What Time is it in the Data Warehouse?

Stefano Rizzi and Matteo Golfarelli

DEIS, University of Bologna, Viale Risorgimento 2, 40136 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 registrations 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 paper we discuss the alternative design solutions that can be adopted, in presence of late registrations, 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 model that represents the fact of interest. In particular, we show how late registrations can be differently supported depending on the flow or stock semantics given to events.

1 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 typically a dimension of analysis: thus, the representation of the history of measure values across a given lapse of time, at a given granularity, is directly supported. On the other hand, though the multidimensional model does not inherently represent the history of attribute values within hierarchies, some ad hoc techniques are widely used to support the so-called *slowly-changing dimensions* [1]. In both cases, time is commonly meant as *valid time* in the terminology of temporal databases [2], i.e., it is meant as the time when the event or the change within a hierarchy *occurred* in the business domain [3]. *Transaction time*, meant as the time when the event or change was *registered* in the database, is typically given little or no importance in data warehouses, 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 stored, 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 behaviour. In particular, the values of one or more measures 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 warehouse. In this context, if the current situation is to

be made timely visible to the decision maker, past events must be updated to reflect the incoming data.[1]

The need for updates typically arises when the early measurements made for events may be subject to errors (e.g., the amount of an invoice may be corrected after the invoice has been registered) or when events inherently evolve over time (e.g., notifications of university enrollments may be received and stored several days after they were issued). Unfortunately, if updates are carried out by physically overwriting past events, some problems may arise:

- Accountability and traceability require the capability of preserving the exact information the analyst based his/her decision upon. If old events are replaced by their "new" versions, 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.

Note that 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 fact, though no update is necessary, still valid time is not sufficient to guarantee accountability. Thus, in more general terms, we will call *late registration* any registration of events that is delayed with respect to the time when the event occurs in the application domain, with the tolerance of the natural delay related to the refresh interval of the data warehouse. A late registration may either imply an update or not.

In this paper we discuss the design solutions that can be adopted, in the presence of late registrations, 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 paper contributions can be summarized as follows:

- Two possible semantics for events are distinguished, namely flow and stock, and it is shown how they can be applied to the events occurring in the application domain, to the events registered in the data warehouse for the first time, and to the events registered later to represent updates (Section 4).
- Three basic categories of queries are distinguished, from the point of view of their different temporal requirements in presence of late registrations (Section 5).
- A set of design solutions to support late registrations is introduced, and their relationship with the three categories of queries and with the two different semantics of events is discussed (Section 6).

[1] In the following, when using the term *update*, we will mean a *logical* update, which does not necessarily imply a *physical* update (i.e., an overwrite).

2 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., [4, 5]). Some other works tackle the problem of temporal evolution and versioning of the data warehouse schema [6–10]. All these works are not related to ours, since there is no mention to the opportunity of representing transaction time in data warehouses in order to allow accountability and traceability in case of late registrations.

In [3] it is distinguished between *transient data*, that do not survive updates and deletions, and *periodic data*, that are never physically deleted from the data warehouse. In [1] two basic paradigms for representing inventory-like information in a data warehouse are introduced: 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 distinction is relevant to our approach, and is recalled in Section 4.

In [11], the importance of advanced temporal support in data warehouses, with particular reference to medical applications, is recognized. In [12] the authors 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, in [13] a storage structure for a bitemporal data warehouse (i.e., one supporting both valid and transaction time) is proposed. All these approaches suggest that transaction time should be modeled, but not with explicit reference to the problem of late registrations.

The approach that is most related to ours is the one presented in [14], where the authors discuss the problem of DW temporal consistency in consequence of delayed discovery of real-world changes and propose a solution based on transaction time and overlapped valid time. Although the paper discusses some issues related to late registrations, 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.

3 Motivating Examples

In the first example we provide, late registrations are motivated by the fact that the represented events inherently evolve over time. Consider a single fact modeling the student enrollments to university degrees; in a relational implementation, a simplified fact table for enrollments could have the following schema[2]:

FT_ENROLL(EnrollDate, Degree, AYear, City, Number)

where EnrollDate is the formal enrollment date (the one reported on the enrollment form). An enrollment is acknowledged by the University secretariat only

[2] For simplicity, we will assume that surrogate keys are not used.

when the entrance fee is paid; considering the variable delays due to the bank processing and transmitting the payment, the enrollment may be registered in the data warehouse even one month after the enrollment has been formally done. This is a case of late registrations. 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 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, motivated by the delay in registering information and by wrong measurements, is that of a large shipping company with several warehouses spread around the country, that maintains a centralized inventory of its products:

FT_INVENTORY(InvDate, Product, Warehouse, Level)

We assume that the inventory fact is fed by weekly snapshots, coming from the different warehouses, of the inventory level for each product. In this scenario, delays in communicating the weekly levels and late corrections sent by the warehouses will produce late registrations, which in turn will raise problems with justifying the decisions made on previous reports.

4 The Semantics of Events

The aim of this section is to introduce the classification of events on which we will rely in Section 6 to discuss the applicability of the design solutions proposed.

As recognized in [1], from the point of view of the conceptual role given to events, facts basically conform to one of two possible models:

- *Transactional fact.* 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* [15]: they refer to a time interval and are cumulatively evaluated at the end of that period; thus, they are additive along all dimensions (i.e., their values can always be summed when aggregating).
- *Snapshot fact.* In this case, events correspond to periodical snapshots of the fact. Measures are mostly *stock measures* [15]: they refer to an instant in time and are evaluated at that instant; thus, they are non-additive along temporal dimensions (i.e., they cannot be summed when aggregating along time, while for instance they can be averaged).

This distinction is based on the semantics of the *stored events*, i.e., the events logically recorded in the data warehouse: in a transactional fact they are meant

as *flow events*, while in a snapshot fact they are meant as *stock events*. Intuitively, while flow events model a "delta" for the fact, stock events measure its "level".

The choice of one model or another is influenced by the core workload the fact is subject to, but mainly depends on the semantics of the *domain events*, i.e., on how the events occurring in the application domain are measured: in the form of flows or in the form of stocks. In the first case, a transactional fact is the more proper choice, though also a snapshot fact can be used provided that (i) an aggregation function for composing the flow domain events into stock stored events is known, and (ii) events are not subject to updates — otherwise, after each update, all the related (stock) events would have to be updated accordingly, which may become quite costly. Conversely, if events are measured as stocks, a snapshot fact is the only possible choice, since adopting a transactional fact would require disaggregating the stock domain events into inflows and outflows — which, in the general case, cannot be done univocally.

A large percentage of facts in the business domain naturally conform to the transactional model. For instance, in an invoice fact, each (domain and stored) event typically represents a single line of an invoice, and its measures quantify some numerical aspects of that line — such as its quantity or amount. In theory, one could as well build an equivalent snapshot fact where each stored event models the cumulated sales made so far, computed by summing up the invoice lines: of course this would be quite impractical, since most query will focus on partial aggregations of invoice lines, that would have to be computed by subtraction of consecutive stored events.

Other facts naturally conform to the snapshot model: for instance a fact measuring, on each hour, the level of a river in different places along its course. Also the centralized inventory fact mentioned in Section 3 conforms to the snapshot model, since both its stored and domain events have stock semantics.

Finally, for some facts both models may reasonably fit: an example is the enrollment fact seen in Section 3, where two different interpretations can be given to events (and to measure Number accordingly) for the same schema. In the first (transactional) interpretation each (flow) event records the number of students from a given city who enrolled, on a given date, to a given degree course for a given academic year. In the second (snapshot) interpretation each (stock) event records, at a given date, the total number of students from a given city who enrolled to a given degree course for a given academic year so far. Two sample sets of events for enrollments according to the two interpretations are shown in Table 1; the designer will choose one or the other interpretation mainly according to the expected workload.

5 Temporal Dimensions and Querying Scenarios

From a conceptual point of view, for every fact subject to late registrations, at least 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 they are *perceived and recorded* in the

Table 1. Enrollment events for the transactional (left) and the snapshot (right) facts

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

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

data warehouse. In the literature on temporal databases, these two dimensions correspond, respectively, to valid time and transaction time [2]. Note that, for a fact that is not subject to late registrations, transaction time is implicitly considered to coincide with valid time (the natural delay due to the refresh interval in neglected).

While we take for granted that valid time must always be represented, since it is a mandatory coordinate for characterizing the event, 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 [16]):

- *Up-to-date queries*, i.e., queries requiring the most recent value estimate for each measure. 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. Representing transaction time is not necessary to solve this kind of queries, since they rely on valid time only.
- *Rollback queries*, i.e. queries requiring a past value estimate for each measure, as for instance the one asking for the current trend of the total number of enrollments for each faculty 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 represented explicitly.
- *Historical queries*, i.e. queries requiring multiple value estimates for each measure. An example of historical query is the one asking for the day-by-day distribution of the enrollments registered overall for a given enrollment date. Also these queries require transaction time to be represented explicitly.

6 Design solutions

In presence of late registrations, two types of design solution can be envisaged depending on the expected workload:

- *Monotemporal schema*, where only valid time is modeled as a dimension. This is the simplest solution: during each refresh cycle, as up-to-date values become available, a new set of events are recorded, which may imply updating events recorded at previous times. The time when the events are recorded

is not represented, and no trace is left of past values in case of updates, so only up-to-date queries are supported.
– *Bitemporal schema*, where both valid and transaction time are modeled as dimensions. This is the most general solution, allowing for all three types of queries to be correctly answered. On each refresh cycle, new events for previous valid times may be added, and their registration time is traced; no overwriting of existing events is carried out, thus no data is lost.

The monotemporal schema for the enrollment example is exactly the one already shown in Section 3, where the only temporal dimension is EnrollDate. When further enrollments for a past enrollment date are to be registered, the events corresponding to that date are overwritten and the new values for measures are reported. Note that this solution can be equivalently adopted for both a transactional and a snapshot fact, and in neither case it supports accountability.

While the monotemporal schema deserves no additional comments, since it is the one commonly implemented for facts that either are not subject to late registrations or only require to support up-to-date queries, the bitemporal schema requires some further clarification. In fact, two specific solutions can be devised for a bitemporal schema, namely *delta solution* and *consolidated solution*, where the events used to represent updates have flow and stock semantics, respectively. These solutions are described in the following subsections.

6.1 The Delta Solution

In the delta solution:

1. each update is represented by a flow event that records a "delta" for the fact;
2. transaction time is modeled by adding to the fact a new temporal dimension, typically with the same grain of the temporal dimension that models the valid time, to represent when each event was recorded;
3. up-to-date queries are answered by aggregating events on all transaction times;
4. rollback queries at a given time t are answered by aggregating events on the transaction times before t;
5. historical queries are answered by slicing the events based on their transaction times.

This solution can easily be applied to a transactional fact: in this case, all stored events (those initially recorded and those representing further updates) have flow semantics. In particular, flow measures uniformly preserve their additive nature for all the events. Consider for instance the enrollment schema. If a delta solution is adopted, the schema is enriched as follows:

FT_ENROLL(EnrollDate, RegistrDate, Degree, AYear, City, Number)

where RegistrDate is the dimension added to model transaction time. Table 2 shows a possible set of events for a given city, degree, and year, including some

Table 2. Enrollment events in the delta solution applied to a transactional fact (events representing updates in italics)

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

positive and negative updates. 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 events.

1. q_1: *daily number of enrollments to Electric Engineering for academic year 05/06*. This up-to-date query is answered by summing up measure Number for all registration dates related to the same enrollment dates, and returns the following result:

EnrollDate	Degree	AYear	City	Number
Oct. 21, 2005	Elec. Eng.	05/06	Rome	11
Oct. 22, 2005	Elec. Eng.	05/06	Rome	6
Oct. 23, 2005	Elec. Eng.	05/06	Rome	3

2. q_2: *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	Degree	AYear	City	Number
Oct. 21, 2005	Elec. Eng.	05/06	Rome	13
Oct. 22, 2005	Elec. Eng.	05/06	Rome	2
Oct. 23, 2005	Elec. Eng.	05/06	Rome	3

3. q_3: *daily net number of registrations of enrollments to Electric Engineering for academic year 05/06*. This historical query is answered by summing up Number for all enrollment dates:

RegistrDate	Degree	AYear	City	Number
Oct. 23, 2005	Elec. Eng.	05/06	Rome	3
Oct. 27, 2005	Elec. Eng.	05/06	Rome	7
Nov. 1, 2005	Elec. Eng.	05/06	Rome	8
Nov. 5, 2005	Elec. Eng.	05/06	Rome	2

In case of a snapshot fact where the domain events have stock semantics, the delta solution is not necessarily the best one. See for instance Table 3, that with reference to the inventory example seen in Section 3 shows a possible set of events for a given week and product assuming that some data sent by local warehouses

Table 3. Inventory events in the delta solution applied to a snapshot fact

InvDate	RegistrDate	Product	Warehouse	Level
Jan. 7, 2006	Jan. 8, 2006	LCD TV	Milan	10
Jan. 7, 2006	*Jan. 12, 2006*	*LCD TV*	*Milan*	*−1*
Jan. 7, 2006	Jan. 12, 2006	LCD TV	Rome	5
Jan. 7, 2006	Jan. 10, 2006	LCD TV	Venice	15
Jan. 7, 2006	*Jan. 14, 2006*	*LCD TV*	*Venice*	*2*

are subject to corrections. In this case, up-to-date and rollback queries that summarize the inventory level along valid time would have to be formulated as nested queries relying on different aggregation operators. For instance, the average monthly level for a warehouse is computed by first summing Level across RegistrDate for each InvDate, then averaging the partial results.

We close this section by considering the particular case of facts where registrations may be delayed but events, once registered, are *not* further updated. In this case accountability can be achieved, for both transactional and snapshot facts, 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 events recorded before a given transaction time. Historical queries make no sense in this context, since each event has only one logical "version". As a matter of fact, the solution adopted can be considered as a special case of delta solution where no update events are to be registered.

6.2 The Consolidated Solution

In the consolidated solution:

1. each update is represented by a stock event that records the consolidated version of the fact;
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 event was current within the data warehouse (*currency interval*);
3. up-to-date queries are answered by slicing the events that are current today (those whose currency interval is still open);
4. rollback queries at a given time t are answered by slicing the events that were current at t (those whose currency interval includes t);
5. historical queries are answered by slicing the events based on their transaction times.

In the inventory example, if a consolidated solution is adopted, the schema is enriched as follows:

FT_INVENTORY(InvDate, CurrencyStart, CurrencyEnd, Product, Warehouse, Level)

Table 4. Inventory events in the consolidated solution applied to a snapshot fact

InvDate	CurrencyStart	CurrencyEnd	Product	Warehouse	Level
Jan. 7, 2006	Jan. 8, 2006	Jan. 11, 2006	LCD TV	Milan	10
Jan. 7, 2006	*Jan. 12, 2006*	–	*LCD TV*	*Milan*	*9*
Jan. 7, 2006	Jan. 12, 2006	–	LCD TV	Rome	5
Jan. 7, 2006	Jan. 10, 2006	Jan. 13, 2006	LCD TV	Venice	15
Jan. 7, 2006	*Jan. 14, 2006*	–	*LCD TV*	*Venice*	*17*

Table 5. Summary of the possible solutions (UQ and HQ stand for up-to-date and historical queries, respectively)

	transactional fact	*snapshot fact*	
	flow domain events	*flow domain events*	*stock domain events*
monotemporal schema	good but only supports UQ	good if no updates, else not recomm.	good but only supports UQ
delta sol. – no upd.	good	good	good
delta sol. – with upd.	good	not recomm. due to update propagation	fair due to nesting
consolidated solution	good but overhead on HQ	not recomm. due to update propagation	good but overhead on HQ

Table 4 shows the consolidated solution for the same set of events reported in Table 3. An example of up-to-date query on these data is *"find the total number of LCDs available on Jan. 7"*, which returns 31. On the other hand, a rollback query is *"find the total number of LCDs available on Jan. 7, as known on Jan. 10"*, which returns 25. Finally, a historical query is *"find the fluctuation on the level of Jan. 7 for each warehouse"*, which requires to progressively compute the differences between subsequent events and returns -1, 0, and 2 for Milan, Rome, and Venice respectively. Thus, while up-to-date and rollback queries are very simply answered, historical queries may ask for some computation.

Similarly, for a transactional fact, applying the consolidated solution is possible though answering historical queries may be computationally more expensive than with a delta solution.

7 Conclusion

In this paper we have raised the problem of late registrations, meant as retrospective registrations of events 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 depending on the semantics of events. Table 5 summarizes the results obtained. Most noticeably, using a snapshot fact when domain events have flow semantics is not recommendable in case of updates, since they should then be

propagated. Besides, for a transactional fact all solutions are fine, though the delta one is preferable since it adds no overhead for historical queries. Conversely, for a snapshot fact the consolidated solution is preferable since aggregation nesting is not required.

The overhead induced by the proposed solutions on the query response time and on the storage space obviously depends on the characteristics of the application domain and on the actual workload. 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 recording of events that are not stable enough to be significant for decision support. The increase in the query response time may be contained by a proper use of materialized views and indexes: a materialized view aggregating events on all transaction times cuts down the time for answering up-to-date queries in the delta solution, while an index on transaction time enables efficient slicing of the events.

References

1. Kimball, R.: The data warehouse toolkit. Wiley Computer Publishing (1996)
2. Jensen, C., Clifford, J., Elmasri, R., Gadia, S.K., Hayes, P.J., Jajodia, S.: A consensus glossary of temporal database concepts. ACM SIGMOD Record **23**(1) (1994) 52–64
3. Devlin, B.: Managing time in the data warehouse. InfoDB **11**(1) (1997) 7–12
4. Letz, C., Henn, E., Vossen, G.: Consistency in data warehouse dimensions. In: Proc. IDEAS. (2002) 224–232
5. Yang, J.: Temporal data warehousing. PhD thesis, Stanford University (2001)
6. Bębel, B., Eder, J., Koncilia, C., Morzy, T., Wrembel, R.: Creation and management of versions in multiversion data warehouse. In: Proc. SAC, Nicosia, Cyprus (2004) 717–723
7. Blaschka, M., Sapia, C., Höfling, G.: On schema evolution in multidimensional databases. In: Proc. DaWaK. (1999) 153–164
8. Eder, J., Koncilia, C., Morzy, T.: The COMET metamodel for temporal data warehouses. In: Proc. CAiSE. (2002) 83–99
9. Golfarelli, M., Lechtenbörger, J., Rizzi, S., Vossen, G.: Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation. Data and Knowledge Engineering (2006, To appear)
10. Quix, C.: Repository support for data warehouse evolution. In: Proc. DMDW. (1999)
11. Pedersen, T.B., Jensen, C.: Research issues in clinical data warehousing. In: Proc. SSDBM, Capri, Italy (1998) 43–52
12. Abelló, A., Martín, C.: The data warehouse: an object-oriented temporal database. In: Proc. JISBD 2003, Alicante, Spain (2003) 675–684
13. Abelló, A., Martín, C.: A bitemporal storage structure for a corporate data warehouse. In: Proc. ICEIS. (2003) 177–183
14. Bruckner, R., Tjoa, A.: Capturing delays and valid times in data warehouses - towards timely consistent analyses. Journ. Intell. Inf. Syst. **19**(2) (2002) 169–190
15. Lenz, H.J., Shoshani, A.: Summarizability in OLAP and statistical databases. In: Proc. SSDBM. (1997) 132–143
16. Kim, J.S., Kim, M.H.: On effective data clustering in bitemporal databases. In: Proc. TIME. (1997) 54–61