Proceedings of the 4th International Conference on Enterprise Information Systems, ICEIS Press, Vol. 1, ISBN 972-98050-6-7, pp 81-88

Temporal Data Warehousing: Business Cases and Solutions

Johann Eder
University of Klagenfurt
Email: eder@uni-klu.ac.at

Christian Koncilia
University of Klagenfurt
Email: koncilia@isys.uni-klu.ac.at

Herbert Kogler
University of Klagenfurt
Email: hkogler@edu.uni-klu.ac.at

Keywords: Data Warehousing, Temporal Databases, Executive Information Systems, OLAP

Abstract: Changes in transaction data are recorded in data warehouses and sophisticated tools allow to analyze these data along time and other dimensions. But changes in master data and in structures, surprisingly, cannot be represented in current data warehouse systems impeding their use in dynamic areas and/or leading to erroneous query results. We propose a temporal data warehouse architecture to represent structural changes and permit correct analysis of data over periods with changing master data. We show how typical business cases involving change in master data can be solved using this approach and we discuss architectural variants for the implementation.

1 Introduction

Data Warehouses are structured collections of data supporting controlling, decision making and revision (Wu and Buchmann, 1997; Hüsemann et al., 2000). Data Warehouses build the basis for analyzing data by means of OLAP tools which provide sophisticated features for aggregating, analyzing, and comparing data and for discovering irregularities. Data Warehouses differ from traditional databases in the following aspects: They are designed tuned for answering complex queries rather than for high throughput of a mix of updating transactions, and they typically have a longer memory, i.e. they do not only contain the actual values (snapshot data) but also historical data needed for the purposes outlined above. Historical data can be stored either as directly as in historical databases or - more frequently - as already aggregated and abstracted data.

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 hierarchically organized in dimensions.

Surprisingly, data warehouses are not well prepared for changes in spite of their requirement for serving as long term memory. Of course they are very well designed for dealing with changes in transaction data, which are represented in the cells of multidimensional data cubes. However, they cannot adequately represent changes in master data which spans the dimension structures of such cubes, with changes in units, changes in formulas computing derived transaction data, or schema changes.

The reason for this disturbing property of current data warehouse technology is the implicitly underlying assumptions that the dimensions are orthogonal. Orthogonality with respect to the dimension time means the other dimensions ought to be time-invariant. This silent assumptions inhibits the proper treatment of changes in dimension data.

The consequences of this deficiency are the restricted applicability of data warehouse technology in dynamic domains with frequent changes of structural data. The storing of data over several periods, foundational for data warehouses is of rather limited use, if these data cannot be compared and aggregated over these periods to allow for trend computations and multi-period comparisons. On the other hand we often found incorrect data as results of OLAP analysis, sometimes the users where aware of the errors stemming from structural changes, more often not.

The goals of our research described in this paper are to make data warehouses more useful in dynamic application areas, and to improve the correctness of OLAP-results. For this purpose we need provisions to correctly answer queries for data in a particular point in time (snapshot data), and for retrieving data stemming from different periods in a way that they can be compared or commonly processed further on. As a direct consequence, it is not sufficient to just maintain schema and structure of a data warehouse to represent the ”new” view, since this would result in the loss of the ability to answer queries regarding previous periods correctly, restring the value of data warehouses for revision purposes. On the other hand, just keeping the correct master data and units for the transaction data is not sufficient either, since aggregation over data from different periods would give wrong results.

In particular, we propose a temporal data warehouse architecture which extends multidimensional data warehouses to achieve the following features:

- representation of changes in master data, units and schema of data warehouses
- identification of structure versions (SV) as changeless periods
- provision of mappings of transaction data between schema versions.
- supporting queries which touch data spanning several structural versions

In the next chapter we provide a series of business cases showing typical examples of changes, a data warehouse has to cope with. In section 3 we introduce the temporal extensions we propose and in discuss, how the cases presented in section 2 can be solved with our approach. In section 4 we show how queries are processed in the proposed temporal data warehouse, and in section 5 we discuss some architectural variants for the implementation of our temporal data warehouse.

2 Business Cases

The data stored in a data warehouse changes over time. Reasons for modifications are for example changing user needs, modifications in the data sources, changing legal and regulativ conditions.

(Sarda, 2001) defines the following general business metadata categories: functions and missions, organization elements, goals, business entities, processes, external events, measures, evaluation, actions and business concepts. He shows the relation between the defined business metadata and the data warehouse and its metadata. Furthermore, he emphasizes that these metadata are dynamic and do change over time.

In this section we will discuss some business cases, i.e. some of the changes in business metadata that can be reflected in the data warehouse with our temporal data warehouse approach. We distinguish between changes on the schema level and changes on the instance level. Furthermore, data on the instance level can be categorized in transaction data and master data.

The schema of a data warehouse is described by *Dimensions*, e.g. Time or Region, and *Dimension-Levels*, e.g. Year, Quarter and Month. Furthermore, the schema defines the hierarchial relations between Dimension-Levels, e.g. Year $\leftarrow$ Quarter $\leftarrow$ Month, where „$\leftarrow$” means „rolls-up to” . Instances are the extensions of these Dimension-Levels and there hierarchical relations, e.g. 1999, $1.Quarter$ and Jan where $1999 \leftarrow 1.Quarter \leftarrow Jan$. We call those instances Dimension-Members. Furthermore, measures are also instances of a data warehouse representing the values that the user wants to analyze.

For the rest of this paper, we will use the following running example from the health care sector: consider a data warehouse to store information about diseases with the dimensions *Time*, *Geography*, *Diagnosis* and *Facts*. Dimension-Levels are defined as follows: $Time$: $Year \leftarrow Month$; $Geography$: $Country \leftarrow Province \leftarrow District \leftarrow City$ and $Diagnosis$: $Group \leftarrow Diagnose$. Dimension facts has two extensions $Costs$ representing the average costs per disease and $Quantity$ representing the number of patients per disease. The history of dimension $Geography$ is visualized in Fig. 2.

2.1 Instance Modifications

2.1.1 Transaction Data Modifications

In data warehouses transaction data are called measures. Measures represent the values, e.g. sales or turnover that the user can analyze. In most cases the user wants to keep track of modifications of these transaction data. Therefore, a very common dimension in data warehouses is the dimension $Time$. Introducing this dimension enables the user to analyze for example how the turnover developed over time. The dimension $Time$ enables us to keep track of changing measures.

Data warehouses are very well designed for dealing with changes in transaction data.

2.1.2 Master Data Modifications

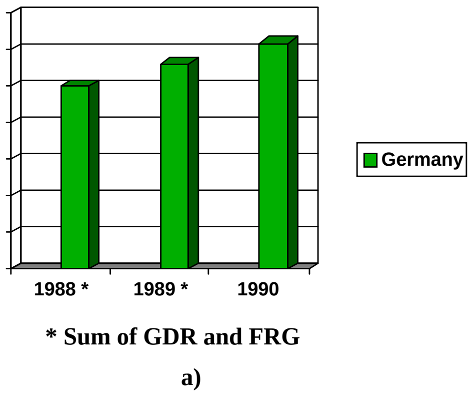
Nevertheless, the dimension $Time$ does not help us to keep track of modifications of master data. Master data are describing the extensions of dimension levels. For example the values $Spain$ and $Germany$ are master data of the dimension level $Country$.

We will now discuss some typical examples of master data modifications:

- **Keys**: It happens quite often that the key that identifies an instance in the data source changes over time. In our running example, the country $Zaire$ was renamed in 1997 and is now known as $Kongo$.

Another example: diagnoses for patients are represented using the „International Statistical Classification of Diseases and Related Health Problems" (ICD) code. However, codes for diagnoses changed from ICD Version 9 to ICD Version 10. Even worse, the same code described different diagnoses in ICD-9 and ICD-10. Other ICD-10 codes are a specialization of ICD-9 codes, i.e. the granularity of codes changed. Let us say that in ICD-9 we have a code X for the diagnose D and in ICD-10 we have two codes X_1 for diagnose D_1 and X_2 for diagnose D_2 where D_1 and D_2 are specializations of D.
In this example we would not be able to correctly answer a query like „Show number of diagnoses for code X for all years" with a non-temporal data warehouse because this code changed from ICD-9 to ICD-10.

- **Regrouping**: Hierarchies represent how the data stored in the data warehouse can be aggregated and disaggregated. Consider that in our running example that $Austria$ became a part of the region $European - Union$ and is no longer a part of the region $Non-EU-Country$ starting with January 1995. Queries like „Show diseases of Non-EU-Countries for the last ten years" would no longer return a correct result in a non-temporal data warehouse.
- **Regrouping to a different Level**: Another important issue that has to be covered by temporal data warehouses is that it has to be able to treat regroupings of instances to a different dimension level in a correct way. In our running example this happened to the Czech Republic that was a $Province$ of Czechoslovakia until 1993 and is now an independent $Country$.
  The major problem when regrouping dimension members between different dimension-levels is, that facts can be computed according to the level of the dimension member.
- **Fact-Formulas**: Facts, e.g. Sales or Turnover, represent what can be analyzed with the data warehouse. We can distinguish between computed facts, e.g. the Cash Flow that can be computed as $NetIncome + Depreciation + Amortization + Depletion$, and non-computed facts, e.g. the Turnover. However, these formulas can change over time. For example the way how to compute the unemployment rate changed in Austria because they joined the European Union in 1995 (in fact, the unemployment rate in Austria dropped dramatically simply because the way how to compute it was adopted to the standard used in the European Union). Correct analyzes would take in account that the underlying formula changed.
- **Units**: Units of facts could change over time. In our running example the unit for the fact $Costs$ changes from Spain Pesetas (ESP) or Deutschmark (DM) to EURO starting with 2002. We could easily compute values from one currency into another by multiplying it with the correct value.
  On the other hand, there are also changes of units where only an approximatively computation can be done. Consider for our running example, that the granularity of the dimension $Time$ changes from $Month$ to Day starting with January 1999. If the data source does not support data on a daily basis for all timepoints before January 1999, we could only re-compute the old values by dividing all monthly values through the corresponding number of days per month.
- **Split / Merge**: In our running example another kind modification had an impact on the structure of the dimension countries, namely mergers (the re-unification of $Germany$) and splits (the „split" of $Yugoslavia$). Non-temporal data warehouses would return senseless results for queries like „Show diseases of Yugoslavia for the last 50 years".
- **Delete / Insert**: The most frequent kind of modifications on the instance level is to delete or insert a new dimension member, e.g. if the product port-folio of a company changes over time. Nevertheless, these kind of modifications are still a problem to most data warehouse architectures when analyzing the stored data. Simply due to the fact that if we insert a new tuple in the warehouse at a timepoint T and the user states a query that implies data from timepoints before T the result for the inserted tuple would be $NULL$. However, the user should be aware of the fact that $NULL$ means in this case $not - applicable$ (because the tuple did not exist) and that it does not mean $no - data - imported - yet$. The same applies of course when deleting a tuple and stating a query for timepoints after T.
- **Attributes**: Attributes of instances are commonly used to store further information about instances that we don't want to put into an unique dimension. In our running example we could want to store the population of states (and of course the population is changing from year to year).
  Another example: for a dimension Products we could store information about the color of the products or an insurance company could store information about the policy period for a dimension Policies. Attributes are quite valuable information for the user of a data warehouse. Imagine for example that the result of query about the sales of a product shows that the sales declined.
  Further investigations of the attributes of this products could show that this started at the same point of time as the color of the product changed.
- **Time**: We treat the dimension Time in a data warehouse as any other dimension. This enables us



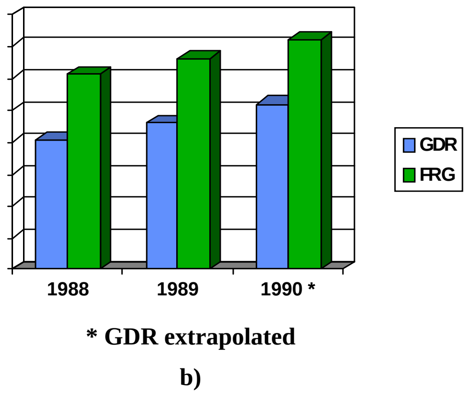


Figure 1: Result of a query with a) SV_2, SV_3 or SV_4 and b) SV_1 as base structure version (see also Fig. 2)

to keep track of modifications of dimension Time. What happens quite frequently to the dimension Time is that its granularity changes over time. This happens if for example the underlying data source is no longer able to deliver data with the needed granularity, or vice-versa if the granularity of the underlying data source becomes finer and therefore enables us to store the data in the data warehouse on a finer level.

2.2 Schema Modifications

- **Dimensions**: The schema of a data warehouse defines what kind of queries the user can ask. Consider that for our running example the user-need arises to distinguish between male and female patients when analyzing diseases. This would lead to a new Dimension *Gender*. On the other hand, it also happens that - due to changes in the data sources - a whole dimension has to be discontinued.
- **Dimension-Levels**: Consider for example that the user wants to aggregate data from months into quarters. Thus, we would need another dimension level *Quarter* between the defined dimension levels *Year* and *Month*.
  Vice-versa it could also happen that the data source for our running example does no longer support data on a monthly basis, but only on a yearly basis. We therefore would have to discontinue the dimension level *Month*.

3 Temporal Data Warehouse

As already mentioned in the Introduction the reason for the problems contemplated in the previous section is the implicitly underlying assumptions that the dimensions are orthogonal. Orthogonality with respect to the dimension $Time$ means the other dimensions ought to be time-invariant. This silent assumptions inhibits the proper treatment of changes in dimension data.

The dimension $Time$ ensures to keep track of the history of measures, i.e. transaction data. Nevertheless, in order to gain correct query results after modifications of master data, i.e. dimension data, we have to track modifications of these data (Chamoni and Stock, 1998; Eder and Koncilia, 2001). Hence, all dimension members and all hierarchical links between these dimension members have to be time stamped with a time interval $[T_s, T_e]$ representing the valid time where T_s is the beginning of the valid time, T_e is the end of the valid time and $T_e \geq T_s$. Furthermore, we time stamp all schema definitions, i.e. dimensions, dimension-levels and their hierarchical relations, in order to keep track of all modifications of the data warehouse schema (Eder et al., 2001).

If we represent all time stamps of all modifications within our data warehouse on a linear time axis the time-lag between two contiguous time points on this axis represents a structure version. This means that a structure version is a view on a temporal data warehouse valid for a given time period $[T_s, T_e]$. Therefore, within one structure version the structure of dimension data on both the schema level and on the instance level is stable. Information about structure versions can be gained from our temporal data warehouse using temporal projection and temporal selection (Jensen and Dyreson, 1998).

For our running example, the structural changes and the resulting structure versions (SV) for the dimension $Geography$ are depicted in Fig. 2. For example, the reunification of Germany in 1990 leads to a new structure version SV_2.

The data returned by the query can, however, originate in several (different) structure versions. Therefore, it is necessary to provide transformation functions mapping data from one structure version to a different structure version.

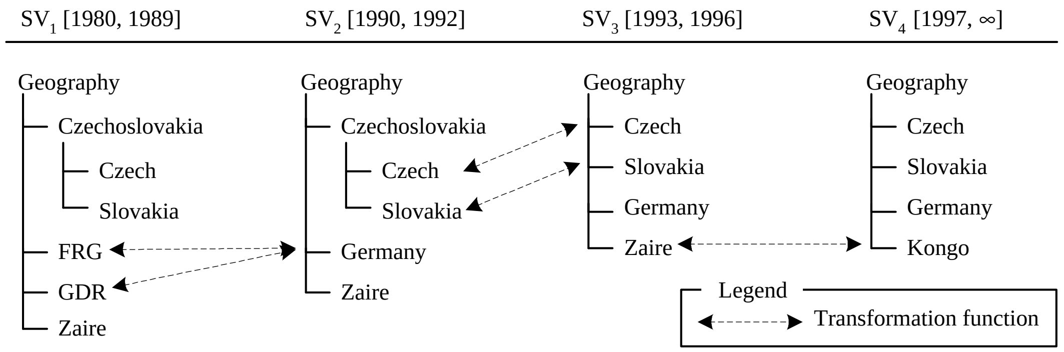


Figure 2: Structure Versions and Transformation Functions for dimension $Geography$ of our running example

Using transformation functions enables us to assure that a successful analysis can be made even though there might be changes in the dimension data and dimension structure. The combination of structure versions and transformation functions enables the user to analyze data with dimension data and dimension structures „backward" or „forward" in the time axis.

Fig. 2 shows some transformation functions, e.g. between the dimension members $Germany$ and FRG. In this example, the defined transformation function enables us to answer queries like „Show number of diseases with code X for Germany for the last 20 Years".

As proposed in (Sarda, 2001) the user of a data warehouse „must be proactively made aware of changes in business metadata". In our opinion, the best way to inform the user about changes in business metadata is to enrich all query results stated against the data warehouse with meaningful user information. In other words, we have to inform the user about modifications that had an impact on the stated query. As shown in Fig. 1 this can be done with footnotes.

Our approach enables us to inform the user of a data warehouse about modifications of the business metadata that had an impact on the stated queries, i.e. we are able to generate footnotes and a description of the modifications automatically and/or semi-automatically. This can be done automatically by indicating the defined transformation functions. Furthermore, it can be done semi-automatically by applying a descriptive text to transformation functions as shown in Fig. 2.

We will show some examples of transformation functions in the following section.

3.1 Solutions to Business Cases

As proposed in section 2 we distinguish between modifications on the instance level and modifications on the schema level.

3.1.1 Instance Modifications

In contrast to modifications on the schema level modifications on the instance level do occur quite often. Our temporal data warehouse approach enables us to deal with the following modifications on the instance level:

- **Keys**: As shown in Fig. 2 we are able to deal with modifications of keys ($Zaire$ was renamed to $Kongo$) by introducing a transformation function between structure versions SV_3 and SV_4 with the weighting factor 1, i.e. 100 persent. This function enables us to transform „old" data stored in structure version SV_3 (Zaire) into the new structure defined in structure version SV_4 (Kongo). On the other hand, we can also enable the user to analyze the stored values with an old structure. Therefore, we have to introduce another transformation functions to map „new" data stored in structure version SV_4 into the „old" structure of SV_3. The weighting factor for this transformation function would also be 1.
  We implicitly introduce a transformation function for each dimension member which does not change from one structure version into another with a weighting factor 1. This enables us to transform for example the data stored for $Kongo$ into each previous structure version, e.g. SV_3, SV_2 or SV_1.
- **Regrouping**: We do not only timestamp dimension members but also the defined hierarchical relations between dimension members. This enables us to keep track of each regrouping of a dimension member. As the dimension member itself did not change, we implicitly introduce a transformation function with the weighting factor 1.
  Furthermore, the cell values of upper-level dimension members are always computed from their subordinate lower level dimension members. Before we transform the data we select the dimension members at level-0 and transform only this subset of cell values and compute the upper-levels of the

resulting values bottom-up as usual.

- **Fact-Formulas**: We do treat the dimension Facts as any other dimension. This enables to store the history of formulas. Furthermore, we are able to introduce transformation functions between Facts. This enables us to transform data for a given fact and then compute the „new" values with „old" formulas and vice-versa, „old" values with „new" formulas.
- **Units**: As our approach allows us to introduce transformation functions for facts we can easily define transformation functions to map for example one currency into another. For example we could introduce a transformation with a weighting factor 0.00601012 that allows us to transform Spain Pesetas into Euro, and vice-versa we can introduce a transformation function with a weighting factor 166.38609 to transform Euro into Spain Pesetas. This enables to exactly transform one value into another.
  On the other hand, we could also want to introduce transformation functions to transform data at least approximatively, e.g. when computing monthly values into daily values. This can be done by introducing a transformation function between each day of the month and the month. The weighting factor of this transformation function would be $1/\#DAYS$, where $\#DAYS$ represents the number of days in the corresponding month. For example, we can compute the value for January the 1st by introducing a transformation function with the weighting factor $1/31$.
  We can easily and exactly transform daily values into monthly values by introducing a transformation function between the month and each day of the month with a weighting factor 1. This results into a formula $January = Day_1 + Day_2 + ... + Day_{31}$ to compute all days of January into a monthly value for January.
- **Split / Merge**: The case of a split or merge of a dimension member is similarly to the case mentioned above when transform monthly values into daily values and vice-versa.
  We can easily and exactly transform the data stored for GDR and FRG as defined in structure version SV_1 (see Fig. 2) into the succeeding structure versions SV_2, SV_3 and SV_4. This can be done by introducing a transformation between structure versions SV_1 and SV_2 for GDR and $Germany$ with a weighting factor 1 and for FRG and $Germany$ with a weighting factor 1.
  On the other hand, we can at least approximatively transform the values stored for example in structure version SV_2 for $Germany$ into values for FRG as defined in structure version SV_1. This can be done by introducing a transformation function with a weighting factor that corresponds for example to the ratio of the population of $Germany$ and FRG, or $Germany$ and GDR respectively.
- **Delete / Insert**: If for example a new disease becomes part of the ICD catalog, we can not transform any „old" data into the resulting „new" structure - simply because there exists no data for the corresponding disease before the insertion. However, we can inform the user about this modification of the ICD catalog and inform him about the meaning of $NULL$ values.
  In contrast to an insert operation there is a relation between a deleted dimension member and the following structure version. For example, if the user request data for a disease that is no longer a part of the ICD catalog we can introduce a transformation function with the weighting factor 0 to represent this modification.
- **Attributes**: As we do timestamp dimension members and hierarchical relations between dimension members we keep track of the history of attributes belonging to dimension members. This enables us to show exactly those attributes that are valid for the selected time period.
- **Time**: (see solutions for „Units")

3.1.2 Schema Modifications

Our generic temporal data warehouse metamodel introduced in (Eder et al., 2001) keeps track of modifications on both the instance level and the schema level. After deleting a dimension the data stored in the data warehouse can be recomputed from one structure version into another by aggregation. The same applies when deleting a dimension level.

However, there is no appropriate way to automatically recompute values from one structure version into another after inserting a new dimension or dimension level. We are currently working on a way to support semi-automatically recomputation of values after inserting new dimensions or dimension levels, called *Transformation Methods*.

4 Querying a Temporal Data Warehouse

When a user issues a query within our temporal data warehouse, he/she has to define a timepoint T. This timepoint specifies a certain base structure version where $T_s \leq T \leq T_e$ and $[T_s, T_e]$ defines the valid time interval of the base structure version.

This base structure version determines which structure has to be used for the analysis. In most cases this will be the current structure version. However, in some cases it will be of interest to use an "older" structure version.

Consider that for our running example depicted in Fig. 2 only the dimension data of dimension

$Geography$ did change over time. Hence, the resulting set of structure versions would be (the symbol ∞ represent that this structure version is valid until now):

Structure Version	T_s	T_e
SV_1	1980	1989
SV_2	1990	1992
SV_3	1993	1996
SV_4	1997	∞

We assume that the user requests data for 1988, 1989 and 1990 and chooses the structure version SV_3 as base structure version. For answering this query the system needs to map the data which are valid in the structure version SV_1 into the structure defined through structure version SV_3. Figure 2.a shows the result of this query.

5 Temporal Data Warehouse-System Architecture

In (Eder et al., 2001) two different approaches for possible architectures of a temporal data warehouse-system are discussed - the „indirect approach" and the „direct approach" (see Fig. 3). Common components of both approaches are the **Admin Tool**, the **Temporal Data Warehouse** and the **Transformer**. The Admin Tool is implemented in Java and allows the administrator to import new data in the temporal data warehouse and to perform modifications in the dimension data and dimension structures. All data is stored in the temporal data warehouse that was built with Oracle 8.1i as relational database management system. The Transformer for the indirect approach is implemented in C++.

5.1 Data Transformation Approaches

5.1.1 Indirect Approach

The main idea of the indirect approach is, as shown in Fig. 3a), that the Transformer generates one data mart for each structure version needed by the user. In most cases, this will only be the actual structure version. Each data mart consists of all fact data that are valid for the same time interval as the corresponding structure version plus it consists of all fact data that could be transformed by the defined transformation functions from all other structure versions.

Each data mart is stored as a Hyperion Essbase Cube. As we use this standard OLAP database for each data mart, the main advantage of the indirect approach is that each data mart offers the whole OLAP functionality, e.g., drill-down, roll-up, slice, dice, etc. and no implementation of a front-end is needed.

5.1.2 Direct Approach

The architecture of the direct approach consists of two additional parts as shown in Fig. 3 b). The Query Analyzer takes a query stated by the user as input and analyzes which data out of which structure version is necessary to answer the query. The result of this analysis is passed to the Transformer and to the Result Analyzer.

The Transformer works as described above. In contrast to the indirect approach, the Transformer is triggered by the user or, in other words, for each stated query the Transformer transforms all required cell values to answer the query.

The Result Analyzer takes its input from the Query Analyzer and from the Transformer. It enriches the result of the Transformer with further user information, i. e., with information about what structural modifications had an impact on the stated query. The Result Analyzer is a subject of ongoing research.

The main advantage of the direct approach is its flexibility.

5.2 Query Transformation Approach

We are currently working on an additional approach. We call this approach „Query Transformation Approach". The Query Analyzer of this approach works like a mediator (Widom, 1995). It splits up a query in n sub-queries where n is the number of structure versions needed to answer the stated query.

In contrast to a mediator which sends the sub-queries through wrappers to different data sources, our approach sends the sub-queries to different structure versions.

The result of the sub-queries and the information of the defined transformation functions is needed from the Transformer to transform and merge the results of the sub-queries.

6 Conclusions

„Nothing is sure but change" goes a saying. Unfortunately, many of our information systems are ill prepared for change and, surprisingly, multidimensional data warehouse systems are among those. We presented a series of typical business cases involving change in structural data or master data we found which cannot be solved with current data warehouse technology, in spite of the common perception that exactly this technology is useful for dealing with data over longer periods of observation.

Our extensions: time-stamping of master data and transformation functions, allow to correctly represent changes in structural data and allows to correctly analyze data in spite of changes.

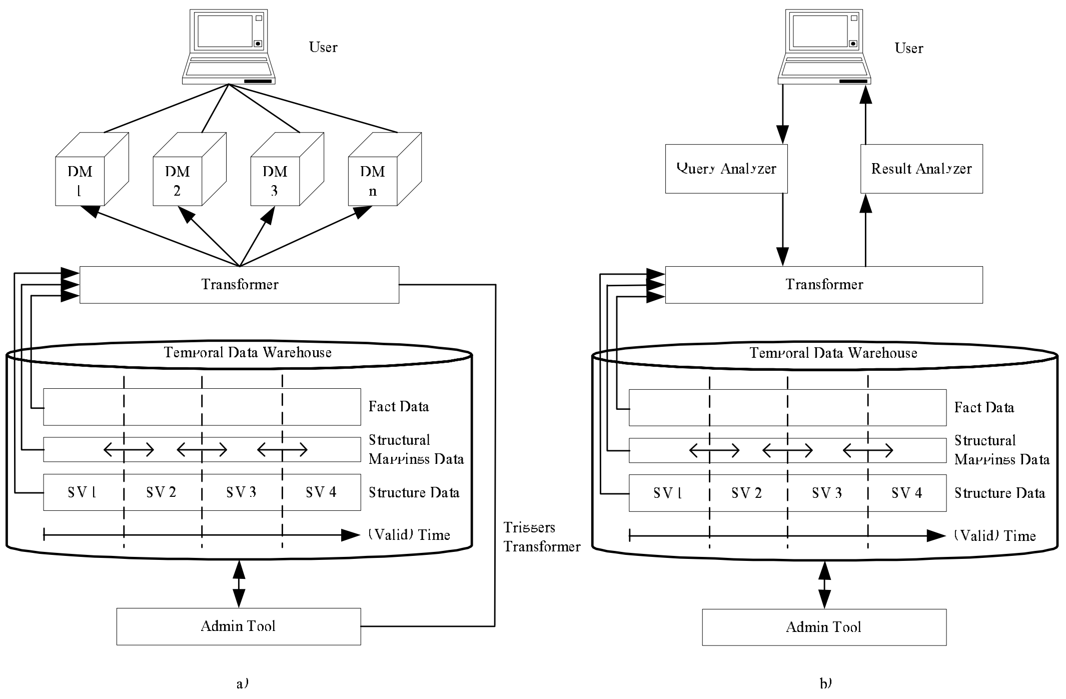


Figure 3: Architectures of a) the Indirect Approach and b) the Direct Approach (Eder et al., 2001)

We showed how typical business cases can adequately be solved by our technique and also discuss architectural variants for such data warehouse systems. It is our ambition to contribute to a broader applicability of data warehouse technology and to a reduction of unintentionally wrong statistics due to incorrect query results.

REFERENCES

Chamoni, P. and Stock, S. (1998). Modellierung temporaler multidimensionaler Daten in Analytischen Informationssystemen. In Kruse, Rudolf, Saake, and Gunter, editors, *Arbeitsbericht 14*, pages 93–106. Otto-von-Guericke-Universität Magdeburg, Magdeburg.

Eder, J. and Koncilia, C. (2001). Changes of Dimension Data in Temporal Data Warehouses. In *Proc. of the DaWak 2001 Conference*, Munich, Germany.

Eder, J., Koncilia, C., and Morzy, T. (2001). A Model for a Temporal Data Warehouse. In *Proc. of the Int. OESSEO 2001 Conference*, Rome, Italy.

Etzion, O., Jajodia, S., and Sripada, S., editors (1998). *Temporal Databases: Research and Practise*. Number LNCS 1399. Springer-Verlag.

Hüsemann, B., Lechtenbörger, J., and Vossen, G. (2000). Conceptual Data Warehouse Design. In *Proc. of the International Workshop on Design and Management of Data Warehouses (DMDW 2000)*, Stockholm.

Jensen, C. S. and Dyreson, C. E., editors (1998). *A consensus Glossary of Temporal Database Concepts - Feb. 1998 Version*, pages 367–405. Springer-Verlag. in [EJS98].

Sarda, N. (2001). Structuring Business Metadata in Data Warehouse Systems for Effective Business Support. In *cs.DB/0110052* . arXiv Archieve. URL: http://arXiv.org/.

Widom, J. (1995). Research Problems in Data Warehousing. *ACM*.

Wu, M. and Buchmann, A. (1997). Research Issues in Data Warehousing. *BTW'97*.