The Power of Arguments in Records.cfc

Posted At : November 16, 2010 10:00 AM | Posted By : Steve
Related Categories: com.sebtools

We have come a long way in our question to easily manage records and files. Taking advantage of the com.sebtools library, we have a single file that effectively provides a component for every table and methods for every basic action on that table. I would like to focus in a bit more on the API for these components now. Specifically, I would like to focus on how we can get the recordset we want - even for operators other than equality.

What we want is to be able to specify the exact recordset that we want - which columns, which rows, which pages - by passing in arguments. Ideally, we could do this just in our data definitions without writing any extra code.

This entry will make more sense if you have read the rest of the series.

- **Super Easy CRUD/File Management**
- **Basics of Manager.cfc XML**
- **Getting Data to Records.cfc**
- **Multi-Table Applications with ProgramManager.cfc**

Filter By Equality

Equality operators are, of course, built in. You can filter by records with values equal to any field simply by passing in an argument with a name matching the field and with the value for which you want to filter. For example, to get an employee with a first name of "Steve" and a last name of "Bryant", we could use the following code. For example:

```
<cfset qEmployees = Application.HR.Employees.getEmployees(FirstName="Steve",LastName="Steve")>
```

You can even do this with **relation fields** (which don't actually exist in the database). For example, you could filter by the "FullName" field (which is a **concatenation relation field** of first and last names):

```
<cfset qEmployees = Application.HR.Employees.getEmployees(FullName="Steve Bryant")>
```

DataMgr arguments

You can also pass in any of the arguments from **DataMgr.getRecords()**.

This includes the "fieldlist" argument to limit the fields returned. I would recommend using this argument unless you are sure that you will need every field (you can go ahead and leave it out then - "SELECT *" will never be generated internally). You can include real fields and/or relation fields in this list. You cannot, however, include SQL. Anything in the list that isn't a field for that table will be ignored.

For example, to get only the EmployeeID and FullName fields of all employees:

```
<cfset qEmployees = Application.HR.Employees.getEmployees(fieldlist="FullName")>
```

You can also take advantage of these arguments for paging. For example, if you want to get employees 11-20 of a recordset, you could use the following code:

```
<cfset qEmployees = Application.HR.Employees.getEmployees(maxrows=10,offset=10)>
```

This will work on any database. The work will be done by the database for any database that supports that functionality.

Filter By Inequality

Let's say that we have a "HireDate" field and a "Salary" field for our employees. Here is the XML for the table definition with those fields:

```
<table entity="Employee" labelField="FullName">
   <field fentity="Department" />
   <field name="FirstName" type="text" Length="80" label="First Name" required="true" />
   <field name="LastName" type="text" Length="80" label="Last Name" required="true" />
   <field name="HireDate" type="date" label="Hire Date" />
   <field name="Salary" type="integer" label="Salary" />
   <field name="FullName" type="relation">
      <relation
         type="concat"
         field="FirstName,LastName"
         delimiter=" "
      />
   </field>
</table>
```

With these fields, we are less likely to look up employees by an exact hire date or salary (though we may want to do that as well) and more likely to want to look up employees hired before or after a date or making more or less than a certain salary. For this, we are going to use DataMgr's **named filters**. While we're doing this, we may as well allow employees to be found by a partial match of the last name.

So, here is our expanded XML:

```
<table entity="Employee" labelField="FullName">
   <field fentity="Department" />
   <field name="FirstName" type="text" Length="80" label="First Name" required="true" />
   <field name="LastName" type="text" Length="80" label="Last Name" required="true" />
   <field name="HireDate" type="date" label="Hire Date" />
   <field name="Salary" type="integer" label="Salary" />
   <field name="FullName" type="relation">
      <relation
         type="concat"
         field="FirstName,LastName"
         delimiter=" "
      />
   </field>
   <filter name="MinSalary" field="Salary" operator="GTE" />
   <filter name="MaxSalary" field="Salary" operator="LTE" />
   <filter name="HiredBefore" field="HireDate" operator="LT" />
   <filter name="HiredBy" field="HireDate" operator="LTE" />
   <filter name="HiredSince" field="HireDate" operator="GTE" />
   <filter name="LastNameLike" field="LastName" operator="LIKE" />
</table>
```

In this example, all of the named filters point to physical fields in the database, but they could just as easily point to relation fields.

Now we could find any employee with a last name starting with "B":

```
<cfset qEmployees = getEmployees(LastNameLike="B%")>
```

Similarly, we could find any employee who was hired in 2009 and who is making at least $250,000:

```
<cfset qEmployees = getEmployees(HiredSince="2009-01-01",HiredBefore="2010-01-01",MinSalary=250000)>
```

As a side note here, while the Employees component doesn't have a built-in method to do it, you could use DataMgr to update these records directly. Here would be code to update all of the employees hired in 2009 and making at least $250,000 so that they would be making only $250,000:

```
<cfset sWhere = {HiredSince="2009-01-01",HiredBefore="2010-01-01",MinSalary=250000}>
<cfset sSet = {Salary=250000}>
<cfset Application.DataMgr.updateRecords("hrEmployees",sSet,sWhere)>
```

You could even delete them all, if you wanted:

```
<cfset sWhere = {HiredSince="2009-01-01",HiredBefore="2010-01-01",MinSalary=250000}>
<cfset Application.DataMgr.deleteRecords("hrEmployees",sSet,sWhere)>
```

In this example, it would have been just as easy to write the SQL for the update (or the delete). Using DataMgr for this automatically ignores logically deleted records and automatically applies cfqueryparam correctly. The real advantage, however, comes in if you want to filter by relation fields - wherein it is nice to take advantage of DataMgr's built in understanding of those fields.

Conclusion

Hopefully it goes without saying, but any of these arguments can be used in any combination. So, for example, we could get rows 11-20 with just the full name of employees making at least $250,000:

```
<cfset qEmployees = getEmployees(maxrows=10,offset=10,fieldlist="FullName",MinSalary=250000)>
```

The order of the arguments doesn't matter.

Now we have a flexible set of arguments for getting just the recordset that we want from any table, including for inequality operations and we still haven't had to write any code outside of our XML table definitions. We will soon though.

The **com.sebtools package** is open source and free for any use.