Managing Time in the Data Warehouse

by Dr. Barry Devlin, IBM

Of the many design issues that arise when building a data warehouse, one of the least understood is the question of handling the time dimension. The historical nature of the warehouse is widely accepted; however, a common theoretically-based but still practical approach is lacking. This article explores some of the most common aspects of the problem and lays out a number of principles for treating time successfully in the warehouse.

Data Warehouse Structure — A Review

Before looking at time management in the warehouse, it is appropriate to first review the logical structure of the data warehouse. Throughout this article, I will use the *three-layer model* of the data warehouse. This structure is now widely accepted as the optimal approach to providing consistent and reliable management information throughout the organization.[2]

The lowest layer, used by the operational applications is *real-time data*. Real-time data is up-to-the-second data and used to run the business. The top layer, used by informational applications, is *derived data* and is used to manage the business. It may exist at a detailed or summary level, and it is used in a read-only mode.

The key to the three-layer architecture, as shown in Figure 1, is the recognition that there are two steps in the transformation from real-time data to derived data.[6]

The first step brings the real-time data to a common base of *reconciled data*. This is a special category of data, built by only removing inconsistencies and errors in the source data. It is stored at a detailed level, and is the definitive, complete and historical record of the real-time data across the whole enterprise. This store is known as *the business data warehouse (BDW)*.

The second step addresses the disparate needs of end users, and derives the subsets or aggregations of data they need. This derived data is sourced only from the BDW and is stored in *business information warehouses (BIWs)*.

The data warehouse, consisting of the BDW and the set of derived BIWs, can thus be seen as a logical set of data, used for managing the business, that has been aligned to the overall data needs of the business. It may consist of many physical copies of that data, optimized for particular uses, but its key feature is that it is structured in such a way that it can be controlled and managed. This structure and the rationale for it is described in more detail in References 3, 4 and 7.

Business or Pleasure?

The warehouse exists to help manage the business. Because a business changes over time, the business data must represent that change, and must therefore contain a time element. In fact, there are often many different time elements required. However, the warehouse itself, like any other application, must also be managed as a function of time. This leads to the immediate conclusion that time in the data warehouse is not just the fourth dimension, but also the fifth, sixth and so on.

First consider time from a business viewpoint. Any time of interest to the business is recorded in the business data itself. For example, a personnel planning application stores dates when promotions are planned. These dates may be in the past or in the future and are part of the business data of that application. The *valid business time* axis measures the times the planned dates are entered or changed and not the planned dates themselves.[1] Of course, when the promotions occur, these events also appear in the valid business time.

When analyzing and tracking business performance, valid business time should clearly form the basis for measuring time and assigning periods of validity to business events or statuses of interest. In some cases, however, the valid business time is not recorded in the operational system, and an alternative such as *business transaction time* or *operational database time* must be used. This is the time when a business event enters or is recorded in the operational system. These times for an event can be the same or later than the valid business time.

BDW database time is the time when the business event is written into the warehouse. With very rare exceptions, this time is later — often considerably later — than the business transaction time.

Of these four times, all are system generated with the notable exception of valid business time, which must be input by a user.

The decision about which time dimension to use and when to use it depends not only on business needs, but also on the way time is managed in the operational environment, and on the technical implementation of warehouse population. In particular, it is vital to distinguish between the use of time elements to a) track the business (valid business time is the real measure here, although business transaction time or operational

database time are often used as surrogates) and b) to manage the temporal integrity of the warehouse (BDW database time is the key in this case).

Tracking the Use of Time in the Business

Traditional data modeling and application design approaches focus almost exclusively on a static view of the world. As a result, the temporal aspects of relationships are effectively ignored in traditional data modeling. To include time in modeling, events (which change the relationships between entities over time) must be included in the model. A number of attempts have been made to extend the traditional data model to include events.[5] Today's modeling tools, however, incorporate little or none of this functionality, and database systems provide no explicit support for time dependency. The result is that designers usually add time dependency of data to application designs largely as an afterthought.

A data warehouse design, however, must explicitly consider the temporal aspects of the data it contains, because it must, by definition, provide a historical view of the business. The most widely used method is the application of *timestamps* to the data. Since data changes at the field level, it is possible to represent time at that level, or at any of the higher levels in the structure, such as records/rows or file/table, depending on the granularity (detail) required. Field-level timestamping generates huge volumes of timestamp data and is seldom needed. The usual approach is to timestamp data at the record/row level, where the record timestamp is updated whenever any field in that record changes. From a business viewpoint, this approach meets almost all needs for tracking time, and is the preferred approach in the BDW.

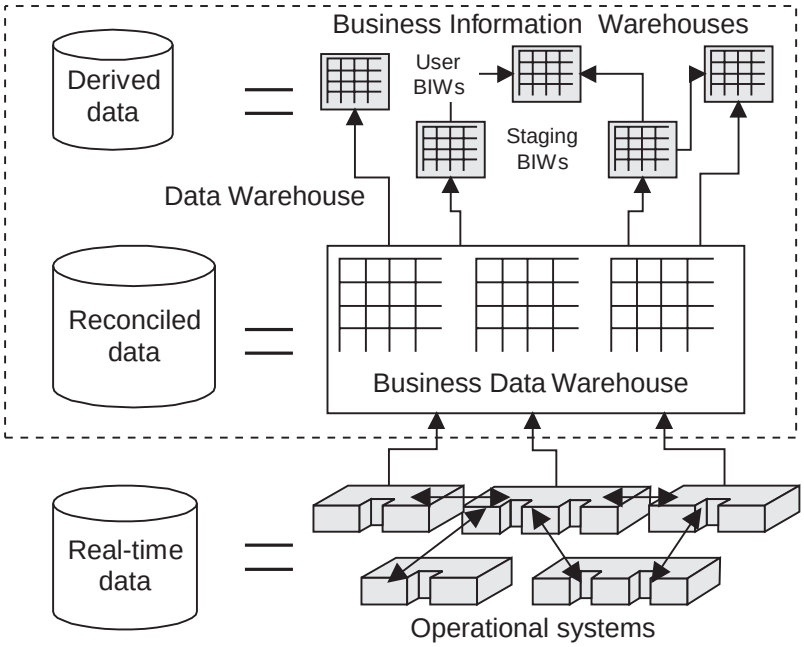

Figure 1. The logical architecture of the data warehouse

Timestamping at file or table level updates the timestamp whenever any record changes. This approach is not appropriate for controlling and auditing the data. It is, however, often the level at which end users wish to track the currency of their data. Snapshots, which represent a view of the business at some point in time, are implicitly or explicitly timestamped at a file/table level.

Temporal Data Structures

Timestamps allow the maintenance of temporal data. When considering temporal data we need to understand how time is reflected in a database, how this relates to the structure of the data, and how a new event affects existing data. There are two basic approaches: *transient* data and *periodic* data. A special case of periodic data, known as a *snapshot*, also needs to be considered. Figure 2 will be used to explain transient data and periodic data. In the figure, each record contains a key field *k* and a timestamp *t*.

Transient Data

The key characteristic of *transient data* is that alterations to and deletions of existing records physically destroy the previous data content. Records can be added (for example, record *k6* at time *t2* in Figure 2) and deleted (record *k3* at time *t3*). There is, however, no evidence in the data after time *t3* of the previous contents of record *k3*, or indeed that it ever existed. Records can also be changed — as in the case of deletion, the previous states of changed records are lost. This type of data is typical of that found in the operational environment.

Periodic data

With *periodic data*, once a record is added to the store, it is never physically deleted, nor is its business content ever physically modified. Rather, new records are always added, even for updates to, or deletions of, existing records. Thus, at time *t2* in Figure 2, record *k3* is updated, but rather than the previous value *C*

being overwritten with a new value *P*, a complete new record with the same "key" *k3*, but with a different timestamp *t2* is added. As a result, the timestamp is an integral part of the key to the record. Instead of deleting a record, this fact is handled by the addition of a specially marked record. In the figure, record *k4* is "deleted" at time *t3* by the addition of a record indicating this key value is no longer valid. Addition of new records is identical to that shown earlier.

Periodic data thus contains a complete record of the changes that have occurred in the data. Periodic data is persistent in nature because it provides a permanent record of the data and its changes.

Periodic data is found in the real-time data of operational systems where a record of the previous states of the data is important. Thus, bank account systems and insurance premiums systems often use this approach. Order-entry systems, on the other hand, are usually based on transient data. This choice reflects different business needs for tracking and auditability. However, in almost all operational systems, the duration for which this persistent data is held is relatively short, due to performance and/or storage volume constraints. This kind of data may be termed semi-periodic data.

Periodic data is also found in the derived data layer of a data warehouse where it is used to support trend analysis of historical data. The reconciled data layer consists entirely of periodic data because of its role as the historical record of the business.

Snapshot Data

Snapshot data is a stable view of the data as it exists at some point in time. It does not contain any record of changes in the data that resulted in its arrival at this state. If it is updated, it may be totally replaced, or at a detail level, records may be changed in a similar way to that in which transient data is changed.

This type of data occurs most often in the derived data layer of a data warehouse. Snapshots usually represent the business data at some time in the past, and a series of snapshots can provide a view of the history of the business. However, predictive or planned business states are also snapshots, in this case representing the future.

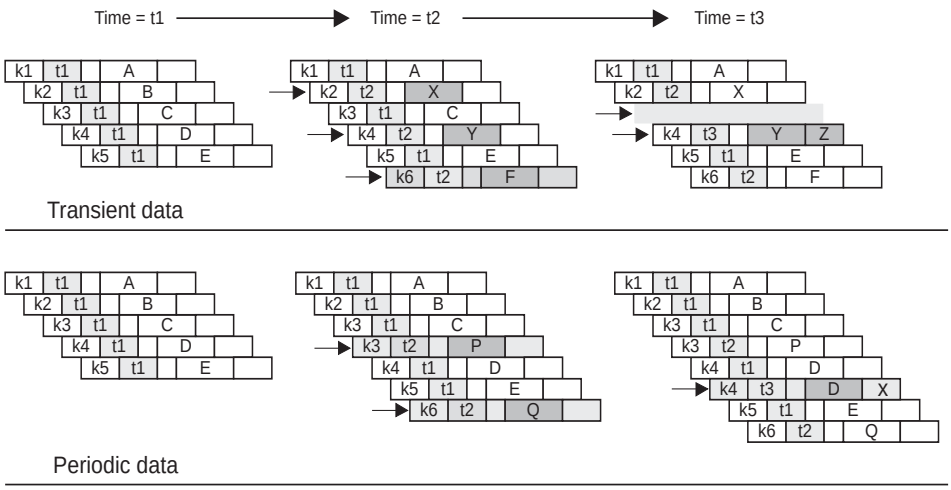

Figure 2. Transient and periodic data

The Structure of Periodic Data in the BDW

The data in the BDW is periodic in nature, representing the history of the changes in the business data over a prolonged period of time. The standard approach to storing this periodic data is to use timestamped status and event records. There are, however, a variety of schemes to maximize the efficiency of timestamps.

Each instance of a business entity, such as an order or a customer, is represented in a warehouse by a set of records, each of which has the same business key, but is differentiated by the timestamp(s) used as part of the key. These timestamps show the period of validity of a particular record. For the moment, we assume that the period of validity is based on event timings from a business point of view rather than from a technical or database view. We will return to this assumption later.

Using a Single Timestamp

The simplest approach, shown in Figure 3, is to implement a single timestamp field, *start time*, identifying the time at which a record became valid. A record is then deemed to be invalid when there is another record with the same business key and a later timestamp.

This structure is ideal for event data where the timestamp represents the time at which the event took place. Account transaction data in a bank is a good example of event data.

For status data, however, where the outcomes of events, rather than the events themselves, are stored, the single timestamp approach presents a problem for two types of relatively common query.

1. **A query that needs to access current data**. In a single timestamp scheme, the only way to identify current records is to find the

latest timestamp of the periodic set, which is an inefficient process.

2. **A query that builds a view of the data at a particular time in the past**. To support this query, the period of validity of each record must be known, in order to compare it with the required time. With the single timestamp approach, the end of the period of validity of one record can only be found from the next record in the periodic sequence, again an expensive process for relational databases.

Using Two Timestamps

To address the above problems, a second timestamp can be added to each record, as shown in Figure 4. This second timestamp, *end time*, identifies the end of the period of validity of the record. In this scheme, the current record is identified by some special value in the end time field.

Setting the end time for a record takes place when the record superseding it is written. The need to find and update the superseded record thus introduces an overhead into the update process. For example, in Figure 4, when record *k1,t2* is added to the BDW, the update process must locate record *k1* with a null value for end time, and replace the value of end time in that record with *t2*. The performance improvements in retrieving data, however, compensate for this added update overhead.

Using Additional Fields

In a perfect world, a scheme using two timestamps would be sufficient. Other problems may arise, however. Operational applications are often designed without consideration for the consequences of storing history information. One important effect of this omission can be seen in the use of business keys. In some businesses,

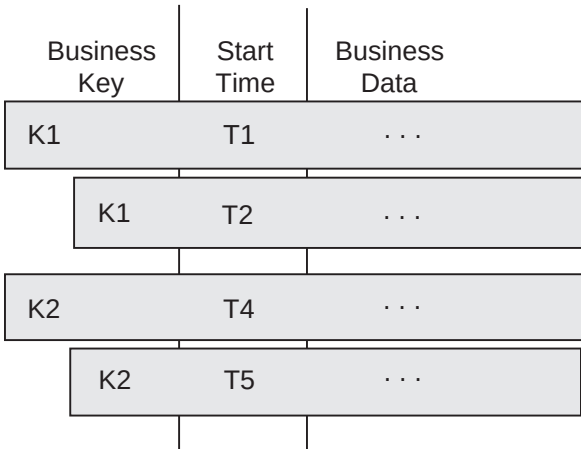

Figure 3. Periodic data using start times

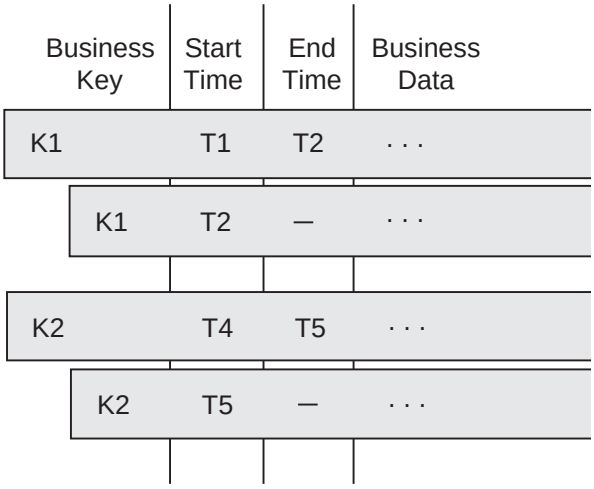

Figure 4. Periodic data using end times

key values are reused over time. An example of this is the use of order numbers in a business with a short selling cycle. These are generated within a fixed range, which is then recycled after some period of time. In this case, carrying these order numbers into a periodic warehouse does not allow a distinction between sets of related records for different orders.

This problem can be addressed without impact on the operational application by introducing another timestamp, *initial time*, on each record, as shown in Figure 5. Initial time identifies the time at which the sequence of related business events began, and is therefore equal to the start time of the first record of the sequence. Within the periodic data, the business key is a composite of the operational business key and the initial time.

At this stage, one might be tempted to ask if more than three timestamps could possibly be required. One final problem remains with these schemes — the difficulty in recognizing what action has caused a particular record to be created. This is particularly needed for records that represent the

completion of a sequence of records, in other words, the deletion of a particular instance of a data entity. In periodic data, no record is ever physically deleted, but is instead marked as no longer valid.

One approach is to designate a special value of end time to represent this state. A more general approach is to introduce one final field — not a timestamp, but a flag — the *action flag*, to indicate the action in the operational system that created each of the records. As Figure 5 shows, this flag can take one of three values:

- **A** — representing the *addition* of a new data instance (the first record in the sequence)
- **C** — a *change* or update of an existing data instance
- **D** — the *deletion* of a data instance (the last record in the sequence)

This approach represents probably the most functionally rich solution, catering to all possible data access and technical requirements.

Choosing the Appropriate Structure

The structures described above represent a variety of possible approaches to representing history in the BDW. Except for the start time timestamp, each is optional. Their use depends mainly on the abilities of the BDW and BIW population functions. End time and initial time can both present difficulties in populating the BDW. The use of just a start time can cause performance difficulties in selecting particular subsets of data in populating the BIW. A combination of start time and end time, supported by an action flag, generally provides an efficient structure for status data in the BDW. Event data can generally be managed using a start time and either an end time or an action flag.

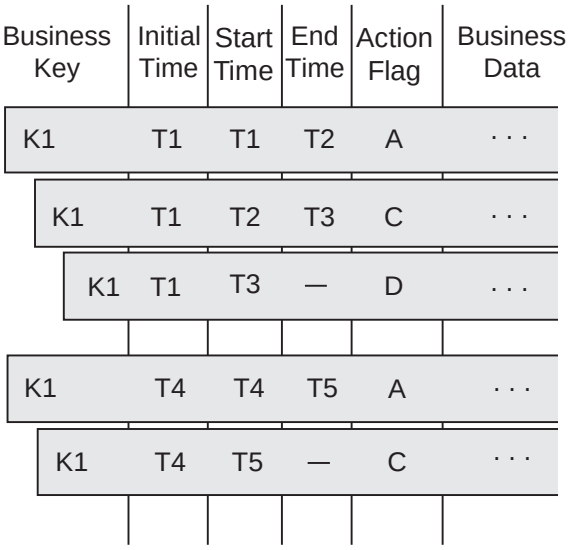

Figure 5. Periodic data using initial time and status flags

Conclusions

Our journey through time in the data warehouse has led to a number of conclusions.

1. The temporal aspects of business data must be distinguished carefully from those relating to the control and management of the data warehouse. Failure to do so will lead to significant difficulties for both end users and IS personnel. If data warehouse control timestamps are to be used as surrogates for business times, the implications for end users must be clearly understood.
2. In the design of the BDW, temporal aspects must be considered at an early stage. At a logical level, the business impact of (and interest in) events that alter data values must be analyzed closely. At a physical level, the timestamps required to control and manage the data warehouse need to be defined up front.
3. In order to map data from a particular operational system into the data warehouse, the designer must understand the concepts of transient and periodic data, and how the operational system handles time dependency.

Experience of a large number of data warehouse implementations has led to the approach outlined above. Applying these principles leads to a comprehensive and well-controlled implementation of historical data in the BDW. Such a well-structured BDW in turn is the ultimate source of information needed by any business analysis of a temporal nature.

About the Author

Dr. Barry Devlin is among the foremost authorities on data warehousing. He defined IBM Europe's own data warehouse architecture in the mid-1980s, and subsequently published the seminal article on the subject in 1988.[2] He continues to work at the leading edge of data warehouse architecture and implementation. Barry is a widely respected writer and lecturer on the subject throughout the world. Addison Wesley published his comprehensive book on the subject in October 1996.[4]