

Time Relational Data Model Patent Summary

Business Perspective

Many business applications need to store data that is date effective. Changes might be future transactions or late transactions or they might be corrections to previous transactions. The Time-chain Solution, a method of the Time Relational Data Model, provides a data structure and a set of methodologies that ensure that such transactions are consistent and support analytical uses. The structure allows processes to run reliably at the same time as transactions are being applied. Such a facility is essential in the modern 24/7 web world.

Background

In a database that supports transaction processing where the data is time-based, a variety of designs over the last thirty years have attempted to meet the processing and audit requirements of ever more sophisticated processing. One of the problems that has caused an evolution of designs is the difference between event time and database time.

Event time is the time an event in the real world has or is expected to happen. A person is hired on a specific date, but the records for the hire are not entered for a few days after the person shows up for work. Or a salary increase should take effect in two weeks, but various people review and approve the change prior to its effective date.

Events can be thought of as describing a period. A person's salary increase takes place on a specific date and the salary is the same until the next salary change occurs. When the next change takes effect, the salary has a new value. We can think of the first event as having a period from the effective date to just before the next change. Each history of a person's salary has such a period. The last change initially has a period for the effective date of the action until the indefinite future. When a new change comes along, the end date of the period changes to just before the subsequent change.

Since this is the nature of the data, modern HR system store historical data as a sequence of periods:

Example 1.

Type of Action	Salary	Effective Date	End Date
Merit Increase	\$90,000/yr	06/01/2005	11/30/2005
Market Adjustment	\$95,000/yr	12/01/2005	05/31/2006
Merit Increase	\$105,000/yr	06/01/2006	future

This structure is excellent for processes such as payroll where knowing the salary at a particular time is important. It also ensures that there is no ambiguity about the period. If a component of compensation was not continuous, the same structure supports that:

Example 2.

Type of Action	Salary	Effective Date	End Date
Performance Bonus	\$1,000/mo	06/01/2005	08/31/2005
Company Bonus	\$2,000/mo	12/01/2005	12/31/2005
Performance Bonus	\$1,500/mo	06/01/2006	08/31/2006

Here the period of each event is clear from the two dates. Many functions of an HR system operate in ways similar to this compensation example: benefit enrollments, deductions and organization structure to name a few. Even addresses, phone numbers and emails have time periods.

This structure has a processing advantage over earlier designs. It is possible to use a simple database query to obtain all the records in effect on a particular day:

Query 1: Select * from Compensation where '07/15/2006' between EffectiveDate and EndDate.

This query will retrieve all the compensation records for all people that were in effect on 07/15. This is a handy facility since often organizations want reports that are consistent for a date.

These structures would be enough for an information system if not for the time delays for putting in data and the possibility that data is entered incorrectly.

The behavior of the payroll process must be different if this person's Merit Increase is entered into the system on 06/01/2006, when it is effective, than if it is entered on 07/01/2006. If it is entered in a timely fashion, the correct value will be available when the payroll of 06/15/2006 is computed. However if the data is not entered until 07/01/2006, two pay periods have been missed.

A solution to this problem is to add a transaction time to the rows of the database either by storing the time-stamp with the row or in a separate table that is referred to by the table.

Example 3.

Type of Action	Salary	Effective Date	End Date	Create Time
Merit Increase	\$90,000/yr	06/01/2005	11/30/2005	06/16/2005 12:45
Market Adjustment	\$95,000/yr	12/01/2005	05/31/2006	11/26/2005 8:15
Merit Increase	\$105,000/yr	06/01/2006	future	06/01/2006 13:26

This structure lets the payroll compute correctly since it can tell when something is early or late. But dealing with the problem of data that was incorrectly entered, and is either corrected by a subsequent transaction or is removed, requires that more information be stored. Another time-stamp is needed.

Example 4.

Type of Action	Salary	Effective Date	End Date	Create Time	End Time
Merit Increase	\$90,000/yr	06/01/2005	11/30/2005	06/16/2005 12:45	06/18/2005 2:53
Merit Increase	\$91,000/yr	06/01/2005	11/30/2005	06/18/2005 2:53	future
Market Adjustment	\$95,000/yr	12/01/2005	05/31/2006	11/26/2005 8:15	future
Merit Increase	\$105,000/yr	06/01/2006	future	06/01/2006 13:26	future

In this example, the 06/01/2005 record was incorrectly entered on 06/18 and then corrected on 6/18. Now the payroll can compute adjustments to take into account the incorrect data. When payroll was run on 6/17 for the 06/01 to 06/15 period an incorrect amount was used. An adjustment for that period can be calculated.

The End Date Problem

In this data model, Effective Dates are never changed in a record, nor is the data that applies to the period such as Salary. Instead, any revision to the period data creates a new record. Adding a new event after an event doesn't change the period data, it creates a new record. But it must update the End Date of the prior record.

If the End Date didn't change, we would be able to create a query that showed the prior state of the database.

Query 2: Select * from Compensation where '07/15/2006' between CreateTs and EndTs.

This query would retrieve records two through four in the example. The query:

Query 3: Select * from Compensation where '06/17/2006' between CreateTs and EndTs

would retrieve only the first record in the example.

And the query

Query 4: Select * from Compensation where '06/17/2006' between CreateTs and EndTs
and '06/17/2006' between EffectiveDate and EndDate

would correctly retrieve the first record in the example.

However, the query

Query 5: Select * from Compensation where '06/17/2006' between CreateTs and EndTs
and '12/01/2006' between EffectiveDate and EndDate

would not retrieve any records. This is because the first record looked like this when entered:

Example 5.

Type of Action	Salary	Effective Date	End Date	Create Time	End Time
Merit Increase	\$90,000/yr	06/01/2005	future	06/16/2005 12:45	future

after the second record is entered, the database looks like this:

Example 6.

Type of Action	Salary	Effective Date	End Date	Create Time	End Time
Merit Increase	\$90,000/yr	06/01/2005	11/30/2005	06/16/2005 12:45	future
Market Adjustment	\$95,000/yr	12/01/2005	future	11/26/2005 8:15	future

The entry of the second record updates the End Date of the first record and so its period changes.

Query 5 produces inconsistent results and therefore is unusable.

If we need to be able to recreate the database at any prior time, we can change the process and replicate any record where the End Date is updated:

Example 7.

Type of Action	Salary	Effective Date	End Date	Create Time	End Time
Merit Increase	\$90,000/yr	06/01/2005	future	06/16/2005 12:45	11/26/2005 8:15
Merit Increase	\$90,000/yr	06/01/2005	11/30/2005	06/16/2005 12:45	future
Market Adjustment	\$95,000/yr	12/01/2005	future	11/26/2005 8:15	future

This structure looks almost the same as the structure of Example 4 that allowed us to consider corrections to the data. The problem with this solution is that it doubles the size of the database. Every record except the last in a history will be duplicated. This is an expensive solution to the regression query problem, but Query 5 will produce reliable results.

If we know that a history is continuous, we can recreate the database as of a prior time without duplicating records because the Effective Dates and End Dates create a chain of events.

If we take the records of Example 6 and use the fact that the End Date of record one is one day less than the Effective Date of record two and the Create Time is in advance of record one, then the End Date of record one before record two came along must have been the End Date of record two:

Example 6 - Part 2.

Type of Action	Salary	Effective Date	End Date	Create Time	End Time
Merit Increase	\$90,000/yr	06/01/2005	11/30/2005	06/16/2005 12:45	future
Market Adjustment	\$95,000/yr	12/01/2005	future	11/26/2005 8:15	future

transposing the dates of records one and two yields:

Merit Increase	\$90,000/yr	06/01/2005	future	06/16/2005 12:45	future
----------------	-------------	------------	--------	------------------	--------

which is the state of record one before record two was applied (exactly like Example 5). In the case of Example 3 where another transaction has occurred, we can apply the same process one more time to revert the record to its original state. The signal that the transposition process must be done is the fact that the create time-stamps of the records are subsequent to the regression date.

The transposing mechanism can be accomplished by a SQL statement and can operate on data than includes

corrections. A similar process can be used when the database can contain corrections. With this database structure and this simple transposing mechanism, regression of a database to its prior state can be accomplished without doubling the size of the database.

A corollary to this mechanism is that the transposing mechanism need only be used if the regression date is prior to the current time of the database. Consequently, the queries in the form:

```
Select * from Compensation where current_timestamp between CreateTs and EndTs  
and '12/01/2006' between EffectiveDate and EndDate
```

will always produce consistent results.

What is patented:

The process of creating the two pairs of dates and the rules for establishing values.

About SynchSource

SynchSource Inc. has developed a suite of open source HR, workforce management, benefits, and payroll solutions that are rapidly establishing the company as a leader in the service bureau software marketplace. SynchSource's fully integrated platform provides Human Resource Outsourcers (HROs), Professional Employment Organizations (PEOs), Payroll Outsourcers, Benefits Administration Outsourcers, and other service providers with a SaaS delivery model designed to empower their clients, accommodate unique requirements without customizations, reduce overall administrative support costs, and provide more value-added services at improved margins. The patented time-relational data model and the patented workflow and security engine are at the core of the SynchSource platform.

SynchSource Inc.
2201 Broadway, Suite #701
Oakland, CA 94612

Phone: +1 866-989-1145
email: info@synchsource.com
www.synchsource.com

