CHAPTER 11

Schema Management

One of Hibernate's most useful features is the automatic generation of schema manipulation commands. This feature, sometimes referred to as the ability to generate Data Definition Language (DDL) scripts, makes it possible (given a valid `*.hbm.xml` file) to create, update, and even drop tables in a target database. You can do this at runtime, during development, or via scripts generated for later use by a system administrator—an invaluable capability if you expect to support multiple target databases (during either development or deployment) or have a high degree of database schema change.

Hibernate supports two basic forms of DDL generation, **update** and **export**. Update is generally used within an application, targeting a specific database that may already contain a portion of the schema (and hence application data) but is missing schema components required by a new application. Export is used to generate the schema from scratch; it is especially useful if the application is not allowed to directly execute DDL (because, say, a database administrator is expected to perform these tasks).

Updating an Existing Schema

The tool `net.sf.hibernate.tool.hbm2ddl.SchemaUpdate` allows an application to bring a schema up to date with the expected schema based on a set of `*.hbm.xml` files. Typically, this is used to address a situation in which an incremental update to an application requires a relatively minor change, such as a new property.

Consider, for example, an application with a user object (and corresponding user table). You've decided to add a property to the user object to track the user's country code (previously the application only supported U.S. addresses). You make the change to your `*.hbm.xml` file and the corresponding Java code, and now would like to reflect the change in the deployed database. This can be done

either from the command line, from an Ant task, or embedded within your application.

`SchemaUpdate` relies heavily on metadata returned by a database driver to understand the existing schema. For this reason, the ability of `SchemaUpdate` to operate properly can vary from driver to driver (and database to database). If you are unable to use `SchemaUpdate` with your preferred database, you may wish to use the `SchemaExport` tool (described later in this chapter) instead.

Schema Updates from within an Application

Listing 11.1 shows an example of `SchemaUpdate` embedded within an application. Note that a `Configuration` object is required, but a `Session` is not (obviously, you should perform any schema manipulation before working with the database).

Listing 11.1 `SchemaUpdate` Example

```
package com.cascadetg.ch11;

/** Various Hibernate-related imports */
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.*;
import net.sf.hibernate.tool.hbm2ddl.SchemaUpdate;

public class SchemaUpdaterExample
{

    /** We use this session factory to create our sessions */
    public static SessionFactory sessionFactory;

    public static void main(String[] args)
    {
        initialization();
    }

    /**
     * Loads the Hibernate configuration information, sets up
     * the database and the Hibernate session factory.
     */
    public static void initialization()
    {
        System.out.println("initialization");
        try
```

Listing 11.1 `SchemaUpdate` Example (*continued*)

```
        {
            Configuration myConfiguration = new
                 Configuration();

            myConfiguration
                    .addClass(com.cascadetg.ch03.Owner.class);
            myConfiguration
                    .addClass(com.cascadetg.ch03.Artifact
                         .class);

            // Load the *.hbm.xml files as set in the
            // config, and set the dialect.
            new SchemaUpdate(myConfiguration)
                    .execute(true, true);

        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Command Line Schema Updates

To use `SchemaUpdate` from the command line, you have to use the command `java net.sf.hibernate.tool.hbm2ddl.SchemaUpdate`, passing in one or more of the command-line options shown in Table 11.1, followed by the path to the `*.hbm.xml` files.

Table 11.1. SchemaUpdate Command-Line Options

`--quiet`	Echo the script to the console
`--properties=filename.properties`	Specify the `hibernate.properties` file
`--config=filename.cfg.xml`	Specify the `hibernate.cfg.xml` file
`--text`	Do not execute the update
`--naming=fully.qualified.class.name`	Specify the naming policy to use (Hibernate ships with `net.sf.hibernate.cfg.DefaultNamingStrategy` (prefers mixed case) and `net.sf.hibernate.cfg.ImprovedNamingStrategy` (prefers underscores).

Ant Task Schema Updates

In addition to the runtime and command-line options, you can also use a build-time Ant task, as shown in Listing 11.2.

Listing 11.2 SchemaUpdate Ant Task

```
<target name="schemaupdate">
    <taskdef name="schemaupdate"
        classname="net.sf.hibernate.tool.hbm2ddl
             .SchemaUpdateTask"
        classpathref="class.path"/>

    <schemaupdate
        properties="hibernate.properties"
        quiet="no">
        <fileset dir="src">
            <include name="**/*.hbm.xml"/>
        </fileset>
    </schemaupdate>
</target>
```

Generating Update and Drop Scripts

Hibernate also includes the `net.sf.hibernate.tool.hbm2ddl.SchemaExport` tool, which allows you to generate scripts for generating a schema (optionally generating `DROP` statements as well). The `SchemaExport` tool has several advantages over the `SchemaUpdate`:

- It can be run at development time, even if you don't have access to the target database.
- It doesn't rely on driver metadata.
- It may be necessary if your application's database connection is not allowed to perform DDL.
- It allows you to send the application's database requirements to a database administrator.

Like `SchemaUpdate`, `SchemaExport` can be run from the command line or from Ant. An example is also shown of how to use `SchemaExport` to simultaneously generate scripts for several databases.

Table 11.2. SchemaExport Command-Line Options

Option	Description
`--quiet`	Don't output the script to the console.
`--drop`	Only generate drop-table statements.
`--text`	Generate script but don't perform against the database.
`--output=my_schema.sql`	Specify the file name to output the script.
`--config=hibernate.cfg.xml`	Specify the Hibernate configuration XML file.
`--properties=hibernate.properties`	Specify the Hibernate configuration properties from a file.
`--format`	Format the generated SQL nicely in the script.
`--delimiter=;`	Set an end-of-line delimiter for the script.

Command-Line Script Generation

You can run `SchemaExport` from the command line using the command `java net.sf.hibernate.tool.hbm2ddl.SchemaExport options mapping_files`. The possible options are as shown in Table 11.2.

Ant Task Script Generation

Listing 11.3 shows an Ant task that can be used to generate a script. The meaning of the options is as shown in Table 11.2.

Listing 11.3 SchemaExport Ant Task

```
<target name="schemaexport">
    <taskdef name="schemaexport"
        classname="net.sf.hibernate.tool.hbm2ddl
             .SchemaExportTask"
        classpathref="class.path"/>

    <schemaexport
        properties="hibernate.properties"
        quiet="no"
        text="no"
        drop="no"
        delimiter=";"
        output="schema-export.sql">
        <fileset dir="src">
            <include name="**/*.hbm.xml"/>
```

(*continues*)

Listing 11.3 SchemaExport Ant Task (*continued*)

```
        </fileset>
    </schemaexport>
</target>
```

Generating Multiple Scripts

Hibernate has the advantageous ability of making it easier to support a wide range of databases. By taking advantage of SchemaExport, you can generate schema generation scripts for a wide variety of databases at development time and include them in the application distribution. This is obviously no substitute for testing, as Hibernate relies on the underlying database for a wide variety of features, but it can be helpful if you are interested in using an application with a new database.

Table 6.5 shows the list of dialects included with Hibernate 2.1.2. The sample code shown in Listing 11.4 takes advantage of this list to generate a set of schema generation scripts for a wide suite of databases. The suite of dialects is looped through, with an attempt made to generate scripts for the sample application shown in Chapter 3.

Listing 11.4 Generating Multiple Scripts

```
package com.cascadetg.ch11;

/** Various Hibernate-related imports */
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.*;
import net.sf.hibernate.tool.hbm2ddl.SchemaExport;
import java.util.Properties;

public class SchemaGeneratorExample
{
    // System constants for the current platform directory
         token
    static String fileSep =
         System.getProperty("file.separator");

    /** We use this session factory to create our sessions */
    public static SessionFactory sessionFactory;

    static String[] db_dialects =
    { "DB2", //
            "net.sf.hibernate.dialect.DB2Dialect", //
            "DB2400", //
```

Listing 11.4 Generating Multiple Scripts

```
            "net.sf.hibernate.dialect.DB2400Dialect", //
            "Firebird", //
            "net.sf.hibernate.dialect.FirebirdDialect", //
            "FrontBase", //
            "net.sf.hibernate.dialect.FrontBaseDialect", //
            "Generic", //
            "net.sf.hibernate.dialect.GenericDialect", //
            "HypersonicSQL", //
            "net.sf.hibernate.dialect.HSQLDialect", //
            "Informix", //
            "net.sf.hibernate.dialect.InformixDialect", //
            "Informix9", //
            "net.sf.hibernate.dialect.Informix9Dialect", //
            "Ingres", //
            "net.sf.hibernate.dialect.IngresDialect", //
            "Interbase", //
            "net.sf.hibernate.dialect.InterbaseDialect", //
            "Mckoi SQL", //
            "net.sf.hibernate.dialect.MckoiDialect", //
            "Microsoft SQL Server", //
            "net.sf.hibernate.dialect.SQLServerDialect", //
            "MySQL", //
            "net.sf.hibernate.dialect.MySQLDialect", //
            "Oracle 9", //
            "net.sf.hibernate.dialect.Oracle9Dialect", //
            "Oracle", //
            "net.sf.hibernate.dialect.OracleDialect", //
            "Pointbase", //
            "net.sf.hibernate.dialect.PointbaseDialect", //
            "PostgreSQL", //
            "net.sf.hibernate.dialect.PostgreSQLDialect", //
            "Progress", //
            "net.sf.hibernate.dialect.ProgressDialect", //
            "SAP DB", //
            "net.sf.hibernate.dialect.SAPDBDialect", //
            "Sybase Anywhere", //
            "net.sf.hibernate.dialect.SybaseAnywhereDialect",
            "Sybase 11.9.2", //
            "net.sf.hibernate.dialect.Sybase11_9_2Dialect", //
            "Sybase", //
            "net.sf.hibernate.dialect.SybaseDialect",};

    public static void main(String[] args)
    {
```

(*continues*)

Listing 11.4 Generating Multiple Scripts (*continued*)

```
        initialization();
    }

    /**
     * Loads the Hibernate configuration information, sets up the
     * database and the Hibernate session factory.
     */
    public static void initialization()
    {
        System.out.println("initialization");
        try
        {
            Configuration myConfiguration = new
                 Configuration();

            myConfiguration
                    .addClass(com.cascadetg.ch03.Owner.class);
            myConfiguration
                    .addClass(com.cascadetg.ch03.Artifact
                         .class);

            Properties myProperties = new Properties();

            for (int i = 0; i < db_dialects.length; i = i + 2)
            {
                String dialect_name = db_dialects[i];
                String dialect_class = db_dialects[i + 1];

                String dialect_file =
                     dialect_name.toLowerCase();
                dialect_file = dialect_file.replace(' ', '_');
                dialect_file += (".sql");

                String path = "com" + fileSep + "cascadetg"
                        + fileSep + "ch11" + fileSep;

                System.out.println("Generating " +
                     dialect_name);

                // Note that this is the only Hibernate property
                // set.  In particular, there is no JDBC
                // connectivity data, nor are we specifying a
                // driver!
                myProperties.put("hibernate.dialect",
                        dialect_class);
```

Listing 11.4 Generating Multiple Scripts (*continued*)

```
                try
                {
                    // Load the *.hbm.xml files as set in the
                    // config, and set the dialect.
                    SchemaExport mySchemaExport = new
                         SchemaExport(
                            myConfiguration, myProperties);

                    mySchemaExport.setDelimiter(";");

                    // Despite the name, the generated create
                    // scripts WILL include drop statements at
                    // the top of the script!
                    mySchemaExport.setOutputFile(path + "create_"
                            + dialect_file);
                    mySchemaExport.create(false, false) ;

                    // Generates DROP statements only
                    mySchemaExport.setOutputFile(path + "drop_"
                            + dialect_file);
                    mySchemaExport.drop(false, false);

                    System.out.println(dialect_name + " OK.");

                } catch (Exception e)
                {
                    System.out.println(e.getMessage());
                }
            }

        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Running the application in Listing 11.4 produces the results shown in Listing 11.5. As can be seen, the `native` identity generator is not supported by many databases. For broader support, a Hibernate-driven identity generator would be a better choice to support a wider range of databases (for more information on identity generation, see Chapter 6). Figure 11.1 shows the resulting schema script files.

Listing 11.5 Generating Multiple Scripts

```
initialization
Generating DB2
DB2 OK.
Generating DB2400
DB2400 OK.
Generating Firebird
Dialect does not support identity-key generation
Generating FrontBase
Dialect does not support identity-key generation
Generating Generic
Dialect does not support identity-key generation
Generating HypersonicSQL
HypersonicSQL OK.
Generating Informix
Informix OK.
Generating Informix9
Informix9 OK.
Generating Ingres
Dialect does not support identity-key generation
Generating Interbase
Dialect does not support identity-key generation
Generating Mckoi SQL
Dialect does not support identity-key generation
Generating Microsoft SQL Server
Microsoft SQL Server OK.
Generating MySQL
MySQL OK.
Generating Oracle 9
Dialect does not support identity-key generation
Generating Oracle
Dialect does not support identity-key generation
Generating Pointbase
Dialect does not support identity-key generation
Generating PostgreSQL
Dialect does not support identity-key generation
Generating Progress
Dialect does not support identity-key generation
Generating SAP DB
Dialect does not support identity-key generation
Generating Sybase Anywhere
Sybase Anywhere OK.
Generating Sybase 11.9.2
Sybase 11.9.2 OK.
Generating Sybase
Sybase OK.
```

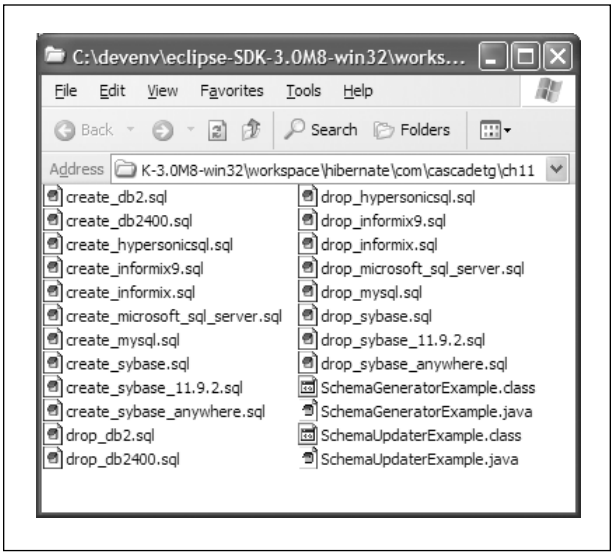

Figure 11.1. Generated Schema Scripts