The version generator

1. The version generator

A versioned object is an object with a composed primary key: We call the first part the version ID and the second part the version number. Objects with the same ID are called versions of the object. If the versioned object is stored in a single SQL table, a new version can obviously be derived as follows:

1. Read the row containing the base version.
2. Create a copy of the row in memory.
3. Update the version number in the in-memory clone.
4. Insert the clone into the database.

There is nothing special. Things look quite different, if we have multiple tables. For example, suggest that we have a database with the tables "Organization", "Person", and "Email Address". Any organization might have associated persons as contacts. Organizations and persons, however, might have an unlimited number of email addresses. Foreign keys are used to join the different tables. The organization can be viewed as a collection of a row in "Organization" and zero or more rows in "Person". How about creating a new version of an organization? This is not so convenient:

1. Read the row containing the organization and clone it, following the above steps.
2. The cloned row doesn't have any associated email addresses, but the base organization probably does. In other words, we need to read the relevant email addresses into memory, update the organization reference and create new instances of the in-memory email addresses, possibly creating a new ID.
3. The cloned organization doesn't have any associated persons, but the base organization probably does. You guess the following: We need to clone the persons as well, updating references and assigning new ID's.
4. We are not yet done: The persons might be referenced by email adresses. So we continue ... you guess.

The version generator is able to create Java code for you, which is doing exactly the above. It takes as input a list of tables and an ID generation procedure for any table, and that's it.

2. Limitations

The current version of the version generator has some restrictions. In particular the following

rules apply:

1. Let `T1`, `T2`, `T3`, ... be the tables being cloned.
2. No table may reference itself. For example, `T3` must not have a reference to `T3`.
3. No table may have forward references. A forward reference is a foreign key to another table, which is positioned behind itself in the table. For example, `T2` and `T3` may reference `T1`, but neither `T2` nor `T3` may reference `T1` nor may `T3` reference `T2`.
4. All foreign keys must reference the primary key. As a consequence, each referenced table must have a primary key.

In essence, the requirements are, that there are no direct or indirect circular references and that there is a set of columns that allow to remember, which rows already have been cloned.

3. Column updaters

Column updaters are used to perform the required modifications for the in-memory rows. For example, there is a column updater that updates a foreign keys value from an object that has already been cloned to its clone. Column updaters are implementing the interface VersionGenerator.ColumnUpdater.

The main thing to understand when writing a column updater is that it is not invoked at runtime, but as a part of the source generation stage. It emits Java code taking an object array as input and modifying the array. As an example, we demonstrate a very basic column updater, which is simply incrementing the version number. We assume that the version number is an integer, stored in the tables second column:

```
public class VerNumIncrementer implements VersionGenerator.ColumnUpdater {
  public void update(JavaMethod pMethod,
                     VersionGenerator.TableInfo pTableInfo,
                     DirectAccessible pConnection,
                     DirectAccessible pMap,
                     DirectAccessible pRow) {
    pMethod.addLine(pRow, "[2] = new Integer(((Integer) ",
                    pRow, "[2]).intValue()+1);");
  }
}
```

The column updaters arguments are as follows:

pMethod
The method being generated.

pTableInfo
Holds the generators view of the table being cloned by the method `pMethod`. The most important property is the `table` property, which returns an instance of Table with the table name and column list.

pConnection
A variable name which is holding an open database connection in the generated source.
pMap
A variable name with a map used to note the column which have already been cloned. The tables keys are the primary keys before cloning, and the tables values are the keys after cloning. The map is used to adjust foreign key references.
pRow
A variable name representing an array of objects, one for any column in the table. The object types correspond to the column data types: For example, a VARCHAR or CHAR column is mapped to a String, INT, or BIGINT, columns are mapped to Integer and Long, respectively, and so on.

4. Using the version generator

To use the version generator, you need a set of tables (subject to the limitations described above). For any table, you also need a ColumnUpdater. If you have that, the generator is invoked like this:

```
import org.apache.ws.jaxme.js.JavaQNameImpl;
import org.apache.ws.jaxme.js.JavaSource;
import org.apache.ws.jaxme.js.JavaSourceFactory;
import org.apache.ws.jaxme.js.pattern.VersionGenerator;
import org.apache.ws.jaxme.js.pattern.VersionGenerator.ColumnUpdater;
import org.apache.ws.jaxme.sqls.Table;

Table[] tables;             // Supplied by you
ColumnUpdater updaters;     // Supplied by you, same length than tables array

JavaSourceFactory factory = new JavaSourceFactory();
VersionGenerator vgen = new VersionGenerator();
vgen.setGeneratingLogging(true);
for (int i = 0;  i < tables.length;  i++) {
  vgen.addTable(tables[i], updaters[i]);
}
JavaSource js = factory.newJavaSource(JavaQNameImpl.getInstance("com.foo", "Bar"),
                                      JavaSource.PUBLIC);
vgen.getCloneMethod(js);
```

You might replace the VersionGenerator with a subclass, because the above code would emit logging statements using the org.apache.ws.jaxme.logging package. If you do not favour this, replace the methods `logEntering`, `logExiting`, `logFinest`, `logFinestEntering`, and `logFinestEntering` with your own implementations.