Relating Tables with Manager.cfc

Posted At : November 9, 2010 10:00 AM | Posted By : Steve
Related Categories: com.sebtools

Today we continue our quest to use com.sebtools to easily manage records and files in our example HR application. So far, we have **defined our data set with XML**, **gotten that definition into Records.cfc**, and **consolated our multi-table application with ProgramManager.cfc**. The last step wasn't necessary for a multi-table application, but does make things nice.

What we don't have, however, is any interaction among our tables. We have Departments and Employees, but no relationship between the two. For this example, an employee can work for one (and only one) department.

There are a few pieces of information that are helpful for most of these sorts of relationships. We would like to know the name of the department for every employee (and perhaps a boolean indicating if that employee is in a department). We would also like to have the number of employees for any given department and a boolean indicating whether or not that department has any employees. Fortunately, **Manager.cfc** can provide all of this for one short line of code.

We can add a new field into the "Employees" table with an fentity="Department" attribute.

What Do We Do?

```
<field fentity="Department" />
```

Here is the resulting complete code for our HR.cfc file:

```
<cfcomponent displayname="HR" extends="com.sebtools.ProgramManager" output="no">

<cffunction name="xml" access="public" output="yes">
<tables prefix="hr">
   <table entity="Department" />
   <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="FullName" type="relation">
         <relation
            type="concat"
            field="FirstName,LastName"
            delimiter=" "
         />
      </field>
   </table>
</tables>
</cffunction>

</cfcomponent>
```

Note that the above code is all that we will have to write. The rest of the code examples are only illustrative of what is representeded internally.

What Do We Get?

A call to Application.HR.Departments.getDepartments() would now return the following columns:

- DepartmentID: Integer primary key
- DepartmentName: varchar
- NumEmployees: integer of the number of employees for each department
- HasEmployees: boolean indicating if the NumEmployees value exceeds 1

A call to Application.HR.Employees.getEmployees() would now return the following columns;

- EmployeeID: Integer primary key
- DepartmentID: Integer
- FirstName: varchar
- LastName: varchar
- FullName: varchar
- Department: varchar (the value of the DepartmentName for each employee's DepartmentID)
- HasDepartment: boolean (a boolean indicating if the DepartmentID is associated with an existing department record)

What Happens Internally?

If everything above is what we want, then it probably doesn't matter how it is achieved. If not, then the internal representations matter so that we can over-ride built-in defaults to achieve what we do want.

If you haven't already, read the **Basics of Manager.cfc XML**. This section won't make much sense without it.

The "fentity" attribute (which is all we set initially in this example) will set a default value for the "ftable" attribute of the same field. So, that field is *effectively*:

```
<field fentity="Department" ftable="hrDepartments" />
```

Any field with an "ftable" attribute will have the following default attribute values set:

- type: "fk:integer".
- name: The name of the primary key field for the referenced table (prepended with "Parent" if it points back to its own table).
- label: The value of the "labelSingular" attribute for the referenced table (prepended with "Parent " if it points back to its own table).
- listshowfield: The makeCompName() of the "methodSingular" attribute of the referenced table (prepended with "Parent" if self-referencing).
- **subcomp**: The value of the "methodPlural" attribute of the referenced table.

In this case, that would mean that this field would *effectively* look like this:

```
<field
   fentity="Department"
   ftable="hrDepartments"
   name="DepartmentID"
   label="Department"
   listshowfield="Department"
   subcomp="Departments"
/>
```

The important things to notice here is that the field automatically sets a usable default field name and label.

Whenever a field with "ftable" exists, Manager.cfc looks for a field with a name matching the "listshowfield" attribute (from above - "Department" in this case). If it doesn't exist, then Manager.cfc creates it as a **label relation** to the "labelField" of the related table ("DepartmentName" in this case).

It creates a "Has#attributes.listshowfield#" field ("HasDepartment" in this case) if it doesn't already exist as a **has relation** for the previous field. This will be a boolean indicating if the employee is in a department.

It creates a field using "Num" and the table's "methodPlural" attribute ("NumEmployees" in this case) in the referenced table if it doesn't already exist as a **count relation**. This will indicate how many employees are in each department.

It creates a "Has" field like above ("HasEmployees") as a boolean indicating whether or not each department has employees.

So, our XML *effectively* looks like this:

```
<tables prefix="hr">
   <table entity="Department">
      <field name="NumEmployees" label="Employees">
         <relation
            type="count"
            table="hrEmployees"
            field="EmployeeID"
            join-field-local="DepartmentID"
            join-field-remote="DepartmentID"
         />
      </field>
      <field name="hasEmployees" label="Has Employees?">
         <relation
            type="has"
            field="NumEmployees"
         />
      </field>
   </table>
   <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="FullName">
         <relation
            type="concat"
            field="FirstName,LastName"
            delimiter=" "
         />
      </field>
      <field name="Department" label="Department">
         <relation
            type="label"
            table="hrDepartments"
            field="DepartmentName"
            join-field-local="DepartmentID"
            join-field-remote="DepartmentID"
         />
      </field>
      <field name="HasDepartment" label="Has Department?">
         <relation
            type="has"
            field="Department"
         />
```

```
      </field>
   </table>
</tables>
```

What Next?

The above code illustrates how Manager.cfc automatically creates a handful of **Datamgr relation fields** for us. Keep in mind, however, that many more types of DataMgr relation fields could easily be created by hand as well.

So far, we have dealt with relatively ideal circumstances on our journey. The next few entries will deal with slightly more real-world scenarios and we will take a look at how the com.sebtools package can still help when your situation doesn't exactly match what Manager.cfc and its associated components were built to handle.

Manager.cfc is part of the **com.sebtools package** which is open source and free for any use.