TEMPORAL DATA: THEORETICAL AND PRACTICAL CONSIDERATIONS

Stefan Berner, Diso Solution AG

The Problem

"How much would this article have cost next August, if I had asked last January?" If your database contains the answer to that question, it is bi-temporal. Even if your questions are not as complex as that, historical or temporal data is a problem which you face in almost any application nowadays. Very often it is a source of new problems for maintenance, usability and acceptance of an application.

Technically spoken there exist different solutions depending on requirements and available tools or libraries. From the point of view of modeling (that is the users view) there should be only one. This paper will focus on this solution to the users problem in the first part and then list some practical tips for the implementation towards the end.

You might know the scenario: during the modeling process somebody "would like to have a history of all data". So a `valid_from/valid_to` (or `from_date/to_date` or whatever the are called) pair of attributes (of type `date`) is added to every entity in your entity model. If later the customer tells the analyst, that some entries in the database have a period of validity "which can also start in the future", he will proudly point to the `valid_from/valid_to` attributes and claim that this problem has already been taken care of.

The first problem is usually, that your customer does not know exactly what he wants to achieve with a "history of all data". Often the wish for a complete record of all changes is driven by fear of loosing information or uncertainty about what information he might later need. "And anyway, disk space doesn't cost anything nowadays". According to my experience 90% or more of the historical data is never being used. And for the major part of them this could have been known before the technical solution had been built! Once more in our profession the solution of a real world problem is postponed into the technical phase (where of course it causes more problems and work, but what do I complain about....).

The standard `valid_from/valid_to` solution is often a superposition of distinct time related aspects. The principal error is the confusion about validity of storage of data and the validity of its impact on the environment. Often there are several meanings (such as valid, active, visible etc.) which are all packed into the standard temporal attributes. That this undifferentiated solution leads to complex algorithms and faulty programs is not astonishing but unavoidable.

Apart from technical drawbacks there is a major theoretical problem: the relational model does not know such a thing as an *invalid* record. An entry in a table is a true statement about the world. An invalid entry implies, that my database is not consistent. Even if defined for the record, if you start to analyze the semantic, `valid_from/valid_to` ALWAYS refer to an attribute (or group of attributes) value such as `status, prize, name` etc.

Iso- and chrono values

Principally we have to make the distinction between the history of the storage of the values (transaction time) and its temporal validity or applicability of the values (valid or effective time). This is the "bi" in bi-temporal.

The validity of storage ("what value for attribute A was stored in the database at time T ") is a pure technical information (actually it is information about information or meta information) and can be maintained by the database system. This information is hardly ever used to carry out the users daily business. Typically it is used for statistics where you want to compare data with the data you had when you run the report last year. Or it is used for auditing purposes. This information is hardly ever used, but its mere presence can help to discipline the users. I refer to this values as iso-values[1].

The other kind of temporal[2] validity is of importance for the user. Here its main purpose is to define a period of time during which some attribute values have an effect, an impact on the users environment. I call this values chrono-values.

[1] iso = greek prefix meaning equal

[2] as I want to include the future from the beginning, I prefer temporal over historical.

Iso-values

Iso-values can have several physical representations for the same logical value over time. Correcting a misspelled value is a typical case for this kind of values.

Example: in a application we need to manage persons. We initially stored a first name "Sandi". Later it was corrected to "Sandy" and when the woman got her first letter she asked for having her first name changed to what she prefers to be addressed: "Sandra". All 3 values "Sandi", "Sandy" and "Sandra" stand for the same *real* value "SANDRA".

The memory image on the time axis looks like Figure 1:

(t0 is the time the attribute was entered into database. t3 is the time it was deleted.)

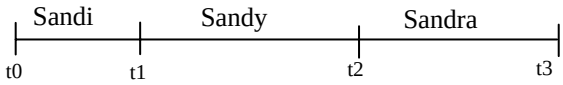

Figure 1

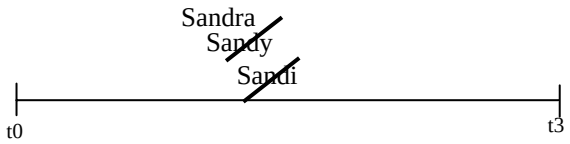

Figure 2

The way the user sees it can be expressed as in Figure 2. This is the way it would look like on a hand written index card for example.

This means the first name had different values, but they no longer are of interest to me. (Except maybe in a statistic about committed errors).

Some remarks

- At every moment the actually stored value was valid for the whole life span of the attribute.
- There never were different first names. There were only different representations for the same real-world fact.
- Changes of iso-values can only be in the past. They are really *historical* values.
- The moment of the change of an iso-value is often referred to as transaction time. It marks the point in time when records in the database changed; not when something in the real world changed.
- The attribute `first name` had the same (=iso) *real* value "SANDRA" throughout the complete recorded period.
- This situation can be characterized by questions such as:
  What would have been her name, if I had asked between t1 and t2? Answer: "Sandy"
  What was her name before t1 (asking between t2 and t3)? Answer: "Sandra"
- The information can be retrieved by using an *asking point in time* in your questions.
  What did my database look like at some time in the past?

Chrono values

Semantically a complete different thing (although sometimes overlapping regarding values) is the validity of properties of an object. Chrono-values can have several meaningful different values over time. Let us look at the situation in Figure 3 where Sandra got married and later divorced:

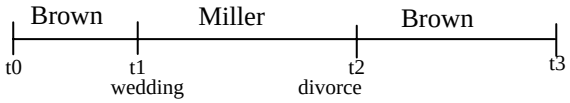

Figure 3

Some remarks

- With the value the meaning for the attribute `Name` has changed. "Miller" is not a different representation for "Brown", it is a different name.
- The attribute name has changed it's meaning over time (=chrono).
- Points in time or periods of chrono-values are often called "valid time / valid period". They indicate the period of time during which the value is valid, and therefore has relevance for the user(s business).
- Every value has a limited validity of its effect.

- There is no difference between past or future periods (including all rules defined for those attributes). Refering to such data as "historic data" in this case is wrong or at least misleading.
- This situation can be characterized by questions such as:
  What was her name before her wedding (t1)?
  What will be her name after her divorce (t2)?
- The information can be retrieved by using an *effecting point in time* in your questions.
  What does my database 'say' about a value, valid at some point in time in the past, presence or future?

Iso-Chrono-Values

Chrono-values can be iso-values as well. Every attribute can change the representation value during its validity period. Note that this can even be the case for the attributes defining the period (`valid-from/valid_to`)! For every period there can be several actual values representing the *real* value. Lets take the example from Figure 3: Sandra actually got married to a man from a German speaking country. Her new name was correctly entered as "Müller", soon changed to "Mueller" to avoid the missing umlaut problem and after some time she was fed up with explaining the spelling so she had it changed to "Miller". This leaves us with Figure 4

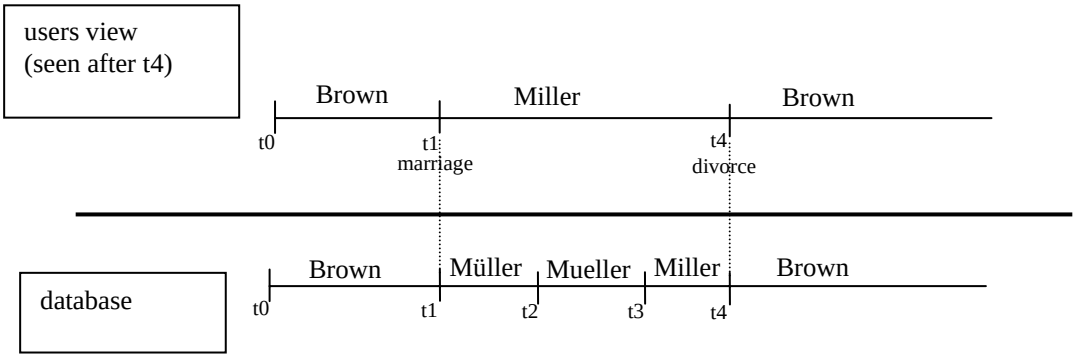

Figure 4

Remark

- From the users point of view (the *effecting time* viewpoint) the values "Müller", "Mueller" and "Miller" were all valid during the whole period t1 - t4. Which one was valid depends on the *asking time*:
- This diagram simplifies a bit: you can change chrono- and iso-values in the past as well. This leads to new tricky situations which you better try to avoid by "discussing them away". Anyway they are beyond the scope of this paper.

A general solution

With the knowledge of the two kind of temporal values, let us have a look at the problem of modeling the requirements of temporal data. As iso-values represents states of the database (they can actually be considered meta-information), the database can handle the journaling itself. All you need is some generic solution or some generator such as the journaling features of Oracle-Designer. Unless your customer has good reasons to model this kind of information, you can simplify your life considerably with some general rule such as "every database change will be recorded in extra journal tables". If you don't want a journal for all data, define some standard way how and where you document the specific requirements for single entities or attributes.

As stated in the introduction: adding the attributes `valid_from/valid_to` to every entity is not a good solution. Once the iso-values are removed from the problem, you might never consider this as a solution again. Let us look at a real life example from a traffic control application. Measure points along roads count the traffic passing the point. They typically count the number of vehicles passing during a defined interval. For every measure point we need to know its exact position which we store as coordinates. For simplicity reasons I omit the whole lot of other attributes needed to make Figure 5 a real example.

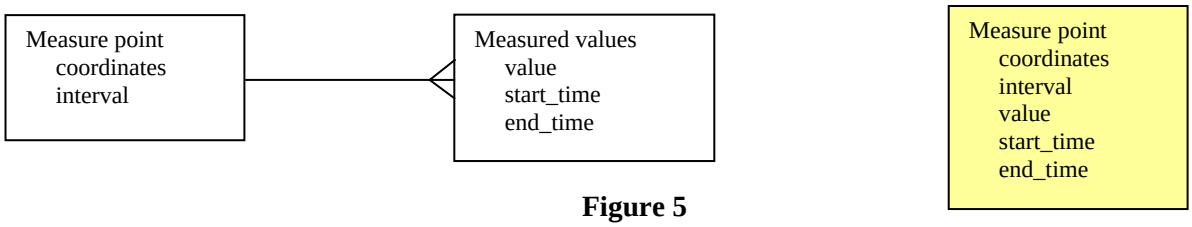

Figure 5

Never ever would a mentally sane datetween a changing measured value and some other values? Is it because the values can change every quarter of an hour? Is the number of expected records or the frequency of changes really the crucial reason?

The measure point can be moved on the road temporally or permanently, the interval can change. All of this changes can have an impact on the statistics created out of this information. Therefore we have to know what the values were during which period.So how can we model then changing attributes coordinates and interval properly? Often it IS tempting, to declare the simple `valid-from/valid-to` solution as the generic global solution. It saves you a lot of specification time and brings you closer to the day you can finally start coding. A better solution you see in Figure 6:

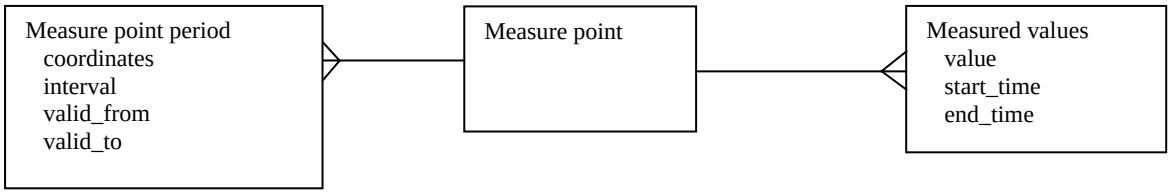

Figure 6

Don't worry about the empty entity "Measure point". The real thing had much more attributes and I'll address the problem of the empty entities further down.

The solution in Figure 6 causes new problems, if `interval` and `coordinates` change at different rates (which actually is the case) you might need to place every attribute in a table of its own. You end up with a crowded diagram (see Figure 8 below) which your customer refuses to read or verify (and you can't blame him for that).

So how can we improve the description without loosing any of the information we need?

The relational model is something like the Turing machine of information modeling. It offers everything needed for the modeling of temporal data. An extension of the *model* as often proposed in literature is in my opinion not necessary[3]. What we need are enhancements of the documentation and (graphical) representation of the model.

Periods of validity always affect the values of an attribute or a group of attributes. As long as the time period is the only extra information, the creation of the extra detail entity can be automated. So why not just mark the attributes in question and leave the extra work to the programmer or better to a generator?

[3] If somebody finds a model enhancement which is a real improvement, I would be the first to use it. But so far I have read a lot and found little convincing.

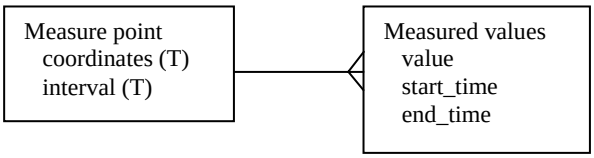

Figure 8

In Figure 7 the signification of (T) behind an attribute is:
The value of this attribute can change over time. We have to keep record of the exact time when this change happened.

Actually the "full grown" diagram would look like Figure 8 as the definition says nothing about grouping temporal attributes.

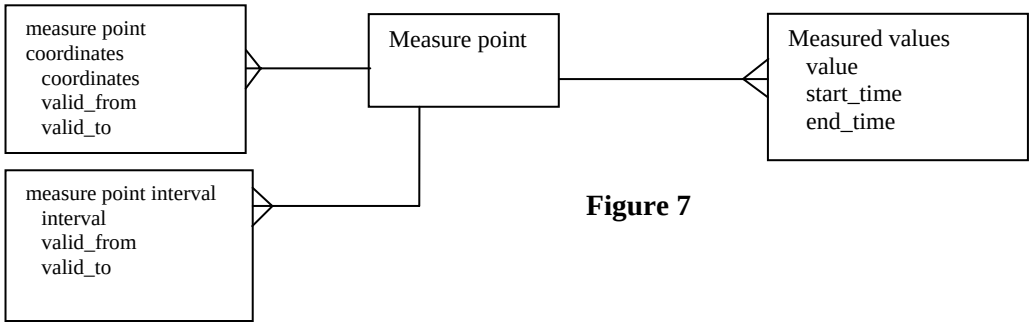

Figure 7

Note that this definition does not imply an implementation as a separate table or in fact any preference of an implementation. It is a mere specification of the fact, that interval and coordinates are chrono values.

Sometimes it makes sense for or is requested by the user, that attribute values can or have to change together (e.g. the components of a postal address). In that case (and in the absence of objects or attribute groups in our design language) you should specify which attributes belong to the same temporal group. This can be indicated with a Gn (n being the number of the group). In addition to that, you might need different granularities. We can extend the T-syntax by an indicator for the granularity "Tx" where x stands for y(ear), m(onth), w(eek), d(ay), h(our), mi(nute), s(econd), m(ili)s(econd). Attributes in the same temporal group must have the same granularity.

Note that the `start_time, end_time` attributes in measured values entity could also be expressed with the (T) syntax. But in that case this two time point are values which are measured, transmitted and validated. In order to provide as much information as possible I would leave them as proper attributes in the model. On top of this, the real entity had more attributes and was an entity which made lots of sense to be noticed by the users. This decision is of course based on the same arguments used for the decision not to use the yellow solution in Figure 5.

What if the measure point itself has a limited validity? Say it was only active from 1995 until 2003. Do we add `valid_from/valid_to` to the base entity (and spoil the whole concept)? "active" is the point. This is a new temporal attribute (with possible values true or false) in its own group.

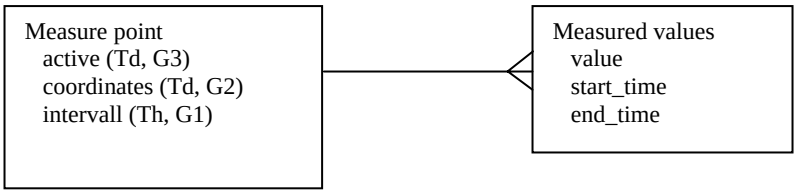

Figure 9

With this approach you will never have problems if you learn, that the measure point is to be included in statistical calculations for a longer period than its active period. Simply add a temporal attribute `statistic_include` and you will never have problems with different interpretations and cumbersome calculations of the applicable period.

For the validity of objects (to be exact instances of objects) there can always be found an attribute such as `active, inactive, valid, state` etc. which -if made temporal- can express its different states over time.

Relationships

What about relationships which can change over time? Here the chosen approach results in a even greater benefit. Consider a model as in Figure 10:

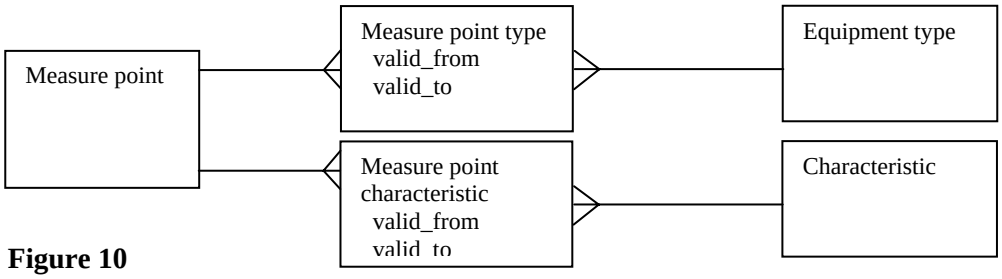

Figure 10

Some measure points can measure different characteristics (number, length, speed, weight of vehicles) concurrently. But they do not measure all of the characteristics all the time, which means the relationship is temporal as well as it is many to many.

But why should a measure point be of many equipment types? The answer is of course "over time". This means the model in Figure 10 can only be understood correctly if you know about the business rule "measure point types may not overlap". Consider now the solution with the (T)-syntax:

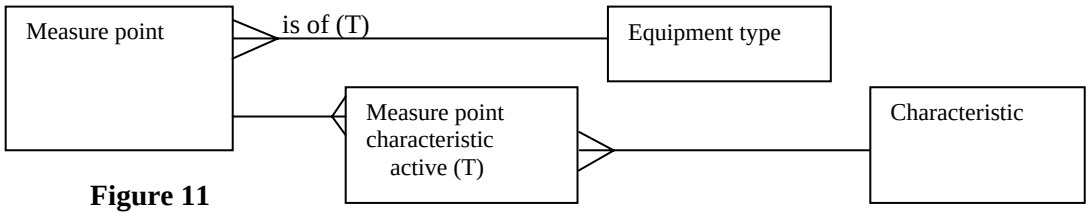

Figure 11

The diagram becomes much more readable, contains the information needed to understand the users requirements and saves you a lot of rearranging time in the diagram. The semantic is: a) one measure point is of one equipment type at any given time, this assignment can change over time and we have to record the times of changing, b) one measure point can measure several characteristics, this assignment has a limited validity and we have to record the times of changing of its active status. Again all the information is present and the diagram becomes much simpler and easier to understand.

Empty entities

Somewhere above we had an example of an empty entity (with no attributes). It was also said, that every real world object has non-temporal attributes. Both is correct. There are situations, where we are not interested in these attributes, which leaves us with an empty entity and later with a primary-key-only table. So what? Who is afraid of a primary-key-only table? You need that key in order to guarantee uniqueness.

BUT: If only the artificial key is fix, how do you know that two instances of an object refer to the same real thing? All attributes can change! If there is no unique attribute left in the non-temporal part, you have an area which is worth a thorough review.

Business rules

You can express your business rules in terms of the (T) syntax. This will simplify a lot of the rules. And some even 'disappear' as they are intrinsic to the (T) concept.

Examples:

- a chrono attribute can by definition never have an overlapping validity within the same non-temporal master
- uniqueness of a temporal attribute is the equivalent of non-overlapping values at any time over all instances.

It will be useful to define a set of operators such as NO-OVERLAP, WITHIN, NO-GAP, UNIQUE(T) etc. with a well defined semantic. This again can simplify your business rule specification a lot.

This topic of business rules for temporal data is interesting enough to fill a paper of its own. So I leave it here with some hints, that you should look into it.

Documentation

My idea how the (T)-syntax can be documented: as it is an important information, it should be on the document you discuss with the users. The simplest solution I found was to include it as part of the attribute name / relationship name. This was the easiest way to have it displayed in the ERM-diagram of Oracle designer.

Another possibility is to add new properties to attribute and relationship definitions or use the note/descriptions properties. (It would be nice to have some tools to create normalized tables out of your temporal, granularity and group information.)

The iso-values requirements you should only document in the diagram if it has a special significance. Usually it will do, marking an entity or a list of attributes with a J(ournal) flag, be it as extra property, be it as remark in the description or notes.

Design of a temporal database

Iso-values can be implemented with journal tables. Although checking the journal table checkbox in Oracle Designer is not the whole business, I won't get into details here.

Designing temporal tables requires some steps and decisions in addition to the normal design process:

1. Valid_to included / excluded
   Is the end-time point the last moment of the period or the first after the period?
2. Do you store open end time points as `null` values or as a maximum date value?
3. Do you store end times or is the start time of the next entry the implicit end time of the last one?
   (See also temporal NULL's below)
4. What is the typical access to the data a) as a list of all historical changes, b) as a snapshot at a defined point in time?
5. How will you differentiate between iso- and chrono-updates and -deletes?
   When (and how) do you create a new or delete an old record and when do you change the content of an existing record? How do your modules handle this difference? How is this resolved in the UI?
6. How many tables do you create for temporal objects?
   one for all attributes (with one or several `valid_from/valid_to` pairs)
   one for the fixed and one for all temporal attributes
   one for the fixed and one for every group of temporal attributes?
7. Is the journalized data part of your application or just backup information which has to be accumulated?

1) and 2) should be decided once for your application and handled consistently.

For 3) - 7) you will need some general scheme but every entity can be subject to a special solution. These decisions have to take into account arguments such as security, the amount of data, typical usage, availability of templates and frameworks etc. Again I won't go into details here as this is expected know-how of a database professional and would fill a paper or book of its own. There are frameworks or templates available, addressing the problems of overlap checking, temporal uniqueness and so on. You might have some parts already implemented in your own environment.

I would like to address 3 aspects I think could be of general interest. This are only outlines of ideas, not complete solutions. Every of these concepts causes new problems and has its own pitfalls which will not be covered in this paper. All I want to do here is showing you a path to a solution. If you have questions or remarks, feel free to contact me.

Cutoff day view

Most often you will choose an implementation where historical data is stored in the tables and a snapshot or cutoff day view (= view of the data at a chosen day) will be implemented by means of views, api's or redundant data. Wouldn't it be nice to have parameterized views giving you the state of your data at a selectable date?

Here is a solution we implemented in a project and which served us well: You need a place to store the time to be used for the select. This can be a temporary table containing a name and a time point per record:

TIME_TABLE

PIT_NAME	PIT_TIME
Now	2005/04/26
my_date	1999/12/31
nextMonth	2005/05/01
application_time	2005/05/01

In this table you can enter any date you wish and give it a name to identify it. Being a temporary table, the values are private to your session (the solution works as well with a normal table).

Now you can create views collecting all the fixed and temporal attributes of an object for a certain date:

```
CREATE VIEW measure_point_COD AS     -- (CutOff Day)
SELECT pit_name, pit_time
       ,mp_id, mpi_interval, mpo_coordinates
  FROM time_table
  CROSS JOIN measure_point
  JOIN measure_point_int ON (mpi_mp_id = mp_id  -- join the interval-attribute
                             AND mpi_from <= pit_time    -- restrict the period
                             AND mpi_to > pit_time
                            )
  JOIN measure_point_coo ON (mpo_mp_id = mp_id   -- join the coordinates attribute
                             AND mpo_from <= pit_time   -- restrict the period
                             AND mpo_to > pit_time
                            )
```

This gives you a record of a non-temporal measure point for every date which is in the time table. You can now select this data from the view meausre_point_COD by restricting the select:

```
SELECT *
  FROM measure_point_COD
 WHERE pit_name = 'Now'   -- returns the measure point as it looks today
```

You can define a standard application time (e.g. set at the start of you application, or changeable by the user at any time in the UI) entered into your time-table. You can then use this entry as the default in your application or even build a new set of views on top of the first level cutoff day views. This leaves you with a non-temporal view of your data at a user-definable time.

```
CREATE VIEW measure_point_NT     ---(Non Temporal)
AS SELECT *
     FROM measure_point_COD
    WHERE pit_name = 'application_time'
```

Some remarks and ideas:

- Do not select with the pit_time in the where-condition as two entries in the time-table could have the same date.
- If you create this kind of views for every base (=non temporal) table, you can join the views and you can work with a complete set of non-temporal 'tables' on a definable day.
- You can compare data at two (or more) dates by selecting twice (or more) from the same view with a different pit_name in the where condition.
- You can define a view on top of your time table which 'adds' often used entries such as 'Now', 'Next Month' etc.

```
          SELECT pit_name ,pit_time FROM time_table
   UNION SELECT 'Now' ,sysdate FROM DUAL
```

- You can add to your COD-views the start- and end-date of the period during which the selected set of values is valid.

Chrono updates and deletes

Another problem you have to solve is the duality of updates and deletes. How can you differentiate between an iso-change (a value is corrected, physical delete) and a chrono-change (a value is replaced by another, logical delete)? You will have to answer this questions in your user interface as well: how can the user choose which kind of change he wants to make? How does he tell the system *this is a correction of a spelling error* or *this is a new value.*

In the database you need a second kind of update and delete statement. If the temporal data is normalized, the normal SQL-update and -delete are iso-changes. A chrono delete is an update of the valid_to-attribute, a chrono update is an update and an insert. One solution is writing special API's, providing the functions and procedures to deal with this case. You will probably need more than just updates and inserts if you have to handle complex business rules (overlapping, within conditions etc.) as well.

One nice solution is to implement INSTEAD-OF triggers for the above mentioned cutoff day views. You can define the update on this view as a chrono update (handling the logic in the triggers) whereas the update on the base table is an iso-update.

Or you can define two sets of views (one iso, one chrono) with according instead-of-triggers and completely disallow updates on the base tables.

Temporal NULL

A temporal NULL is the situation where you don't have a value for a certain period, where you have a gap in your temporal flow. This gaps can lead to a lot of logic in selecting and checking the data. To only name a few: gap, overlaps, outer joins, within constraints etc. Not having gaps allows you to omit valid_to altogether, as the new valid_from automatically terminates the preceding entry.

Some of the problems can be solved in an elegant way if you don't allow gaps. If a measure point is not active for some time, enter a record specifying the period of inactivity (`active = FALSE`). In other cases add a new state (`not relevant, inactive, removed` etc.) to the list of possible states. Of course this solution creates new problems. Make sure you analyze the situation thoroughly and balance the pros and cons.

Conclusion

The whole business of temporal data is very complex. To address it in a promising way you have to structure it. My suggestions:

- find out what the real requirements are and solve only those
- treat iso- and chrono-values independently in the analysis AND in the implementation phase.
- find a method to document and present temporal aspects in your requirements documentation. Make sure the user understands the problem and the solution.
- find and define generic solutions for the different design and implementation aspects: (de-)normalization, temporal business rules, historical vs. snapshot view
- make sure you have somebody with experience and good know-how in charge of you temporal data concepts.

Stefan Berner is a principal consultant with Diso Solution AG in Guemligen, Switzerland.
e-mail: sberner@diso.ch