HOWTO - Use DB Sequences

by Brian McCallister

Table of contents

1. Introduction

It is easy to use OJB with with database generated sequences. Typically a table using database generated sequences will autogenerate a unique id for a field as the default value for that field. This can be particularly useful if multiple applications access the same database. Not every application will be using OJB and find it convenient to pull unique values from a high/low table. Using a database managed sequence can help to enforce unique id's across applications all adding to the same database. All of that said, care needs to be taken as using database generated sequences imposes some portability problems.

OJB includes a sequence manager implementation that is aware of database sequences and how to use them. It is known to work against Oracle, SAP DB, and PostgreSQL. MySQL has its own sequence manager implementation because it is special. This tutorial will build against PostgreSQL, but working against Oracle or SAP will work the same way.

Additional information on sequence managers is available in the Sequence Manager documentation.

2. The Sample Database

Before we can work with OJB against a database with a sequence, we need the database. We will create a simple table that pulls its primary key from a sequence named 'UniqueIdentifier'.

```
CREATE TABLE thingie
(
    name VARCHAR(50),
    id INTEGER DEFAULT NEXTVAL('UniqueIdentifier')
)
```

We must also define the sequence from which it is drawing values:

```
CREATE SEQUENCE UniqueIdentifier;
```

So that we have the following table:

```
                    Table "public.thingie"
Column |         Type          |                 Modifiers
--------+-----------------------+-------------------------------------------
name   | character varying(50) |
id     | integer               | default nextval('UniqueIdentifier'::text)
```

If we manually insert some entries into this table they will have their `id` field set automagically.

```
INSERT INTO thingie (name) VALUES ('Fred');
INSERT INTO thingie (name) VALUES ('Wilma');
SELECT name, id FROM thingie;

 name  | id
-------+----
 Fred  |  0
 Wilma |  1
(2 rows)
```

3. Using OJB

3.1. The Database Repository Descriptor

The next step is to configure OJB to access our `thingie` table. We need to configure the corrct sequence manager in the `repository-database.xml`.

The default `repository-database.xml` uses the High/Low Sequence manager. We will delete or comment out that entry, and replace it with the `org.apache.ojb.broker.util.sequence.SequenceManagerNextValImpl` manager. This manager will pull the next value from a named sequence and use it. The entry for our sequence manager in the repository is:

```
<sequence-manager
    className="org.apache.ojb.broker.util.sequence.SequenceManagerNextValImpl"
/>
```

This needs to be declared within the JDBC Connection descriptor, so an entire `repository-database.xml` might look like:

```
<jdbc-connection-descriptor
    jcd-alias="default"
    default-connection="true"
    platform="PostgreSQL"
    jdbc-level="2.0"
    driver="org.postgresql.Driver"
    protocol="jdbc"
    subprotocol="postgresql"
    dbalias="test"
    username="tester"
    password=""
    eager-release="false"
    batch-mode="false"
    useAutoCommit="1"
    ignoreAutoCommitExceptions="false"
    >

    <connection-pool
        maxActive="21"
        validationQuery=""/>

    <sequence-manager
className="org.apache.ojb.broker.util.sequence.SequenceManagerNextValImpl" />
</jdbc-connection-descriptor>
```

3.2. Defining a Thingie Class

For the sake of simplicity we will make a very basic Java Thingie:

```
public class Thingie
{
     /** thingie(name) */
    private String name;

    /** thingie(id) */
    private int id;

    public String getName()           { return this.name; }
    public void setName(String name)  { this.name = name; }

    public int getId()                { return this.id; }
}
```

We also need a class descriptor in `repository-user.xml` that appears as follows:

```
<class-descriptor
    class="Thingie"
    table="THINGIE"
    >
    <field-descriptor
        name="id"
        column="ID"
        jdbc-type="INTEGER"
```

```
        primarykey="true"
        autoincrement="true"
        sequence-name="UniqueIdentifier"
        />
    <field-descriptor
        name="name"
        column="NAME"
        jdbc-type="VARCHAR"
        />
</class-descriptor>
```

Look over the `id` field descriptor carefully. The `autoincrement` and `sequence-name` attributes are important for getting our desired behavior. These tell OJB to use the sequence manager we defined to auto-increment the the value in `id`, and they also tell the sequence manager which database sequence to use - in this case `UniqueIdentifier`

We could allow OJB to create an extent-aware sequence and use it, however as we are working against a table that defaults to a specific named sequence, we want to make sure to pull values from that same sequence. Information on allowing OJB to create its own sequences is available in the Sequence Manager documentation.

3.3. Using Thingie

Just to demonstrate that this all works, here is a simple application that uses our Thingie.

```
import org.apache.ojb.broker.PersistenceBroker;
import org.apache.ojb.broker.PersistenceBrokerFactory;

public class ThingieDriver
{
    public static void main(String [] args)
    {
        PersistenceBroker broker =
PersistenceBrokerFactory.defaultPersistenceBroker();

        Thingie thing = new Thingie();
        Thingie otherThing = new Thingie();

        thing.setName("Mabob");
        otherThing.setName("Majig");

        broker.beginTransaction();
        broker.store(thing);
        broker.store(otherThing);
        broker.commitTransaction();

        System.out.println(thing.getName() + " : " + thing.getId());
        System.out.println(otherThing.getName() + " : " + otherThing.getId());
        broker.close();
    }
}
```

When it is run, it will create two Thingie instances, store them in the database, and report on their assigned `id` values.

```
java -cp [...] ThingieDriver

Mabob : 2
Majig : 3
```