<SQLTags:> Tag library (Pre Beta)

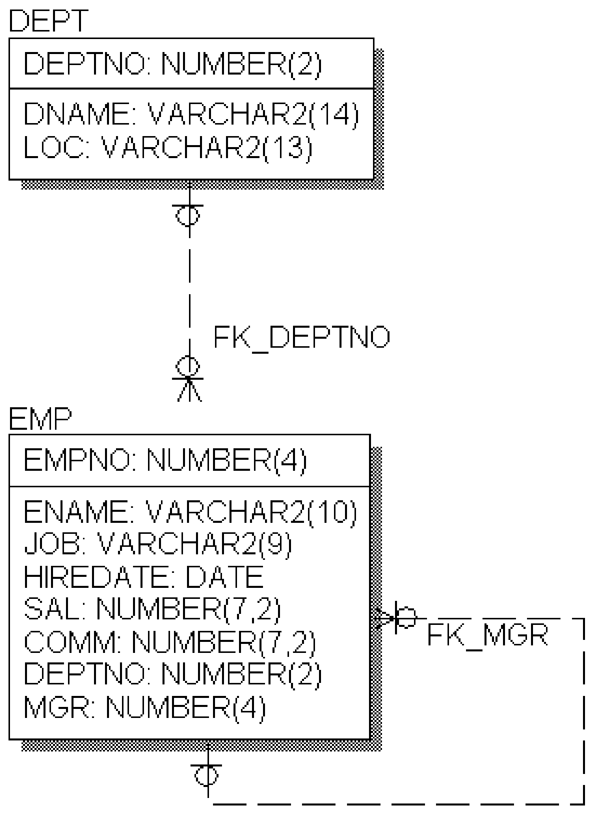
Version: 1.0

Table of Contents

Overview

<SQLTags:> is a new and innovative approach to accessing a database within JSP—a custom JSP tag for every table—and a set of built-in tags for complex queries and routine operations: cursor, where, fetch, first, last, next, previous, commit, and rollback to name a few. <SQLTags:> is a generator that creates a JSP tag for each table within a given database. So, if the database has an "EMP" and a "DEPT" table, then the JSP pages will use "emp" and "dept" tags to access the data. It's that simple!

The generated tags have bean-style access to all of the columns and imported (or parent) foreign keys in the table, they can use the exported (or children) foreign keys to iterate through rows from related tables, they contain methods for querying and modifying data (insert, update, delete, select, fetch), they automatically link data from the "HTTP Request Object" to the corresponding columns in the table, and much more.



This figure, an "Entity-Relationship Diagram", provides all of the necessary information required to build a <SQLTags:> page. ***Note**: this figure will serve as the basis for most of the examples in this text.*

Each table, column, and foreign key is documented in the diagram.

The snipped below shows how to output the names of every employee in department 5.

```
<sqltags:dept id="d"  DEPTNO="5">
   <sqltags:emp id="e"
         parentName="d"
         foreignKey="FK_DEPTNO">
     <%= e.getENAME() %> <BR>
   </sqltags:emp>
</sqltags:dept>
```

Requirements

JSP Requirements

This custom tag library requires a servlet container that supports the JavaServer Pages Specification, version 1.2.

JDK Requirements

This tag generator requires JDK version 1.2 or later in order to compile the generated Java classes.

Configuration

The basic process for using <SQLTags:> is as follows: run GeneratorGUI to create a custom tag library specific to your database, create a Properties file that describes the JDBC connection, and use the generated tags in your JSP pages. Whenever changes are made to the database structure, simply re-run GeneratorGUI to update the generated tag library.

Follow these steps to configure your web application to use the a generated tag library:

1. Put the sqltags Jar file into the WEB-INF/lib directory.
2. Add the sqltags Jar file to your CLASSPATH.
3. Make sure that “javac” is accessible from the command-line via your PATH so the generator can compile the generated Java files.
4. Run “GeneratorGUI” to create a new custom Jar file for your database. It will contain all of the generated table-tag classes and a “taglib.tld”.

   ```
   java com.aitworks.sqltags.generator.GeneratorGUI
   ```

5. Create a run-time properties file with at least the following properties defined:

   ```
   # Sample <SQLTags:> properties file
   connectionTimeToLiveSeconds=240
   connectionUrl=jdbc:oracle:thin:@localhost:1521:ORCL
   maxPoolSize=2
   minPoolSize=1
   poolSize=1
   password=tiger
   userName=scott
   databaseDriver=oracle.jdbc.OracleDriver
   ```

6. Add a <taglib> element to your web application deployment descriptor in the /WEB-INF/web.xml file as follows:

   ```
   <taglib>
     <taglib-uri>/WEB-INF/lib/sqltags.jar</taglib-uri>
     <taglib-location>/WEB-INF/lib/sqltags.jar</taglib-location>
   </taglib>
   ```

7. To use the generated tags in your JSP pages, add the following directive at the top of each page:

   ```
   <%@page import="com.aitworks.sqltags.jsptags.*,
                   generatedPackage.* " %>
   <%@taglib uri="/WEB-INF/lib/sqltags.jar" prefix="sqltags" %>
   ```

Documentation

Simple Usage Example

Here is a JSP page that prints out information about all employees in the “EMP” table:

```
<%@page import="com.aitworks.sqltags.jsptags.*,
                generatedPackage.*" %>
<%@taglib uri="/WEB-INF/lib/sqltags.jar" prefix="sqltags" %>

<%-- open a database connection using properties file --%>
<sqltags:connection id="connect"
                    initSrc=”/WEB-INF/sqltags.properties”
                    name="init">

<%-- open a database query --%>
<table>
<sqltags:emp id="emp" where="order by ename">
  <%-- loop through the rows of your query --%>
    <tr>
      <td><%= emp.getEMPNO() %></td>
      <td><%= emp.getENAME() %></td>
      <td><%= emp.getFK_DEPTNO_PARENT().getDNAME() %></td>
    </tr>
</sqltags:emp>
</table>

<%-- close a database connection --%>
</sqltags:connection>
```

Running the <SQLTags:> GeneratorGUI

Executing the Graphical Interface

Make sure that “javac” is accessible from the command-line via your PATH so the generator can compile the Java classes during the generate process. Run “GeneratorGUI” to create a new custom Jar file specifically for your database. It will contain all of the generated table-tag classes, the <SQLTags:> standard tags, and a “taglib.tld”.

```
java com.aitworks.sqltags.generator.GeneratorGUI
```

[Screen shot]

Generator Properties

The generator uses the following properties to manage the generation process:

Property	Description
Connection string	jdbc:oracle:
User name	scott
Password	tiger
Schema	SCOTT
Database Driver	oracle.jdbc.OracleDriver
More …	Descriptions to come

Tags in Detail

Here are general descriptions of the tags included in the <SQLTags:> tag library. Some details, such as the possible properties of the connection, cursor, table, statement tags are not covered here. An enumeration of all the configurable aspects of these tags is in the Tag Reference section.

Initialization

Setting Properties

A standard Java “properties” file contains the configuration parameters for the <SQLTags:> environment. Below is an example properties file for connecting to “localhost” using the “scott” schema within the “ORCL” instance from the database demos.

```
# Sample <SQLTags:> properties file
connectionTimeToLiveSeconds=240
connectionUrl=jdbc:oracle:thin:@localhost:1521:ORCL
maxPoolSize=2
minPoolSize=1
poolSize=1
password=tiger
userName=scott
databaseDriver=oracle.jdbc.OracleDriver
```

Generated Tags

Each generated tag is specific to a table in the database. The “EMP” table as defined in the Figure 1 will be used as the basis for this discussion.

Accessors and Mutators

Each column in the table has accessors or “getters” and mutators or “setters” for each column in the table. For example, the <sqltags:emp/> tag would have the following methods available for use within the JSP page:

Column	
Accessors	**Mutators**
getENAME()	setENAME()
getJOB()	setJOB()
getHIREDATE()	setHIREDATE()
getSAL()	setSAL()
GetCOMM()	setCOMM()
getDEPTNO()	setDEPTNO()
getMGR()	setMGR()

The mutators are also available to tag using tag attributes. For example, in the snippet below, prior to the “doStart” call in the tag, the setENAME(“jack frost”) and setJOB(“singer”) methods would be invoked to initialize their values.

```
<sqltags:emp id=”e” ENAME=”jack frost” JOB=”singer”/>
```

Typically the accessors are used within the body of the JSP text to output the data from the table. The example below demonstrates how to output the ENAME column to the page.

```
<sqltags:emp id=”e” ENAME=”jack frost” JOB=”singer”>
   Name is <%= e.getENAME() %>
</sqltags:emp>
```

Additionally, there are accessors for the imported (or parent) foreign keys as well.

Foreign Key Accessors
getFK_MGR_PARENT()
getFK_DEPTNO_PARENT()

These foreign key accessors are useful for pulling in data from a parent table in the context of a child record. For example, given a specific employee: EMPNO=5, the department name is returned as follows:

```
<sqltags:emp id=”e” EMPNO=”5” operation=”select”>
   <%= e.getFK_DEPTNO_PAREMT().getDNAME() %>
</sqltags:emp>
```

Column Attributes

Each column has a corresponding attribute that causes the mutator to be invoked just prior to the “doStart” call for the tag (see above). Additionally, there are two more attributes defined for each column: “_SELECT” and “_BIND”. The “_SELECT” attribute is used to reformat the select-list item so the column can be displayed as needed in the page. The “_BIND” attribute provides a mechanism to decode that formatting so the data is ready for insert, update, and delete. For example, use the following to deal with the “HIREDATE” column in “mm/dd/yyyy” format (with an Oracle database).

```
<sqltags:emp id=”e”
             HIREDATE_SELECT=”to_char(HIREDATE,’mm/dd/yyyy’)”
             HIREDATE_BIND=”to_date(?,’mm/dd/yyyy’)” />
```

Two items of significance: 1) the “_BIND” essentially undoes the “_SELECT” and 2) all of the data manipulation calls in <SQLTags:> (i.e. insert, update, delete) use JDBC “prepared statements” therefore a “?” character is required in place of the column name in the “_BIND” attribute. **Emphasis: one “?” character is required in each and every “_BIND” attribute as the placeholder for the column’s data.**

Built-in Data Manipulation Operations

Each generated table tag knows how to create insert, update, and delete statements for the table. These methods take the values of all column attributes and, using the “_BIND” values, build the correct SQL statements for each operation. The example shows how to perform an insert into the EMP table.

```
<sqltags:emp id=”e” EMPNO=”1” ENAME=”jack frost” JOB=”singer”>
   <%= e.insert() %>
</sqltags:emp>
```

DML using the Operator, ButtonName, and Properties Attributes

Additionally, a special purpose attribute, operation, can be used to automatically invoke one of the operations. The example below is equivalent to the previous example but does not require any “snippet” java code within the JSP page.

```
<sqltags:emp id=”e”
             EMPNO=”1” ENAME=”jack frost” JOB=”singer”
             operation=”insert” />
```

Also, if the desired “operation” is not known until run-time, it can be passed to the JSP page via the “request object”. The attribute “buttonName” is used to identify the request object parameter that contains the operation to be performed. In the next example the request parameter, btn, contains the value “insert” and thus an insert is performed as in the previous two examples.

```
<sqltags:emp id=”e”
             EMPNO=”1” ENAME=”jack frost” JOB=”singer”
             buttonName=”btn” />
```

Finally, another special-purpose attribute, “properties”, can be used to tell the tag to scan the “request object” for parameter names that match column names from the table. Whenever a parameter name matches a column name, the corresponding mutator is called for that column passing in the value of the parameter. For example, if EMPNO, ENAME, and JOB were submitted to our example page within the “request object” their values would automatically be incorporated into the tag.

When matching request object parameter names to column names the tag ignores differences in case, therefore, a parameter named, EmpNo, would match the database column, EMPNO.

```
<sqltags:emp id=”e” properties=”true” >
   name is <%= e.getENAME() %><BR>
</sqltags:emp>
```

Wow!

Array Processing

Request parameters that follow the “column_name[index]” format are automatically treated like an array. Different columns with matching “indexes” are grouped together into rows. Each “row” can have a distinct “operation” performed on it. For example, if the request object contained the following six elements:

```
EMPNO[E0]=”1”, ENAME[E0]=”joe”, operation[E0]=”insert”
EMPNO[E1]=”2”, ENAME[E1]=”jane”, operation[E1]=”update”
```

Then row “E0” would be inserted into the database and row “E1” would be updated. By default if no array index is specified, an index of “0” is used.

Paging

The “cursor” tag and all generated tags support the concept of paging. When paging is activated the “displaySize” attribute controls how many items from the tag are output to the page. The “previous” and “next” tags are then used to navigate between the pages. The example below demonstrates listing all employees in order by ENAME on pages of 5 employees per page.

```
<sqltags:emp id=”e” where=”order by ename”
                    paging=”true” displaySize=”5” >
   name is <%= e.getENAME() %><BR>
   <sqltags:last name=”e”>
      <sqltags:next name=”e” href=”./thisPage.jsp”>
           next</sqltags:next>
   </sqltags:last>
</sqltags:emp>
```

Typically, the next/previous links are nested within a “last” tag so they show up on the bottom of the list and not on every row.

Caching

The “cursor” tag and all generated tags support the concept of caching. When caching is activated the “cacheSize” attribute controls how many items from the query are saved to the cache. Caching is typically used in conjunction with paging to improve performance. The Caching feature is currently listed as experimental and should be used with extreme caution.

The CacheSize is rounded up to an even multiple of the pageSize.

Special-Purpose Attributes

Attribute	**description**
buttonName	Reference to the request object parameter that contains on of the following operations: insert, update, delete, select
cacheScheme	Reserved for future use.
cacheSize	Size of the primary key cache when caching is active.
caching	Activates caching. Can be either "true" or "false", defaults to "false"
columns	Defines the columns to be used in any of the following generated SQL statement: select, insert, update, and delete. Useful for updates on small subsets of columns.
displaySize	Number of rows to display when paging is active.
foreignKey	Foreign key name is required when nesting tables in a parent-child configuration. The "child" table specifies the foreign key that defines the nesting.
id	Required attribute. Defines the scriptlet variable used in the page within the tag's scope.
handlerClass	Defines a java class that implements the SQLTagsHandlerContract and provides pre and post methods for data validation and other user-defined features.
handlerID	Scriptlet id for handler class used within the JSP page.
hasFetch	Tells the tag to delegate the fetching to a nested "<sqltags:fetch> tag.
operation	Defines the operation to be performed. Valid values are: insert, update, delete, and select.
orderBy	Defines the "order by" clause for the tag. Only useful when tag is nested (based on foreignKey) because the "where" clause is defined by the foreign key relationship.
parentName	Also required when nesting based on foreign key. Defines the parent tag's ID at the other end of the foreign key.
paging	Activates paging. Valid values are "true" and "false", defaults to "false"
preInsertSQL	Defines an SQL statement that is executed just prior to the "insert". If any columns returned by this SQL statement match column names in the table, those values will be assigned to the corresponding column prior to invoking the statement.
preUpdateSQL	Defines an SQL statement that is executed just prior to the "update". If any columns returned by this SQL statement match column names in the table, those values will be assigned to the corresponding column prior to invoking the statement.
properties	Tells the tag to scan the request object for parameter names that match column names in the table. If any parameters match, their values are automatically assigned to the corresponding column.
where	Defines the where clause for the tag. Useful only for very simple

	where clauses without bind variables. For more complex where clauses, use the nested <sqltags:where> tag.

Connection Tag

Opening connections

The “connection” tag establishes a JDBC connection to the database using the properties found in the “initSrc” properties file. Every “connection” tag requires an “id” attribute. The “name” attribute is used to put the initialization properties into the session for use by other connection tags or other tags that require initialization parameters. The “autoCommit” and “readOnly” attributes modify the default behavior of the JDBC connection. The “connection” tag establishes a connection pool and simply releases the connection back to the pool at when the connection tag is closed.

```
<%-- open a database connection using properties from “INIT” --%>
<sqltags:connection id="connect"
                name="init"
                initSrc=”/WEB-INF/sqltags.properties”
                autoCommit="true"
                readOnly="false">
    <%-- do stuff --%>

<%-- release connection back into the pool --%>
</sqltags:connection>
```

Cursor and Statement Tags

The “cursor” and “statement” tags provide a mechanism to send arbitrary SQL statement to the database for processing.

```
<%-- generic SQL --%>
<sqltags:cursor id="cursor">
   <sqltags:statement name="cursor">
      select * from emp order by ename
   </sqltags:statement>
   <sqltags:fetch name="cursor">
       <%= cursor.getString("ENAME") %> <BR>
    </sqltags:fetch>
</sqltags:cursor>
```

Where and OrderBy Tags

The “where” and “orderBy” tags provide an easier way to compose complex “where” and “order by” clauses to the database. These tags are simply a way to deal with the difficult syntax required when limited to “where” and “orderBy” attributes.
The “orderBy” tag should only be used when the tag is a

```
<%-- where example --%>
<sqltags:emp id="emp">
   <sqltags:where name="emp">
        where ename like ‘A%’
        order by ename
   </sqltags:where>
   <sqltags:fetch name="emp">
      <%=emp.getENAME()%> <BR>
    </sqltags:fetch>
</sqltags:emp>
```

The “where” and “orderBy” tags are valid within the “cursor” or any “generated” tags. The required “name” attribute identifies to which “cursor” or “generated” tag the “where/orderBy” tag refers.

```
<%-- order by example --%>
<sqltags:emp id="emp" parentName="dept" foreignKey="FK_DEPTNO">
   <sqltags:orderBy name="emp">
order by ename
   </sqltags:orderBy>
   <sqltags:fetch name="emp">
      … fetching …
    </sqltags:fetch>
</sqltags:emp>
```

Fetch Tag

The “fetch” tag performs a fetch from the enclosing “cursor” or “generated” tag’s result set based on the current “where/orderBy” clause for that tag.

```
<%-- fetch example --%>
<sqltags:emp id="emp" parentName="dept" foreignKey="FK_DEPTNO">
   <sqltags:orderBy name="emp">
order by ename
   </sqltags:orderBy>
   <sqltags:fetch name="emp">
      <tr><TD><%= emp.getEMPNO() %></TD>
          <TD><%= emp.getENAME() %></TD>
      </tr>
    </sqltags:fetch>
</sqltags:emp>
```

Commit and Rollback Tags

Transaction processing is handled with the “commit” and “rollback” tags. Either tag is only acceptable within the scope of a “connection” tag. Transaction control is only useful when the connection is initialized with autoCommit set to false.

The “commit” tag causes the current transaction to be committed.

```
<%-- commit transaction --%>
<sqltags:commit />
```

The “rollback” tag causes the current transaction to be rolled back.

```
<%-- rollback transaction --%>
<sqltags:rollback />
```

First, Last, Next, and Previous Tags

The “first” tag output only for the first record displayed within cursor or generated tag’s output.

The “last” tag output only for the last record displayed within cursor or generated tag’s output.

The “next” tag works very much like the standard “A” tag; however, it adds the ability attributes required to properly address the next set of records.

The “previous” tag works very much like the standard “A” tag; however, it adds the ability attributes required to properly address the previous set of records.

```
<%-- first, last, next, previous --%>
<table>
<sqltags:cursor id="cursor" sql="select * from emp">
    <sqltags:first name="cursor">
       <%-- output header on first row --%>
       <tr><td>emp name</td></tr>
    </sqltags:first>

    <%-- this section is output for every row --%>
    <tr><td><%= cursor.getString("ENAME") %></td></tr>

    <sqltags:last name="cursor">
     <%-- output next/prev links on last row --%>

        <%-- previous and next links --%>
       <sqltags:previous name="cursor">prev</sqltags:previous>
       <sqltags:next     name="cursor">next</sqltags:next>
    </sqltags:last>
</sqltags:cursor>
</table>
```

Tag Summary

Tag	**Description**
Commit	Commit the current transaction.
connection	Get a JDBC Connection to the database.
cursor	Create and execute a database query.
fetch	Fetch data from either a cursor or generated table tag.
first	True for first row displayed on the page for the given cursor or generated tag.
generated	Generated tag will match name of the underlying table.
last	True for last row displayed on the page for a given cursor or generated tag.
Next	Anchor to next page of items; only useful when paging=”true”.
orderBy	Defines the “order by” clause for a generated table tag; usually for a foreign key operation where the where clause is inferred from the foreign key name.
previous	Anchor to previous page if items; only useful when paging=”true”.
rollback	Rollback the current transaction.
statement	Defines the SQL statement for a cursor.
where	Defines the where clause for a generated table tag.

Tag Reference

Connection Tag

The Connection tag provides a mechanism to establish a JDBC connect to the database. The specific connection properties are contained in the “properties” file that is referenced by the “name” attribute.

Attribute	***Attribute Description***
autoCommit	Sets the JDBC connection to “autoCommit”.
id *	JSP scriptlet variable used to access the tag class within the JSP page.
name	Reference to “id” of the Initialization tag where properties are defined.
initSrc	Location of the properties file. Without a leading “/” character the file is assumed to be relative to the WEB-INF directory within the “Servlet Context”.
readOnly	Sets the JDBC connection to “readOnly”.
Scriptlet Method*	***Method Description***
commit()	Commits the current transaction.
getAutoCommit()	Returns the autoCommit property for the connection.
getConnection()	Returns the current java.sql.Connection.
getException()	Returns the “String” value of the exception, if any, within the tag referenced by the “name” attribute.
getReadOnly()	Returns the readOnly property for the connection.
rollback()	Rolls back the current transaction.

Examples

```
<%-- Define the "Properties file" from initSrc --%>
<%-- open a database connection properties from "INIT" --%>
<sqltags:connection
    id="connect"
    initSrc="/WEB-INF/sqltags.properties"
    name="init"
    autoCommit="true"
    readOnly="false">
        <%-- stuff within the connection %>
</sqltags:connection>
```

* The value of the “id” attribute is used within the JSP page as a “scriptlet” variable.

Cursor Tag

The Cursor tag provides a mechanism to run any valid SQL statement. Complex SQL statements should enclosed Statement and Fetch tags.

Attribute	***Attribute Description***
cacheScheme	Defines the caching scheme to be used.
cacheSize	Sets the size of the cache.
caching	Activates automatic caching when set to “true”.
id *	JSP scriptlet variable used to access the tag class within the JSP page.
displaySize	Defines the number of rows that are displayed on a single page. Works in conjunction with startRow to control paging.
paging	Activates automatic paging of results when set to “true”.
primaryKey	Lists the columns that make up the primary key for the query. Only used for caching queries.
sql	Sets the SQL statement to be executed by the cursor tag. The sql attribute is only used for simply SQL statements; for more complex queries, use the “statement” tag embedded within the cursor tag.
Scriptlet Method*	***Method Description***
getArrayIndex()	Returns the current “array index”
getArrayIndexes()	Returns the Enumeration of available “array indexes”
getException()	Returns current exception
getSql()	Returns the current SQL statement
getString(key)	Returns the String value of the select-list item named key
setArrayIndex()	Sets the current “array index”
Examples	

```
<sqltags:cursor id="cursor" sql="select * from emp">
   <%= getString("ENAME") %><BR>
</sqltags:cursor>
```

** The value of the “id” attribute is used within the JSP page as a “scriptlet” variable.*

Generated Tags

The Generated tags provide a mechanism to access the underlying table.

Attribute	***Attribute Description***
COLUMN	Sets the value of the COLUMN named COLUMN
COLUMN_SELECT	Sets the “select-item” for COLUMN
COLUMN_BIND	Sets the “bind” format for COLUMN
cacheScheme	Reserved for future use.
cacheSize	Size of the primary key cache when caching is active.
caching	Activates caching. Can be either “true” or “false”, defaults to “false”
columns	Defines the columns to be used in any of the following generated SQL statement: select, insert, update, and delete. Useful for updates on small subsets of columns.
displaySize	Number of rows to display when paging is active.
foreignKey	Foreign key name is required when nesting tables in a parent-child configuration. The “child” table specifies the foreign key that defines the nesting.
handlerClass	Defines a java class that implements the SQLTagsHandlerContract and provides pre and post methods for data validation and other user-defined features.
handlerID	Scriptlet id for handler class used within the JSP page.
hasFetch	Tells the tag to delegate the fetching to a nested “<sqltags:fetch> tag.
id	Required attribute. Defines the scriptlet variable used in the page within the tag’s scope.
operation	Defines the operation to be performed. Valid values are: insert, update, delete, and select.
orderBy	Defines the “order by” clause for the tag. Only useful when tag is nested (based on foreignKey) because the “where” clause is defined by the foreign key relationship.
paging	Activates paging. Valid values are “true” and “false”, defaults to “false”
parentName	Also required when nesting based on foreign key. Defines the parent tag’s ID at the other end of the foreign key.
preInsertSQL	Defines an SQL statement that is executed just prior to the “insert”. If any columns returned by this SQL statement match column names in the table, those values will be assigned to the corresponding column prior to invoking the statement.

preUpdateSQL	Defines an SQL statement that is executed just prior to the “update”. If any columns returned by this SQL statement match column names in the table, those values will be assigned to the corresponding column prior to invoking the statement.
properties	Tells the tag to scan the request object for parameter names that match column names in the table. If any parameters match, their values are automatically assigned to the corresponding column.
where	Defines the where clause for the tag. Useful only for very simple where clauses without bind variables. For more complex where clauses, use the nested <sqltags:where> tag.

Scriptlet Method*	***Method Description***
getCOLUMN()	Returns the value of COLUMN named “COLUMN”.
setCOLUMN()	Sets the value of COLUMN
getArrayIndex()	Returns the current “array index”
getArrayIndexes()	Returns the Enumeration of available “array indexes”
getException()	Returns current exception
getSql()	Returns the current SQL statement
getWhere()	Returns the current where clause
setArrayIndex()	Sets the current “array index”

Examples

```
<sqltags:emp id="e" where="order by ename">
   <%= e.getENAME() %><BR>
</sqltags:emp>
```

** The value of the “id” attribute is used within the JSP page as a “scriptlet” variable.*

Previous Tag

The Previous tag provides a mechanism to output a “NEXT” link when additional records exist within a “paged” record set. This tag works within any “generated” tag or the Cursor tag, but is only useful when the parent tag has activated paging.

Attribute	***Attribute Description***
alt	Passed on to the generated <A> tag.
href	URL for active link of next records (?startRow=n with appropriate value will be appended to the end of the href).
noHref	Output sent to browser when there is no previous page.
onClick	Passed on to the generated <A> tag.
parentName	Reference to the Cursor or “generated” tag’s “id” attribute.
Examples	

```
<sqltags:cursor id="cursor" sql="select * from emp">
    <sqltags:last name="cursor">
       <sqltags:previous parentName="cursor" href="demo.jsp">
           Previous page</sqltags:previous>
    </sqltags:last>
</sqltags:cursor>
```

Next Tag

The Next tag provides a similar mechanism as “Previous” only it is used to link to the previous page of records.

Attribute	***Attribute Description***
alt	Passed on to the generated <A> tag.
href	URL for active link of next records (?startRow=n with appropriate value will be appended to the end of the href).
noHref	Output sent to browser when there is no next page.
onClick	Passed on to the generated <A> tag.
parentName	Reference to the Cursor or “generated” tag’s “id” attribute.
Examples	

```
<sqltags:cursor id="cursor" sql="select * from emp">
    <sqltags:last name="cursor">
       <sqltags:previous parentName="cursor" href="demo.jsp">
           Previous page</sqltags:previous>
    </sqltags:last>
</sqltags:cursor>
```

Simple Built-in Tags

The simple "Built-in" tags only have a single attribute, "name". This attribute identifies the master or "parent" tag. These simple tags only provide a tag-based mechanism to invoke specific methods available within the "parent" tags.

Tag	***Parent***	***Tag Description***
commit	connection	Performs a database commit.
exception	cursor, generated, connection	Outputs the exception contained within tag identified by "name" attribute to the browser if the exception is not null or blank.
fetch	cursor, generated	Performs a "fetch()" on either Cursor or generated table tag identified as by the "name" attribute.
first	cursor, generated	Evaluates enclosed JSP if the current row being "fetched" is the first row to be displayed.
last	cursor, generated	Evaluates enclosed JSP if the current row being "fetched" is the last row to be displayed on the page.
rollback	connection	Performs a database rollback.
statement	cursor	Defines the SQL statement for the cursor tag identified by the "name" attribute.
where	generated	Defines the where clause for the tag whose "id" attribute matches the "name" attribute.

Examples

See the “/example” directory in the software bundle.

Java Docs

See the “/javadoc” directory in the software bundle.

Revision History

Beta Release