7 The Web and JDBC

7.1 Introduction

By default, applets are not allowed to load libraries or read and write files. In addition, applets are not allowed to open sockets to machines other than those they originated from. These restrictions cause a number of problems for those wishing to develop applets that work with databases. For example, many drivers rely on the ability to load native code libraries that actually generate the connection to the specified database. One way around these restrictions is to turn them off in the browser being used. This is acceptable for an Intranet being used within a single organization; however, it is not acceptable as a general solution.

Another possibility is to use drivers that are 100% pure Java, such as the mSQL driver (mSQL is described briefly in Section 7.2). However, even using an mSQL driver, the applet is still restricted to connecting to a database on the originating host. Thus the developers must ensure that the Web server that serves the applets is running on the same host as the mSQL daemon. This may or may not be a problem.

Another option is to use a separate database server application (note application and not applet) which runs on the same host as the Web server. Applets can then connect to the database server application, requesting that it connect to databases, execute updates, perform queries etc. The database server application is then the program that connects to and interacts with the database. In such a setup the applet does not directly communicate with the database system. In turn, the server application is not hindered by the restrictions imposed on applets and can therefore connect to any database it has access to. This approach is illustrated in the remainder of this chapter. We introduce the concept of sockets and discuss how they are implemented in Java before using sockets to enable a client applet to communicate with a server application. We conclude this chapter with a brief discussion of signed applets.

7.2 MiniSQL

MiniSQL or mSQL is a lightweight database server originally developed as part of the Minerva Network Management Environment. Its creator, David Hughes, has continued its development and makes mSQL available as a shareware product for a very small fee. MSQL provides fast access to stored data with low memory requirements through a subset of ANSI SQL (i.e. it does not support views or subqueries etc.). It is available for Unix-compatible operating systems as well as for Windows
However, it is worth noting that the Unix version tends to be ahead of the PC-oriented versions. The mSQL package includes the database engine, a terminal "monitor" program, a database administration program, a schema viewer and C and Java language APIs. A Java JDBC driver for mSQL is also available.

mSQL is a very popular choice among Java developers because it is available on a wide variety of platforms, the mSQL driver is 100% Pure Java (and thus there is no problem about loading a native library when writing applets) and of course it is shareware. The downside is that the performance of mSQL is not as good as that of commercial database systems. With regard to applets it is worth noting that an applet (by default) is not allowed to make a network connection to any other computer other than the machine from which it was loaded. Thus the Web server and the mSQL demon must be running on the same machine. For more information on mSQL see Jepson (1998).

### 7.3 Sockets in Java

A "socket" is an end point in a communication link between separate processes. In Java, sockets are objects which provide a way of exchanging information between two processes in a straightforward and platform-independent manner (see the classes in the java.net package). To achieve this, the streams model, already used for file access, is exploited. Associated with each socket are two streams, one for input and one for output. Thus to pass information from one process to another you write that information to the output stream of one socket and read it from the input stream of another socket (assuming the two sockets are connected). This is illustrated in Figure 7.1. This has the great advantage that once the network connection has been

---

**Server**

1. Create a server socket
   
   ```java
   s = new ServerSocket(5432);
   ```

2. Wait for a connection
   
   ```java
   Socket conn = s.accept();
   ```

3. Get an output stream

4. Get an output stream

5. Write data onto stream

6. Close output stream, connection and server socket

---

**Client**

3. Open a connection
   
   ```java
   Socket soc =
       new Socket("Manuel",
                   5432);
   ```

5. Get an input stream

6. Read data from input stream

8. Close stream and socket
   
   ```java
   stream.close();
   soc.close();
   ```

**Figure 7.1** Socket-to-socket communication.