

# Understanding DataEase Groups

Graham Smith  
PLM Consulting, Inc.  
gsmith@plmconsulting.com  
October 2005

There are few things in DataEase that create a bigger misunderstanding than Grouping. This is in part due to the fact that users of the DOS program never really understood what Grouping did, and in part because Grouping works differently in DataEase than it did in DFD. It is also because Grouping is generally viewed as a way of sorting or organizing data without much thought given to just how it works. And like so many other things in DataEase, understanding how something works is the first step to using it correctly. The following is taken from Understanding DataEase : A Primer.

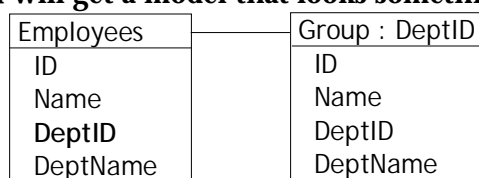
*Most people think of groups as just another way of ordering the information. In fact, the ordering of data is largely just a useful by-product of the way groups work. The command "In Groups" is, actually, a Relational Operator.*

*Grouping creates what could be, arguably, called a pseudo-table. I say arguably because Pete Tabord [chief developer at Sapphire] has gone to great lengths to explain to me and others that no matter what it looks like to us, it is not a pseudo-table. Never-the-less, if you create a Group in a Report, you will see that it creates a new table-like structure that, in many ways, is analogous to a Form – Subform construction. It even uses those terms in the report body. A good term to describe this from the perspective of the Model is a node.*

*Turning our attention now to the body of the Report, we will see that we have four containers. The Main Form, the Main Form Record, the Group Subform, and the Group Record. Whatever fields we chose in the Main node in the QBM will be in the Main container in the Body. Same for the Group node/container. This said, it becomes rather self-evident that to produce a Report Body with things where you want them, you have to select those things in the proper nodes in the QBM. But you would be amazed at how long it can take to fully grasp this simple concept.*

## Grouping in Reports

Before trying to tackle DQL's we need to first look at grouping in Reports. Let's take a simple example to start with. Employees grouped by Department. When you select the DeptID and group on it, you will get a model that looks something like this:



The Model shows something that looks the same as a Form – Subform would in a Form. So if we look at the Model as being representative of what the body of the Report is going to look like, it is apparent that we need to select DeptName in the Employees node rather than in the Group node if we want it to appear along side the DeptID. Selecting it in the Group node would cause it to be repeated for every record in the group like this:

```

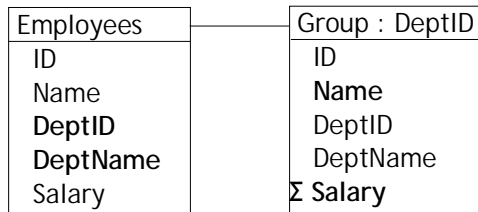
DeptID: 001
  Bob Jones  Accounting
  Jane Smith Accounting
DeptID: 002
  Jim Brown  Sales
  Sally Lu   Sales
  
```

Instead of what is really wanted:

```

DeptID: 001  Accounting
  Bob Jones
  Jane Smith
DeptID: 002  Sales
  Jim Brown
  Sally Lu
  
```

To include the Salary for each person and the total for the department, select the Salary in the Group Node and sum on it:



You could just sum on Salary in the Employees node, but you then only get a sum at the end of the report. Test out different variations of this and examine the results. You will soon get a better feel for how to structure the Model to produce the desired results.

As an aside, you need to be aware that when you put a sum on a field, the QBM Dialog will usually also select (bold) that field to be displayed. However, there are times when this is not what is needed. You can un-check Display on any summary field and the summary will still display but the “item” value in each record will not. As a further aside, the same goes for sorting – you can sort on a field without displaying it in the body.

### Nesting Groups

It is possible to create more than one group in a Report by “nesting” the groups. For example, using the above table, you could group first on DeptID then on Salary. This would give you an output something like this:

DeptID: 001 Accounting  
Salary: \$2000 per month  
Bob Jones  
Jane Smith  
Salary: \$3000 per month  
Jim Johnson  
Sandy Fields  
etc. . .

## Grouping in Forms

Most people never think about doing this, but data can be grouped in Forms just as it can in Reports. In fact, doing this points out the main point made before that Grouping is a relational construct rather than a method of sorting.

There is, however, a catch, you cannot search in the Group: subform. For this reason, it rather limits what can be done with these forms.

## Grouping in Procedures

I wish I could say that Procedures (DQL) need only be used to process (add/modify/delete) records. Alas, there are times when Reports simply cannot do what you need to do. Before delving into Procedures, it is worthwhile referring back to [Understanding DataEase : A Primer](#) to refresh memories.

*...Report Documents are interactive between the Model and the Document Body (Format). Procedures, however, add a third layer to this equation that pretty much pushes all the previous rules out the window. In other words, Procedures have their own sets of rules as to how they behave and how you must deal with them.*

*The DQL component [was] grafted on to the QBM-Document model in such a way as to let the DQL create the Model which would, in turn, create the Body.*

*...Procedures are a one-way street when it comes to the Body. Changes in the DQL will necessitate changes in the Model which will necessitate changes in the Body.*

Since the DQL is going to be responsible for creating the Model, and since the Model cannot be managed as it is in a Report, it becomes necessary to write the DQL in such a way as to produce a Model which will, in turn, produce a Body with things in their correct place to begin with. The trick is to examine the DQL and think of it in terms of where in the Model each field is going to go.

If we return to our previous example, and create a Procedure rather than a Report, we can see how placement of things in the list records effects the way the body will be laid out.

```
for Employees ;  
list records  
  DeptID in groups ;  
  DeptName ;
```

```
Name .  
end .
```

**This will produce an output that looks like our first report example:**

```
DeptID: 001  
  Bob Jones  Accounting  
  Jane Smith Accounting  
DeptID: 002  
  Jim Brown  Sales  
  Sally Lu   Sales
```

**Whereas:**

```
for Employees ;  
list records  
  DeptName ;  
  DeptID in groups ;  
  Name .  
end .
```

**Will put things where we want them:**

```
DeptID: 001  Accounting  
  Bob Jones  
  Jane Smith  
DeptID: 002  Sales  
  Jim Brown  
  Sally Lu
```

The reason is that when the command “in groups” is encountered, it creates a new node in the Model at that point. Everything after it goes into the new group sub-form container until a “.” is encountered which ends the group. By putting some fields above the command “in groups”, those items are going to be included in the main form container. Interestingly, DataEase supports the use of multiple list record which may be used in a similar manner, but care must be taken when doing this because the results may not always be what is expected.

**Example:**

```
for Employees ;  
list records  
  DeptID in groups ;  
  Name .  
list records  
  DeptName ;  
end .
```

Will produce essentially the same body layout as the previous DQL with the difference of where in the main form container DeptName will be placed (ahead of DeptID or after it) when the layout is first created.

Avoid Putting the Cart Before the Horse

In DFD, it was not possible to put the Items at the Same level as the Group Header in the

**Format. This meant that the only way you could do a layout was like this:**

```
DeptID: 001  Accounting
      Bob Jones
      Jane Smith
DeptID: 002  Sales
      Jim Brown
      Sally Lu
```

**However, in DE65, the subform container can be moved about within the Group record container to achieve a result like this:**

```
DeptID: 001  Accounting  Bob Jones
                        Jane Smith
DeptID: 002  Sales       Jim Brown
                        Sally Lu
```

**When doing this, however, be aware that there *can* be some odd problems if you put the subform container “higher” than the field being grouped on in the Group record. The reasons are complicated but come down to creating a situation in which DataEase tries to display the values of the fields in the subform are before it has the value for the main record. It gets confused, as it were.**

**When Not To Group**

**In DFD, it was necessary to put records in groups if you had a nested FOR or an ALL in the list records. This is not necessary in DE65 and can actually make things more complicated if you do it. By complicated, I mean that it creates a completely unnecessary container that just takes up space in the body.**

**Groups within Nested For's**

**There is a rather curious bug (currently under investigation) in DE65 that only shows up under some circumstances and can produce some rather odd problems. The bug shows up if you have a nested for below a group as follows:**

```
for TableName ;
list records
  FieldName1 in groups ;
  FieldName2 .

  for RelatedTableName ;
  list records
    FieldName .
  end .
end .
```

**For the most part this structure works without any problems but I have had some odd things turn up, particularly when variables are involved. Until the exact problem can be tracked down and fixed, here is a suggestion. The same DQL can be written in a way that**

**will not produce a problem with variables :**

```
for TableNme ;  
list records  
  FieldName1 in groups ;  
  FieldName2 ;  
  all RelatedTableNme FieldName .  
end .
```

## A Brief Note on Bugs

As a general rule, I have been replacing the use of ALL with FOR. It is perhaps because of this that I was the “lucky” person to find this bug. A bug can go unnoticed for a very long time until someone does something different, and DE65 developers are always looking for new ways to do old things. Replacing ALL with FOR is a case in point.

It can also be difficult to determine exactly where a bug lies. In this particular case, the question is whether it is the combination of nested FOR's and GROUP that is the problem or whether the problem is something having to do with Variables. It may just be that the combination causes a problem to manifest itself that may not otherwise show up.

And here is where you can help. When you run into something you believe is a bug. Try and see if you can create a reproducible example of it and then submit a bug report. You can report bugs at [www.sapphiredev.com](http://www.sapphiredev.com)

One of the reasons for bringing this up at this point is because there are a few things about Groups that may or may not be bugs as such. As has been previously stated, Grouping works differently in DataEase than it did in DFD and these things may be the result of trying to force it to do things in a way it does not want to do them.

## Incrementing Variables in Groups

A technique used in DFD to increment variables in groups has been dubbed the “Group Break” since it involves keeping track of the grouping field and resetting the variables when the group changes. Here is an example of the DQL in a DFD application.

```
define "tCount" number .  
define "tDept" text 15 .  
  
for EMPLOYEES ;  
  
if tDept not= Department then  
  tDept := Department .  
  tCount := 0 .  
end .  
  
tCount := tCount + 1 .  
  
list records
```

```
    Department in groups with group-totals ;
    tCount .
end .
```

**When this same DQL is set up in a DE65 application, it is necessary to change the procedure in order to force the temp variable into the same container as the group. There are two ways to do this:**

```
list records
  tCount ;
  Department in groups with group-totals .
```

-- or --

```
list records
  Department in groups with group-totals .
list records
  tCount .
```

**Unfortunately, neither of these work - they don't ever increment the variable. This is even true if the group break itself is removed. So far there are only two ways I have found to work around this. The first is to write the values to a holder file first, the second is to change the list records to read:**

```
list records
  Department in groups with group-totals ;
  tCount : max .
```

**It seems that for the variable to increment correctly, the variable has to be contained within the Group: node. If it is not there, then it essentially gets "passed over".**

**There is a variation of this that is even a bigger can of worms.**

```
list records
  tCount ;
  Department in groups with group-totals ;
  Salary .
```

**In this case, the variable does increment, but displays the results after the group changes. In other words, it displays the value for the previous group. Once again, this can be cured by putting the variable after the group with a : max.**

### 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.

There are different solutions depending on whether the relationship is predefined or an ad-hoc. Note that ad-hocs add their own level of challenges. The first solution is to eliminate the variable completely:

```

for EMPLOYEES ;
list records
  Gender in groups ;
  Name ;
  any RelatedDepartments DepartmentName .
end .

```

This produces a much simpler DQL and puts the relationship subordinate to the Group in the model. Whenever possible, this should be used. It cannot, however, be used for an ad-hoc.

```

for EMPLOYEES ;
list records
  Gender in groups ;
  Name ;
  any Departments with (DeptID =Employees DeptID) DepartmentName .
end .

```

While this produces a model that looks like it should work, it is still tied back to the main record which only changes with the change of the group. This can be overcome through the use of a variable as follows:

```

define "tID" numeric string 5 .
for EMPLOYEES ;
  tID := DeptID .
list records
  Gender in groups ;
  Name ;
  any Departments with (DeptID =tID) DepartmentName .
end .

```

The tID variable changes with each record because each record is still being read. Which leads to the question of why the following fails:

```

define "tID" numeric string 5 .
define "tDept" text 25 .
for EMPLOYEES ;
  tID := DeptID .
  tDept := any Departments with (DeptID =tID) DepartmentName .
list records
  Gender in groups ;
  Name ;
  tDept .
end .

```

**The answer would seem to be that even though tID is changing with each record, the relationship itself is not rederiving. And this would seem to be the crucial point – it is not so much that the value is getting stuck as it is that the derivation is not being triggered except when the group changes.**

**As has been pointed out, using Employees DeptID puts the relationship at the level of the Main table. There are, however, two ways to put the relationship at the group level. If you look at the model of one of these DQLs, you will see that there is a node labeled Group: Gender. You can actually uses this reference in the DQL as follows:**

```

for EMPLOYEES ;
list records
  Gender in groups ;
  Name ;
  any Departments with (DeptID =Group: Gender DeptID) DepartmentName .
end .

```

**Another approach to dealing with this is to take advantage of the fact that you can name the various nodes in the model. This allows you to name the group and use that name in the ad-hoc derivation as follows:**

```

for EMPLOYEES ;
list records
  Gender in groups named "grpA" ;
  Name ;
  any Departments with (DeptID =gprA DeptID) DepartmentName .
end .

```

**This gives us a very clean and relatively easy DQL to read. This can be very important in larger DQLs. It also makes the Model slightly easier to understand. For this reason, it is worth considering naming groups as a matter of course. (see Note)**

**There are times, however, where it is not possible to use any of the methods mentioned here. The most common reason is when it is not a simple lookup that is needed, but a variable that needs to be incremented with data from a non-related table. When this happens, there is still a way to make this work – that is to insert a dummy loop inside the main loop like this:**

```

define "tSales" number .
for EMPLOYEES ;
  for OneRecordTable

```

```

    tSales := sum of Sales with (EmpID = Employees EmpID) Amount.
end .
list records
  Gender in groups ;
  Name ;
  tSales .
end .

```

The reason this works is very instructive, as it goes back to a similar DFD technique. The idea is that “jumping out” of the main record loop to an unrelated single record table, allows you to create and use relationships that you might not be able to use inside the main loop. In this case, it keeps the relationship inside the OneRecordTable loop from being in any way connected to the Main table, thus causing it to derive with each record. This same technique may have a number of uses that I have not yet fully explored.

**Note:** Introducing a new relational operator (in this case a Group) into an DQL or changing one, may force the model to substantially alter the body of the procedure. In the case of naming a group, this causes a major change in the model and will most certainly alter the body. This puts this technique into the category of something best done at the start of the DQL rather than later.

## Summary

The key to using groups effectively is to structure the Report or Procedure so that fields go into the correct container. Ensuring that this is done will guarantee that the Body will have things where you want them. Failure to do this will guarantee endless headaches.

As has been found, however, Groups are not without certain problems. For this reason, you must take care and, as with all things, thoroughly test all Reports and Procedures.