

Understanding DataEase

DQL Problems and Annoyances

Graham Smith
PLM Consulting, Inc.
gsmith@plmconsulting.com
October 2006

Preface

As has been pointed out in other articles in this series, DQL was not originally intended to be a part of the DataEase program (refer to *Understanding DataEase – A Primer*). As a result of the way in which DQLs were implemented, situations were created in which things could and would likely go wrong. In hindsight, it would have been much better if a completely new version of DQL had been created from scratch, but the time, money, and will power simply did not exist to tackle a project of that enormity at the time. And, like everything else, the longer it has remained as it is, the more difficult it will be to replace.

The Script - Model - Body Connection

Perhaps this section should more appropriately be titled *The Script – Model – Body Disconnection* because of the problems that can be found when a DQL script is altered. But before we discuss that, it is worthwhile reviewing another section of the *Primer*.

The QBM-Document (which underlies both Forms and Report Documents) is interactive between the Model and the Document Body. That is to say changes to the Model change the body and vice versa. To get DQL into DataEase, the DQL component was grafted into the QBM-Document in such a way as to let the DQL create the Model which would, in turn, create the Body. But there is no vice versa in DQLs.

To complicate things further, there are variables to deal with. Consider the following DQL:

```
define "tName" text 20 .
for Companies
with State ="M*" ;
tName := Name .
list records
    State ;
    Name ;
    tName .
end .
```

This would seem to be simple enough but since tName is not a part of the underlying table, how does it fit into the model? The answer is that it is treated similarly to something called a “Form Only Virtual Field” (FOV). This is a field that only exists on the form and is not a part of the table. These type of fields can be useful in certain circumstances but have long been known to be a bit eccentric in their behavior. The reason is that they don't have a solid footing in the model and changes to the model can cause them to stop working.

The situation is the same in the above DQL if you change the WITH statement. What *can*

happen is that, while tName continues to appear on the body, it is blank when the DQL is run. The same thing can happen with other changes to the DQL as well. The only solution if this happens is to delete the field from the body and re-compile the DQL so that the field is recreated.

But the truly odd thing about this is that it does not happen all the time. If the above DQL had been saved and opened from the catalog, then changed, the problem would be far less likely to happen. From a technical perspective, what appears to be happening is that the projection of tName on the body is becoming disconnected from the data model under some circumstances. Exactly what that combination of circumstances are is up for debate, but it does point out something that is good to know – when making changes to DQLs, it is prudent to save changes and test frequently.

Summing Temp Variables

Back in the DFD days, it was always debatable as to whether or not it was safe and proper to use statistics with temp variables. Particularly in the early days when memory was very expensive and very scarce, people would have problems with sums coming out wrong when this was tried. For that reason, it was considered to be a bad practice and a lot of people avoided it. But as time went on, it became more common to do this and it soon became a given that it would work. But it seems that DE has a little gotcha that revolves around when certain things take place internally. Consider the following DQL:

```
define "tValue" number .
for TableA ;
tValue := Value .
list records
  Value : item sum ;
  tValue : item sum .
end .
```

What you will find when you run this query is that the sum for tValue will be lower than the sum for Value by the amount of Value in the last record. Exactly why this happens is beyond my understanding but it has to do with when things happen. For some reason, the last tValue assignment never becomes a part of the sum. My guess is that it has something to do with the difference between what happens internally when one record changes to the next as opposed to the last record ending the loop. Something just doesn't happen in that last step. Another example of the same problem as above can be seen in the following DQL:

```
define "tValue" number .
for TableA ;
tValue := tValue + Value .
list records
  tValue : item max .
end .
```

The max statistical value should be the same as the last value for tValue in the items, but once again, it is low by the value of the last record.

I'm probably more sensitive to this issue than others might be because of on particular application I developed and support. About half of the DQLs, and there are several hundred, deal with numbers and there are a whole pile of them that use temp variables as well. As a result I have come to develop a set of techniques I follow as a routine. We'll have a look at these in a bit but first we need to look back at a problem discussed in the Understanding DataEase article on Groups.

Oddly enough, however, if the list records included a field in groups, this problem would not show up. Once again, this points to something that happens in the flow of the actions internal to the query that is creating this problem. But in this case, a different problem can surface.

Formatting Temp Variables Summary Values

Lets take the previous DQL and expand it a bit:

```
define "tValue" number .
for TableA ;
tValue := Value .
list records
  OrderDate in groups ;
  tValue : item sum .
end .
```

This DQL will produce a output that will have three levels. By that I mean that tValue will appear as an Item, a group Sum, and an overall Sum. In each of these cases, tValue will be a floating point number and the format will need to be changed to an Integer or a Fixed Point number.

The problem happens when you get to the overall Sum. For some reason, you will find that attempting to change the format will produce an "Error 503 Setting Field Summary Type". Exactly what this error means and why it happens is another of those mysteries.

There is a workaround to both of these problems, which is to use an outer loop and a separate variable that does not need to be summed. This is another technique developed by Adrian Jones that has come to have wider value than was originally considered.

The idea is to use a table with only a single record in it (sometimes called OneRecordTable because we are not very imaginative sometimes) as an outer loop so that temp variable fields can be placed in the format without having to sum them up. Here is an example of the first DQL restated:

```

define "tValue" number .
define "tValueS" number .
for OneRecordTable ;
for TableA ;
tValue := Value * 10 .
tValueS := tValueS + tValue .
list records
  OrderNbr ;
  tValue .
end .
list records
  tValueS .
end .

```

What this will do is place tValueS at the end of the report. There will be no problems because it does not need to be summed and it can be formatted as you wish. Because this has proven to be a very useful technique, I frequently include the outer loop in some DQLs where it is not needed just in case it is decided to add some summary information at a later time (if you add this to an existing query, you will have to completely redo the report format).

Deriving Variables from Related Tables in Groups

It is sometimes necessary to use a variable to hold the value from a related record. When grouping is involved, this can create a rather strange problem.

```

define "tDept" text 15 .
for EMPLOYEES ;
  tDept := any RelatedDepartments DepartmentName .
list records
  Gender in groups ;
  Name ;
  tDept .
end .

```

When this DQL is run, it will appear to “get stuck” on the Department for the first record in each group. According to Pete Tabord, the reason is that adding a group “shoves the main form and it's permanent relationships down one level in the MultiView. But it does not move the relative position of the temps or ad-hocs, thus they are only recalculating the temp value with the group changes, not the individual record.” If you look at the Model of this DQL, you will see that RelatedDepartments is at the same level as the Group, thus it is only going to derive when the group changes.

When Everything Goes Wrong

There are times when it seems that it is impossible to find a solution because a whole

bunch of different problems all conspire to plague the same DQL.

```
define "tIRA" number .
define "t401" number .
for Distributions ;
  tIRA := sum of _Dist_IRA named "nIRA" with (To401K not=yes) Amt .
  t401 := sum of _Dist_IRA named "n401K" with (To401K =yes) Amt .
list records
  PartialType in groups ;
  DistributionDate in order ;
  EmpID : item count ;
  Name ;
  GrossAmount : item sum ;
  TaxableAmt : item sum ;
  tIRA : item sum ;
  t401 : item sum .
end .
```

This DQL came directly from a DFD database that was converted to DE65 this past year. This is fairly typical of the type of DQLs that I had to deal with in the migration. Every time I made a change to this to fix one problem, another cropped up. Finally, in desperation I did something that I should have done in the beginning.

Solving Problems with Virtual Fields

Each day that I sit down to work on a database, I recite the mantra “DFW is not DFD”. It makes me feel better but doesn't really help because I am a creature of habit and have 15 years of doing things certain ways to get over. While I have gotten used to dealing with the fact that there were things I used to do in DFD that won't work the same in DFW, I often forget that the reverse is also sometimes true. That is, there are things that I used to avoid doing in DFD that are much more acceptable in DFW. And one of those is the use of Virtual Fields.

I leaned to be very frugal in the use of virtual fields in DFD since they added a lot of overhead to things. This was particularly true of virtuals that referenced related forms. But the overhead is so much lower and the advantages of these fields is so much greater in DE that I find myself creating a lot more of them than I used to.

In the above DQL, there are a number of reports that need to separate out the amount of money in a Distribution that went into an IRA as opposed to the amount that went into a 401K. In DFD, this was always dealt with in the DQL, but in DE it is more practical to make these virtual fields in the table. Doing so, reduced the above DQL to this:

```
for Distributions ;
list records
  PartialType in groups ;
  DistributionDate in order ;
  EmpID : item count ;
  Name ;
  GrossAmount : item sum ;
  TaxableAmt : item sum ;
  AmountToIRA : item sum ;
  AmountTo401K : item sum .
end .
```

Solving Problems with Holder Files

I've been fond of holder files for a long time now, and find them to be even more valuable in DE than they were in DFD.

A good example of this can be found in an application I am working on now. In DFD, there were a series of complicated DQLs that were written that all pulled data from the same set of tables. They were quite complicated and were nearly identical except for selection criteria and how some values were calculated. After looking at them, I realized that I was going to encounter a number of problems of the sort that I have documented here. This was further emphasized when I was told that they frequently had to make changes to the DQLs to add, remove, and change data.

Rather than having to deal with half a dozen fussy DQL reports, I decided to create a holder file and create half a dozen not so fussy DQL procedures that would write to the holder file and one simple Report to print the data out. I saved myself a lot of grief and changes are a snap to make.

Summary

I wish I could say that there was a bright light at the end of this tunnel - that these and other DQL problems are going to be fixed "real soon now", but that's just not going to happen. The bigger problem is that marketing pressures forced the development team to wedge DQLs into a structure that they were never designed to fit. To make it worse, there was an insistence that the syntax remain the same as it was in DFD and that it behave the same. Unfortunately, both of these requirements were doomed to problems from the very beginning.

This is a classic case of a singing dog. People got so wrapped up in trying to get the dog to sing in tune that they missed the larger point that by all rights, the dog should not be able to sing at all.

It has been argued that DQL, as it currently exists, should never have been placed into the

product in the first place. But it has equally been argued that if it had not been then DFW5 would have never been accepted. This is the kind of debate that can never be resolved, we can only accept the fact that it is there and move on.

There is no generic set of rules that I can give you that will solve all the problems in DQLs, but there are a couple big tips that I have found help me a lot.

- Wherever possible, use a Report rather than a Procedure. DFD users are so used to doing things with DQLs that we often forget that Reports are the real native reporting tool in DE and that they are much more sophisticated than Quick Reports were in DFD. I'd be willing to wager that half the DQLs that exist right now in DE apps could be done better using Reports instead.
- Test, test, and test again. Try different things. Experiment. If you have not found two different ways to do the same thing, you haven't tried.
- Report all bugs to Sapphire, unless you are positive that they have already been reported. They will never get fixed if they don't get documented.

As a final note, the issues I have discussed here have not passed notice by Pete and Trina Tabord of Sapphire (head of development and head of Q/A, respectively). I had a long discussion with them last year about DQLs and they both have some definite ideas about how to solve some of the problems. The first phase of this came with the development of DE7 and the change to a new version of the language (C++) and the compiler. The second phase is underway now with a cleanup of the code.

The third phase has yet to begin but is already in planning. This will consist of a review of how the entire internal DQL process works. I'm reminded here of a song by Mark Knopfler. The chorus goes, "who put old pigweed in the mulligan stew". I'm probably not telling any secrets when I say that there have been a lot of hands in the DataEase stew over the years and not all the changes made were fully documented (imagine that), so it's going to take some time to figure out what some of the funny tasting bits are and where they came from.

So, perhaps there is some light at the end of the tunnel after all. Let's just hope it's not an express train.