Using DdlUtils' API

Table of contents

1. The model

At the core of DdlUtils lies the database model in package `org.apache.ddlutils.model`. It consists of classes that represent the database schema:

UML diagram of the database model

Using the appropriate methods on these classes, you can build the database model manually, or you read it from XML or from the database (see the next paragraphs for details). More info about the classes can be found in the javadoc.

2. Reading from XML

Most of the time you have a schema in an XML file, and you want to read it into memory in order to do something with it. This is quite easily accomplished with a few lines of code:

```
import org.apache.ddlutils.io.DatabaseIO;
import org.apache.ddlutils.model.Database;

...

public Database readDatabaseFromXML(String fileName)
{
    return new DatabaseIO().read(fileName);
}
```

3. Writing to XML

Writing a model to XML is just as easy as reading from XML:

```
import org.apache.ddlutils.io.DatabaseIO;
import org.apache.ddlutils.model.Database;

...

public void writeDatabaseToXML(Database db, String fileName)
{
    new DatabaseIO().write(db, fileName);
}
```

4. Reading the model from a live database

Reading the database model from a live database is only slightly more involved because we first need to create a platform instance via the data source pointing to the database:

```
import javax.sql.DataSource;
import org.apache.ddlutils.Platform;
import org.apache.ddlutils.PlatformFactory;
import org.apache.ddlutils.model.Database;

...

public Database readDatabase(DataSource dataSource)
{
    Platform platform =
PlatformFactory.createNewPlatformInstance(dataSource);

    return platform.readModelFromDatabase("model");
}
```

5. Changing a database

Changing a database essentially means one of two things: resetting a database to the schema defined by the model, or altering the database to have the same model. The key difference is that with the latter, data in the database is retained as much as possible. Only major changes to the table structure or type-incompatible alterations of columns will result in loss of data, most changes will simply retain the data.

Note:
Whether a change of e.g. a column type affects the data contained in the table, depends on the database that you use. Most databases are able to convert between different datatypes and will apply these conversions when the column type is changed.

Both types of modification differ only in how the SQL is created, the general procedure is the same: create the sql and execute it:

```
import javax.sql.DataSource;
import org.apache.ddlutils.Platform;
import org.apache.ddlutils.PlatformFactory;
import org.apache.ddlutils.model.Database;

...

public void changeDatabase(DataSource dataSource,
                           Database   targetModel,
                           boolean    alterDb)
{
    Platform platform =
PlatformFactory.createNewPlatformInstance(dataSource);

    if (alterDb)
    {
        platform.alterTables(targetModel, false);
    }
    else
    {
        platform.createTables(targetModel, true, false);
    }
}
```

Note:
Databases like Oracle allow for more than one separate schema in one database. To cater for these databases, there are

variants of these methods where you can specify the catalog and schema.

6. Inserting data into a database

DdlUtils provides a simple way to insert data into the database. For this it uses DynaBeans which are essentially dynamically created beans with variable properties. For each table defined by the database model, a so-called dyna class is created that represents the table with its columns. Of this dyna class, instances - the dyna beans - are then created which can be inserted by DdlUtils into the database:

```
import javax.sql.DataSource;
import org.apache.commons.beanutils.DynaBean;
import org.apache.ddlutils.Platform;
import org.apache.ddlutils.PlatformFactory;
import org.apache.ddlutils.model.Database;

...

public void insertData(DataSource dataSource,
                       Database   database)
{
    Platform platform =
PlatformFactory.createNewPlatformInstance(dataSource);

    // "author" is a table of the model
    DynaBean author = database.createDynaBeanFor("author", false);

    // "name" and "whatever" are columns of table "author"
    author.set("name",     "James");
    author.set("whatever", new Integer(1234));

    platform.insert(database, author);
}
```

7. Getting data from a database

In the same way as inserting data into a database, DdlUtils uses dyna beans for retrieving data from the database. You issue a SQL select against the database and get dyna beans back. This means that the table that the select goes against, has to be part of the database model used by DdlUtils.

In the following sample, the titles of all books in the database are printed to stdout:

```
import java.util.ArrayList;
import java.util.Iterator;
import javax.sql.DataSource;
import org.apache.commons.beanutils.DynaBean;
import org.apache.ddlutils.Platform;
import org.apache.ddlutils.PlatformFactory;
import org.apache.ddlutils.model.Database;
import org.apache.ddlutils.model.Table;
```

```
...
public void dumpBooks(DataSource dataSource,
                      Database   database)
{
    Platform  platform =
PlatformFactory.createNewPlatformInstance(dataSource);
    ArrayList params   = new ArrayList();

    params.add("Some title");

    Iterator it = platform.query(database,
                                 "select * from book where title = ?",
                                 params,
                                 new Table[] {
database.findTable("book") });

    while (it.hasNext())
    {
        DynaBean book = (DynaBean)it.next();

        System.out.println(book.get("title"));
    }
}
```

There are two things to note in this sample code:

First, we specified so-called query hints in the call to the `query`. Query hints are an array of tables whose columns are used by the query statement. The reason why they should be used is that not all databases provide sufficient information in the JDBC result set object to determine the table to which a column belongs to. Since this info is need by DdlUtils to properly extract the value and convert it to the corresponding Java type, it is safer to specify these hints. What DdlUtils does in this case, is to search for a column of that name within the specified tables and use the mapping for this column. This of course can fail if you use aliases in the query statement (and the JDBC driver handles them in a strange way), or if more than one table has a column of this name. But in most cases you'll get the expected results.

The other thing to note is that DdlUtils does not parse the query statement. This means that if you use delimited identifier mode (i.e. identifiers can contain whitespaces, non-alphanumeric characters etc., but they also need to be enclosed in double quotes), then you'll have to specify the query statement accordingly - DdlUtils won't do that for you. If you'd like to be on the safe side, then you could write the above statement like this:

```
import java.util.ArrayList;
import java.util.Iterator;
import javax.sql.DataSource;
import org.apache.commons.beanutils.DynaBean;
import org.apache.ddlutils.Platform;
import org.apache.ddlutils.PlatformFactory;
import org.apache.ddlutils.model.Database;
import org.apache.ddlutils.model.Table;
```

```
...

public void dumpBooks(DataSource dataSource,
                      Database   database)
{
    Platform  platform =
PlatformFactory.createNewPlatformInstance(dataSource);
    ArrayList params   = new ArrayList();
    String    sql;

    params.add("Some title");

    if (platform.isDelimitedIdentifierModeOn())
    {
        sql = "select * from \"book\" where \"title\" = ?";
    }
    else
    {
        sql = "select * from book where title = ?";
    }

    Iterator it = platform.query(database,
                                 sql,
                                 params,
                                 new Table[] {
database.findTable("book") });

    while (it.hasNext())
    {
        DynaBean book = (DynaBean)it.next();

        System.out.println(book.get("title"));
    }
}
```