Taming the Trickiness of Temporality

written by DaltonRuer | August 31, 2021

In my previous post Merging with Time Travel I demonstrated how you can combine Qlik's Merge functionality to take advantage of Snowflake Time Travel. After all, everyone else in the world is talking about Change Data Capture and Data Pipelines so I figured I better get on that bandwagon.

But why should we settle for historical values, when we can travel into the future and see the values that haven't occurred yet. I know what you are thinking "databases only store current values, or previous values." But that's where you are wrong my friend. Before I explain how you can handle the data in Qlik, I will explain what temporality is, and why it is important.

The Need for Temporality

Imagine you run a credit card processing business. Hurray for you. That's big time $ if you succeed. In order to do so you will have to compete for customers with some other pretty saavy businesses. Take a little friendly advice, don't lead with an advertisement like this:

Use our Credit Card we have 27% interest rates

Nobody will sign up. You need to come up with something a little sneakier than that. Oooh I have an idea. You will offer 0% interest for 3 months in order to get the customers to put the card in their wallets and start purchasing with it. Woohoo! A guy named Qlik Dork just signed up as your first customer. You set his account interest rate to be 0%, and you set an alarm on your smart phone to change it to 27% in 3 months. When the alarm goes off you change his interest rate from 0% to 27%. That was easy now the fees will come rolling in. But 1 customer isn't enough.

Soon Qlik Dork's friends see him using his card and ask him about it. Soon you have 10 customers swiping/inserting/tapping that beautiful plastic. It's getting harder for you to maintain all of those alarms in your smart phone, so you upgrade it to a model with more memory. Soon you have 100 customers. 500 customers. 1,000 customers. Oh boy, that's an awful lot of alarms.

Never fear, temporal tables are hear. Instead of storing only the current interest rate and having to remember to change it at the right point in the future, you can enter the future dates when your teaser rates are supposed to change. Voila you are storing future values that haven't occurred yet.

It's all Temporal

In case the conversation ever comes up at a friendly gathering, that kind of data is called uni-temporal. You are recording the time in the future when the value will become valid. Since we are having so much fun let's continue by briefly touching on bi-temporal and tri-temporal data tables

as well. Not only would you store the time when the value becomes valid, you also store the transaction time when it is recorded and the time when a decision was made about the value.

Perhaps running your own credit card processing business was far fetched. Temporality is also used for grocery stores and we all have to buy groceries. I mean, you don’t think the poor store manager hand inputs hundreds of price changes in a single minute when sales go into effect do you? That would be crazy. Future prices are recorded far in advance, and the point of sale applications simply apply the changes at the appropriate times.

Still can’t relate? Ok think about Prime Days. Where the price of the item is tiny for the next 60 seconds, and then afterwards immediately returns to a giant price. On thousands of items. Globally. In all time zones.

TMI

Normally I don’t elaborate so in depth about database modeling, but in this case the use case I will show came from one of my colleagues who is a phenomenal DBA. He can make temporal data sing in a data warehouse cause that’s his jam. But he’s been learning more about Qlik’s data modeling and reached out to find how in the world Qlik could handle this type of scenario. Where there isn’t a single value to associate between tables. So try and embrace the craziness for a few minutes as I think you will be glad you did.

Temporal Values

For the remainder of the article let’s go ahead and imagine the Prime Days example. Here is what a temporal table for the products might look like.

Glad we checked now, because it looks like the hottest new thing on the market, Product ID 1 is going on sale for 1 day only on September 7'th. Savings of $100. We better get it then, because the next day it jumps to $120.99. Since we've been considering purchasing Product ID 2 for a while, we better jump on it since the price will increase a few dollars on 12/1/2021.

Products

Product ID	Price	StartDate	EndDate
1	110.58	10/1/1992	9/6/2021
1	10.58	9/7/2021	9/7/2021
1	120.99	9/8/2021	12/31/9999
2	117.12	10/1/1992	11/30/2021
2	120.50	12/1/2021	12/31/9999

Notice that the last record for each product, or whatever the record might be, needs to represent inphinity. I simply used 9999 as the inphinity placeholder. It could be any date in the very distant future to indicate that the "value" will remain. If a new value were inserted for Product ID 2 that was going to start on March 3, 2022, then the end date would be changed for 120.50 to March 2, 2022 and the new record would show what the value will change to and it will go to "inphinity."

If we were serious about this temporality stuff, we might look into that bi-temporal stuff and add a Transaction Date field that said "It was September 1, 2021 when we changed our end date for the 120.50 value, and inserted the new record that starts March 3, 2022. Or if we were really really really serious we might add a third date, tri-temporal, that says this value has already had something done with it on this date so don't you dare change the value anymore. You must create a new record starting after that date.

If you are a "data warehouse" type of person, you might think this is very close to a slowly changing dimension. And you would be correct it is close. But isn't the same. The fact

that the dates are in the future and specify a **RANGE** of dates, means it isn't recording what "has changed" like a data warehouse would do. We aren't just recording the historical value changes of the price as a slowly changing dimension. We are literally recording what will happen in the future in that table. Of course we also need to track purchases so let's imagine that Customer ID purchased both products on 8/17/2021.

Transactions

Product ID	OrderDate	Customer ID
1	8/17/2021	1
2	8/17/2021	1

What is the price they paid? If you said $110.58 + $117.12, then you know your data.

Interval

Notice above I bolded and uppercased the word RANGE so it would really stand out. As humans we are really good at determining if a given value is within the interval that the range represents. We are trained early on that in school. A's are 90-100 and your score is a 93. Woohoo you run home and tell your mommy that you got an A. There is no difference here. You looked at the OrderDate and compared it to the ranges to know what price would be effective. Don't worry about the fact that they would have brought the current price into a fact table to record actual transactions. We are going to try and reproduce the FACTS, and understand how to handle any other data that is similar.

If we simply load those 2 tables into Qlik Sense and trusted the Associative Model we would end up with a mess:

[Customer ID]	Product ID	OrderDate	Price
1	1	8/17/2021	10.58
1	1	8/17/2021	110.58
1	2	8/17/2021	117.12
1	2	8/17/2021	120.50
1	1	8/17/2021	120.99

Wouldn't it be nice if Qlik provided a handy dandy mechanism to handle this type of relationship where you have a value like an OrderDate and you want to compare it to an **interval** of values for a Start-End range? Then update the associative engine with the **match**es. Turns out they do. It's got a pretty obvious name as well IntervalMatch, and it's lights out easy to use.

```
41  Inner Join IntervalMatch(OrderDate, [Product ID])
42  Load StartDate, EndDate, [Product ID]
43  Resident Products
```

Voila it did the same mental gymnastics you did. But since there will be millions of such intervals to match, I think it's best we let it do it's thing and you can take a lunch break.

[Customer ID]	Product ID	OrderDate	Price
1	1	8/17/2021	110.58
1	2	8/17/2021	117.12

More Use Cases

To be honest the whole temporality thing is a one off use case that I did recently for a DBA guru friend of mine who had this type of data and he was curious how Qlik Sense modeling could handle the equivalent of a SQL "Between" clause for the join. I thought it would be a fun way to tie in time travel one more time, while also introducing the IntervalMatch function which is incredibly powerful.

Before you go away thinking it was a waste of time because you will never run a credit card processing system, or a grocery store or run production for Amazon Prime Days. Give me just a few more moments of your time so I can talk about other use cases that you may need to handle.

Staffing – How many employees were working between 7:30 and 7:45 AM? Boom! Perfect fit for IntervalMatch.

Accounting – A summary account in a balance sheet is for accounts 3100 – 3216. Boom! Perfect fit for IntervalMatch.

Now those hit close to home don't they? These are the things you make your living on, aren't they?

Background (Just for fun)

As I was getting started in my Qlik journey as a customer there were many times I ran into trouble. Most of the time I would ask a consultant who worked with my health system, George Kovakas, what we could do to solve the problem. The vast majority of the time his answer was "I think IntervalMatch could handle that." Sure enough a few minutes later we had the solution banged out. It kind of became a running joke "I have this problem that I know you will think can be solved by IntervalMatch can handle, but I don't know how." Well by golly if he couldn't spin it sideways and solve the problem. The staffing use case above. Yeah that was real. We built a Master Calendar with a record for every single minute of every day, ran interval match and end users could click on 15, 30, 60 minute increments to know exactly how many

unique staff had worked. Same with the accounting use case. We simply had to create the map of what the balance sheet titles were and what range of accounts they represented. Voila we had financial reporting under control.

Two very common, but vastly different use cases, solved by this very powerful Qlik scripting function. Not only was the function pivotal in solving the challenges at hand then, it set the stage for me understanding how to take advantage of what Qlik really offered. To me the power was in the incremental loads and the fact that I only wanted to load minimal data as it changed, rather than having to write super complex SQL statements that I could write, but that required a full reload every time. So I've never stopped sharing my appreciation for George's insights.

Get out there and have some fun with this amazing function.