DataMgr 2.1 Beta 2 and Cascading Relation Fields

Posted At : June 12, 2007 12:55 PM | Posted By : Steve
Related Categories: DataMgr

The second (and hopefully final) beta of DataMgr 2.1 is ready for download.

This release has one new feature - the getVersion() method which will return a string indicating the version of DataMgr (so, "DataMgr 2.1 Beta 2" for this release).

Other than that, this release fixes a few bugs (including ones which will help add Derby support soon).

I think I haven't explained the new "Cascading Relation Fields" feature well yet, so I thought that I would use an example from a recent discussion I had about DataMgr.

Cascading Relation Fields

A user has the following database structure:

```
<tables>
 <table name="users">
  <field ColumnName="userId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
  <field ColumnName="userName" CF_DataType="CF_SQL_VARCHAR" Length="255" />
 </table>
 <table name="users2projects">
  <field ColumnName="userId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
  <field ColumnName="projectId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
 </table>
 <table name="projects">
  <field ColumnName="projectId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
  <field ColumnName="projectName" CF_DataType="CF_SQL_VARCHAR" Length="255" />
 </table>
 <table name="teams">
  <field ColumnName="teamId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
  <field ColumnName="teamName" CF_DataType="CF_SQL_VARCHAR" Length="255" />
 </table>
</tables>
```

Each simple PrimaryKey field is also Increment="true", but I removed that for brevity.

In English, each user can belong to one or more projects and each project is part of one (and only one) team.

The challenge here is the need to identify the team names for each user. This is something that in DataMgr 2.0 would have to be handled outside of DataMgr, but can be handled easily in DataMgr 2.1.

First thing is to add a label to the projects table that shows the team name:

```
<table name="projects">
 <field ColumnName="projectId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
 <field ColumnName="projectName" CF_DataType="CF_SQL_VARCHAR" Length="255" />
 <field ColumnName="teamName">
  <relation type="label" table="teams" join-field="teamId" field="teamName" />
 </field>
</table>
```

Now getRecords("projects") will include a column named "teamName" that will have the value from the teamName field where the teamId field values match. (this is the same as in DataMgr 2.0).

We can also get a list of projects for the users table:

```
<table name="users">
 <field ColumnName="userId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
```

```
  <field ColumnName="userName" CF_DataType="CF_SQL_VARCHAR" Length="255" />
  <field ColumnName="projects">
    <relation
      type="list"
      table="projects"
      join-table="users2projects"
      join-field="projectId"
      field="projectId"
    />
  </field>
</table>
```

Now getRecords("users") will include a column named "projects" with a comma delimited list of the projectid values for each user. (still possible for DataMgr 2.0)

It would be just as easy to get the projectName instead:

```
<table name="users">
  <field ColumnName="userId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
  <field ColumnName="userName" CF_DataType="CF_SQL_VARCHAR" Length="255" />
  <field ColumnName="projectnames">
    <relation
      type="list"
      table="projects"
      join-table="users2projects"
      join-field="projectId"
      field="projectName"
    />
  </field>
</table>
```

Similarly, getting the teamId from the projects table is easy (and possible in DataMgr 2.0):

```
<table name="users">
  <field ColumnName="userId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
  <field ColumnName="userName" CF_DataType="CF_SQL_VARCHAR" Length="255" />
  <field ColumnName="teams">
    <relation
      type="list"
      table="projects"
      join-table="users2projects"
      join-field="projectId"
      field="teamId"
    />
  </field>
</table>
```

This is all possible in DataMgr 2.0. What isn't possible, however, is to get the name of the teams for the users. This is because DataMgr 2.0 relation fields can point to real fields in the related table, but not to other relation fields. This changes in DataMgr 2.1, allowing the following:

```
<table name="users">
  <field ColumnName="userId" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
  <field ColumnName="userName" CF_DataType="CF_SQL_VARCHAR" Length="255" />
  <field ColumnName="teamnames">
    <relation
      type="list"
      table="projects"
      join-table="users2projects"
      join-field="projectId"
     field="teamName"
    />
```

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

This means that DataMgr 2.1 easily solves the problem at hand by pointing one relation field at another relation field.

It is important to note, however, that cascading relation fields cannot accurately retrieve values from relation fields of type="list" (It wouldn't have worked if each project could have more than one team).

All of the above examples assumed the user of Active Schema. This is all still possible, however, even if you are having DataMgr use database introspection instead of Active Schema (if you aren't using XML like the above, then DataMgr is using database introspection).

Without Active Schema

In order to do the above tasks without Active Schema, you just need to add the relation fields via the setColumn() method. So, for the first example, use the following code;

```
<cfscript>
sRelation = StructNew();
sRelation["type"]="label";
sRelation["table"]="teams";
sRelation["join-field"]="teamId";
sRelation["field"]="teamName";
DataMgr.setColumn(tablename="projects",columnname="teamName",relation=sRelation);
</cfscript>
```

The other examples will work in the same way. Note that since I use dashes "-" in my attribute names in the XML, you can't always use dot syntax to set those keys of your structure - requiring the use of bracket notation as above.

Cascading Relation Fields are a really powerful feature. They can even be using in combination with custom relation fields to make them more powerful.

All of this is available in **DataMgr 2.1 Beta 2** (which should be pretty stable).