which are constructed on data of the data warehouse schema, and the results of which are necessary for users to analyze. In graphical OLAP tools data warehouse data are presented as multidimensional data cubes, whose edges reflect dimensions, but subcubes contain measure values [HRU96].

**Data Warehouse Evolution Problems**

In the chapter the problems related to data warehouse development are discussed. These problems can invalidate existing data warehouse schemata, data updating processes and report on data warehouse schemata. The classification and description of different data warehouse evolution problems are given. Existing approaches are analysed for solving these problems in data warehouses, which are defined as a set of materialized views on data sources or as a multidimensional data models.

**Classification of Problems**

The usage of materialized views in the complex data warehouse architecture, where each of the components may change over time, can cause several problems, such as problems related to the view maintenance, adjustment and adaptation. Basing on the classification of the evolution problems of materialized views [RLN97], the problems that are caused by the development of a data warehouse represented as a multidimensional model are classified in the thesis. According to this classification, four data warehouse evolution problems are defined, namely, maintenance, adjustment, adaptation and maintenance after adaptation.
Data Warehouse Maintenance Problem

Data warehousing update process in accordance with changes in source data is called data warehouse maintenance. Data warehouse maintenance problem has been known and studied for a long time, and there are many methods to solve it both for materialized views and for data warehouses, which are built with multidimensional schemata.

Approaches to materialized view maintenance include not only solutions for updating materialized view data in accordance with changes in base relations, but also solutions that can determine, whether changes in data sources affect materialized views [SMR+98], [Wid95]. There are also solutions, which make a view self-maintainable [GM95], [QGM+96], [Huy97], which support concurrent updating of a materialized view and source relation data [ZGH+95], [ZGW98] [AAS+97], and solutions, which maintain temporal materialized views [Yan01]. Usually maintenance process of a materialized view consists of identification of data source changes, which affect the view, determination of the different data content in the view and in the data source, and execution of the view data updating operation.

Maintenance of a multidimensional data warehouse is usually performed by ETL processes, which are presented as workflow processes in some solutions [BFM99]. In the simplest situation, these processes can be implemented as loading procedures, which update the data in each data warehouse table. But in order to facilitate design and modification of ETL processes, some solutions [VGD99] utilize metadata that describe schemata of a data warehouse and data sources, as well as parts of ETL processes by predefined transformations. Maintenance approached for multidimensional schemata are also intended for solution of several specific problems, such as communicating with data sources and obtaining data from data sources, which is performed by maintenance mediators [Wie98] [BCV99] [GRV+98], [CMM01], as well as processing of unstructured, semi-structured data [CAW98] [BCV99], [CAM01], and data obtained from web sources, [AMM98], [ML01], [Zhu99], [KM00].

Data Warehouse Adjustment Problem

Business requirements of data warehouse users can change over time, so together with them data warehouse schemata can also change. Here comes an important issue of how to maintain a data warehouse, when its’ schemata are changing, is it possible to avoid a full data warehouse data recalculation. This problem is called a data warehouse adjustment.

In case of materialized views to solve the adjustment problem, several approaches are proposed in the literature [GMR95], [GMR+01], [MD96], [Moh97]. According to these approaches view definition is supplemented with additional structures. In case of the view redefinition, this additional information is used to calculate values of new or changed columns or rows of the view, not recalculating the view from scratch.

In case of a multidimensional data warehouse, in the literature [HMV99] [VMR02], [BFB08] [Ban07] [Bla00] [KPR04] there are proposed methods that define operations that change a data warehouse schema, and the consequences of these operations. Each method has advantages and disadvantages. For example, some methods [HMV99] [VMR+02] deal only with changes in the structure of dimensions and can not be used for adjustment of facts. Several methods can also be used in combination, to cover bigger range of data warehouse schema changes for different
representations of a data warehouse schema in relational database. For example, the paper [Ban07] does not support such changes, when a new measure is added to a fact or a measure is deleted from a fact. But in the method proposed in the paper [Bla00] special hierarchies, such as non-onto hierarchy, soft hierarchy, etc., are not supported. But in the paper [KPR04], in addition, the effect of various evolution operations on the snowflake data warehouse schema is discussed. Not all approaches consider the question of the implementation of evolution operations at the data level and how reports that were executed on the changed data warehouse schema have to be adapted. The solutions also do not support schema versions and repositories used in the solutions do not meet any standard.

Data Warehouse Adaptation and Maintenance after Adaptation

Regarding changes in data source schemata, two problems should be solved. This schema evolution should be propagated in a data warehouse and a data warehouse schema and ETL processes should be adapted, according to new data source schemata. These problems are called, respectively, data warehouse adaptation and maintenance after adaptation.

In case of materialized views, the problem of adaptation and maintenance after adaptation basically means how to rewrite the view definition query, which is affected by the data source changes, and how to update the view data after changes to avoid view recomputation. The adaptation problem of materialized views in data warehouses is mostly studied by a single group of authors [LNR97], [NR98], [CZR06]. These authors offer the evolvable view environment, which supports the adaptation of a view definition in response to changes in data sources, and further maintenance after adaptation.

For the adaptation of a multidimensional data warehouse solutions are offered in the literature [Mar00], [MP03]. These solutions use the information about ETL processes to adapt a data warehouse schema in accordance with source changes. These solutions support a number of possible changes that may occur in a data source. For each change, a process is defined, which must be carried out automatically or automatedly to propagate the change. But these solutions are more focused on data integration problems and do not take into account differences between elements of multidimensional schema. For example, the adaptation process may vary depending on whether it is necessary to adapt a dimension or a fact table.

Other approaches [CMM+03], [VMP03] adapt only ETL processes and do not alter a data warehouse schema. The solutions, which adapt ETL processes, are not complete, because in case of new data structures (for example, in case of creation of a new table or column in a table), they do not adapt the data warehouse, because ETL processes can continue to work without errors. But this new information could be useful for a data warehouse, and therefore it should be added to existing data warehouse schema.

Solutions for multidimensional data warehouse adaptation deal only with the problems connected with data warehouse adaptation after changes in data sources, but do not consider changes, which have to be performed in the existing data warehouse data.
Data Warehouse Versioning Approach

Several papers [EKM02], [MW04], [WB05], [SNP05], [GLR+06] consider database schema versioning approach to solving schema evolution problems. Schema versioning implies that a database system provides the access to all data, both retrospectively and prospectively, through user definable version interfaces [Rod95]. These approaches practically provide that after changes in a data warehouse schema, versions of the schema are created. These versions reflect business requirements, which were valid during the given time period, and this period is called version validity. In the literature, data warehouse schema versioning approaches are used only for solving the evolution problems in multidimensional data warehouses.

Multiversion data warehouse approach is quite a new solution of data warehouse evolution problems. Data warehouse versioning approach can be used for both data warehouse adaptation and adjustment problems. In case of usage of multiple schema versions not only data history, but also history of schema changes is retained, so the versioning approach seems more appropriate to support evolution. But the creation and maintenance of multiple schemata is more costly and leads to additional tasks, such as the determination of necessity of version creation; choice of data warehouse elements, for which versions are created; physical storage and maintenance of multiple versions; determination of operations resulting in a creation of new schema versions; transformation of existing data warehouse data from one version to another; and the main task of execution of queries to multiversion data warehouses. The approaches proposed in the literature solve only some of these tasks. These approaches do not solve such problems as, for example, storage of schemata of multiversion data warehouse in a relational database; definition of user reports, construction of queries, and adaptation of existing reports to reflect the existence of multiple versions. Although the majority of approaches are based on metadata that describe data warehouse versions, none of these approaches uses metadata that correspond to any metadata standard, such as Common Warehouse Metamodel (CWM) [OMG02], which makes it difficult to integrate solutions with existing data warehouse systems and to compare them with other authors' approaches.

Conclusions

In general, all solutions described in this chapter deal only with one type of evolution problems, for example, maintenance, adjustment or adaptation. But a data warehouse need to conform to any problems, i.e. all evolution problems must be solved together. Simultaneous utilization of several proposed solutions is difficult, because they use different data structures and methods.

In several solutions a data warehouse is considered as a set of materialized views on source data. But often in the development of a data warehouse, the main effort is devoted to implement complex ETL processes. Solutions with materialized views ignore ETL processes and do not offer any adaptation techniques for ETL processes. This means that solutions with materialized views can be used only for a very limited set of simple data warehouses. Most of the papers about maintenance, adjustment and adaptation of materialized views were published rather long time ago (before 2000). This shows that currently this approach is not topical.

Solutions, which adapt a data warehouse schema in accordance with changes in business requirements and data sources, can cause a loss of history and do not reflect the data warehouse
evolution, because in these solutions there is always only one actual version of a data warehouse schema. But data warehouses must maintain historically correct information, so it is correct to use multiversion data warehouse approach. Therefore, multiversion approach was chosen in the thesis to support data warehouse evolution. This approach is more recent in comparison with other approaches described in the literature in the area of data warehousing, because the first publication on the usage of data warehouse schema versions appeared only in 2002. Existing solutions of data warehouse evolution problems with multiversion approach have some outstanding challenges that were addressed in the thesis.

Development of E-learning Data Mart

The purpose of this chapter is to describe the e-learning data mart, which is used in examples in the following chapters. The development challenges of the e-learning data mart motivated the author of the thesis to turn to the data warehouse evolution problems. The e-learning data mart has been gradually developed in the University of Latvia since 2004. During the usage of this data mart, a number of changes both in data sources and in business requirements have happened, i.e. the data mart has developed. The evolution of the e-learning data mart illustrates the problems described in the previous chapters.

Description of E-learning Data Mart

The e-learning data mart was developed as a part of the data warehouse of the University of Latvia within the framework of the e-university project. The e-university project existed from 2002 until 2008 and one of the goals of the e-university project was to provide support for the learning process with the proper course management system and course materials. Until the end of the year 2007 WebCT system was used to organize courses, and since 2008 Moodle system has been used. Courses that are organized in such systems and are taught through the internet are called e-courses.

The main goal of the e-learning data mart is to provide an analysis of the influence of the e-learning environment on various university processes. The main objectives of the analysis were: evaluation of the usage of the system; estimation of the activities of instructors; assessment of the usage of tools; usage of courses by students. Primary users of the e-learning data mart are the e-learning management staff. For the implementation of the data mart Oracle RDBMS was selected. Oracle Discoverer reporting tool was used for reports on the schema of the e-learning data mart. Oracle Discoverer allows to define and execute ad-hoc reports on data warehouse schemata.

The model of the e-learning data mart [SN05] consists of three star schemata, which are united by the common dimensions:

E-course structure star schema incorporates data about the course structure, i.e., the number and file size of tools in a course. A tool is a single component of the course management system, such as mail, calendar, discussions, etc.

Usage star schema contains number of registered and active students. Registered students are those students, who have registered for the e-course in the e-learning environment as users with the role ‘student’. Active students are registered students, who have connected to the course at least once.
Activity star schema includes information about student activities in the e-learning environment during course acquisition: hits count, data amount and time period, which records the duration of a user’s usage of a course tool.

Development of E-learning Data Mart

During the development of the e-learning data mart, the evolution problems related to maintenance, evolution of business requirements and changes in data sources have been solved. These changes affected ETL processes and the schemata of the data warehouse. To solve these problems the data warehouse administrator had to manually modify the data warehouse design, to create a new data warehouse schema version and ETL processes.

As a result of these modifications, two activity star schema versions (Fig. 1 and 2) were obtained, where the first version is no longer updated, but contains historical data about the usage of the course management system since September 2004. But the second version of the activity star schema, which was implemented in the beginning of 2005, is current and is updated with ETL processes.

Changes that occurred in the data mart affected not only the schema of data mart and ETL processes, but also reports on the data mart schemata. The used reporting tool Oracle Discoverer does not support simultaneous work with multiple versions of a data warehouse schema. All these
Data mart development problems required a lot of time and resources, so it was decided to automate the processing of changes and to develop a data warehouse reporting tool that is able to display data from multiple versions of schemata in a single report.

Solutions for development and maintenance problems of the e-learning data mart were described and published in [SN05].

Data Warehouse Evolution Framework

This chapter describes the data warehouse evolution framework that is capable of handling the above-mentioned data warehouse evolution problems: both source changes and changes in data warehouse schemata caused by the development of business requirements. The reports definition and execution tool proposed in this thesis is a component of the data warehouse evolution framework. The ideas of the framework were implemented in the prototype, which is described in this chapter, but the reports definition and execution tool has been further developed within the framework of this thesis. The operation of the developed tool is largely based on the data warehouse metadata, therefore this chapter gives an insight into how the metadata necessary for the tool are created and changed.

Components of the Data Warehouse Evolution Framework

To support the above-mentioned data warehouse evolution problems: maintenance, adjustment and adaptation, the data warehouse framework, which is depicted in Fig. 3, is proposed.

The framework is composed of the development environment and user environment. In the development environment the data warehouse metadata repository and other components, which will be described later, are located and ETL processes and change processing are conducted. In the user environment reports on one or several data warehouse schema versions are defined and executed by data warehouse users and developers.

Fig. 3. Data warehouse evolution framework

The basic element of the development environment is the adaptation component that processes changes in source schemata, such as table renaming, deletion, addition, attribute renaming, deletion, addition and change of a data type; identifies the potential changes in a data warehouse and possible new versions; adapts a data warehouse schema or creates a new version according to the choice of the data warehouse administrator; creates the necessary version metadata;
and adjusts ETL processes. The metadata management tool that incorporates the graphical user interface client tool is used by administrator or developer to design a data warehouse schema and specify ETL processes. ETL processes are generated by the metadata deployment tool that uses the mapping metadata from the metadata repository. The data warehouse loader executes generated ETL scripts. The data transportation procedure transfers data warehouse data and schema metadata, which are necessary for reports, from the development environment into the user environment into the reporting metadata repository.

To implement the functionality of the data warehouse evolution framework, data from the metadata repository are used. Metadata repository consists of three parts, which are designed for storage of different types of metadata:

Data warehouse repository stores schema metadata, which are data about data warehouse schema versions that are necessary for definition and execution of reports, including links between different versions. Data warehouse repository contains mapping metadata, which define the logics of ETL processes. The mapping metadata describe data source objects (such as tables and columns in a relational database or files) that are necessary for ETL processes. In addition the data warehouse repository contains also adaptation metadata, where adaptation component stores information used to adapt a data warehouse after changes in data sources. Schema and mapping metadata of the data warehouse repository are maintained by the metadata management tool.

Data warehouse change repository stores potential solutions for adaptation and version creation for a data warehouse schema, from which an administrator chooses the most appropriate solutions, which are further implemented.

Source change repository accumulates changes in the structure of data sources, which are tracked by agents that are incorporated into data warehouse data sources.

In addition to the data warehouse, in the user environment there is also the reporting metadata repository that contains the data warehouse schema metadata, which are transferred from the data warehouse repository of the development environment, and the reporting metadata, which describe data warehouse schema elements and are created by a data warehouse developer by the reports definition tool.

Data warehouse users work with the reports execution tool that allows to define ad-hoc queries, display reports as tables and graphs and analyze report data using hierarchies and other OLAP features. Using links between data warehouse schema versions in the metadata repository, the reporting tool can run queries on data, which correspond to multiple data warehouse schema versions or one version depending on a choice of a user. In case of many versions, a user can choose which version is used to display results of a query if it is possible.

In the user environment an access mechanism is also implemented. It is the metadata that define which reports can be used by a particular user and which data a particular user can access. These metadata are used by the reports execution tool. The access mechanism is set by the developer by the reports definition and execution tools.
Operation of the Data Warehouse Evolution Framework

If in case of changes in business requirements a data warehouse schema is changed by the administrator then all changes are conducted by the metadata management tool, which allows to create a new data warehouse version or alter an old version. The mapping metadata of ETL processes are adapted according to a new data warehouse schema version.

Source changes are processed before the execution of ETL processes run by the data warehouse administrator in the development environments. Initially the adaptation component analyzes changes in the source change repository and detects changes that affect a data warehouse schema and ETL processes. The adaptation component processes these changes using data from the adaptation metadata of the data warehouse repository. For each change, the adaptation component generates solutions that create a new data warehouse version or adapt the data warehouse schema and ETL processes. The information about the solutions is stored in the data warehouse change repository. The administrator analyses the proposed solutions and chooses the most suitable solutions. If the administrator decides to create a new data warehouse version, the adaptation component changes the schema metadata in the data warehouse repository to reflect the new data warehouse version. If the administrator chooses to conduct adaptation without creation of a new version, the adaptation component does not need to create a new version in the metadata and it just updates the last schema version in the metadata.

Then the adaptation component adjusts the specification of ETL processes in the mapping metadata. The adaptation component also creates new data structures to store data of the new data warehouse schema version or adapts the existing data warehouse schema directly in the database.

The adaptation component generates the metadata deployment script, which is executed by the metadata deployment tool that generates the executable ETL process script. The ETL process is executed by the data warehouse loader. Database changes are made to match the data warehouse database implementation with the schema metadata.

At the end of adaptation or adjustment a metadata deployment script is generated either by the adaptation component in case of source changes or by the metadata management tool if data warehouse business requirements are modified. The deployment script is executed by the metadata deployment tool that generates the executable ETL process script, which is then executed by the data warehouse loader. The data warehouse administrator is notified about the successful adaptation and adjustment and he can then run the changed ETL processes.

When a data warehouse schema is changed, the reporting and schema metadata, which describe a data warehouse, in the reporting metadata repository in the user environment become inadequate to a changed schema or new schema version and reports on a data warehouse can not run any more. Therefore, during the new data transfer from the development environment into the user environment, the data transportation procedure updates also the reporting and schema metadata to reflect the new data warehouse schema version. In the proposed approach in case of changes in a data warehouse schema and in case of creation of a new schema version, user-defined reports continue to execute thanks to the query generation algorithm of the reports execution tool, which is one of the major innovations of this thesis.
Conclusions

Data warehouse evolution framework is able to handle not only changes in data sources, but also direct changes in a data warehouse scheme. Besides, in the evolution framework data warehouse versions are supported both in the development environment as well as in reports in the user environment. On the other literature are the data warehouse evolution for solutions of the proposed architecture distinguishes it is that it supports the right number of evolutionary problems, not just a subset. The proposed framework differs from other solutions of data warehouse evolution problems presented in the literature by the fact that it supports many evolution problems at once, not just one problem.

The fundamental part of the framework, which was developed in this thesis, is the reports definition and execution tool and the metadata that describe data warehouse schemata and support multiple versions of these schemata, as well as define transformations for elements from one schema version to another. The framework described in this chapter defines how changes in a data warehouse schema can occur and how these changes should be implemented. The components of the framework were implemented in the prototype, where standard features of Oracle Warehouse Builder [Ora03a] were used for the implementation of several framework components, while other components related to support of versioning were programmed from scratch.

The data warehouse evolution framework proposed in this chapter was described and published in the paper [So107a]. The solution for data warehouse adaptation after changes in data sources with the adaptation component and metadata, as well as the data warehouse adaptation framework, were described and published in the paper [SN06].

The Formal Model for Multiversion Data Warehouse and Metadata

In this chapter the formal model for multiversion data warehouse is given. The formal model describes a data warehouse at the logical, physical and semantic levels. Common Warehouse Metamodel (CWM) [OMG02], [PCT+03] was used as the basis of a multidimensional data warehouse formal model and the metadata. CWM is a data warehouse metadata standard developed by the Object Management Group to facilitate the exchange of metadata between data warehousing applications. CWM contains several packages, which describe various aspects of a data warehouse. CWM has been extended to support data warehouse evolution.

In CWM metamodel data warehouse elements are modelled as classes, but in the formal model described in this chapter in addition to class diagrams formal definitions of multidimensional data warehouse schema versions, implementation of versions and used terms and concepts are also given to formally describe changes in a data warehouse schema. CWM class diagrams were supplemented by additional data structures to support data warehouse schema versions at the logical and semantic levels, and by user rights defined at the physical level. The original reporting metamodel was also developed.

Logical Level of the Data Warehouse Formal Model

The logical level of the formal model describes data warehouse schema versions and it does not depend on any specific implementation. The logical level model is based on the OLAP package of CWM, which describes data necessary for OLAP tools. The logical level model describes the
multidimensional data warehouse schema, which consists of a fact table, dimensions, dimension hierarchies, their levels (Fig. 4).

![Fig. 4. Logical level formal model](image)

To reflect multiple versions of a data warehouse schema, CWM was extended by two classes: *SchemaVersion* and *VersionTransformation* (Fig. 5). Each schema version (class *SchemaVersion*) has a validity period defined by attributes *ValidFrom* and *ValidTill*. Validity periods of all versions do not overlap and are recorded one by one. Attribute *IsValid* indicates whether this version is now valid. Each version, except for the first one, has a link to a previous version, from which it was created. This link is implemented as an association between instances of the class *SchemaVersion*.

![Fig. 5. Schema versions at the logical level](image)

According to the proposed approach it is possible to create versions for the following schema elements: measures, attributes, fact tables, dimensions, associations between fact tables and dimensions, hierarchies, levels and associations between levels and hierarchies. Schema elements, which exist in a schema version or which does not exist in a version but can be obtained from other existing schema elements, are linked to the class *SchemaVersion* through the class *VersionTransformation*. *SchemaVersion* contains several *VersionTransformations*, which correspond to each schema element that exists in a schema or is calculated from other elements. Versions of schema elements are supported as follows:

If any schema element exists in a schema version then for it an association *toElement* is created. This association connects the element with an instance of the class *VersionTransformation*, where the attribute *Conversion* equals to NULL.

If a schema element does not exist in a schema version, but it can be obtained by a transformation function from other schema elements that exist in that version, then this element is also connected to an instance of the class *VersionTransformation* by an association *toElement*, but in the instance of *VersionTransformation* the attribute *Conversion* contains this transformation function and elements that exist in a version and are used in the transformation function are connected to the same instance of the class *VersionTransformation* by an association *fromElements*. 
In this case also an additional association with the name `fromVersion` is created between `VersionTransformation` and such `SchemaVersion`, where the calculated schema element exists.

This class diagram allows to create multiple version transformations for each changed element, i.e. the same schema element can be connected to multiple schema versions through different version transformations. The proposed class diagram allows to transform data of elements from any version to any other version.

Below the formal definition of the logical level of a data warehouse schema is given. A schema is a tuple \( DWS=\langle F,D,FDA,V,MAP \rangle \), where \( F \) is a set of facts, \( D \) is a set of dimensions, \( FDA \) is a set of fact dimension associations, \( V \) is a set of schema versions, where \( v_n \in V \) is a currently valid schema version, and \( MAP \) is a set of attribute and measure mappings.

Fact dimension association \( fda \in FDA \) is a tuple \( \langle \text{dim}, \text{fact} \rangle \), which associates a dimension \( \text{dim} \in D \) with a fact \( \text{fact} \in F \).

Dimension \( \text{dim} \in D \) is a tuple \( \langle \text{ATT}, \text{LEV}, H, \text{AL}, \text{HL} \rangle \), where \( \text{ATT} \) is a set of attributes, \( \text{LEV} \) is a set of levels, \( H \) is a set of hierarchies, \( \text{AL} \) is a set of attribute level associations and \( \text{HL} \) is a set of hierarchy level associations. Attribute level association \( \text{al} \in \text{AL} \) is a tuple \( \langle \text{att}, \text{lev} \rangle \), which associates an attribute \( \text{att} \in \text{ATT} \) with a level \( \text{lev} \in \text{LEV} \). Hierarchy level association \( \text{hl} \in \text{HL} \) is a tuple \( \langle \text{lev}, \text{h} \rangle \), which associates a level \( \text{lev} \in \text{LEV} \) with a hierarchy \( \text{h} \in H \).

Fact \( \text{fact} \in F \) is a tuple \( \langle \text{MS}, \text{MA} \rangle \), where \( \text{MS} \) is a set of measures and \( \text{MA} \) is a set of acceptable measure aggregations. Acceptable measure aggregation \( \text{ma} \in \text{MA} \) is a tuple \( \langle \text{ms}, \text{dim}, \text{agg} \rangle \), which defines that an aggregation function \( \text{agg} \in \{ \text{SUM}, \text{AVG}, \text{MIN}, \text{MAX}, \text{COUNT}, \text{DETAIL} \} \) can be applied to a measure \( \text{ms} \) in regard to a dimension \( \text{dim} \).

Schema version \( v \in V \) is a tuple \( \langle \tau_{\text{from}}, \tau_{\text{to}}, VT \rangle \), where \( \tau_{\text{from}} \) and \( \tau_{\text{to}} \) are times, which define the validity period of the schema version, \( VT \) is a set of version transformations. Version transformation \( vt \in VT \) is a tuple \( \langle \text{el}, \text{conv}, E \rangle \), which states that an element \( \text{el} \in ELEM \ (ELEM=F \cup D \cup FDA \cup ATT \cup LEV \cup H \cup AL \cup HL \cup MS) \) is obtained from a set of elements \( E \subseteq ELEM \) by a conversion function \( \text{conv} \). If an element \( \text{el} \) is obtained exists in the schema version (i.e. corresponds to any element of the physical level) then \( vt=\langle \text{el}, \text{null}, \emptyset \rangle \). Such version transformation is called a direct version transformation. Through version transformations all elements that exist in a version or are obtained from other existing elements are connected to a schema version.

Physical Level of the Data Warehouse Formal Model

The physical level of the formal model (Fig. 6) describes relational database schema of a data warehouse and mapping of a multidimensional schema to relational database data structures. The physical level model is based on the Relational package of CWM, which describes the relational data. The logical and physical levels are connected by objects defined in the Transformation package of CWM.

Physical level does not include versioning information because physically in the database there is only one schema version and versioning is implemented at the logical level. According to the proposed approach, it is also possible to support multiple physical versions of tables (i.e. in case...
of changes different tables that store versions of schema elements are physically created), but in change handling procedures described in the thesis it is considered that physically in the database all modified columns and data of these columns are stored in a single table to simplify the creation, modification and execution of ETL processes. The solution with one physical schema version has been chosen, because the more there are tables in the database, the more procedures of ETL processes need to be developed. Besides, if the same data are stored in different tables, then it is more difficult to ensure referential integrity.

The class diagram at the physical level does not differ significantly from the CWM packages Relational and Transformation. However, it was supplemented by classes that describe the privileges of users for data and schema elements of a data warehouse. A data warehouse can store sensitive data, which cannot be accessed by all users. Besides, the access to some data can be allowed to one group of users and prohibited to other. User privileges allow not only to limit the access to data in accordance with security requirements, but also to filter out information, which users are not interested in.

**Fig. 6. Physical level formal model**

Below the formal definition of the physical level of data warehouse relational database schema is given.

A **relational database schema** is a tuple \( DBS = < CS > \), where \( CS \) is a set of column sets. A column set reflects a database table, view or SQL query.

**Column set** \( cs \in CS \) is a tuple \( < C, KEY, DOMA > \), where \( C \) is a set of columns, \( KEY \) is a set of keys and \( DOMA \) is a set of column domain associations.

**Key** \( key \in KEY \) is a tuple \( < type, KC, pk > \), where \( type \in \{ 'PRIMARY', 'FOREIGN' \} \), \( KC \subseteq C \) is a set of key columns. If \( type = 'FOREIGN' \) (foreign key), then \( pk \in KEY \) is a primary key, \( KEY \) is a set of keys of any column set. If \( type = 'PRIMARY' \) (primary key), then \( pk \) is null.

**Column domain association** \( doma \in DOMA \) is a tuple \( < c, dom > \), which associates domain \( dom \) with column \( c \in C \).

A **mapping** of measures and attributes of a data warehouse map \( \in MAP \) is a tuple \( < el, mapfun, EC > \), where \( el \in ATT \cup MS \) is a mapped element, mapfunc is a function, which specifies how the element \( el \) is obtained from a set of columns \( EC \subseteq C \). In the thesis the simpler case is considered, when attributes and measures are obtained directly from database columns by the mapping function “copy”, i.e. \( map = < el, 'copy', \{ c \} > \), where \( c \in C \).
A set of user privileges is a tuple \( UP = \langle PRIV \rangle \), where \( PRIV \) is a set of privileges. Privilege \( priv \in PRIV \) is a tuple \( \langle usr, cs, c, pcond \rangle \), where \( usr \) is a data warehouse user, \( cs \in CS \) is a column set, \( c \in C \) is a column and \( pcond \) is a condition, which must be fulfilled so that a user can access data in a column \( c \). If a user has access to all columns of a column set \( cs \), then \( c \) is null.

**Data Warehouse Reporting Formal Model**

The reporting formal model describes a structure of user-defined reports. CWM contains the package Information Visualization, which describes how data warehouse schema elements are rendered, for example, in reports, graphs, websites, etc. CWM model is not sufficient, because it is very general and does not describe precisely report elements. Therefore, a new reporting model (Fig. 7) was design for the reports definition and execution tool. The reporting formal model can describe both table-type and matrix-type reports. Besides, it is possible to define report conditions, totals, parameters, user-defined joins between tables.

![Fig. 7. Reporting formal model](image)

In the thesis both the detailed description of the reporting formal model and the formal definition of data warehouse reports were given.

**Semantic Level of the Data Warehouse Formal Model**

Data warehouse users must understand the semantics of data that appear in reports from business perspective. They also must be able to analyse these data using all necessary features, including OLAP operations drill-down and roll-up, using hierarchies. Besides, it is desirable that users can modify or construct reports themselves from elements, which are familiar to them, so that reports creation becomes transparent. For these purposes, it is necessary to describe each element of the data warehouse model in business language. This description could also be used by users to express their requirements for information and changes in requirements making the understanding between users and developers of data warehouse clearer. The description of data warehouse elements in business language is stored in semantic metadata (Fig. 8).

The semantic level is based on the CWM package Business Nomenclature. The CWM was supplemented with the class TermVersion, which is connected to the particular schema version. The term version reflects the meaning of some data warehouse schema element, which was valid during the particular period of time defined by attributes ValidFrom and ValidTill of the corresponding schema version. Different versions of terms may be created in case of changes of the data
warehouse business requirements. For example, one term ‘Usage time’ can be defined by two term versions ‘Total time in hours’ and ‘Total time in seconds’, which correspond to different measures.

**Fig. 8.** Semantic level formal model

Below the formal definition of the semantic level is given.

**Business domain** is a tuple $BD=<TAX,GLOS,TGMAP,CTMAP>$, where $TAX$ is a set of taxonomies, $GLOS$ is a set of glossaries, $TGMAP$ is a set of associations between taxonomies and glossaries and $CTMAP$ is a set of associations between concepts and terms.

Taxonomy $tax \in TAX$ consists of a set of concepts: $tax=<CONC>$.

Glossary $glos \in GLOS$ is a tuple $<TERM,PTMAP,CONTMAP,SEMAP>$, where $TERM$ is a set of terms, $PTMAP$ is a set of associations between preferred terms and synonyms, $CONTMAP$ is a set of associations between related terms and $SEMAP$ is a set of item definitions.

**Term** $term \in TERM$ is a tuple $<termdef,TV>$, where $termdef$ is a term definition (word or phrase, which describes a term) and $TV$ is a set of term versions. **Term version** $tv \in TV$ is a tuple $<tvdef,E,v>$, where $tvdef$ is a definition of a term version (word or phrase, which describes a term version), $E \subseteq FDA \cup ATT \cup LEV \cup H \cup AL \cup HL \cup MS$ is a set of data warehouse schema elements, which are defined by a term version, and $v \in V$ is a data warehouse schema version, where the term version was valid. Schema elements and schema versions are the classes from the logical level formal model.

**Association between a preferred term and a synonym** $ptmap \in PTMAP$ is a tuple $<prefterm,synterm>$, where $prefterm \in TERM$ is a term, which is preferred and $synterm \in TERM$ is a synonym of a preferred term. **Association between related terms** $contmap \in CONTMAP$ is a tuple $<term_1, term_2>$, which associates related terms $term_1 \in TERM$ and $term_2 \in TERM$.

**Item definition** $semap \in SEMAP$ is a tuple $<term,se>$, which states that a report item $se \in SE$ corresponds to a term $term \in TERM$. Item is the class from the reporting formal model.

**Association between a taxonomy and a glossary** $tgmap \in TGMAP$ is a tuple $<tax,glos>$, which associates a taxonomy $tax \in TAX$ and a glossary $glos \in GLOS$, which are related.

**Association between a concept and a term** $ctmap \in CTPMAP$ is a tuple $<conc,term>$, which states that a term $term \in TERM$ is used to describe a concept $conc \in CONC$.

**Data Warehouse Schema Metadata**

In addition to the formalization of a multiversion data warehouse, multiversion data warehouse schema metadata and reporting metadata are described in the thesis. These metadata in general are the implementation of the formal model. These metadata are used in the data warehouse evolution framework. The semantic, logical and physical level metadata correspond to the schema
metadata of the data warehouse repository. The reporting metadata correspond to the reporting repository metadata of the data warehouse evolution framework. In the thesis the approach is considered to storage of multiple data warehouse schema versions in relational database. In relational databases there are no built-in means to describe in detail the multidimensional model. Therefore, for this purpose the schema metadata are used. These metadata describe the nature of the data warehouse multidimensional schema (fact tables, dimensions, hierarchies, etc.), schema versions and storage solution for the multiversion data warehouse in relational database. Reporting metadata are closely connected with the schema metadata and they describe reports definition from data warehouse schema elements.

Conclusions
The main result of this chapter is the formal model that describes a data warehouse schema at the logical, physical and semantic levels, as well as reports on a data warehouse. As the basis for the development of the proposed formal model, CWM was used. The packages of CWM were extended to make it possible to define data warehouse schema versions, term versions and reports.

In addition to the class diagram of each level, as well as reporting formal model, a formal definition of data warehouse was also given. The formal definition further in the thesis is used to formally describe the implementation process of data warehouse schema changes.

The thesis also describes the data warehouse schema metadata and reporting metadata, which are used in the data warehouse evolution framework in the data warehouse and reporting repositories. The metadata were constructed in accordance with the described formal model.

The formal model of a multiversion data warehouse and the model of the multiversion data warehouse metadata repository were described and published in the papers [Sol08a] [Sol08b], [SN10].

Data Warehouse Schema Changes
In the thesis the changes that are supported in the proposed approach and that may occur in a data warehouse schema as a result of evolution are described. The list of changes was obtained from changes that are mentioned in the literature and changes that took place in the e-learning data mart. These changes can be initiated by changing data warehouse business requirements, as well as by changes in data sources. These changes influence a data storage schema, ETL processes and reports. Besides, as a result of these changes, a new data warehouse schema version is created.

The description of each change in the thesis consists of a formal procedure, which accordingly adjusts an instance of the data warehouse formal model, and the description of the process to be conducted to propagate changes in the proposed data warehouse schema metadata and a physical data warehouse schema in relational database.

General Description of Changes
Three types of changes are distinguished according to their operation: physical, logical and semantic changes. Physical changes operate with database objects and an instance of the data warehouse physical level model. Logical changes modify mainly an instance of the data warehouse
logical level model. Semantic changes can adjust an instance of the data warehouse semantic level model, as well as an instance of the logical level model.

The following changes are supported in the approach proposed in the thesis:

- Physical changes: addition of a new attribute to a dimension, change of a data type of a dimension attribute, deletion of a dimension attribute, renaming of a dimension attribute, creation of a new dimension, deletion of a dimension, renaming of a dimension, addition of a new measure to a fact table, change of a data type of a measure, deletion of a measure, renaming of a measure, creation of a new fact table, deletion of a fact table, renaming of a fact table;

- Logical changes: connection of a dimension to a fact table, disconnection of a dimension from a fact table, creation of a new dimension hierarchy, deletion of a dimension hierarchy, creation of a new hierarchy level, deletion of a level from a hierarchy, deletion of a level from a dimension, addition of a level to a hierarchy, connection of an attribute to a level, disconnection of an attribute from a level;

- Semantic changes: change of a meaning of an attribute, change of a meaning of a measure.

Formally, as a result of every change a new data warehouse schema version \( v' \) is constructed from the previous version \( v_\nu \). Each change procedure is described as a sequence of steps, which modify the data warehouse schema \( DWS=<F,D,FDA,V,MAP> \), the database schema \( DBS=<T,VW,Q,CS> \) and the business domain \( BD=<TAX,GLOS,TGMAP,CTMAP> \). The output of the change procedure is a new data warehouse schema \( DWS'=<F',D',FDA',V',MAP'> \) at the logical level, a new database schema \( DBS'=<T',VW',Q',CS'> \) at the physical level and a new business domain \( BD'=<TAX',GLOS',TGMAP',CTMAP'> \) at the semantic level. If the schema is not changed at the physical or semantic level, then the description of the physical or semantic level is omitted. If as a result of a change a schema element is deleted, then a data warehouse administrator determines the possibility to create a version transformation, which calculates the missing schema element from other schema elements of the schema version. The examples of the formal procedures of physical, logical and semantic changes are given below (in the thesis the description of all change procedures is given).

The following common functions are used in the formal description of change procedures:

\( v' = \text{duplicate}(v) \) is a function, which creates a version \( v' \), which contains the same elements as a version \( v \). This function is used to create a new data warehouse schema version from the previous version. It is made as follows:

- If \( v = <\tau_{\text{from}}, \tau_{\text{to}}, VT> \), then \( v' = <\tau_{\text{now}}, \text{null}, VT'> \), where \( \tau_{\text{now}} \) is the current time and \( VT' \) is a set of version transformations, where for each version transformation \( vt=<el, conv, E> \in VT \) a corresponding version transformation is constructed \( vt'=<el, conv, E> \in VT' \). This means that in the new version \( v' \) the same version transformations, which existed in the previous version \( v' \), are created.

- If \( \tau_{\text{to}} \) is \( \text{null} \) (i.e. the version was valid until the new version creation) then the version \( v \) is updated \( v = <\tau_{\text{from}}, \tau_{\text{now}}-\text{I}, VT> \) (i.e. the validity period end is recorded for the previous version).

\( VT = \text{vtrans}(\text{elem}) \) is a recursive function, which returns the set of version transformations \( \{vt_1, ..., vt_I\} \), which require an input element \( \text{elem} \in ELEM \). This function is used to obtain all version
transformations, in which the particular element was used. The version transformations returned by this function are the following:

- \( v_t = \langle \text{elem}, \text{null}, \text{null} \rangle \) or
- \( v_t = \langle \text{el}, \text{conv}, E \rangle \) and \( \text{elem} \in E \subseteq \text{ELEM} \), where \( \text{el} \in \text{ELEM} \), \( \text{conv} \) is any conversion function. In the last case the function \( \text{vtrans}(\text{elem}) \) returns also \( \text{vtrans}(\text{el}) \).

The following notation is used in the formal description of change procedures:

- \( \text{new} \) means the creation of a new instance of a class from the physical, logical or semantic level model. For example, \( \text{c} = \text{new Column} \) means that a new instance of the class Column is created.
- \( \text{Set} \) is the assignation of an existing instance of a class from the physical, logical or semantic level model. For example, \( \text{Set} \ \text{cs} = \langle \text{C}, \text{KEY}, \text{DOMA} \rangle \) means that a tuple \( \langle \text{C}, \text{KEY}, \text{DOMA} \rangle \) is assigned to the column set \( \text{cs} \).
- \( \text{return} \) is a result of a procedure operation.
- \( \text{For each} \ ... \ \text{do} \) denotes operations executed in a loop. For example, \( \text{For each} \ \text{att}_j \in \{\text{att}_1, ..., \text{att}_i\} \ \text{do} \ \text{c}_j = \text{new Column} \) means that for each attribute from the set \( \{\text{att}_1, ..., \text{att}_i\} \) an operation, which creates a new column, is executed.

Example of a Physical Change – Addition of a New Attribute to a Dimension

**State before the change:** A new attribute \( \text{att} \) is added to the dimension \( \text{dim} = \langle \text{ATT}, \text{LEV}, \text{H}, \text{AL}, \text{HL} \rangle \). The column set \( \text{cs} = \langle \text{C}, \text{KEY}, \text{DOMA} \rangle \) corresponds to the dimension \( \text{dim} \). Data warehouse schema elements are described by terms from the glossary \( \text{glos} \in \text{GLOS}, \text{glos} = \langle \text{TERM}, \text{PTMAP}, \text{CONTMAP}, \text{SEMAP} \rangle \), which is associated with the taxonomy \( \text{tax} \in \text{TAX} \). Both the glossary \( \text{glos} \) and taxonomy \( \text{tax} \) belong to the business domain \( \text{BD} \).

Procedure for changes at the physical level: A new column \( \text{c}' \) is created in the column set \( \text{cs} \).
1. \( \text{c}' = \text{new Column} \);
2. \( \text{Set} \ \text{cs} = \langle \text{C} \cup \{ \text{c}' \}, \text{KEY}, \text{DOMA} \cup \{ \langle \text{c}', \text{dom}_{\text{att}} \rangle \} \rangle \), where \( \text{dom}_{\text{att}} \) is a domain, which corresponds to \( \text{att} \);
3. \( \text{return} \ \text{DBS}' = \langle \text{T}, \text{VW}, \text{Q}, \text{CS} \rangle \).

Procedure for changes at the logical level: A new attribute \( \text{att} \) is attached to the dimension \( \text{dim} \). A mapping with a function ‘copy’ is created to connect the column set \( \text{c}' \) at the physical level with \( \text{att} \). This means that the attribute \( \text{att} \) is obtained directly from the database column \( \text{c}' \). The version transformation \( \text{v}_{\text{u}} \) can be constructed for the attribute \( \text{att} \), if it can be calculated from other attributes.
1. \( \text{v}' = \text{duplicate}(\text{v}_{\text{u}}) = \langle \tau_{\text{now}}, \text{null}, \text{VT}' \rangle \);
2. \( \text{Set} \ \text{dim} = \langle \text{ATT} \cup \{ \text{att} \}, \text{LEV}, \text{H}, \text{AL}, \text{HL} \rangle \);
3. \( \text{Set} \ \text{v}' = \langle \tau_{\text{now}}, \text{null}, \text{VT}' \cup \{ \langle \text{att}, \text{null}, \text{Ø} \rangle \} \rangle \);
4. If \( \text{att} \) can be calculated from other attributes \( \{ \text{att}_i, ..., \text{att}_j \} \) of the dimension \( \text{dim} \) by the function \( \text{conv} \), then \( \text{Set} \ \text{v}_{\text{u}} = \langle \tau_{\text{now}}, \tau_{\text{now}}, \text{VT}' \cup \{ \langle \text{att}, \text{conv}, \{ \text{att}_i, ..., \text{att}_j \} \rangle \} \rangle \);
5. \( \text{return} \ \text{DWS}' = \langle \text{F}, \text{D}, \text{FDA}, \text{V} \cup \{ \text{v}' \}, \text{MAP} \cup \{ \langle \text{att}, \text{copy}', \{ \text{c}' \} \} \rangle \rangle \).

Procedure for changes at the semantic level: A new term \( \text{term} \) with one term version \( \text{tv} \) and term definition \( \text{termdef} \) is added to the glossary \( \text{glos} \). If no concept, which is described the term \( \text{term} \), exists in the business domain \( \text{BD} \), then the corresponding new concept \( \text{conc} \) is created in the
taxonomy $tax \in TAX$. The term $term$ is connected with the concept $conc$ by the association. If the corresponding concept $conc$ exists in the business domain $BD$ in some taxonomy $tax$, then it is connected with the term $term$ by the association.

1. $tv=$new TermVersion($<tvdef,\{att\},v'>$);
2. $term=$new Term($<termdef,\{tv\}>$);
3. Set $glos=<\text{TTERM} \cup \{term\}, \text{PTMAP}, \text{CONTMAP}, \text{SEMAP}>$;
4. If no concept, which is described by the term $term$, exists in the business domain $BD$, then
   - $conc=$new Concept;
   - $tax=<\text{CONC} \cup \{conc\}>$, where $tax$ is a taxonomy, which contains concepts that are semantically similar to the concept $conc$. The related taxonomy is determined by the data warehouse administrator.
5. $ctmap=$new AssociationBetweenConceptAndTerm($<conc,term>$), where $conc$ is a (new or existing) concept described by the term $term$;
6. return $BD'=<\text{TAX},GLOS,TGMAP,CTMAP \cup \{ctmap\}>$.

Example of a Logical Change – Disconnection of a Dimension from a Fact Table

State before the change: A dimension $dim=<\text{ATT},\text{LEV},H,AL,HL>$ is disconnected from a fact table $fact=<\{ms_1,...,ms_i\}>$, the measures $\{ms_1,...,ms_i\}$ are mapped to columns $\{c_1,...,c_i\}$.

Procedure for changes at the physical level: no changes.

Procedure for changes at the logical level: Version transformations are created for measures of the fact table $fact$ if it is possible. In other case the fact dimension association between $fact$ and $dim$ is removed from the version $v'$.

1. $v'=$duplicate($v_u$)=$<\tau_{now},null,VT'>$;
2. For each $ms_j \in \{ms_1,...,ms_i\}$ do $ms'_j=$new Measure;
3. Set $fact=<\{ms_1,...,ms_i,ms'_1,...,ms'_i\}>$;
4. For each $ms_j \in \{ms_1,...,ms_i\}$ do $map'_j=$new Mapping($<ms'_j,\text{\textquoteright kopija'},\{c_j\}>$);
5. If it is possible to transform (divide) values of measures $\{ms'_1,...,ms'_i\}$ with respect to dimension $dim$ by the function $conv$, then the following version transformations are constructed:
   - For each $ms_j \in \{ms_1,...,ms_i\}$ do $vt_j =$new VersionTransformation($<ms_j,conv,\{ms'_j\}>$);
   - Set $v'=<\tau_{now},null,VT' \cup \{vt_1,...,vt_i\}>$; 
     Otherwise: Set $v'=<\tau_{now},null,VT' \setminus \text{vtrans}(ms_1) \cup ... \cup \text{vtrans}(ms_i)) \cup \{<fda',null,\emptyset>\}>$;
6. If it is possible to transform (aggregate) values of measures $\{ms_1,...,ms_i\}$ with respect to dimension $dim$ by the function $conv'$, then the following are constructed:
   - For each $ms'_j \in \{ms'_1,...,ms'_i\}$ do $vt'_j =$new VersionTransformation($<ms'_j,conv,\{ms_j\}>$);
   - Set $v'=<\tau_{now},null,VT\cup \{vt'_1,...,vt'_i\}>$;
7. return DWS'=$<F,D,FDA \cup fda',V \cup \{v'\},\text{MAP} \cup \{map'_1,...,map'_i\}>$.

Procedure for changes at the semantic level: no changes.
Example of a Semantic Change – Change of a Meaning of an Attribute

State before the change: A meaning of an attribute $att$ of a dimension $dim$ was changed from $tvdef$ to $tvdef'$. The attribute is described by a version $tv$ of a term $term=<termdef,TV>$. The attribute is mapped to a column $c$.

Procedure for changes at the physical level: no changes.

Procedure for changes at the logical level: In a new version $v'$ a new attribute $att'$ is created. The attribute $att$ is deleted from the new schema version, but in all associations of this attribute with dimension levels the attribute $att$ is substituted with the new attribute $att'$.

1. $v'=duplicate(v_u)=<\tau_{now},null,VT'>$;
2. Set $dim=<ATT\cup\{att'\},LEV,H,AL,HL>$;
3. For each $al\in AL$ & $al=<att,lev>$ (where $lev$ is any level) do
   - Set $dim=<ATT,LEV,H,AL\cup\{<att',lev>\}\{al\},HL>$;
4. Set $dim=<ATT\setminus\{att\},LEV,H,AL,HL>$;
5. Set $v'=<\tau_{now},null,VT'\setminus\vtrans(att)\cup\{<att',null,\emptyset>\}>$;
6. return $DWS'=<F,D,FDA,V\cup\{v'\},MAP\cup\{<att','copy',{c}>\}>$.

Procedure for changes at the semantic level: New version $tv$ of the term $term$ is created and connected to the attribute $att'$.

1. $tv=new\ TermVersion(<tvdef',\{att'\},v'>)$;
2. Set $term=<termdef,TV\cup\{tv\}>$;
3. return $BD'=<TAX,GLOS,TGMAP,CTMAP>$.

Conclusions

In this chapter the change procedures are specified that should be executed to propagate changes, which occurred in a data warehouse schema, in the formal model and in the corresponding metadata. The described changes are supported in the metadata, which were developed in the thesis. In the description of changes the case was considered when each data warehouse dimension and fact table corresponds to exactly one physical table in a relational database and each fact table measure and dimension attribute corresponds to exactly one column in a table. This case was implemented with the function 'copy', which is used in mappings of measures and attributes. In the physical and logical level metadata it is also possible to describe the case, when attributes and measures are calculated from multiple columns of tables with some functions. These functions can be very different and it is not possible to define in general procedures for all changes supported in the proposed model for any arbitrary function. If a measure or attribute is derived from a table column by a function, then a change processing procedure has to be created differently for each specific situation.

The descriptions of the change processing procedures are given with the purpose to consider typical changes in a data warehouse schema and to verify the completeness of the developed metadata to determine, whether the description of results of all possible changes is supported in the metadata and whether it is possible to create versions for modified schema elements in the metadata. This is necessary so that the reports definition and execution tool developed in the thesis could correctly interpret changes.
Multiversion data warehouse change procedures were described and published in the papers [Sol08a], [Sol08b].

Query Generation on Data Warehouse and Usage of Reports

In the thesis an approach for building SQL queries using the above-described data warehouse schema and reporting metadata is proposed. The described query construction algorithm was implemented in the reports execution tool developed in the thesis. When a report is executed, queries are generated for one or more data warehouse schema versions according to the approach proposed in the thesis.

Query Generation Process

A user can define a report with the reports execution tool choosing terms or term versions from the semantic-level metadata. During the report definition, the reports execution tool generates automatically the reporting metadata according to the report created by a user. The reporting metadata stores the report definition. When a user executes a report by the reports execution tool, he is asked to enter values for parameters previously defined in the reporting metadata, when a report was defined. Then the query construction algorithm generates an SQL query basing on the data warehouse schema metadata, reporting metadata and entered parameter values, and executes this query. The structure of generated queries is given in Figure 9. SQL standard clauses are used in the query structure.

Fig. 9. Structure of a generated SQL query

Elements of the list of chosen items can be dimension attributes, fact table measures, aggregated measures, functions, whose arguments are dimension attributes, or functions, whose arguments are dimension attributes and fact table measures. The list of used column sets is a list of tables, views or subqueries used in a report. The list of joins is a formula of conditions that join tables, views or subqueries used in a report. The list of conditions is a formula of conditions that are connected with AND and OR logical operators. The list grouping of grouping items includes elements used to group results of an aggregate function. The list of grouping conditions is a formula of conditions with aggregate functions connected with AND and OR logical operators. The list of sort items consists of items chosen for a report, which are used to sort report data.

SQL query generation by the algorithm described in this chapter is conducted in several steps.

In the step of analysis of chosen items and determination of used column sets, items chosen by a user for a report during report definition are analysed. As a result, the list of chosen report items is constructed, which consists of column names and parameter values and functions that compute report items from columns and parameters. This list is used in the SELECT clause of the SQL query. Additionally, in this algorithm step the list of names of tables, views and subqueries, which is used in the FROM clause of the SQL query, is obtained. In this list one column set may appear several times with different aliases, if columns from several column sets that define the same
table, view or subquery from the physical-level metadata are used in a report. In this algorithm step, the list of sort items is also created. Chosen items are included in this list in a sequence corresponding to a sort index of each chosen item that is stored in the reporting metadata.

In the step of generation of joins, joins between tables, views or subqueries, which were included in the list of used column sets that was obtained in the previous algorithm step. As a result, the join formula in SQL notation, which is then used in the WHERE clause of the SQL query, is obtained.

In the step of generation of the list of conditions, conditions defined by a user are processed. As a result, the set of conditions is obtained, which is united into the condition formula in SQL notation, which is used in the WHERE clause of the SQL query.

In the step of grouping and construction of conditions for aggregates, chosen report items are analysed and items used in grouping are distinguished. Conditions, which contain aggregate functions, are also analysed. As a result, the following elements are obtained: the list of grouping items, which is used in the GROUP BY clause of the SQL query, and the formula of grouping conditions, which is used in the HAVING clause of the SQL query.

In the step of addition of restrictions of user rights, the query is supplemented by restrictions, which control user rights for used columns and column sets.

In the step of simplification and optimization of the query, the SQL query is simplified to speed up the execution of the query. To facilitate report data analysis using OLAP operations (roll-up, drill-down and slicing), the algorithm supplements the generated SQL query by SQL extensions ROLLUP and GROUPING SETS, which compute data, obtained by different possible OLAP operations, basing on hierarchies defined in the data warehouse schema metadata and locations of report items.

In the step of version analysis, items chosen for the report are analysed and data warehouse schema versions, which were valid during the time period of the report, are obtained. If one version is obtained then data returned by the SQL query generated in the previous steps are presented altogether in the report. If more then one version, which was valid during the time period of the report, remains, then a user is prompted to select one of the data presentation options that are determined for the particular report by building the special relationship matrix, which defines relationships between versions and schema elements. Depending on the existence of different schema elements in schema versions, the following options of report presentation are available: (a) present report data in accordance with one particular version; (b) present elements from different versions in one report; and (c) present data in multiple reports separately for each version. Further transformation of the query generated in the previous steps depends on the report data presentation option. A separate query is constructed for each schema version. This query contains the version validity restriction, schema elements, which exist in the version, and calculated schema elements, which does not exist in the schema version, but can be obtained by version transformations. As a result of this algorithm step, the SQL query is obtained that consists of the union of queries constructed for each data warehouse schema version.
Report Data Analysis

Data warehouse reports should support high interactivity with a user including OLAP operations: roll-up, drill-down, etc. One of the possibilities to analyze report data is to use hierarchies defined in the logical metadata. According to the proposed metamodel, different versions of hierarchies, levels and associations between attributes and hierarchy levels can be created. When a user runs a report, the query construction algorithm identifies all hierarchies and their structure that exist in the particular data warehouse version, which is selected by a user to present report data. If a user chooses to present elements from different versions in one report, then only hierarchies, which exist in all versions, are available as well as hierarchies that consist of attributes, which either exist in the versions or are obtained by version transformations from existing attributes.

In the data warehouse evolution framework reports term versions are used to separate different meanings of the same schema element. If any schema element has multiple term versions, when a user runs a report, which includes this schema element, he is informed that for the same schema element two term versions exist. The user has to choose the preferred term version and then the appropriate schema element version is included in the report.

Conclusions

In this chapter the query construction algorithm is stated that builds SQL queries to one or more data warehouse schema versions. The proposed algorithm is one of the key innovations of this thesis. The algorithm consists of 7 steps and is based on the report definitions in the metadata and on the data warehouse schema metadata. This algorithm allows users to choose different data presentation options, if a report is run on several data warehouse schema versions. The proposed algorithm constructs a query corresponding to the SQL standard, which means that the algorithm may be implemented in any relational database that supports the SQL standard. The described algorithm was implemented in the reports definition and execution tool that supports multiversion data warehouses.

The reports definition and query construction process for multiple data warehouse schema versions using the data warehouse metadata was described and published in a paper [Sol08c].

Technical Implementation of the Reporting Tool

The theoretical ideas of the thesis were validated by using them in the implementation of the reports definition and execution tool. In the thesis, there are described solutions of various challenges related to the technical implementation of the reports definition and execution tool. The detailed definition of requirements for the reports definition and execution tool is given. The architecture and technologies used for the implementation are described. The design options for reports and their implementation are also addressed.

Architecture of the Reporting Tool

The architecture of the reporting tool consists of a server with a relational database for storing data warehouse data and metadata, data collection procedures, which manage the reporting metadata and data warehouse schema metadata, reports definition and execution tools located on a web server. The relational database management system Oracle was used in the tool
implementation in the approbation project. Data collection procedures were implemented as PL/SQL procedures. Tomcat web server was used for the deployment of the reporting tool. Reports definition and execution tools were developed as Java servlets, which generate HTML code that can be used in browsers without additional software installation. The open source reporting engine JasperReports was used to display reports graphically.

The reporting tool consists of the *reports execution tool* and the *reports definition tool*. With the reports execution tool, users define and execute reports on one or more data warehouse schema versions. With the report definition tool, the developer manages the data warehouse metadata and defines user rights for report data and for reports. The prototype of the reports definition tool was implemented. It allows to manage the data warehouse schema metadata. The functionality of this tool was implemented (by the programmer, who is not the author of the thesis) according to the design developed by the author of the thesis. The prototype of the reports execution tool was also implemented. It allows to define and execute reports on multiple data warehouse schema versions. The implementation of the reports execution tool was conducted by the author of the thesis.

**Interface**

In the reports execution tool reports are defined by developers and users, who have privileges to create or modify reports. When a user runs a report by the reports execution tool, the tool generates the request for data in XML format and sends it to the data collection procedures.

![Fig. 10. Report execution – version selection](image)

Using the reporting metadata and mappings between the physical and logical levels of the data warehouse schema metadata, the data collection procedures reformulate the request for data into the SQL query according to the query construction algorithm and the option for report data presentation, which is chosen by the user. The generated SQL query is executed in the database, and the data are returned to the reports execution tool, which processes the data and prepares the report.
Users have the ability to analyze data in reports by several OLAP operations, create graphs, export reports in MS Excel or PDF format. The example of the interface of the reporting tool displaying version selection form is shown in Figure 10.

The following solutions from the results of the thesis were implemented in the reports definition and execution tools: support of data warehouse schema versions in the metadata, selection of schema versions in the interface of the reports execution tool, the algorithm for construction of SQL queries on one or more data warehouse schema versions. The reports execution tool was successfully used in the approbation project.

**Evaluation of the Proposed Approach**

In the thesis the benefits and limitations of the proposed were assessed. The proposed approach was compared with a traditional data warehouse solution. As an illustration, the data mart, which was developed within the framework of the approbation project, and its evolution problems were used. The second method used in the evaluation was testing of the proposed approach examples from other authors’ scientific papers on data warehouse evolution.

**Comparison of the Proposed Approach with the Solution without Schema Versions**

To evaluate the proposed approach, the e-learning data mart was used. In the approbation project, two versions of the activity star schema, which are depicted in figures 1 and 2, were used. In connection with the evolution of the activity star schema, the following issues were addressed: (1) processing of the occurred schema changes; (2) storage of two schema versions in relational database; (3) description of the star schema in the metadata; (4) presentation of the star schema data in reports.

Within the framework of the approbation project, reports, which were executed in the e-learning data mart, were used to construct queries. To illustrate the approach proposed in the thesis, the report showing the number of times students used the specific tool in their course (number of hits). This report was developed in both versions of the activity star schema.

**Solution without Scheme Versions**

The solution to the data warehouse evolution problems, which was used in a real data warehouse project before the development of the proposed approach, was considered in the thesis. The first version of the activity star schema was created in accordance with the design of the data mart and physically in the database it was composed of the tables shown in Figure 1. When the business requirements of the data mart changed, then physically in the data base new tables were created: Activity-new, Person, Session, Grade, Study Program, Time-new. The columns of these tables correspond to the new version of the star schema, which is shown in Figure 2. ETL processes were modified and since the emergence of the second schema version, they were executed for the second schema version, i.e. the first star schema version was not updated.

Since the reporting tool Oracle Discoverer, which was previously used in the project, does not allow to execute reports on multiple schema versions, the reports in this tool were created separately for the first and the second version of the star schema. If it was necessary to obtain data for the period, which covers the validity periods of both schema versions, then users had to execute two reports and data had to be summed up manually.
Usage of the Proposed Approach in the Approbation Project

To illustrate the proposed approach, the evolution problems of the activity star schema were solved according to the proposed approach. It was described, how the multiversion data warehouse was built within the approbation project and how queries, which obtain data necessary for reports, were generated by the tool proposed in the thesis. The appropriate physical and logical change processing procedures were executed. Further in the chapter the result of execution of these procedures in the approbation project is reflected.

Physical Level

According to the proposed approach, physically in the relational database all schema versions are stored in one physical structure. The initial version of the activity star schema was created according to the schema, which is shown in Figure 1. To create a new star schema version, in the database new tables were created: Person, Session, Grade and Study Program. The columns of these tables correspond to the new schema version (Fig. 2). In each of these tables new fictive records with data “All together” were created with primary key identifiers respectively P, S, G, and SP. The table Activity was supplemented by additional columns: Person_ID, Session_ID, Grade_ID and Study_Program_ID, which were filled with the identifiers, P, S, G and SP for all data existing in the table. The table Time was supplemented with the columns Time and Hour. The new columns of the existing data rows of the dimension Time were filled with default data, such as '00:00:00'. The table Tool was extended by new columns Required, Type, SubCategory and Category to further create the appropriate hierarchies. In this case, it is possible to fill the new columns manually, since the information about the classification of all the tools is available and the number of records in the table is relatively small.

Logical Level

In the logical-level metadata two schema versions are created, according to the schema, which is shown in Figure 1 and Figure 2. The elements, which remain unchanged in both versions, are connected to both versions by version transformations without conversion functions.

New dimensions with appropriate hierarchies were also created and connected only to the second star schema version by version transformations without conversion. New measures were created: Time, Hits_count_2_version, Data_amount_2_version.

<table>
<thead>
<tr>
<th>Table 1. Version transformations</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromVersion</td>
</tr>
<tr>
<td>V_2</td>
</tr>
<tr>
<td>V_2</td>
</tr>
<tr>
<td>V_1</td>
</tr>
<tr>
<td>V_1</td>
</tr>
<tr>
<td>V_1</td>
</tr>
<tr>
<td>V_1</td>
</tr>
</tbody>
</table>

Version transformations shown in table 1 are generated for the modified measures. The version transformations of the measures are generated only in one direction from the second version to the first version, since it is not possible to distribute the aggregated measure data to match the second schema version. Version transformations are also created for the new attributes of the Tool dimension, because their values for the existing data were filled at the physical level.
Semantic Level

At the semantic level new term versions are created for the measures *Hits count* and *Data amount* of the fact table *Activity* because the granularity of the measures changed, which also means that semantically the measures represent different information in the two schema versions. The term versions, which existed for the modified measures, and which were created for the new measures, are given in Table 2.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Term</th>
<th>Term version</th>
<th>Schema version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hits_count</td>
<td>Number of hits</td>
<td>Total number of hits by roles</td>
<td>V₁</td>
</tr>
<tr>
<td>Hits_count_2_version</td>
<td>Number of hits</td>
<td>Number of hits of each user</td>
<td>V₂</td>
</tr>
<tr>
<td>Data_amount</td>
<td>Amount of transferred data</td>
<td>Total amount of data transferred by users by roles</td>
<td>V₁</td>
</tr>
<tr>
<td>Data_amount_2_version</td>
<td>Amount of transferred data</td>
<td>Amount of data transferred by each user</td>
<td>V₂</td>
</tr>
</tbody>
</table>

Report Definition and Execution

When all necessary schema metadata and data structures in the database are created, the approbation illustration report is defined in the reporting metadata, i.e. the appropriate reporting metadata are created for the report.

When a user runs the report, he or she is prompted to enter parameter values. For the approbation illustration report the parameters are course name and dates, which define the time period of the report. Depending on the entered date parameters, the time period of the report is obtained, which spans both activity star schema versions. In the thesis two cases of the example report are considered. One case is when measure data are presented for each tool (the level *Name* of the tool category hierarchy) and the second case is when measure data are presented for each subcategory (the level *SubCategory* of the tool category hierarchy).

```sql
SELECT DISTINCT TOOL.NAME, TIME.DATE1, TIME.MONTH, TIME.YEAR, TIME.TERM, TIME.STUDY_YEAR,
TOOL.SUBCATEGORY, TOOLCATEGORY, SUM(ACTIVITY.HITS_COUNT)
FROM COURSE, TOOL, TIME, ACTIVITY, ROLE
WHERE (ROLE.ID=ACTIVITY.ROLE_ID AND COURSE.ID=ACTIVITY.COURSE_ID AND TOOL.ID=ACTIVITY.TOOL_ID AND
TIME.ID=ACTIVITY.TIME_ID) AND (TIME.DATE1 >= TO_DATE('01.09.2004','dd.mm.yyyy') AND
TIME.DATE1 <= TO_DATE('01.09.2005','dd.mm.yyyy')) AND
TIME.DATE1 BETWEEN TO_DATE('01.09.2004','dd.mm.yyyy') AND TO_DATE('131.01.2005','dd.mm.yyyy')
GROUP BY ROLLUP(TIME.YEAR, TIME.MONTH, TIME.DATE),
ROLLUP(TIME.STUDY_YEAR, TIME.TERM, TIME.DATE)
UNION
SELECT DISTINCT TOOL.NAME, TIME.DATE1, TIME.MONTH, TIME.YEAR, TIME.TERM, TIME.STUDY_YEAR,
TOOL.SUBCATEGORY, TOOLCATEGORY, SUM(ACTIVITY.HITS_COUNT)
FROM COURSE, TOOL, TIME, ROLE,
(SELECT TIME_ID, ROLE_ID, COURSE_ID, TOOL_ID, SUM(HITS_COUNT) HITS_COUNT, SUM(DATA_AMOUNT)
DATA_AMOUNT, TEACHING_ID FROM ACTIVITY
GROUP BY TIME_ID, ROLE_ID, COURSE_ID, TOOL_ID, TEACHING_ID)
ACTIVITY
WHERE (ROLE.ID=ACTIVITY.ROLE_ID AND COURSE.ID=ACTIVITY.COURSE_ID AND TOOL.ID=ACTIVITY.TOOL_ID AND
TIME.ID=ACTIVITY.TIME_ID) AND
(TIME.DATE1 => TO_DATE('01.09.2004','dd.mm.yyyy') AND
TIME.DATE1 <= TO_DATE('01.09.2005','dd.mm.yyyy')) AND
COURSE.NAME='Data Warehouses' AND ROLE.ROLE='Student') AND
TIME.DATE1 BETWEEN TO_DATE('01.09.2004','dd.mm.yyyy') AND SYSDATE
GROUP BY ROLLUP(TIME.YEAR, TIME.MONTH, TIME.DATE1),
ROLLUP(TIME.STUDY_YEAR, TIME.TERM, TIME.DATE1)
ORDER BY TOOL.NAME, TIME.DATE1, SUM(ACTIVITY.HITS_COUNT)
```

Fig. 11. Query on both schema versions
All seven steps of the SQL query construction algorithm were executed. As a result, it was determined that it is possible to present the example report of the first case according to the first schema version. In the second case the elements from both schema versions can be presented in one report. In both cases, data can be also presented in multiple reports separately for each schema version. As a result of the algorithm execution, the query shown in Figure 11 was generated for the first case report.

*Comparison of the Approaches*

The proposed approach was compared with the commercial reporting tool that does not support data warehouse schema versions. It was inferred that the key advantage of the proposed approach is new opportunities that were not implemented in the commercial tool so far. In the tool developed within the framework of the thesis, a user can analyze information available in several schema versions both in separate reports constructed on each version and altogether in one report, which is more convenient, but the previously used reporting tool requires users to merge manually information from separate reports.

*Test Cases of the Proposed Approach*

In the thesis four test cases of the proposed approach were examined. These test cases were obtained from the scientific publications on data warehouse evolution [HMV99], [MW04], [GLR+06]. Besides, two specially designed test cases were considered. These cases were used to test the usage of the proposed approach to solve data warehouse evolution problems. The results of these test cases verify that the approach proposed in the thesis supports typical situations that happen during the data warehouse evolution, the proposed data warehouse schema metadata completely describe data warehouse schema versions, and that change procedures described in the thesis are able to handle different types of changes in a data warehouse schema. The cases were chosen in such a way that the biggest possible range of changes supported in the approach proposed in the thesis is covered.

*Conclusions*

In this chapter the test cases from scientific papers about data warehouse evolution and the usage example, which is based on the e-learning data mart evolution, were described to check all data warehouse schema changes supported in the approach proposed in the thesis. In table 3 the executed physical, logical and semantic changes in the data warehouse schema are summarized. These changes were analysed in the test cases and in the example from the approbation project. For each change, in the table the test cases, in which the change was implemented, are marked with the symbol ✓. The considered test cases verify the completeness of the proposed metadata and the possibility to describe the supported changes in a data warehouse schema. As a result of the analysis of test cases it was concluded that the metadata proposed in the thesis are able to describe correctly all the necessary data warehouse schema elements, as well as multiple schema versions. The change processing procedures proposed in the thesis correctly adjust the existing implementation of the data warehouse schema in database and the physical, logical and semantic level metadata for the supported changes.
Table 3. Summary of the data warehouse schema change examples

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition of a new attribute to a dimension</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change of a data type of an attribute</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion of a dimension attribute</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Renaming of a dimension attribute</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation of a new dimension</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion of a dimension</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Renaming of a dimension</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addition of a new measure to a fact table</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change of a data type of a measure</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion of a measure</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Renaming of a measure</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation of a new fact table</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion of a fact table</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Renaming of a fact table</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical Changes</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection of a dimension to a fact table</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disconnection of a dimension from a fact table</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation of a new dimension hierarchy</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion of a dimension hierarchy</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation of a new hierarchy level</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion of a level from a hierarchy</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion of a level from a dimension</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addition of a level to a hierarchy</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection of an attribute to a level</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disconnection of an attribute from a level</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semantic Changes</td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change of a meaning of an attribute</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change of a meaning of a measure</td>
<td>![checkmark]</td>
<td>![checkmark]</td>
<td></td>
<td></td>
<td>![checkmark]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results of the comparison of the proposed approach to support a multiversion data warehouse and to generate reports on multiple versions with the traditional approach were published in the paper [Sol09a].

Conclusion

In the thesis the problems that occur during the data warehouse lifecycle were investigated. These problems are data warehouse maintenance in accordance with changes in data sources; adjustment, when a data warehouse schema is modified in accordance with new business requirements or because of other reasons; adaptation, when data source schema is modified; and maintenance after adaptation. For each data warehouse evolution problem, the possible solutions in the literature were considered. These solutions depend on the interpretation of the data warehouse concept. Two data warehouse interpretations were distinguished, which are a set of materialized
views on data source data and multidimensional database, where ETL processes extract data from data sources, transform them and load into a data warehouse.

It was concluded that solutions mentioned in the literature and analysed in the thesis can be applied only to solve one kind of evolution problems and can not be combined, but the author of the thesis believes that during the data warehouse development it is necessary to be able to solve as many problems as possible. A number of existing solutions do not retain the history of data warehouse development. To process correctly all the data warehouse evolution situations and not to lose data historical changes, it is desirable to use data warehouse schema versioning approach, which is also the newest research direction related to data warehouse evolution. The essence of this approach is that it supports simultaneous existence of multiple schema versions. Schema version is a ‘schema that reflects the business requirements during a given time interval, called its validity, that starts upon schema creation and extends until the next version is created’ [GLR+06]. In this area there are also several unsolved problems, such as the storage of the multiversion data warehouse schema in relational database, presentation of multiple versions in metadata, construction of queries and reports and adaptation of existing reports to reflect the existence of multiple versions.

The main contribution of this thesis is the approach to support multiversion data warehouses, which includes both the physical representation of schema versions in the database, and the possibility to model multiple versions of the logical schema of the data warehouse, as well as the algorithm for construction and execution of reports on multiple data warehouse schema versions.

One of main results of the thesis is the proposed data warehouse metamodel, which is created based on CWM metadata standard. The metadata consist of the physical, logical and semantic levels and specification of reports. At the physical level the relational database structure of the data warehouse schema is described. At the logical level multidimensional data warehouse elements and schema versions are reflected. At the semantic level data warehouse schema elements are described in the business language understandable for users. But in the reporting metadata the specification of user-defined reports on data warehouse schemata is stored. The metadata are based on the Common Warehouse Metamodel standard, which allows to integrate and exchange the metadata with other data warehousing applications that also support this standard. The standard metadata were extended with the metadata, which describe data warehouse schema versions and term versions. The main information about relationships between different versions is described by version transformations, which define how a changed or deleted data warehouse schema element can be obtained from other schema elements. These transformations are defined by the data warehouse administrator, when the new data warehouse schema version is created.

The metadata model proposed in the thesis was created in accordance with the formal model of the multiversion data warehouse, which describe a data warehouse at the logical, physical and semantic levels. Besides, the model describes reports on data warehouse schema versions.

In the thesis changes, which occur as a result of the data warehouse evolution, were also researched. The list of changes was obtained from changes mentioned in the literature sources and from changes, which were encountered in the practice of the author of the thesis developing and maintaining data warehouses. The changes supported in the proposed approach were described.
They can be caused by both the evolving data warehouse business requirements and changes in data sources. For each change, the formal procedure, which adjusts the instance of the data warehouse formal model accordingly, and the process, which must be conducted to propagate the changes in the proposed data warehouse schema metadata and in the physical data warehouse schema in relational database, were described.

One of the contributions of the thesis is the algorithm for construction of SQL queries on one or several data warehouse schema versions. This algorithm builds an SQL query according to the report definition in the reporting metadata and automatically converts the query, which was constructed for one of schema versions, into another query, which is aligned with other schema version, using version transformations defined in the logical level metadata. If there are no corresponding version transformations, the report results are shown separately for different schema versions.

To approbate the algorithm for construction of SQL queries and the metadata proposed in the thesis, the reports definition and execution tool was developed. This tool supports multiversion data warehouses. The main purpose of the data warehouse reporting tool is to provide the quick and convenient way for information analysis, i.e. data warehouse reports should support interaction with users, including OLAP operations rollup, drill-down, slicing. For this purpose, the algorithm supplements the generated query with the SQL extensions to pre-calculate data resulting from potential OLAP operations.

In the thesis the benefits and limitations of the proposed approach were evaluated, based on the example of the data mart developed in the approval project, which was compared with a traditional data warehouse solution. Version transformations and metadata were created according to the two schema versions of the data mart developed in the approbation project. In the reports execution tool developed within the framework of the thesis reports were constructed and executed. A number of advantages of the proposed approach were stated compared with the traditional data warehouse solution. According to the author’s opinion, the main advantage is the possibility to present data from multiple data warehouse schema versions in one report simply, quickly and automatically.

The approach proposed in the thesis was also evaluated, based on the test cases from other authors’ scientific publications on data warehouse evolution. All changes in the data warehouse schema supported in the proposed approach were tested with the test cases, and it was found out that the metadata proposed in the thesis are able to describe correctly all the necessary data warehouse schema elements, as well as several schema versions. It was concluded that the proposed change processing procedures adjust properly the existing implementation of the data warehouse schema in database, as well as adapt the physical, logical and semantic-level metadata to the supported changes.

In the thesis the data warehouse evolution framework was also proposed. It supports different types of evolution problems, including data warehouse adaptation and adjustment. The partial functionality of the framework was implemented in the prototype, where the commercial data warehouse development tool Oracle Warehouse Builder was used. The framework is composed of several components, which can be substituted by other components with similar functionality, if it
is necessary. Changing one framework component to another requires minimal or no changes in other components, so that the framework operates in accordance with requirements of a situation.

The evolution framework was created by extending the data warehouse adaptation framework, which was developed and used to adapt a data warehouse after changes in data source schemata. The developed metadata and reporting tool are the components of the evolution framework. The developed data warehouse schema evolution-oriented reporting tool is used to define reports at the University of Latvia.

It is planned to supplement the proposed approach in some directions, including definition of more sophisticated version transformations, further optimization of generated queries, extending the diversity of reports, personalization of reports.

The further research in the field of data warehousing is possible. One of the further directions of the research is a policy, which can define how version transformations are created automatically or automated. Regarding the semantic changes, further research is possible: how to determine this kind of changes in the data warehouse, or how to build versions of data.
References

Publications of the Author in International Scientific Editions and Conference Proceedings


Other Publications, Reports, Articles of the author


[Sol10] Solodovnikova D. ‘Query Definition and Presentation Tool for a Data Warehouse with Multiple Schema Versions’, 68th LU scientific conference, Information Technology Section, February 2010

Other Authors’ Publications


[CZR06] Chen S., Zhang X., Rundensteiner E.A. „A Compensation-Based Approach for View Maintenance in Distributed Environments“, IEEE Transactions on Knowledge and Data Engineering, Volume 18, Number 8, 2006, pp. 1068-1081.


[Ora03] Oracle Corporation, Oracle® Database Data Warehousing Guide 10g Release 1 (10.1), 2003, p. 806