How To create and update Databases using SchemaManager and SchemaUpdateSnippets

Preliminary

The SchemaManager (`org.clazzes.jdbc2xml.schema.SchemaManager`) and SchemaEngine provide means to maintain the database scheme of an application, allowing you to add and delete tables, columns, relations and data in the database along application updates.

Schema History Table

A designated schema table, by default named SCHEMA_HISTORY, is used to keep track of the current scheme. It will be automatically at SchemaManager's first run. SCHEMA_HISTORY contains the following columns:

`VERSION`	`varchar(10), not null, primary key`
`DESCRIPTION`	`varchar(512), nullable`
`CREATION_DATE`	`date, nullable`
`SERIALNR`	`integer(5), not null`

Custom name for Schema History Table

In heterogenous environments as well as in heavily modularized software architectures a single database may be shared by multiple parties each requiring a couple of tables.

To allow multiple modules (applications, libraries, other OSGi bundles) to use JDBC2XML's SchemaManager concurrently within one database, as of JDBC 1.1.1 SchemaManager hold the name of the schema history table in an overwritable property, `versionHistoryTable`. See JDBCTOXML-11.

Project Configuration

SchemaManager requires a DataSource (`javax.sql.DataSource`) and a list of TableInfo (`org.clazzes.jdbc2xml.schema.TableInfo`) Objects, from which database structures will be created if an "empty" database is detected. Furthermore, an implementation of ISchemaEngine (`org.clazzes.jdbc2xml.schema.ISchemaEngine`) is required.

Optionally, a base version (default value 0.1.00) and a base description String (default "initial database schema") may be specified.

Database updates are passed as a Map<String, ISchemaUpdateSnippet> (org.clazzes.jdbc2xml.schema.ISchemaUpdateSnippet) - details see below.

To perform the operations, call `SchemaManager.start()`.

Configuration using Spring or OSGi/Blueprint

If you are using OSGi with Blueprint or Spring to set up your project, you can configure a SchemaManager instance by adding the following to your blueprint `services.xml` (or Spring configuration file):

```
<bp:reference id="dialectFactory" interface="org.clazzes.jdbc2xml.schema.IDialectFactory">
</bp:reference>

<bp:reference id="schemaEngineFactory" interface="org.clazzes.jdbc2xml.schema.ISchemaEngineFactory">
</bp:reference>

<bp:bean id="sqlDialect" factory-ref="dialectFactory" factory-method="newDialect">
        <bp:argument ref="jdbcUrl"><!-- bean jdbcUrl specified above -->
        </bp:argument>
</bp:bean>

<bp:bean id="schemaEngine" factory-ref="schemaEngineFactory" factory-method="newSchemaEngine">
        <bp:property name="dialect" ref="sqlDialect">
        </bp:property>
</bp:bean>


<bp:bean id="initialSchema" class="foo.schema.InitialSchema"></bp:bean>

<bp:bean id="databaseSetup" class="org.clazzes.jdbc2xml.schema.SchemaManager" init-method="start">
        <bp:property name="dataSource" ref="dataSource"></bp:property>
        <bp:property name="schemaEngine" ref="schemaEngine"></bp:property>
    <!-- optional but recommended: special name for schema history table: -->
    <bp:property name="versionHistoryTable" value="MYLIB_SCHEMA_HISTORY"/>
    <!-- optional but recommended: explicit first version -->
        <bp:property name="baseVersion" value="0.1.00" />
        <bp:property name="baseTables">
                <!-- List of TableDefinitions here (see below), typical: -->
        <bp:bean factory-ref="initialSchema" factory-method="getSchema" />
        </bp:property>
    <!-- Add Update-Snippets here, example for updates from 0.1.00 to 0.1.01 and on to 0.2.00
        <bp:property name="upateSnippets">
                <bp:map>
            <bp:entry key="0.1.00" value="foo.schema.SchemaUpdate_0_1_01"></bp:entry>
            <bp:entry key="0.1.01" value="foo.schema.SchemaUpdate_0_2_00"></bp:entry>
                </bp:map>
        </bp:property>
    -->
</bp:bean>
```

By default, JDBC2XML provides an implementation of IDialectFactory and ISchemaEngineFactory as an OSGi service or via ServiceRegistry for Spring.

Setting up an initial database schema

Inital Schema (Initial Table List)

To create an initial database schema, SchemaManager needs a list of TableInfo objects.

The recommended strategy is to create an InitialSchema class providing this list through a getter.

This is an example:

```
package foo.schema;

import java.sql.Types;
import java.util.Arrays;
import java.util.List;

import org.clazzes.jdbc2xml.schema.ColumnInfo;
import org.clazzes.jdbc2xml.schema.ForeignKeyInfo;
import org.clazzes.jdbc2xml.schema.PrimaryKeyInfo;
import org.clazzes.jdbc2xml.schema.TableInfo;

public class InitialSchema {

    private List<TableInfo> setup;

    public InitialSchema() {
        // Create a table
        TableInfo exampleTable = new TableInfo(TableDefs.TABLENAME_ADDRESSBOOK);
        exampleTable.setColumns(
            Arrays.asList(new ColumnInfo[] {
                new ColumnInfo(TableDefs.COL_ADDRESSBOOK_ID, Types.BIGINT, 20, null, false, null,true),
                new ColumnInfo(TableDefs.COL_ADDRESSBOOK_NAME, Types.VARCHAR, 256, null, false, null),
                new ColumnInfo(TableDefs.COL_ADDRESSBOOK_ADDRESS_REF, Types.BIGINT, 20, null, true, null),
                new ColumnInfo(TableDefs.COL_ADDRESSBOOK_BIRTHDAY, Types.DATE, 12, null, false, null)
        }));

        // Example for creating a primary key
        exampleTable.setPrimaryKey(
                new PrimaryKeyInfo("PK_EXAMPLE", COL_ADDRESSBOOK_ID)
        );

        // Example for creating a foreign key reference
        exampleTable.setForeignKeys(Arrays.asList(new ForeignKeyInfo[] {
                new ForeignKeyInfo("FK_EXAMPLE_ADDRESS", TableDefs.COL_ADDRESSBOOK_ADDRESS_REF, TableDefs.
TABLENAME_ADDRESSES, TableDefs.COL_ADDRESS_ID)
        }));

        // ...

        this.setup = Arrays.asList(
                exampleTable,
                // ...
        );

    }

    public List<TableInfo> getSchema() {
                return this.schema;
    }

}
```

TableDefs, a central place for table and column names

You may have notice the usage of `TableDefs.*` members.

Table and column names should never be re-typed everywhere as literals, it is highly recommended to use constants.

Putting these constants in a dedicated class, say `TableDef`, allows to use this as an easily accessible list of all tables and columns in the database.

This is an example:

```
package foo.schema;

public class TableDefs {

    // It is adviseable to provide the Strings used as names for tables and columns as constants,
    // so they can be reused savely to construct SQL statements

    // 0.1.00
    public static final String TABLENAME_ADDRESSBOOK = "ADDRESSBOOK";
    public static final String COL_ADDRESSBOOK_ID = "ID";
    public static final String COL_ADDRESSBOOK_NAME = "NAME";
    public static final String COL_ADDRESSBOOK_ADDRESS_REF = "ADDRESS";
    public static final String COL_ADDRESSBOOK_BIRTHDAY = "BIRTHDAY";
    // 0.1.01
        public static final String COL_ADDRESSBOOK_GENDER = "GENDER";

}
```

Triggering the creation of the initial schema

To trigger the creation of the initial schema when coming across an empty database, `InitialSchema.getSchema()` has to be injected into `SchemaManager.setBaseTables()` before calling `SchemaManager.start()`.

Using Blueprint/Spring, you can do this by inserting the following snippet in the bean definition for SchemaManager:

```
<bp:bean id="initialSchema" class="foo.schema.InitialSchema"></bp:bean>

<bp:property name="baseTables">
        <bp:bean factory-ref="initialSchema" factory-method="getSchema" />
</bp:property>
```

Updating a database schema with ISchemaUpdateSnippet

To update the database or it's content with schema updates, you must create a new implementation of ISchemaUpdateSnippet (`org.clazzes.jdbc2xml.schema.ISchemaUpdateSnippet`) for each consecutive update. SchemaManager takes a `Map<String, Class<? extends ISchemaUpdateSnippet>>` which contains the update classes keyed by the originating (e.g. previous) version.

An example for an implementation of a schema update snippet could look like this:

```
package foo.schema;

import java.sql.SQLException;
import java.sql.Types;
import java.util.Arrays;
import org.clazzes.jdbc2xml.schema.ColumnInfo;
import org.clazzes.jdbc2xml.schema.ISchemaEngine;
import org.clazzes.jdbc2xml.schema.ISchemaUpdateSnippet;
import org.clazzes.jdbc2xml.schema.PrimaryKeyInfo;
import org.clazzes.jdbc2xml.schema.TableInfo;

public class SchemaUpdate_0_1_01 implements ISchemaUpdateSnippet {

        // This is only accessed through the getter
        private static final String TARGET_VERSION = "0.1.01";

        @Override
        public String getTargetVersion() {
                return TARGET_VERSION;
        }

        @Override
        public String getUpdateComment() {
                return "Adding column "+TableDefs.COL_ADDRESSBOOK_GENDER+" to table "+TableDefs.
TABLENAME_ADDRESSBOOK+".";
        }

        @Override
        public void performUpdate(ISchemaEngine schemaEngine) throws SQLException {
                TableInfo ti = schemaEngine.fetchTableInfo(TableDefs.TABLENAME_ADDRESSBOOK, null);
                schemaEngine.addColumn(ti, new ColumnInfo(TableDefs.COL_ADDRESSBOOK_GENDER, Types.VARCHAR, 1,
null, true, null));
        }
}
```

The return values of `ISchemaUpdateSnippet.getTargetVersion()` and `ISchemaUpdateSnippet.getUpdateComment()` are written to the `SCHEMA_HISTORY` table. The update itself is performed in `ISchemaUpdateSnippet.performUpdate()`. In the above example, it adds a column called `GENDER` to the ADDRESSBOOK table created via the InitialSchema class above.

To add an entire table you would use the `ISchemaEngine.createTable()` method, like this:

```
@Override
public void performUpdate(ISchemaEngine schemaEngine) throws SQLException {
        TableInfo tiGroup = new TableInfo(TB_GROUP);
        tiGroup.setColumns(Arrays.asList(new ColumnInfo[] {
                new ColumnInfo(TableDefs.COL_ID, Types.VARCHAR, 36, null, false, null),
                new ColumnInfo(TableDefs.COL_NAME, Types.VARCHAR, 100, null, false, null),
                new ColumnInfo(TableDefs.COL_DESCRIPTION, Types.VARCHAR, 512, null, true, null)
        }));
        tiGroup.setPrimaryKey(new PrimaryKeyInfo(PK_GROUP, TableDefs.COL_ID));
        tiGroup.setIndices(Arrays.asList(new IndexInfo(IDX_GROUP_01, TableDefs.COL_NAME, true, null)));

        schemaEngine.createTable(tiGroup, true);
}
```

Executing a PreparedStatement also works, using ISchemaEngine.getConnection() to retrieve the database connection:

```
@Override
public void performUpdate(ISchemaEngine schemaEngine) throws SQLException {
        String sql = "UPDATE "+TableDefs.TB_EXAMPLE_TABLE_NAME+" SET "+TableDefs.COL_EXAMPLE_NAME+"=?";

        PreparedStatement ps = schemaEngine.getConnection().prepareStatement(sql);

        ps.setNull(1, Types.VARCHAR);

        ps.execute();
}
```

To create the map of updates in Blueprint/Spring and inject them into SchemaManager, use the following xml-Snippet:

```
<!-- SchemaManager bean definition starts here ... -->
<bp:property name="upateSnippets">
        <bp:map>
                <bp:entry key="0.1.00" value="org.clazzes.example.jdbc2xml.updates.SchemaUpdate0_1_01"></bp:
entry>
                <!-- more entries come here: "key" is the schema version to update, "value" the qualified
classname of the schema update -->
        </bp:map>
</bp:property>
<!-- ... and continues here -->
```

Schema Maintainance Strategies

The JDBC2XML Schema management tools allow for 2 different strategies:

Frozen (Initial) Table List

The legacy strategy is:

- At the start of a project, create and use `InitalSchema.java`
- After the first commit, `InitalSchema` are considered frozen, all changes go into `SchemaUpdates`, up to one update per source code commit

Advantage: Rock solid.

Disadvantage: No place to look for the complete and exact current scheme, except actual databases. `TableDefs.java` provide some information, but may become confusing in the long term.

Possible but dangerous: Evolving (Initial) Table List

To keep the (Initial) Schema up do date, one might use this strategy:

- keep the `InitalSchema` up to date, so an empty database always gets the current scheme in one shot
- `SchemaUpdates` are only applied to existing databases

Advantage: Immediate overview over exact current scheme.

Disadvantage: Very real danger of messing something up, because

- schema updates have to be coded in 2 different places in 2 different ways
- the bean definition has to be maintained in 2 places but just 1

Conclusion: **DO NOT DO THIS**. This strategy may be ok in very early stages, but at some point it has to be

Recommendation: Freeze Initial Table Definition not later than the first Release Candidate (RC)

It may be ok to start a new project using a fast changing (Initial) Table List.

But, please, freeze it at some point. Once the first test server is setup up, internally or at a friendly customer, the Frozen Initial Table List Strategy is the only valid one!

Real world Example

This HowTo is currently evolving while an additional developer gets acostumed to the SchemaEngine, for developing SDS' org.clazzes.sds.impl.schema package which is **work in progress**!