In this chapter, you will learn how Spring can simplify your database access tasks. Data access is a common requirement for most enterprise applications, which usually require accessing data stored in relational databases. As an essential part of Java SE, JDBC (Java Database Connectivity) defines a set of standard APIs for you to access relational databases in a vendor-independent fashion.

The purpose of JDBC is to provide APIs through which you can execute SQL statements against a database. However, when using JDBC, you have to manage database-related resources by yourself and handle database exceptions explicitly. To make JDBC easier to use, Spring establishes a JDBC accessing framework by defining an abstract layer on top of the JDBC APIs.

As the heart of the Spring JDBC framework, JDBC templates are designed to provide template methods for different types of JDBC operations. Each template method is responsible for controlling the overall process and allows you to override particular tasks of the process. In this way, you can minimize your database access effort while retaining as much flexibility as possible.

Besides the JDBC template approach, the Spring JDBC framework supplies another more object-oriented approach for you to organize your data access logic. It enables you to model each database operation as a fine-grained operation object. Compared to the JDBC template approach, the JDBC operation object approach is just an alternative approach to organizing your data access logic. Some developers may prefer the former, while others like the latter better.

Upon finishing this chapter, you will be able to use the Spring JDBC framework to access relational databases. As part of the Spring data access module, the Spring JDBC framework is consistent with other parts of the module. So, learning about the JDBC framework is an ideal introduction to the entire data access module.

7-1. Problems with Direct JDBC

Suppose you are going to develop an application for vehicle registration, whose major functions are the basic CRUD (create, read, update, and delete) operations on vehicle records. These records will be stored in a relational database and accessed with JDBC. First, you design the following Vehicle class, which represents a vehicle in Java:
Setting Up the Application Database

Before developing your vehicle registration application, you have to set up the database for it. For the sake of low memory consumption and easy configuration, I have chosen Apache Derby (http://db.apache.org/derby/) as my database engine. Derby is an open source relational database engine provided under the Apache License and implemented in pure Java.

**Note** You can download the Apache Derby binary distribution (e.g., v10.3) from the Apache Derby website and extract it to a directory of your choice to complete the installation.

---

Derby can run in either the embedded mode or the client/server mode. For testing purposes, the client/server mode is more appropriate because it allows you to inspect and edit data with any visual database tools that support JDBC—for example, the Eclipse Data Tools Platform (DTP).

**Note** To start the Derby server in the client/server mode, just execute the `startNetworkServer` script for your platform (located in the `bin` directory of the Derby installation).

---

After starting up the Derby network server on localhost, you can connect to it with the JDBC properties shown in Table 7-1.

**Note** You require Derby’s client JDBC driver `derbyclient.jar` (located in the `lib` directory of the Derby installation) to connect to the Derby server.