How to Use CachedRowSet in JDBC

Last Updated on 27 March 2018 | Print ✉ Email

Complete Java Masterclass - updated for Java 10

Scrollable result sets and updatable result sets which are powerful and flexible, but they have a drawback: it needs to keep the database connection open during the entire user interaction. This behavior may affect your application's performance in case there are many concurrent connections to the database.

In addition, being tied to database connection make `ResultSet` objects unable to be transferred between components or tiers in an application.

In this situation, you can use a ***row set*** which doesn't need to keep the database connection always open, and is leaner than a result set.

In this JDBC tutorial, we will help you understand row set and how to use one of its implementations - cached row set.

1. Understanding RowSet

A row set contains all data from a result set, but it can be disconnected from the database. A row set may make a connection with a database and keep the connection open during its life cycle, in which case it is called *connected row set*.

A row set may also make connection with a database, get data from it, and then close the connection. Such a row set is called *disconnected row set*.

You can make changes to data in a disconnected row set, and commit changes to the database later (the row set must re-establish the connection with the database).

In JDBC, a row set is represented by the **`RowSet`** interface which is defined in the **`javax.sql`** package. The `javax.sql package` is an extension of JDBC, besides the primary package `java.sql`.

The `RowSet` interface extends the `java.sql.ResultSet` interface, which means you can use a row set just like a result set.

The **`javax.sql.rowset`** package provides the following interfaces that extend the `RowSet` interface:

- A **`CachedRowSet`**stores data in memory so you can work on the data without keeping the connection open all the time. `CachedRowSet` is the super interface of the ones below.
- A **`FilteredRowSet`**allows filtering data without having to write SQL SELECT queries.
- A **`JoinRowSet`**combines data from different `RowSet` objects, which is equivalent to SQL JOIN queries.
- A **`JdbcRowSet`**is a thin wrapper around a `ResultSet` that makes it possible to use the result set as a JavaBeans component.
- A **`WebRowSet`**can read and write data in XML format, making it possible to transfer the data through tiers in a web application.

2. Understanding CachedRowSet

A `CachedRowSet` object is a container for rows of data that caches its rows in memory, which makes it possible to operate (scroll and update) without keeping the database connection open all the time.

A `CachedRowSet` object makes use of a connection to the database only briefly: while it is reading data to populate itself with rows, and again while it is committing changes to the underlying database. So the rest of the time, a `CachedRowSet` object is disconnected, even while its data is being modified. Hence it is called disconnected row set.

Being disconnected, a `CachedRowSet` object is much leaner than a `ResultSet` object, making it easier to pass a `CachedRowSet` object to another component in an application.

You can modify data in a `CachedRowSet` object, but the modifications are not immediately reflected in the database. You need to make an explicit request to accept accumulated changes (insert, update and delete rows). The `CachedRowSet` then reconnects to the database and issues SQL statements to commit the changes.

3. Creating a CachedRowSet Object

You can create a `CachedRowSet` object either from a reference implementation provided by JDK (default), or from an implementation of database vendor (if available).

The following code snippet creates a `CachedRowSet` object by using a `RowSetFactory` which is created by the `RowSetProvider`:

```java
RowSetFactory factory = RowSetProvider.newFactory();
CachedRowSet rowset = factory.createCachedRowSet();
```

This creates a `CachedRowSet` object from the implementation class `com.sun.rowset.CachedRowSetImpl`. It's equivalent to the following statement:

```java
CachedRowSet rowset = new com.sun.rowset.CachedRowSetImpl();
```

However, it's recommended to create a `CachedRowSet` object from a `RowSetFactory` because the reference implementation may be changed in future.

4. Populating Data to a CachedRowSet

There are two ways for populating data from the database to a `CachedRowSet` object:

- Populate data from an existing `ResultSet` object.

- Populate data by executing a SQL command.

Let's see each way in details.

Populate data to a `CachedRowSet` object from a ResultSet object:

Given a `ResultSet` object which is created from a `Statement`, the following code populates data from the result set to the cached row set:

```java
ResultSet result = statement.executeQuery(sql);

RowSetFactory factory = RowSetProvider.newFactory();
CachedRowSet rowset = factory.createCachedRowSet();

rowset.populate(result);
```

Now you can close the connection and still be able to scroll through rows in the rowset.

Populate data to a `CachedRowSet` object by executing SQL command:

In this case, you need to set database connection properties and SQL statement for the `CachedRowSet` object, and then call the **`execute()`** method. For example:

```java
String url = "jdbc:mysql://localhost:3306/college";
String username = "root";
String password = "password";

RowSetFactory factory = RowSetProvider.newFactory();
CachedRowSet rowset = factory.createCachedRowSet();

rowset.setUrl(url);
rowset.setUsername(username);
rowset.setPassword(password);
rowset.setCommand(sql);

rowset.execute();
```

After the `CachedRowSet` object is populated , you can iterate over its rows by using `ResultSet`'s methods because `CachedRowSet` extends `ResultSet`. For example, the following code snippet iterates all rows in the row set and print details of each row:

```
rowset.populate(result);

while (rowset.next()) {
    String name = rowset.getString("name");
    String email = rowset.getString("email");
    String major = rowset.getString("major");

    System.out.printf("%s - %s - %s\n", name, email, major);
}
```

5. Modifying Data in a CachedRowSet

You can make changes to data (insert, update and delete rows) in a `CachedRowSet` object just like you do with an updatable `ResultSet`. But the changes are not immediately reflected in the database until you explicitly request to accept changes.

Before making any changes, you must set table name for the row set so it knows the table needs to be updated:

```
rowset.setTableName("student");
```

For example, the following code snippet updates the 5th row in the row set:

```
rowset.absolute(5);

rowset.updateString("name", name);
rowset.updateString("email", email);
rowset.updateString("major", major);

rowset.updateRow();
```

The following code inserts a new row to the row set:

```
rowset.moveToInsertRow();

rowset.updateNull("student_id");
rowset.updateString("name", name);
rowset.updateString("email", email);
rowset.updateString("major", major);

rowset.insertRow();
rowset.moveToCurrentRow();
```

Note that you must call `updateNull(column_name)` for the primary key column of the table if that column's values are auto-generated. Otherwise an exception throws.

And the following code removes the current row in the row set:

```
rowset.deleteRow();
```

6. Committing Changes to the Database

To actually save the accumulated changes (update, insert and delete rows) to the underlying database, call:

```
rowset.acceptChanges();
```

If the `CachedRowSet` object is populated from a `ResultSet` object, pass the `Connection` object to the method:

```
rowset.acceptChanges(connection);
```

Note that the **`acceptChanges()`** method throws `SyncProviderException` if it found conflicts when trying to synchronize with the database. So you must handle this exception. Also make sure to disable auto commit mode:

```java
connection.setAutoCommit(false);
```

7. A Complete CachedRowSet Example Program

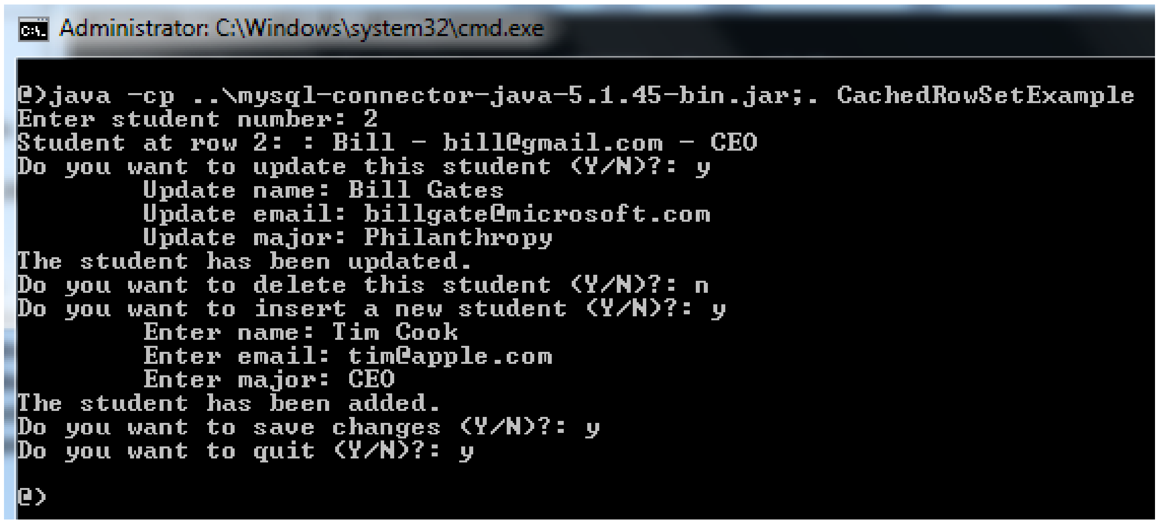
Let’s see a complete program that demonstrates how to use `CachedRowSet`. The following program populates rows from `student` table in a MySQL database named `college`. Then it asks the user to update, delete and insert rows interactively.

Here’s the code of the program:

```java
import java.sql.*;
import javax.sql.rowset.*;
import javax.sql.rowset.spi.*;
import java.io.*;

/**
 * This program demonstrates how to use CachedRowSet in JDBC.
 *
 * @author www.codejava.net
 */
public class CachedRowSetExample {
    static Console console = System.console();
    static String answer;
    static boolean quit = false;

    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/college";
        String username = "root";
        String password = "password";

        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            conn.setAutoCommit(false);

            String sql = "SELECT * FROM student";

            Statement statement = conn.createStatement();

            ResultSet result = statement.executeQuery(sql);

            RowSetFactory factory = RowSetProvider.newFactory();
            CachedRowSet rowset = factory.createCachedRowSet();

            rowset.setTableName("student");

            rowset.populate(result);

            while (!quit) {
                if (!readStudent(rowset)) continue;

                updateStudent(rowset);

                deleteStudent(rowset);

                insertStudent(rowset);

                saveChanges(rowset, conn);

                askToQuit();

            }

        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
            ex.printStackTrace();
        }

    }

    static void readStudentInfo(String position, ResultSet result)
            throws SQLException {
        String name = result.getString("name");
```

```java
            String email = result.getString("email");
            String major = result.getString("major");

            String studentInfo = "%s: %s - %s - %s\n";
            System.out.format(studentInfo, position, name, email, major);
        }

        static boolean readStudent(ResultSet result) throws SQLException {
            int row = Integer.parseInt(console.readLine("Enter student number: "));

            if (result.absolute(row)) {
                readStudentInfo("Student at row " + row + ": ", result);
                return true;
            } else {
                System.out.println("There's no student at row " + row);
                return false;
            }
        }

        static void updateStudent(ResultSet result) throws SQLException {
            answer = console.readLine("Do you want to update this student (Y/N)?: ");

            if (answer.equalsIgnoreCase("Y")) {
                String name = console.readLine("\tUpdate name: ");
                String email = console.readLine("\tUpdate email: ");
                String major = console.readLine("\tUpdate major: ");

                if (!name.equals("")) result.updateString("name", name);
                if (!email.equals("")) result.updateString("email", email);
                if (!major.equals("")) result.updateString("major", major);

                result.updateRow();

                System.out.println("The student has been updated.");
            }

        }

        static void deleteStudent(ResultSet result) throws SQLException {
            answer = console.readLine("Do you want to delete this student (Y/N)?: ");

            if (answer.equalsIgnoreCase("Y")) {
                result.deleteRow();

                System.out.println("The student has been removed.");
            }

        }

        static void insertStudent(ResultSet result) throws SQLException {
            answer = console.readLine("Do you want to insert a new student (Y/N)?: ");

            if (answer.equalsIgnoreCase("Y")) {
                String name = console.readLine("\tEnter name: ");
                String email = console.readLine("\tEnter email: ");
                String major = console.readLine("\tEnter major: ");

                result.moveToInsertRow();

                result.updateNull("student_id");
                result.updateString("name", name);
                result.updateString("email", email);
                result.updateString("major", major);

                result.insertRow();
                result.moveToCurrentRow();

                System.out.println("The student has been added.");
            }

        }

        static void saveChanges(CachedRowSet rowset, Connection conn) {
            answer = console.readLine("Do you want to save changes (Y/N)?: ");

            if (answer.equalsIgnoreCase("Y")) {
                try {
                    rowset.acceptChanges(conn);
                } catch (SyncProviderException ex) {
                    System.out.println("Error commiting changes to the database: " + ex);
                }
            }
        }

        static void askToQuit() {
            answer = console.readLine("Do you want to quit (Y/N)?: ");
            quit = answer.equalsIgnoreCase("Y");
        }
    }
```

The following screenshot illustrates how to run and use the program:



```
@>java -cp ..\mysql-connector-java-5.1.45-bin.jar;. CachedRowSetExample
Enter student number: 2
Student at row 2: : Bill - bill@gmail.com - CEO
Do you want to update this student (Y/N)?: y
        Update name: Bill Gates
        Update email: billgate@microsoft.com
        Update major: Philanthropy
The student has been updated.
Do you want to delete this student (Y/N)?: n
Do you want to insert a new student (Y/N)?: y
        Enter name: Tim Cook
        Enter email: tim@apple.com
        Enter major: CEO
The student has been added.
Do you want to save changes (Y/N)?: y
Do you want to quit (Y/N)?: y

@>
```

References:

RowSet Javadoc

CachedRowSet Javadoc

ResultSet Javadoc

Recommended Course: **Complete Java Master Class**