Bogdan CZEJDO, Kenneth MESSA, Tadeusz MORZY,
Mikolaj MORZY, Janusz CZEJDO

DATA WAREHOUSES WITH DYNAMICALLY CHANGING SCHEMAS AND DATA SOURCES

Summary: Research in the data warehousing area focuses on design issues, data maintenance and query optimization. Recently new research areas appeared that are related to dynamicity of data sources. Dynamicity of data sources can be categorized into: data updates, schema and instance changes, and constraint modifications. Existing data warehouse systems manage data updates. However, they are unable to follow schema and instance changes and constraint modifications.
In this paper we analyze schema and instance changes caused by dynamically changing external data sources. We advocate the need to apply the external data source schema changes to a data warehouse and we present modeling issues involving star schema evolution and data warehouse versioning. Finally, we show query processing in presence of different data warehouse versions.

Key words: temporal data warehouses, advanced OLAP queries, schema change in data warehouse

1. INTRODUCTION

Integration of different, autonomous and heterogeneous external data sources (EDS) is crucial for today's businesses. Two basic approaches to consolidate distributed EDSs and provide integrated information to users [1,2,3] are the query-driven approach and the data warehousing approach. In the query-driven approach EDSs are integrated only at the logical level by merging all local schemas into a single global logical schema (no integration of EDS contents takes place, all data is stored only locally inside the EDSs). User queries executed against the global schema are translated by mediators into one or more queries executed against local EDSs. The mediators join the answers from the EDSs and return the final answer to the user. This approach has several advantages. No central database is required to physically integrate data from external data sources. There are no extract-transform-load processes to move data from EDSs to centralized data repository. There is no latency in data, all data is up-to-date.

The data warehousing approach is based on the centralized data repository. Data is extracted from EDSs, transformed (i.e. filtered, cleansed, enriched), and loaded into a centralized data repository called a data warehouse. As opposed to the query-driven approach, the data warehouse integrates at the global level both schemas and data. Integrated global schema consists of a collection of tables/views defined over export schemas of EDSs. Queries submitted to the data warehouse are executed locally, without accessing original EDSs, which considerably increases the query performance. It improves the availability of data and protects the data warehouse from the network delays or even the inaccessibility of external data. Local processing at EDSs is not affected by global applications running in the data warehouse. The data warehouse provides users with additional information such as aggregates, summaries or historical data. These are the main reasons why the data warehousing approach became such popular technology for numerous enterprises requiring high query performance and high data availability [4,5,6].

Until now research in data warehousing concentrated mainly on design issues, query performance and optimization, data maintenance, data refresh strategies and implementation issues. New file organizations have been proposed along with new access methods and new index structures (e.g. bitmap indexes). Most data warehouse models assumed, that data sources and data warehouse schema are static and that only the data changes. However, this

assumption doesn't hold in the real world applications. Changes occur frequently both in EDSs and in the data warehouse schema and instance. Most often those changes concern dimensions and dimension members (e.g., assigning a dimension member to another parent member, merging two dimension members, etc.). After such change queries touching data affected by the change begin to yield incorrect results. Contemporary data warehouses are unable to handle such changes, which hinders their functionality.

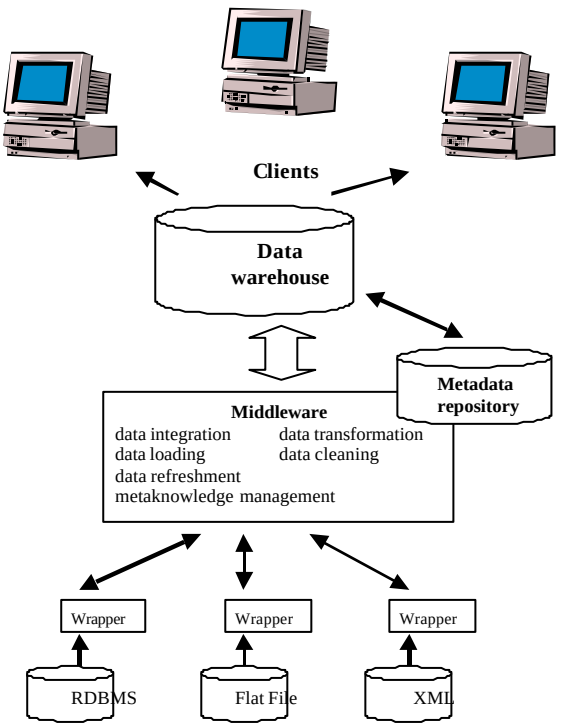
In this paper we discuss the data warehouse evolution triggered by changes in EDSs and data warehouse schema/instance. We present how EDS schema changes affect data warehouse schema, middleware level, and data warehouse content. We illustrate our discussion by an example of the TurboMachine Company data warehouse. We describe the effects of data warehouse evolution on various applications and we examine the necessary changes that have to be propagated to the applications in order to make them work.

This paper is organized as follows. In Section 2 we present the generic data warehouse architecture. In Section 3, we discuss the impact of EDSs changes on the data warehouse schema, the data warehouse content, and the evolution of the middleware under those changes. Section 4 presents an example of the data warehouse system for the TurboMachine Company. Section 5 illustrates the evolution of the sample data warehouse and its middleware under EDSs changes. The paper is concluded in Section 6 with a summary and a future work agenda.

2. DATA WAREHOUSE ARCHITECTURE

Picture 1 depicts the architecture of a data warehouse. The data warehouse is designed to integrate autonomous and heterogeneous external data sources. The component EDSs may vary from proprietary applications and legacy systems to modern relational, object or object-relational database systems. They may include flat files, spreadsheets, XML documents, news wires or multimedia contents. All EDSs usually differ in data models, require different user interfaces, and present different functionality.

Picture 1. Logical architecture of a data warehouse system



EDSs are connected to the data warehouse through wrappers which form a part of the middleware level. Wrappers extract data from EDSs, transform extracted data into the common data warehouse model, monitor changes to EDSs and propagate these changes to the data warehouse. The middleware is responsible for data cleansing and discovering inconsistencies in the source data, integration and transformation of data, data loading and refreshment, archiving, performing periodical backups, ensuring data quality, etc. Data warehouses are usually implemented in a multi-tier architecture, with the bottom tier being a relational database working as the data warehouse server. Internal tiers contain local data marts with copies of related fragments of the data warehouse. Top tiers contain only query and reporting tools, data mining tools, etc. Sometimes the top tier is a thin client containing only the web browser.

3. DYNAMIC EDSS

The data warehouse integrates autonomous and heterogeneous EDSs. Autonomy means that the EDSs preserve the autonomous and full control over its data. Heterogeneity means that the EDSs use different data models and different user and programming interfaces. Local autonomy and heterogeneity of EDSs mean that they were developed independently and are not aware of the integration issues. An important consequence of the autonomy of EDSs is that they may evolve in time independently and that they change their data and schemas without being controlled from the global data warehouse level.

The changes in the EDSs can be categorized into [7]:

- content changes such as insert/update/delete a tuple
- schema changes such as create/alter/drop a column or create/drop a table
- instance changes such as create/merge/split/drop a dimension
- constraint changes such as create/alter/drop an integrity constraint.

The existing data warehousing systems deal only with the first type of changes, namely, with the content changes. The content changes of EDSs are detected and propagated to the data warehouse in one of the following ways. For EDSs that are database systems the source can collect all updates that occurred during a specified interval of time and send all updates periodically to the data warehouse. All updates can be shipped either as a collection of data (data shipment) or as a collection of transactions (transaction shipment). The size of the refresh period depends on the data warehouse usage type, types of data source, the balance of work, etc. Changes shipped to the data warehouse can be applied either all at once (batch update) or incrementally (incremental update). The entire process of extracting, filtering, transforming, cleaning, transmitting, and loading updates into the data warehousing system is called data warehouse refreshment.

Most of the research in the data warehouse refreshment has focused on transactional incremental data warehouse refresh under content changes of EDSs. However, little research has examined the data warehouse refresh under schema changes and constraints modifications. In this paper we focus on the schema and instance changes propagation from the EDSs to the data warehouse. Due to the lack of space we omit the issue of propagating constraint modifications and the issue of the data warehouse maintenance under the evolving constraints.

Schema changes of EDSs are very common in the real world applications [8]. Integrating data from evolving EDSs raises new challenges in the maintenance and evolution of data warehouse systems. These challenges can be classified into four groups:

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

1.1. EDS SCHEMA CHANGES AND THE DATA WAREHOUSE SCHEMA

A data warehouse schema is usually defined as a set of materialized views over schemas of EDSs participating in the warehouse. Every change in the schema of an EDS invalidates the schema of the entire data warehouse. One possible solution is to isolate and hide those changes from the data warehouse. Isolation can be achieved by the modification of the middleware level, but this solution is limited by the time as further changes of the EDS schema would lead to even greater inconsistency between the EDSs schemas and the data warehouse global schema. Besides, hiding the changes from the data warehouse hinders the functionality of the data warehouse because as the result some important data may be inaccessible to data warehouse applications.

Another solution is to propagate all changes happening in the EDSs to the data warehouse. Changes should be incorporated into the metadata repository of the data warehouse. The metadata repository stores administrative data necessary to manage the data warehouse, such as: descriptions of external data sources, their contents and schemas, data warehouse schema, view and derived data definitions, dimensions, categories and hierarchies, descriptions of pre-defined queries and reports, data mart locations and contents, data partitions, data extraction, cleansing, and transformation rules, defaults, data refresh and purge rules, user profiles, user groups, etc.

1.2. EDS SCHEMA CHANGES AND THE MIDDLEWARE

Adopting middleware level to changes taking place in the EDSs is a difficult task. Many external data sources are not capable of signaling changes to the middleware level and the data warehouse (the so-called non-cooperating data sources). In such cases it is necessary to develop new solutions and algorithms to allow for dynamic adaptation of the middleware to the changes. To detect changes in the EDSs schemas the following techniques can be applied: analysis of the log files (if the source allows for logging changes or transactions), polling (issuing queries to detect changes in the data structure), custom made programs or screen scraping. For external sources that are capable of notifying middleware and data warehouse levels of changes in source schema (the so-called cooperating data sources) the most popular technique are triggers (small programs that execute automatically whenever a defining condition is fulfilled).

After the change has been discovered it must be propagated to the middleware level, which is the most difficult task. Changes applied to middleware wrappers must ensure that the wrappers preserve their capabilities to query external sources, translate queries and updates from the EDSs to the data warehouse and vice versa, perform data cleansing and data transformations, backup and archive source data, etc.

1.3. EDS SCHEMA CHANGE AND THE DATA WAREHOUSE CONTENT

All changes occurring in the EDSs schemas affect also the data warehouse content and must be properly handled. One possible solution is to create a separate instance of the data warehouse content for each instance of the data warehouse schema. This leads to the idea of a multi-versioning of the data warehouse and creates new challenges with respect to the

maintenance, evolution, and query processing in such multi-version data warehouse. As we will show later, this approach is feasible mainly for data warehouses that are subject to dimension and hierarchy changes. For EDSs schema changes related to creating, altering, or dropping attributes and tables more appropriate is to adjust the data warehouse content to the change. This requires additional knowledge that must be provided by a user, e.g., when creating new attribute in a given table the user must provide correct values of the newly created attribute for all tuples already stored in the data warehouse. Moreover, it also may be necessary to update some aggregates and summaries contained in the data warehouse. The adjustment of the data warehouse content can be performed either in batch or incremental mode.

The biggest problem is how to process a change in the EDS that is related to modifying the meaning of an attribute or an aggregate. This must be answered in order to provide the logical independence between the conceptual level of the data warehouse and the external level consisting of the front-end applications. In other words, the problem is how to accommodate changes in the data warehouse in such a way that the existing access and reporting tools are not affected by the changes and keep delivering correct results. To some extent, this problem can be solved by the proper data warehouse schema design and new data warehouse models that are aware of the changes and modifications in both data warehouse schema and the meaning of attributes, aggregates and summaries.

1.4. EDS Semantic Change and the Data Warehouse Content

The most popular architecture for data warehouses are multidimensional data cubes, where transaction data (called cells, fact data or measures) are described in terms of master data (also called dimension members) hierarchically organized in dimensions, where the facts of the upper levels are computed from the facts of the lower levels by some consolidation and aggregation functions. This multi-dimensional view provides long term data that can be analyzed along the time axis, whereas most EDSs only supply snapshots of data at one point of time. Available warehouse systems are therefore prepared to deal with changing measures, e.g., changing profit or turnover. Surprisingly, they are not able to deal with modifications in dimensions, although from the EDS point of view there is nothing fundamentally different between the measures data and the dimensions data stored in external data sources.

Consider the following example: data warehouse contains data about sales and production for a given company in different European countries. All data (for measures, dimensions and categories) is shipped from EDSs that are operational database systems managing production and sales in each country. One day the change appears: two dimension members "West Germany" and "East Germany" are merged together to form one dimension member "Germany". On the other side one dimension member "Czechoslovakia" is split into two distinct members "Czech Republic" and "Slovakia". Another possible change is the extension of the hierarchy member "European Union" with 10 new dimension members (namely the East European countries). The question is now: is it possible to get correct results for queries like “show a comparison of production in France and Germany during last 20 years” (at some point in time the numbers for "Germany" will increase due to the added production of the "East Germany") or "examine the sales in Czech Republic during last 20 years" (there was no dimension member "Czech Republic" 15 years ago). Without knowing the above changes such queries end up with incorrect results.

The problem here lies in the implicitly underlying assumption that the dimensions are orthogonal. Orthogonality with respect to the dimension time means that the other dimensions ought to be time-invariant. This silent assumption inhibits the proper treatment of changes in dimension data. One has to be aware that the dimensions data, i.e., the structure, the schema

and the instances of the dimensions of a data warehouse may change over time. This problem has recently attracted a lot of research attention [9,10,11]. One of the proposed solutions is multi-versioning of the data warehouse instances joined with temporal extensions to the existing data warehouse model. For example, the COMET model fulfills these assumptions and offers the following features:

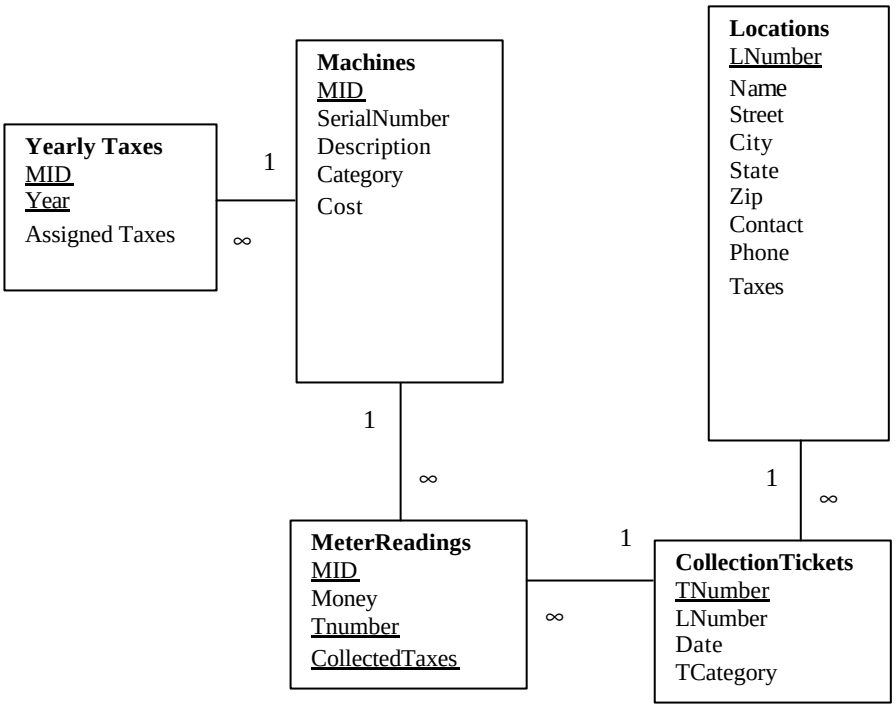
- representation of changes in data warehouses schema,
- identification of periods without a significant change,
- mapping and transformation functions between structure versions,
- processing queries reading data that is contained in several versions,

The detail description of the proposed solution is beyond the scope of this paper. Nevertheless, it is worth noticing that this approach allows the data warehouse content adjustment in response to changes occurring in the EDSs that modify the meaning of selected attributes, in particular changes that modify the content and structure of dimensions.

4. CREATION OF DATA WAREHOUSE FROM DIFFERENT DATA SOURCES

Let us consider a database for the TurboMachine Company that leases machines to other companies (the other companies are referred here as locations) and collects (usually once a week) a leasing fee that is based on a meter describing the extent of use of each machine. For each location the weekly (but can be more often) leasing fees are grouped into a collection ticket. The TurboMachine Company has many branches. The database for one of the branches includes four tables: **`Machines`**, **`Locations`**, **`CollectionTickets`**, and **`MeterReadings`** as shown in Picture 2.

Picture 2. The database for a branch of the TurboMachine Company

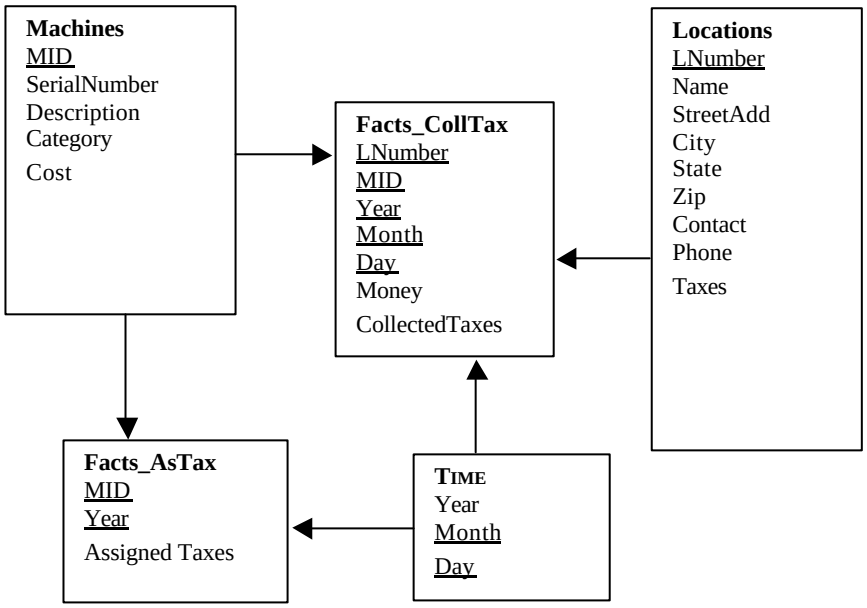


The role of each table is as follows. **`Machines`** contains and keeps records of the information for a specific machine. **`Locations`** contains the information about each company which has/had leased a machine. Table **`Yearly Taxes`** includes the important attribute ***`AssignedTaxes`*** that can change every year for each machine. **`MeterReadings`**

has a log of all collections made at a location for a particular date and for a specific machine. It includes an important attribute, ***CollectedTaxes***. Even though there is some guidance on how to compute this attribute for each collection, practically, it is often determined individually. Therefore it is necessary to have it as a regular (not derived) attribute. **CollectionTickets** has a log of summary information about all collections made at a location for a given date. One of the obvious constraints is that the collected taxes should not exceed the assigned taxes.

All branches of the TurboMachine Company contain identical set of tables. In the process of the creation of a data warehouse, data from all branches was used to populate the data warehouse. Let us assume that the data warehouse schema for the TurboMachine Company is a relatively typical star schema as shown in Picture 3.

Picture 3. The initial star schema for the data warehouse for the TurboMachine Company



Conceptually, the model includes dimension tables: **Machines**, **Locations**, and **Time**, and the fact tables **Facts_CollTax** and **Facts_AsTax**. Here, for simplicity, we assume that the table **Time** does not need to be included in the following discussion, since all time attributes can be identified in the fact table **Facts_CollTax**. Populating the dimension tables **Machines** and **Locations**, and the fact table **Facts_AsTax** by data contained in the local databases is conceptually straightforward. Populating the fact table **Facts_CollTax** is a little more complicated. It involves creating a join of each pair of **MeterReadings** and **CollectionTickets**, and then union all of the results. The reason for performing a join is to create a fact table that is in the center of the star (needs to have foreign keys for all dimension tables).

Very often, a data warehouse schema contains summary tables in order to improve performance. The discussion on the choice of summary tables is beyond the scope of this paper, but many aspects are already described in. The summary tables are obtained by grouping **Facts_CollTax** by various index attributes. Table **T23** is obtained by grouping **Facts_CollTax** by the index attributes *LNumber*, *Year*, *Month*, *Day* and aggregating *Money* to obtain *SumOfMoney* and *Taxes* to obtain *SumOfTaxes*. During this operation the *MID* attribute is removed, thus we will refer to this operation as **R**, for remove, with the appropriate argument. The summary table **T23** can be defined using the following statement.

```
CREATE TABLE T23
  (LNumber, Year, Month, Day, SumOfMoney, SumOfTaxes) AS
SELECT LNumber, Year, Month, Day, SUM(Money), SUM(Taxes)
FROM T11 GROUP BY LNumber, Year, Month, Day;
```

Similarly we can create table **T24** by grouping **Facts_CollTax** by index attributes *LNumber*, *MID*, *Year* and *Month*, and aggregating *Money* to obtain *SumOfMoney*.

```
CREATE TABLE T24 (LNumber, Year, Month, MID, SumOfMoney) AS
SELECT LNumber, MID, Year, Month, SUM(Money),
FROM Facts_CollTax GROUP BY LNumber, MID, Year, Month;
```

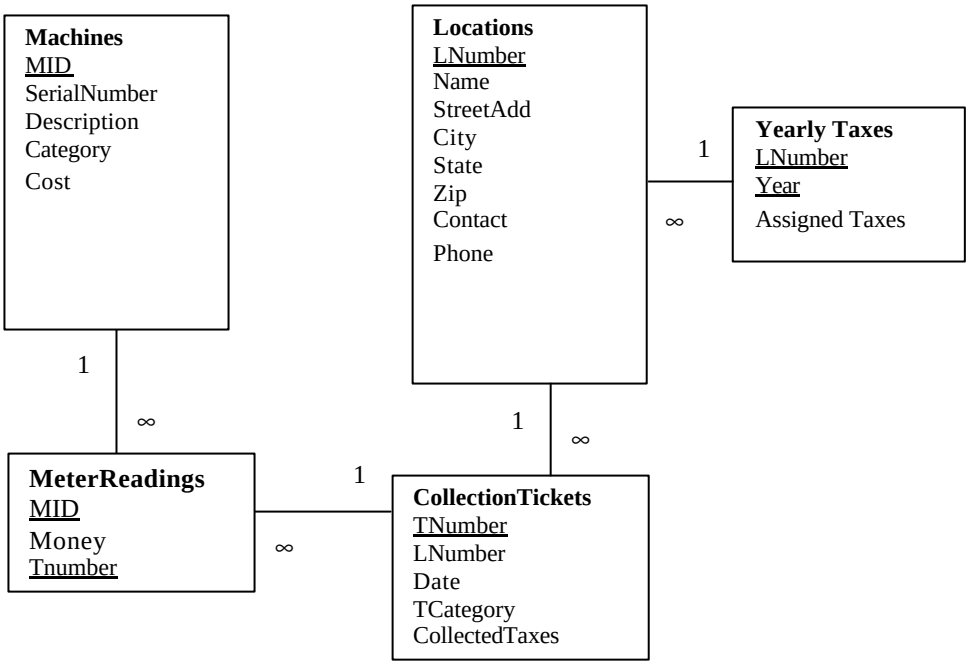
The next level (third level) tables can be created from table **Facts_CollTax** from the first level or from some already computed tables from the second level. For example, to create table **T35** we can use table **Facts_CollTax**, **T23**, or **T24**. If we choose table **T24**, then we need to do a grouping of **T24** by index attributes *LNumber*, *Year* and *Month*, and aggregating *Money* to obtain *SumOfMoney*. The following statement can be used to define the summary table **T35**.

```
CREATE TABLE T35 (LNumber, Year, Month, SumOfMoney) AS
SELECT LNumber, Year, Month, SUM(SumOfMoney),
FROM T24 GROUP BY LNumber, Year, Month;
```

5. IMPACT OF EVOLVING DATA SOURCES ON DATA WAREHOUSE SYSTEM

The schema of external data sources can change significantly, as we mentioned previously. In our case, let us assume that the new method of computing and collecting taxes was introduced based on taxing the locations rather than machines. This results in changing several attributes in existing tables: **Yearly Taxes**, **CollectionTickets**, and **MeterReadings** as shown in Picture 4.

Picture 4. Database for a branch of the TurboMachine Company



Yearly Taxes also contains the information about assigned taxes, but this time it is related to location for the specific year. Also **MeterReadings** does not contain the information about collected taxes any more but rather **CollectionTickets** includes important for our consideration attribute *CollectedTaxes*. Here again, the constraint is that the collected taxes should not exceed the assigned taxes is still valid but it should be expressed differently.

Conceptually, the model includes dimension tables: **Machines**, **Locations**, and **Time**, and fact tables **Facts_CollTax** and **Facts_AsTax**. Populating the dimension tables **Machines** and **Locations** would not change. Population of the fact tables **Facts_CollTax** and **Facts_AsTax** is done differently, but changes are conceptually straightforward.

Population of the summary table **T23** needs to be more significantly changed. This table is obtained in several steps. First, **T23A** is obtained by grouping **Facts_CollTax** by appropriate index attributes of **Facts_CollTax**.

```
CREATE TABLE T23A(LNumber, Year, Month, Day, SumOfMoney) AS
SELECT LNumber, Year, Month, Day, SUM(Money), SUM(Taxes)
FROM Facts_CollTax
GROUP BY LNumber, Year, Month, Day;
```

Then, the table **T23A** is joined with the fact table **Facts_CollTax** in order to include in the resulting table **T23** the attribute *SumOfTaxes*. The creation of the table **T35** and **T43** would be done without change. However, creation of the tables **T24** and **T36** would be slightly modified reflecting the fact that they do not have the *SumOfTaxes* attribute.

```
CREATE TABLE T24 (LNumber, Year, Month, MID, SumOfMoney) AS
SELECT LNumber, MID, Year, Month, SUM(Money),
FROM Facts_CollTax
GROUP BY LNumber, MID, Year, Month;
```

As was discussed before, the new data warehouse schema requires some data warehouse content adjustments. The first approach is based on archiving the original data warehouse schema, and corresponding data. This solution leads to multi-version data warehouse system and raises new challenges in the maintenance and evolution of such data warehousing systems and applications.

Another approach is based on adjustment of the data warehouse content accordingly to changes in the data warehouse schema caused by EDS changes. For example, adding, removing or modifying the existing attribute in the data warehouse may be necessary as shown below.

```
CREATE TABLE T23
(LNumber, Year, Month, Day, SumOfMoney, Taxes) AS
SELECT LNumber, Year, Month, Day, SumOfMoney, SumOfTaxes
FROM OLD_Facts_CollTax;
```

In order to distinguish here between old and new table **Facts_CollTax** we referred to the old **Facts_CollTax** table **OLD_Facts_CollTax**. Similarly removing the *SumOfTaxex* attribute from the old table **T24** (called **OLD_T24**) is necessary as shown below.

```
CREATE TABLE T24 (LNumber, MID, Year, Month, SumOfMoney) AS
SELECT LNumber, MID, Year, Month, SumOfMoney)
FROM OLD_Facts_CollTax;
```

Moreover, it may be possible to create new fact tables, and/or summary tables. A good example is the creation of new **`Facts_AsTax`** table from old **`Facts_AsTax`** table. Depending on the way the taxes collection changed it may or may not be possible.

6. CONCLUSIONS

In this paper, we discussed the problem of the data warehouse evolution resulting from the changes in the underlying external data sources. We show how source schema changes affect the data warehouse schema, the middleware level and the data warehouse content. We presented issues concerning the changes in dimension hierarchy and their impact on correct query processing. We showed a process of creation of a data warehouse on the example of the TurboMachine Company. We illustrated the evolution of the sample data warehouse schema and its middleware under data sources changes on the example of the same TurboMachine Company.

Bibliography

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

[2] A. Elmagarmid, M. Rusinkiewicz, and A. Sheth, eds. *Management of Heterogeneous and Autonomous Database Systems.* San Francisco, CA*: Morgan Kaufmann Publishers, Inc*., 1999.

[3] Wiederhold G.: *Mediators in the architecture of future information systems*, IEEE Computer C-25, 1, 1992.

[4] Adamson C. and Venerable M.: *Data Warehouse Design Solutions*, John Wiley & Sons, Inc. 1998.

[5] Kimball R.: *The Data Warehouse Toolkit*, John Wiley & Sons, Inc. 1996.

[6] Bischoff J. and Alexander T.: *Data Warehouse: Practical Advice from the Experts*, New Jersey Prentice-Hall, Inc., 1997.

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

[8] Quass D. and Widom J.: *On-Line Warehouse View Maintenance*, Proceedings of the SIGMOD, 1997.

[9] Eder J., Koncilla C., Morzy T.: *The COMET Metamodel for Temporal Data Warehouse*, Proceedings of the CAISE 2002, pp.83-99

[10] Eder J., Koncilla C., *Changes of Dimensions Data in Temporal Data Warehouses*, Proceedings of the DaWak Conf. 2001, pp.85-94

[11] Eder J., Koncilla C., Morzy T.: *A Model for a Temporal Data Warehouse*, Proceedings of the Intl. OESSEO 2001 Conf. Rome, Italy, 2001, pp.86-97

Bogdan Czejdo, Kenneth Messa
Department of Mathematics and Computer Science
Loyola University New Orleans, USA
{czejdo,messa}@loyno.edu

Tadeusz Morzy, Mikolaj Morzy
Institute of Computing Science
Poznan University of Technology, Poland
{Tadeusz.Morzy,Mikolaj.Morzy}@cs.put.poznan.pl

Janusz Czejdo
Edinboro University of Pennsylvania, USA
jczejdo@edinboro.edu