Hibernate Tutorial 01 Object/Relational Mapping

By Gary Mak
hibernatetutorials@metaarchit.com
September 2006

1. Installation

1.1. Installing JDK

JDK is an essential toolkit provided for Java application development. You can go to http://java.sun.com/j2se/1.5.0/download.jsp to download JDK 5.0. Install it into a folder say “C:\jdk1.5.0”.

1.2. Installing Eclipse Web Tools Platform (WTP)

Eclipse is an IDE for developing Java application. WTP provides some tools for Web and J2EE development, such as XML editor and SQL editor. You can go to http://www.eclipse.org/webtools/ and download WTP All-in-one 1.0. Unzip it into a folder say “C:\eclipse”.

1.3. Installing HSQLDB

HSQLDB is an open source SQL relational database engine written in Java. You can go to http://www.hsqldb.org/ and download HSQLDB 1.8.0. Unzip it into a folder say “C:\hsqldb”.

1.4. Installing DbVisualizer

DbVisualizer is a Java-based visual database development tool. You can go to http://www.minq.se/products/dbvis/download.html and download DbVisualizer 5.0 free version. Unzip it into a folder say “C:\ DbVisualizer-5.0”

2. Setting up the database

Suppose we need to develop an online bookshop application. It will use relational database to store its application data.

2.1. Creating a HSQLDB database instance

To create a new HSQLDB database instance, use the following command. Some files (BookShopDB.*) will be created for the first time running this command.

```
java -cp lib/hsqldb.jar org.hsqldb.Server -database.0 BookShopDB -dbname.0 BookShopDB
```

Use DbVisualizer to connect this database using the following properties:

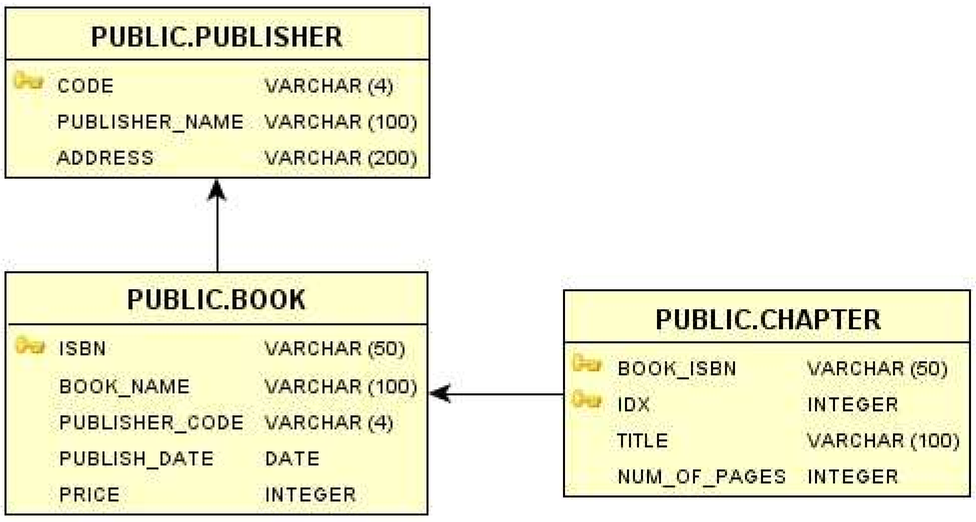
Driver (JDBC)	HSQLDB Server
Database URL	jdbc:hsqldb:hsql://localhost/BookShopDB
Userid	sa
Password	<empty>

2.2. Creating the tables (relational model)

For the first step, we create the tables for our online bookshop using the following SQL statements:

```
CREATE TABLE PUBLISHER (
    CODE                VARCHAR(4)      NOT NULL,
    PUBLISHER_NAME      VARCHAR(100)    NOT NULL,
    ADDRESS             VARCHAR(200),
    PRIMARY KEY (CODE)
);

CREATE TABLE BOOK (
    ISBN                VARCHAR(50)     NOT NULL,
    BOOK_NAME           VARCHAR(100)    NOT NULL,
    PUBLISHER_CODE      VARCHAR(4),
    PUBLISH_DATE        DATE,
    PRICE               INT,
    PRIMARY KEY (ISBN),
    FOREIGN KEY (PUBLISHER_CODE) REFERENCES PUBLISHER
);

CREATE TABLE CHAPTER (
    BOOK_ISBN           VARCHAR(50)     NOT NULL,
    IDX                 INT             NOT NULL,
    TITLE               VARCHAR(100)    NOT NULL,
    NUM_OF_PAGES        INT,
    PRIMARY KEY (BOOK_ISBN, IDX),
    FOREIGN KEY (BOOK_ISBN) REFERENCES BOOK
);
```

We can use DbVisualizer to generate the following diagram for our schema. It is the “relational model” for our online bookshop.



Next, let's input some data for these tables using the following SQL statements:

```
INSERT INTO PUBLISHER (CODE, PUBLISHER_NAME, ADDRESS) VALUES ('MANN', 'Manning', 'Address for
Manning');

INSERT INTO BOOK (ISBN, BOOK_NAME, PUBLISHER_CODE, PUBLISH_DATE, PRICE) VALUES ('1932394419',
'Hibernate Quickly', 'MANN', '2005-08-01', 35);

INSERT INTO CHAPTER (BOOK_ISBN, IDX, TITLE, NUM_OF_PAGES) VALUES ('1932394419', 1, 'Why
Hibernate?', 25);

INSERT INTO CHAPTER (BOOK_ISBN, IDX, TITLE, NUM_OF_PAGES) VALUES ('1932394419', 2, 'Hibernate
basics', 37);
```

3. Programming with basic JDBC

The traditional way of accessing a relational database is to use JDBC (Java Database Connectivity).

3.1. Creating a Eclipse project

For the first time of developing our Java application, we create a project "BookShop" in Eclipse and add the HSQLDB JDBC driver to its Java build path. The location of that driver is ${HSQLDB_INSTALL_DIR}/lib/hsqldb.jar.

3.2. JDBC initialization and cleanup

We must first load the JDBC driver and create a connection to that database before we can execute any SQL statements. Be careful, you must remember to close that connection in any case (whether an exception is raised or not).

```
Class.forName("org.hsqldb.jdbcDriver");
Connection connection = DriverManager.getConnection(
    "jdbc:hsqldb:hsql://localhost/BookShopDB", "sa", "");
try {
    // Using the connection to query or update database
} finally {
    connection.close();
}
```

3.3. Using JDBC to query database

For demo purpose, we query for a book whose ISBN is “1932394419”. Below is the JDBC code fragment for this task:

```
PreparedStatement stmt = connection.prepareStatement("SELECT * FROM BOOK WHERE ISBN = ?");
stmt.setString(1, "1932394419");
ResultSet rs = stmt.executeQuery();
while (rs.next()) {
    System.out.println("ISBN : " + rs.getString("ISBN"));
    System.out.println("Book Name : " + rs.getString("BOOK_NAME"));
    System.out.println("Publisher Code : " + rs.getString("PUBLISHER_CODE"));
    System.out.println("Publish Date : " + rs.getDate("PUBLISH_DATE"));
    System.out.println("Price : " + rs.getInt("PRICE"));
    System.out.println();
}
rs.close();
stmt.close();
```

3.4. Using JDBC to update database

For demo purpose, we update the name of the book whose ISBN is “1932394419”. Below is the JDBC code fragment for this task:

```
PreparedStatement stmt = connection.prepareStatement(
    "UPDATE BOOK SET BOOK_NAME = ? WHERE ISBN = ?");
stmt.setString(1, "Hibernate Quickly 2nd Edition");
stmt.setString(2, "1932394419");
int count = stmt.executeUpdate();
System.out.println("Updated count : " + count);
stmt.close();
```

4. Object/Relational Mapping

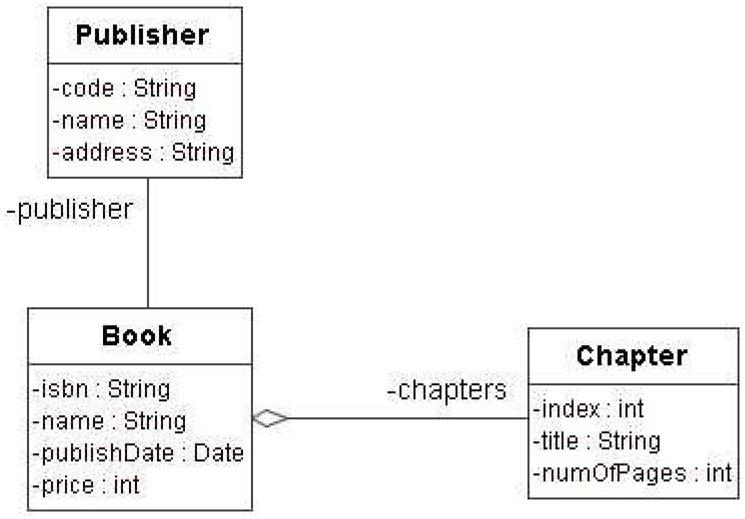
In the previous sections, we have learned how to use JDBC to access relational database. As widely known, Java is an Object-Oriented programming language. That means we should use an “object model” to represent our domain concepts.

4.1. Object model

We use normal JavaBeans to build our object model. These JavaBeans are called “Plain Old Java Objects” (POJOs). This term is used to distinguish from Enterprise JavaBeans (EJBs). Note that each of these POJOs must have a no-argument constructor.

```
public class Publisher {
    private String code;
    private String name;
    private String address;

    // Getters and Setters
}

public class Book {
    private String isbn;
    private String name;
    private Publisher publisher;
    private Date publishDate;
    private int price;
    private List chapters;

    // Getters and Setters
}

public class Chapter {
    private int index;
    private String title;
    private int numOfPages;

    // Getters and Setters
}
```

We use “Class Diagram” in UML to represent the object model for our online bookshop.



You can see that there are differences between an object model and a relational model. For example, we used a foreign key to reference PUBLISHER from BOOK table, but it is a many-to-one association in Book class. We used a foreign key to reference BOOK from CHAPTER table, but there's nothing referencing Book class from Chapter class. In opposite, a Book object has a list of Chapter objects (one-to-many association).

For the incompatibility of these models, we need to do some conversion when we "retrieve" and "persist" our object model. This is called "Object/Relational Mapping" (O/R Mapping or ORM).

4.2. Retrieving object graphs

Suppose we have a web page in our application for showing the detail of a book (including ISBN, book name, publisher name, publisher address, publish date and all the chapters inside the book). We can use the following JDBC code fragment to retrieve a book object together with a publisher object and a list of chapter objects. Such a group of objects with association is called an "object graph".

```
PreparedStatement stmt = connection.prepareStatement(
    "SELECT * FROM BOOK, PUBLISHER WHERE BOOK.PUBLISHER_CODE = PUBLISHER.CODE
     AND BOOK.ISBN = ?");
stmt.setString(1, isbn);
ResultSet rs = stmt.executeQuery();

Book book = new Book();
if (rs.next()) {
    book.setIsbn(rs.getString("ISBN"));
    book.setName(rs.getString("BOOK_NAME"));
    book.setPublishDate(rs.getDate("PUBLISH_DATE"));
    book.setPrice(rs.getInt("PRICE"));
    Publisher publisher = new Publisher();
    publisher.setCode(rs.getString("PUBLISHER_CODE"));
    publisher.setName(rs.getString("PUBLISHER_NAME"));
    publisher.setAddress(rs.getString("ADDRESS"));
    book.setPublisher(publisher);
}
rs.close();
stmt.close();

List chapters = new ArrayList();
stmt = connection.prepareStatement("SELECT * FROM CHAPTER WHERE BOOK_ISBN = ?");
stmt.setString(1, isbn);
rs = stmt.executeQuery();
while (rs.next()) {
    Chapter chapter = new Chapter();
    chapter.setIndex(rs.getInt("IDX"));
    chapter.setTitle(rs.getString("TITLE"));
    chapter.setNumOfPages(rs.getInt("NUM_OF_PAGES"));
    chapters.add(chapter);
}
book.setChapters(chapters);
rs.close();
stmt.close();
return book;
```

4.3. Persisting object graphs

Suppose we are providing a web page for the users to input the information of a book, including the details of publisher, and what chapters are inside. When he is done, the whole object graph will be saved to the database.

```
PreparedStatement stmt = connection.prepareStatement(
    "INSERT INTO PUBLISHER (CODE, PUBLISHER_NAME, ADDRESS) VALUES (?, ?, ?)");
stmt.setString(1, book.getPublisher().getCode());
stmt.setString(2, book.getPublisher().getName());
stmt.setString(3, book.getPublisher().getAddress());
stmt.executeUpdate();
stmt.close();

stmt = connection.prepareStatement(
    "INSERT INTO BOOK (ISBN, BOOK_NAME, PUBLISHER_CODE, PUBLISH_DATE, PRICE)
     VALUES (?, ?, ?, ?, ?)");
stmt.setString(1, book.getIsbn());
stmt.setString(2, book.getName());
stmt.setString(3, book.getPublisher().getCode());
stmt.setDate(4, new java.sql.Date(book.getPublishDate().getTime()));
stmt.setInt(5, book.getPrice());
stmt.executeUpdate();
stmt.close();

stmt = connection.prepareStatement(
    "INSERT INTO CHAPTER (BOOK_ISBN, IDX, TITLE, NUM_OF_PAGES) VALUES (?, ?, ?, ?)");
for (Iterator iter = book.getChapters().iterator(); iter.hasNext();) {
    Chapter chapter = (Chapter) iter.next();
    stmt.setString(1, book.getIsbn());
    stmt.setInt(2, chapter.getIndex());
    stmt.setString(3, chapter.getTitle());
    stmt.setInt(4, chapter.getNumOfPages());
    stmt.executeUpdate();
}
stmt.close();
```

5. Problems

Now, you have experienced Object/Relational Mapping. The method you are using is through JDBC. Is it good enough? Let's investigate some problems inside this method.

5.1. Too many SQL statements

Using JDBC means you can execute any kind of SQL statements. For a simple task, you have to code many SELECT, INSERT, UPDATE, DELETE statements repeatedly.

5.2. Too many copy codes

When you perform object retrieval, you need to copy the fields in a ResultSet to the properties of an object. When you perform object persistence, you need to copy the properties of an object to the parameters in PreparedStatement.

5.3. Manually handled associations

When you perform object retrieval, you need to perform table join or read from several tables to get a graph of objects. When you perform object persistence, you need to update several tables accordingly.

5.4. Database dependent

The SQL statements you wrote for one database may not work with another brand of database. Although the chance is very small, you may have to migrate to another database for some reasons.