Create Tables with Data

Posted At : October 30, 2006 2:00 PM | Posted By : Steve
Related Categories: DataMgr

I've written in the past about using DataMgr to make sure that the tables and columns you need exist. I have been asked a few times this week about making sure that those newly created tables are automatically loaded with data.

If you don't know how to have DataMgr create the tables and columns you need, you can watch the **"Synchronize Database Structure" presentation** now.

To review, in order to have DataMgr create tables and columns in DataMgr, pass XML to the loadXML() method of DataMgr (you can view the **CFC doc** for syntax).

For example, the following XML:

```
<tables>
   <table name="Examples">
      <field ColumnName="SampleID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true"
Increment="true" />
      <field ColumnName="SampleName" CF_DataType="CF_SQL_VARCHAR" Length="50" />
      <field ColumnName="SampleDescription" CF_DataType="CF_SQL_LONGVARCHAR" />
      <field ColumnName="MyDate" CF_DataType="CF_SQL_DATE" />
   </table>
</tables>
```

This would create the "Examples" table with the fields as detailed above.

In order to create this table with two rows of data, you could use the following XML.

```
<tables>
   <table name="Examples">
      <field ColumnName="SampleID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true"
Increment="true" />
      <field ColumnName="SampleName" CF_DataType="CF_SQL_VARCHAR" Length="50" />
      <field ColumnName="SampleDescription" CF_DataType="CF_SQL_LONGVARCHAR" />
      <field ColumnName="MyDate" CF_DataType="CF_SQL_DATE" />
   </table>
   <data table="Examples">
      <row SampleName="Bob" SampleDescription="This is the description of Bob." />
      <row SampleName="Coca-Cola" SampleDescription="Taste tests show this isn't as popular as Pepsi,
but ads make people think it tastes better." />
   </data>
</tables>
```

DataMgr will also let you use column names that are not normally valid in most databases. For example, you could have a column named "Sample Name" (with a space). The above XML format will not, of course, support that column name. DataMgr can support it with a slightly more verbose syntax.

```
<tables>
   <table name="Examples">
      <field ColumnName="SampleID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true"
Increment="true" />
      <field ColumnName="Sample Name" CF_DataType="CF_SQL_VARCHAR" Length="50" />
      <field ColumnName="SampleDescription" CF_DataType="CF_SQL_LONGVARCHAR" />
      <field ColumnName="MyDate" CF_DataType="CF_SQL_DATE" />
   </table>
   <data table="Examples">
      <row>
         <field name="Sample Name" value="Bob" />
         <field name="SampleDescription" value="This is the description of Bob." />
```

```
      </row>
      <row>
         <field name="Sample Name" value="Coca-Cola" />
         <field name="SampleDescription" value="Taste tests show this isn't as popular as Pepsi, but ads
make people think it tastes better." />
      </row>
   </data>
</tables>
```

It is also possible that you might have some data that you want to ensure will always be available in the table (even if it isn't empty). To do that, you can use the "permanentRows" attribute:

```
<tables>
   <table name="categories">
      <field ColumnName="CatID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" Increment="true"
/>
      <field ColumnName="CatName" CF_DataType="CF_SQL_VARCHAR" Length="50" />
   </table>
   <data table="categories" permanentRows="true">
      <row CatName="Coats" />
      <row CatName="Shoes" />
   </data>
</tables>
```

DataMgr will use this data to ensure that the categories of "Coats" and "Shoes" always exist (but without adding duplicates).

You may also want to include relational data (products for Coats and Shoes, for example):

```
<tables>
   <table name="categories">
      <field ColumnName="CatID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" Increment="true"
/>
      <field ColumnName="CatName" CF_DataType="CF_SQL_VARCHAR" Length="50" />
   </table>
   <table name="products">
       <field ColumnName="Prod" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" Increment="true"
/>
       <field ColumnName="CatID" CF_DataType="CF_SQL_INTEGER" />
      <field ColumnName="ProdName" CF_DataType="CF_SQL_VARCHAR" Length="50" />
   </table>
   <data table="categories" permanentRows="true">
      <row CatName="Coats" />
      <row CatName="Shoes" />
   </data>
   <data table="products">
      <row ProdName="Air Jordan's">
         <field name="CatID" reltable="categories" relfield="CatID" CatName="Shoes" />
      </row>
      <row ProdName="The Ambassador">
         <field name="CatID" reltable="categories" relfield="CatID" CatName="Coats" />
      </row>
   </data>
</tables>
```

In the above example, I related the data by one field: "CatName", but I could have used multiple fields if I wanted. If the CatName field was an invalid field name, like "Cat Name", the above syntax would fail. So, the final example covers that unfortunate situation:

```
<tables>
   <table name="categories">
      <field ColumnName="CatID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" Increment="true"
```

```
/>
      <field ColumnName="Cat Name" CF_DataType="CF_SQL_VARCHAR" Length="50" />
   </table>
   <table name="products">
       <field ColumnName="Prod" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" Increment="true"
/>
       <field ColumnName="CatID" CF_DataType="CF_SQL_INTEGER" />
      <field ColumnName="ProdName" CF_DataType="CF_SQL_VARCHAR" Length="50" />
   </table>
   <data table="categories">
      <row>
         <field name="Cat Name" value="Coats" />
      </row>
      <row>
         <field name="Cat Name" value="Shoes" />
      </row>
   </data>
   <data table="products">
      <row ProdName="Air Jordan's">
         <field name="CatID" reltable="categories" relfield="CatID">
            <relfield name="Cat Name" value="Shoes" />
         </field>
      </row>
      <row ProdName="The Ambassador">
         <field name="CatID" reltable="categories" relfield="CatID">
            <relfield name="Cat Name" value="Coats" />
         </field>
      </row>
   </data>
</tables>
```

Keep in mind, that the first example will cover most situations. The options are available, however, to have DataMgr ensure that the table structure and data that you need for your application are available.

I know that many of my recent entries have been covering features new to the upcoming **DataMgr 2** . This isn't one of them. You can use this functionality right now.

Feel free to **download DataMgr** as use it for any purpose.