2011-03-15 - The LinkedIn Bitemporal Data Group

So here's an appropriate topic to begin with. What is bitemporal data, and why is it important?

In an important sense, all data is bitemporal. It is bitemporal because it can change for one of two reasons. It can change because what it represents has changed, or it can change because a mistake was discovered in the data.

Let's begin by focussing on rows in tables in databases. That's the kind of data we will be talking about. And to keep things simple, we'll talk about customers as objects, and the Customer table as the table whose rows represent customers.

An insert is a response to a change in the real world. For example, the business acquires a new customer. Let's say that that customer has come into existence. We then perform an insert on the Customer table to create a row of data to represent that new object. Similarly, if a person who was a customer ceases being a customer, then that person no longer exists as a customer of ours. To reflect that change, a delete transaction removes that row of data. And if a customer, for example, changes her name from Smith to Jones, then we submit an update transaction to change the string "Smith" to "Jones" in the name column of the row that represents her.

These modifications to the Customer table happen over time, of course; they happen one at a time. But they all have this in common, that they happen because things have altered in the real world. Customers have come and went; other customers have changed their names, or other things about themselves that we keep track of.

Suppose we wanted to keep track of all these changes so that we could see, for example, who our customers were on the previous July 1st, or a sequential list of all the addresses customer C-123 has lived at. Then we could add a time period to the Customer table. If we didn't have one of the new time period datatypes, we could add a pair of dates or timestamps. Before each update, we would copy the row about to be updated, and make its end date the date of the update. We would then update the row, using that same date as the start date, and insert it as a new row.

A table with a time period like this attached to it keeps track of changes in what computer scientists call valid time. I call this kind of time effective time, because that's what I've heard it called over several decades of consulting work. Many working IT professionals will call a table with this kind of time period added to it a history table, or a version table.

There are many different ways to construct history/version tables. Four of them are described in Chapter 4 of my book. But all of them have one fatal flaw. All of them get stuck when trying to keep track of changes made to correct erroneous data.

