Theory: JDBC Statements

620 users solved this topic. Latest completion was about 4 hours ago.

18 minutes 0 / 5 problems solved

Skip this topic Start practicing

§1. What is a Statement

We have established a connection with a database in the previous topic, and now we are ready to add records to the database tables and retrieve results from them. To perform actions on a database, we need to use SQL statements. An interface `java.sql.Statement` represents such statements in the JDBC API.

At first, we need to establish a connection with the database in order to execute statements from our application. Then should create a `Statement` object using a `Connection` object. More precisely, we need to call the `createStatement()` method of the `Connection` that creates a `Statement`.

§2. Statement execution

Once the `Statement` object is created, we can execute SQL statements by calling its execution methods. The most generic method is `execute(String sql)`. It performs a given SQL statement and returns `true` if there is a return data, otherwise, the method returns `false`. For example, for the SELECT statement it returns `true` and for the INSERT statement `false`.

However, the `Statement` interface has other more specific execution methods. One of them is `executeUpdate(String sql)`. Unlike the `execute` the `executeUpdate` method returns the number of rows affected by the SQL statement.

> Use `executeUpdate` method for INSERT, DELETE and UPDATE statements or for statements that return nothing, such as CREATE or DROP.

Let's create an SQLite database westeros.db and then create a table of the Greater Houses of the Seven Kingdoms using the `executeUpdate` method.

Current topic:

JDBC Statements

Topic depends on:

- SELECT FROM statement
- Basic UPDATE statement
- Basic CREATE statement
- Basic INSERT statement
- Connecting to a database with JDBC

Topic is required for:

JDBC Prepared Statements

Table of contents:

↑ JDBC Statements

§1. What is a Statement

§2. Statement execution

§3. Processing ResultSet

§4. Conclusion

Feedback & Comments

```
public class Westeros {
    public static void main(String[] args) {
        String url = "jdbc:sqlite:C:/sqlite/westeros.db";

        SQLiteDataSource dataSource = new SQLiteDataSource();
        dataSource.setUrl(url);

        try (Connection con = dataSource.getConnection()) {
            // Statement creation

            try (Statement statement = con.createStatement()) {

                // Statement execution


        statement.executeUpdate("CREATE TABLE IF NOT EXISTS HOUSES(" +

                        "id INTEGER PRIMARY KEY," +

                        "name TEXT NOT NULL," +

                        "words TEXT NOT NULL)");

            } catch (SQLException e) {

                e.printStackTrace();

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

}
```

Since JDBC spec required `Statement` be closed when no longer reachable, we have used the try-with-resources statement for creating `Statement` objects.

Once we execute the program above, we will create a table `HOUSES` that stores an `id` of the house, its `name`, and `words`. Now, let's add several houses to the table. For that, we will use `executeUpdate` again and add the following code:

```
int i = statement.executeUpdate("INSERT INTO HOUSES VALUES " +
        "(1, 'Targaryen of King''s Landing', 'Fire and Blood')," +
        "(2, 'Stark of Winterfell', 'Summer is Coming')," +
        "(3, 'Lannister of Casterly Rock', 'Hear Me Roar!')");
```

As you can guess, the value of `i` will be equal to 3, since we have inserted 3 houses to the database.

Note, `executeUpdate` method requires to wrap ***text*** values into a single quote character (`'`). If the value contains this character, you have to replace it with double single quotes (`''`) to be parsed correctly.

Since the real words of the Stark of Winterfell house are "Winter is coming", we have to update it. For that, we will execute the SQL UPDATE statement using `executeUpdate` method:

```
int u = statement.executeUpdate("UPDATE HOUSES " +
        "SET words = 'Winter is coming' " +
        "WHERE id = 2");
```

Since we've updated only one record the value of `u` will be equal to 1.

Now, when you've created several records with Great Houses of Westeros, we would definitely need to retrieve it back from the database. For that, we need to execute the SQL SELECT statement. The appropriate `Statement`

method for the execution of SELECT statements is `executeQuery(String sql)`. This method is similar to the already discussed methods, however, it returns a `ResultSet` object. The `ResultSet` object represents a table that contains records from the database result set.

§3. Processing ResultSet

For processing `ResultSet`, we can use its `next()` method. Each call of the `next()` moves a pointer to the record forward one position, starting from the first record. For retrieving column values we will use `ResultSet` getter methods of the appropriate type. For example, for the column with a `TEXT` type and `INTEGER` type, we can use `getString` and `getInt` methods respectively. ResultSet getters can accept two types of arguments: column index (starting from 1) and column label.

> It is possible to use `getString` getter for retrieving columns values with any type. However, in that case, the value will be converted to the `java.lang.String` type.

Let's look at the example, where we retrieve and print all records from the `HOUSES` table one by one. For that we need to add the following code:

```
try (ResultSet greatHouses = statement.executeQuery("SELECT * FROM HOUSES")) {
        while (greatHouses.next()) {
            // Retrieve column values
            int id = greatHouses.getInt("id");
            String name = greatHouses.getString("name");
            String words = greatHouses.getString("words");

            System.out.printf("House %d%n", id);
            System.out.printf("\tName: %s%n", name);

            System.out.printf("\tWords: %s%n", words);

        }

    }
```

> Since JDBC spec required `ResultSets` be closed when no longer reachable, we have used the try-with-resources statement for creating `ResultSet` objects.

Note that we have called the `next()` method inside the while loop. Since the `next()` method returns a boolean value (`true` if there are more records in the `ResultSet`), calling this method inside a while loop is a convenient way to process the `ResultSet`.

§4. Conclusion

To sum up, we can use the `Statement` JDBC interface for the execution of SQL statements. There are 3 methods that can execute statements:

- `execute(String sql)` the most generic method
- `executeUpdate(String sql)` that we should use for execution INSERT, UPDATE and DELETE statements and statements that return nothing
- `executeQuery(String sql)` method that is recommended to use with a SELECT statement. The return type of `executeQuery` method is a `ResultSet` object that represents a table of records returned by the executed statement.

Later we will discuss the drawbacks of the `Statement` interface and present the way to cope with them by using the `PreparedStatement`.

Report a typo

64 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (6) Hints (2) Useful links (0) Show discussion