STRANGE MONTHS

We run a report showing data from last month's second Monday until the day before this month's second Monday. The time range from the first month's second Monday to the next month's second Monday is entered via a data-entry form.

I want to run the report for an entire year, but can't work out the right DQL to group the data from each month's second Monday until the day before next month's second Monday.

Any suggestions?

In the last issue, we published how you can group records based on a 'month' that doesn't start on the first by 'sliding' the dates so that they all fall within the same month. Thus:

```
for Table ;
  list records
    month ( DateField – 10 ) in groups ;
```

which will list from the 11th of one month to the 10th of the next.

You can use that principle here, but there are several issues to tackle first.

One problem is that the number of days to subtract to 'slide' dates into the same month is not a fixed number, but depends on where in that month is the second Monday.

The 'months' you need to describe will not be the same length as calendar months. For example, I'm looking at a 1998 calendar, where the second Monday in July is the 13th, and the day before the second Monday in August is the 9th, a total for my 'July' of 28 days. The day before the second Monday in September is the 13th, giving 'August' 35 days.

Thus I need to check to see if a given date is less than the second Monday date, and 'slide' it back to the previous month (and for the sake of simplicity, that might as well just be the last day of the previous month). Otherwise
I leave the date as it is.

Thus dates from 13th July to 31st July 1998 'remain' in July. Dates from 1st August to 9th August are 'slid' back to July; dates greater than this 'remain' in August.

Now we have to work out what is the second Monday in any given month. We only have relatively primitive date functions in DataEase, but we can combine them to achieve any date manipulation we want, with the downside that the derivation itself can look quite complicated and hard to debug.

Let's start with getting the date of the first of the month. This is pretty simple:

```
date ( month ( MyDate ) ,
01 , year ( MyDate ) )
```

Now we need, using that date as a basis, to find the first Monday. We need to advance the start of the month by a sufficient number of days to reach the next Monday. If the first of the

month is a Monday, we need to advance that date by zero days.

Weekday returns a value from 1-7, where '1' is Monday, and '7' is Sunday. If the first of the month is a Tuesday (weekday '2'), we will add 6 days to it. If Wednesday, 5 days, and so on, up to one day if it is a Sunday, and zero days if it is Monday.

Thus, the larger the weekday, the smaller the number to subtract. So at first glance we could simply subtract from 8 to get the number of days to add:

```
8 – Tuesday [2] = add six days
8 – Wednesday [3] = add five days
```

and so on.

However:

```
8 – Monday [1] = add seven days.
```

But we don't want to add anything if this is a Monday.

Here the mod function comes to the rescue. This returns the remainder when one value is divided by another. "7 divided by 7" has a remainder of zero, so that gets us the right number of days to add.

Hence the derivation to advance from any given date to the next Monday, keeping this date if it is a Monday, becomes:

```
MyDate + mod ( 8 –
weekday ( MyDate ) , 7 )
```

And to get the first Monday of the month for a given date is:

```
date ( month ( MyDate ) ,
01 , year ( Mydate ) ) +
mod ( 8 – weekday ( date ( month ( MyDate ) , 01 , year ( Mydate ) ) ) , 7 )
```

Already this is looking quite horrendous, but you could always substitute temporary variables or virtual fields to get values like FirstOfMonth.

(Note that this would be treating variables and virtuals like they are functions. With a function, you are interested in the data type and value that is the result of that function.)

Oh, we wanted the second Monday. So add seven days to this result.

For the last day of the previous month, we need:

```
date ( month ( MyDate ) ,
01 , year ( MyDate ) ) – 1
```

So for our grouping, we're looking at:

```
for Table ;
  list records
    month ( if ( MyDate < SecondMondayInMonth , LastDayPrevMonth , MyDate ) ) in groups ;
```

Or, plugging it all in:

```
month (
if ( MyDate <
date ( month ( MyDate ) ,
01 , year ( Mydate ) ) +
mod ( 8 – weekday ( date ( month ( MyDate ) , 01 , year ( Mydate ) ) ) , 7 ) + 7 ,
date ( month ( MyDate ) ,
01 , year ( MyDate ) ) – 1 , MyDate ) ) in groups ;
```

Note, however, that this will group all dates of the same month but different years together, so if the data set returned by the 'for' statement crosses several years, you should first group on the year.

This solution groups on a virtual, and therefore does not take advantage of any index. If performance is or becomes an issue, you should add this derivation as a real, indexed field to your table, and group on that instead. The principles are the same, though.

Finally, by grouping on a field, you only include values that are contained in records in that table. If for some reason there are no records for 'January' you will not get an empty 'January' group, but simply NO January group. If this is needed, you should start with a 'header' table that contains the months, and perhaps has the DateFrom and DateTo ranges already worked out:

```
for 2ndMondayMonths ;
  list records
    NameOfMonth ;
    all Events with
      ( DateOfEvent between NameOfMonth DateFrom to NameOfMonth DateTo ) DateOfEvent in order .
```