



Kimball Design Tip #35: Modeling The Spans

By Ralph Kimball

In the past couple of years I have seen an increased demand for applications that need to ask questions about time spans. One person captured it nicely when he said "each record in my fact table is an episode of constant value over a region of time". Time spans can start and stop at arbitrary points in time. In some cases time spans link together to form an unbroken chain, in other cases the time spans are isolated, and in the worst cases the time spans overlap arbitrarily. But each time span is represented in the database by a single record. To make these variations easier to visualize, let's imagine that we have a database filled with atomic transactions, such as deposits and withdrawals from bank accounts. We'll also include open account and close account transactions. Each transaction implicitly defines an episode of constant value over a region in time. A deposit or a withdrawal defines a new value for the account balance that is valid until the next transaction. This time span could be one second or it could be many months. The open account transaction defines the status of the account as continuously active over a time span until a close account transaction appears.

Before we propose a database design, let's remind ourselves of some of the time span questions we want to ask. We'll start off by limiting our questions to a granularity of individual days, rather than parts of days like minutes and seconds. We'll return to minutes and seconds at the end. The easy questions we have always been good at answering include:

- Show all the transactions that occurred within a given time span.
- Determine if a selected transaction occurred within a given time span.
- Define time spans using complex calendar navigation capabilities including seasons, fiscal periods, day numbers, week numbers, pay days, and holidays.

For these cases all we need is a single time stamp on the transaction fact table record. The first question picks up all the transactions whose time stamp is in an interval specified in the user's query. The second question retrieves the time stamp from a selected transaction and compares it to the interval. The third set of questions replaces the simple time stamp with a calendar date dimension filled with lots of helpful calendar attributes. This date dimension is connected to the fact table through a standard foreign key - primary key join. This is all vanilla dimensional design and only requires a single time key in the fact table record to represent the required time stamp. So far, so good.

By the way, when using complex calendar navigation the queries become much easier if the verbose date dimension includes first day and last day markers for each defined span of time, such as "last day of quarter". This field would have the value "N" for all the days except the last day of the applicable quarter. The last day would have the value "Y" in the special field. These markers allow the complex business time spans to be easily specified in the queries. Note that the use of a verbose date dimension means the application is not navigating a time stamp on the fact table. More about this in the last section.

A second moderately hard category of time span questions include

- Show everyone who was a customer at some point within a time span.

- Show the last transaction for a given customer within a time span.
- Show the balance of an account at an arbitrarily selected point in time.

We'll continue to make the simplifying assumption that all the time spans are described by calendar days, not by minutes and seconds. It is possible to answer all these questions with the single time stamp design given above but this approach requires complex and inefficient queries. For instance, to answer the last question, we would need to search the set of account transactions for the latest transaction at or prior to the desired point in time. In SQL, this would take the form of a correlated sub-SELECT embedded in the surrounding query. Not only is this probably slow, but the SQL is not readily produced by end user tools.

For all of these moderately hard time span questions, we simplify the applications enormously by providing twin time stamps on each fact record, indicating the beginning and end of the time span implicitly defined by the transaction.

With the twin time stamp design, we knock off the above three example questions easily:

1. Search for all the open account transactions whose begin date occurs on or prior to the end of the time span, and whose end date occurs on or after the beginning of the time span. 2. Search for the single transaction whose begin date is on or before the end of the time span and whose end date is on or after the end of the time span. 3. Search for the single transaction whose begin date is on or before the arbitrary point in time and whose end date is on or after the arbitrary point in time.

In all these cases the SQL uses a simple BETWEEN construct. The "value between two fields" style of this SQL is indeed allowable syntax. I learned this recently.

When we use the twin time stamp approach, we have to be honest about one major drawback. In almost all situations, we have to visit each fact table record twice. Once when we first insert it (with an open ended end timestamp), and once more when a superceding transaction occurs that actually defines the real end timestamp. The open ended end time stamp probably should be a real value somewhere out in the future, so that applications don't trip over null values when they try to execute the BETWEEN clause.

We have saved the hardest questions to the end: time spans to the second. In this case we ask the same basic questions as in the first two sections, but we allow the time span boundaries to be defined to the nearest second. In this case, we'll put the same two twin time stamps in the fact record, but we have to give up our connection to a robust time dimension. Both our beginning and ending time stamps must be conventional RDBMS date/time stamps. We must do this because we usually cannot create a single time dimension with all the minutes or all the seconds over a significant period of time. Dividing the time stamp into a day component and a seconds-in-a-day component would make the BETWEEN logic horrendous. So for these ultra-precise time spans we live with the limitations of SQL date/time semantics and give up the ability to specify seasons or fiscal periods to the nearest second.

If you are really a diehard, you could consider four time stamps on each transaction record if your time spans are accurate to the second. The first two would be RDBMS date/time stamps as described in the preceding paragraph. But the third and fourth would be calendar day (only) foreign keys connecting to a verbose calendar day dimension as in the first two sections of this article. That way you can have your cake and eat it too. You can search for ultra-precise time spans, but you can also ask questions like "Show me all the power outages that occurred on a holiday".

Even with these powerful techniques, I am sure there are some tricky time span questions I haven't considered. I collect these tricky puzzles. E-mail new ones to me at ralph@kimballgroup.com.