HOWTO - Stored Procedure Support

by Ron Gallagher

Table of contents

1. Introduction

OJB supports the use of stored procedures to handle the basic DML operations (INSERT, UPDATE, and DELETE). This document will describe the entries that you'll need to add to your repository in order to get OJB to utilize stored procedures instead of 'traditional' INSERT, UPDATE or DELETE statements.

Please note that there will be references to 'stored procedures' throughout this document. However, this is just a simplification for the purposes of this document. Any place you see a reference to 'stored procedure', you can assume that either a stored procedure or function can be used.

Information presented in this document includes the following:

- Basic repository entries
- Common attributes for all procedure descriptors
- An overview of the insert procedure, update procedure and delete procedure descriptors.
- Information about the argument descriptors that are supported for all procedure
- A simple example and a more complex example

2. Repository entries

For any persistable class (i.e. "com.myproject.Customer") where you want to utilize stored procedures to handle persistence operations instead of traditional DML statements (i.e. INSERT, UPDATE or DELETE), you will need to include one or more of the following descriptors within the corresponding class-descriptor for the persistable class:

- `insert-procedure` - identifies the stored procedure that is to be used whenever a class needs to be inserted into the database.
- `update-procedure` - identifies the stored procedure that is to be used whenever a class needs to be updated in the database.
- `delete-procedure` - identifies the stored procedure that is to be used whenever a class needs to be removed from the database.

All of these descriptors must be nested within the class-descriptor that they apply to. Here is an example of a simple class-descriptor that includes each of the procedure descriptors listed above:

```
<class-descriptor  class="com.myproject.Customer" table="CUSTOMER">
<field-descriptor column="ID" jdbc-type="DECIMAL" name="id" primarykey="true"/>
<field-descriptor column="NAME" jdbc-type="VARCHAR" name="name"/>
<insert-procedure name="CUSTOMER_PKG.ADD">
    <runtime-argument field-ref="id" return="true"/>
    <runtime-argument field-ref="name"/>
</insert-procedure>
<update-procedure name="CUSTOMER_PKG.CHG">
    <runtime-argument field-ref="id"/>
    <runtime-argument field-ref="name"/>
</update-procedure>
<delete-procedure name="CUSTOMER_PKG.CHG">
    <runtime-argument field-ref="id"/>
</delete-procedure>
</class-descriptor>
```

3. Common attributes

All three procedure descriptors have the following attributes in common:

- `name` - This is the name of the stored procedure that is to be used to handle the specific persistence operation.
- `return-field-ref` - This identifies the field in the class where the return value from the

stored procedure will be stored. If this attribute is blank or not specified, then OJB will assume that the stored procedure does not return a value and will format the SQL command accordingly.

The basic syntax that is used to call a procedure that has a return value looks something like this:

```
{?= call &lt;procedure-name&gt;[&lt;arg1&gt;,&lt;arg2&gt;, ...]}
```

The basic syntax that is used to call a procedure that **does not** include a return value looks something like this:

```
{call &lt;procedure-name&gt;[&lt;arg1&gt;,&lt;arg2&gt;, ...]}
```

When OJB assembles the SQL to call a stored procedure, it will use the value of the 'name' attribute in place of 'procedure-name' in these two examples.

In addition, if the procedure descriptor includes a value in the 'return-field-ref' attribute that is 'valid', then the syntax that OJB builds will include the placeholder for the result parameter.

The previous section referred to the idea of a 'valid' value in the 'return-field-ref' attribute. A value is considered to be 'valid' if it meets the following criteria:

- The value is not blank
- There is a field-descriptor with a 'name' that matches the value in the 'return-field-ref' attribute.

If the 'return-field-ref' attribute is not 'valid', then the placeholder for the result parameter will not be included in the SQL that OJB assembles.

4. insert-procedure

The insert-procedure descriptor identifies the stored procedure that should be used whenever a class needs to be inserted into the database. In addition to the common attributes listed earlier, the insert-procedure includes the following attribute:

- include-all-fields

This attribute provides an efficient mechanism for passing all attributes of a persistable class to a stored procedure. If this attribute is set to true, then OJB will ignore any nested argument descriptors. Instead, OJB will assume that the argument list for the stored procedure includes arguments for all attributes of the persistable class and that those arguments appear in the same order as the field-descriptors for the persistable class.

The default value for this attribute is 'false'.

Note:
If the field-descriptors in your repository do not 'align' exactly with the argument list for the stored procedure, or you want to maintain explicit control over the values that are passed to the stored procedure, then either set the 'include-all-fields' attribute to 'false' or leave it off the insert-procedure descriptor.

5. update-procedure

The update-procedure descriptor identifies the stored procedure that should be used whenever a class needs to be updated in the database. In addition to the common attributes listed earlier, the update-procedure includes the following attribute:

- include-all-fields

This attribute provides the same capabilities and has the same caveats as the include-all-fields attribute on the insert-procedure descriptor.

6. delete-procedure

The delete-procedure descriptor identifies the stored procedure that should be used whenever a class needs to be deleted from the database. In addition to the common attributes listed earlier, the delete-procedure includes the following attribute:

- include-pk-only
  This attribute provides an efficient mechanism for passing all of the attributes that make up the primary key for a persistable class to the specified stored procedure. If this attribute is set to true, then OJB will ignore any nested argument descriptors. Instead, OJB will assume that the argument list for the stored procedure includes arguments for all attributes that make up the primary key for the persistable class (i.e. those field-descriptors where the 'primary-key' attribute is set to 'true'). OJB will also assume that those arguments appear in the same order as the corresponding field-descriptors for the persistable class.
  The default value for this attribute is 'false'.

Note:
If the field-descriptors in your repository that make up the primary key for a persistable class do not 'align' exactly with the argument list for the stored procedure, or you want to maintain explicit control over the values that are passed to the stored procedure, then either set the 'include-pk-only' attribute to 'false' or leave it off the delete-procedure descriptor.

7. Argument descriptors

Argument descriptors are the mechanism that you will use to tell OJB two things:

1. How many placeholders should be included in the argument list for a stored procedure?
2. What value should be passed for each of those arguments?

There are two types of argument descriptors that can be defined in the repository:

- runtime arguments used to set a stored procedure argument equal to a value that is only known at runtime.
- constant arguments used to set a stored procedure argument equal to constant value.

You may notice that there is no argument descriptor specifically designed to pass a *null* value to the procedure. This capability is provided by the runtime argument descriptor.

The argument descriptors are essentially the 'mappings' between stored procedure arguments and their runtime values. Each procedure descriptor can include 0 or more argument descriptors in it's definition.

After reading that last comment, you may wonder why OJB allows you to configure a procedure descriptor with no argument descriptors since the primary focus of OJB is to handle object persistence. How could OJB perform any sort persistence operation using a stored procedure that did not involve the passage of at least one value to the stored procedure? To be honest, it is extremely unlikely that you would ever set up a procedure descriptor with no argument descriptors. However, since there is no minimum number of arguments required for a stored procedure, we did not want to implement within OJB a requirement on the number of arguments that was more restrictive than the limits imposed by most/all database systems.

7.1. runtime-argument descriptors

A runtime-argument descriptor is used to set a stored procedure argument equal to a value that is only known at runtime.

Two attributes can be specified for each runtime-argument descriptor:

- field-ref
  The 'field-ref' attribute identifies the specific field descriptor that will provide the argument's value. If this attribute is not specified or does not resolve to a valid field-descriptor, then a null value will be passed to the stored procedure.
- return
  The 'return' attribute is used to determine if the argument is used by the stored procedure as an 'output' argument.
  If this attribute is set to true, then the corresponding argument will be registered as an output parameter. After execution of the stored procedure, the value of the argument will be 'harvested' from the CallableStatement and stored in the attribute identified by the field-ref attribute.
  If this attribute is not specified or set to false, then OJB assumes that the argument is simply an 'input' argument, and it will do nothing special to the argument.

7.2. constant-argument descriptors

A constant-argument descriptor is used to set a stored procedure argument equal to constant value.

There is one attribute that can be specified for the constant-argument descriptor:

- value
  The 'value' attribute identifies the value for the argument.

8. A simple example

This section provides background information and a simple example that illustrates how OJB's support for stored procedures can be utilized.

The background information covers the following topics:

- The basic requirements
- The database objects including the table that will be manipulated, the sequence that will be used by the stored procedures to assign primary key falues, the insert and update triggers that maintain the four 'audit' columns and the package that provides the stored procedures that will handle the persistence operations.

Click here to skip the background information and go straight to the implementation.

8.1. The basic requirements

These are the requirements that must be satisfied by our example

1. All insert, update and delete operations are to be performed by stored procedures.
2. All primary key values are to be by the stored procedure that handles the insert operation. The value that is assigned should be reflected in the object that 'triggered' the insert operation.
3. For auditing purposes, all tables will include the following set of columns:
   - `USER_CREATED` - This will contain the id of the user who created the record
   - `DATE_CREATED` - The date on which the record was created created
   - `USER_UPDATED` - The id of the user who last modified the record
   - `USER_UPDATED` - The date on which the record was last modified

   In addition to the inclusion of these columns on each table, the following requirements related to these columns had to be supported:
   1. The values of the two date-related audit columns were to be maintained at the database level via insert and update triggers.
      - The insert trigger will set both `DATE_CREATED` and `DATE_UPDATED` to the current system date.

- The update trigger will set DATE_UPDATED to the current system date. The update trigger will also ensure that the original value of DATE_CREATED is never modified.

2. The values of the two user-related audit columns are to be maintained at the database level via insert and update triggers.
   - The insert and update triggers will ensure that USER_CREATED and USER_UPDATED are appropriately populated.
   - The update trigger will ensure that the original value of USER_CREATED is never modified.
3. Any changes that are made by the insert or update triggers to any of the four 'audit' columns had to be reflected in the object that caused the insert or update operation to occur.

8.2. The database objects

The database objects that are described in this section utilize Oracle specific syntax. However, you should not infer from this that the stored procedure support provided by OJB can only be used to access data that is stored in an Oracle database. In reality, stored procedures can be used for persistence operations in any database that supports stored procedures.

- The table that will be manipulated,
- The sequence that will be used by the stored procedures to assign primary key values
- The insert and update triggers that maintain the four 'audit' columns
- The package that provides the stored procedures that will handle the persistence operations.

Click here to skip the information about the database objects and go straight to the implementation.

8.3. The CUSTOMER table

This example will deal exclusively with persistence operations related to the a table named 'CUSTOMER' that is built using the following DDL:

```
CREATE TABLE CUSTOMER
( ID NUMBER(18) NOT NULL
, NAME VARCHAR2(50) NOT NULL
, USER_CREATED VARCHAR2(30)
, DATE_CREATED DATE
, USER_UPDATED VARCHAR2(30)
, DATE_UPDATED DATE
, CONSTRAINT PK_CUSTOMER PRIMARY KEY (ID)
);
```

8.4. The sequence

This sequence will be used to assign unique values to CUSTOMER.ID.

```
             CREATE SEQUENCE CUSTOMER_SEQ;
```

8.5. The insert and update triggers

These two triggers will implement all of the requirements listed above that are related to the four audit columns:

```
CREATE OR REPLACE TRIGGER CUSTOMER_ITR
BEFORE INSERT ON CUSTOMER
FOR EACH ROW
BEGIN
--
-- Populate the audit dates
--
SELECT SYSDATE, SYSDATE
```

```
INTO :NEW.DATE_CREATED, :NEW.DATE_UPDATED
FROM DUAL;
--
-- Make sure the user created column is populated.
--
IF :NEW.USER_CREATED IS NULL
THEN
SELECT SYS_CONTEXT('USERENV','TERMINAL')
  INTO :NEW.USER_CREATED
  FROM DUAL;
END IF;
--
-- Make sure the user updated column is populated.
--
IF :NEW.USER_UPDATED IS NULL
THEN
SELECT SYS_CONTEXT('USERENV','TERMINAL')
  INTO :NEW.USER_UPDATED
  FROM DUAL;
END IF;
END;
/

CREATE OR REPLACE TRIGGER CUSTOMER_UTR
BEFORE UPDATE ON CUSTOMER
FOR EACH ROW
BEGIN
--
-- Populate the date updated
--
SELECT SYSDATE
INTO :NEW.DATE_UPDATED
FROM DUAL;
--
-- Make sure the user updated column is populated.
--
IF :NEW.USER_UPDATED IS NULL
THEN
SELECT SYS_CONTEXT('USERENV','TERMINAL')
  INTO :NEW.USER_UPDATED
  FROM DUAL;
END IF;
--
-- Make sure the date/user created are never changed
--
SELECT :OLD.DATE_CREATED, :OLD.USER_CREATED
INTO :NEW.DATE_CREATED, :NEW.USER_CREATED
FROM DUAL;
END;
/
```

8.6. The package

This Oracle package will handle all INSERT, UPDATE and DELETE operations involving the `CUSTOMER` table.

```

CREATE OR REPLACE PACKAGE CUSTOMER_PKG AS
--
-- This procedure should be used to add a record to the CUSTOMER table.
--
PROCEDURE ADD ( AID           IN OUT CUSTOMER.ID%TYPE
              , ANAME         IN     CUSTOMER.NAME%TYPE
              , AUSER_CREATED IN OUT CUSTOMER.USER_CREATED%TYPE
              , ADATE_CREATED IN OUT CUSTOMER.DATE_CREATED%TYPE
              , AUSER_UPDATED IN OUT CUSTOMER.USER_UPDATED%TYPE
              , ADATE_UPDATED IN OUT CUSTOMER.DATE_UPDATED%TYPE );
--
-- This procedure should be used to change a record on the CUSTOMER table.
--
```

```
PROCEDURE CHANGE ( AID           IN     CUSTOMER.ID%TYPE
                 , ANAME         IN     CUSTOMER.NAME%TYPE
                 , AUSER_CREATED IN OUT CUSTOMER.USER_CREATED%TYPE
                 , ADATE_CREATED IN OUT CUSTOMER.DATE_CREATED%TYPE
                 , AUSER_UPDATED IN OUT CUSTOMER.USER_UPDATED%TYPE
                 , ADATE_UPDATED IN OUT CUSTOMER.DATE_UPDATED%TYPE );
--
-- This procedure should be used to delete a record from the CUSTOMER table.
--
PROCEDURE DELETE ( AID IN CUSTOMER.ID%TYPE );
END CUSTOMER_PKG;
/
CREATE OR REPLACE PACKAGE BODY CUSTOMER_PKG AS
--
-- This procedure should be used to add a record to the CUSTOMER table.
--
PROCEDURE ADD ( AID           IN OUT CUSTOMER.ID%TYPE
              , ANAME         IN     CUSTOMER.NAME%TYPE
              , AUSER_CREATED IN OUT CUSTOMER.USER_CREATED%TYPE
              , ADATE_CREATED IN OUT CUSTOMER.DATE_CREATED%TYPE
              , AUSER_UPDATED IN OUT CUSTOMER.USER_UPDATED%TYPE
              , ADATE_UPDATED IN OUT CUSTOMER.DATE_UPDATED%TYPE )
IS
  NEW_SEQUENCE_1 CUSTOMER.ID%TYPE;
BEGIN
  SELECT CUSTOMER_SEQ.NEXTVAL
    INTO NEW_SEQUENCE_1
    FROM DUAL;
  INSERT INTO CUSTOMER ( ID, NAME, USER_CREATED, USER_UPDATED )
     VALUES ( NEW_SEQUENCE_1, ANAME, AUSER_CREATED, AUSER_UPDATED )
    RETURNING ID, USER_CREATED, DATE_CREATED, USER_UPDATED, DATE_UPDATED
         INTO AID, AUSER_CREATED, ADATE_CREATED, AUSER_UPDATED, ADATE_UPDATED;
END ADD;
--
-- This procedure should be used to change a record on the CUSTOMER table.
--
PROCEDURE CHANGE ( AID           IN     CUSTOMER.ID%TYPE
                 , ANAME         IN     CUSTOMER.NAME%TYPE
                 , AUSER_CREATED IN OUT CUSTOMER.USER_CREATED%TYPE
                 , ADATE_CREATED IN OUT CUSTOMER.DATE_CREATED%TYPE
                 , AUSER_UPDATED IN OUT CUSTOMER.USER_UPDATED%TYPE
                 , ADATE_UPDATED IN OUT CUSTOMER.DATE_UPDATED%TYPE )
IS
BEGIN
  UPDATE CUSTOMER
     SET NAME         = ANAME
       , USER_CREATED = USER_CREATED
       , USER_UPDATED = AUSER_UPDATED
   WHERE ID           = AID
    RETURNING USER_CREATED, DATE_CREATED, USER_UPDATED, DATE_UPDATED
         INTO AUSER_CREATED, ADATE_CREATED, AUSER_UPDATED, ADATE_UPDATED;
END CHANGE;
--
-- This procedure should be used to delete a record from the CUSTOMER table.
--
PROCEDURE DELETE ( AID IN CUSTOMER.ID%TYPE )
IS
BEGIN
  DELETE
    FROM CUSTOMER
   WHERE ID    = AID;
END DELETE;
END CUSTOMER_PKG;
/
```

Please note the following about the structure of the `CUSTOMER_PKG` package:

- The `AID` argument that is passed to the the `ADD` procedure is defined as `IN OUT`. This allows the procedure to return the newly assigned `ID` to the caller.
- In the `ADD` and `CHANGE` procedures, the arguments that correspond to the four 'audit' columns

are defined as `IN OUT`. This allows the procedure to return the current value of these columns to the 'caller'.

8.7. The implementation

Getting OJB to utilize the stored procedures described earlier in this document is as simple as adding a few descriptors to the repository. Here is a class-descriptor related to the `CUSTOMER` table that includes all of the necessary descriptors.

```
<class-descriptor class="com.myproject.Customer" table="CUSTOMER">
    <field-descriptor column="ID" jdbc-type="DECIMAL" name="id"
primarykey="true"/>
    <field-descriptor column="NAME" jdbc-type="VARCHAR" name="name"/>
    <field-descriptor column="USER_CREATED" jdbc-type="VARCHAR"
name="userCreated"/>
    <field-descriptor column="DATE_CREATED" jdbc-type="TIMESTAMP"
name="dateCreated"/>
    <field-descriptor column="USER_UPDATED" jdbc-type="VARCHAR"
name="userUpdated"/>
    <field-descriptor column="DATE_UPDATED" jdbc-type="TIMESTAMP"
name="dateUpdated"/>
    <insert-procedure name="CUSTOMER_PKG.ADD">
      <runtime-argument field-ref="id" return="true"/>
      <runtime-argument field-ref="name"/>
      <runtime-argument field-ref="userCreated" return="true"/>
      <runtime-argument field-ref="dateCreated" return="true"/>
      <runtime-argument field-ref="userUpdated" return="true"/>
      <runtime-argument field-ref="dateUpdated" return="true"/>
    </insert-procedure>
    <update-procedure name="CUSTOMER_PKG.CHG">
      <runtime-argument field-ref="id"/>
      <runtime-argument field-ref="name"/>
      <runtime-argument field-ref="userCreated" return="true"/>
      <runtime-argument field-ref="dateCreated" return="true"/>
      <runtime-argument field-ref="userUpdated" return="true"/>
      <runtime-argument field-ref="dateUpdated" return="true"/>
    </update-procedure>
    <delete-procedure name="CUSTOMER_PKG.CHG">
      <runtime-argument field-ref="id"/>
    </delete-procedure>
</class-descriptor>
```

Some things to note about this class-descriptor:

1. In the insert-procedure descriptor, the first runtime-argument descriptor correspnds to the "AID" argument that is passed to the CUSTOMER_PKG.ADD routine. The "return" attribute on this runtime-argument is set to "true". With this configuration, OJB will 'harvest' the value that is returned by the CUSTOMER_PKG.ADD stored procedure and store the value in the "id" attribute on the com.myproject.Customer class.
2. In both the insert-procedure and update-procedure descriptors, the runtime-argument descriptors that correspond to the four 'audit' columns all have the "return" argument set to "true". This allows any updates that are made by the procedure or the insert/update triggers to be reflected in the "Customer" object that caused the insert/update operation to occur.

9. A complex example

This example builds upon the simple example that was presented earlier by introducing some additional requirements beyond those that were specified in the simple example. Some of these additional requirements may seem a little contrived. To be honest, they are. The only purpose of these additional requirements is to create situations that illustrate how the additional capabilities provided by OJB's support for stored procedures can be utilized.

The additional requirements for this example include the following:

- All procedures will include two additional arguments. These two new arguments will be added to the end of the argument list for all existing procedures.
  - ASOURCE_SYSTEM - identifies the system that initiated the persistence operation. This will provide a higher level of audit tracking capability. In our example, this will always be "SAMPLE".
  - ACOST_CENTER - identifies the 'cost center' that should be charged for the persistence operation. In our example, this argument will always be null.
- For all "ADD" and "CHG" stored procedures, the value that was assigned to the "DATE_UPDATED" column will no longer be returned to the caller via an "IN OUT" argument. Instead, it will be returend to the caller via the procedure's return value.

Based on these new requirements, the class-descriptor for the "com.myproject.Customer" class will look like this. The specific changes are detailed below.

```
<class-descriptor class="com.myproject.Customer" table="CUSTOMER">
    <field-descriptor column="ID" jdbc-type="DECIMAL" name="id"
primarykey="true"/>
    <field-descriptor column="NAME" jdbc-type="VARCHAR" name="name"/>
    <field-descriptor column="USER_CREATED" jdbc-type="VARCHAR"
name="userCreated"/>
    <field-descriptor column="DATE_CREATED" jdbc-type="TIMESTAMP"
name="dateCreated"/>
    <field-descriptor column="USER_UPDATED" jdbc-type="VARCHAR"
name="userUpdated"/>
    <field-descriptor column="DATE_UPDATED" jdbc-type="TIMESTAMP"
name="dateUpdated"/>
    <insert-procedure name="CUSTOMER_PKG.ADD"
          return-field-ref="dateUpdated"> <!-- See note 1 -->
      <runtime-argument field-ref="id" return="true"/>
      <runtime-argument field-ref="name"/>
      <runtime-argument field-ref="userCreated" return="true"/>
      <runtime-argument field-ref="dateCreated" return="true"/>
      <runtime-argument field-ref="userUpdated" return="true"/>
      <runtime-argument field-ref="dateUpdated"/> <!-- See note 2 -->
      <constant-argument value="SAMPLE"/> <!-- See note 3 -->
      <runtime-argument/> <!-- See note 4 -->
    </insert-procedure>
    <update-procedure name="CUSTOMER_PKG.CHG"
          return-field-ref="dateUpdated"> <!-- See note 1 -->
      <runtime-argument field-ref="id"/>
      <runtime-argument field-ref="name"/>
      <runtime-argument field-ref="userCreated" return="true"/>
      <runtime-argument field-ref="dateCreated" return="true"/>
      <runtime-argument field-ref="userUpdated" return="true"/>
      <runtime-argument field-ref="dateUpdated"/> <!-- See note 2 -->
      <constant-argument value="SAMPLE"/> <!-- See note 3 -->
      <runtime-argument/> <!-- See note 4 -->
    </update-procedure>
    <delete-procedure name="CUSTOMER_PKG.CHG">
      <runtime-argument field-ref="id"/>
      <constant-argument value="SAMPLE"/> <!-- See note 3 -->
      <runtime-argument/> <!-- See note 4 -->
    </delete-procedure>
</class-descriptor>
```

Here are an explanation of each modification:

- **Note 1:** The value that is returned by the "ADD" and "CHG" stored procedures will now be stored in the "dateUpdated" attribute on the "com.myproject.Customer" class.
- **Note 2:** Since the ADATE_UPDATED argument is no longer defined as an "IN OUT" argument, we have removed the "return" attribute from the corresponding runtime-argument descriptor.
- **Note 3:** This is the first of two new arguments that were added to the argument list of each procedure. This argument represents the 'source system', the system that initiated the

persistence operation. In our example, we will always pass a value of 'SAMPLE'.

- **Note 4:** This is the second of two new arguments that were added to the argument list of each procedure. This argument represents the 'cost center' that should be charged for the persistence operation. In our example, we have no cost center, so we need to pass a null value. This is accomplished by including a 'runtime-argument' descriptor that has no 'field-ref' specified.