MVC Heresy with SQLTags

by Steve A. Olson
12/19/2003

<!-- co: is this a first sentence or a proposed description for the front page? I like the next graf better as an intro. Ca -->
SQLTags (http://sqltags.org) is an open source project that builds a custom JSP tag for every table in your relational database.

Heresy: A deliberate deviation from the orthodox doctrine. SQLTags is, indeed, a deliberate deviation from the orthodoxy of the Model-View-Control (MVC) design pattern; therefore, SQLTags is heresy, but is it necessarily "bad?" In this article, I will introduce the SQLTags toolkit and when it can be successfully employed.

<!-- co: when you do the html version, just hot-link the text "Beyond MVC" or "Struts mailing list". Ca -->

Notwithstanding the recent criticism by N. Alex Rupp in "Beyond MVC" (http://today.java.net/pub/a/today/2003/12/11/mvc.html), the MVC design pattern is, currently, the gold standard for highly complex web applications developed within the Servlet environment. Just subscribe to any of the Struts mailing lists (http://jakarta.apache.org/struts/using.html#Lists) to witness how vibrant this community is. However, as stated on the Struts main page (http://jakarta.apache.org/struts/index.html) and paraphrased here, Struts and, by extension, MVC are not the best choice for every project. The Struts community asserts that only "very simply applications, with a handful of pages," are candidates for non-MVC implementations, and with this I agree; however, the definition of "simple" has now been altered by the introduction of SQLTags.

SQLTags implements a very simple concept: Database schema as Tag Library. Each table is a JSP tag—a tag that "knows" all of the columns; a tag that "knows" how to automatically assign HTTP request parameters to those columns; a tag that "knows" its own primary key and uses it in update and delete statements; a tag that "knows" all of its imported and exported foreign keys and, more importantly, how to use them to join the related tables; a tag that "knows" how to write insert, update, delete, and select statements; a tag the "knows" how to process several rows at once; a tag that "knows" how to page its output; in short, a tag that "knows" the underlying table as well as the Servlet environment.

This simple concept extends the realm of "very simple" to what was previously moderately complex, data-driven web applications.

<!-- co: I think the above material is an "introduction". This header should be "SQLTags Features" or similar. Ca -->

Introduction

SQLTags is a free, open source, object-relational mapping toolkit that provides a new and innovative approach to data-driven web applications development. At the SQLTags core is a generator that builds a JavaBean and JSP tag for each table within a given JDBC schema. The SQLTags Generator packages everything (including built-in tags and support classes) into a single Java Archive or "jar" file for easy deployment into any Java 1.4/Servlet 2.3/JSP 1.2 (or later) J2EE Application Server. Best of all, SQLTags is quick and easy to learn and understand.

The SQLTags generator accepts input that identifies a valid (and available) JDBC Driver, JDBC URL, user name, password, schema name, and other options which are used to reverse-engineer the given database schema into the corresponding "jar" file. The resulting "jar" file can be deployed into any complaint Application Server. On the target server, the "deployment descriptor" or `/WEB-INF/web.xml` file is updated to include "context parameters" that tell SQLTags how to connect to the same or an identical schema. Once the generated "jar" file is successfully deployed into the target Application Server, JSP pages can use the generated (and built-in) tags contained within the "jar" file to implement the desired functionality. Whenever the database is changed, simply use the SQLTags Generator to "re-generate" the "jar" file.

Additionally, SQLTags works well with other Java and JSP industry standards like: **JavaBeans**—each SQLTags tag is a JavaBean, **JSTL**—SQLTags beans are easily accessible within JSTL's Expression Language (EL), **JSP 2.0**—SQLTags tags are easily accessible using the "`${`" syntax , and **Struts**—SQLTags tags work well with the Struts "html" tags. SQLTags is both application server and database independent.

<!-- co: a lot of the material in this section is list-like and could be rendered as lists when you do the html version. I'm thinking particularly of the steps in the first graf and the attributes in the "Attributes" graf. Ca -->

Functional Overview

There are three steps required to use a SQLTags generated "jar" file: 1- run the generator to create the "jar" file, 2- deploy the generated "jar" file into the target web application, and 3- author JSP pages using the tags within the target web application. This section will provide an overview of the functionality available to the JSP developer when using a generated "jar" file.

Built-in tags. SQLTags adds a number of built-in tags to the generated "jar" file. These can be broken down into three main categories: **paging support**: `first`, `last`, `next`, and `previous`; **database support**: `connection`, `commit`, `rollback`, `exception`, `cursor`, `statement`, `fetch`, `where`, and `orderby`; and **authorization**: `authorize`, `allow`, and `deny`.

The most commonly used built-in tags are `connection`, `exception`, `where`, and `fetch`. The `connection` tag is required in every page that needs a connection to the database (almost every page). Connections can come from the built-in SQLTags connection pool defined within the "deployment descriptor" or from a container managed DataSource. The `exception` tag outputs JDBC error messages only when things go awry with database operations. The `where` and `fetch` tags are provided as a mechanism to more easily compose complex where clauses and explicitly iterate through the ResultSets. The `cursor`, `statement`, and `fetch` tags are available to provide functionality similar to the JSTL "sql" tag library. (At some point in the future, perhaps, SQLTags will integrate with the JSTL sql tag library.)

Generated tags. As noted above, each table is represented as a tag and each of those tags have common attributes and methods as well as unique, per-column attributes and methods. The common attributes available to all generated tags are: `id`, `tableAlias`, `buttonName`, `columns`, `displaySize`, `foreignKey`, `handlerClass`, `hasFetch`, `maxRows`, `operation`, `orderBy`, `parentName`, `paging`, `preInsertSQL`, `preUpdateSQL`, `properties`, `startRowParameter`, `where`, and `from`. Additionally, for each column, *COL*, there are four tag attributes: *COL*, *COL*_FORMAT, *COL*_SELECT, and *COL*_BIND.

Attributes. The "*COL*" attributes allow the value of the column to be explicitly set, *COL*_FORMAT provides a mechanism for specifying the display format for date and number columns, *COL*_SELECT provides a mechanism to override the select-list item for the column, and a *COL*_BIND provides a mechanism to override the bind expression used in a PreparedStatement (every SQLTags SQL statement uses the PreparedStatement). Some of the more commonly used attributes are: **id**—defines the snippet variable that can be used within JSP snippets or EL expressions; **operation**—defines an explicit operation to be carried out on the table (insert, update, delete, deleteChildren, or select); **buttonName**—defines a request parameter that contains the operation; **foreignKey** combined with **parentName—**used to nest tags in a parent-child configuration; **paging="true"** and **displaySize**—define paging behavior; **properties="true"**—tells the tag to scan the HTTP Request for parameter names that match column names; **where**—defines the where condition and causes the tag body to iterate once for each row that satisfies the given condition.

Methods. Some of the more useful methods available to all generated tags are: `insert()`, `update()`, `delete()`, and `delete(where)`. Additionally, because each table tag is a JavaBean, each column, *COL*, has `set`*`COL`*`()` and `get`*`COL`*`()` methods and each foreign key, FK, has a `getFK()` method—again very useful for EL expressions.

Foreign Keys. SQLTags makes extensive use of foreign keys. Take a look at any Entity-Relationship Diagram, the glue that holds any database together is the foreign keys. Indeed, in my experience the majority of the navigation within a data-driven web application centers on the foreign keys—either through parent table lookups or through parent-child nesting. For example, given an Employee, what Department is she assigned to; or given a Department who are the Employees contained with it; or given an Invoice, list the Invoice Line Items and related Product Names. These questions are all answered by foreign keys.

<!-- co: don't number lines in code listings for us. See the stylesheet for info on our <pre><code>...</code></pre> formatting for code listsings. Ca -->

SQLTags provides powerful ways to leverage foreign keys using "declarative access." For example, imagine two tables `EMP` and `DEPT` with a foreign key, `FK_DEPTNO`, defined within `EMP` referencing `DEPT` (see Entity-Relationship Diagram, below). When SQLTags generates our hypothetical `EMP` JavaBean, it would contain a `getFK_DEPTNO()` method that returns a `DEPT` JavaBean. Consider the following JSP 2.0 snippet.

```
1 <x:emp id=”e” where=”order by ENAME”>
2    ${e.ENAME} belongs to Department ${e.FK_DEPTNO.DNAME}!<br/>
3 </x:emp>
```

As expected, the column values are easily accessible using EL (beginning of line 2) based on standard JavaBean syntax; however and perhaps not expected, the foreign keys are also available (end of line 2) in exactly the same way! Need the Department name, DNAME, for a given Employee? Simply use the foreign key FK_DEPTNO as the getter for the related Department. Wow!

SQLTags also provides mechanism for nesting children records within their parents, again, using foreign keys. Consider the following JSP 2.0 snippet.

```
<x:dept id=”d” where=”order by DNAME”>
   <p>${d.DNAME} has the following employees:
   <x:emp id=”e” foreignKey=”FK_DEPTNO” parentName=”d”>
      ${e.ENAME}
   </x:emp>
   </p>
</x:dept>
```

Again, note how easily the list of Employees for a given Department can be obtained simply by knowing the foreign key name and the "id" of the DEPT tag.

Finally, consider how simple it is for a programmer (or even a designer, heaven forbid) to look at an Entity-Relationship Diagram and start connecting the dots.

Running the SQLTags Generator

<!-- co: I'm not a fan of this kind of self-deprecation – if your article is worth writing, it's worth reading. Ca -->

Well, if you are still reading, perhaps you would like to try SQLTags for yourself. Well, read on and I'll try to walk you through it.

Download the `sqltags-1-0-6.war` (or latest version) web application file from the SQLTags web site at http://sqltags.org/downloads.jsp and deploy it into your favorite J2EE Application Server. Note: SQLTags requires Java 1.4, Servlet 2.3 or later, JSP 1.2 or later and a suitable JDBC driver for your specific database. The SQLTags web application comes with pre-installed JDBC drivers for Oracle, MySQL, and Sybase. The JDBC Driver for Microsoft SQL Server is also included but is in the `WEB-INF/extras` directory but must be manually moved into `WEB-INF/lib` in order to access a SQL Server database.

Note: *the "`javac`" executable must be accessible via the PATH variable (by the application server) in order to compile the generated JavaBean classes*.

To deploy the `sqltags-1-0-6.war` file into the Tomcat Application Server, for example, simply copy the "war" file into the `CATALINA_HOME/webapps` directory, restart the server, and access the `/sqltags-1-0-6/` URI on your deployment host using your browser. For example, if you were running tomcat on your local workstation on port 8080 you would use the following URL: `http://localhost:8080/sqltags-1-0-6/` to run the SQLTags web application. The SQLTags web application contains information and samples for running the generator, deploying the "jar" file, and building JSP pages.

<!-- co: if there are fields to fill in, it might be helpful to provide a screenshot as a figure so readers will know they're looking at the right thing when they try it. That said, we have a 500-pixel horizontal limit for images, so resize your browser window nice and narrow before taking the shot. Ca -->

To execute the generator from your browser, simply click on the "***run …"*** link near the top of the page (using the above example, the URL would be `http://localhost:8080/sqltags-1-0-6/generate.jsp`). Fill-in all the fields and click on the "Generate!" button. That's it, really!

The generate screen contains detailed descriptions for each input field. Basically, you'll need to specify the JDBC Driver and URL, the database user name and password, the JDBC Schema name, a Java package for the generated Java code, a "jar" file name, and flags to control inclusion of the built-in tags and source code. Once the generator completes, you should see output in the browser window showing that each table was processed. The generated "jar" file is saved into the `/WEB-INF/tmp` directory using the file name specified in the "JAR Filename" input field. (Note: future releases of SQLTags will likely place the generated "jar" file in the `/tmp` directory under the web application.)

Note: SQLTags will not generate foreign key references to tables that are either not contained within the generated "jar" file due to wildcard exclusion or do not have a primary key.

Deploying a Generated Tag Library

Deployment of the generated "jar" file is accomplished in two steps: 1- copy the generated "jar" file into the `WEB-INF/lib` directory of the target web application and 2- modify the `WEB-INF/web.xml` (deployment descriptor) to define the SQLTags "context parameters." You'll likely need to restart the web application for the changes to take effect.

Context Parameters. Context parameters are added to the `WEB-INF/web.xml` deployment descriptor immediately after the `<web-app>` tag using the following syntax.

```
<context-param>
   <param-name>SQLTags.useCM</param-name>
   <param-value>true</param-value>
</context-param>
```

<!-- co: obviously this calls for html formatting as a list, or possibly a table. Ca -->

The list of SQLTags context parameters:

`SQLTags.useCM:` Valid values are "true" or "false"; default value is "true"; defined: "use internal Connection Manager". When "false" SQLTags will use a DataSource and ignore the other SQLTags JDBC context parameters (listed below).

`SQLTags.dataSource:` Identifies the default dataSource to use by the `connection` tag when no dataSource attribute is supplied and `useCM` is "false"; default value is `jdbc/SQLTagsDS`. ***Note**: a DataSource is used when `useCM` is "false" or when a `connection` tag explicitly references a specific DataSource from the "`dataSource`" attribute*.

`SQLTags.bindStrings:` Should almost always be set of "false"; when "true" all database bindings are processed as strings, when "false" all numerical and date data types are processed as the correct native Java type. Greatly helps with compatibility among different databases.

The following parameters are only used when `SQLTags.useCM` is "true". When `useCM` is "false" the JDBC parameters are defined externally within a DataSource.

`SQLTags.databaseDriver:` The JDBC database driver as specified by JDBC vendor. Examples: `oracle.jdbc.OracleDriver`, `org.gjt.mm.mysql.Driver`.

`SQLTags.connectionUrl:` The JDBC connection URL as specified by JDBC vendor. Examples: `jdbc:mysql://localhost/scott?user=root`, `jdbc:oracle:thin:@localhost:1521:ORCL`.

`SQLTags.maxPoolSize:` Maximum number of JDBC connections for the connection pool.

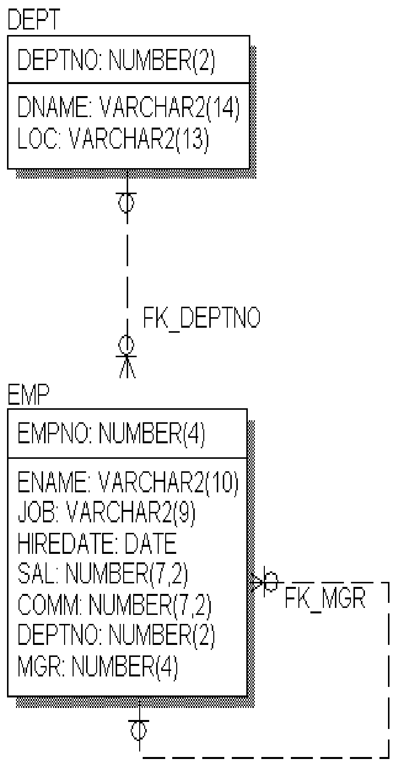
`SQLTags.poolSize:` Default size of the JDBC connection pool.

`SQLTags.userName:` Database user for JDBC connections.

`SQLTags.password:` Database user's password.

Authoring JSP Pages with SQLTags

SQLTags development starts with a well-defined data model. So, for the purposes of this article, I will refer to the following "classic" EMP-DEPT data model.



Sample 1. Listing all EMP data with related DEPT name.

In this first example, I will demonstrate a simple JSP page that displays a listing of all rows from the EMP table and their assigned DEPT.

```
<%@ taglib uri="demoTags.jar" prefix="sqltags" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
<head><title>JSP Page</title></head>
<body>
<%-- uses default SQLTags DataSource configuration from web.xml.  --%>
<sqltags:connection id="connect" >

    <sqltags:emp id="e" where="order by ename">
         <c:out value="${e.ENAME}"/> assigned to
         <c:out value="${e.FK_DEPTNO.DNAME}"/>
         <br>
```

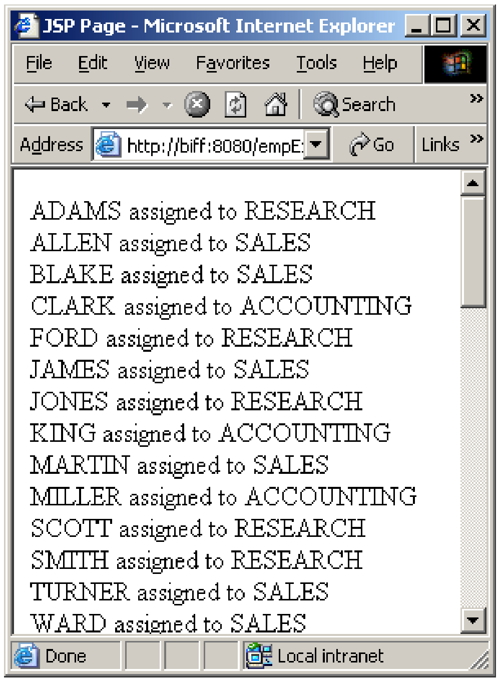
```
13     </sqltags:emp>
14
15 </sqltags:connection>
16 </body>
17 </html>
```

<!-- co: since we don't do line numbers, you'll need to rephrase this material to refer to the code by the calls that accomplish the stated functionality, e.g., "the <sqltags:connect> tag accesses the jdbc connection pool. This tag must wrap all other sqltags tags. Ca -->

Lines 01 and 02 introduce the two tag libraries used within the JSP page. Line 07 provides access to the SQLTags JDBC connection pool which must wrap all of the other SQLTags tags. Line 09 introduces the EMP tag and defines a "where clause" that matches all rows and orders the results by ENAME. Lines 10 uses the JSTL "out" tag to output the ENAME property from the EMP JavaBean (ENAME is a column from the EMP table). In line 11, the DNAME property from the related DEPT table is output for each EMP in the table.

<!-- co: for the benefit of the slow or incredulous reader, you should indicate where the data is coming from, ie, "assuming we have some suitable records in our database, viewing the above page..." ca -->

Viewing the above page from the browser would yield the following output:



```
ADAMS assigned to RESEARCH
ALLEN assigned to SALES
BLAKE assigned to SALES
CLARK assigned to ACCOUNTING
FORD assigned to RESEARCH
JAMES assigned to SALES
JONES assigned to RESEARCH
KING assigned to ACCOUNTING
MARTIN assigned to SALES
MILLER assigned to ACCOUNTING
SCOTT assigned to RESEARCH
SMITH assigned to RESEARCH
TURNER assigned to SALES
WARD assigned to SALES
```

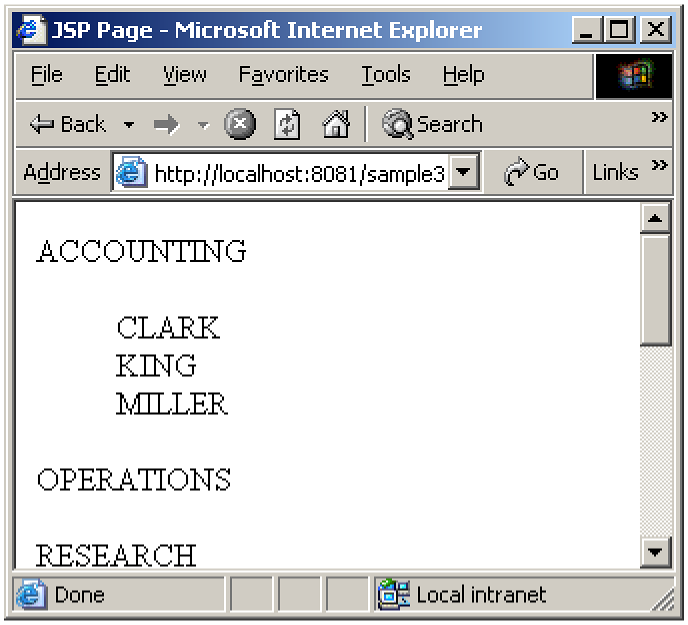
Sample 2. Parent-Child nesting of EMP within DEPT.

In the second example, I will demonstrate a simple JSP page that implements the parent-child configuration listing EMPs within each DEPT.

```
01 <%@ taglib uri="demoTags.jar" prefix="sqltags" %>
02 <%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
03 <html>
04 <head><title>JSP Page</title></head>
05 <body>
06 <%-- uses default SQLTags DataSource configuration from web.xml.
07   --%>
08 <sqltags:connection id="connect" >
09
10   <sqltags:dept id="d" where="order by DNAME">
11       <c:out value="${d.DNAME}" />
12       <BLOCKQUOTE>
13       <sqltags:emp id="e" foreignKey="FK_DEPTNO" parentName="d">
14         <c:out value="${e.ENAME}"/>
15         <br>
16       </sqltags:emp>
17       </BLOCKQUOTE>
18    </sqltags:dept>
19
20 </sqltags:connection>
21 </body>
22 </html>
```

Again, lines 01 and 02 introduce the two tag libraries used within the JSP page. Line 08 provides access to the SQLTags JDBC connection pool which must wrap all of the other SQLTags tags. Line 10 introduces the DEPT tag and defines a "where clause" that matches all rows and orders the results by DNAME. Line 11 outputs the department name and line 12 wraps the employees within each department within a "blockquote" to provide a nested look in the browser. Line 13 shows the mechanism used to nest the EMP tag within its related DEPT. The "foreignKey" attribute tells the EMP tag which foreign key to reference and the "parentName" attribute identifies the DEPT tag instance that defines the current DEPT. Line 14 simply outputs each employee name one per line.

Viewing the above page from the browser would yield the following output:


JSP Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search
Address http://localhost:8081/sample3
Go Links
Done
Local intranet

```
ACCOUNTING

      CLARK
      KING
      MILLER

OPERATIONS

RESEARCH
```

I've just scratched the surface of what is possible with SQLTags. There is a lot more functionality available within the tool set. Hopefully, you've seen enough to whet your interest to investigate further.

<!-- co: since I don't think we've signed on for a whole series, please rephrase this in a way that just suggests the advanced topics. Ca -->

Preview of Advanced Topics

In future articles, I hope to review some of the more advanced areas of SQLTags including: multi-row insert, update, and delete in HTML forms; deployment into some specific Application Servers; using Container Managed DataSources; using authentication and access control with `allow` and `deny` tags; using "handler" classes to easily deal with BLOB and CLOB column types, paging of results, etc.

<!-- co: this references section is unneccessary, since you'll have all the links in-lined in the html version (with their related text) anyways. Ca -->

References

SQLTags	http://sqltags.org/
Simpler Java	http://www.theserverside.com/resources/article.jsp?l=SimplerJava
Beyond MVC	http://today.java.net/pub/a/today/2003/12/11/mvc.html
Struts Mailing Lists	http://jakarta.apache.org/struts/using.html#Lists
Struts Home	http://jakarta.apache.org/struts/index.html
Practical JSTL, Part 1	http://today.java.net/pub/a/today/2003/10/07/jstl1.html
Practical JSTL, Part 2	http://today.java.net/pub/a/today/2003/11/27/jstl2.html

<!-- I'd like to see this merge with the "preview of advanced topics" as a summary that says what else is possible with sqltags and sells the whole idea of using sqltags for these kinds of web apps. Ca -->

Summary

MVC is a great design pattern, but SQLTags can be successfully employed when used to develop a data-driven web application that closely follows a well defined data model with limited processing logic. Basically, for data-in, data-out applications, SQLTags is just the ticket.

Steve A. Olson is a founder and Chairman of the Board of Applied Information Technologies, Inc. (http://ait-inc.com) and is currently serving the House and Senate within the Office of Legislative Information at the Congressional Research Service on Capital Hill.