DataMgr Rethinks the Data Access Layer

Posted At : March 22, 2007 9:55 PM | Posted By : Steve
Related Categories: DataMgr

Prolific blogger, Peter Bell, just posted an entry about "**Rethinking the Data Access Layer**" wherein he mentions DataMgr and ORM solutions and lists his requirements for a database abstraction tool.

I couldn't resist responding with how DataMgr meets those requirements.

Speaking in Tongues

DataMgr provides one API that works regardless of your database. This currently includes MS SQL, Access, MySQL, and PostGreSQL. Others should be easy to add.

Get Associations

Peter mentions the desire to have CategoryService.getAssociatedProducts(CategoryID). While DataMgr won't do this directly (it is a database astraction layer, not an ORM), it does provide capabilities that are helpful in this regard.

Assuming you use XML to define your relationship (you could use setColumn() instead):

```
<table name="categories">
  <field ColumnName="CategoryID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true"  />
  <field ColumnName="CategoryName" CF_DataType="CF_SQL_VARCHAR" Length="80" />
  <field ColumnName="Products">
    <relation table="products" type="list" field="ProductID" join-field="CategoryID" />
  </field>
</table>
```

Then getRecords("categories") would return a field called "products" with a comma-delimited list of ProductID values from the products table for the category in each row.

Cascading Deletes

Taking the example above, the relationship element has an attribute of "onDelete" with a default value of ignore. In the above example, the deletion of a category will have no effect on the products table.

You can change the element to this:

```
<relation table="products" type="list" field="ProductID" join-field="CategoryID" onDelete="Cascade" />
```

In which case all related products would be deleted.

Similarly, you could do this instead:

```
<relation table="products" type="list" field="ProductID" join-field="CategoryID" onDelete="Error"  />
```

In which case DataMgr would throw an error if you tried to delete a category that had related products.

The advantage of this over database constraints is that this works with logical deletes as well.

Manage Joins

I'm not honestly sure I understood what Peter want here, but DataMgr handles several types of joins. The list type (mentioned) earlier can also be used in saving for a many to many relationship (a join-table attribute is needed).

It also has a label relation field:

```
<table name="categories">
  <field ColumnName="ProductID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
  <field ColumnName="ProductName" CF_DataType="CF_SQL_VARCHAR" Length="80" />
  <field ColumnName="Category">
```

```
    <relation table="categories" type="label" field="CategoryName" join-field="CategoryID" />
  </field>
</table>
```

Using this XML, getRecords("products") would return a recordset with a "Category" field that would show the CategoryName of the category for the product in each row.

Magic Fields

In DataMgr, these are "Special" fields.

```
<table name="categories">
  <field ColumnName="ProductID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" />
  <field ColumnName="ProductName" CF_DataType="CF_SQL_VARCHAR" Length="80" />
  <field ColumnName="Category">
    <relation table="categories" type="label" field="CategoryName" join-field="CategoryID" />
  </field>
   <field ColumnName="DateAdded" CF_DataType="CF_SQL_DATE" Special="CreationDate" />
</table>
```

They include the ability to store the date a row was added or the date it was last updated as well as the ability to automatically handle manual record sorting as well as logical deletions.

They don't have the ability to handle other types of automated data that Peter mentioned, but I would be open to adding more Special types in future versions.

Everything I Need - and Nothing More

I think this is where DataMgr shines. It doesn't enforce an OO paradigm (although it could be used in one - one small ORM has already been built on top of DataMgr). The **performance overhead of DataMgr** is similarly minimal.

Aggregate Subqueries

Peter mentioned this need in the comments. DataMgr handles this as well.

The best examples are found in the **aggregates page** of the demonstration site.

Plenty of other examples are available as well. Check out the **DataMgr demonstration site** , or download **CodeCop** for a working application running on DataMgr.