IGI PUBLISHING **ITJ3904**

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.igi-pub.com

This paper appears in the publication, **International Journal of Data Warehousing and Mining**, Volume 3, Issue 4
edited by **David Taniar© 2007, IGI Global**

Managing Late Measurements In Data Warehouses

Matteo Golfarelli, University of Bologna, Italy

Stefano Rizzi, University of Bologna, Italy

ABSTRACT

Though in most data warehousing applications no relevance is given to the time when events are recorded, some domains call for a different behavior. In particular, whenever late measurements of events take place, and particularly when the events registered are subject to further updates, the traditional design solutions fail in preserving accountability and query consistency. In this article, we discuss the alternative design solutions that can be adopted, in presence of late measurements, to support different types of queries that enable meaningful historical analysis. These solutions are based on the enforcement of the distinction between transaction time and valid time within the schema that represents the fact of interest. Besides, we provide a qualitative and quantitative comparison of the solutions proposed, aimed at enabling well-informed design decisions.

Keywords: *database design; data warehouse; spatiotemporal database; timeliness of information*

INTRODUCTION

Time is commonly understood as a key factor in data warehousing systems since the decisional process often relies on computing historical trends and on comparing snapshots of the enterprise taken at different moments. Within the multidimensional model, time is usually a dimension of analysis; thus, the representation of the history of fact values across a given lapse of time, at a given granularity, is directly supported. For instance, in a relational implementation for the sales domain, for each day there will be a set of rows in the fact table reporting the values of fact QuantitySold on that day for different products and stores. On the other hand, although the multidimensional model does not inherently represent the history of values for dimensions and their properties, some ad hoc techniques were devised to support the so-called *slowly-changing dimensions* (Kimball, 1996). In both cases, time is commonly meant as valid time in the terminology of temporal databases (Jensen et al., 1994) (i.e., it is meant as the time when an event or change *occurred* in the business domain) (Devlin, 1997). Transaction time, meant as the time when the event or change was *registered* in the data warehouse, is typically given little

or no importance since it is not considered to be relevant for decision support.

One of the underlying assumptions in data warehouses is that, once an event has been registered (under the form of a row in the fact table), it is never modified so that the only possible writing operation consists in appending new events (rows) as they occur. While this is acceptable for a wide variety of domains, some applications call for a different behavior. In particular, 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 warehouse. This typically happens when the early measurements made for events may be 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).

In this context, if the up-to-date situation is to be made timely visible to the decision makers, past events must be continuously updated to reflect the incoming data. Unfortunately, if updates are carried out by physically *overwriting* past registrations of events, some problems may arise:

- Accountability and traceability require the capability of preserving the exact information the analyst based his or her decision upon. If the old registration for an event is replaced by its latest version, past decisions can no longer be justified.
- 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.

Remarkably, the same problems may arise when events are registered in the data warehouse only once, but with a significant delay with respect to the time when they occurred in the application domain (e.g., there may be significant delays in communicating the daily price of listed shares on the stock market): no update is necessary in this case, yet valid time is not sufficient to guarantee accountability. Thus, in more general terms, we will use term *late measurement* to denote any measurement of an event that is sensibly delayed with respect to the time when the event occurs in the application domain; a late measurement may either imply an update to a previous measurement (as in the case of late corrections to orders) or not (as in the case of shares).

In this article, we discuss and compare the design solutions that can be adopted, in presence of late measurements, to enable meaningful historical analysis aimed at preserving accountability and consistency. These solutions are based on the enforcement of the distinction between transaction time and valid time within the schema that represents the fact of interest.

The rest of the article is organized as follows. In the second and third sections, respectively, we survey the related literature and present the working examples. In the fourth section, we distinguish two possible semantics for facts and give definitions of events and registrations. In the fifth section, we distinguish three basic categories of queries from the point of view of their temporal requirements in presence of late measurements. The sixth and seventh sections introduce, respectively, two classes of design solutions: monotemporal and bitemporal, that are then quantitatively compared in the eighth section. The ninth section concludes by discussing the applicability of the solutions proposed.

RELATED LITERATURE

Several works concerning temporal data warehousing can be found in the literature. Most of them are related to consistently managing updates in dimension tables of relational data warehouses—the so-called *slowly-changing*

dimensions (e.g., Letz, Henn, & Vossen, 2002; Yang, 2001). Some other works tackle the problem of temporal evolution and versioning of the data warehouse schema (Bebel, Eder, Koncilia, Morzy, & Wrembel, 2004; Blaschka, Sapia, & Höfling, 1999; Eder, Koncilia, & Morzy, 2002; Golfarelli, Lechtenbörger, Rizzi, & Vossen, 2006; Quix, 1999). All these works are not related to ours since there is no mention of the opportunity of representing transaction time in data warehouses in order to allow accountability and traceability for late measurements.

Devlin (1997) distinguishes between *transient data*, that do not survive updates and deletions, and *periodic data*, that are never physically deleted from the data warehouse. Kimball (1996) introduces two basic paradigms for representing inventory-like information in a data warehouse: 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. This is then generalized to define a classification of facts based on the conceptual role given to events:

- For a *transactional fact*, each event may either record a single transaction or summarize a set of transactions that occur during the same time interval. Most measures are flow measures (Lenz & Shoshani, 1997): they refer to a time interval and are cumulatively evaluated at the end of that period.
- For a *snapshot fact*, events correspond to periodical snapshots of the fact. Measures are mostly stock measures (Lenz et al., 1997): they refer to an instant in time and are evaluated at that instant.

A similar characterization is proposed by Bliujute, Saltenis, Slivinskas, and Jensen (1998), who distinguish between *state-oriented data* like sales, inventory transfers, and financial transactions, and *event-oriented data*, like unit prices, account balances, and inventory levels. Both distinctions are relevant to our approach and are recalled in the fourth section. Bliujute et al. (1998) also propose a *temporal star schema* that incorporates timestamps into the fact table to model valid time; though such schema is somehow related to the design solutions we propose, it does not take transaction time into consideration and is not analyzed in the light of the late measurements problem.

Pedersen and Jensen (1998) recognize the importance of advanced temporal support in data warehouses, with particular reference to medical applications. Abelló and Martín (2003a) claim that there are important similarities between temporal databases and data warehouses, suggest that both valid time and transaction time should be modeled within data warehouses, and mention the importance of temporal queries. Finally, Abelló and Martín (2003b) propose a storage structure for a bitemporal data warehouse (i.e., one supporting both valid and transaction time). All these approaches suggest the importance of transaction time in data warehouses, but not with explicit reference to the problem of late measurements.

Kimball (2000) raises the problem of late-arriving fact records, generically stating that a bitemporal solution may be useful to cope with them. In the same direction, Bruckner and Tjoa (2002) discuss the problem of data warehouse temporal consistency in consequence of delayed discovery of real-world changes and propose a solution based on transaction time (which they call *revelation time*) and overlapped valid time. Although the article discusses some issues related to late measurements, no emphasis is given to the influence that the semantics of the captured events and the querying scenarios pose on the feasibility of the different design solutions.

WORKING EXAMPLES

In this section, we propose three examples that justify the need for managing late measurements and will be used in the rest of the article to discuss and compare the different design solutions.

In the first example, late measurements (with updates) are motivated by the fact that the represented events inherently evolve over

time. Consider a fact modeling the number of enrollments to university degrees; in a relational implementation, a simplified fact table for enrollments could have the following schema (we intentionally do not introduce surrogate keys in the fact table in order to avoid to unnecessarily complicate the examples):

FT_ENROLL(EnrollDate, Degree, AcademicYear, Number)

where EnrollDate is the formal enrollment date (the one reported on the enrollment form). An enrollment is acknowledged by the University secretariat only when the entrance fee is paid; considering the variable delays due to the bank processing and transmitting the payment, the enrollment may be communicated and stored in the university database—and, from there, loaded into the data warehouse—even one month after the enrollment has been done. This is a case of late measurements. Besides: (i) notices of payments for the same enrollment date are spaced out over long periods, and (ii) after paying the fee, students may decide to switch their enrollment from one degree to another. Thus, updates are necessary in order to correctly track enrollments.

The main reason why, in this example, the enrollment date may not be sufficient is related to the soundness of analysis. In fact, most queries on this fact will ask for evaluating the current trend of the number of enrollments as compared to last year. But if the current, partial data on enrollments were compared to the consolidated ones at exactly one year ago, the user would wrongly infer that this year we are experiencing a negative trend for enrollments!

The second example is related to a fact representing the quantities in the lines of orders received by a company selling PC consumables, according to the following schema:

FT_ORDER(OrderNumber, OrderDate, Product, Quantity).

Though the first registration of an order may not involve notable delays, the orders received may be subject to later corrections, which implies late measurements.

The third example, motivated by the delay in communicating information, is that of a fact monitoring the price of listed shares on the stock market:

FT_SHARE(Date, Share, Price).

We assume that this fact is daily fed by importing a file that reports the current quotations; occasional delays in communicating the daily prices will produce late measurements, which in turn will raise problems with justifying the decisions made on previous reports.

EVENTS AND REGISTRATIONS

The aim of this section is to introduce the classification of events and registrations on which we will rely in the next sections to discuss the applicability of the design solutions proposed.

In general terms, the facts to be monitored for decision support fall into two broad categories according to the way they are measured in the application domain. *Flow facts* (called *flow measures* in Lenz et al., 1997) 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 order quantity and number of enrollments. *Stock facts* (called *stock measures* in Lenz et al., 1997) are monitored by periodically sampling and measuring their state; examples of stock facts are the price of a share and the level of a river.

Definition 1 (Event): Given fact F, we call events the results of the monitoring of F. Each event is identified by a set of coordinates, i.e., values for the dimensions of analysis of F. We call the valid time of event e_i the instant vt_i when e_i takes place in the application domain. Event e_i yields a non empty sequence of measurements m_{ij}, $j = 1,\ldots,n_i$ ($n_i \geq 1$).

Each new measurement for an event provides a revised value, typically more accurate than the previous one. Obviously, in order to

avoid information loss, each measurement received for an event must be recorded into the data warehouse, which is done when next refresh takes place.

Definition 2 (Registration): Given fact F and a measurement m_{ij} for event e_i, we call registration r_{ij} for m_{ij} the recording[1] of m_{ij} on the data warehouse, done on transaction time tt_{ij} ($tt_{ij} \geq vt_i$). For each i, r_{i1} is called the first registration for e_i and the r_{ij}'s with $j > 1$ (if any) are called the update registrations for e_i. Given $t \geq vt_i$, the current registration for e_i at time t is the one done on transaction time tt_{ij^*} where $j^* = max\{j \mid tt_{ij} \leq t\}$.

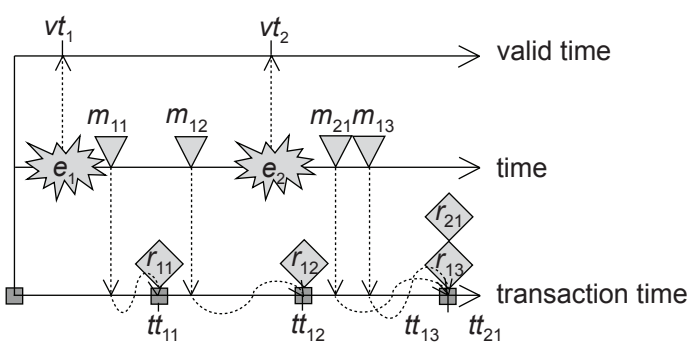
With reference to our working examples:

- An event for the (flow) order fact measures the quantity ordered for a given product within a given order issued on a given date (its coordinates). In this case, each event corresponds to a single order line (no aggregation is done) and its valid time is the order date. The first registration of each event is done when the related order is received; an update registration may arise if the ordering customer asks for modifying a quantity in her order.
- An event for the (flow) enrollment fact measures the net number of enrollments made on a given date for a given degree and academic year (its coordinates). In this case, each event aggregates a set of enrollments and its valid time is the enrollment date; after the first registration, a sequence of update registrations is typically made for each event as new data on enrollments made on previous dates are received.
- An event for the (stock) share fact is the observation, made on a given date (valid time), of the price for a given share (date and share are the event coordinates). One single (first) registration is commonly made in this case for each event.

The delay between the time when a measurement is received by the operational database and the transaction time of the corresponding registration in the data warehouse depends on the duration of the *refresh interval* of the data warehouse (i.e., on the time between two consecutive periodical refreshes (typically ranging between 1 and 7 days)). Since, from the point of view of a data warehouse user, a measurement is known only when it is registered, in the following we will assume that *each measurement is synchronous with its registration*.

To clarify this point, Figure 1 shows an example where two events are characterized, respectively, by three and one measurements. The central axis represents the flow of time,

Figure 1. Late and non-late registrations; stars, triangles, diamonds, and squares stand for events, measurements, registrations, and data warehouse refreshes, respectively



and reports the instants when events take place and their measurements are received by the operational database. Such instants are then projected on the other two axes that represent, respectively, valid and transaction time. The instant when an event takes place is its valid time (e.g., vt_1 is the valid time for e_1). Each measurement is registered in the data warehouse not immediately when it is received by the operational database, but when next refresh takes place: for instance, measurement m_{12} is registered only at time tt_{12}, which becomes its transaction time.

Definition 3 (Late measurement): Given fact F and event e_i, measurement m_{ij} is said to be late when its transaction time tt_{ij} is later than the time of the first refresh made after vt_i. Fact F is subject to late measurements when it yields at least one late measurement. In particular, it is subject to late measurements without updates when $n_i = 1$ for every i (i.e., no update registrations are made), and subject to late measurements with updates when it is $n_i > 1$ for at least one i (i.e., at least one update registration is made for at least one event).

With reference to Figure 1, the first measurement m_{11} is not late since it is registered at the end of the refresh interval where e_1 took place. The other three measurements are late: for instance, m_{21} is late since its transaction time tt_{21} is later than tt_{12}, which is the time of the first refresh after e_2.

QUERYING SCENARIOS

From a conceptual point of view, as understood in the previous section, for every fact two different temporal dimensions may be distinguished. The first one refers to the time when events actually *take place* in the application domain, while the second one refers to the time when their measurements are *perceived and recorded* in the data warehouse. Consistently with the literature on temporal databases (Jensen et al., 1994), we called these two dimensions, respectively, valid time and transaction time.

While we take for granted that valid time must always be represented in registrations, since it is a mandatory coordinate for characterizing events, the need for representing also transaction time depends on the nature of the expected workload. From this point of view, three types of queries can be distinguished (the terminology is inspired by Kim and Kim, 1997):

Definition 4 (Types of queries: Given the fact F and query q that involves the set of events E_q, we will say q is an up-to-date query when for each event $e_i \in E_q$ only the now-current measurement is required; q is a rollback query when, given time t, for each event $e_i \in E_q$ only the current measurement at time t is required; q is a historical query when, given time interval $T = [t_1, t_2]$, for each event $e_i \in E_q$ all the measurements that were current at any $t \in T$ are required.

Intuitively, up-to-date queries require the most recent measurements for events. An example of up-to-date query on the enrollment fact is the one asking for the daily number of enrollments to a given degree made during last week. In fact, this query is solved correctly by considering the most up-to-date data available for the number of enrollments by enrollment dates. Registering transaction time is not necessary to solve this kind of queries, since they rely on valid time only. For a fact not subject to late measurements, all possible queries belong to this category, since the first registration for each event is not significantly delayed with respect to the event valid time, and no update registrations take place.

On the other hand, a rollback query requires a past measurement for each event. Consider for instance the query asking for the current trend of the total number of enrollments for each degree as compared to last year. In order to get consistent results, the comparison must be founded on registration dates rather than enrollment dates. Thus, this kind of query requires that transaction time is explicitly represented in registrations.

Finally, historical queries require multiple measurements for events. An example of historical query is the one asking for the daily distribution of the number of enrollments received for a given enrollment date. Also these queries require transaction time to be represented explicitly.

Depending on the presence of late measurements and on the composition of the expected workload, two main types of design solution can be envisaged for a fact: *monotemporal*, where only valid time is modeled as a dimension of analysis; *bitemporal*, where both valid and transaction time are modeled as dimensions of analysis. The details of these two types of solutions are discussed in following sections.

MONOTEMPORAL SOLUTIONS

Monotemporal solutions are those commonly implemented for facts that either are not subject to late measurements or are only required to support up-to-date queries. They are the simplest solutions: update registrations are done by physically overwriting the registrations made at previous times for the same event, so that one single registration (the most recent one) is kept in the database for each event. The transaction times of registrations are not represented and no trace is left of past measurements so only up-to-date queries are supported and, in case of late measurements, accountability is not guaranteed. For instance, the schema of the monotemporal solutions for the order, the enrollment, and the share facts are exactly the ones already shown in the third section, where the only temporal dimensions are, respectively, OrderDate, EnrollDate, and Date (valid times).

Discussing in detail how registrations are made in a monotemporal solution requires to clarify that, from the semantic point of view, all registrations conform to one of two models: *flow registrations*, that are additive along all dimensions of analysis (i.e., the fact values can always be summed when aggregating), and *stock registrations*, that are non-additive along temporal dimensions (i.e., the fact values cannot be summed when aggregating along time, while for instance they can be averaged). While in a bitemporal solution, as we will see in the seventh section, two different models may be adopted for first registrations and update registrations, in a monotemporal solution all registrations share the same model. Thus, we will call *flow solution* the one relying on flow registrations, and *stock solution* the one relying on stock registrations. The choice of a flow or stock solution is influenced by the core workload the fact is subject to, but mainly depends on the category (flow or stock) of the fact, as clarified in the next subsections.

Flow Facts

For a flow fact, the flow solution is typically the most natural choice. Each event e_i is represented by a single flow registration, associated to valid time vt_i, reporting the value of the last measurement for e_i. An update registration for an event is made by physically replacing the previous registration for the same event. For instance, for the order quantity fact, the flow registrations might be those reported in Table 1, each representing an event, i.e. a single line of an order. When an update measurement is received, due to a correction to a given order

Table 1. Flow solution for the order line fact

OrderNumber	OrderDate	Product	Quantity
11001	Mar. 15, 2007	CD-R	100
11001	Mar. 15, 2007	DVD+R	20
11203	Apr. 15, 2007	CD-RW	80
11203	Apr. 15, 2007	DVD+R	25

line, the registration corresponding to that order line is overwritten and the new value for the fact is reported.

On the other hand, for some flow facts both flow and stock solutions can be reasonably used. This is true, for instance, for the enrollment fact introduced in the third section. A sample set of flow registrations for enrollments is shown in Table 2. Each registration records the number of students who enrolled, on a given date, to a given degree course for a given academic year (i.e., the *exact* value of the measurement). Table 3 shows how the same events could be represented by stock registrations: here each registration records, at a given date, the *total* number of students who enrolled to a given degree course for a given academic year so far. In this case, for instance, the first registration for the event with valid time October 22 is made by summing the (flow) measurement 2 for that event to the stock registration 5 for the event with valid time October 21.

The choice of one or another solution for a flow fact depends first of all on the expected workload, and in particular on the relative weight of queries asking for flow and stock information respectively. For instance, the total number of enrollments for an academic year can be obtained in the flow solution by summing up all pertinent registrations, which may be costly, while in the stock model it is sufficient to read a single registration (the most recent one). On the other hand, consider a query asking for the number of enrollments to Electrical Engineering made on October 22. While in the flow solution such query is answered by reading one registration (the one reporting Number = 2 in Table 2), in the stock solution the result must be computed as the difference between the values of Number registered in two consecutive days.

The main factor to be considered before choosing to use a stock solution for a flow fact is whether events are subject to updates. In fact, in this case, after each update all the related stock registrations would have to be updated accordingly, which may become quite costly. For instance, suppose that on October 23 it is known that the number of enrollments made on October 21 is not 5 but 8. While in the flow solution it is sufficient to modify the registration dated October 21, in the stock solution also all the registrations for the following dates should be updated by adding 3.

Table 2. Flow solution for the enrollment fact

EnrollDate	Degree	AcademicYear	Number
Oct. 21, 2005	Elec. Eng.	05/06	5
Oct. 22, 2005	Elec. Eng.	05/06	2
Oct. 23, 2005	Elec. Eng.	05/06	3

Table 3. Stock solution for the enrollment fact

EnrollDate	Degree	AcademicYear	Number
Oct. 21, 2005	Elec. Eng.	05/06	5
Oct. 22, 2005	Elec. Eng.	05/06	7
Oct. 23, 2005	Elec. Eng.	05/06	10

Stock Facts

Differently from flow facts, stock facts naturally conform to the stock solution; for instance, a sample set of stock registrations for the share fact is reported in Table 4: like in the flow solution applied to a flow fact, each registration records the exact value of the measurement. In principle, adopting a flow solution for a stock fact is still possible, although not recommended. In fact, it would require disaggregating the (stock) measurements made in the application domain into a net flow to be registered, which implies that, before each new flow registration can be made, the current stock level must be computed by aggregating all registrations for the previous events. For instance, Table 5 shows the flow registrations corresponding to the stock registrations in Table 4. Registering as –2 the measurement 10 made on January 9, requires to first compute the stock level 12, valid on January 8, which can only be done by accessing all previous registrations.

BITEMPORAL SOLUTIONS

These are the most general solutions to be adopted in presence of late measurements and they allow all three types of queries to be correctly answered. On each refresh cycle, new update registrations for previous events may be added, and their transaction time is traced; no overwriting of previous registrations is carried out, thus no measurement is lost.

In a bitemporal solution, we distinguish between the model (flow or stock) adopted for the first registrations of events and that adopted for update registrations. We will call *delta solutions* those where update registrations conform to the flow model, *consolidated solutions* those where update registrations conform to the stock model. In particular, in delta solutions:

1. An update measurement m_{ij} for event e_i is represented by a flow registration that records value $m_{ij} - m_{ij-1}$ (i.e., a "delta" for the fact with respect to the previous registration for e_i);
2. Transaction time is modeled by adding to the schema a new temporal dimension, typically with the same grain of the temporal dimension that models the valid time, to represent when each registration was made;
3. Up-to-date queries are answered by aggregating, for each event, all registrations;
4. Rollback queries at time t are answered by aggregating, for each event, all registrations whose transaction time is before t;
5. Historical queries on time interval T are answered by selectively aggregating, for each event, the registrations whose transaction time is included in T.

In consolidated solutions:

1. An update measurement m_{ij} for event ei is represented by a stock registration that records the consolidated value for the first registration m_{i1} of event ei, under a form depending on the model (flow or stock) adopted for m_{i1};
2. Transaction time is modeled by adding to the fact two new temporal dimensions, used as timestamps to mark the time interval during which each registration is current (*currency interval*);
3. Up-to-date queries are answered by selecting, for each event, the registration that

Table 4. Stock solution for the share fact

Date	Share	Price
Jan. 7, 2006	BigTel	9
Jan. 8, 2006	BigTel	12
Jan. 9, 2006	BigTel	10

Table 5. Flow solution for the share fact

Date	Share	Price
Jan. 7, 2006	BigTel	9
Jan. 8, 2006	BigTel	3
Jan. 9, 2006	BigTel	–2

is current now (the one whose currency interval is still open);
4. Rollback queries at time *t* are answered by selecting, for each event, the registration that was current at *t* (the one whose currency interval includes *t*);
5. Historical queries on time interval *T* are answered by selecting, for each event, the registrations that were current for at least one $t \in T$ (those whose currency interval overlaps with *T*).

The reason for using two timestamps in a consolidated solution is that each registration records a *state* of the event, which is valid during a time interval, rather than an instant measurement like in a flow solution.

Since in principle these two types of solutions can be combined with either the flow or the stock model for first registrations, four different specific solutions can be distinguished, which we will call *delta-flow*, *delta-stock*, *consolidated-flow,* and *consolidated-stock*, respectively. In the following subsections, we will discuss how these solutions are implemented for flow and stock facts subject to late measurements with updates, and for facts subject to late measurements without updates.

Flow Facts with Updates

As seen in the sixth section, a flow fact can be represented within the data warehouse either by flow or stock registrations.

In case of a flow fact represented by flow (first) registrations, the delta solution leads to events, first registrations and update registrations that share the same flow semantics, which means that additivity is preserved for all registrations. Consider for instance the enrollment schema; if a delta-flow solution is adopted, the schema is enriched as follows:

FT_ENROLL(EnrollDate, RegistrDate, Degree, AcademicYear, Number)

where RegistrDate is the dimension added to model transaction time. Table 6 shows a possible set of registrations for a given degree and year, including some positive and negative updates. While each first registration records the exact value of its measurement, each update registration records the difference between its measurement and the previous one.

With reference to these sample data, in the following we report some simple examples of queries of the three types together with their results, and show how they can be computed by aggregating registrations.

1. **q1:** *Daily number of enrollments to electric engineering for academic year 05/06.* This up-to-date query is answered by summing up Number for all registration dates related to the same enrollment dates, and returns the following result:

EnrollDate	Number
Oct. 21, 2005	11
Oct. 22, 2005	6
Oct. 23, 2005	3

Table 6. Delta-flow solution for the enrollment fact (update registrations in italics)

EnrollDate	RegistrDate	Degree	AcademicYear	Number
Oct. 21, 2005	Oct. 27, 2005	Elec. Eng.	05/06	5
Oct. 21, 2005	*Nov. 1, 2005*	*Elec. Eng.*	*05/06*	*8*
Oct. 21, 2005	*Nov. 5, 2005*	*Elec. Eng.*	*05/06*	*–2*
Oct. 22, 2005	Oct. 27, 2005	Elec. Eng.	05/06	2
Oct. 22, 2005	*Nov. 5, 2005*	*Elec. Eng.*	*05/06*	*4*
Oct. 23, 2005	Oct. 23, 2005	Elec. Eng.	05/06	3

2. **q2:** *Daily number of enrollments to electric engineering for academic year 05/06 as known on Nov. 2.* This rollback query is answered by summing up Number for all registration dates before Nov. 2:

EnrollDate	Number
Oct. 21, 2005	13
Oct. 22, 2005	2
Oct. 23, 2005	3

3. **q3:** *Number of registrations of enrollments to electric engineering received daily for academic year 05/06.* This historical query is answered by summing up Number, for each registration date, along all enrollment dates:

RegistrDate	Number
Oct. 23, 2005	3
Oct. 27, 2005	7
Nov. 1, 2005	8
Nov. 5, 2005	2

For a flow fact represented by flow registrations, also the consolidated solution is possible. In a consolidated-flow solution, the enrollment schema is enriched as follows:

FT_ENROLL(EnrollDate, CurrencyStart, CurrencyEnd, Degree, AcademicYear, Number)

where CurrencyStart and CurrencyEnd delimit the currency interval. Table 7 shows the set of registrations corresponding to those in Table 6: while the first registrations still report the same value for the fact, update registrations now report the exact value of measurements rather than a delta.

Adopting one or the other solution (delta-flow or consolidated-flow) for a flow fact has a deep impact on the response to the workload. For instance it is easy to see that, while in the delta-flow solution queries q_1 and q_2 are answered by accessing several registrations for each event involved, in the consolidated-flow solution they are answered by reading exactly one registration (respectively, the one that is current now and the one that was current on November 2) for each event involved.

In case of a flow fact represented by stock registrations, as seen in the sixth section, a monotemporal solution leads to an update propagation problem. This problem also occurs with a bitemporal solution: in fact, since stock registrations are computed by accumulating past flow measurements, each update measurement received for a past event e_i would lead to recording a whole set of update registrations, one for each event with valid time after e_i. Consequently, for a flow fact subject to late registrations with updates, we will not consider stock solutions recommendable.

Stock Facts with Updates

As seen in the sixth section, using flow registrations for a stock fact is not recommendable; thus, we will assume that a stock solution is adopted.

Table 7. Consolidated-flow solution for the enrollment fact

EnrollDate	CurrencyStart	CurrencyEnd	Degree	AcademicYear	Number
Oct. 21, 2005	Oct. 27, 2005	Oct. 31, 2005	Elec. Eng.	05/06	5
Oct. 21, 2005	*Nov. 1, 2005*	*Nov. 4, 2005*	*Elec. Eng.*	*05/06*	*13*
Oct. 21, 2005	*Nov. 5, 2005*	—	*Elec. Eng.*	*05/06*	*11*
Oct. 22, 2005	Oct. 27, 2005	Nov. 4, 2005	Elec. Eng.	05/06	2
Oct. 22, 2005	*Nov. 5, 2005*	—	*Elec. Eng.*	*05/06*	*6*
Oct. 23, 2005	Oct. 23, 2005	—	Elec. Eng.	05/06	3

Since both measurements and first registrations have stock semantics, the most immediate choice is the consolidated-stock solution, that gives stock semantics also to update registrations. In the share example, the schema is then enriched as follows:

FT_SHARE(Date, CurrencyStart, CurrencyEnd, Share, Price)

and may be populated for instance by the sample set of registrations reported in Table 8.

An example of up-to-date query on these data is "*find the minimum price of BigTel from Jan. 7 to 9,"* which returns 8.5. A rollback query is "*find the minimum price of BigTel from Jan. 7 to 9, as known on Jan. 10,"* which returns 9. Finally, a historical query is *"find the fluctuation on the price of Jan. 7 for BigTel,"* which returns –0.5 and requires to progressively compute the differences between subsequent registrations. Thus, while up-to-date and rollback queries are very simply answered, historical queries may ask for some computation.

In case of a stock fact, also the delta-stock solution can be applied. See for instance Table 9 that shows the delta solution for the same set of measurements reported in Table 8. In this case, up-to-date and rollback queries that aggregate the fact along valid time would have to be formulated as nested queries relying on different aggregation operators. For instance, the average monthly price for a share is computed by first summing Price along RegistrDate for each Date, then averaging the partial results. On the other hand, a historical query like the one above is very simply answered.

Facts without Updates

In the case of facts where measurements may be delayed but done exactly once for each event, accountability can be achieved, for both flow and stock solutions, by adding a single temporal dimension RegistrDate that models the transaction time. Up-to-date queries are solved without considering transaction times, while rollback queries require to select only the registrations made before a given transaction time. Historical queries make no sense in this context, since only one measurement is made for each event. As a matter of fact, the solution adopted can be considered as a special case of

Table 8. Consolidated-stock solution for the share fact

Date	CurrencyStart	CurrencyEnd	Share	Price
Jan. 7, 2006	Jan. 7, 2006	Jan. 11, 2006	BigTel	9
Jan. 7, 2006	*Jan. 12, 2006*	—	*BigTel*	*8.5*
Jan. 8, 2006	Jan. 10, 2006	—	BigTel	12
Jan. 9, 2006	Jan. 10, 2006	Jan. 12, 2006	BigTel	10
Jan. 9, 2006	*Jan. 13, 2006*	—	*BigTel*	*10.5*

Table 9. Delta-stock solution for the share fact

Date	RegistrDate	Share	Price
Jan. 7, 2006	Jan. 7, 2006	BigTel	9
Jan. 7, 2006	*Jan. 12, 2006*	*BigTel*	*–0.5*
Jan. 8, 2006	Jan. 10, 2006	BigTel	12
Jan. 9, 2006	Jan. 10, 2006	BigTel	10
Jan. 9, 2006	*Jan. 13, 2006*	*BigTel*	*0.5*

delta solution where no update registrations are recorded.

COMPARISON AND DISCUSSION

This section aims at providing a simple quantitative comparison of the different solutions, in terms of storage space and query performance. Let E, μ, and $|E_q| \leq E$ denote, respectively, the total number of events recorded, the average number of measurements per event, and the number of events involved in query q.

The results are collected in Table 10. The first column of data reports the total number of registrations stored in the fact table (we neglect that, in consolidated solutions, each registration is longer due to the additional timestamp that represents the end of the currency interval). The other three columns report the execution cost for different types of queries, estimated as the number of registrations to be accessed (independently of the execution plans adopted, and assuming that each registration is read only once):

- For up-to-date queries we assume that, at query formulation time, all measurements for the involved events are already available.
- For rollback queries, we consider a border effect related to the distribution along time of the measurements for each event, which reduces by a factor ρ ($0 \leq \rho \leq 1$) the number of registrations to be read (see Figure 2). Such factor heavily depends on the relationship between the width of the time interval defined by the valid times of the involved events, T_q, the relative positioning of the reference time for the query, t, and the average delay of measurements, δ. Figure 3 shows how ρ varies, assuming that measurements delays are normally distributed in time, in function of δ (expressed in numbers of refreshes), when T_q spans 12 refreshes and t falls exactly at the end of T_q.
- For historical queries, there still is a reduction factor ρ' that additionally depends on

Figure 2. Distribution of measurements for events; in gray the measurements that are not read by a rollback query with reference time t

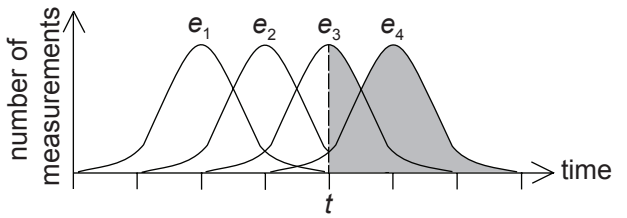
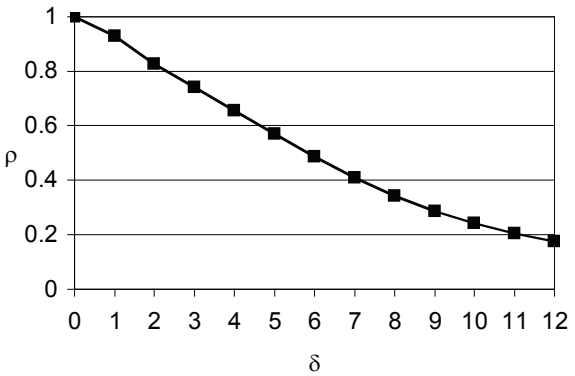


Table 10. Quantitative comparison of the design solutions (UQ, RQ, and HQ stand for up-to-date, rollback, and historical queries, respectively)

		number of tuples	*UQ*	*RQ*	*HQ*
monotemp.	*flow*	E	$\|E_q\|$	—	—
	stock	E	$\|E_q\|$	—	—
bitemp.	*delta-flow*	μE	$\mu \|E_q\|$	$\rho \mu \|E_q\|$	$\rho' \mu \|E_q\|$
	delta-stock	μE	$\mu \|E_q\|$	$\rho \mu \|E_q\|$	$\rho' \mu \|E_q\|$
	cons.-flow	μE	$\|E_q\|$	$\rho \|E_q\|$	$\rho' \mu \|E_q\|$
	cons.-stock	μE	$\|E_q\|$	$\rho \|E_q\|$	$\rho' \mu \|E_q\|$

Figure 3. Reduction factor in function of the average measurement delay



the width of the query reference interval *T*.

Overall, the overhead induced by the design solutions proposed on the query performance and on the storage space heavily depends on the characteristics of the application domain and on the actual workload. In a bitemporal solution, frequent updates determine a significant increase in the fact table size, but this may be due to a wrong choice of the designer, who promoted early registrations of events that are not stable enough to be significant for decision support. The increase in the query response time may be reduced by a proper use of materialized views and indexes: a materialized view aggregating registrations on all transaction times cuts down the time for answering up-to-date queries in delta solutions, while an index on transaction time enables efficient slicing of registrations in consolidated solutions.

CONCLUSION

In this article we have raised the problem of late measurements, meant as retrospective updates to events registered in a data warehouse, and we have shown how conventional design solutions, that only take valid time into account, may fail to provide query accountability and consistency. Then, we have introduced some alternative design solutions that overcome this problem by modeling transaction time as an additional dimension of the fact, and we have discussed their applicability. Table 11 summarizes the results obtained. Most noticeably, the most recommended solutions for a flow and a stock fact with updates are, respectively, consolidated-flow and consolidated-stock. Delta-flow and delta-stock, in fact, create some overhead with up-to-date and rollback queries.

In commercial platforms, late registrations are only partially supported. To the best of our knowledge, the most sophisticated solution is the one adopted by SAP-BW, that supports bitemporal solutions. In particular, BW distinguishes between *cumulative* and *non-cumulative key figures* (corresponding to flow and stock facts). The former are directly modeled in the fact table through a delta-flow solution. The latter can be handled by adopting two different time granularities: at the coarsest one, consolidated values for events are historicized within a support table; at the finest one, delta values are stored within the fact table limitedly to a user-defined time interval. This solution, while guaranteeing good querying performances, limits the expressivity achievable with rollback and historical queries.

We close this section by observing that, in real applications, multiple related facts are normally stored in the same fact table (e.g., in the order example, quantity and unit price for each order line). How do they coexist in presence of late measurements? For simplicity we

Table 11. Applicability of the design solutions

		without updates		*with updates*	
		flow fact	*stock fact*	*flow fact*	*stock fact*
monotemp.	*flow*	fair, only UQ supported	not recomm.	fair, only UQ supported	not recomm.
	stock	fair, only UQ supported	fair, only UQ supported	not recomm.	fair, only UQ supported
bitemp.	*delta-flow*	good	not recomm.	fair, overhead on UQ and RQ	not recomm.
	delta-stock	good	good	not recomm.	fair, overhead on UQ and RQ
	cons.-flow	—	—	good	not recomm.
	cons.-stock	—	—	not recomm.	good

will reason on the order example, focusing on the event reported below and assuming that (i) on March 17 a correction is received stating that the quantity is not 100 but 110, and (ii) on March 19 another correction is received taking to 1.1 the unit price.

OrderNumber	OrderDate	Quantity	UnitPrice
11001	Mar. 15, 2007	100	1

We consider three sample cases:

1. Accountability is required for no one of the facts. A monotemporal solution can be adopted; every time a new measurement is made for one of the facts, the related registration is overwritten leaving the other fact unchanged. Thus, on March 20 only one registration is present:

OrderNumber	OrderDate	Quantity	UnitPrice
11001	Mar. 15, 2007	110	1.1

2. Accountability is required for Quantity only. A consolidated-flow solution is adopted; when a new measurement is made for Quantity, an update registration is added that reports the new value for Quantity and the previous value for UnitPrice. If a new measurement is made for UnitPrice, the value of UnitPrice is updated within all the related registrations. Thus, on March 20 we have two registrations present:

Order-Number	Order-Date	Curren-cyStart	Curren-cyEnd	Quan-tity	Unit-Price
11001	Mar. 15, 2007	Mar. 15, 2007	Mar. 16, 2007	100	1.1
11001	Mar. 15, 2007	Mar. 17, 2007	—	110	1.1

3. Accountability is required for both facts. A consolidated-flow/stock solution is adopted; any new measurement for each fact creates a new registration:

Order-Number	Order-Date	Curren-cyStart	Curren-cyEnd	Quan-tity	Unit-Price
11001	Mar. 15, 2007	Mar. 15, 2007	Mar. 16, 2007	100	1
11001	Mar. 15, 2007	Mar. 17, 2007	Mar. 18, 2007	110	1
11001	Mar. 15, 2007	Mar. 19, 2007	—	110	1.1

Remarkably, having two different solutions coexist (like in cases 2 and 3) leads to

no additional overhead in query performance, except for some specific historical queries which require to distinguish the updates made to one fact from those made to the other.

REFERENCES

Abelló, A., & Martín, C. (2003a). The data warehouse: An object-oriented temporal database. *Proceedings Jornadas de Ingeniería del Software y Bases de Datos* (pp. 675-684), Alicante, Spain.

Abelló, A., & Martín, C. (2003b). A bitemporal storage structure for a corporate data warehouse. *Proceedings International Conference on Enterprise Information Systems* (pp. 177-183), Angers, France.

Bebel, B., Eder, J., Koncilia, C., Morzy, T., & Wrembel, R. (2004). Creation and management of versions in multiversion data warehouse. *Proceedings ACM Symposium on Applied Computing* (pp. 717-723), Nicosia, Cyprus.

Blaschka, M., Sapia, C., & Höfling, G. (1999). On schema evolution in multidimensional databases. *Proceedings International Conference on Data Warehousing and Knowledge Discovery* (pp. 153-164), Florence, Italy.

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* (pp. 27-41), Riga, Latvia.

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.

Devlin, B. (1997). Managing time in the data warehouse. *InfoDB*, *11*(1), 7-12.

Eder, J., Koncilia, C., & Morzy, T. (2002). The COMET metamodel for temporal data warehouses. *Proceedings International Conference on Advanced Information Systems Engineering* (pp. 83-99), Toronto, Canada.

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.

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.

Kim, J. S., & Kim, M. H. (1997). On effective data clustering in bitemporal databases. *Proceedings International Symposium on Temporal Representation and Reasoning* (pp. 54-61), Daytona Beach, US.

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

Kimball, R. (1996). The data warehouse toolkit. *Wiley Computer Publishing*.

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

Letz, C., Henn, E., & Vossen, G. (2002). Consistency in data warehouse dimensions. *Proceedings International Database Engineering and Application Symposium* (pp. 224-232), Edmonton, Canada.

Pedersen, T. B., & Jensen, C. (1998). Research issues in clinical data warehousing. *Proceedings Statistical and Scientific Database Management Conference* (pp. 43-52), Capri, Italy.

Quix, C. (1999). Repository support for data warehouse evolution. *Proceedings International Workshop on Design and Management of Data Warehouses*, Heidelberg, Germany.

Yang, J. (2001). Temporal data warehousing. PhD thesis, Stanford University, UK.

ENDNOTE

[1] Importantly, as made clear in the sixth and seventh sections, the value actually stored within a registration is not necessarily the value of m_{ij}, depending on the specific design solution adopted.

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 50 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 a full professor at the University of Bologna, where he is the head of the Data Warehousing Laboratory. He has published about 70 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, and what-if analysis.