24.6 Connecting to and Querying a Database

The example of Fig. 24.23 performs a simple query on the `books` database that retrieves the entire `Authors` table and displays the data. The program illustrates connecting to the database, querying the database and processing the result. The discussion that follows presents the key JDBC aspects of the program.

```
      1   // Fig. 24.23: DisplayAuthors.java
 2   // Displaying the contents of the Authors table.
          3   import java.sql.Connection;
          4   import java.sql.Statement;
        5   import java.sql.DriverManager;
          6   import java.sql.ResultSet;
       7   import java.sql.ResultSetMetaData;
         8   import java.sql.SQLException;
                        9
        10   public class DisplayAuthors {
 11      public static void main(String args[]) {
 12         final String DATABASE_URL = "jdbc:derby:bo
      13         final String SELECT_QUERY =
 14            "SELECT authorID, firstName, lastName F
                        15
 16         // use try-with-resources to connect to an
               17         try (
 18            Connection connection = DriverManager.g
 19               DATABASE_URL, "deitel", "deitel");
 20            Statement statement = connection.create
 21            ResultSet resultSet = statement.execute
```

```
                    22
    23            // get ResultSet's meta data
24              ResultSetMetaData metaData = resultSet.
25              int numberOfColumns = metaData.getColum
                    26
27              System.out.printf("Authors Table of Boo
                     28
29              // display the names of the columns in
30              for (int i = 1; i <= numberOfColumns; i
31                 System.out.printf("%-8s\t", metaData
                   32             }
       33              System.out.println();
                     34
     35              // display query results
    36              while (resultSet.next()) {
37                 for (int i = 1; i <= numberOfColumns
38                    System.out.printf("%-8s\t", resul
                 39                }
     40                 System.out.println();
                   41             }
                     42          }
  43          catch (SQLException sqlException) {
  44             sqlException.printStackTrace();
                   45          }
                     46       }
                      47    }
```

Authors Table of Books Database:

AUTHORID	FIRSTNAME	LASTNAME
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel

4	Dan	Quirk
5	Michael	Morgano

Fig. 24.23

Displaying the contents of the `Authors` table.

Lines 3–8 import the JDBC interfaces and classes from package `java.sql` used in this program. Method `main` connects to the `books` database, queries the database, displays the query result and closes the database connection. Line 12 declares a `String` constant for the database URL. This identifies the name of the database to connect to, as well as information about the protocol used by the JDBC driver (discussed shortly). Lines 13–14 declare a `String` constant representing the SQL query that will select the `authorID`, `firstName` and `lastName` columns from the database's `authors` table.

24.6.1 Automatic Driver Discovery

JDBC supports **automatic driver discovery**—it loads the database driver into memory for you. To ensure that the program can locate the driver class, you must include the class's location in the program's classpath when you execute the program. You did this for Java DB in Section 24.5 when

you executed the `setEmbeddedCP.bat` or `setEmbeddedCP` file on your system—that step configured a `CLASSPATH` environment variable in the command window for your platform. After doing so, you can run this application simply using the command

```
java DisplayAuthors
```

24.6.2 Connecting to the Database

The JDBC interfaces we use in this example extend the `AutoCloseable` interface, so you can use objects that implement these interfaces with the `try`-with-resources statement (lines 17–45). Lines 18–21 create this example's `AutoCloseable` objects, which are closed when the `try` block terminates (line 42) or if an exception occurs during the `try` block's execution. Each object created in the parentheses following keyword `try` must be separated from the next by a semicolon (`;`).

Lines 18–19 create a `Connection` object that manages the connection between the Java program and the database. `Connection` objects enable programs to create SQL statements that manipulate databases. The program initializes `connection` with the result of a call to the `DriverManager` class's `static` method

`getConnection`. The method's arguments are:

- a `String` that specifies the database URL,
- a `String` that specifies the username, and
- a `String` that specifies the password.

This method attempts to connect to the database specified by its URL argument—if it cannot, it throws a `SQLException` (package `java.sql`). The username and password for the `books` database were set in Section 24.5 when you created the database. If you used a different username and password there, you'll need to replace the username (second argument) and password (third argument) passed to method `getConnection` in lines 18–19.

The URL locates the database. In this chapter's examples, the database is on the local computer, but it could reside on a network. The URL `jdbc:derby:books` specifies:

- the protocol for communication (`jdbc`),
- the **subprotocol** for communication (`derby`), and
- the database name (`books`).

The subprotocol `derby` indicates that the program uses a Java DB/Apache Derby-specific subprotocol to connect to the database—recall that Java DB is simply the Oracle branded version of Apache Derby. Figure 24.24 lists the JDBC driver names and database URL formats of several popular RDBMSs.

Software Engineering Observation 24.3

Most database management systems require the user to log in before accessing the database contents. `DriverManager` *method* `getConnection` *is overloaded with versions that enable the program to supply the username and password to gain access.*

RDBMS	Database URL format
MySQL	`jdbc:mysql://`*hostname*`:`*portNumber*`/`*databaseName*
ORACLE	`jdbc:oracle:thin:@`*hostname*`:`*portNumber*`:`*databaseName*
DB2	`jdbc:db2:`*hostname*`:`*portNumber*`/`*databaseName*
PostgreSQL	`jdbc:postgresql://`*hostname*`:`*portNumber*`/`*databaseName*
Java DB/Apache Derby	`jdbc:derby:`*dataBaseName* (embedded; used in this chapter) `jdbc:derby://`*hostname*`:`*portNumber*`/`*databaseName* (network)
Microsoft SQL Server	`jdbc:sqlserver://`*hostname*`:`*portNumber*`;databaseName=`*dataBaseName*
Sybase	`jdbc:sybase:Tds:`*hostname*`:`*portNumber*`/`*databaseName*

Fig. 24.24

Popular JDBC database URL formats.

24.6.3 Creating a `Statement` for Executing Queries

Line 20 of Fig. 24.23 invokes `Connection` method `createStatement` to obtain an object >that implements interface `Statement` (package `java.sql`). You use a `Statement` object to submit SQL statements to the database.

24.6.4 Executing a Query

Line 21 uses the `Statement` object's `executeQuery` method to submit a query that selects all the author information from the `Authors` table. This method returns an object that implements interface `ResultSet` and contains the query results.

24.6.5 Processing a Query's `ResultSet`

Lines 24–41 process the `ResultSet`. Line 24 obtains the `ResultSet`'s `ResultSetMetaData` object (package `java.sql`). The **metadata** describes the `ResultSet`'s contents. Programs can use metadata programmatically to obtain information about the `ResultSet`'s column names

and types. Line 25 uses `ResultSetMetaData` method `getColumnCount` to retrieve the number of columns in the `ResultSet`. Lines 30–32 display the column names.

Software Engineering Observation 24.4

Metadata enables programs to process `ResultSet` *contents dynamically when detailed information about the* `ResultSet` *is not known in advance.*

Lines 36–41 display the data in each `ResultSet` row. First, the program positions the `ResultSet` cursor (which points to the row being processed) to the first row. Method `next` (line 36) returns `boolean` value `true` if it's able to position to the next row; otherwise, the method returns `false` to indicate that the end of the `ResultSet` has been reached.

Common Programming Error 24.6

Specifying column index 0 when obtaining values from a `ResultSet` *causes a* `SQL-Exception`*—the first column index in a* `ResultSet` *is always 1.*

If the `ResultSet` has rows, lines 37–39 extract and display the contents of each column in the current row. Each column can be extracted as a specific Java type—`ResultSetMetaData` method `getColumnType` returns a constant integer from class `Types` (package `java.sql`) indicating a given column's type. Programs can use these values in a `switch` statement to invoke `ResultSet` methods that return the column values as appropriate Java types. For example, if the a column's type is `Types.INTEGER`, `ResultSet` method `getInt` gets the column value as an `int`. For simplicity, this example treats each value as an `Object`. We retrieve each column value with `ResultSet` method `getObject` (line 38), then display the `Object`'s `String` representation. `ResultSet` *get* methods typically receive as an argument either a column number (as an `int`) or a column name (as a `String`) indicating which column's value to obtain. Unlike array indices, `ResultSet` *column numbers start at 1*.

Common Programming Error 24.5

Initially, a `ResultSet` *cursor is positioned before the first row. A* `SQLException` *occurs if you attempt to access a* `ResultSet`*'s contents before positioning the* `ResultSet` *cursor to the first row with method* `next`.

Performance Tip 24.1

If a query specifies the exact columns to select from the database, the `ResultSet` *contains the columns in the specified order. For this scenario, using the column number to obtain the column's value is more efficient than using the column name. The column number provides direct access to the specified column. Using the column name requires a search of the column names to locate the appropriate column.*

Error-Prevention Tip 24.1

Using column names to obtain values from a `ResultSet` *produces code that is less error prone than obtaining values by column number—you don't need to remember the column order. Also, if the column order changes, your code does not have to change.*

When the end of the `try` block is reached (line 42), the `close` method is called on the `ResultSet`, `Statement` and `Connection` objects that were obtained at the beginning of the `try`-with-resources statement.

Common Programming

Error 24.7

A `SQLException` *occurs if you attempt to manipulate a* `ResultSet` *after closing the* `Statement` *that created it. The* `ResultSet` *is discarded when the* `Statement` *is closed.*

Software Engineering Observation 24.5

Each `Statement` *object can open only one* `ResultSet` *object at a time. When a* `Statement` *returns a new* `ResultSet`*, the* `Statement` *closes the prior* `ResultSet`*. To use multiple* `ResultSet`*s in parallel, separate* `Statement` *objects must return the* `ResultSet`*s.*