SURVEY ARTICLE

A Survey on Temporal Data Warehousing

Matteo Golfarelli, DEIS - University of Bologna, Italy

Stefano Rizzi, DEIS - University of Bologna, Italy

ABSTRACT

Data warehouses are information repositories specialized in supporting decision making. Since the decisional process typically requires an analysis of historical trends, time and its management acquire a huge importance. In this paper we consider the variety of issues, often grouped under term temporal data warehousing, implied by the need for accurately describing how information changes over time in data warehousing systems. We recognize that, with reference to a three-levels architecture, these issues can be classified into some topics, namely: handling data/schema changes in the data warehouse, handling data/schema changes in the data mart, querying temporal data, and designing temporal data warehouses. After introducing the main concepts and terminology of temporal databases, we separately survey these topics. Finally, we discuss the open research issues also in connection with their implementation on commercial tools.

Keywords: *data warehouse; data mart; design; evolution; temporal databases; versioning*

INTRODUCTION

At the core of most business intelligence applications, *data warehousing systems* are specialized in supporting decision making. They have been rapidly spreading within the industrial world over the last decade, due to their undeniable contribution to increasing the effectiveness and efficiency of the decisional processes within business and scientific domains. This wide diffusion was supported by remarkable research results aimed at improving querying performance, at refining the quality of data, and at outlining the design process, as well as by the quick advancement of commercial tools.

In the remainder of the paper, for the sake of terminological consistency, we will refer to a classic architecture for data warehousing systems, illustrated in Figure 1, that relies on three levels:

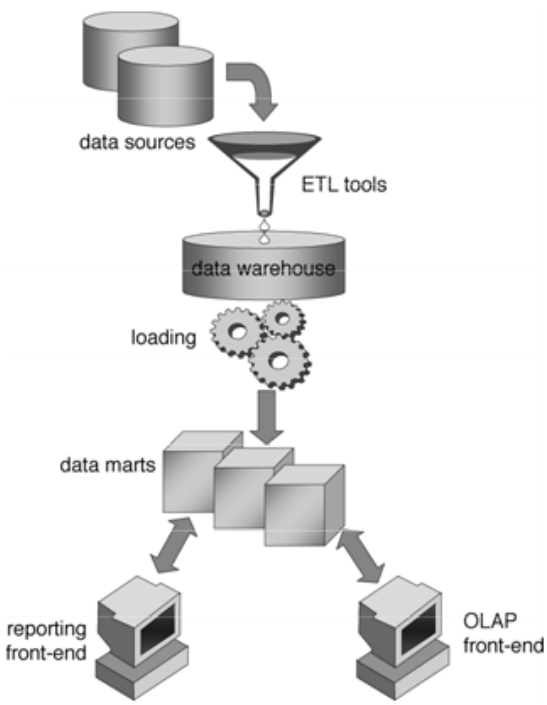
1. The *data sources*, that store the data used for feeding the data warehousing systems. They are mainly corporate operational databases, hosted by either relational or legacy platforms, but in some cases they

may also include external web data, flat files, spreadsheet files, etc.
2. The *data warehouse* (also called *reconciled data level*, *operational data store* or *enterprise data warehouse*), a normalized operational database that stores detailed, integrated, clean and consistent data extracted from data sources and properly processed by means of ETL tools.
3. The *data marts*, where data taken from the data warehouse are summarized into relevant information for decision making, in the form of *multidimensional cubes*, to be typically queried by OLAP and reporting front-ends.

Cubes are structured according to the *multidimensional model*, whose key concepts are fact, measure and dimension. A *fact* is a focus of interest for the decisional process; its occurrences correspond to *events* that dynamically occur within the business world. Each event is quantitatively described by a set of numerical *measures*. In the multidimensional model, events are arranged within an n-dimensional space whose axes, called *dimensions* of analysis, define different perspectives for their identification. Dimensions commonly are discrete, alphanumerical attributes that determine the minimum granularity for analyzing facts. Each dimension is the root of a *(roll-up) hierarchy* that includes a set of *levels*, each providing a way of selecting and aggregating events. Each level can be described by a set of *properties*.

As a consequence of the fact that the decisional process typically relies on computing historical trends and on comparing snapshots of the enterprise taken at different moments, one of the main characterizations of data warehousing systems is that of storing historical, non volatile data. Thus, time and its management acquire a huge importance. In this paper we discuss the variety of issues, often grouped under term *temporal data warehousing*, implied by the need for

Figure 1: Three-levels architecture for a data warehousing system



accurately describing *how information changes over time*. These issues, arising by the never ending evolution of the application domains, are even more pressing today, as several mature implementations of data warehousing systems are fully operational within medium to large business contexts. Note that, in comparison with operational databases, temporal issues are more critical in data warehousing systems since queries frequently span long periods of time; thus, it is very common that they are required to cross the boundaries of different versions of data and/or schema. Besides, the criticality of the problem is obviously higher for systems that have been established for a long time, since unhandled evolutions will determine a stronger gap between the reality and its representation within the database, which will soon become obsolete and useless (Golfarelli et al, 2006).

So, not surprisingly, there has been a lot of research so far regarding temporal issues in data warehousing systems. Basically, the approaches devised in the literature can be accommodated in the following (sometimes overlapping) categories:

- *Handling changes in the data warehouse* (discussed in the third section). This mainly has to do with maintaining the data warehouse in sync with the data sources when changes on either of these two levels occur.
- *Handling data changes in the data mart* (fourth section). Events are continuously added to data marts; while recorded events are typically not subject to further changes, in some cases they can be modified to accommodate errors or late notifications of up-to-date values for measures. Besides, the instances of dimensions and hierarchies are not entirely static.
- *Handling schema changes in the data mart* (fifth section). The data mart structure may change in response to the evolving business requirements. New levels and measures may become necessary, while others may become obsolete. Even the set of dimensions characterizing a fact may be required to change.
- *Querying temporal data* (sixth section). Querying in presence of data and schema changes require specific attention, especially if the user is interested in formulating queries whose temporal range covers different versions of data and/or schema.
- *Designing temporal data warehouses* (seventh section). The specific characteristics of temporal data warehouses may require ad hoc approaches for their design, especially from the conceptual point of view.

The paper outline is completed by the second section, that introduces the main concepts and terminology of temporal databases, and by the eighth section, that summarizes some open issues and draws the conclusions.

TEMPORAL DATABASES

Databases where time is not represented are often called *transient databases*. Within a transient database, only the current representation of real-world objects is stored and no track of changes is kept, so it is impossible to reconstruct how the object was in the past. Conversely, *temporal databases* focus on representing the inherent temporal nature of objects through the time-dependent recording of their structure and state. Two different time dimensions are normally considered in temporal databases, namely *valid time* and *transaction time* (Jensen et al., 1994). Valid time is the "real-world time", i.e., it expresses the time when a fact is true in the business domain. Transaction time is the "database system time", i.e., it expresses the time when facts are registered in the database. Temporal database systems are called *valid-time databases*, *transaction-time databases* or *bi-temporal databases* depending on their capacity to handle either or both of these two time dimensions (Tansel et al., 1993). The main benefit of using a bi-temporal database is that not only the history of the changes an object is subject to is recorded, but it is also

possible to obtain the same result from a query independently of the time when it is formulated (which might not happen if transaction time is not properly represented).

In the real world, objects change in both their state and their structure. This means that, within a database, both the values of data and their schema may change. Obviously, values of data are constantly modified by databases applications. On the other hand, modifying the database schema is a less frequent, though still common, occurrence in database administration. With reference to changes in the database schema, the literature commonly distinguishes three possibilities (Roddick, 1995):

- *Schema modification* is supported when a database system allows changes to the schema definition of a populated database, which may lead to loss of data.
- *Schema evolution* is supported when a database system enables the modification of the database schema without loss of existing data.
- *Schema versioning* is supported when a database system allows the accessing of all data, both retrospectively and prospectively, through user-definable version interfaces.

The significant difference between evolution and versioning is that the former does not require the maintenance of a schema history, while in the latter all past schema versions are retained. Note that, in the context of schema evolution and versioning, most authors agree that there is no need to distinguish valid time from transaction time (McKenzie & Snodgrass, 1990).

On the language side, TSQL2 (Snodgrass, 1995) is the most noticeable attempt to devise a query language for relational temporal databases. TSQL2 is a temporal extension to the SQL-92 language standard, augmented to enable users to specify valid-time and transaction-time expressions for data retrieval. As to querying in presence of schema versioning, while TSQL2 only allows users to punctually specify the schema version according to which data are queried, other approaches also support queries spanning multiple schema versions (Grandi, 2002).

The concepts introduced in this section were originally devised for operational databases, and in particular for relational databases. While in principle they can also be applied to data warehousing systems, that in a ROLAP implementations are based on relational databases, the peculiarities of the multidimensional model and the strong relevance of time in the OLAP world call for more specific approaches.

HANDLING CHANGES IN THE DATA WAREHOUSE

When considering temporal data, it is first of all necessary to understand how time is reflected in the database, and how a new piece of information affects existing data. From this point of view, Devlin (1997) proposes the following classification:

- *Transient data*: alterations and deletions of existing records physically destroy the previous data content.
- *Periodic data*: once a record is added to a database, it is never physically deleted, nor is its content ever modified. Rather, new records are added to reflect updates or deletions. Periodic data thus represent a complete record of the changes that have occurred in the data.
- *Semi-periodic data*: in some situations, due to performance and/or storage constraints, only the more recent history of data changes is kept.
- *Snapshot data*: a data snapshot is a stable view of data as it exists at some point in time, not containing any record of the changes that determined it. A series of snapshots can provide an overall view of the history of an organization.

Data sources normally adopt either a transient or a (semi-)periodic approach, depending

on whether the application domains requires keeping history of past data or not. The historical depth of a data warehouse is typically not less than the one of its data sources, thus data warehouses more often contain periodic data. Conversely, data marts normally conform to the snapshot model.

In order to model historical data in the data warehouse, Abello and Martín (2003) propose a bi-temporal storage structure where each attribute is associated to two couples of timestamps, so as to track the history of its values according to both valid and transaction time. Each attribute, or each set of attributes having the same behaviour with reference to changes (i.e., such that whenever an attribute in the set changes its value, all the others change too), is stored in a separate table so that a change occurred to one concept does not affect the other concepts. Obviously, such normalized and time-oriented structure is not suited for querying, that will take place on denormalized data marts fed from the data warehouse.

Since the data warehouse can be thought of as a set of derived, materialized views defined over a set of source schemata, the problem of evolving the content and the schema of derived views in connection to the source changes is highly relevant in the context of temporal data warehouses. Bellahsene (2002) distinguishes two subproblems: view maintenance and view adaptation.

View maintenance consists in maintaining a materialized view in response to data modifications of the source relations. Considering the width of the problem, we refer the reader to Gupta & Mumick (1995) for a taxonomy of view maintenance problems and a description of the main techniques proposed in the literature. A specific issue in view maintenance is how to provide temporal views over the history of source data, that may be non-temporal. We mention two approaches in this direction. Yang & Widom (1998) describe an architecture that uses incremental techniques to automatically maintain temporal views over non-temporal source relations, allowing users to ask temporal queries on these views. De Amo & Halfeld Ferrari Alves (2000) present a self-maintainable temporal data warehouse that, besides a set of temporal views, includes a set of auxiliary relations containing only temporal information. Such auxiliary relations are used to maintain the data warehouse without consulting the source databases and to avoid storing the entire history of source databases in the warehouse.

View adaptation consists in recomputing a materialized view in response to changes either in the schema of the source relations or in the definition of the view itself. Changes in the source schemata may be due to an evolution of the application domain they represent, or to a new physical location for them. Changes in the definition of the view (i.e., in the data warehouse schema) may also be due to new requirements of the business users who query the data marts fed by the data warehouse. Among the approaches in this direction we mention the one by Bellahsene (1998), who proposes an extended relational view model to support view adaptation, aimed at maintaining data coherence and preserving the validity of the existing application programs. Performing a schema change leads to creating a new view, by means of an extended view definition language that incorporates two clauses: *hide*, which specifies a set of attributes to be hidden, and *add*, that allows a view to own additional attributes that do not belong to source relations. In the EVE framework (Lee, Nica, & Rundensteiner, 2002), in order to automate the redefinition of a view in response to schema changes in the data sources, the database administrator is allowed to embed her preferences about view evolution into the view definition itself. The preference-based view rewriting process, called *view synchronization*, identifies and extracts appropriate information from other data sources as replacements of the affected components of the original view definition, in order to produce an alternative view that somehow preserves the original one. Finally, the DyDa framework (Chen, Zhang, & Rundensteiner, 2006) supports compensating queries, that cope with erroneous

results in view maintenance due to concurrent updates in data source, in presence of data and schema changes.

The key idea of adaptation techniques is to avoid recomputing the materialized view from scratch by relying on the previous materialization and on the source relations. For instance, Bellahsene (2002) focuses on the adaptation of the data warehouse in response to schema changes arising on source relations located on multiple sites. To adapt the extent of the data warehouse in response to these changes, she adopts rewriting algorithms that make use of containment checking, so that only the part of the new view that is not contained in the old view will be recomputed. In the same context, a distinctive feature of the *AutoMed* system (Fan & Poulovassilis, 2004) is the capability of handling not only schema evolutions in materialized data integration scenarios, but also changes in the data model in which the schema is expressed (e.g., XML vs. relational). This is achieved by applying sequences of primitive transformations to a low-level hypergraph-based data model, in whose terms higher-level modeling languages are defined.

With reference to the problem of keeping the data warehouse in sync with the sources, Wrembel and Bebel (2007) propose a metamodel for handling changes in the operational data sources, which supports the automatic detection of structural and content changes in the sources and their automatic propagation to the data warehouse.

Finally, Combi & Oliboni (2007) focus on the management of time-variant semi-structured XML data within the data warehouse. In particular, they propose a representation based on graphs whose nodes denote objects or values and are labeled with their validity interval; the constraints related to correct management of time are then discussed.

Handling DATA Changes in the Data Mart

Content changes result from user activities that perform their day-to-day work on data sources by means of different applications (Wrembel & Bebel, 2007). These changes are reflected in the data warehouse and then in the data marts fed from it.

The multidimensional model provides direct support for representing the sequence of events that constitute the history of a fact: by including a temporal dimension (say, with date granularity) in the fact, each event is associated to its date. For instance, if we consider an ORDER fact representing the quantities in the lines of orders received by a company selling PC consumables, the dimensions would probably be product, orderNumber, and orderDate. Thus, each event (i.e., each line of order) would be associated to the ordered product, to the number of the order it belongs to, and to the order date.

On the other hand, the multidimensional model implicitly assumes that the dimensions and the related levels are entirely static. This assumption is clearly unrealistic in most cases; for instance, considering again the order domain, a company may add new categories of products to its catalog while others can be dropped, or the category of a product may change in response to the marketing policy.

Another common assumption is that, once an event has been registered in a data mart, it is never modified so that the only possible writing operation consists in appending new events as they occur. While this is acceptable for a wide variety of domains, some applications call for a different behavior; for example the quantity of a product ordered in a given day could be wrongly registered or could be communicated after the ETL process has run.

These few examples emphasize the need for a correct handling of changes in the data mart content. Differently from the problem of handling schema changes, the issues related to data changes have been widely addressed by researchers and practitioners, even because in several cases they can be directly managed in commercial DBMSs. In the following subsections we separately discuss the issues related to changes in dimensional data and factual data, i.e., events.

Changes in Dimensional Data

By this term we mean any content change that may occur within an instance of a hierarchy, involving either the dimension itself, or a level, or a property. For instance, considering a product hierarchy featuring levels type and category, the name of a product may change, or a new category may be introduced so that the existing types have to be reassigned to categories.

The study of changes in dimensional data has been pioneered by Kimball (1996), who coined the term *slowly-changing dimension* to point out that, differently from data in fact tables, changes within the dimension tables occur less frequently. He proposed three basic modeling solutions for a ROLAP implementation of the multidimensional model, each inducing a different capability of tracking the history of data. In the *Type I* solution he simply proposes to overwrite old tuples in dimension tables with new data: in this case, tracking history is not possible but changes in the hierarchy data keep the data mart up-to-date. Conversely, in the *Type II* solution, each change produces a new record in the dimension table: old events stay related to the old versions of hierarchies, while new events are related to the current version. In order to allow two or more tuples representing the same hierarchy instance to be included in the dimension table, surrogate keys must necessarily be adopted. Finally, the *Type III* solution is based on augmenting the schema of the dimension table by representing both the current and the previous value for each level or attribute subject to change.

Other solutions, based on these basic ones, have been proposed over time. In particular, a complete historicization of the dimension tables determines higher expressivity. This can be obtained for instance as an extension of Type II, by adding to the dimension table schema a couple of timestamps storing the validity interval for each tuple, plus an attribute storing the surrogate key of the first version of the tuple. This solution is sometimes called *Type VI* (I+II+III) since it covers all the previous ones.

The solutions discussed so far have different querying capabilities; with reference to the terminology proposed by SAP (2000), three main querying scenarios can be distinguished:

- *Today is yesterday*: all events are related to the current value of the hierarchy. This scenario is supported by all the discussed solutions.
- *Today or Yesterday*: each event is related to the hierarchy value that was valid when the event occurred. This scenario, that reconstructs the historical truth, is supported by Type II and VI solutions.
- *Yesterday is Today*: each event is related to the hierarchy value that was valid at a given time in the past. This scenario is supported by Type VI solution only.

Other solutions for handling changes in dimensional data have been devised thereafter. Two relevant proposals, that study the problem from a more conceptual point of view, are by Bliujute et al. (1998) and Pedersen and Jensen (1999). The first one proposes a temporal star schema that, differently from the traditional one, omits the time dimension table and timestamps each row in every table instead, treating the fact table and the dimension tables equally with respect to time. Similarly, the second one proposes to handle changes by adding timestamps to all the components of a multidimensional schema: the values of both dimensions and facts, the inter-level partial order that shapes hierarchy instances and the fact-dimension relationships. Another model that supports changes in data by timestamping dimensional data is COMET (Eder, Koncilia, & Morzy, 2002), that also supports schema versioning using a fully historicized meta-model. Finally, Chamoni and Stock (1999) suggest to couple the multidimensional cube with meta-cubes that store dimension structures together with their timestamps.

A model supporting data changes should be coupled with meaningful operators to carry them out. An interesting proposal in this

direction comes from Hurtado, Mendelzon, & Vaisman (1999b), who introduces a set of high-level operators based on sequences of elemental operators (Hurtado, Mendelzon, & Vaisman, 1999a) for both schema and data changes. The operators for data changes are *reclassify,* that changes the roll-up partial order between levels, *split*, that reorganizes a hierarchy after one instance has been replaced by two or more ones, *merge*, that merges two instances of a hierarchy into a single one, and *update*, that simply changes the value of an instance without affecting the roll-up partial order. Since changes to hierarchy instances could affect summarizability, the definition of models and operators is usually coupled with a set of constraints aimed at enforcing data consistency (Hurtado, Mendelzon, & Vaisman, 1999b; Eder, Koncilia, & Morzy, 2002; Letz, Henn, & Vossen, 2002).

Changes in Factual Data

We start this section by preliminarily mentioning the two basic paradigms introduced by Kimball (1996) for representing inventory-like information in a data mart: the *transactional model*, where each increase and decrease in the inventory level is recorded as an event, and the *snapshot model*, where the current inventory level is periodically recorded. A similar characterization is proposed by Bliujute et al. (1998), who distinguish between *event-oriented data*, like sales, inventory transfers, and financial transactions, and *state-oriented data*, like unit prices, account balances, and inventory levels. This has been later generalized to define a classification of facts based on the conceptual role given to events (Golfarelli & Rizzi, 2007b):

- *Flow facts* (*flow measures* in Lenz & Shoshani, 1997) record a single transaction or summarize a set of transactions that occur during the same time interval; they are monitored by collecting their occurrences during a time interval and are cumulatively measured at the end of that period. Examples of flow facts are orders and enrollments.
- *Stock facts* (*stock measures* in Lenz & Shoshani, 1997) refer to an instant in time and are evaluated at that instant; they are monitored by periodically sampling and measuring their state. Examples are the price of a share and the level of a river.

By the term *changes in factual data* we mean any content change an event may be subject to, involving either the values of its measures or the dimensional elements it is connected to. Changes in factual data are a relevant issue in all those cases where the values measured for a given event may change over a period of time, to be consolidated only after the event has been for the first time registered in the data mart. These *late measurements* typically happens when the early measurements made for events are subject to errors (e.g., the amount of an order may be corrected after the order has been registered) or when events inherently evolve over time (e.g., notifications of university enrollments may be received and registered several days after they were issued). This problem becomes even more evident as the timeliness requirement takes more importance (Jarke, Jeusfeld, Quix, & Vassiliadis, 1999). This is the case for *zero-latency* data warehousing systems (Bruckner & Tjoa, 2002), whose goal is to allow organizations to deliver relevant information as fast as possible to knowledge workers or decision systems that need to react in near real-time to new information.

In these contexts, if the up-to-date state is to be made timely visible to the decision makers, past events must be continuously updated to reflect the incoming late measurements. Unfortunately, if updates are carried out by physically overwriting past registrations of events, some problems may arise. In fact, accountability and traceability require the capability of preserving the exact information the analyst based her decision upon. If the old registration for an event is replaced by its latest version, past decisions can no longer be justified. Besides, in some applications, accessing only up-to-date versions of information is not sufficient to ensure the correctness of analysis. A typical case is that of queries requiring to compare the progress of an

ongoing phenomenon with past occurrences of the same phenomenon: since the data recorded for the ongoing phenomenon are not consolidated yet, comparing them with past consolidated data may not be meaningful (Golfarelli & Rizzi, 2007b).

Supporting accountability and traceability in presence of late measurements requires the adoption of a bi-temporal solution where both valid and transaction time are represented by means of timestamps. Only few approaches in the literature are specifically focused on studying this specific topic. Kimball (2000) states that a bi-temporal solution may be useful to cope with late measurements. Bruckner & Tjoa (2002) discuss the problem of temporal consistency in consequence of delayed discovery of real-world changes and propose a solution based on valid time, revelation time and loading time. Loading time is the point in time when a new piece of information is loaded in the data mart, while revelation time is the point in time when that piece of information was realized by at least one data source. Finally, Golfarelli & Rizzi (2007b) propose to couple valid time and transaction time and distinguish two different solutions for managing late measurements: *delta solution*, where each new measurement for an event is represented as a delta with respect to the previous measurement, and transaction time is modeled by adding to the schema a new temporal dimension to represent when each registration was made in the data mart; and *consolidated solution*, where late measurements are represented by recording the consolidated value for the event, and transaction time is modeled by two temporal dimensions that delimit the time interval during which each registration is current.

Handling SCHEMA Changes in the Data Mart

According to (Wrembel & Bebel, 2007), schema changes in the data mart may be caused by different factors:

- Subsequent design iterations in the context of an incremental approach to data mart design.
- Changes in the user requirements, triggered for instance by the need for producing more sophisticated reports, or by new categories of users that subscribe to the data mart.
- Changes in the application domain, i.e., arising from modifications in the business world, such as a change in the way a business is done, or a changing in the organizational structure of the company.
- New versions of software components being installed.
- System tuning activities.

For instance, it may be necessary to add a subcategory level to the product hierarchy to allow more detailed analysis, or to add a measure revenueInEuro due to the introduction of a new currency.

As stated in the second section, depending on how previous schema versions are managed, two main classes of approaches may be distinguished: *schema evolution*, that allows modifications of the schema without loss of data but does not maintain the schema history, and *schema versioning*, where past schema definitions are retained so that all data may be accessed through a version specified by the user. In the two following subsection these two classes of approaches will be separately surveyed.

Evolution

The main problem here is to support a set of operators for changing the data mart schema, while enabling lossless migration of existing data from the past schema version to the new one.

In this context, FIESTA is a methodology where the evolution of multidimensional schemata is supported on a conceptual level, thus for both ROLAP and MOLAP implementations (Blaschka, Sapia, & Höfling, 1999; Blaschka, 2000). Core of the approach is a schema evolution algebra which includes a formal multidimensional data model together with a wide set

of schema evolution operations, whose effects on both schema and instances are described. Essentially, the operations allow dimensions, hierarchy levels, properties and measures to be added and deleted from the multidimensional schema. Since OLAP systems are often implemented on top of relational DBMSs, the approach also shows how a multidimensional schema can be mapped to a relational schema by means of a meta-schema that extends the catalogue of the underlying DBMS. Each sequence of evolution operations is then transformed into a sequence of relational evolution commands that adapt the relational database schema together with its instances, and update the contents of the meta-schema accordingly.

Conversely, in (Kaas, Pedersen, & Rasmussen, 2004) the evolution problem is investigated with particular reference to its impact on the logical level for ROLAP implementations, namely, on star and snowflake schemata. Eight basic evolution operators are defined (insert/delete dimension, level, property, and measure). For each of them, the changes implied on star and snowflake schemata are described and their impact on existing SQL queries in reporting tools is discussed. Remarkably, an in-depth comparison reveals that the star schema is generally more robust than the snowflake schema against schema changes.

A comprehensive approach to evolution is the one jointly devised at the Universities of Toronto and Buenos Aires. The fundamentals are laid by Hurtado, Mendelzon, & Vaisman (1999a), who propose a formal model for updating dimensions at both the schema and instance level, based on a set of modification operators (generalize, specialize, relate/unrelated/delete level are those defined at the schema level). An incremental algorithm for efficiently maintaining a set of materialized views in the presence of dimension updates is also presented. This work is then extended by Vaisman, Mendelzon, Ruaro, & Cymerman (2004) by introducing *TSOLAP*, an OLAP server supporting dimension updates and view maintenance, built following the OLE DB for OLAP proposal. The approach is completed by *MDDLX*, an extension of MDX (Microsoft's language for OLAP) with a set of statements supporting dimension update operators at both schema and instance levels.

A relevant aspect related to evolution is how changes in schema affect the data mart quality, which is discussed in (Quix, 1999). A set of schema evolution operators is adapted from those for object-oriented databases; for each operator, its impact on the quality factors (such as completeness, correctness, and consistency between the conceptual and logical schema) as emerged in the context of the *DWQ Project - Foundations of Data Warehouse Quality* (Jarke, Jeusfeld, Quix, & Vassiliadis, 1999) is discussed. The tracking of the history of changes and the consistency rules to enforce when a quality factor has to be re-evaluated due to evolution is supported by an ad hoc meta-model.

Versioning

According to the frequently cited definition by Inmon (1996), one of the characteristic features of a data warehouse is its non-volatility, which means that data is integrated into the data warehousing system once and remains unchanged afterwards. Importantly, this feature implies that the re-execution of a single query will always produce the same result. In other words, past analysis results can be verified and then inspected by means of more detailed OLAP sessions at any point in time. While non-volatility in the presence of changes at the data level can be achieved by adopting one of the solutions discussed in the third section, non-volatility in the presence of changes at the schema level requires some versioning approach to be undertaken. In fact, it is easy to see that the ability to re-execute previous queries in the presence of schema changes requires access to past schema versions, which cannot be achieved with an evolution approach.

The first work in this direction is COMET (Eder, Koncilia, & Morzy, 2002), a metamodel that supports schema and instance versioning. All classes in the metamodel are timestamped with a validity interval, so multiple, subsequent

versions of cubes can be stored and queried. Transformation of data from one version into the (immediate) succeeding or preceding one is supported; though the paper reports no details on how a new version can be obtained from the previous one, a comprehensive set of constraints that the versions have to fulfill in order to ensure the integrity of the temporal model is proposed.

The peculiarity of the timestamp-based versioning model proposed by Body, Miquel, Bédard, and Tchounikine (2003) is that hierarchies are deduced from the dimensions instances, so that explicitly defining the multidimensional schema is not necessary. In this way, schema changes are implicitly managed as a result of handling changes in instances. On the other hand, the versioning approach proposed by Ravat, Teste, & Zurfluh (2006) uses a constellation of star schemata to model different versions of the same fact, and populates versions by means of mapping functions.

A comprehensive approach to versioning is presented by Wrembel and Bebel (2007). Essentially, they propose two metamodels: one for managing a multi-version data mart and one for detecting changes in the operational sources. A multi-version data mart is a sequence of versions, each composed of a schema version and an instance version. Remarkably, besides "real" versions determined by changes in the application domain or in users' requirements, also "alternative" versions are introduced, to be used for simulating and managing hypothetical business scenarios within what-if analysis settings.

Another approach to versioning specifically oriented to supporting cross-version queries is the one by Golfarelli, Lechtenbörger, Rizzi and Vossen (2006). Here, multidimensional schemata are represented as graphs of simple functional dependencies, and an algebra of graph operations to define new versions is defined. Data migration from the old to the new version is semi-automated, i.e., based on the differences between the two versions the system suggests a set of migration actions and gives support for their execution. The key idea of this approach is to support flexible cross-version querying by allowing the designer to enrich previous versions using the knowledge of current schema modifications. For this purpose, when creating a new schema version the designer may choose to create *augmented schemata* that extend previous schema versions to reflect the current schema extension, both at the schema and the instance level. In a nutshell, the augmented schema associated with a version is the most general schema describing the data that are actually recorded for that version and thus are available for querying purposes. Like for migration, a set of possible augmentation actions is proposed to the designer (e.g., the designer may choose to manually insert values of a newly added attribute for hierarchy instances whose validity was limited to previous versions).

To the best of our knowledge, only two approaches use both valid and transaction time in the context of versioning. Koncilia (2003) presents a bi-temporal extension of the COMET metamodel, aimed at representing not only the valid time of schema modifications, but also the transaction time. Rechy-Ramírez and Benítez-Guerrero (2006) introduce a conceptual model for bi-temporal versioning of multidimensional schemata, aimed at enabling modifications in the data mart schema without affecting the existing applications. Each version has a temporal pertinence composed by a valid time and a transaction time, thus enabling the existence of two or more versions with the same valid time, but different transaction times. Associated to this model, there are 16 operators for schema changing and a SQL-like language to create and modify versions.

querying temporal data

The development of a model for temporal data warehousing is of little use without an appropriate query language capable of effectively handling time. In principle, a temporal query could be directly formulated on a relational schema using standard SQL, but this would be exceedingly long and complex even for a skilled user.

In this direction, Bliujute, Saltenis, Slivinskas, & Jensen (1998) discuss the performance of their temporal star schema considering five types of temporal queries. Golfarelli & Rizzi (2007b) distinguish three querying scenarios in presence of late measurements:

- *Up-to-date queries*, that require the most recent measurement for each event;
- *Rollback queries*, that require a past version measurement for each event;
- *Historical queries*, that require multiple measurements for events, i.e., are aimed at reconstructing the history of event changes.

To cope with schema changes, Mendelzon and Vaisman (2000) proposed the *Temporal OLAP* (TOLAP) query language. TOLAP, based on the temporal multidimensional model proposed by Hurtado et al. (1999b), fully supports schema evolution and versioning, differently from best-known temporal query languages such as TSQL2 (Snodgrass, 1995), that supports versioning in a limited way only. TOLAP combines the temporal features of TSQL2 with some high-order features of SchemaLog in order to support querying multidimensional data with reference to different instants in time in a concise and elegant way. All three querying scenarios (today is yesterday, yesterday is today, and today or yesterday) are supported. Also meta-queries, e.g. concerning the instant changes to data took place, can be expressed.

Several approaches face the problem of formulating cross-version querying, i.e., formulating queries that span different schema versions. For instance, Morzy and Wrembel (2004) propose a SQL extension aimed at expressing queries on multiple (either real or alternative) schema versions. Each query is decomposed into a set of partial queries, one for each schema version involved. The results of partial queries are separately presented, annotated with version and metadata information; in some cases, partial queries results can be merged into a common set of data. In (Wrembel & Bebel, 2007), the problem of cross-version queries is addressed by allowing users to specify either implicitly (by specifying a time interval for the query) or explicitly (by specifying a set of version identifiers) the set of versions for querying. Similarly, in (Golfarelli & Rizzi, 2007a) the relevant versions for answering a query are either chosen explicitly by the user or implicitly by the system based on the time interval spanned by the query, as shown in the prototype implementation X-Time.

In the context of querying, a number of works are related to the so-called *temporal aggregation problem*, that was studied mainly in the context of MOLAP systems and consists in efficiently computing and maintaining temporal aggregates. In fact, time dimensions typically lead to a high degree of sparseness in traditional array-based MOLAP cubes because of their large cardinality, and to significant overhead to answer time-parameterized range queries. For instance, the work by Tao, Papadias, & Faloutsos (2004) focuses on approximate temporal aggregate processing. Specifically, for count queries, its goal is to provide answers guaranteed to deviate from the exact ones within a given threshold. Riedewald, Agrawal, & El Abbadi (2002) proposed efficient range aggregation in temporal data warehouses by exploiting the append-only property of the time-related dimension. Their framework allows large amounts of new data to be integrated into the warehouse and historical summaries to be efficiently generated, independently of the extent of the data set in the time dimension. Feng, Li, Agrawal, & El Abbadi (2005) proposed a general approach to improve the efficiency of range aggregate queries on MOLAP data cubes in a temporal data warehouse by separately handling time-related dimensions to take advantage of their monotonic trend over time. Finally, Yang & Widom (2001) introduce a new index structure called the SB-tree, which supports fast lookup of aggregate results based on time, and can be maintained efficiently when the data changes along the time line.

Designing Temporal Data Warehouses

It is widely recognized that designing a data warehousing system requires techniques that are radically different from those normally adopted for designing operational databases (Golfarelli & Rizzi, 1999). On the other hand, though the literature reports several attempts to devise design methodologies for data warehouses, very few attention has been posed on the specific design issues related to time. Indeed, as stated by Rizzi et al. (2006), devising design techniques capable of taking time and changes into account is one of the open issues in data warehouse research.

Pedersen and Jensen (1999) recognize that properly handling time and changes is a must-have for multidimensional models. Sarda (1999) summarizes the distinguishing characteristics of time dimensions: they are continuously valued and constantly increasing, they can be associated with multiple user-defined calendars, they express the validity of both facts and other dimensions (either in the form of time instants or validity intervals). Sarda also proposes a design methodology for temporal data warehouses featuring two phases: logical design, that produces relations characterized by a temporal validity, and physical design, that addresses efficient storage and access.

Considering the leading role played by temporal hierarchies within data marts and OLAP queries, it is worth adopting ad hoc approaches for their modeling not only from the logical, but also from the conceptual point of view. While all conceptual models for data marts allow for temporal hierarchies to be represented like any other hierarchies, to the best of our knowledge the only approach that provides ad hoc concepts for modeling time is the one by Malinowski & Zimányi (2008), based on a temporal extensions of the MultiDim conceptual model. Different temporality types are allowed (namely, valid time, transaction time, lifespan, and loading time), and temporal support for levels, properties, hierarchies, and measures is granted.

Finally, Golfarelli & Rizzi (2007b) discuss the different design solutions that can be adopted in presence of late measurements, depending on the flow or stock nature of the events and on the types of queries to be executed.

OPEN ISSUES AND CONCLUSIONS

In this survey we classified and discussed the issues related to temporal data warehousing. An in-depth analysis of the literature revealed that the research community not always devoted a comprehensive attention to all these aspects. As a matter of fact, a wide agreement on the possible design solutions has been reached only with reference to changes in dimensional data. As to changes in factual data and changes in schema, though some interesting solutions have been proposed, no broad and shared framework has been devised yet.

Similarly, on the commercial side, changes in data have been supported since almost a decade ago. Already in year 2000, systems such as Business Warehouse by SAP (2000) were allowing to track changes in data and to effectively query cubes based on different temporal scenarios by letting users choose which version of the hierarchies to adopt for querying. On the other hand, today there still is very marginal support to changes in schema by commercial tools. For instance, *SQL Compare* compares and synchronizes SQL Server database schemata, and can be used when changes made to the schema of a local database need to be pushed to a central database on a remote server. Also, the *Oracle Change Management Pack* is aimed to report and track the evolving state of meta-data, thus allowing to compare database schemata, and to generate and execute scripts to carry out the changes. In both cases, formulating a single query spanning multiple databases with different schemata is not possible.

We believe that, considering the maturity of the field and the wide diffusion of data warehousing systems, in the near future decision makers will be more and more demanding for

advanced temporal support. Thus, it is essential that both vendors and researchers be ready to deliver effective solutions. In this direction we envision two main open issues. On the one hand, some research aspects indeed require further investigation. For instance, support for cross-version queries is not satisfactory yet, and its impact on performance has not been completely investigated; similarly, the effectiveness of view adaptation approaches is still limited. On the other hand, in order to encourage vendors to add full temporal support to commercial platforms, the solutions proposed in the literature should be better harmonized to converge into a complete, flexible approach that could be effortlessly accepted by the market.

REFERENCES

Abelló, A., & Martín, C. (2003). A Bi-temporal Storage Structure for a Corporate Data Warehouse. *Proceedings International Conference on Enterprise Information Systems*, Angers, France, 177-183.

Bellahsene, Z. (1998). View Adaptation in Data Warehousing Systems. *Proceedings International Conference on Database and Expert Systems Applications*, Vienna, Austria, 300-309.

Bellahsene, Z. (2002). Schema Evolution in Data Warehouses. *Knowledge and Information Systems*, 4(3), 283-304.

Blaschka, M. (2000). *FIESTA - A Framework for Schema Evolution in Multidimensional Databases*. PhD Thesis, Technische Universitat Munchen, Germany.

Blaschka, M., Sapia, C., & Höfling, G. (1999). On Schema Evolution in Multidimensional Databases. *Proceedings International Conference on Data Warehousing and Knowledge Discovery*, Florence, Italy, 153-164.

Bliujute, R., Saltenis, S., Slivinskas, G., & Jensen, C. S. (1998). Systematic Change Management in Dimensional Data Warehousing. *Proceedings International Baltic Workshop on Databases and Information Systems*, Riga, Latvia, 27–41.

Body, M., Miquel, M., Bédard, Y., & Tchounikine, A. (2003). Handling Evolutions in Multidimensional Structures. *Proceedings International Conference on Data Engineering*, Bangalore, India, 581-591.

Bruckner, R., & Tjoa, A. (2002). Capturing Delays and Valid Times in Data Warehouses - Towards Timely Consistent Analyses. *Journal of Intelligent Information Systems*, 19(2), 169-190.

Chamoni, P. & Stock, S. (1999). Temporal Structures in Data Warehousing. *Proceedings International Conference on Data Warehousing and Knowledge Discovery*, Florence, Italy, 353-358.

Chen, S., Zhang, X., & Rundensteiner, E. (2006). A Compensation-Based Approach for View Maintenance in Distributed Environments. *IEEE Transactions of Knowledge and Data Engineering*, 18(8), 1068-1081.

Combi, C. & Oliboni, B. (2007). Temporal semistructured data models and data warehouses. In *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, Wrembel & Koncilia (Eds.), IRM Press, 277-297.

De Amo, S., Halfeld Ferrari Alves, M. (2000). Efficient Maintenance of Temporal Data Warehouses. *Proceedings International Database Engineering and Applications Symposium*, Yokohoma, Japan 188-196.

Devlin, B. (1997). Managing Time In The Data Warehouse. *InfoDB*, 11(1), 7-12.

Eder, J., & Koncilia C. (2001). Changes of Dimension Data in Temporal Data Warehouses. *Proceedings International Conference on Data Warehousing and Knowledge Discovery*, Munich, Germany, 284-293.

Eder, J., Koncilia, C., & Morzy, T. (2002). The COMET Metamodel For Temporal Data Warehouses. *Proceedings International Conference on Advanced Information Systems Engineering*, Toronto, Canada, 83-99.

Fan, H., & Poulovassilis, A. (2004). Schema Evolution in Data Warehousing Environments - A Schema Transformation-Based Approach. *Proceedings International Conference on Conceptual Modeling*, Shanghai, China, 639-653.

Feng, Y., Li, H.-G., Agrawal, D., & El Abbadi, A. (2005). Exploiting Temporal Correlation in Temporal Data Warehouses. *Proceedings International Conference on Database Systems for Advanced Applications*, Beijing, China, 662-674.

Golfarelli, M. & Rizzi, S. (1999). Designing the data warehouse: key steps and crucial issues. *Journal of Computer Science and Information Management*, 2(1), 1-14.

Golfarelli, M., Lechtenbörger, J. Rizzi, S., & Vossen, G. (2006). Schema Versioning in Data Warehouses: Enabling Cross-Version Querying via Schema Augmentation. *Data and Knowledge Engineering*, 59(2), 435-459.

Golfarelli, M. & Rizzi, S. (2007a). X-Time: Schema Versioning and Cross-Version Querying in Data Warehouses. *Proceedings International Conference on Data Engineering*, Istanbul, Turkey, 1471-147.

Golfarelli, M. & Rizzi, S. (2007b). Managing late measurements in data warehouses. *International Journal of Data Warehousing and Mining*, 3(4), 51-67.

Grandi, F. (2002). A Relational Multi-Schema Data Model and Query Language for full Support of Schema Versioning. *Proceedings SEBD*, Portoferraio, Italy, 323-336.

Gupta, A., & Mumick, I. S. (1995). Maintenance of materialized views: problems, techniques, and applications. *Data Engineering Bulletin*, 18(2), 3-18.

Hurtado, C., Mendelzon, A., & Vaisman, A. (1999a). Maintaining Data Cubes under Dimension Updates. *Proceedings International Conference on Data Engineering*, Sydney, Austrialia, 346-355.

Hurtado, C., Mendelzon A., & Vaisman A. (1999b). Updating OLAP Dimensions. *Proceedings International Workshop on Data Warehousing and OLAP*, Kansas City, USA, 60-66.

Inmon, W. (1996). *Building the data warehouse*. John Wiley & Sons.

Jarke, M., Jeusfeld, M., Quix, C., & Vassiliadis, P. (1999). Architecture and Quality in Data Warehouses: An Extended Repository Approach. *Information Systems*, 24(3), 229–253.

Jensen, C., Clifford, J., Elmasri, R., Gadia, S. K., Hayes, P. J., & Jajodia, S. (1994). A Consensus Glossary of Temporal Database Concepts. *ACM SIGMOD Record*, 23(1), 52-64.

Kaas, C., Pedersen, T. B., & Rasmussen, B. (2004). Schema Evolution for Stars and Snowflakes. *Proceedings International Conference on Enterprise Information Systems*, Porto, Portugal, 425-433.

Kimball, R. (1996). *The Data Warehouse Toolkit*. Wiley Computer Publishing.

Kimball, R. (2000). Backward in Time. *Intelligent Enterprise Magazine*, 3(15).

Koncilia, C. (2003). A Bi-Temporal Data Warehouse Model. *Short Paper Proceedings Conference on Advanced Information Systems Engineering*, Klagenfurt/Velden, Austria.

Lee, A., Nica, A., & Rundensteiner, E. (2002). The EVE Approach: View Synchronization in Dynamic Distributed Environments. *IEEE Transactions on Knowledge and Data Engineering*, 14(5), 931-954.

Lenz, H. J. & Shoshani, A. (1997). Summarizability in OLAP and Statistical Databases. *Proceedings Statistical and Scientific Database Management Conference*, Olympia, US, 132-143.

Letz, C., Henn, E., & Vossen, G. (2002). Consistency in Data Warehouse Dimensions. *Proceedings International Database Engineering and Application Symposium*, Edmonton, Canada, 224-232.

Malinowski, E. & Zimányi, E. (2008). A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models. *Data & Knowledge Engineering*, 64, 101-133.

McKenzie, E., & Snodgrass, R. (1990). Schema Evolution and the Relational Algebra. *Information Systems*, 15(2), 207-232.

Mendelzon, A., & Vaisman, A. (2000). Temporal queries in OLAP. *Proceedings Conference on Very Large Data Bases*, Cairo, Egypt, 242-253.

Morzy, T. & Wrembel, R. (2004). On querying versions of multiversion data warehouse. *Proceedings International Workshop on Data Warehousing and OLAP*, Washington, DC, 92-101.

Pedersen, T. B., & Jensen, C. (1998). Research Issues in Clinical Data Warehousing. *Proceedings Statistical and Scientific Database Management Conference*, Capri, Italy, 43-52.

Pedersen, T. B. & Jensen, C. (1999). Multidimensional Data Modeling for Complex Data. *Proceedings International Conference on Data Engineering*, Sydney, Austrialia, 336-345.

Quix, C. (1999). Repository Support for Data Warehouse Evolution. *Proceedings International*

Workshop on Design and Management of Data Warehouses, Heidelberg, Germany.

Ravat, F., Teste, O., & Zurfluh, G. (2006). A Multiversion-Based Multidimensional Model. *Proceedings International Conference on Data Warehousing and Knowledge Discovery*, 65-74.

Rechy-Ramírez, E.-J. & Benítez-Guerrero, E. (2006). A Model and Language for Bi-temporal Schema Versioning in Data Warehouses. *Proceedings International Conference on Computing*, Mexico City, Mexico.

Riedewald, M., Agrawal, D. & El Abbadi, A. (2002). Efficient integration and aggregation of historical information. *Proceedings SIGMOD Conference*, Madison, Wisconsin, 13-24.

Rizzi, S., Abelló, A., Lechtenbörger, J., & Trujillo, J. (2006). Research in Data Warehouse Modeling and Design: Dead or Alive? *Proceedings International Workshop on Data Warehousing and OLAP*, Arlington, USA, 3-10.

Roddick, J. (1995). A Survey of Schema Versioning Issues for Database Systems. *Information and Software Technology*, 37(7), 383-393.

SAP Institute (2000). *Multi-dimensional Modeling with SAP BW*. SAP America Inc. and SAP AG.

Sarda, N. L. (1999). Temporal Issues in Data Warehouse Systems. *Proceedings International Symposium on Database Applications in Non-Traditional Environments*, Kyoto, Japan, 27-34.

Tansel, A. U., Clifford, J., Gadia, S. K., Jajodia, S., Segev, A., & Snodgrass, R. T. (1993). *Temporal databases: theory, design and implementation*. Benjamin Cummings.

Snodgrass, R. T. (1995). *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers.

Tao, Y., Papadias, D., & Faloutsos, C. (2004). Approximate Temporal Aggregation. *Proceedings International Conference on Data Engineering*, Boston, Massachusetts, 190-201.

Vaisman, A., Mendelzon, A., Ruaro, W., & Cymerman, S. (2004). Supporting Dimension Updates in an OLAP Server. *Information Systems*, 29, 165-185.

Vaisman, A., & Mendelzon, A. (2001). A Temporal Query Language for OLAP: Implementation and a Case Study. *Proceedings DBPL*.

Wrembel, R. & Bebel, B. (2007). Metadata Management in a Multiversion Data Warehouse. *Journal of Data Semantics*, 8, 118-157.

Yang, J. & Widom, J. (1998). Maintaining Temporal Views over Non-Temporal Information Sources for Data Warehousing. *Proceedings International Conference on Extending Database Technology*, Valencia, Spain, 389-403.

Yang, J. & Widom, J. (2001). Incremental Computation and Maintenance of Temporal Aggregates. *Proceedings International Conference on Data Engineering*, Heidelberg, Germany, 51-60.

Matteo Golfarelli received his PhD for his work on autonomous agents in 1998. In 2000 he joined the University of Bologna as a researcher. Since 2005 he is associate professor, teaching information systems and database systems. He has published over 60 papers in refereed journals and international conferences in the fields of data warehousing, pattern recognition, mobile robotics, multi-agent systems. He served in the PC of several international conferences and as a reviewer in journals. His current research interests include all the aspects related to business intelligence and data warehousing, in particular multidimensional modeling, what-if analysis and BPM.

Stefano Rizzi received his PhD in 1996 from the University of Bologna, Italy. Since 2005 he is full professor at the University of Bologna, where he is the head of the Data Warehousing Laboratory. He has published about 100 papers in refereed journals and international conferences mainly

in the fields of data warehousing, pattern recognition, and mobile robotics. He joined several research projects on the above areas and has been involved in the PANDA thematic network of the European Union concerning pattern-base management systems. His current research interests include data warehouse design and business intelligence, in particular multidimensional modeling, data warehouse evolution, OLAP preferences and what-if analysis.