

Abstract of a Presentation to the National ERWin User's Group: June, 2008.

Dr. Tom Johnston
MindfulData.com

Randy Weis and Tom Johnston have, between them, over fifty years experience in designing, building and supporting business applications and databases. They are currently co-authoring a bi-monthly series of articles which are archived at DMReview.com, and are downloaded directly to DMDirect subscribers. These articles describe a proposed best practice for managing temporal data in relational databases, using the RDBMS technology available today.

The management of temporal data in relational databases refers to the management of data which either (a) is currently effective, (b) was effective for some period of time in the past, (c) will be effective beginning sometime in the future, or even (d) might be effective at sometime in the future. In particular, it refers to managing that data in such a way that it is as readily and easily available as current data only. Thus, techniques and artifacts such as restoring backups and running logfiles forward to the desired recovery point, or taking periodic full database snapshots, are not considered in their articles. For a full taxonomy of these and other means by which non-current data is managed, see the taxonomy provided in Part 1 of this series, which appeared in the May 2007 issue of DM Review.

Temporal data management structures, and corresponding extensions to SQL DDL and DML, should be defined by the standards committees, and provided for us by vendors. Dr. Richard Snodgrass, one of the leading computer scientists in the field, wrote a proposal for temporal extensions to SQL, which was intended to be part of the SQL99 release. But due to objections primarily from C. J. Date and Hugh Darwen, his proposal was rejected by the British SQL standards committee. To the best of our knowledge, no progress towards reconciliation and resolution has been made in the ensuing years.

Randy and Tom are practicing IT professionals who find that the need for temporal database functionality, of the real-time access sort, is becoming increasingly important to businesses. They believe that we can't wait for Date, Darwen and the academics to shake hands and make up, and that we need a solution right now! But since temporal functionality will inevitably become part of RDBMSs at some time in the future, a key design principle they have adopted is to isolate the functionality as far as possible, to encapsulate it so that as commercial products begin to implement that functionality, the DIY (do it yourself) solutions described in their articles can gradually be phased out with minimal impact to existing business systems.

This three-hour presentation will be a combination of theoretical background and live access to a temporal database of (made-up data on) insurance clients, their policies, and claims against those policies. After explaining the temporal database structures they describe in their articles, and the role each temporal component plays, Randy and Tom will take us through an extensive set of both updates and queries against these structures.

In accordance with their encapsulation principle, updates will be written as though they would be applied to normal tables, but with the addition of a date or timestamp. Those updates will then be translated, and not necessarily one for one, into updates against these temporal structures. As for queries, they will be formulated against views of the underlying temporally-structured tables. Queries will appear to their authors, be they end users or programmers, as queries against ordinary tables, but with an "effective as of" date also included.

However, there will be some updates, and some queries which cannot be represented as "normal" updates or queries with a date added. These more complex transactions will be examined in correspondingly greater detail. One such update, of particular interest, is an update that corrects an error in a

temporal table, but also leaves the error in the table. The point of doing this is to support queries which ask, not “What was X like on date Y?”, but rather “What did we say X was like on date Y, when we were asked on date Z (and is this different from what we now believe X was like on date Y)?”

A final note. Randy and Tom realize, of course, that there has been an established best practice among IT professionals that has been around for many years. It is to “temporalize” a table by adding a date to its primary key. Their position is not that this best practice is wrong, but rather that it is incomplete, and that it lets the procedurally expressed logic of updating and querying these tables be distributed across application code and native SQL queries. They do not propose to replace this best practice, but rather to complete the changes to table structures needed to support full temporal functionality, and also to gather all the temporal logic of updating and querying these tables into an encapsulated data access layer. To the degree that their proposal achieves what it sets out to do, this will mean that for both end-users and developers, temporal functionality can be invoked declaratively. It will mean that as commercial DBMSs gradually do implement this functionality, the syntax of temporal access may change, but the semantics will remain stable. Application logic built around those embedded updates and queries will not have to change.