Dealing with time in databases and data models: design

Prof. dr. Bas van Gils

Bas.vanGils@strategy-alliance.com
Managing partner @ Strategy Alliance
Professor @ Antwerp Management School

Introduction

In this article, I will discuss the use of "time in data models." I've been thinking about this topic for a while now and must admit that my thinking has evolved quite a bit over the last few months. I don't think I have a complete answer yet so I will give an overview of my current thinking. Suggestions for improvement are more than welcome!

Terminology

I will first have to explain some terminology. Most notably, I will have to define the terms data and data model. I will try to be brief and refer to reader to my book *Data in Context* (link at Amazon) for more details.

In my view, data represents our understanding of a domain in such a way that it can stand for that domain. The corollary is that data is a representation. It is something we can touch/ handle/ manage/ use. Referencing the title of the book, it is important to note that data that may be good enough (and thus: can stand for reality) in one context may not be good enough for another at all. Along the same line, a model is a purposefully created artefact that captures our understanding of a domain in such a way that it can stand for that domain. Again, there is room for subjectivity: a model that is good enough for one purpose may not work for another purpose. Typical purposes are a) to gain and communicate understanding about some domain, b) to design a database that stores data about that domain, c) to analyze problems in that domain, etc.

There are similarities between data and model, particularly in the sense that they are (subjective) representations about a domain. My view is that the relationship between these two concepts is mainly in the form of abstraction. A short explanation will have to suffice: a model can either represent an abstraction of the domain that we are considering, or it could represent an abstraction of data which is an abstraction of a domain itself! In order to assess whether data can, indeed, stand for a given domain, my claim is that data must have structure and meaning. In my view, the relational model - particularly the way Chris Date explains it in e.g. *Database Design and Relational Theory* - (link on Amazon) is a good way to handle structure and meaning. As a (very/too) brief introduction:

- A database consists of a set of relations that are assigned to relation variables (relvars). Relations are often called tables, but there are noteworthy differences.
- A relation consists of tuples (rows) of the same type.
- A Tuple represents a proposition about the domain and are therefore either true or false.
- The header of a relation represents its predicate which is neither true or false. Think of a predicate as a proposition "with holes in it".
- Tuples consist of typed values with a semantic indicator.
- The information principle states that all information in a database is represented in exactly one way: by explicit values in attribute positions in tuples in relations.
- The closed world assumption states that, loosely, everything that is in/can be derived from the database is presumed to be true, whereas everything that is not in/cannot be derived from the database is presumed to be false.

There is much more to say about this, but this short introduction will have to do for now. By way of example, consider the following situation:

PERSON

FNAME: NAME	LNAME: NAME	BDATE: DATE
John	Doe	01-dec-1976
Mary	Doe	12-nov-1975
Mary	Watson	13-Mar-1977
Ed	Walsh	NULL

The relvar PERSON is currently assigned to a relation with 4 tuples. The predicate is: the PERSON with first name FNAME (of type NAME) and last name LNAME (of type NAME) has birthdate BDATE (of type DATE).

Filling in the values of the tuple that is shown first leads to the proposition: the PERSON with first name "John" and last name "Doe" has birthdate "01-Dec-1976" (this is only slightly sloppy: I left out the type names of the explicit values in this proposition). To verify whether this specific proposition is true or false, we could ask for a birth certificate and do a check.

The "tuple" (quotes deliberate) that is shown last is particularly interesting. The marker NULL denotes the absence of a value. To repeat: it is not a value but signifies the absence of a value. Trying to write out the proposition would lead to: The PERSON with first name "Ed" and last name "Wash" has birthdate "NULL". Note that this is a predicate: there is still a "hole" in it so we cannot assess whether it is true or false.

Thinking about time

Time is a tricky concept: many books and scientific publications have been written about it. Since *data* and *models* are representations, our main objective is to try to understand how to represent time. Therefore, I will not go into the philosophical discussion about what time is for now and focus mainly on how to represent it in a data model. Also, I will mainly work through some examples to share my thinking.

Transaction time and valid time

The first observation is based on the book *Developing time-oriented database applications in SQL* by Snodgrass (see e.g. Amazon). The book makes several interesting observations (yet I also have some conceptual issues with the theory that is presented). The distinction that interests me is indicated as transaction time versus valid time. The two denote when we become to know something and when it is valid respectively.

To illustrate the difference (very loosely, for now), consider again the PERSON relvar. Suppose we add an attribute (column) with name RDATE of type DATE such that the predicate for the relation would become: The PERSON with first name FNAME (of type NAME) and last name LNAME (of type NAME) has birthdate BDATE (of type DATE) which was recorded in the databases on RDATE (of type DATE). The RDATE attribute would be an example of a transaction time as it indicated when something became known to us. The BDATE would be a valid time (and in this case it is unlikely to ever change, barring the case where someone made a mistake in reporting a birth date).

This distinction allows us to reason about birthdays, who is older than whom etc. It also allows us to reason about what we knew at a specific point in time: it could be that we know someone, but didn't find out when s/he had her birthday until after a fact and therefore we were, logically, unable to send a card in time. Useful and interesting.

Time intervals

A second thing to worry about is time intervals. For the time being, I will set the valid/transaction time discussion aside and only focus on valid time. Trying to represent this well in databases/ data models has been bugging me for a long time. Not long ago, I reached out to Chris Date to ask his opinion on the matter. I thought I had read most of his major publications, but it looks like I missed the one most pertinent to the topic at hand: *Time and Relational Theory: Temporal Databases in the Relational Model and SQL* which he wrote together with Hugh Darwen and Nikos A. Lorentzos (link on Amazon).

Needless to say, I obtained a copy of the book (second edition) and have been studying it for close to two months now. The book is complex and densely written but I think I'm starting to understand its main points.

I'll work my way through an example once more. In this case, the example is about the period in which you are insured for something. In this fictitious world, you are insured from the moment you start paying until you stop paying – simple is that. Let's say payments are monthly to simplify the matter somewhat. The thing is: when you purchase your insurance for the first time,

the end date is probably still unkown! An approach that I often see in practice is:

INSURANCE

PRS: NAME	SDATE: DATE	EDATE: DATE
John Doe	01-jan-2002	31-mar-2002
Mary Doe	01-feb-2002	NULL

Leaving out the data types for convenience, the predicate for this relvar is: The INSURANCE of the person with name PRS starts at SDATE and ends et EDATE. Problem solved? Or is it!? Note that the tuple for John Doe clearly represents a proposition about the real world. However, the tuple for May Doe does not: it is a predicate (it is a proposition with a "hole" in it). This leads to all kinds of nasty things, so will discard this first attempt at a design for our database.

As a second attempt, we could split up the relation: one relvar will focus on the start of insurance intervals and the other on the end of insurance intervals. It would lead to something like:

ISTART

PRS: NAME	SDATE: DATE
John Doe	01-jan-2002
Mary Doe	01-feb-2002

IEND

PRS: NAME	EDATE: DATE
John Doe	31-mar-2002

The predicates for these relvars should be obvious. Certainly this design seems a little bit better: we lost the NULL "value". There are issues still, though. First, we have to make sure that there can only be an entry in IEND if there is a corresponding entry in ISTART. This can be achieved through a foreign key constraint. Second, we need to ensure that the end date (in IEND) is after the start date (in ISTART). This can be achieved with a constraint such as:

```
CONSTRAINT CorrectDates IS_EMPTY
  ( ( ISTART JOIN IEND ) WHERE
    EDATE < SDATE )
```

For the purpose of understanding (i.e. creating some conceptual model that helps to understand this domain), this probably works well enough yet in more advanced examples you'll quickly run into a wall. However, when the purpose shifts to designing a database, this will cause further (performance related) issues with the amount of JOINs you'll have to do. There simply has to be a better way of dealing with this.

In a third attempt, I thought about introducing a new type called DATE-INTERVAL. The idea is that [01-jan-2012 , 31-jan-2012] would denote a specific interval which include the start date and end date, whereas [01-jan-2012 , 01-feb-2012) would denote an interval with the start date included and the end date excluded. If the "grain" of our DATE-INTERVALS is one day (i.e. we only consider calendar dates) then the two intervals are conceptually the same.

Aside: they are two possible representations (POSREPs) of the same interval in the real world. End of aside.

This is also where I went wrong initially. I thought about representing an open-ended period (from now until someone decides we have to stop) as [01-jan-2012, ∞), where ∞ denotes *infinity*. Since isn't a real value, this is a direct violation of the information principle, the very foundation of relational theory. Therefore, we should reject this design: it would be similarly bad as allowing NULL "values".

Going through the aforementioned book by Date, Darwen and Lorentzos, I found another solution which seems to work well. The idea is to work with a SINCE-version of a relvar and a DURING-version of the relvar as illustrated below:

ISTART

PRS: NAME	SDATE: DATE
Mary Doe	01-feb-2002

I-DURING

PRS: NAME	DURING: PERIOD
John Doe	[01-jan-2002,31-mar-2002]

Again, the predicates should be straightforward. As before, we have to be very careful with some constraints between the two relations to ensure that they work correctly (that is: to ensure that they correctly represent our understanding of the domain such that it can stand for that domain). These are beyond the scope of this article - yet I will address them in a future article on implementing this type of solution.

It should be easy to see that a set of operators on DATE-INTERVALS would be helpful to determine which intervals overlap, are touching, etc.

It is not super difficult to define these conceptually. However, implementing them in an SQL-database might be tricky for several reasons. To give a first idea:

- Working with dates and times in SQL is a nightmare. I also believe (but still have to convince myself) that different relational database management platforms implement the standard slightly differently which is not helpful at all.

- Defining types of your own in SQL is possible, but you have to know the standard AND the specifics of the underlying platform to get it right.

- SQL has issues (i.e., the NULL-problem) which complicates matters further, particularly when it comes to defining proper integrity constraints.

Conclusion

In this article, I have discussed some of the issues around dealing with time in data models and databases. The first one deals with valid time versus transaction time. It takes a while to wrap your head around it, but it seems doable. The second issue deals with intervals of time. Here I have only scratched the surface, mainly from a conceptual perspective. The journey so far can be summarized as:

- One entity/relation: all details in one place using an attribute for start date and an (optional) attribute for end date.

- One entity/relation for common properties of insurance, with separate entities/relations for the details about start and end dates of insurances.

- One entity/relation for common properties of insurance, with separate entities to track completed (DURING) insurance periods and ongoing (SINCE) insurance periods.

The point has to be reiterated: this data model only represents the conceptual exploration of the period in which an insurance is held. How such a model is represented in a (SQL) database has been touched upon only briefly.

In my view, implementation is a different ball game than design and I highly recommend readers to pick up the book by Date, Darwen and Lorentzos to get a better understanding. I know for a fact that I'll have to re-read it a few more times before it all becomes crystal clear.

For the time being, I have learned that dealing with time (and particularly with time intervals) should be trigger alarm bells when creating a design. It seems so easy to simply add some attributes to your entity types and then build/generate your physical data model. Hopefully this short exploration convinces you that such topics require more careful thought.

I hope you find this paper interesting. If you have some thoughts or comments, please feel free to drop me a note. I'll be thinking about this topic for a while longer. Thanks!