24.8 RowSet Interface

In the preceding examples, you learned how to query a database by explicitly establishing a `Connection` to the database, preparing a `Statement` for querying the database and executing the query. In this section, we demonstrate the `RowSet` **interface**, which configures the database connection and prepares query statements automatically. The interface `RowSet` provides several *set* methods that allow you to specify the properties needed to establish a connection (such as the database URL, username and password of the database) and create a `Statement` (such as a query). `RowSet` also provides several *get* methods that return these properties.

Connected and Disconnected `RowSets`

There are two types of `RowSet` objects—connected and disconnected. A **connected** `RowSet` object connects to the database once and remains connected while the object is in use. A **disconnected** `RowSet` object connects to the database, executes a query to retrieve the data from the database and then closes the connection. A program may change the data in a disconnected `RowSet` while it’s disconnected. Modified data then can be updated in the database after a disconnected

`RowSet` reestablishes the connection with the database.

Package `javax.sql.rowset` contains two subinterfaces of `RowSet`—`JdbcRowSet` and `CachedRowSet`. `JdbcRowSet`, a connected `RowSet`, acts as a wrapper around a `ResultSet` object and allows you to scroll through and update the rows in the `ResultSet`. Recall that by default, a `ResultSet` object is nonscrollable and read only —you must explicitly set the result-set type constant to `TYPE_SCROLL_INSENSITIVE` and set the result-set concurrency constant to `CONCUR_UPDATABLE` to make a `ResultSet` object scrollable and updatable. A `JdbcRowSet` object is scrollable and updatable by default. `CachedRowSet`, a disconnected `RowSet`, caches the data of a `ResultSet` in memory and disconnects from the database. Like `JdbcRowSet`, a `CachedRowSet` object is scrollable and updatable by default. A `CachedRowSet` object is also *serializable*, so it can be passed between Java applications through a network, such as the Internet. However, `CachedRowSet` has a limitation—the amount of data that can be stored in memory is limited. Package `javax.sql.rowset` contains three other subinterfaces of `RowSet`.

Portability Tip 24.4

A `RowSet` *can provide scrolling capability for drivers that do not support scrollable* `ResultSet`*s.*

Using a `RowSet`

Figure 24.30 reimplements the example of Fig. 24.23 using a `RowSet`. Rather than establish the connection and create a `Statement` explicitly, Fig. 24.30 uses a `JdbcRowSet` object to create a `Connection` and a `Statement` automatically.

```java
// Fig. 24.30: JdbcRowSetTest.java
// Displaying the contents of the Authors table
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class JdbcRowSetTest {
   // JDBC driver name and database URL
   private static final String DATABASE_URL = "j
   private static final String USERNAME = "deite
   private static final String PASSWORD = "deite

   public static void main(String args[]) {
      // connect to database books and query dat
      try (JdbcRowSet rowSet =
         RowSetProvider.newFactory().createJdbcR

         // specify JdbcRowSet properties
         rowSet.setUrl(DATABASE_URL);
         rowSet.setUsername(USERNAME);
         rowSet.setPassword(PASSWORD);
         rowSet.setCommand("SELECT * FROM Author
         rowSet.execute(); // execute query

         // process query results
         ResultSetMetaData metaData = rowSet.get
         int numberOfColumns = metaData.getColum
         System.out.printf("Authors Table of Boo

         // display rowset header
         for (int i = 1; i <= numberOfColumns; i
            System.out.printf("%-8s\t", metaData
         }
         System.out.println();

         // display each row
         while (rowSet.next()) {
            for (int i = 1; i <= numberOfColumns
               System.out.printf("%-8s\t", rowSe
            }
            System.out.println();
         }
      }
      catch (SQLException sqlException) {
         sqlException.printStackTrace();
         System.exit(1);
      }
   }
}
```

```
Authors Table of Books Database:
```

AUTHORID	FIRSTNAME	LASTNAME
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel
4	Dan	Quirk
5	Michael	Morgano

Fig. 24.30

Displaying the contents of the `Authors` table using `JdbcRowSet`.

Class `RowSetProvider` (package `javax.sql.rowset`) provides `static` method `new-Factory` which returns an object that implements the `RowSetFactory` interface (package `javax.sql.rowset`). This object can be used to create various types of `RowSet`s. Lines 16– 17 in the `try`-with-resources statement use `RowSetFactory` method `createJdbcRowSet` to obtain a `JdbcRowSet` object.

Lines 20–22 set the `RowSet` properties that the `DriverManager` uses to establish a database connection. Line 20 invokes `JdbcRowSet` method `setUrl` to specify the database URL. Line 21 invokes `JdbcRowSet` method `setUsername` to specify the username. Line 22 invokes `JdbcRowSet` method `setPassword` to specify the password. Line 23 invokes `JdbcRowSet` method `setCommand` to specify the SQL query that will populate the `RowSet`. Line 24 invokes `JdbcRowSet` method `execute` to execute the SQL query. Method `execute` performs four actions—it establishes a `Connection` to the database, prepares the query `Statement`, executes the query and stores the `ResultSet` returned by the query. The `Connection`, `Statement` and `ResultSet` are encapsulated in the `JdbcRowSet` object.

The remaining code is almost identical to Fig. 24.23, except that line 27 (Fig. 24.30) obtains a `ResultSetMetaData` object from the `JdbcRowSet`, line 38 uses the `JdbcRowSet`'s `next` method to get the next row of the result and line 40 uses the `JdbcRowSet`'s `getObject` method to obtain a column's value. When the end of the `try` block is reached, the `try`-with-resources statement invokes `JdbcRowSet` method `close` to close the `RowSet`'s encapsulated `ResultSet`, `Statement` and `Connection`. In a `CachedRowSet`, invoking `close` also releases the resources held by that `RowSet`. The application's output is identical to Fig. 24.23.