A Case Study of Temporal Data

By Richard T. Snodgrass,
Professor of Computer Science,
University of Arizona

TERADATA®
Raising Intelligence

A Case Study of Temporal Data

Table of Contents

Executive Overview

Information is the key asset of many companies – and for most, this asset contains time-referenced data. It can be frustrating that standard SQL has so few language facilities for such data. Fortunately, Teradata® Database 13.10 has many new features in its SQL designed specifically for temporal support.

This case study will examine how the new SQL language features in Teradata Database naturally reflect the expression in English of modifications and how these features greatly reduce the length and complexity of such modifications in conventional SQL. The result is that developers can now much more easily convert their applications to support time-varying data and create new applications that exploit stored data about the past for deeper insight into the future.

A Case Study of Temporal Data

Bitemporal Tables Supported

Take for example a mortgage company that is challenged with achieving high data quality on information stored concerning customers and their loans. A customer service person (CSR) reports an error to the IT personnel; errors are also discovered by batch jobs producing quarterly reports. The more information the IT personnel have access to, the better able they are to analyze and correct these errors.

For this reason, it is critical that changes to critical tables concerning customers and their loans be tracked. This implies that each such table (an example will be given shortly of a table recording who owns what property) has what is termed *transaction-time support*, to maintain a history of changes [Snodgrass & Ahn 1986, Snodgrass et al. 1996b]. As this table also needs to model changes in reality, it requires *valid-time support*, to indicate when the data were considered valid [Snodgrass et al. 1996a]. The result is termed a *bitemporal table*, reflecting these two aspects of underlying temporal support. Teradata Database 13.10 supports bitemporal tables, greatly easing the development of such applications.

With bitemporal tables, IT can first determine when the erroneous data were stored (a transaction time), rollback the table to that point, and look at the valid-time history. IT can then determine the correct valid-time history. With that history, IT can tell the CSR what needs to be changed, or, if the error was in the processing of a user transaction, IT may update the database manually.

Because transaction-time support is included, these changes will be logged as well, enabling someone later to see what happened when the change itself was in error. The support for both valid time and transaction time permits a sophisticated analysis of the evolution of the table, with all the data directly at hand. The alternatives – going back through paper records to reconstruct the sequence of changes that were made, or attempting to extract that sequence from backup tapes or other secondary data sources – are simply not practical in such a dramatically changing environment.

Following the Property

A property owner table captures both the history in reality of the owner(s) of a property over time, as well as the sequence of database states, denoting the transactions applied to this table. This bitemporal table is easy to specify in Teradata's SQL (note the new keywords VALIDTIME and TRANSACTIONTIME).

```
CREATE MULTISET TABLE Prop_Owner (
   customer_number INTEGER,
   property_number INTEGER,
   property_VT PERIOD(DATE) NOT NULL AS VALIDTIME,
   property_TT PERIOD (TIMESTAMP(6) WITH TIME ZONE)
      NOT NULL AS TRANSACTIONTIME)
   PRIMARY INDEX(property_number);
```

The valid timestamp is specified as having a granularity of day, as a property cannot change hands multiple times in a single day. The transaction timestamp is specified at a granularity of microsecond, to differentiate rapidly executing transactions.

The property_number column constitutes a primary key in both valid time and transaction time. Specifically, the state of the table at any day in valid time, as stored at any instant in transaction time, should include at most one row in the table for any particular property, meaning that that property has one owner at that valid time, as recorded at that transaction time. Because the table was declared to be bitemporal (via inclusion of both valid and transaction time), this temporal integrity constraint will be checked automatically by Teradata Database 13.10.

Valid-time state tables admit nine kinds of modifications: current, sequenced and non-sequenced versions of INSERT, DELETE and UPDATE. (We'll explain these terms shortly.) Transaction-time state tables are much simpler: only current versions of INSERT, DELETE and UPDATE are relevant. So, what is the situation with bitemporal tables? It turns out that here again only nine kinds of modifications apply, all current in transaction time: valid-time current, valid-time sequenced, and valid-time non-sequenced versions of INSERT, DELETE and UPDATE.

A Case Study of Temporal Data

Let's follow the history, over both valid time and transaction time, of an apartment in Boston at 123 Main Street for the month of January, 2010. On January 10, this apartment was purchased by Eva Nielsen. We record this information as a current valid-time, current transaction-time insertion.

When the database management system (DBMS) starts up, the default *temporal qualifier* is exactly that: current in valid time and current in transaction time. The English in italics followed by the SQL statement, using the new Teradata functionality.

Eva Nielsen (whose customer number is 145) buys the apartment at 123 Main Street in Boston (whose property number is 7797) today (which happens to be January 10, 2010).

```
INSERT INTO Prop_Owner (customer_number,
property_number)
VALUES (145, 7797);
```

This information is valid starting now. The DBMS encodes this information using the special valid-time and transaction-time columns, all under the covers. Note that the transaction-time extent of *all* modifications is from "now," in this case, "2010-01-10" to "until closed," which is encoded as "9999-12-31." This is also the valid-time extent, given the default temporal qualifier (which can be changed by the user).

customer_number	property_number	property_VT	property_TT
145	7797	(2010-01-10, 9999-12-31)	(2010-01-10, 9999-12-31)

The interplay between valid time and transaction time can be confusing, so it is useful to have a visualization of the information content of a bitemporal table (see Figure 1.)

In this figure, the horizontal axis tracks transaction time, and the vertical axis tracks valid time. Information about a row, or about multiple rows associated with a primary key value, are depicted as two-dimensional polygonal regions in the diagram. Arrows extending rightward denote "until closed" in transaction time; arrows extending upward denote "forever" in valid time. Here we

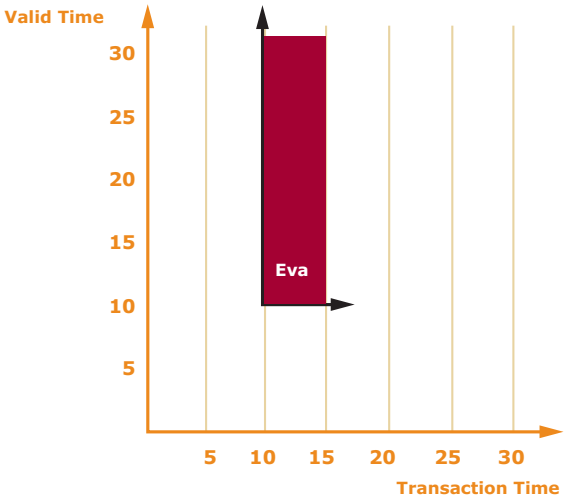

Figure 1: A bitemporal time diagram corresponding to Eva purchasing the apartment, performed on January 10.

have but one region, associated with Eva Nielsen, that starts at time 10 (all times are relative to January 2010, so '10' corresponds to January 10, 2010) in transaction time and extends to "until closed," and begins also at time 10 in valid time and extends to "forever." The arrow pointing upward extends to the largest valid time value ("forever"); the arrow pointing to the right extends to "now," that is, it advances day by day to the right (a transaction time in the future is meaningless).

Let's examine a simple update.

Today (which happens to be January 15) Peter Olsen (whose customer number is 827) buys this apartment, transferring ownership from Eva to him.

```
UPDATE Prop_Owner
SET customer_number = 827
WHERE property_number = 7797;
```

Figure 2 shows the *bitemporal time diagram* corresponding to the above insertion. The valid-time extent of a current modification is always "now" to "forever," so from time 15 on, the property is owned by Peter; at the rest of the time, from time 10 to 15, the

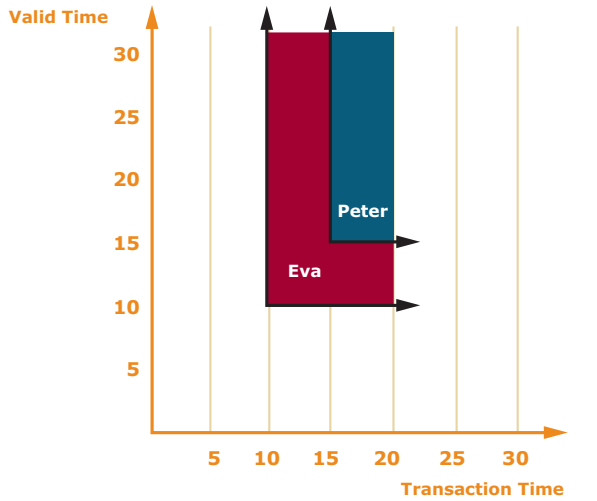

Figure 2. A current update: Peter buys the apartment, performed on January 15.

property was owned by Eva. Both regions extend to the right to "until closed." This time diagram captures two facts – Eva owning the apartment and Peter owning the apartment – each associated with a bitemporal region.

Additionally, this figure captures the evolving information content of the property owner table. Consider a transaction time-slice, which returns the valid-time history at a given transaction time. Such a time-slice can be visualized as a vertical line intersecting the x-axis at the given time.

At transaction time 5 (January 5), the table has no record of the apartment being owned by anyone. At transaction time 12, the table records that the apartment was owned by Eva from January 10 to "forever." If we time traveled back to January 12, and asked for the history of the apartment, that would be the response. We *thought* then that Eva owns the apartment, and that is what the property owner table recorded then. At transaction time 17 the table records that the apartment was owned by Eva from January 10 to January 15, at which time ownership transferred to Peter, who now owns it to "forever." And that is the history as best known (denoted by the right pointing arrows). It is what we think is true about the valid-time history.

Automatic Time Handling

In addition to contending with valid time, we also must ensure that the transaction-time extent of the modification is from "now" to "until closed." One important property of tables with transaction-time support is that they are append-only.

As these tables capture the state of the stored table over time, once we have recorded that the state was such and such at a particular time, we can't go back and change that later because we can't change the bits stored on the disk at that prior time. The changes always accumulate in the table with transaction-time support. The practical ramification is that we never physically delete a row from such a table; the only physical modifications allowed are to insert rows into the table and to change the transaction-stop time of a row from "until closed" to "now," thereby logically deleting the row.

Teradata Database 13.10 handles this automatically. The resulting property owner table contains three rows. A careful matching of the dates in this table to the time diagram will aid in understanding how a bitemporal state table encodes the regions found in the time diagram.

customer_number	property_number	property_VT	property_TT
145	7797	(2010-01-10, 9999-12-31)	(2010-01-10, *2010-01-15*)
145	*7797*	*(2010-01-10, 2010-01-15)*	*(2010-01-15, 9999-12-31)*
827	*7797*	*(2010-01-15, 9999-12-31)*	*(2010-01-15, 9999-12-31)*

In previous versions of the Teradata DBMS, all of this must be done manually. You are encouraged to implement this simple update in conventional (nontemporal) SQL. It requires a surprisingly complex series of five INSERT and UPDATE statements, some 31 lines in all.

All nine types of modifications allowed on bitemporal tables must be implemented as a combination of INSERTs with a transaction time of "now" to "until closed" (which is represented with "forever") and UPDATEs that set the transaction-stop time to "now." Any other modification to a bitemporal table will violate its

semantics. Fortunately, Teradata Database 13.10 handles this for us. So let's follow the continued evolution of the property owner table.

On January 20, we find out that Peter has sold the property on this day to someone else, with the mortgage handled by another mortgage company. From the mortgage company's point of view, the property no longer exists as of (a valid time of) January 20. This is captured by performing a current deletion on January 20 against this table.

Peter Olsen sells the apartment today (on January 20, 2010).

```
DELETE FROM Prop_Owner
WHERE property_number = 7797;
```

Figure 3 shows the resulting time diagram. If we now request the valid-time history as best known, we will learn that Eva owned the apartment from January 10 to January 15, and Peter owned the apartment from January 15 to January 20. Note that all prior states are retained. We can still time travel back to January 18 and request the valid-time history, which will state that on that day we thought that Peter still owned the apartment. In the previous diagram (Figure 2), Peter's region was a rectangle. The current deletion has chopped off the top-right corner, so that the region is now L-shaped.

The DELETE handles both current and future rows. The resulting table contains four rows. The third row was terminated at "now," with the fourth row newly inserted. The modified rows and columns are highlighted with a *slanted* font.

customer_number	property_number	property_VT	property_TT
145	7797	(2010-01-10, 9999-12-31)	(2010-01-10, 2010-01-15)
145	7797	(2010-01-10, 2010-01-15)	(2010-01-15, 9999-12-31)
827	7797	(2010-01-15, 9999-12-31)	(2010-01-15, *2010-01-20*)
827	*7797*	*(2010-01-15, 2010-01-20)*	*(2010-01-20, 9999-12-31)*

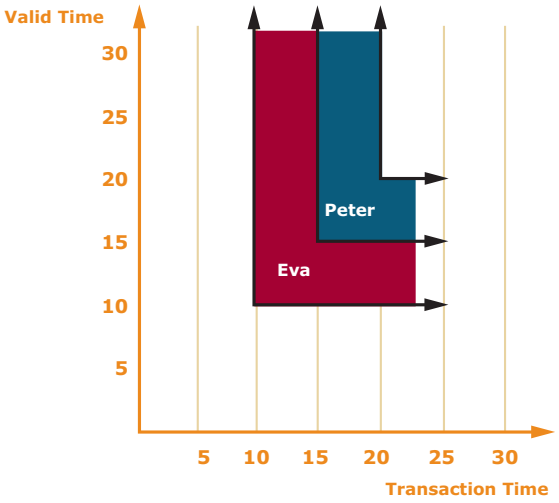

Figure 3. A current deletion: Peter sells the apartment, performed on January 20.

In current modifications, valid time and transaction time are coupled: the valid time at which the modification takes effect is "now." Similarly, the transaction time at which the modification is recorded is "now." Sequenced modifications decouple the valid time from the transaction time, allowing the former to be supplied by the user.

Sequenced modifications generalize current modifications to apply over a specified period of applicability. For bitemporal tables, the modification is sequenced only on valid time; the modification is *always* a current modification on transaction time, from "now" to "until closed."

We consider a sequenced insertion. On January 23, we find out that Eva had purchased the apartment not on January 10, but on January 3, a week earlier. So we insert those additional days, to obtain the time diagram shown in Figure 4.

A Case Study of Temporal Data

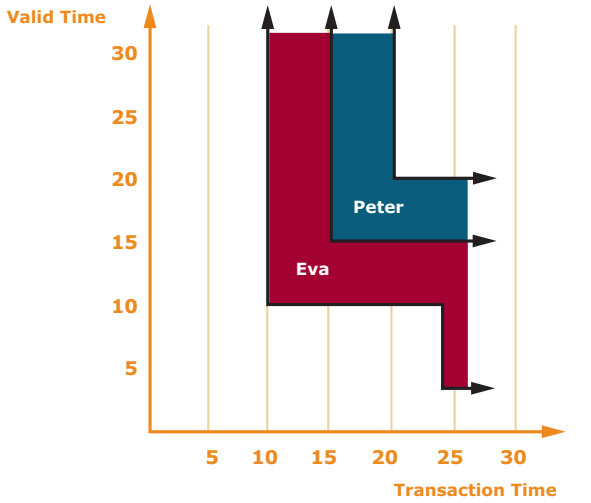

Figure 4. A sequenced insertion performed on January 23: Eva actually purchased the apartment on January 3.

This insertion is termed a *retroactive* modification, as the period of applicability (here, January 3 through 10) is before the modification date (here, January 23). Sequenced (and non-sequenced) modifications can also be *post-active*, an example being a promotion that will occur in the future (in valid time). (A valid end time of "forever" is generally not considered a post-active modification; only the valid begin time is considered.) A sequenced modification might even be simultaneously retroactive, post-active, and current, when its period of applicability starts in the past and extends into the future (e.g., a fixed-term assignment that started in the past and ends at a designated date in the future).

Eva actually purchased the apartment January 3 (performed on January 23).

```
SEQUENCED VALIDTIME
INSERT INTO Prop_Owner (customer_number,
property_number, property_VT)
VALUES (145, 7797, PERIOD (DATE '2010-01-03',
DATE '2010-01-10'));
```

We learn on January 26 that Eva bought the apartment not January 10, as initially thought, nor on January 3, as later corrected, but on January 5. This requires a sequenced deletion.

customer_number	property_number	property_VT	property_TT
145	7797	(2010-01-10, 9999-12-31)	(2010-01-10, 2010-01-15)
145	7797	(2010-01-10, 2010-01-15)	(2010-01-15, 9999-12-31)
827	7797	(2010-01-15, 9999-12-31)	(2010-01-15, 2010-01-20)
827	7797	(2010-01-15, 2010-01-20)	(2010-01-20, 9999-12-31)
145	*7797*	*(2010-01-03, 2010-01-10)*	*(2010-01-23, 9999-12-31)*

Eva actually purchased the apartment January 5.

```
SEQUENCED VALIDTIME PERIOD (DATE '2010-01-03',
DATE '2010-01-05')
DELETE FROM Prop_Owner
WHERE property_number = 7977;
```

We specify a period of applicability of January 3 through 5, with the result shown in the time diagram in Figure 5.

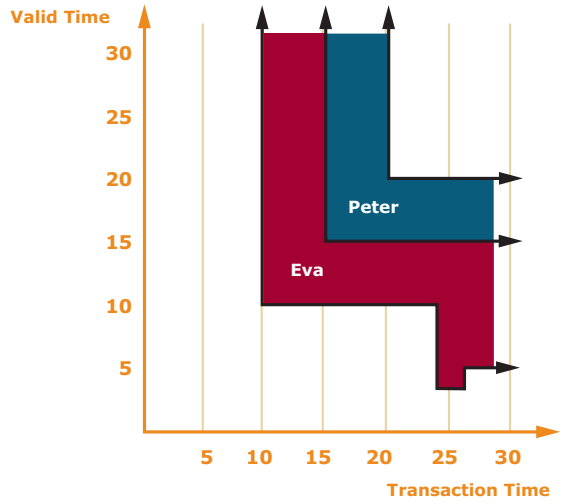

Figure 5. A sequenced deletion performed on January 26: Eva actually purchased the apartment on January 5.

A Case Study of Temporal Data

We need to terminate the current row, and insert a new row with a smaller period of validity, as shown below.

customer_number	property_number	property_VT	property_TT
145	7797	(2010-01-10, 9999-12-31)	(2010-01-10, 2010-01-15)
145	7797	(2010-01-10, 2010-01-15)	(2010-01-15, 9999-12-31)
827	7797	(2010-01-15, 9999-12-31)	(2010-01-15, 2010-01-20)
827	7797	(2010-01-15, 2010-01-20)	(2010-01-20, 9999-12-31)
145	7797	(2010-01-03, 2010-01-10)	(2010-01-23, *2010-10-26)*
145	*7797*	*(2010-01-05, 2010-01-10)*	*(2010-01-26, 9999-12-31)*

We learn on January 28 that Peter bought the apartment on January 12, not January 15 as previously thought. This requires a sequenced update.

Peter actually purchased the apartment on January 12.

```
SEQUENCED VALIDTIME PERIOD (DATE '2010-01-12',
DATE '2010-01-15')
UPDATE Prop_Owner
SET customer_number = 827
WHERE property_number = 7797;
```

This update states a period of applicability of January 12 through 15, setting the customer number to 827, which results in the time diagram in Figure 6. Effectively, the ownership must be transferred from Eva to Peter for those three days, resulting in the following table.

Figure 6. A sequenced update performed on January 28: Peter actually purchased the apartment on January 12.

customer_number	property_number	property_VT	property_TT
145	7797	(2010-01-10, 9999-12-31)	(2010-01-10, 2010-01-15)
145	7797	(2010-01-10, 2010-01-15)	(2010-01-15, *2010-01-28)*
827	7797	(2010-01-15, 9999-12-31)	(2010-01-15, 2010-01-20)
827	7797	(2010-01-15, 2010-01-20)	(2010-01-20, 9999-12-31)
145	7797	(2010-01-03, 2010-01-10)	(2010-01-23, 2010-01-26)
145	7797	(2010-01-05, 2010-01-10)	(2010-01-26, 9999-12-31)
145	*7797*	*(2010-01-10, 2010-01-12)*	*(2010-01-28, 9999-12-31)*
827	*7797*	*(2010-01-12, 2010-01-15)*	*(2010-01-28, 9999-12-31)*

A Case Study of Temporal Data

We saw before that no mapping was required for non-sequenced modifications on valid-time state tables; such statements treat the (valid) timestamps identically to the other columns. When considering the transaction timestamps, we just perform the second stage mapping discussed above.

As an example, consider the modification "delete all records for customer Eva." This modification is clearly (valid-time) non-sequenced: (1) it depends heavily on the representation, looking for rows with a particular value, (2) it does not apply on a per instant basis, and (3) it mentions "records," that is, the recorded information, rather than "reality." The result of this deletion, evaluated on January 30, is shown in Figure 7.

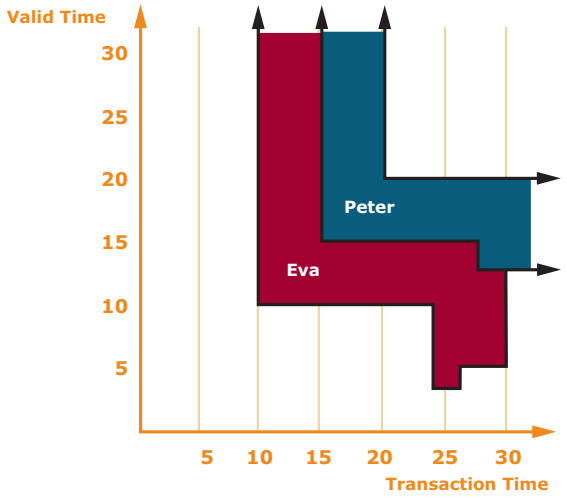

Figure 7. A non-sequenced deletion performed on January 30: Delete all records for customer Eva.

Delete all records for customer Eva.

```
NONSEQUENCED VALIDTIME DELETE Prop_Owner
WHERE customer_number = 145;
```

The result is Table 1.

customer_number	property_number	property_VT	property_TT
145	7797	(2010-01-10, 9999-12-31)	(2010-01-10, 2010-01-15)
145	7797	(2010-01-10, 2010-01-15)	(2010-01-15, 2010-01-28)
827	7797	(2010-01-15, 9999-12-31)	(2010-01-15, 2010-01-20)
827	7797	(2010-01-15, 2010-01-20)	(2010-01-20, 2010-01-28)
145	7797	(2010-01-03, 2010-01-10)	(2010-01-23, 2010-01-26)
145	7797	(2010-01-05, 2010-01-10)	(2010-01-26, *2010-01-30)*
145	7797	(2010-01-10, 2010-01-12)	*(2010-01-28, 2010-01-30)*
827	7797	(2010-01-12, 2010-01-20)	(2010-01-28, 9999-12-31)

Table 1: After a non-sequenced deletion.

In summary, a bitemporal table combines both valid time and transaction time into a single structure. It contains two period timestamps: the valid-time period of validity and the transaction-time period of presence. Primary key constraints on such tables are generally valid-time sequenced and transaction-time current. Expressing such a constraint requires a simple SQL PRIMARY KEY constraint.

We examined simple variants of temporal modifications. In conventional SQL, these statements can be long and complex: the worst case is one in which a non-temporal update of only a few lines expanded to some 60 lines of SQL. All are very natural to write given the temporal extensions provided in Teradata Database 13.10, requiring but a few lines.

Now that we have a populated bitemporal table, we can discuss bitemporal queries.

A Case Study of Temporal Data

Time-Slice Queries

A common query or view over a temporal table is to capture the state of the enterprise at some point in the past (or future), termed a *valid time-slice*. For tables with transaction-time support, one can also reconstruct the state of the monitored table as of a date in the past; such a query is termed a *transaction time-slice*. As a bitemporal table captures valid and transaction time, both the valid time-slice and the transaction time-slice are relevant.

Time slices are useful also in understanding the information content of a bitemporal table. A *transaction time-slice* of a bitemporal table takes as input a transaction-time instant and results in a *valid-time state table* that was present in the database at that specified time. Teradata Database 13.10 has special syntax for such time slices: TRANSACTIONTIME AS OF.

Give the history of owners of the apartment in Boston at 123 Main Street as of January 1, 2010, assuming a New York time zone.

```
SEQUENCED VALIDTIME AND
TRANSACTIONTIME AS OF TIMESTAMP '2010-01-01
23:59:59-05:00'
SELECT customer_number
FROM Prop_Owner
WHERE property_number = 7797;
```

Applying this to our table (as shown in Figure 7) results in an empty table, as no history was yet known about that property.

Taking a transaction time-slice as of January 14, effectively rolling the state of the database forward two weeks, results in a history with one entry.

customer_number	VALIDTIME
145	(2010-01-10, 9999-12-31)

On January 14, we thought that Eva was the current owner of that property. We now know that Peter purchased the property on January 12, and that Eva never owned the property at all on January 14, but that is 20-20 hindsight. The information we had on January 14 indicated that Eva bought the property on the 10th, and still owns it.

The time-slice as of January 18 tells a different story.

customer_number	VALIDTIME
145	(2010-01-10, 2010-01-15)
827	(2010-01-15, 9999-12-31)

On January 18 we thought that Eva had purchased the apartment on January 10, and sold it to Peter, who now owns it. A transaction time-slice can be visualized on the time diagram as a vertical line situated at the specified date. This line gives the valid-time history of the enterprise that was stored in the table on that date. Figure 8 illustrates this transaction time-slice.

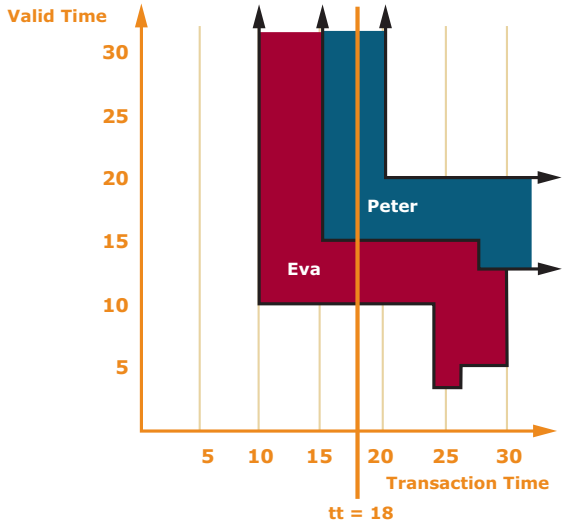

Figure 8. A transaction time-slice as of January 18.

Continuing, we take a transaction time-slice as of January 29.

customer_number	VALIDTIME
145	(2010-01-05, 2010-01-12)
827	(2010-01-12, 2010-01-20)

A Case Study of Temporal Data

On January 29, we thought that Eva had purchased the apartment on January 5, and sold it to Peter on January 12, who sold the property to someone else on January 20.

Finally, to take the current transaction time-slice, just omit the time, which defaults to "now."

Give the history of owners of the apartment at 123 Main Street in Boston as best known.

```
SEQUENCED VALIDTIME SELECT customer_number
FROM Prop_Owner
WHERE property_number = 7797;
```

This yields the following result:

customer_number	VALIDTIME
827	(2010-01-12, 2010-01-20)

Only Peter ever had ownership of the property, since all records with customer Eva were deleted. Peter's ownership was for all of 8 days, January 12 to January 20.

We can also cut the pie (or, more accurately, the time diagram) horizontally. A *valid time-slice* of a bitemporal table takes as input a valid-time instant and results in a *transaction-time state table* capturing when information concerning that specified valid time was recorded in the database. A valid time-slice is expressed in SQL similarly to the transaction time-slice, with VALIDTIME AS OF.

When was information about the owners of the apartment at 123 Main Street in Boston on January 4, 2010 recorded in the property owner table?

```
VALIDTIME AS OF DATE '2010-01-04' AND
NONSEQUENCED TRANSACTIONTIME
SELECT customer_number, property_TT
FROM Prop_Owner
WHERE property_number = 7797;
```

(A technical note: Teradata Database 13.10 does not support sequenced transaction time. But in this case, the same effect can be obtained through the use of NONSEQUENCED TRANSACTION-TIME and adding TT to the select list.)

Applying this time-slice to our table (as shown in Figure 7) results in one row, indicating that this information, that the property was owned by Eva on January 4, was inserted into the table on January 26 and subsequently deleted, as it was found to be incorrect, on January 26.

customer_number	property_TT
145	(2010-01-23, 2010-01-26)

The valid time-slice on January 13 is more interesting. Such a time-slice can be visualized as the horizontal line shown in Figure 9. This time-slice results in:

customer_number	property_TT
145	(2010-01-10, 2010-01-15)
145	(2010-01-15, 2010-01-28)
827	(2010-01-28, 9999-12-31)

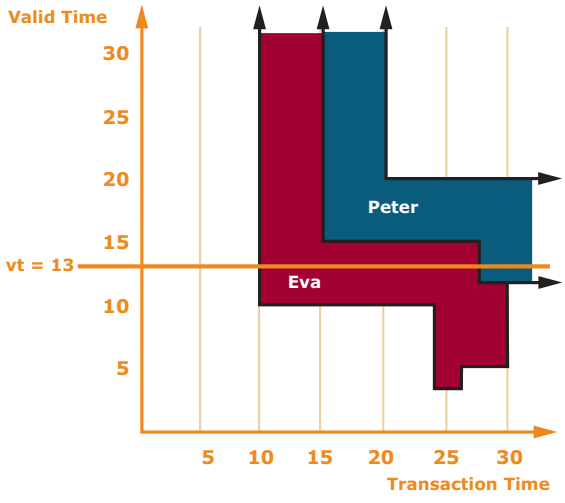

Figure 9. A valid time-slice on January 13.

A Case Study of Temporal Data

While the horizontal line in Figure 9 intersects two regions, *three* rows result from the time-slice. This has to do with the way that the regions in the time diagram are sliced up into rectangles, each associated with a row in the Prop_Owner table.

A *bitemporal time-slice* takes as input *two* instants, a valid-time and a transaction-time instant, and results in a *snapshot* state, of the information regarding the enterprise at that valid time, as recorded in the database at that transaction time. This query is illustrated in Figure 10. The result is the fact located at the intersection of the two lines, in this case, Eva. Such queries require two times to be specified.

customer_number
145

List the owner of the apartment at 123 Main Street in Boston on January 13 as stored in the property owner table on January 18.

```
VALIDTIME AS OF DATE '2010-01-13' AND
TRANSACTIONTIME AS OF TIMESTAMP '2010-01-18
23:59:59-05:00'
SELECT customer_number
FROM Prop_Owner
WHERE property_number = 7797;
```

The current bitemporal time-slice uses "now" for both input instants. As that is the default, this is a particularly simple query.

Give the owner of the apartment at 123 Main Street in Boston today as best known.

```
SELECT customer_number
FROM Prop_Owner
WHERE property_number = 7797;
```

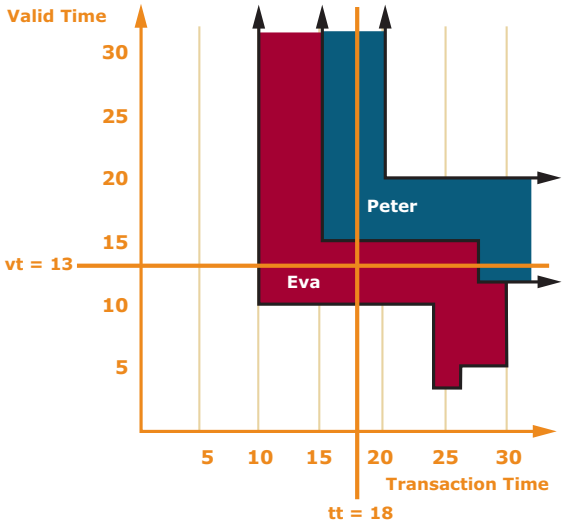

Figure 10. A bitemporal time-slice on a valid time of January 13 and as of a transaction time of January 18.

The Spectrum of Bitemporal Queries

There are three major kinds of queries on valid-time state tables: current ("valid now"), sequenced ("history of") and non-sequenced ("at some time"). There are three analogous kinds of queries on transaction-time state tables: current ("as best known"), sequenced ("when was it recorded") and non-sequenced (e.g., "when was... erroneously changed"). As a bitemporal table includes both valid-time and transaction-time support, and as these two types of time are orthogonal, it turns out that all nine combinations are possible on such tables.

To illustrate, we will take a non-temporal query and provide all the variations of that query. Before doing that, we add one more row to the Prop_Owner table.

Peter Olsen bought another apartment, at 12 Oyster Bay Road in Boston on January 15, 2010; this was recorded on January 31, 2010.

```
SEQUENCED VALIDTIME
INSERT INTO Prop_Owner (customer_number,
property_number, property_VT)
VALUES (827, 3621, PERIOD (DATE '2010-01-15', DATE
'9999-12-31'));
```

Overlaying this information on the time diagram, shown in Figure 11, we see that for 5 days Peter owned two properties, on Oyster Bay Road and on Main Street; he sold the Main Street property on January 20, but retains the Oyster Bay Road property.

We start with a non-temporal query, a simple equijoin, pretending that the Prop_Owner table is a snapshot table.

What properties are owned by the customer who owns property 7797?

```
SELECT P2.property_number
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number;
```

We now enumerate the nine kinds of bitemporal queries that are analogous to this non-temporal query, applying each on the state illustrated in Figure 11 and given in tabular form in Table 2.

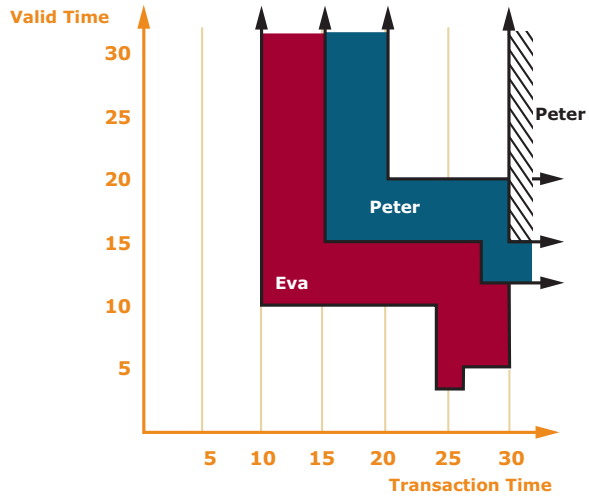

Figure 11. A sequenced insertion, performed on January 31, 2010: Peter bought another apartment on January 15.

customer_number	property_number	property_VT	property_TT
145	7797	(2010-01-10, 9999-12-31)	(2010-01-10, 2010-01-15)
145	7797	(2010-01-10, 2010-01-15)	(2010-01-15, 2010-01-28)
827	7797	(2010-01-15, 9999-12-31)	(2010-01-15, 2010-01-20)
827	7797	(2010-01-15, 2010-01-20)	(2010-01-20, 2010-01-28)
145	7797	(2010-01-03, 2010-01-10)	(2010-01-23, 2010-01-26)
145	7797	(2010-01-05, 2010-01-10)	(2010-01-26, 2010-01-28)
145	7797	(2010-01-05, 2010-01-12)	(2010-01-28, 2010-01-30)
827	7797	(2010-01-12, 2010-01-20)	(2010-01-28, 9999-12-31)
827	3621	(2010-01-15, 9999-12-31)	(2010-01-31, 9999-12-31)

Table 2. The bitemporal state illustrated in Figure 11.

A Case Study of Temporal Data

1. Valid-time current and transaction-time current

What properties are *owned by the customer who owns property 7797,* as best known*?*

```
SELECT P2.property_number
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number;
```

Current in valid time is implemented by requiring that the period of validity overlap "now;" current in transaction time is implemented by requiring a transaction-stop time of "until closed." The result, a snapshot table, is in this case the empty table, because now, as best known, no one owns property 7797. (Peter owned it for some nine days in January, but doesn't own it now.)

2. Valid-time sequenced and transaction-time current

What properties are or were *owned by the customer who owned* at the same time *property 7797, as best known?*

```
SEQUENCED VALIDTIME SELECT P2.property_number
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number;
```

Sequenced in valid time is implemented by selecting the overlap of the periods of validity, when the underlying rows were *both* valid. The result, a valid-time state table, is the following.

property_number	VALIDTIME
3621	(2010-01-15, 2010-01-20)

For those five days in January, Peter owned both properties.

3. Valid-time non-sequenced and transaction-time current

What properties were owned by the customer who owned at any time *property 7797, as best known?*

```
NONSEQUENCED VALIDTIME SELECT P2.property_number
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number;
```

Non-sequenced in valid time is implemented by ignoring the valid timestamps. The result, a snapshot table, is the following.

property_number
3621

Peter owned both properties. While in this case there was a time when Peter owned both properties simultaneously, the query does not require that. Even if Peter had bought the second property on a valid time of January 31, that property would still be returned by this query.

4. Valid-time current and transaction-time sequenced

What properties did we think *are owned by the customer who owns property 7797?*

An initial approach might be to use SEQUENCED TRANSACTIONTIME. However, the SEQUENCED TRANSACTIONTIME qualifier on single tables is not supported in this Teradata release on anything but a single-table SELECT in a view or derived table. But we can rephrase this in this case as a nonsequenced transaction time query with an explicit predicate.

```
NONSEQUENCED TRANSACTIONTIME
SELECT P2.property_number,
   ((P1.property_TT P_INTERSECT P2.property_TT) AS
recorded
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number
   AND P1.property_TT OVERLAPS P2.property_TT;
```

Sequenced in transaction time is implemented identically to sequenced in valid time: by selecting the overlap of the periods of presence, when the underlying rows were *both* present. Note that the result of a transaction-time sequenced query is *not* a transaction-time state table. While the result does indicate what was recorded in the property owner table, it itself was not in existence until the query is performed. We thus use recorded column name to highlight this distinction. The result, a snapshot table with an additional time-stamp column, is the empty table, because there was no time in which we thought that Peter currently owns both properties.

5. Valid-time sequenced and transaction-time sequenced

When did we think that some property, at some time *was owned by the customer who owned* at the same time *property 7797?*

Again, the trick is to convert sequenced transactiontime to nonsequenced.

```
SEQUENCED VALIDTIME AND NONSEQUENCED
TRANSACTIONTIME
SELECT P2.property_number,
   (P1.property_TT P_INTERSECT P2.property_TT) AS
recorded
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number
   AND P1.property_TT OVERLAPS P2.property_TT;
```

Here we have sequenced in both valid time and transaction time. This is the most involved of all the queries, but the parallel between valid time and transaction time should be apparent in the above query. We must compute the overlap of the underlying rectangles, with the result being a valid-time state table with additional recorded timestamp column. Figure 12 shows the two rectangles that are involved and the overlap that is computed. One row results.

property_number	recorded	VALIDTIME
3621	(2010-01-31, 9999-12-31)	(2010-01-15, 2010-01-20)

For those five days in January, Peter owned both properties. That information was recorded on January 31, and is still thought to be true (a transaction-stop time of "until closed").

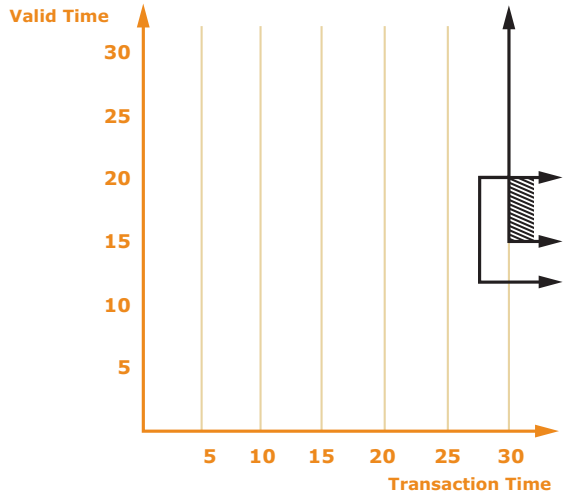

Figure 12. A query sequenced in both valid time and transaction time, computing the intersection of two rectangles.

6. Valid-time non-sequenced and transaction-time sequenced

When did we think that some property, at some time was owned by the customer who owned at any time *property 7797?*

As with the two previous queries, the trick is to convert sequenced transactiontime to nonsequenced.

```
NONSEQUENCED VALIDTIME AND NONSEQUENCED TRANSACTIONTIME
SELECT P2.property_number,
   (P1.property_TT P_INTERSECT P2.property_TT) AS recorded
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number
   AND P1.property_TT OVERLAPS P2.property_TT;
```

As before, non-sequenced in valid-time is implemented by ignoring the valid timestamps. The result, a snapshot table with additional timestamp columns, is the following.

property_number	recorded
3621	(2010-01-31, 9999-12-31)

From January 31 on, we thought that Peter had owned those two properties, perhaps not simultaneously.

7. Valid-time current and transaction-time non-sequenced

When was it recorded that a property is owned by the customer who owns property 7797?

```
NONSEQUENCED TRANSACTIONTIME
SELECT P2.property_number, BEGIN(P2.property_TT) AS recorded_Start
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number
   AND P1.property_TT OVERLAPS P2.property_TT;
```

Non-sequenced in transaction time is implemented by not testing for full overlap in transaction time (sequenced) and by not testing the transaction-stop time for "until closed" (current). The result, a snapshot table, is empty, because we never thought that Peter currently owns two properties.

8. Valid-time sequenced and transaction-time non-sequenced

When was it recorded that a property is or was *owned by the customer who owned at the same time property 7797?*

```
SEQUENCED VALIDTIME AND NONSEQUENCED TRANSACTIONTIME
SELECT P2.property_number, BEGIN(P2.property_TT) AS recorded_Start
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number
   AND P1.property_TT OVERLAPS P2.property_TT;
```

This query is similar to valid-time sequenced/transaction-time current, with a different predicate for transaction time. The result, a valid-time state table with an additional timestamp column, is the following.

property_number	recorded_Start	VALIDTIME
3621	(2010-01-31)	(2010-01-15, 2010-01-20)

For those five days in January, Peter owned both properties; this information was recorded on January 31.

9. Valid-time non-sequenced and transaction-time non-sequenced

When was it recorded that a property was owned by the customer who owned at some time *property 7797?*

```
NONSEQUENCED VALIDTIME AND NONSEQUENCED TRANSACTIONTIME
SELECT P2.property_number, BEGIN(P2.property_TT) AS recorded_Start
FROM Prop_Owner AS P1, Prop_Owner AS P2
WHERE P1.property_number = 7797
   AND P2.property_number <> P1.property_number
   AND P1.customer_number = P2.customer_number
   AND P1.property_TT OVERLAPS P2.property_TT;
```

The result, a snapshot table with an additional timestamp column, is the following.

property_number	recorded_Start
3621	2010-01-31

Current in valid time translates in English to "at now;" sequenced translates to "at the same time;" and non-sequenced translates to "at any time." Current in transaction time translates to "as best known;" sequenced translates to "when did we think;" and non-sequenced translates to "when was it recorded" or "when was it corrected."

Of these nine types of queries, a few are more prevalent. The most common is the current/current query: "now, as best known." These queries correspond to queries on the non-temporal version of the table. (The following queries also utilize the Customer and Property tables; we assume that these two tables are also bitemporal.)

A Case Study of Temporal Data

What is the estimated value of the property at 12 Oyster Bay Road? (current/current)

```
SELECT estimated_value
FROM Property AS P
WHERE P.address = '12 Oyster Bay Road';
```

Current/current queries return a snapshot result.

Who owns the property at 12 Oyster Bay Road? (current/current)

```
SELECT name
FROM Prop_Owner AS PO, Customer AS C, Property AS P
WHERE P.address = '12 Oyster Bay Road'
   AND P.property_number = PO.property_number
   AND C.customer_number = PO.customer_number;
```

Perhaps the next most common kind of query is a sequenced/current query, "history, as best known." These queries ignore transaction time, and return a valid-time state table. Sequenced/current queries over one table are simple to specify.

How has the estimated value of the property at 12 Oyster Bay Road varied over time? (sequenced/current)

```
SEQUENCED VALIDTIME SELECT estimated_value
FROM Property AS P
WHERE P.address = '12 Oyster Bay Road';
```

Who has owned the property at 12 Oyster Bay Road? (sequenced/current)

```
SEQUENCED VALIDTIME SELECT name
FROM Prop_Owner AS PO, Customer AS C, Property AS P
WHERE P.address = '12 Oyster Bay Road'
   AND P.property_number = PO.property_number
   AND C.customer_number = PO.customer_number;
```

Transaction time is supported in the Prop_Owner table to track the changes, and to correct errors. A common query searches for the transaction that stored the current information in valid time. This is a current/non-sequenced query.

When was the estimated value for the property at 12 Oyster Bay Road stored? (current/nonsequenced)

```
NONSEQUENCED TRANSACTIONTIME
SELECT estimated_value, BEGIN(property_TT) AS
recorded_Start
FROM Property
WHERE address = '12 Oyster Bay Road';
```

This query will return a snapshot table giving one or more estimated values, along with the date of the transaction recording that value.

Sequenced/non-sequenced queries allow one to determine when invalid information about the history was recorded.

Who has owned the property at 12 Oyster Bay Road, and when was this information recorded? (sequenced/nonsequenced)

```
SEQUENCED VALDTIME AND NONSEQUENCED TRANSAC-
TIONTIME
SELECT name, BEGIN(PO.property_TT) AS PO_recorded,
   BEGIN(C.property_TT) AS C_recorded,
   BEGIN(P.property_TT) AS P_recorded_Start
FROM Prop_Owner AS PO, Customer AS C, Property AS P
WHERE P.address = '12 Oyster Bay Road'
   AND P.property_number = PO.property_number
   AND C.customer_number = PO.customer_number
   AND PO.property_TT OVERLAPS C.property_TT
   AND C.property_TT OVERLAPS P.property_TT
   AND PO.property_TT OVERLAPS P.property_TT;
```

This returns a valid-time state table, with three additional columns stating when that information was recorded in the underlying tables. Subsequent queries could then isolate the identified problem.

Finally, non-sequenced/non-sequenced queries can probe the interaction between valid time and transaction time.

A Case Study of Temporal Data

List all retroactive changes made to the Prop Owner table. (non- sequenced/nonsequenced)

```
NONSEQUENCED VALIDTIME AND NONSEQUENCED
TRANSACTIONTIME
SELECT customer_number, property_number,
property_VT,
   BEGIN(property_TT) AS recorded_Start
FROM Prop_Owner
WHERE BEGIN(property_VT) < BEGIN(property_TT);
```

This returns the following valid-time state table, indicating that many of the modifications were retroactive.

customer_number	property_number	recorded_Start	property_VT
145	7797	2010-01-15	(2010-01-10, 2010-01-15)
827	7797	2010-01-20	(2010-01-15, 2010-01-20)
145	7797	2010-01-23	(2010-01-03, 2010-01-10)
145	7797	2010-01-26	(2010-01-05, 2010-01-10)
145	7797	2010-01-28	(2010-01-05, 2010-10-12)
827	7797	2010-01-28	(2010-01-12, 2010-01-20)
827	3621	2010-01-31	(2010-01-15, 9999-12-31)

Summary

A bitemporal table combines both valid time and transaction time into a single structure. It contains two periods, the valid-time period of validity and the transaction-time period of presence.

We examined several variants of modifications on such tables, all current in transaction time: validtime current inserts, deletions, and updates; valid-time sequenced insertions, deletions, and updates; and valid-time nonsequenced deletions.

We then considered time-slice queries, in transaction, valid, and bitemporal varieties. We showed that there are nine versions of any non-temporal query (all combinations of current/sequenced/nonsequenced in valid time and transaction time). A benefit of a bitemporal table is that it admits the full generality of temporal queries.

Acknowledgement

I thank Ramesh Bhashyam for checking all these modifications and queries on Teradata Database 13.10, for general help in understanding Teradata's temporal SQL constructs, and for many in-depth discussions on the semantics of temporal data in the context of SQL.

References

[Snodgrass & Ahn 1986] Richard T. Snodgrass and Ilsoo Ahn, "Temporal Databases," *IEEE Computer* 19(9):35–42, September, 1986.

[Snodgrass et al. 1996a] Richard T. Snodgrass, Michael H. Böhlen, Christian S. Jensen and Andreas Steiner, "Adding Valid Time to SQL/Temporal," change proposal, ANSI X3H2-96-501r2, ISO/IEC JTC1/SC21/ WG3 DBL MAD-146r2, November 1996, 77 pages. (See http: //www.cs.arizona.edu/people/rts/sql3.html for more information.)

[Snodgrass et al. 1996b] Richard T. Snodgrass, Michael H. Böhlen, Christian S. Jensen and Andreas Steiner, "Adding Transaction Time to SQL/Temporal," change proposal, ANSI X3H2- 96-502r2, ISO/IEC JTC1/SC21/ WG3 DBL MAD-147r2, November 1996, 47 pages.

[Snodgrass 1999] Richard T. Snodgrass, **Developing Time-Oriented Database Applications in SQL**, Morgan Kaufmann Publishers, Inc., San Francisco, CA, July 1999, 504+xxiv pages.

About the Author

Richard T. Snodgrass joined the University of Arizona in 1989, where he is a Professor of Computer Science. He holds a B.A. degree in Physics from Carleton College and M.S. and Ph.D. degrees in Computer Science from Carnegie Mellon University. He is an ACM Fellow.

Richard's research interests are the science of computer science, compliant databases, and temporal databases.

Richard was Editor-in-Chief of the ACM *Transactions on Database Systems* from 2001 to 2007, was ACM SIGMOD Chair from 1997 to 2001, and has chaired the ACM Publications Board, the ACM History Committee, and the ACM SIG Governing Board Portal Committee. He co-directs TimeCenter, an international center for the support of temporal database applications on traditional and emerging DBMS technologies.