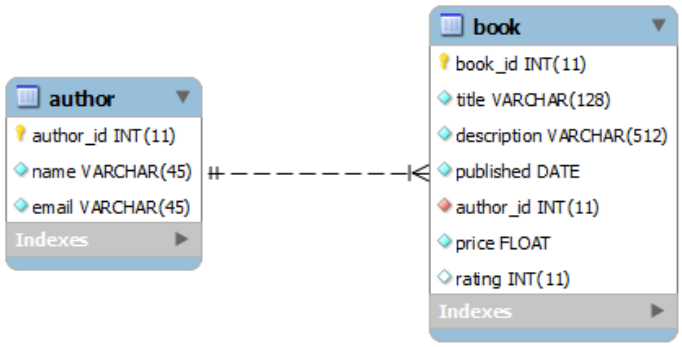
JDBC Examples for Calling Stored Procedures (MySQL)

This Java tutorial provides various examples that help you understand how to work with stored procedure using JDBC. You will learn how to:

- Calling a simple stored procedure which has only IN parameters.
- Creating a stored procedure from Java.
- Calling a stored procedure which has IN, OUT and INOUT parameters and retrieve the values of these parameters.
- Calling a stored procedure which returns a result set and process this result set.

We will use MySQL database for the examples, and suppose that you know how to create a stored procedure using MySQL Workbench tool. Here's the database structure:



Imagine we have some dummy data for the table **author** as following:

author_id	name	email
10	Cay Horstmann	horstmann@gmail.com
11	Dane Cameron	cameron@gmail.com
12	Richard Warburton	richard@gmail.com
13	Bruce Eckel	bruce@gmail.com
14	Joshua Bloch	bloch@gmail.com
NULL	NULL	NULL

And dummy data for the table **book** as following:

book_id	title	description	published	author_id	price	rating
7	Thinking in Java	Teach you core Java in depth	2012-09-12	13	39	5
8	Effective Java	Core Java Best Practices	2008-05-08	14	33.63	4
9	Java SE 8 for the Really Impatient	Master Java 8	2014-01-24	10	37	4
10	Java 8: The Fundamentals	Learn the fundamentals of Java 8	2014-03-15	11	28	4
11	Java 8 Lambdas	Mastering Pragmatic Functional Programming	2014-04-07	12	32	4
12	Java Puzzlers	Java Traps, Pitfalls, and Corner Cases	2005-07-04	14	40.2	5
13	Core Java for the Impatient	Mastering Core Java	2015-02-12	10	35.96	3
14	Object-Oriented Design and Patterns	Mastering OOP design	2005-06-02	10	122.35	4
15	Thinking in C++	Mastering C++	2000-03-25	13	59.13	5

1. Calling a Simple Stored Procedure from Java

In MySQL Workbench, create a new routine (expand the database and you see a node called **Routines**. Right click and select **Create Routine...)** and paste the following code:

```
CREATE PROCEDURE `booksdb`.`create_author` (IN name VARCHAR(45), email VARCHAR(45))
BEGIN
    DECLARE newAuthorID INT;

    INSERT INTO author (name, email) VALUES (name, email);

    SET newAuthorID = (SELECT author_id FROM author a WHERE a.name = name);

    INSERT INTO book (title, description, published, author_id, price, rating)
        VALUES (CONCAT('Life Story of ', name),
                CONCAT('Personal Stories of ', name),
                date('2016-12-30'), newAuthorID, 10.00, 0);
END
```

As you can see, this stored procedure is named as ***create_author***. It has two input parameters ***name*** and ***email***.

In the body of the procedure (code between **BEGIN** and **END**), we insert a row into the table **author**. Then select the ID value of this recently inserted row (**author_id**), store it into a variable named **newAuthorID**. Then we insert a new row into the table **book**, in which we use the

author name for the title and description of the book. Notice that the variable **newAuthorID** is used in the second INSERT statement to set foreign key for the new row.

Within the workbench, you can call this stored procedure by executing the following query:

```
call create_author('Patrick Maka', 'patrick@gmail.com')
```

Now, let's see how to call this stored procedure using JDBC.

Here are the steps to call a simple stored procedure from Java code with JDBC:

```
CallableStatement statement = connection.prepareCall("{call procedure_name(?, ?, ?)}");

// setting input parameters on the statement object
// statement.setString(parameterIndex, parameterValue);

statement.execute();
statement.close();
```

Notice the **CALL** statement syntax:

```
"{call procedure_name(?, ?, ?)}
```

The procedure's parameters are denoted by the question marks, separated by comma. Then we use the `setXXX()` methods on the statement object to set value for the parameters, just like setting parameters for a `PreparedStatement`.

Invoking `execute()` method on the statement object will run the specified stored procedure. This method returns true if the stored procedure returns a result set, false if not, and throw `SQLException` in cases of an error occurred.

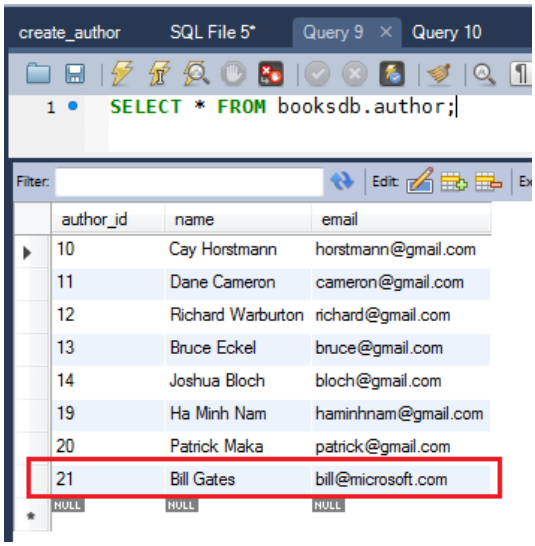
The following is a test Java program that calls the stored procedure **create_author** which we created previously:

```
import java.sql.*;

/**
 * A Java program demonstrates how to call a MySQL stored procedure
 * using JDBC.
 *
 * @author www.codejava.net
 */
public class StoredProcedureCallExample1 {

    public static void main(String[] args) {
        String dbURL = "jdbc:mysql://localhost:3306/booksdb";
        String user = "root";
        String password = "P@ssw0rd";


        try (

            Connection conn = DriverManager.getConnection(dbURL, user, password);

            CallableStatement statement = conn.prepareCall("{call create_author(?, ?)}");
        ) {

            statement.setString(1, "Bill Gates");
            statement.setString(2, "bill@microsoft.com");

            statement.execute();
            statement.close();

            System.out.println("Stored procedure called successfully!");

        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}
```

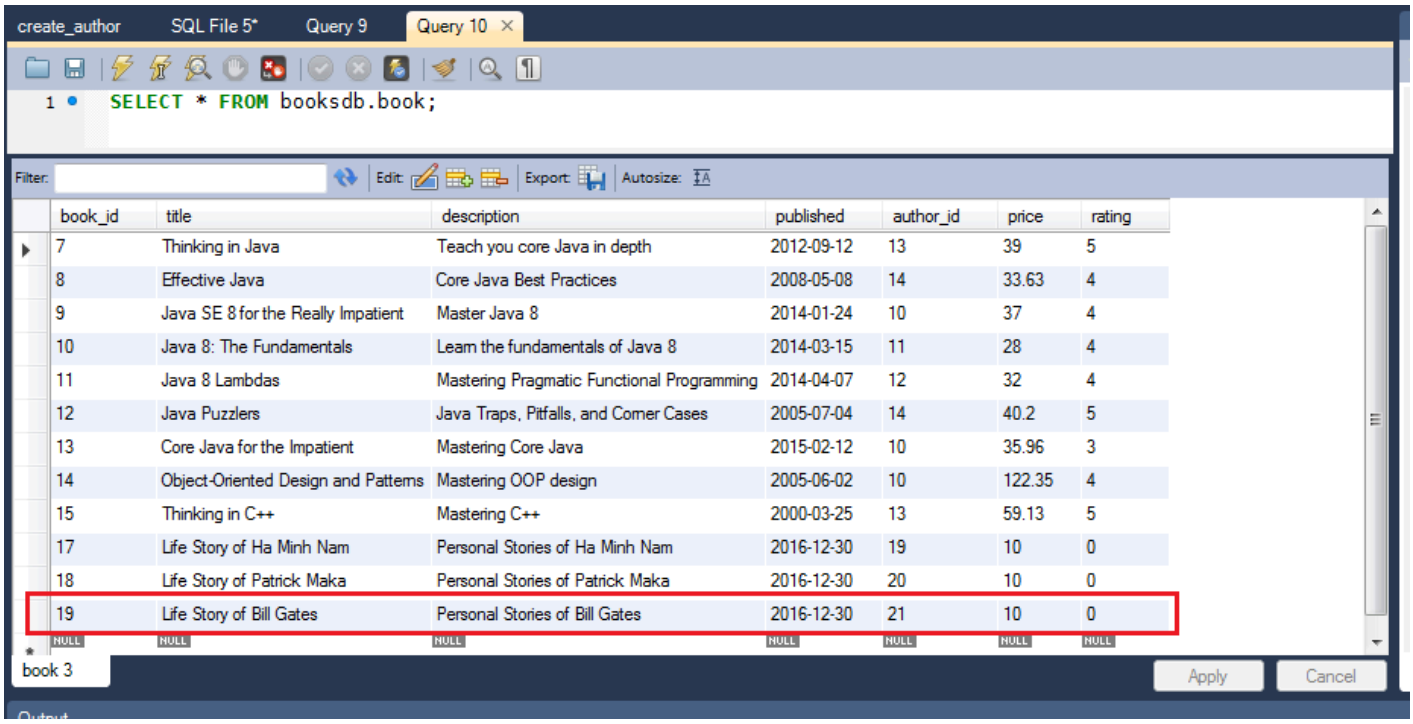
Compile and run this program. You should see the following output:

```
Stored procedure called successfully!
```

Let's verifying the database. Querying all rows from the table **author** we see a new row was added:



And checking the table **book** also lets us see a new row added:



2. Creating a Stored Procedure from Java

Besides using a database tool like MySQL Workbench, we can create a stored procedure from within a Java program by executing the "`CREATE PROCEDURE`" SQL statement, just like executing a normal SQL statement.

The following Java program creates a simple MySQL stored procedure called **delete_book** which removes a row from the table book based on the specified book ID:

```
1  import java.sql.*;
2
3  /**
4   * A Java program demonstrates how to create a MySQL stored procedure
5   * using JDBC.
6   *
7   * @author www.codejava.net
8   */
9  public class StoredProcedureCreateExample {
10
11     public static void main(String[] args) {
12         String dbURL = "jdbc:mysql://localhost:3306/booksdb";
13         String user = "root";
14         String password = "P@ssw0rd";
15
16         try (
17             Connection conn = DriverManager.getConnection(dbURL, user, password);
18
19             Statement statement = conn.createStatement();
```

```
        ) {

            String queryDrop = "DROP PROCEDURE IF EXISTS delete_book";

            String queryCreate = "CREATE PROCEDURE delete_book (IN bookID INT) ";
            queryCreate += "BEGIN ";
            queryCreate += "DELETE FROM book WHERE book_id = bookID; ";
            queryCreate += "END";

            // drops the existing procedure if exists
            statement.execute(queryDrop);

            // then creates a new stored procedure
            statement.execute(queryCreate);

            statement.close();

            System.out.println("Stored procedure created successfully!");

        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}
```

Note that we have to execute two queries: the first one is to drop the stored procedure if exists; and the second actually creates the stored procedure.

Running this program would produce the following output:

```
Stored procedure created successfully!
```

Switch to MySQL Workbench and refresh the *Object Browser* pane, you should see the newly created stored procedure appears there.

3. Calling a Stored Procedure Having OUT and INOUT parameters from Java

Consider the following stored procedure:

```
CREATE PROCEDURE `summary_report`(
        IN title VARCHAR(45),
        OUT totalBooks INT,
        OUT totalValue DOUBLE,
        INOUT highPrice DOUBLE
    )
BEGIN
    DECLARE maxPrice DOUBLE;

    SELECT COUNT(*) AS bookCount, SUM(price) as total
        FROM book b JOIN author a ON b.author_id = a.author_id
        AND b.title LIKE CONCAT('%', title, '%')
    INTO totalBooks, totalValue;


    SELECT MAX(price) FROM book WHERE price INTO maxPrice;

    IF (maxPrice > highPrice) THEN
        SET highPrice = maxPrice;
    END IF;
END
```

This stored procedure has 4 parameters:

- `IN title VARCHAR(45)`: input parameter. The procedure searches for books whose titles contain the words specified by this parameter.
- `OUT totalBooks INT`: The procedure counts total of the matching books and stores the value into this output parameter.
- `OUT totalValue DOUBLE`: The procedure counts total value of the matching books and stores the value into this output parameter.
- `INOUT highPrice DOUBLE`: This is both input/output parameter. The procedure selects the max price in all books and if it is greater than the parameter value, assigns it to the parameter.

Now, let's see how to execute this stored procedure using JDBC code.

To retrieve the values of the **`OUT`** and **`INOUT`** parameters, JDBC requires these parameters must be registered before calling the procedure, by invoking the following method on `CallableStatement`object:

`void registerOutParameter(int parameterIndex, int sqlType)`

For example, the following code registers 3 output parameters for the procedure **summary_report** above:

```
CallableStatement statement = conn.prepareCall("{call summary_report(?, ?, ?, ?)}");
```

```
2
3  statement.registerOutParameter(2, Types.INTEGER);
4  statement.registerOutParameter(3, Types.DOUBLE);
5  statement.registerOutParameter(4, Types.DOUBLE);
```

After the procedure has been called, we can use the `getXXX()` method on the `CallableStatement`object to retrieve the values of the output parameters. For example, the following code gets values of the 3 output parameters returned by the procedure **summary_report**:

```
1  Integer totalBook = (Integer) statement.getObject(2, Integer.class);
2  Double totalValue = statement.getDouble(3);
3  Double highPrice = statement.getDouble("highPrice");
```

As you can see, there are three ways to retrieve the values: by index and type; by index; and by parameter name.

And following is full source code of a test program:

```
1  import java.sql.*;
2
3  /**
4   * A Java program demonstrates how to use JDBC to call a MySQL stored procedure
5   * and retrieve values of the OUT and INOUT parameters.
6   *
7   * @author www.codejava.net
8   */
9  public class StoredProcedureCallExample2 {
10
11     public static void main(String[] args) {
12         String dbURL = "jdbc:mysql://localhost:3306/booksdb";
13         String user = "root";
14         String password = "P@ssw0rd";
15
16         try (
17             Connection conn = DriverManager.getConnection(dbURL, user, password);
18             CallableStatement statement = conn.prepareCall("{call summary_report(?, ?, ?, ?)}");
19         ) {
20
21             statement.registerOutParameter(2, Types.INTEGER);
22             statement.registerOutParameter(3, Types.DOUBLE);
23             statement.registerOutParameter(4, Types.DOUBLE);
24
25
26             statement.setString(1, "Java");
27             statement.setDouble(4, 50);
28
29             statement.execute();
30
31             Integer totalBook = (Integer) statement.getObject(2, Integer.class);
32             Double totalValue = statement.getDouble(3);
33             Double highPrice = statement.getDouble("highPrice");
34
35             System.out.println("Total books: " + totalBook);
36             System.out.println("Total value: " + totalValue);
37             System.out.println("High price: " + highPrice);
38
39
40             statement.close();
41
42         } catch (SQLException ex) {
43             ex.printStackTrace();
44         }
45     }
46 }
```

Running this program would give the following output:

```
Total books: 7

Total value: 245.79000091552734

High price: 122.3499984741211
```

4. Calling a Stored Procedure Returning a Result Set from Java

A stored procedure can returns a result set. Consider the following procedure:

```
1  CREATE PROCEDURE `get_books`(IN rate INT)
2  BEGIN
3      SELECT * FROM book WHERE rating >= rate;
4  END
```

Let's see how to retrieve this result set in Java. The following code snippet shows you how to retrieve and process a result set returned from a stored procedure using JDBC code:

```
CallableStatement statement = conn.prepareCall("{call get_books(?)}");

statement.setInt(1, 5);

boolean hadResults = statement.execute();

while (hadResults) {
    ResultSet resultSet = statement.getResultSet();


    // process result set
    while (resultSet.next()) {
        // retrieve values of fields
String title = resultSet.getString("title");

    }

    hadResults = statement.getMoreResults();
}
```

And here is the full source code of a demo program:

```
import java.sql.*;

/**
 * A Java program demonstrates how to use JDBC to call a MySQL stored procedure
 * that returns a result set and process this result set.
 *
 * @author www.codejava.net
 */
public class StoredProcedureCallExample3 {

    public static void main(String[] args) {
        String dbURL = "jdbc:mysql://localhost:3306/booksdb";
        String user = "root";
        String password = "P@ssw0rd";

        try (
            Connection conn = DriverManager.getConnection(dbURL, user, password);
            CallableStatement statement = conn.prepareCall("{call get_books(?)}");
        ) {

            statement.setInt(1, 5);

            boolean hadResults = statement.execute();

            // print headings
            System.out.println("| Title | Description | Rating |");
            System.out.println("==================================");

            while (hadResults) {
                ResultSet resultSet = statement.getResultSet();

                // process result set
                while (resultSet.next()) {
                    String title = resultSet.getString("title");
                    String description = resultSet.getString("description");
                    int rating = resultSet.getInt("rating");

                    System.out.println(
                            "| " + title + " | " + description + " | " + rating + " |");
                }

                hadResults = statement.getMoreResults();
            }

            statement.close();

        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}
```

Running this program would print the following output:

```
| Title | Description | Rating |
```

| Thinking in Java | Teach you core Java in depth | 5 |

| Java Puzzlers | Java Traps, Pitfalls, and Corner Cases | 5 |

| Thinking in C++ | Mastering C++ | 5 |