

## 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 ResultSets.*

# 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.

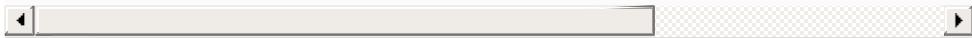
---

```
1  // Fig. 24.30: JdbcRowSetTest.java
2  // Displaying the contents of the Authors table
3  import java.sql.ResultSetMetaData;
4  import java.sql.SQLException;
5  import javax.sql.rowset.JdbcRowSet;
6  import javax.sql.rowset.RowSetProvider;
7
8  public class JdbcRowSetTest {
9      // JDBC driver name and database URL
10     private static final String DATABASE_URL = "j
11     private static final String USERNAME = "deite
12     private static final String PASSWORD = "deite
13
14     public static void main(String args[]) {
15         // connect to database books and query dat
16         try (JdbcRowSet rowSet =
17             RowSetProvider.newFactory().createJdbcR
18
19             // specify JdbcRowSet properties
20             rowSet.setUrl(DATABASE_URL);
21             rowSet.setUsername(USERNAME);
22             rowSet.setPassword(PASSWORD);
23             rowSet.setCommand("SELECT * FROM Author
24             rowSet.execute(); // execute query
25
26             // process query results
27             ResultSetMetaData metaData = rowSet.get
28             int numberOfColumns = metaData.getColu
```

```

29         System.out.printf("Authors Table of Boo
30
31             // display rowset header
32             for (int i = 1; i <= numberOfRowsColumns; i
33                 System.out.printf("%-8s\t", metaData
34                     }
35             System.out.println();
36
37             // display each row
38             while (rowSet.next()) {
39                 for (int i = 1; i <= numberOfRowsColumns
40                     System.out.printf("%-8s\t", rowSe
41                     }
42             System.out.println();
43             }
44         }
45     catch (SQLException sqlException) {
46         sqlException.printStackTrace();
47             System.exit(1);
48         }
49     }
50 }

```



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 `newFactory` which returns an object that implements the `RowSetFactory` interface (package `javax.sql.rowset`). This object can be used to create various types of `RowSets`. 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.