6.3 Using JDBC `CallableStatements` to Execute Stored Procedures

Starting with MySQL server version 5.0 when used with Connector/J 3.1.1 or newer, the `java.sql.CallableStatement` interface is fully implemented with the exception of the `getParameterMetaData()` method.

For more information on MySQL stored procedures, please refer to Using Stored Routines (Procedures and Functions).

Connector/J exposes stored procedure functionality through JDBC's`CallableStatement` interface.

Note

Current versions of MySQL server do not return enough information for the JDBC driver to provide result set metadata for callable statements. This means that when using`CallableStatement`, `ResultSetMetaData` may return`NULL`.

The following example shows a stored procedure that returns the value of`inOutParam` incremented by 1, and the string passed in using `inputParam`as a `ResultSet`:

Example 6.3 Connector/J: Calling Stored Procedures

```
CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), \
                        INOUT inOutParam INT)
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;

    SELECT inputParam;

    SELECT CONCAT('zyxw', inputParam);
END
```

To use the `demoSp` procedure with Connector/J, follow these steps:

1. Prepare the callable statement by using`Connection.prepareCall()`.

   Notice that you have to use JDBC escape syntax, and that the parentheses surrounding the parameter placeholders are not optional:

Example 6.4 Connector/J: Using `Connection.prepareCall()`

```
import java.sql.CallableStatement;

...

    //
    // Prepare a call to the stored procedure 'demoSp'
    // with two parameters
    //
    // Notice the use of JDBC-escape syntax ({call ...})
    //

    CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");



    cStmt.setString(1, "abcdefg");
```

> **Note**
>
> `Connection.prepareCall()` is an expensive method, due to the metadata retrieval that the driver performs to support output parameters. For performance reasons, minimize unnecessary calls to`Connection.prepareCall()` by reusing`CallableStatement` instances in your code.

2. Register the output parameters (if any exist)

To retrieve the values of output parameters (parameters specified as`OUT` or `INOUT` when you created the stored procedure), JDBC requires that they be specified before statement execution using the various`registerOutputParameter()` methods in the `CallableStatement`interface:

Example 6.5 Connector/J: Registering output parameters

```
import java.sql.Types;
...
//
// Connector/J supports both named and indexed
// output parameters. You can register output
// parameters using either method, as well
// as retrieve output parameters using either
// method, regardless of what method was
// used to register them.
//
// The following examples show how to use
// the various methods of registering
// output parameters (you should of course
// use only one registration per parameter).
//

//
// Registers the second parameter as output, and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter(2, Types.INTEGER);

//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter("inOutParam", Types.INTEGER);
...
```

3. Set the input parameters (if any exist)

   Input and in/out parameters are set as for `PreparedStatement`objects. However, `CallableStatement` also supports setting parameters by name:

   Example 6.6 Connector/J: Setting `CallableStatement` input parameters

```
...

    //
    // Set a parameter by index
    //

    cStmt.setString(1, "abcdefg");

    //
    // Alternatively, set a parameter using
    // the parameter name
    //

    cStmt.setString("inputParameter", "abcdefg");

    //
    // Set the 'in/out' parameter using an index
    //

    cStmt.setInt(2, 1);
    //
    // Alternatively, set the 'in/out' parameter
    // by name
    //

    cStmt.setInt("inOutParam", 1);

...
```

4. Execute the `CallableStatement`, and retrieve any result sets or output parameters.

   Although `CallableStatement` supports calling any of the `Statement`execute methods (`executeUpdate()`, `executeQuery()` or`execute()`), the most flexible method to call is `execute()`, as you do not need to know ahead of time if the stored procedure returns result sets:

   Example 6.7 Connector/J: Retrieving results and output parameter values

```
...

    boolean hadResults = cStmt.execute();

    //
    // Process all returned result sets
    //

    while (hadResults) {
        ResultSet rs = cStmt.getResultSet();

        // process result set
        ...

        hadResults = cStmt.getMoreResults();
    }

    //
    // Retrieve output parameters
    //
    // Connector/J supports both index-based and
    // name-based retrieval
    //

    int outputValue = cStmt.getInt(2); // index-based

    outputValue = cStmt.getInt("inOutParam"); // name-based

...
```

User Comments

Posted by Rajani S on August 30, 2016

Edit | Delete

Hi,

The statement "cStmt.setString("inputParameter", "abcdefg");" throws the following error:

```
Exception in thread "main" java.lang.NullPointerException
at com.mysql.jdbc.CallableStatement.getNamedParamIndex(CallableStatement.java:1381)
at com.mysql.jdbc.CallableStatement.setString(CallableStatement.java:2165)
```

It is supposed to be "cStmt.setString("inputParam", "abcdefg");"

Thanks