iBATIS IN ACTION

SAMPLE CHAPTER

Clinton Begin
Brandon Goodin
Larry Meadors

MANNING

iBATIS in Action

by Clinton Begin
Brandon Goodin
Larry Meadors

Chapter 2

brief contents

2 What is iBATIS?

This chapter covers

- When to use iBATIS
- When not to use iBATIS
- Getting started
- Future direction

In the previous chapter we discussed in detail the philosophy behind iBATIS and how the framework came to be. We also stated that iBATIS is a hybrid solution that borrows ideas from various other methods of working with a relational database. So what exactly is iBATIS? This chapter will answer that question.

iBATIS is what is known as a *data mapper*. In his book *Patterns of Enterprise Application Architecture* (Addison-Wesley Professional, 2002), Martin Fowler describes the Data Mapper pattern as follows:

> *A layer of Mappers*[1] *that moves data between objects and a database while keeping them independent of each other and the mapper itself.*

Martin does a good job of distinguishing between data mapping and metadata mapping, which is where an object/relational mapping tool fits in. Such a tool maps the tables and columns of the database to the classes and fields of the application. That is, an object relational mapper maps database metadata to class metadata. Figure 2.1 shows an object/relational mapping between a class and a database table. In this case, each field of the class is mapped to a single corresponding column in the database.

iBATIS is different in that it does not directly tie classes to tables or fields to columns, but instead maps the parameters and results (i.e., the inputs and outputs) of a SQL statement to a class. As you'll discover throughout the rest of the book, iBATIS is an additional layer of indirection between the classes and the tables, allowing it more flexibility in how classes and tables can be mapped, without requiring any changes to the data model or the object model. The layer of indirection we're talking about is in fact SQL. This extra layer of indirection allows iBATIS to do a better job of isolating the database design from the object model. This means relatively few dependencies exist between the two. Figure 2.2 shows how iBATIS maps data using SQL.

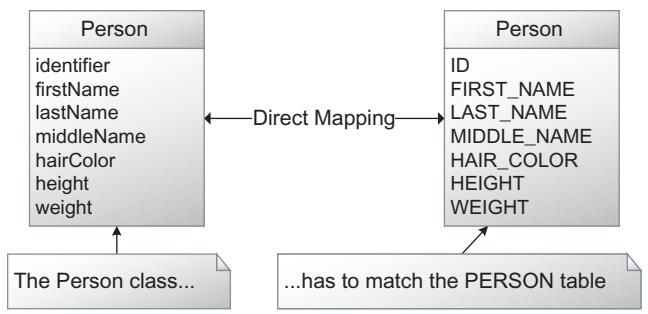

Figure 2.1
Object/relational mapping

[1] Mapper: An object that sets up a communication between two independent objects.
—Martin Fowler in *Patterns of Enterprise Architecture*

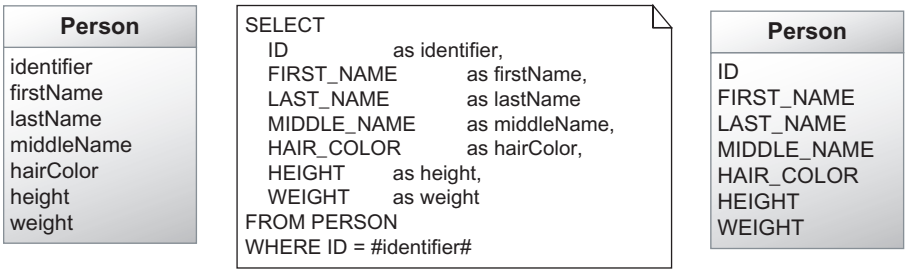

Figure 2.2 iBATIS SQL mapping

As you can see in figure 2.2, the mapping layer of iBATIS is actual SQL. iBATIS lets you write your SQL. iBATIS takes care of mapping the parameters and results between the class properties and the columns of the database table. For this reason, and to eliminate any confusion around the various mapping approaches, the iBATIS team often refers to the data mapper as a SQL mapper.

2.1 Mapping SQL

Any SQL statement can be viewed as a set of inputs and outputs. The inputs are the parameters, typically found in the `WHERE` clause of the SQL statement. The outputs are the columns found in the `SELECT` clause. Figure 2.3 depicts this idea.

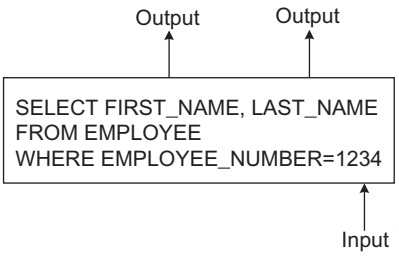

Figure 2.3 SQL can be viewed as inputs and outputs.

The advantage to this approach is that the SQL statement leaves a great deal of flexibility in the hands of the developer. One can easily manipulate the data to match the object model without changing the underlying table design. Furthermore, developers can actually introduce multiple tables or results from built-in database functions or stored procedures. The full power of SQL is at their fingertips.

iBATIS maps the inputs and outputs of the statement using a simple XML descriptor file. Listing 2.1 shows an example of this.

Listing 2.1 A sample SQL mapping descriptor

```
<select id="getAddress"
      parameterClass="int"
      resultClass="Address">
SELECT
    ADR_ID           as id,
    ADR_DESCRIPTION  as description,
    ADR_STREET       as street,
    ADR_CITY         as city,
    ADR_PROVINCE     as province,
    ADR_POSTAL_CODE  as postalCode
  FROM ADDRESS
  WHERE ADR_ID = #id#
</select>
```

Here we can see a SQL `SELECT` statement that returns address data. From the `<select>` element we can see that it takes an Integer object as a parameter, which is marked by the `#id#` token in the `WHERE` clause. We can also see that the result is an object instance of the `Address` class, which is assumed to contain the properties of the same name as the aliases assigned to each column in the `SELECT` clause. For example, the alias *id* would be mapped to a property of the `Address` class also called `id`. Believe it or not, that is all it takes to map a SQL statement that receives an integer as a parameter and returns an `Address` object as output. The Java code used to execute this statement would be

```
Address address = (Address) sqlMap.queryForObject("getAddress",
                                                   new Integer(5));
```

The SQL mapping approach is a very portable concept that can be applied to any full-featured programming language. For example, the C# code from iBATIS.NET is nearly identical:

```
Address address = (Address) sqlMap.QueryForObject("getAddress", 5);
```

Of course there are more advanced options for mapping, especially around results. But we'll discuss those in greater detail in part 2, "iBATIS basics." Right now, it's more important to understand the features and benefits of iBATIS and how it works.

2.2 How it works

More than anything else, iBATIS is an alternative to writing JDBC and ADO.NET code. APIs like JDBC and ADO.NET are powerful, but tend to be verbose and repetitive. Consider the JDBC example in listing 2.2.

Listing 2.2 Example of well-written JDBC code

```
public Employee getEmployee (int id) throws SQLException {
    Employee employee = null;
    String sql = "SELECT * FROM EMPLOYEE " +
                 "WHERE EMPLOYEE_NUMBER = ?";
    Connection conn = null;
    PreparedStatement ps = null;
    ResultSet rs = null;
    try {
      conn = dataSource.getConnection();
      ps = conn.prepareStatement(sql);
      ps.setInt(1, id);
      rs = ps.executeQuery();
      employee = null;
      while (rs.next()) {
        employee = new Employee();
        employee.setId(rs.getInt("ID"));
        employee.setEmployeeNumber(rs.getInt("EMPLOYEE_NUMBER"));
        employee.setFirstName(rs.getString("FIRST_NAME"));
        employee.setLastName(rs.getString("LAST_NAME"));
        employee.setTitle(rs.getString("TITLE"));
      }
    } finally {
      try {
        if (rs != null) rs.close();
      } finally {
        try {
          if (ps != null) ps.close();
        } finally {
          if (conn != null) conn.close();
        }
      }
    }
    return employee;
  }
```

Our SQL is buried here

It's easy to see the overhead created by the JDBC API. Every line is necessary, though, so there's no easy way to reduce it. At best, a few of the lines can be extracted into utility methods, most notably the closing of resources such as the `PreparedStatement` and the `ResultSet`.

Under the hood, iBATIS will run nearly the same JDBC code. iBATIS will get a connection to the database, set the parameters, execute the statement, retrieve the results, and close all of the resources. However, the amount of code that you need to write is significantly reduced. Listing 2.3 shows the code needed for iBATIS to run the exact same statement.

Listing 2.3 iBATIS, which is much less verbose than JDBC

```
<select id="getEmployee"
            parameterClass="java.lang.Integer"
            resultClass="Employee">
       SELECT ID              as id,
            EMPLOYEE_NUMBER   as employeeNumber,
            FIRST_NAME        as firstName,
            LAST_NAME         as lastName,
            TITLE             as title
      FROM EMPLOYEE
      WHERE EMPLOYEE_NUMBER = #empNum#
</select>
```

There is no comparison. The iBATIS code is more concise and easier to read, and thus easier to maintain. We'll discuss more of the benefits of iBATIS later in this chapter. But for now, you're probably wondering how this gets executed from the Java code. As you've seen in earlier examples,, it's a very simple single line of code:

```
Employee emp = (Employee) sqlMap.queryForObject("getEmployee",
                                                new Integer(5));
```

There's nothing to it. This line of code executes the statement, sets the parameters, and retrieves the results as a real Java object. The SQL is encapsulated and externalized neatly in an Extensible Markup Language (XML) file. iBATIS manages all of the resources behind the scenes, and the net effect is the same as the JDBC code we saw earlier in listing 2.2.

This begs the question, does iBATIS work the same way for all systems? Is it best suited for a particular kind of application? The next few sections will answer that, starting with how well iBATIS works with small applications.

2.2.1 iBATIS for small, simple systems

Small applications often work with only a single database, and they often have a fairly simple user interface and domain model. The business logic is very basic, or

perhaps nonexistent for some simple CRUD (Create, Read, Update, Delete) applications. There are three reasons why iBATIS works well with small applications.

First, iBATIS itself is small and simple. It doesn't require a server or any sort of middleware. No additional infrastructure is required at all. iBATIS has no third-party dependencies. A minimal installation of iBATIS consists of only two JAR files that total about 375KB of disk space. There is no setup beyond the SQL Mapping files themselves, so in a matter of minutes you can have a working persistence layer.

Second, iBATIS does not impose on the existing design of the application or the database. Therefore, if you have a small system that is already partially implemented or perhaps even released, it is still easy to refactor the persistence layer to use iBATIS. Because iBATIS is simple, it won't overcomplicate the architecture of the application at all. The same might not be true of object/relational mapping tools or code generators that make assumptions about the design of either the application or the database.

Finally, if you've been working with software for any length of time, you'll likely agree that it's almost inevitable that any *small* piece of software will one day become a large piece of software. All successful software has a tendency to grow. It's a good thing, then, that iBATIS is also very good for large systems, and that it can grow to meet the needs of even enterprise-class applications.

2.2.2 iBATIS for large, enterprise systems

iBATIS was designed for enterprise-class applications. More than anything, iBATIS has a great number of advantages in this area over other solutions. The original creator of iBATIS has only ever had the *luxury* of working applications ranging from large-scale to enterprise-class systems. These systems typically involved not one, but many databases, none of which he had control over. In chapter 1 we discussed various types of databases, including enterprise databases, proprietary databases, and legacy databases. iBATIS was written largely in response to the need to deal with such databases. As a result, iBATIS has a great many features that lend themselves well to the enterprise environment.

The first reason has been stated in other areas, but is so important that it cannot be overstated: *iBATIS does not make any assumptions about the design of your database or your object model.* Regardless of how mismatched these two designs are, iBATIS will work with the application. Furthermore, iBATIS does not make assumptions about the architecture of your enterprise system. If you have partitioned your database vertically by business function, or horizontally by technology, iBATIS will still allow you to effectively work with that data and integrate it into your object-oriented application.

Second, iBATIS has features that allow it to effectively work with very large data sets. iBATIS supports features like row handlers that allow batch processing of very large record sets, one record at a time. It also supports fetching a range of results, allowing you to fetch only the data absolutely necessary for your immediate needs. If you have 10,000 records and only want records 500 to 600, then you can easily fetch just those records. iBATIS supports driver hints that allow it to perform such operations very efficiently.

Finally, iBATIS allows you to map your objects to the database in multiple ways. It's pretty rare that an enterprise system functions only in a single mode. Many enterprise-class systems need to perform transactional functions throughout the day and perform batch functions during the night. iBATIS allows you to have the same class mapped in multiple ways to ensure that each function is supported in the most efficient way possible. iBATIS also supports multiple fetch strategies. That is, you can choose to have some data loaded lazily, while other complex object graphs are loaded with a single SQL join to avoid serious performance problems.

This sounds a lot like a sales pitch. So while we're in the mood, why don't we go into some reasons why you want to use iBATIS? We'll do exactly that in section 2.5. To be fair, a little later in section 2.5 we'll discuss some times when you may *not* want to use iBATIS.

2.3 Why use iBATIS?

There are a great many reasons to use iBATIS in nearly any system. As you learned earlier in this chapter, a framework like iBATIS offers opportunities to inject architectural benefits into your application. Here we'll discuss these benefits and the features that make them possible.

2.3.1 Simplicity

iBATIS is widely regarded as being one of the simplest persistence frameworks available today. Simplicity is at the heart of the design goals of the iBATIS team, and it takes priority over nearly everything else. This simplicity is achieved by maintaining a very solid foundation upon which iBATIS is built: JDBC and SQL. iBATIS is easy for Java developers because it works like JDBC, only with much less code. Almost everything you knew about JDBC applies to iBATIS as well. You can almost think of iBATIS as JDBC code described in XML format. That said, iBATIS includes a number of other architectural benefits that JDBC does not have, which we'll discuss next. iBATIS is also easy to understand for database administrators

and SQL programmers. iBATIS configuration files can be easily understood by nearly anyone with any SQL programming experience.

2.3.2 Productivity

The primary purpose of any good framework is to make the developer more productive. Generally a framework exists to take care of common tasks, reduce boilerplate code, and solve complex architectural challenges. iBATIS succeeds in making developers more productive. In one case study presented at a Java Users Group in Italy (www.jugsardegna.org/vqwiki/jsp/Wiki?IBatisCaseStudy), Fabrizio Giannneschi found that iBATIS reduced the amount of code in the persistence layer by a significant 62 percent. This savings was primarily due to the fact that no JDBC code had to be written. The SQL was still handcoded, but as you saw earlier in this chapter, the SQL is not the problem—it's the JDBC API, and ADO.NET is no different.

2.3.3 Performance

The topic of performance will spark debate among framework authors, users, and even commercial software vendors. The fact is, at a low level all frameworks incur some sort of overhead. Generally, if you compare handcoded JDBC to iBATIS and iterate it 1,000,000 times in a `for` loop, you'll likely see a performance hit in favor of JDBC. Fortunately, this is not the kind of performance that matters in modern application development. What is much more significant is how you fetch data from the database, when you fetch it, and how often. For example, using a paginated list of data that dynamically fetches records from the database can significantly increase the performance of the application because you aren't unnecessarily loading potentially thousands of records from the database at once. Similarly, using features like lazy loading will avoid loading data that isn't necessarily used in a given use case. On the other hand, if you know for certain that you have to load a complex object graph that involves a large amount of data from a number of tables, loading it using a single SQL statement will improve performance greatly. iBATIS supports a number of performance optimizations that we'll discuss in more detail later. For now, it is important to know that iBATIS can usually be configured and used in such a way that is simple, yet it performs as well as JDBC, or possibly better. Another important consideration is that not all JDBC code is well written. JDBC is a complex API that requires a lot of care to code correctly. Unfortunately, much JDBC code is poorly written and therefore will not even perform as well as iBATIS at a low level.

2.3.4 Separation of concerns

With typical JDBC code, it was not uncommon to find database resources such as connections and result sets strewn throughout the application at all layers. We've all seen the nasty applications with database connections and statements in JSP pages, results being iterated over, and HTML in between it all. It's truly nightmarish. In chapter 1 we discussed the importance of application layering. We saw how the application is layered at a high level and also how the persistence layer is layered internally. iBATIS helps to support this layering by managing all of the persistence-related resources, such as database connections, prepared statements, and result sets. It provides database-independent interfaces and APIs that help the rest of the application remain independent of any persistence-related resources. With iBATIS, you're always working only with true objects and never with arbitrary result sets. iBATIS actually makes it hard to break layering best practices.

2.3.5 Division of labor

Some database administrators love their database so much that they won't let anyone else write the SQL for it. Others are just so good at it that everyone else wants them to write the SQL. Whatever the reason, it's always nice to leverage the strengths in your development team. If you have someone who is particularly good at writing SQL but not so hot at writing Java or C#, then let them write the SQL unimpeded. iBATIS allows this to happen. Because the SQL statements are largely separated from the application source code, SQL programmers can write the SQL the way it was meant to be written, without having to worry about string concatenation. Even if the same developers write both the Java code and the SQL, a common request from DBAs while performance tuning a database is "Show me the SQL." This is not an easy thing to do with JDBC, as the SQL is often wound up in a series of concatenated strings or perhaps even dynamically built from a combination of iteration and conditionals. With an object relational mapper the situation is even worse, as you usually have to run the application and log the statements, and once you find them, you may not be able to do anything to change them. iBATIS allows full freedom to enable anyone to develop, view, and change the SQL statements run against the database.

2.3.6 Portability: Java, .NET, and others

iBATIS is a very portable concept. Because of its relatively simple design, it can be implemented for nearly any language or platform. At the time of this writing, iBATIS supports the three most popular development platforms: Java, Ruby, and C# for Microsoft .NET.

The configuration files aren't entirely compatible across the platforms at this time, but there are plans to make that more of a reality. More important, the *concept* and *approach* are very portable. This allows you to be consistent in the design of all of your applications. iBATIS works with more languages and more types of applications than any other framework, regardless of the design of the application. If consistency across your applications is important to you, then iBATIS will work very well for you.

2.3.7 Open source and honesty

Earlier we called this section a "sales pitch." The truth is, iBATIS is free, open source software. We don't make a dime on it whether you use it or not. You've already bought the book, so we've made as much money as we're going to. That said, one of the greatest advantages of open source software is *honesty*. We have no reason to stretch the truth or lie to you. We'll be very up-front and say that iBATIS isn't the perfect solution for all problems. So let's do something that is rarely done in commercial software documentation. Let's discuss some reasons why you may not want to use iBATIS, and suggest alternatives where appropriate.

2.4 When not to use iBATIS

Every framework is built around rules and constraints. Lower-level frameworks like JDBC provide a flexible and complete feature set, but they are harder and more tedious to use. High-level frameworks like object/relational mapping tools are much easier to use and save you a lot of work, but they are built with more assumptions and constraints that make them applicable to fewer applications.

iBATIS is a mid-level framework. It's higher level than JDBC, but lower level than an object relational mapper. That puts iBATIS in a unique position that makes it applicable to a unique set of applications. In the previous sections we discussed why iBATIS is useful in various types of applications, including small, rich client applications and large, enterprise, web applications—and everything in between. So when does iBATIS not fit? The next few sections detail situations where iBATIS is not the best solution and offer recommendations for alternatives.

2.4.1 When you have full control…forever

If you are guaranteed to have full control of your application design and database design, then you are a very lucky person indeed. This is rare in an enterprise environment or any business where the core competency is not software development. However, if you work at a software company that develops

a shrink-wrapped product for which you have full design control, then you might be in this situation.

When you have full control, you have a good reason to use a full object/relational mapping solution such as Hibernate. You can fully leverage the design benefits and productivity gains that an object relational mapper can provide. There will probably not be any interference from an enterprise database group, or a legacy system to integrate with. Furthermore, the database is probably deployed with the application, which places this into the category of an application database (see chapter 1). A good example of a packaged application that uses Hibernate is JIRA from Atlassian. They supply their issue-tracking software as a shrink-wrapped product over which they have full control.

It's important to consider where the application will be in the future, though. If there is any chance that the database could fall out of the control of the application developers, then you might want to carefully consider the impacts that could have on your persistence strategy.

2.4.2 When your application requires fully dynamic SQL

If the core functionality of your application is the dynamic generation of SQL, then iBATIS is the wrong choice. iBATIS supports very powerful dynamic SQL features that in turn support advanced query capability and even some dynamic update functions. However, if every statement in your system is dynamically generated, then you're better off sticking with raw JDBC and possibly building your own framework.

Much of the power of iBATIS is that it allows you complete freedom to manually write and manipulate SQL directly. This advantage is quickly lost when the majority of your SQL is built dynamically from some SQL generator class.

2.4.3 When you're not using a relational database

There are JDBC drivers available for more than relational databases. There are JDBC drivers for flat files, Microsoft Excel spreadsheets, XML, and other types of data stores. Although some people have been successful using such drivers with iBATIS, we don't recommend them for the majority of users.

iBATIS doesn't make many assumptions about your environment. But it does expect that you're using a real relational database that supports transactions and relatively typical SQL and stored procedure semantics. Even some well-known databases don't support very key features of a relational database. Early versions of MySQL did not support transactions, and therefore iBATIS did not work well with it. Luckily today MySQL does support transactions and has a fairly compliant JDBC driver.

If you're not using a real relational database, we recommend that you stick with raw JDBC or even a lower-level file I/O API.

2.4.4 When it simply does not work

iBATIS has a lot of great features that continue to be implemented as the needs of the community grow. However, iBATIS does have a direction as well as design goals that will sometimes conflict with the needs of some applications. People do amazing things with software, and there have been cases where iBATIS simply did not work because of a complex requirement. Although we may have been able to add features to support the requirement, it would have added significant complexity or perhaps would have changed the scope of the iBATIS framework. As a result, we would not change the framework. To support these cases, we try to offer pluggable interfaces so that you can extend iBATIS to meet nearly any need. The simple fact is, sometimes it just doesn't work. In these cases it's best to find a better solution, rather than try to twist iBATIS (or any framework) into something that it is not.

So instead of continuing to discuss the whys and why nots, let's look at a simple example.

2.5 iBATIS in five minutes

The iBATIS framework is a very simple one, and getting started with it is equally simple. How simple? Well, simple enough that we can build an entire application that uses iBATIS in five minutes—not a big Enterprise Resource Planning (ERP) solution or massive e-commerce website, but a simple command-line tool to execute a SQL statement from an iBATIS SQL Map and output the results to the console. The following example will configure a simple static SQL statement to query a simple database table and write it to the console like this:

```
java -classpath <…> Main

Selected 2 records.
{USERNAME=LMEADORS, PASSSWORD=PICKLE, USERID=1, GROUPNAME=EMPLOYEE}
{USERNAME=JDOE, PASSSWORD=TEST, USERID=2, GROUPNAME=EMPLOYEE}
```

Not exactly the prettiest data output, but you get the picture of what it is going to do. In the next few sections, we will walk through the steps to get you from nothing to this level of functionality.

2.5.1 Setting up the database

For the purpose of the sample application, we will use a MySQL database. The iBATIS framework works with any database, as long as it has a JDBC-compliant driver. You simply need to supply a driver class name and a JDBC URL in the configuration.

Setting up a database server is beyond the scope of this book, so we will only provide you with what you need to do on the assumption that the database is already set up and functional. Here is the MySQL script that creates the table we will use and adds some sample data to it:

```
#
# Table structure for table 'user'
#

CREATE TABLE USER_ACCOUNT (
  USERID           INT(3) NOT NULL AUTO_INCREMENT,
  USERNAME         VARCHAR(10) NOT NULL,
  PASSSWORD        VARCHAR(30) NOT NULL,
  GROUPNAME        VARCHAR(10),
  PRIMARY KEY      (USERID)
);

#
# Data for table 'user'
#

INSERT INTO USER_ACCOUNT (USERNAME, PASSSWORD, GROUPNAME)
      VALUES ('LMEADORS', 'PICKLE', 'EMPLOYEE');
INSERT INTO USER_ACCOUNT (USERNAME, PASSSWORD, GROUPNAME)
      VALUES ('JDOE', 'TEST', 'EMPLOYEE');
COMMIT;
```

If you have a different database server already set up with other data that you would like to execute some SQL queries over, feel free to use it for the example. You will need to modify the query in the `SqlMap.xml` file to have your SQL in it and will also need to modify the `SqlMapConfig.xml` file to configure iBATIS to use your database instead. To make it work, you have to know the driver name, the JDBC URL, and a username and password to connect with.

2.5.2 Writing the code

Because this application is our first full example, and an introduction to using iBATIS, the code will be much simpler than a real application would be. We discuss type safety and exception handling later, so we will not be considering those topics here. Listing 2.4 contains the complete code.

Listing 2.4 Main.java

```
import com.ibatis.sqlmap.client.*;
import com.ibatis.common.resources.Resources;

import java.io.Reader;
import java.util.List;

public class Main {
    public static void main(String arg[]) throws Exception {
        String resource = "SqlMapConfig.xml";
        Reader reader = Resources.getResourceAsReader (resource);
       SqlMapClient sqlMap = SqlMapClientBuilder.buildSqlMapClient(reader);
        List list = sqlMap.queryForList("getAllUsers", "EMPLOYEE");
        System.out.println("Selected " + list.size() + " records.");
        for(int i = 0; i < list.size(); i++) {
            System.out.println(list.get(i));
        }
    }
}
```

Configures iBATIS

Executes the statement

Prints the results

That's it! We've configured iBATIS, executed the statement, and printed the results in about 10 lines of Java code. That's all the Java code required for a fully functional iBATIS application. Later, we will refine how things happen, but for now, let's move on to the basics of the configuration.

2.5.3 Configuring iBATIS (a preview)

Since we cover the configuration of iBATIS in depth in the next chapter, we discuss it only briefly here. You won't find much in the way of explanation of the options here, but we cover the essential information.

First, let's look at the `SqlMapConfig.xml` file. This is the starting point for iBATIS, and ties all of the SQL Maps together. Listing 2.5 contains the `SqlMapConfig.xml` file for our simple application.

Listing 2.5 The SQL map configuration for the simplest iBATIS application ever written

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
  <transactionManager type="JDBC" >
```

❶ **Provides DOCTYPE and DTD for validation**

```
    <dataSource type="SIMPLE">
      <property name="JDBC.Driver"
                value="com.mysql.jdbc.Driver"/>
      <property name="JDBC.ConnectionURL"
                value="jdbc:mysql://localhost/test"/>
      <property name="JDBC.Username"
                value="root"/>
      <property name="JDBC.Password"
                value="blah"/>
    </dataSource>
  </transactionManager>
  <sqlMap resource="SqlMap.xml" />
</sqlMapConfig>
```

❷ **Provides name of built-in transaction manager**

❸ **Provides your SQL Maps**

As you may have guessed, this is where we tell iBATIS how to connect to the database and which SQL Map files are available. Since it is an XML document, we need to provide a doctype and DTD for validation ❶. SIMPLE is the name of a built-in transaction manager ❷. Here is where you provide the name of your JDBC driver, the JDBC URL, a username, and a password that lets you connect to the database. Then you provide your SQL Maps ❸. Here, we only have one SQL Map, but you can have as many as you want. There are a few other things you can do here, but we cover them all in the next chapter.

Now that you have seen the main configuration file, let's take a look at the `SqlMap.xml` file (listing 2.6). This is the file that contains the SQL statement that we will be running.

Listing 2.6 The simplest SQL Map ever

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
      "http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap>
    <select id="getAllUsers" parameterClass="string"
                             resultClass="hashmap">
        SELECT * FROM USER_ACCOUNT WHERE GROUPNAME = #groupName#
    </select>
</sqlMap>
```

In the XML code in listing 2.6, we're accepting a String parameter (`parameterClass`) for the `GROUPNAME` parameter, and mapping the results (`resultClass`) to a HashMap.

WARNING Using a Map (e.g., HashMap, TreeMap) as your domain model is not recommended, but this does show the level of mapping flexibility that iBATIS provides. You don't necessarily always need a JavaBean to map to—you can map directly to Maps and primitives as well.

Believe it or not, you have now seen all of the code and configuration required to use iBATIS. We have intentionally spread it out for printing, but even with that, it is only about 50 lines of code, including Java and XML. But the more important point is that 45 of the 50 lines are configuration and are only written once in an application, not for every single statement. As you saw earlier in this chapter, JDBC can end up costing you 50 lines of code or more per statement.

2.5.4 Building the application

Usually when building a large application, you will use something like Ant to make it simpler. Because this is only one class, we are not going to bother building an Ant script for it. To build it, the only JAR files you need on the classpath are `ibatis-common-2.jar` and `ibatis-sqlmap-2.jar`, so we will just key them in on the command line to the Java compiler:

```
javac -classpath <your-path>ibatis-common-2.jar;
<your-path>ibatis-sqlmap-2.jar Main.java
```

Of course, that should all be on one line, and instead of `<your-path>`, you should substitute the actual path to the JAR files. If everything is OK, there should be no output from the compiler to the screen, but a `Main.class` file should be created in the current directory.

2.5.5 Running the application

We will have a few more JARs when we run the application, but not many. To run our application, the only JAR files we need on the classpath are `ibatis-common-2.jar`, `ibatis-sqlmap-2.jar`, `commons-logging.jar`, and our JDBC driver (in this case, `mysql-connector-java.jar`), so next enter this command:

```
java -classpath <your-path>;mysql-connector.jar;commons-logging.jar;
                ibatis-common-2.jar;ibatis-sqlmap-2.jar;.
                Main
```

Again, as with the compilation, this should all go on one line, and `<your-path>` should be replaced with the actual paths on your system.

The program should run and tell you how many records were selected, and then output the data in a rough format, something like this:

```
Selected 2 records.
{USERID=1, USERNAME=LMEADORS, PASSSWORD=PICKLE, GROUPNAME=EMPLOYEE}
{USERID=2, USERNAME=JDOE, PASSSWORD=TEST, GROUPNAME=EMPLOYEE}
```

The iBATIS framework is designed to be very flexible. It can be a very lightweight and simple framework that simply executes SQL and returns data, or it can be used to do much more.

One key to that flexibility is in the proper configuration of the framework. In the next chapter, we are going to look at the two main types of configuration files, and then we'll look at some patterns for solving difficult situations through the use of configuration.

NOTE The configuration files are very standard XML documents. That means that if you have a modern XML editor, the Document Type Definition (DTD) can be used to validate the document, and in some cases even provide code completion while editing.

You've now seen iBATIS in its simplest form. Before we continue, let's talk a bit about where iBATIS is going, so that you can be confident in your decision to use it.

2.6 The future: where is iBATIS going?

iBATIS has gained a great deal of momentum in recent months. As a result, the team has grown, the product has moved, and we've started talking about supporting new platforms. Let's discuss the future in more detail.

2.6.1 Apache Software Foundation

Recently iBATIS became a part of the Apache Software Foundation. We chose to move to Apache because we believe in their mission and respect their attitude. Apache is more than a bunch of servers and infrastructure; it's a system and a true home for open source software. Apache focuses on the community that surrounds software rather than the technology behind it, because without a community the software is a dead project.

What this means to iBATIS users is that iBATIS is not under the direction of a single entity, nor is it dependent on a single entity. Nobody owns iBATIS—it belongs to the community. Apache is there to protect the software and ensure that it stays that way. That said, the Apache license does not restrict the use of open source software as some licenses such as the GPL might. The Apache license is not a *viral* license, which means that you can use the software freely in a commercial environment without worrying about being compliant with unreasonable conditions.

Although Apache doesn't focus on infrastructure, they do have some very good infrastructure. Currently iBATIS makes use of Subversion source control (SVN), Atlassian's JIRA for issue tracking, Atlassian's Confluence for collaborative wiki documentation, and Apache's mailing list servers for communication among the development team, users, and the community in general.

Apache has what it takes to protect iBATIS and ensure that it will be around for as long as there are people who want to use it.

2.6.2 Simpler, smaller, with fewer dependencies

Unlike some frameworks, the iBATIS project has no goals to branch out into new areas and take over the world. iBATIS is a very focused project and with each release we only hope to make it smaller and simpler and maintain independence from third-party libraries.

We believe that iBATIS has much room for innovation. There are a lot of new technologies and design approaches that iBATIS can benefit from to make configuration more concise and easier to work with. For example, both C# and Java have attribute (a.k.a. annotation) functionality built in. In future versions, iBATIS will likely leverage this to reduce the amount of XML needed to configure the framework.

There is also a lot of room for tools development. The iBATIS design lends itself well to graphical tools such as integrated development environments. It is also possible for iBATIS configurations to be generated from database schemas, for which there are already tools available. You can see examples of some of the tools on our website at http://ibatis.apache.org.

2.6.3 More extensions and plug-ins

iBATIS already has a number of extension points. We'll talk about them in detail in chapter 12. You can already implement your own transaction manager, data source, cache controllers, and more. But we have a goal to make iBATIS even more extendible. We'd like to see a pluggable design at nearly every layer of JDBC architecture, meaning you'd be able to implement your own `ResultSet` handlers and SQL execution engines. This would help us support more complex or legacy systems that operate in a proprietary way. It would also enable developers to take greater advantage of customized features of particular databases or application servers.

2.6.4 Additional platforms and languages

As you've noticed in both chapters 1 and 2, we've discussed iBATIS for .NET and Java. The remainder of this book will focus mostly on the Java APIs, but most of the information is transferable to the .NET platform as well. We'll also discuss

.NET in more detail in appendix. iBATIS has been implemented for Ruby as well, but Ruby is a significantly different language, and therefore iBATIS for Ruby is quite different as well. We'll not discuss the Ruby implementation here.

In addition to Java and C#, the iBATIS team has discussed implementing iBATIS in other languages, including PHP 5 and Python. We believe that iBATIS can contribute significant value to almost any platform where the choices are limited to a low-level database API and a high-level object/relational mapping tool. iBATIS can fill the middle ground and again allow you to implement all of your applications in a consistent way across the board.

We've also discussed drafting a specification that would make it easier to migrate iBATIS to different platforms and ensure reasonable consistency. Of course, we'd like iBATIS to take full advantage of unique language and platform features, but we'd also like to see some level of similarity to ensure that they can all be called iBATIS and be recognizable to developers experienced with iBATIS in another language.

2.7 Summary

In this chapter you learned that iBATIS is a unique data mapper that uses an approach called SQL mapping to persist objects to a relational database. iBATIS is consistently implemented in both Java and .NET, and there is significant value in a consistent approach to persistence in your applications.

You also learned how iBATIS works. Generally, under the hood iBATIS will run well-written JDBC or ADO.NET code that would otherwise be hard to maintain when coded manually. You found that when compared to JDBC, iBATIS code is less verbose and easier to code.

We discussed how iBATIS, despite its simple design, is a very appropriate framework for both small and large enterprise applications alike. iBATIS has many features that support enterprise-level persistence requirements. Features such as row handlers allow large data sets to be processed efficiently, one record at a time, to ensure that you don't completely drain the system memory.

We also discussed a number of important features that distinguish iBATIS from the competition, and we made a strong case for using iBATIS. These features include the following:

- *Simplicity*—iBATIS is widely regarded as the simplest persistence framework available.
- *Productivity*—Concise code and simple configuration reduces the code to 62 percent of the corresponding JDBC code.
- *Performance*—Architectural enhancements like join mapping speed up data access.
- *Separation of concerns*—iBATIS improves design to ensure future maintainability.
- *Division of labor*—iBATIS helps break up the work to allow teams to leverage expertise.
- *Portability*—iBATIS can be implemented for any full-featured programming language.

After our sales pitch, we admitted that iBATIS is not a silver bullet, because no framework is. We discussed situations where iBATIS would likely not be the ideal approach. For example, if you have full control over the application and the database now and forever, then a full-blown object relational mapper is probably a better choice. On the other hand, if your application works primarily with dynamically generated SQL code, then raw JDBC is the way to go. We also mentioned that iBATIS is primarily designed for relational databases, and if you are using flat files, XML, Excel spreadsheets, or any other nonrelational technology, then you're better off with a different API altogether.

Finally, we ended the chapter by discussing the future of iBATIS. The team has a lot of great design goals for the future, and the Apache Software Foundation will ensure that there is an energetic community capable of supporting it for years to come.

JAVA

iBATIS IN ACTION

Clinton Begin • Brandon Goodin • Larry Meadors

Unlike some complex and invasive persistence solutions, iBATIS keeps O/RM clean and simple. It is an elegant persistence framework that maps classes to SQL statements and keeps the learning curve flat. The iBATIS approach makes apps easy to code, test, and deploy. You write regular SQL and iBATIS gives you standard objects for persistence and retrieval. There's no need to change existing database schemas—iBATIS is tolerant of legacy databases (even badly designed ones).

iBATIS in Action is a comprehensive tutorial on the framework and an introduction to the iBATIS philosophy. Clinton Begin and coauthors lead you through the core features, including configuration, statements, and transactions. Because you'll need more than the basics, it explores sophisticated topics like Dynamic SQL and data layer abstraction. You'll also learn a useful skill: how to extend iBATIS itself. A complete, detailed example shows you how to put iBATIS to work. Topics are clearly organized and easily accessible for reference.

What's Inside

- A comprehensive iBATIS tutorial
- Learn iBATIS techniques, patterns, and best practices
- How to build a complete web application
- Inside story from the authoritative source

Clinton Begin is a senior developer at ThoughtWorks Canada and the creator of iBATIS. **Brandon Goodin** is a consultant who has contributed to the iBATIS project since 2003. **Larry Meadors** is a consultant and trainer who has been involved with iBATIS since version 1.x.

"Unique and invaluable, this book will be at my side for years to come."
—Nathan Maves, Senior Java Architect
Sun Microsystems

"This book really shines."
—Benjamin Gorlick
Global Engineered Products, LLC.

"The writing is good, relaxed, and sometimes fun."
—Dick Zetterberg, Transitor AB

"Gets new users going and gives experienced users in-depth coverage of advanced features."
—Jeff Cunningham
The Weather Channel Interactive

"Easy flow, good breakdown of topics, relevant and thorough—valuable for all."
—Rick Reumann
Nielsen Media Research

Ask the Authors


Ebook edition

www.manning.com/begin

ISBN 1-932394-82-6

$44.99 US/$58.99 Canada