Reladomo Primary Key Generator

July 22, 2007

Table of Contents

1. Reladomo PrimaryKey Generator

Primary key generator is an optional feature in Reladomo that allows Reladomo objects to declare how the primary key is going to be generated.

Two strategies can be used when declaring a primary key generator: **Max or SimulatedSequence**.

1.1. Max

When using Max strategy, the primary key generator will get the maximum value for the primary key column in the database and it will increment it to generate the next primary key value.

1.2. SimulatedSequence

A sequence is a database object from which multiple users may generate integers across the system. Sequences can be used to automatically generate primary key values. A user can create different sequences to generate primary keys for different tables, or can even share the same sequence across many tables.

The SimulatedSequence strategy uses a database table to keep the sequence name and the next value of the sequence. In this table, each row represents a different sequence. The user will need to supply a MithraSequence object that is used to access the sequence values from the database and a MithraSequenceObjectFactory object that will return the corresponding MithraSequence object. MithraSequence and MithraSequenceObjectFactory will be explained in the next sections.

1.2.1. com.gs.fw.common.mithra.MithraSequence

For simulated sequence strategy, Reladomo users need to provide a MithraObject that will map to the sequence table in the database. This Reladomo object must implement the MithraSequence interface:

```
public interface MithraSequence
{
    public void setSequenceName(String sequenceName);

    public long getNextId();

    public void setNextId(long nextValue);
}
```

The MithraSequence implementation will act as an adapter in the case that attributes in the user-provided MithraSequence object do not match the ones that the primary key generator uses. This gives the user a way to convert the application programming interface of the user-provided MithraSequence object to the interface that the primary key generator expects:

```
<MithraObject objectType="transactional">
    <PackageName>com.gs.fw.common.mithra.test.domain</PackageName>
    <ClassName>MithraTestSequence</ClassName>
    <DefaultTable>MITHRA_TEST_SEQUENCE</DefaultTable>


  <Attribute name="simulatedSequenceName" javaType="String" columnName="SEQUENCE_N
>

  <Attribute name="nextValue" javaType="long" columnName="NEXT_VALUE"/
>
</MithraObject>
```

```
public class MithraTestSequence extends
 MithraTestSequenceAbstract implements MithraSequence
{
    public long getNextId()
    {
        return this.getNextValue();
    }

    public void setNextId(long nextValue)
    {
        this.setNextValue(nextValue);
    }
}
```

1.2.2. com.gs.fw.common.mithra.MithraSequenceObjectFactory

In addition, Reladomo users need to provide a sequence object factory class. This class must implement the MithraSequenceObjectFactory interface:

```
public interface MithraSequenceObjectFactory
{
     public MithraSequence getMithraSequenceObject(String
 sequenceName, Object sourceAttribute, int initialValue);
}
```

This factory will provide the primary key generator with a common way to access the user provided MithraSequence. The getMithraSequenceObject() method must return the simulated sequence object, and should create and insert it in the database if it does not already exist.

```
public class MithraTestSequenceObjectFactory implements
 MithraSequenceObjectFactory
```

```
{
    public MithraSequence getMithraSequenceObject(String sequenceName,
 Object sourceAttribute, int initialValue)
    {
        Operation op =
 MithraTestSequenceFinder.simulatedSequenceName().eq(sequenceName);
        MithraTestSequence mithraTestSequence =
 MithraTestSequenceFinder.findOne(op);
        if (mithraTestSequence == null)
        {
            mithraTestSequence = new MithraTestSequence()
            mithraTestSequence.setSequenceName(sequenceName);
            mithraTestSequence.setNextId(initialValue);
            mithraTestSequence.insert();
        }
        return mithraTestSequence;
    }
}
```

2. Primary Key Generator XML syntax

In order to make use of this feature, some modifications to the primary key attribute tags in the MithraObject XML will be needed.

For Max strategy:

```
<Attribute name="attributeName"
           javaType="int or long"
           columnName="dbColumn name"
           primaryKey="true"
           primaryKeyGeneratorStrategy="Max" />
```

Max strategy example:

```
<MithraObject objectType="transactional">
    <PackageName>com.gs.fw.common.mithra.test.domain</PackageName>
    <ClassName>AccountTransactionMax</ClassName>
    <DefaultTable>ACCOUNT_TRANSACTION_MAX</DefaultTable>
    <SourceAttribute name="deskId" javaType="String"/>
    <Attribute name="transactionId"
               javaType="long"
               columnName="TRANSACTION_ID"
               primaryKey="true"
               primaryKeyGeneratorStrategy="Max"/>
                    ...
</MithraObject>
```

For SimulatedSequence strategy:

```
<Attribute name="attributeName"
           javaType="int or long"
           columnName="dbColumn name"
           primaryKey="true"
           primaryKeyGeneratorStrategy="SimulatedSequence">
        <SimulatedSequence sequenceName="sequenceName"

  sequenceObjectFactoryName="sequenceObjectFactory"
                         hasSourceAttribute="false"
                         batchSize="10"
                         initialValue="1"
                         incrementSize="1"/>
</Attribute>
```

- **sequenceName** - name of the sequence to be created.
- **sequenceObjectFactoryName** - fully qualified classname for the sequence object factory. This is a user provided class that follows a factory method pattern for MithraSequence objects.
- **hasSourceAttribute** - this attribute accepts the values "true" or "false". If "true" a sequence table is created in every database, otherwise the sequence table is created in the default database.
- **batchSize** - An optional value representing the size of a batch of primary key values. This primary key values batch will improve performance since it will allow the primary key generator to get the next value from the local batch instead of going to the database every time a new value is requested. This feature is very useful specially when inserting MithraList objects. This value must be a positive number. **Default is 10**.
- **initialValue** - specifies the initial value for the sequence. **Default is 1**.
- **incrementSize** - specifies the interval between sequence numbers. This value can be any positive or negative integer, but it cannot be 0. If this value is negative, then the sequence descends. If the increment is positive, then the sequence ascends. **Default is 1**.

Simulated sequence strategy example:

```
<MithraObject objectType="transactional">
    <PackageName>com.gs.fw.common.mithra.test.domain</PackageName>
    <ClassName>AccountTransaction</ClassName>
    <DefaultTable>ACCOUNT_TRANSACTION</DefaultTable>
    <SourceAttribute name="deskId" javaType="String"/>
    <Attribute name="transactionId"
               javaType="int"
               columnName="TRANSACTION_ID"
               primaryKey="true"
               primaryKeyGeneratorStrategy="SimulatedSequence">
         <SimulatedSequence sequenceName="AccountTransactionSequence"

  sequenceObjectFactoryName="com.gs.fw.common.mithra.test.domain.MithraTestSequenc
                          hasSourceAttribute="false"/>
    </Attribute>
    ...
</MithraObject>
```

Notice, since the values for batchSize, initialValue, and incrementSize where not declared, the sequence will be initialized using the default values mentioned above for those attributes.

There are some restrictions when using simulated sequences, for example, simulated sequences can be shared among Reladomo objects. Two or more Reladomo objects can use the same simulated sequence as long as the Reladomo objects have the same source attribute type, and that the values for batchSize, initialValue, and incrementalValue are the same in each Reladomo object definition.

In addition, Reladomo objects that have compound primary keys can use primary key generators for more than one of the primary key attributes. If so, the primary key generators can use the same strategy:

```
<MithraObject objectType="transactional">
    <PackageName>com.gs.fw.common.mithra.test.domain</PackageName>
    <ClassName>SpecialAccountMax</ClassName>
    <DefaultTable>SPECIAL_ACCOUNT_MAX</DefaultTable>
    <SourceAttribute name="deskId" javaType="String"/>
    <Attribute name="specialAccountId"
               javaType="long"
               columnName="SPECIAL_ACCOUNT_ID"
               primaryKey="true"
               primaryKeyGeneratorStrategy="Max"/>
    <Attribute name="accountSpecialCode"
               javaType="long"
               columnName="ACCOUNT_SPECIAL_CODE"
               primaryKey="true"
               primaryKeyGeneratorStrategy="Max"/>
     ...
</MithraObject>
```

Or different strategies:

```
<MithraObject objectType="transactional">
    <PackageName>com.gs.fw.common.mithra.test.domain</PackageName>
    <ClassName>SpecialAccount</ClassName>
    <DefaultTable>SPECIAL_ACCOUNT</DefaultTable>

    <SourceAttribute name="deskId" javaType="String"/>
    <Attribute name="specialAccountId"
               javaType="long"
               columnName="SPECIAL_ACCOUNT_ID"
               primaryKey="true"
               primaryKeyGeneratorStrategy="SimulatedSequence">
        <SimulatedSequence sequenceName="AccountSequence"

  sequenceObjectFactoryName="com.gs.fw.common.mithra.test.domain.MithraTestSequenc
                           hasSourceAttribute="false"
                           batchSize="2"
                           initialValue="100000000"
                           incrementSize="1"/>
    </Attribute>
    <Attribute name="accountSpecialCode"
               javaType="long"
               columnName="ACCOUNT_SPECIAL_CODE"
               primaryKey="true"
               primaryKeyGeneratorStrategy="Max"/>
    ...
```

```
</MithraObject>
```

3. Primary Key Generator Programming Model

Every transactional Reladomo object will provide a convenience method to generate and set the primary key value. In addition, this method will return the generated primary key value. The use of this method is optional since the Reladomo object will check the primary key values (for the primary keys that are using primary key generators) and generate the values if they have not been set when insert() is called on the MithraObject. The method will be of the form:

```
public <primaryKey attribute type> generateAndSet<primaryKey attribute
 name>();
```

The MithraList object does not provide any convenience method(s) to work with primary key generators. In the case of the SimulatedSequence strategy, the MithraList object will ensure that the batch has enough capacity for the list.

Here is an example of how to insert a MithraObject that uses a SimulatedSequence. In this case transactionId is the primary key for the AccountTransaction object:

```
...
AccountTransaction accountTransaction0 = new AccountTransaction();
accountTransaction0.setDeskId("A");
accountTransaction0.setTransactionDescription("Account Transaction
 0");
accountTransaction0.setTransactionDate(new
 Timestamp(System.currentTimeMillis()));
long transactionId0 =
 accountTransaction0.generateAndSetTransactionId();
accountTransaction0.insert();
...
```

Inserting a MithraList using a simulated sequence:

```
...
AccountTransactionList list1 = new AccountTransactionList();
AccountTransaction accountTransaction1 = null;
for (int i = 0; i < 25; i++)
{
    accountTransaction1 = new AccountTransaction();
    accountTransaction1.setTransactionDate(new
 Timestamp(System.currentTimeMillis()));
    accountTransaction1.setTransactionDescription("Transaction
 failure: "+i);
    accountTransaction1.setDeskId("A");
    list1.add(accountTransaction1);
}
list1.insertAll();
...
```

In this case the convenience method to generate and set the primary key is not being called. When the insertAll() is called on the MithraList, the list ensures that the primary key generator has a batch with enough capacity for the list. In addition, the MithraList will call insert on each MithraObject in the list and, as we mentioned before, the MithraObject will check, generate, and set the primary key values if necessary.

When Max strategy is used in a single-column or compound primary key, the MithraObject must be called within the context of a transaction, for example:

```
...
 MithraManagerProvider.getMithraManager().executeTransactionalCommand(
            new TransactionalCommand()
            {
                public Object executeTransaction(MithraTransaction
 tx) throws Throwable
                {
                    AccountTransactionMax transaction0 = new
 AccountTransactionMax();
                    transaction0.setTransactionDescription("New
 Account");
                    transaction0.setTransactionDate(new
 Timestamp(System.currentTimeMillis()));
                    transaction0.setDeskId("A");
                    transaction0.generateAndSetTransactionId();
                    transaction0.insert();
                    return null;
                }
            });
...
```

The same applies for a MithraList which MithraObject is using the Max strategy:

```
...
MithraManagerProvider.getMithraManager().executeTransactionalCommand(
    new TransactionalCommand()
    {
        public Object executeTransaction(MithraTransaction tx) throws
 Throwable
        {
            AccountTransactionMaxList list0 = new
 AccountTransactionMaxList();
            AccountTransactionMax transaction0 = null;

            for (int i = 0; i < 20; i++)
            {
                transaction0 = new AccountTransactionMax();
                transaction0.setTransactionDescription("New Account");
                transaction0.setTransactionDate(new
 Timestamp(System.currentTimeMillis()));
                transaction0.setDeskId("A");
            }
            list.insertAll();
            return null;
        }
    });
...
```