Introduction to Relation Fields

Posted At : July 26, 2007 12:58 PM | Posted By : Steve
Related Categories: DataMgr

I added relation fields to DataMgr in version 2.0. Upon receiving a question about them by email, I noticed that I haven't ever written a good introduction about how to use them (though I have mentioned them several times).

What is DataMgr?

A little review, in case you aren't familiar with DataMgr. It is a data-access layer which handles CRUD functionality as well as Active Schema (if you choose) without generating any code. For more information read the **DataMgr 2.1 announcement**.

Instantiating DataMgr

The examples assume that you have DataMgr instantiated into Application scope. To do this, you can use the following code:

```
<cfif NOT StructKeyExists(Application,"DataMgr")>
   <cfset Application.DataMgr = CreateObject("component","DataMgr").init(mydsn,"MSSQL")>
</cfif>
```

This instantiates DataMgr against the datasource indicated in the "mydsn" variable using a SQL Server database. Other databases are available as well (currently Access, MySQL, PostGreSQL, SQL Server).

What is a Relation Field?

A relation field acts as though it is a field in a table even though it does not exist in that table in the database. Instead, it references another field and DataMgr makes it act like it is a real field.

For example, a products table:

Products
ProductID
CategoryID
ProductName
Price

is related to a categories table:

Categories
CategoryID
CategoryName

If I always want to know the category name for any product, I can create a relation field to hold that value in a column named "Category".

To do that, I need to define the relation field with DataMgr (code examples assume DataMgr is instantiated into Application.DataMgr):

```
<cfscript>
sRelation = StructNew();
sRelation["type"] = "label";
sRelation["table"] = "Categories";
sRelation["field"] = "CategoryName";
```

```
sRelation["join-field"] = "CategoryID";
Application.DataMgr.setColumn(
  tablename="Products",
  columnname="Category",
  Relation=sRelation
);
</cfscript>
```

I am passing a Relation structure to the setColumn method of DataMgr. This method can be used to ensure the existence of a column in a table, to set a special behavior on a column, or to create a relation field (as in this example). The keys that I am using in this example are as follows:

- type: The type of relation field (this example is a "label" relation field, but DataMgr has other types as well).
- table: The table from which DataMgr should get the value for the relation field.
- field: The field from which DataMgr should get the value for the relation field.
- join-field: The name of the field that should match from each table. The join-field key is really a short-cut for the following keys (used when the value of each matches):
    - join-field-local: The name of the field in the main table ("products" in this example) whose value should match that of the join-field-remote field in the related table ("categories" in this example).
    - join-fiel-remote: The name of the field in the remote table ("categories" in this example) whose value should match that of the join-field-local field in the local table ("products" in this example).

Now when I get the records from the "Products" table, DataMgr will also return a "Category" column with the value of the CategoryName field for the product's category.

```
<cfset qProducts = Application.DataMgr.getRecords("products")>
```

I can also filter by the category name if I want. So, if I want only products for the category named "Appliances":

```
<cfset sData = StructNew()>
<cfset sData["Category"] = "Appliances">
<cfset qProducts = Application.DataMgr.getProducts("Products",sData)>
```

I could even pass "Category" into DataMgr when saving a product to give the product the appropriate CategoryID for that category. I can also include (or remove) the Category field from the "fieldlist" argument of getRecords() to indicate whether it should be included in the result.

All of this behavior works as though the "Category" column were part of the "Products" table without that column actually existing in the table.

The above example is of a relation type of "label". Other relation field types work similarly. The types of relation fields currently available (as of DataMgr 2.1) are:

- Aggregates (avg,count,max,min,sum)
- Labels (label)
- Concatinations (concat)
- Lists (list)
- Custom (custom)

The DataMgr documentation covers the other types of relation fields in detail (as well as more expanded options for their use). Note that as of DataMgr 2.1, relation fields can not only get data from other fields but from other relation fields as well (excepting "list" relation fields).

If you have any questions about how to use relation fields, please **let me know**.

DataMgr is open source and free for any use. It can be **downloaded from RIAForge**.