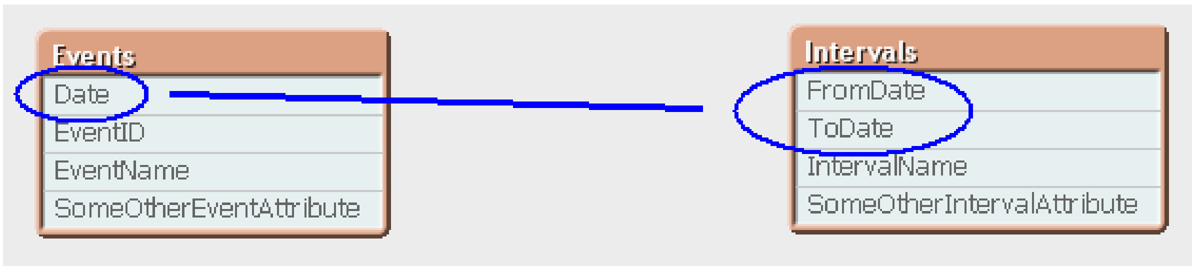
QlikView Design Blog : IntervalMatch

Posted by Henric Cronström Apr 4, 2013

A common problem in business intelligence is when you want to link a number to a range. It could be that you have a date in one table and an interval – a "From" date and a "To" date – in another table, and you want to link the two tables. In SQL, you would probably join them using a BETWEEN clause in the comparison.



But how do you solve this in QlikView, where you should avoid joins?

The answer is to use IntervalMatch.

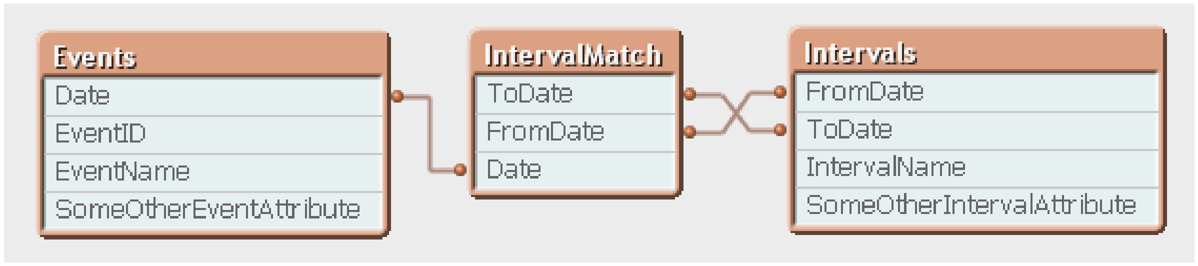
IntervalMatch is a prefix that can be put in front of either a Load or a SELECT statement. The Load/SELECT statement needs to contain two fields only: the "From" and the "To" fields defining the intervals. The IntervalMatch will generate all the combinations between the loaded intervals and a previously loaded numeric field.

Typically, you would first load the table with the individual numbers (The Events), then the table with the Intervals, and finally an intervalmatch that creates a third table that bridges the two first tables.

```
Events:
Load * From Events;

Intervals:
Load * From Intervals;

IntervalMatch:
IntervalMatch (Date)
Load distinct FromDate, ToDate resident Intervals;
```



The resulting data model contains three tables:

1. The Events table that contains exactly one record per event.
2. The Intervals table that contains exactly one record per interval.
3. The IntervalMatch table that contains exactly one record per combination of event and interval, and that links the two previous tables.

Note that this means that an event may belong to several intervals, if the intervals are overlapping. And an interval can of course have several events belonging to it.

This data model is optimal, in the sense that it is normalized and compact. All QlikView calculations operating on these tables e.g. *Count(EventID)* will work and will be evaluated correctly. This means that it is ***not*** necessary to join the intervalmatch table onto one of the original tables. Joining it onto another table may even cause QlikView to calculate aggregations incorrectly, since the join can change the number of records in a table.

Further, the data model contains a composite key (the FromDate and ToDate fields) which will manifest itself as a QlikView synthetic key. But have no fear. This synthetic key ***should*** be there; not only is it correct, but it is also optimal given the data model. You do ***not*** need to remove it.

IntervalMatch can also be used with an additional key between the tables – i.e. when you have Slowly Changing Dimensions. But more about that in a later post.

HIC

For more on IntervalMatch and some script examples, see the technical brief IntervalMatch and Slowly Changing Dimensions.

2991 Views Tags: intervalmatch, slowly_changing_dimension, intervals, prefix

Apr 4, 2013 12:16 PM amirvas

Can also add additional dimension in cases where Employees change departments and you want to find the original person responsible for creating the record and/or sales etc. so that credit applies accordingly

I tend to LEFT JOIN the intervaltable into the fact table which removes the valid synthetic key just to keep things simple

Apr 4, 2013 5:51 PM Henric Cronström amirvas in response to

Your case is a a Slowly Changing Dimension. A record in the fact table has an EmployeeID and a Date. To find the department to which the employee belongs, you need both keys. An employee can (probably) only belong to one department at a time, and in such a case a Left Join can be used: QlikView will still calculate the aggregations correctly since the number of records in the fact table won't change.

However, a left join will add columns to the fact table - which will increase the memory usage. A solution without a left join could probably use less memory.

HIC

Apr 4, 2013 5:55 PM amirvas Henric Cronström in response to

Very true.

Apr 4, 2013 10:22 PM Trey Bayne

Saying that a synthetic key is ok here is just like saying that allowing QV to auto-concatenate is ok. Most of the time, QV is doing exactly what it should. The problem is that QV obscures what it does in synthetic keys. Doing a left join to the fact table guarantees what you are getting.

Apr 5, 2013 3:33 AM Henric Cronström Trey Bayne in response to

I think it is quite the opposite. The Synthetic keys highlight that there is a composite key and that this needs special treatment: Only existing combinations should be used; NULL values should be handled; etc.

QlikView does the only possible from an algorithmic point:

1. Lists relevant combinations (the $Syn table)
2. Uses only the relevant combinations to link the two original tables (the $Syn key)

This is clearly visible when you look at the internal table view. You can even do a preview on the $Syn table. So, as I see it, nothing is obscured.

A left join seems like sweeping the dust under the rug. If you want to avoid synthetic keys properly, you should create your own composite keys instead.

Concerning the auto-concatenate: I agree that we probably shouldn't have done it this way when we first implemented it in 1994. But now it is there, and to remove it would cause greater problems.

HIC

Apr 5, 2013 2:29 PM Barry Harmsen Henric Cronström in response to

Hi Henric,

I understand that you will want to keep the intermediate table in case of overlapping intervals, as that creates a many-to-many relationship. What I don't see why it should be kept when using slowly changing dimensions.

I often use the extended IntervalMatch to do an SCD2 lookup; replace the business key in the fact table (for example, the Employee ID from the source system) with the surrogate key used in the dimension table (a unique key that links to the different versions of the Employee record). This ensures that the fact table links to the correct version of the SCD.

Once that link has been created, all the extra stuff; the original Employee ID, Start date, End date, can all be dropped, they're no longer needed. The end result is that you've replaced one key with another key; so there's no extra data in the fact table.

What would be very useful is a variant of the IntervalMatch statement that returns the key, instead of the start- and enddates of the interval.

Kind regards,

Barry

Apr 6, 2013 6:18 AM Henric Cronström Barry Harmsen in response to

Keeping the intermediate table is a risk-free approach. But of course you can instead build a solution where you have the composite key directly in the fact table. The problem is only that when you join the intermediate table onto the fact table, you face two problems:

- Maintaining the number of records in the fact table (a join can potentially change this)
- Minimizing the number of columns (every additional column will use a considerable amount of RAM)

But if you know that you have these two problems under control, you can skip the intermediate table and do what you suggest.

HIC

Apr 8, 2013 10:56 AM Richard_Chilvers

Thanks for a concise and useful explanantion of how this works. Somewhat clearer than the official description. I have already got this working but it was tricky ! It may be worth noting that care has to be taken to ensure exactly one record per interval. I experienced problems with aggregations which seems to be where this wasn't the case.

Apr 11, 2013 8:28 AM anantmaxx

Thanx Usefull!!