Custom Relations Part III: Cascading

Posted At : December 28, 2007 2:00 PM | Posted By : Steve
Related Categories: DataMgr

Last time I covered how to use custom relations to add a field composed of complicated SQL. The real problem, however, is to reference that field from another table. Fortunately, this problem is easy to solve.

Using a custom relation field, my friend was able to create a "HomePhone" field using a SQL statement executed in a subquery.

He also needed to display the home phone in a list of attendees for a given class. The attendees table has a "MemberID" field and a "ClassID" field. Now that the "HomePhone" field is effectively set up in the "Members" table, it is a simple matter to retrieve it from the "Attendees" table using a normal label relation field:

```
<tables>
<table name="Attendees">
<field ColumnName="MemberID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
<field ColumnName="MemberID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
...
<field ColumnName="HomePhone">
<relation
type="label"
table="Members"
field="HomePhone"
join-field="MemberID"
/>
</field>
</table>
</tables>
```

This could also be done using setColumn:

```
<cfset sRelation = StructNew()>
<cfset sRelation["type"] = "label">
<cfset sRelation["table"] = "Members">
<cfset sRelation["field"] = "HomePhone">
<cfset sRelation["join-field"] = "MemberID">

<cfset Application.DataMgr.setColumn(tablename="Attendees",ColumnName="HomePhone",Relation=sRelation)>
```

Now querying the Attendees table will return the "HomePhone" field from the "Members" table, even though that field was a custom relation field with hand-written SQL.

As you can see custom relation fields allow you to dramatically enhance the built-in functionality in DataMgr and keep your code DRY in a way that would be difficult to do without using DataMgr.

DataMgr is open source and free for any use.