Starting One Level Higher

Filtering a Form to Show a Single Record:

Adrian Jones MBE, BA (Hons), February 2006 Adri@n10net.com

Executive Summary

The user wants to keep working on the same record as they move around the system. Custom functions can be used to keep track of the record identifier, but a function will kill the use of an index, and that kills performance.

However, we can take advantage of one of the jewels in the DataEase crown – relationships – to keep indices in use and performance at an optimum.

Rather than start with the Customer table, we create a table to parent Customer … and in fact, any table in our system. A virtual field uses the get/setglobal functions to derive its value. The parent table itself is filtered using a constant, which means we instantly locate the record in question. The virtual is the match field in its relationships to the tables that it parents; once it derives, the subform records (the ones we really want) are quickly located. The relationship works even if the match fields are not present on the form, and the screen will appear to the user as though only the Customer table is involved.

We also learn that this field can act as the match for any field type: this is demonstrated using text, numeric string and date field types.

The core application is not affected; we are not adding fields to tables that have nothing to do with that table's purpose. The concepts here form the basis for further techniques, and future articles will develop the concept of 'starting one level higher'.

Introduction To The Problem

A form's default behaviour allows access to all the records in the table that the form uses. Sometimes, though, this is simply 'too much data', both in terms of performance (especially across a file server network), and also from the user's perspective. Who wants to trawl yet again through all that information looking for the one and only record of interest, especially when you had it on screen only five minutes ago?

The action 'form open related' will create a subset of information based on the relationship used, and extends the drill-down model we are familiar with from the DOS world. However, you cannot use this action in the middle of a processing DQL, and once you close the form from which you started the drill-down session, you have to start again with the record selection.

This article explores an idea that lets you get directly and efficiently to the subset of data you want and keeps a reference to that data that you can return to at any time during a session.

Detailed Discussion

When designing a form, you can add a filter that will restrict the number of records the user can see. As long as the filter uses indexed fields, even a large dataset can be filtered quickly, presenting the subset as though they are the only records in that table.

To add a filter to a form in designer view:

1. From the Document menu, select 'Query By Model.
2. Click on the 'Select This Table's Records If:" button to display the Selection Expression Builder.
3. In the text area labeled 'Build a Table Selection Filter:', add your filter. For example, to restrict the Customers in my demonstration application, we could enter 'LastName = "Barker"'.

Okay the dialog, switch to user view, and move to the first record. You'll see the first 'Barker' in the dataset, and the status bar will display "On 1… of x". Because LastName is an indexed field, this will happen near-instantly.

If you switch to table view, you'll see that all records have the last name of Barker. Look at the View menu, and the option 'All Records' is checked; 'Queried Records' is disabled. As far as this form is concerned, it only contains Barker's.

But this is not very useful. Obviously, you will want to specify at run-time the details of the subset, and not hard-code it into a form.

A first thought might be to use the set/getglobal[1] CDFs. These let you place a value in memory that remains there throughout this session. But there is a problem: functions prevent the use of an index, and that kills performance.

To illustrate this:

1. Create and run the following procedure:

```
define "t"     number .

t := setglobal ( 1 , "Barker" ) .
```

2. Create a new form over Customer, and add this filter:

```
LastName = Getglobal ( 1 )
```

[1] And/or set/getarray

3. Switch the form to user view and, keeping your eye on the status bar, go to the first record.

This time you should see a flickering message. The first record may appear quickly, but DataEase continues to search for all matching records, which can take a while.

If you try this out on a standalone machine, you might find the performance acceptable against the demo app's 50,000-odd records in Customer. However, the performance degradation on a file server network like Novell will be painful.

This is because a function returns a value, and it has to calculate that value for each record. Picture 'The Simpsons' where Bart repeatedly asks Homer: "Are we there yet", to which he repeatedly replies "no". That is what your function is doing: asking for and getting back the same answer 50,000 times.

The reason why the hard-coded version is so quick is because it is a constant. DataEase does not need to calculate its value. It can go straight to the LastName index, and do all the clever stuff it does with indices to neatly retrieve the records you want.

Compare this to the following two DQLs[2]. The first uses a function to work out which records it wants:

```
For Customer with LastName = getglobal ( 1 ) ;
  List records
    vNameComboR .
```

And the second one uses a global variable assigned to the same value:

```
Define global "gLastName" text 50 .

gLastName := getglobal ( 1 ) .

for Customer with LastName = gLastName ;
  list records
    vNameComboR .
```

The difference should be obvious. With the function, each record has to be assessed to see if it matches the criteria. With the global variable, the value is set once, allowing DataEase to use the index.

But how can we get a value into that filter without using a function. Unfortunately, there is no variable we can use, at the moment anyway. With DfW, global variables in DQLs now persist throughout the session. That means you can set a global gCustomer at any

[2] In the demo application, I've combined these two fragments into a single procedure "UTIL_Func_v_Global", which displays two messages to show how long the searching took. Since the smallest time unit that DataEase can deal with is a second, I've added a 'while' loop to make it the performance hit more obvious on a standalone machine. Remove this if you are trying this out on a network!

point, and return to it later in an entirely different DQL.[3] However globals cannot currently be accessed in form and report documents.

In my search for something to use as a variable in a form filter, it occurred to me that this was how a relationship worked. In a main/subform situation, once the main record has its values, the subform records are filtered out of what might be a huge dataset very quickly, as long as indices are involved.

In addition, DfW allows us to create nested subforms. Whereas in the DOS world, we were restricted to two levels of data on the same form, we can now have several. So if I could find a parenting table, I could move InvoiceHeader down one level, and have InvoiceItems beneath this.

Solution: Start One Level Higher

For my own applications, I have created a generic "parent anything" table. In the demo, you will see a table called Looper, which contains a field vMatch, virtual, text 100.

To illustrate the possibilities, I've created three relationships between Looper and Customer. The table below lists the match fields, and the relationships names:

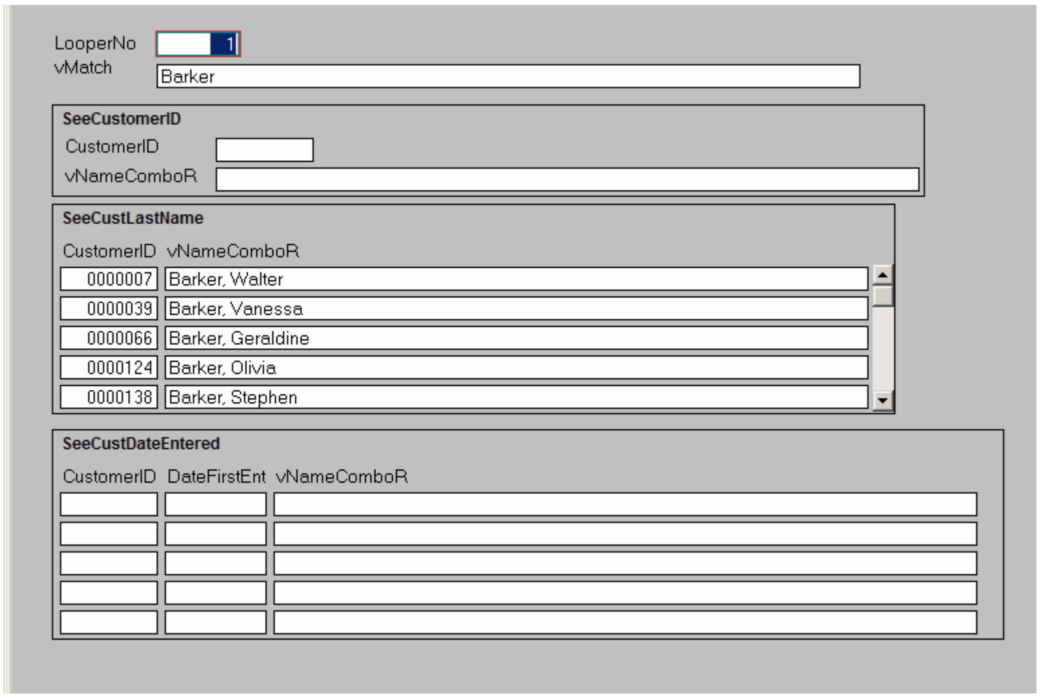
LH Table	*RH Table*	*LH Match*	*RH Match*	*LH Rel Name*	*RH Rel Name*
Looper	Customer	vMatch	CustomerID	NA[4]	SeeCustomerID
Looper	Customer	vMatch	LastName	NA	SeeCustLastName
Looper	Customer	vMatch	DateFirstEntered	NA	SeeCustDateEntered

Note that vMatch[5] is matched with a numeric string (CustomerID), a text field (LastName) and a standard date (DateFirstEntered). Also note that the relationship names reflect the Customer column being used for the match.

A form that starts with Looper and has SeeCustomerID as a subform will display the one and only Customer record, since CustomerID is unique. For the other two relationships, it will display the set of records that match the value in vMatch.

To illustrate this, look at the form called ViewCustFromLooper. Here I've added all three relationships as a subform. Below is a screen shot:

[3] In DOS, globals only remain in memory while the control procedure that created them is running.
[4] NA (Not Applicable') is my name for any relationship that I DON'T want to use.
5 I may have set off 'virtual field' and 'field type mismatch' alarm bells for some readers. You may have been told never to use a virtual in a relationship. It is true that in some circumstances, using a virtual in a relationship is a very bad idea. However, this is not one of them! An article on the use of virtuals in general is in the pipeline. As for never matching different field types and lengths, well, explore ViewCustFromLooper in the my demonstration application to see this is not strictly true…



LooperNo	1
vMatch	Barker

SeeCustomerID

CustomerID	
vNameComboR	

SeeCustLastName

CustomerID	vNameComboR
0000007	Barker, Walter
0000039	Barker, Vanessa
0000066	Barker, Geraldine
0000124	Barker, Olivia
0000138	Barker, Stephen

SeeCustDateEntered

CustomerID	DateFirstEnt	vNameComboR

Note that the scrollbar has been removed from the SeeCustomerID subform. As far as the user is concerned, the Customer details are the only thing on screen.

How does the match field gets its value?

The field is derived:

```
if ( LooperNo <= 255 , getglobal ( LooperNo ) , blank )
```

LooperNo contains a set of integers (from 1 to 1,000 in this case). The GetGlobal function can store up to 255 separate values in memory, and accesses them via a numeric index. So, as long as LooperNo is not beyond the range of the GetGlobal function, we retrieve the value stored in that location. (If it is beyond the range, this derivation will crash the application, hence the need for the 'if' statement just in case.)

vMatch is now imitating our DQL global variable. We still have to make sure it gets a value, but it only has to get that once. Having got a value, the relationship passes it on to all the matching records. The virtual will derive immediately the form is opened, and the matching records will also be immediately displayed.

In the screen shot, you'll see that I still have the word 'Barker' in memory. As a result, only the SeeCustLastName subform finds any matching records, which is does instantly.

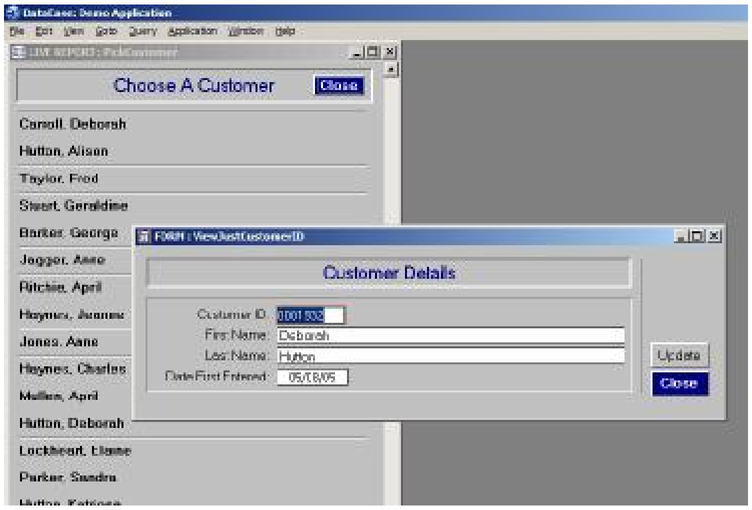
To test the values and enter them manually, I made vMatch into an allow-entry virtual. Enter '48000' and '01/01/06' and see which subforms get records … and how quick it is.

To get the document to always display the record(s) you want without having to first move to a record, open Document Properties and change the initial action to "First record'. This will move you to the first record in Looper, or whatever is your parenting table.

If you want to use another Looper record for the getglobal index, you could hard-code this into the filter for the form ('LooperNo = 2').

You do not have to display any fields from Looper itself. The forms PickCustomer and ViewJustCustomerID illustrate this. Open PickCustomer, scroll down to and click on a name. ViewJustCustomerID, which has Looper as its top table, will open with this person's details.[6] If you check in the OML behind the vNameComboR editbox object on PickCustomer, you will see that all we are doing is opening the form, using the DocumentOpen CDF.

Having picked a name, you can close both forms, and re-open ViewJustCustomerID; it will still display the same record.


DataEase: Demo Application
File Edit View Goto Query Application Window Help
Choose A Customer
Close
FORM : ViewJustCustomerID
Customer Details
Update
Close

Carroll, Deborah
Hutton, Alison
Taylor, Fred
Stuart, Geraldine
Barker, George
Jagger, Anne
Ritchie, April
Haynes, Joanne
Jones, Anne
Haynes, Charles
Mullen, April
Hutton, Deborah
Lockheart, Elaine
Parker, Sandra
Hutton, Katrina

Customer ID: 0001 [illegible]
First Name: Deborah
Last Name: Hutton
Date First Entered: 05/08/05

Single Record Subforms

With the Customer Details screen above, the fact that the actual data they can see is a subform is hidden from the user. There are no fields displayed from the main Looper table. In addition, I've removed the scrollbar from the subform (right-click on the subform object, choose Layout and uncheck 'Add Scrollbar' on the Layout dialog).

The ability to hide the structure from the user is also available in DataEase DOS[7]. If when creating a form you choose to only display one subform record at a time, you are asked if this is a One-To-One relationship. If you were to answer 'yes' to this, the subform would never scroll, and would appear to the user as though part of the main form.

[6] You can compare the performance by running PickSlowCustomer, which opens ViewSlowCustomerID. This form starts with the Customer table, and is filtered using the getglobal function.
[7] Though, of course, not the ability to add nested subforms.

Concluding Remarks

Looper gets its record quickly via a hard-coded filter. vMatch derives immediately, and the target subform records are found via the relationship. Quick and efficient.

Looper can be used to parent any table in the system, and can have relationships to other tables that return single or multiple records.

The use of getglobal as a means to persist a value throughout a session may not be common DataEase coding, but it is very similar to how a web application knows which records to display as you move from page to page. Indeed it is even possible to write such values to a file, in a similar manner to a cookie, which can keep references to records over multiple sessions.

Another 'DataEase-think' area that may need adjusting is the idea that one form fits all. An out-of-the-box form can be used to enter, view, update and delete records. But if you want more control over the user experience, you may need to create modify-only and other forms that are much more function-specific. Again, this is not dissimilar to development for the web.

But we have not changed our Customer table in any way to accommodate these ideas. We simply add extra relationships, and create extra forms over our tables.

There are many uses for the Looper table, as well as for starting one level higher. This article outlines a starting position for these possibilities.

The Essentials Step-By-Step

1. Create table Looper, with two fields:
    a. LooperNo, number, integer, unique, indexed.
    b. vMatch, text 100, virtual, derived:
    ```
    if ( LooperNo <= 255 , getglobal ( LooperNo ) , blank )
    ```
2. Populate Looper with a set of integers, say 1,000.
3. Create one or more relationships between Looper and your target tables, pairing vMatch with an appropriate field (such as the identifier) in the target.
4. Build a document over Looper, with your target table as a subform. Set the form to show first record, and filter Looper to show the record that corresponds to the global value you will set. If you are using global (2), use a filter: "`LooperNo = 2`". Remove the subform scrollbar if you only want to show a single record.
5. Create a DQL, OML script or button action that populates the global with the match value.

The ideas in this article have been developed over a number of years, most recently in a mission-critical system running in a 24/7 environment. In this application, a crucial requirement was to be able to filter out all but the current records. The main table contains over 250,000 records (expanding annually by around 25K); but there might be only 100-200 'live' records. An audit routine wraps around data-entry to track changes at field-level. At any time, any archived records could be reactivated. Rather than create two separate sets of data, one for live and one for archive material, I implemented a system that filters in a similar manner to the discussion in this article.

§ ***If you have any comments on this article, or suggestions for future material, please e-mail me at adri@n10net.com, or call +44(0)7793 148660.***

- ***I offer consultancy, technical support and mentoring services and DataEase-related products, using DataEase for Windows and ASP.NET with SQL Server. I am based in London, UK.***

About The Author: Adrian Jones is the former editor of 'Dialogue, the DataEase magazine', the creator of the 'template' applications that accompany DataEase 6, and a freelance DataEase consultant. A leading thinker in the use of the Windows product, he started with DataEase DOS 2.53 to solve the needs of his publishing business in the 1980s. His consultancy skills have been used by manufacturing, banking and charity organizations, and most recently by the Office Of Chief Medical Examiner in New York as part of the post 9/11 recovery. In July 2004, he received an MBE for 'services to British and local families in New York'.