**JDBC CallableStatement Stored Procedures Example**

## CallableStatement with Single ResultSet Example

Consider the following MySQL stored procedure.

DELIMITER $$

USE `mysql\_database`$$

CREATE PROCEDURE `retreive\_users` ()

BEGIN

select \* from users;

END$$

DELIMITER ;

The following program demonstrates how to call the **retreive\_users()** stored procedure and generate a single result set.

package com.javaguides.jdbc.storedprocedure;

import java.sql.CallableStatement;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

/\*\*

\* Simple CallableStatement Example

\* @author Ramesh Fadatare

\*

\*/

public class SimpleCallableStatementExample {

public static void main(String[] args) {

String jdbcUrl = "jdbc:mysql://localhost:3306/mysql\_database?useSSL=false";

String username = "root";

String password = "root";

String sql = "call retreive\_users()";

try (Connection conn = DriverManager.getConnection(jdbcUrl, username, password); CallableStatement stmt = conn.prepareCall(sql); ResultSet rs = stmt.executeQuery();) {

while (rs.next()) {

System.out.println("ID = " + rs.getInt(1) + ", NAME = " + rs.getString(2) + ", Email = " +

rs.getString(3) + ", Country = " + rs.getString(4) + ", Password = " + rs.getString(5));

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

Output:

ID = 1, NAME = A, Email = tony@gmail.com, Country = US, Password = secret

ID = 2, NAME = Pramod, Email = pramod@gmail.com, Country = India, Password = 123

ID = 3, NAME = A, Email = a@gmail.com, Country = India, Password = 123

ID = 4, NAME = B, Email = b@gmail.com, Country = India, Password = 123

ID = 5, NAME = C, Email = c@gmail.com, Country = India, Password = 123

ID = 6, NAME = D, Email = d@gmail.com, Country = India, Password = 123

## CallableStatement with Multiple ResultSet Example

Consider the following MySQL stored procedure having multiple select statements.

DELIMITER $$

USE `mysql\_database`$$

BEGIN

select distinct name from users where id = 1;

select distinct email from users;

select count(id) as users\_count from users;

END

DELIMITER ;

The following program demonstrates how to call the **retreive\_different\_results()** stored procedure and get multiple result set.

package com.javaguides.jdbc.storedprocedure;

import java.sql.CallableStatement;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

/\*\*

\* CallableStatement Example

\* @author Ramesh Fadatare

\*

\*/

public class CallableMultipleResultSetExample {

public static void main(String[] args) {

String jdbcUrl = "jdbc:mysql://localhost:3306/mysql\_database?useSSL=false";

String username = "root";

String password = "root";

String sql = "call retreive\_different\_results()";

try (Connection conn = DriverManager.getConnection(jdbcUrl, username, password); CallableStatement stmt = conn.prepareCall(sql);) {

boolean hasRs = stmt.execute();

System.out.println();

// Get Product Names

if (hasRs) {

try (ResultSet rs = stmt.getResultSet()) {

while (rs.next()) {

System.out.println("NAME = " + rs.getString(1));

}

}

}

// Get Total Price

if (stmt.getMoreResults()) {

try (ResultSet rs = stmt.getResultSet()) {

if (rs.next()) {

System.out.println("Email = " + rs.getString(1));

}

}

}

// Get Max/Min Price

if (stmt.getMoreResults()) {

try (ResultSet rs = stmt.getResultSet()) {

if (rs.next()) {

System.out.println("Users count = " + rs.getInt(1));

}

}

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

## JDBC Statement Batch Insert Example

The [**Statement interface**](http://www.javaguides.net/2018/10/jdbc-statement-interface.html) provides below two methods to perform batch operations

1. addBatch(String sql)
2. executeBatch()

### Statement.addBatch(String sql)

Adds the given SQL command to the current list of commands for this Statement object. The commands in this list can be executed as a batch by calling the method executeBatch.

### Statement.executeBatch()

Submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts. The int elements of the array that is returned are ordered to correspond to the commands in the batch, which are ordered according to the order in which they were added to the batch.

The below example demonstrates the batch insertion operations using [**Statement interface**](http://www.javaguides.net/2018/10/jdbc-statement-interface.html):

package com.javaguides.jdbc.batch;

import java.sql.BatchUpdateException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Arrays;

/\*\*

\* Insert Batch operation using Statement Interface

\* @author Ramesh Fadatare

\*

\*/

public class BatchInsertExample {

public static void main(String[] args) {

batchUpdate();

}

private static void batchUpdate() {

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "root");

// Step 2:Create a statement using connection object

Statement statement = connection.createStatement()) {

connection.setAutoCommit(false);

statement.addBatch("INSERT INTO Users VALUES (2, 'Pramod', 'pramod@gmail.com', 'India', '123');");

statement.addBatch("INSERT INTO Users VALUES (3, 'A', 'a@gmail.com', 'India', '123');");

statement.addBatch("INSERT INTO Users VALUES (4, 'B', 'b@gmail.com', 'India', '123');");

statement.addBatch("INSERT INTO Users VALUES (5, 'C', 'c@gmail.com', 'India', '123');");

statement.addBatch("INSERT INTO Users VALUES (6, 'D', 'd@gmail.com', 'India', '123');");

int[] updateCounts = statement.executeBatch();

System.out.println(Arrays.toString(updateCounts));

connection.commit();

} catch (BatchUpdateException batchUpdateException) {

printBatchUpdateException(batchUpdateException);

} catch (SQLException e) {

printSQLException(e);

}

}

public static void printSQLException(SQLException ex) {

for (Throwable e: ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

public static void printBatchUpdateException(BatchUpdateException b) {

System.err.println("----BatchUpdateException----");

System.err.println("SQLState: " + b.getSQLState());

System.err.println("Message: " + b.getMessage());

System.err.println("Vendor: " + b.getErrorCode());

System.err.print("Update counts: ");

int[] updateCounts = b.getUpdateCounts();

for (int i = 0; i < updateCounts.length; i++) {

System.err.print(updateCounts[i] + " ");

}

}

}

Output:

[1, 1, 1, 1, 1]

## JDBC Statement Batch Insert Example

The [**Statement interface**](http://www.javaguides.net/2018/10/jdbc-statement-interface.html) provides below two methods to perform batch operations

1. addBatch(String sql)
2. executeBatch()

### Statement.addBatch(String sql)

Adds the given SQL command to the current list of commands for this Statement object. The commands in this list can be executed as a batch by calling the method executeBatch.

### Statement.executeBatch()

Submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts. The int elements of the array that is returned are ordered to correspond to the commands in the batch, which are ordered according to the order in which they were added to the batch.

The below example demonstrates the batch insertion operations using [**Statement interface**](http://www.javaguides.net/2018/10/jdbc-statement-interface.html):

package com.javaguides.jdbc.batch;

import java.sql.BatchUpdateException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Arrays;

/\*\*

\* Update Batch operation using Statement Interface

\* @author Ramesh Fadatare

\*

\*/

public class BatchUpdateExample {

public static void main(String[] args) {

batchUpdate();

}

private static void batchUpdate() {

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "root");

// Step 2:Create a statement using connection object

Statement statement = connection.createStatement()) {

connection.setAutoCommit(false);

statement.addBatch("update users set name = 'Sam' where id = 1;");

statement.addBatch("update users set name = 'a' where id = 2;");

statement.addBatch("update users set name = 'b' where id = 3;");

statement.addBatch("update users set name = 'c' where id = 4;");

int[] updateCounts = statement.executeBatch();

System.out.println(Arrays.toString(updateCounts));

connection.commit();

} catch (BatchUpdateException batchUpdateException) {

printBatchUpdateException(batchUpdateException);

} catch (SQLException e) {

printSQLException(e);

}

}

public static void printSQLException(SQLException ex) {

for (Throwable e: ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

public static void printBatchUpdateException(BatchUpdateException b) {

System.err.println("----BatchUpdateException----");

System.err.println("SQLState: " + b.getSQLState());

System.err.println("Message: " + b.getMessage());

System.err.println("Vendor: " + b.getErrorCode());

System.err.print("Update counts: ");

int[] updateCounts = b.getUpdateCounts();

for (int i = 0; i < updateCounts.length; i++) {

System.err.print(updateCounts[i] + " ");

}

}

}

Output:

[1, 1, 1, 1, 1]

## JDBC PreparedStatement - Insert a Record Example

In this example, we will use the *users* database table. Before inserting a record to a database, we need to first create a users table in a database. Here is article [**JDBC Statement Create Table Example**](http://www.javaguides.net/2018/10/jdbc-statement-create-table-example.html) will create a users table in a mysql\_database database.

package com.javaguides.jdbc.preparestatement.examples;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

/\*\*

\* Insert PrepareStatement JDBC Example

\*

\* @author Ramesh Fadatare

\*

\*/

public class InsertPStatementExample {

private static final String INSERT\_USERS\_SQL = "INSERT INTO users" +

" (id, name, email, country, password) VALUES " +

" (?, ?, ?, ?, ?);";

public static void main(String[] argv) throws SQLException {

InsertPStatementExample createTableExample = new InsertPStatementExample();

createTableExample.insertRecord();

}

public void insertRecord() throws SQLException {

System.out.println(INSERT\_USERS\_SQL);

// Step 1: Establishing a Connection

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "root");

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(INSERT\_USERS\_SQL)) {

preparedStatement.setInt(1, 1);

preparedStatement.setString(2, "Tony");

preparedStatement.setString(3, "tony@gmail.com");

preparedStatement.setString(4, "US");

preparedStatement.setString(5, "secret");

System.out.println(preparedStatement);

// Step 3: Execute the query or update query

preparedStatement.executeUpdate();

} catch (SQLException e) {

// print SQL exception information

printSQLException(e);

}

// Step 4: try-with-resource statement will auto close the connection.

}

public static void printSQLException(SQLException ex) {

for (Throwable e: ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

}

## JDBC PreparedStatement Update a Record Example

In this example, we will use the users database table. Before updating a record to a database, we need to first create a users table in a database. Here is article JDBC Statement Create Table Example will create a users table in a mysql\_database database.

package com.javaguides.jdbc.preparestatement.examples;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

/\*\*

\* Update PreparedStatement JDBC Example

\* @author Ramesh Fadatare

\*

\*/

public class UpdatePStatementExample {

private static final String UPDATE\_USERS\_SQL = "update users set name = ? where id = ?;";

public static void main(String[] argv) throws SQLException {

UpdatePStatementExample updateStatementExample = new UpdatePStatementExample();

updateStatementExample.updateRecord();

}

public void updateRecord() throws SQLException {

System.out.println(UPDATE\_USERS\_SQL);

// Step 1: Establishing a Connection

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "root");

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(UPDATE\_USERS\_SQL)) {

preparedStatement.setString(1, "Ram");

preparedStatement.setInt(2, 1);

// Step 3: Execute the query or update query

preparedStatement.executeUpdate();

} catch (SQLException e) {

// print SQL exception information

printSQLException(e);

}

// Step 4: try-with-resource statement will auto close the connection.

}

public static void printSQLException(SQLException ex) {

for (Throwable e: ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

}

## JDBC PreparedStatement Select Records Example

Here we have a users table in a database and we will query a list of users from database table by id.

package com.javaguides.jdbc.preparestatement.examples;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

/\*\*

\* Select PreparedStatement JDBC Example

\*

\* @author Ramesh Fadatare

\*

\*/

public class SelectPStatementExample {

private static final String QUERY = "select id,name,email,country,password from Users where id =?";

public static void main(String[] args) {

// using try-with-resources to avoid closing resources (boiler plate code)

// Step 1: Establishing a Connection

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "root");

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(QUERY);) {

preparedStatement.setInt(1, 1);

System.out.println(preparedStatement);

// Step 3: Execute the query or update query

ResultSet rs = preparedStatement.executeQuery();

// Step 4: Process the ResultSet object.

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

String email = rs.getString("email");

String country = rs.getString("country");

String password = rs.getString("password");

System.out.println(id + "," + name + "," + email + "," + country + "," + password);

}

} catch (SQLException e) {

printSQLException(e);

}

// Step 4: try-with-resource statement will auto close the connection.

}

public static void printSQLException(SQLException ex) {

for (Throwable e: ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

}

Output:

1,Ram,tony@gmail.com,US,secret

we will discuss how to add a list of values dynamically to IN clause using JDBC **[PreparedStatement](http://www.javaguides.net/2018/10/jdbc-preparedstatement-interface.html" \t "_blank)**.

Consider we have a users table in a database and we have few records in database users table. Let's create a list of user *ids* and dynamically create a SQL statement using *StringBuilder*.

Below code creates a dynamic SQL query with a number of placeholders ? in IN clause:

private static String createQuery(int length) {

String query = "select id, name from users where id in (";

StringBuilder queryBuilder = new StringBuilder(query);

for (int i = 0; i < length; i++) {

queryBuilder.append(" ?");

if (i != length - 1)

queryBuilder.append(",");

}

queryBuilder.append(")");

return queryBuilder.toString();

}

## JDBC PreparedStatement with a list of parameters in an IN clause Example

package com.javaguides.jdbc.preparestatement.examples;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

/\*\*

\* JDBC PreparedStatement with list of parameters in a IN clause Example

\* @author Ramesh Fadatare

\*

\*/

public class JDBCPreparedStatementDynamic {

public static void main(String[] args) {

List < Integer > ids = new ArrayList < > ();

ids.add(1);

ids.add(2);

ids.add(3);

ids.add(4);

processDynamicQuery(ids);

}

public static void processDynamicQuery(List < Integer > ids) {

String query = createQuery(ids.size());

System.out.println("Query=" + query);

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "root");

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(query)) {

ResultSet rs = null;

int parameterIndex = 1;

for (Iterator < Integer > iterator = ids.iterator(); iterator.hasNext();) {

Integer id = (Integer) iterator.next();

preparedStatement.setInt(parameterIndex++, id);

}

System.out.println(preparedStatement);

rs = preparedStatement.executeQuery();

while (rs.next()) {

System.out.println("User ID=" + rs.getInt("id") + ", Name=" + rs.getString("name"));

}

// close the resultset here

try {

rs.close();

} catch (SQLException e) {}

} catch (SQLException e) {

e.printStackTrace();

}

}

private static String createQuery(int length) {

String query = "select id, name from users where id in (";

StringBuilder queryBuilder = new StringBuilder(query);

for (int i = 0; i < length; i++) {

queryBuilder.append(" ?");

if (i != length - 1)

queryBuilder.append(",");

}

queryBuilder.append(")");

return queryBuilder.toString();

}

}

Output:

Query=select id, name from users where id in ( ?, ?, ?, ?)

com.mysql.jdbc.JDBC42PreparedStatement@579bb367: select id, name from users where id in ( 1, 2, 3, 4)

User ID=1, Name=A

User ID=2, Name=Pramod

User ID=3, Name=A

User ID=4, Name=B

## JDBC PreparedStatement Batch Insert Example

In the previous article, we have seen [**JDBC Statement - Batch Insert Example**](http://www.javaguides.net/2018/10/jdbc-statement-batch-insert-example.html). It is also possible to have a parameterized batch insert or update, as shown in the following code fragment, where the *con* is a [**Connection**](http://www.javaguides.net/2018/10/jdbc-connection-interface.html) object:

package com.javaguides.jdbc.batch;

import java.sql.BatchUpdateException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.Arrays;

/\*\*

\* Insert Batch operation using PreparedStatement Interface

\* @author Ramesh Fadatare

\*

\*/

public class BatchInsertExample {

public static void main(String[] args) {

parameterizedBatchUpdate();

}

public static void printSQLException(SQLException ex) {

for (Throwable e: ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

private static void parameterizedBatchUpdate() {

String INSERT\_USERS\_SQL = "INSERT INTO users" + " (id, name, email, country, password) VALUES " +

" (?, ?, ?, ?, ?);";

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "root");

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(INSERT\_USERS\_SQL)) {

connection.setAutoCommit(false);

preparedStatement.setInt(1, 20);

preparedStatement.setString(2, "a");

preparedStatement.setString(3, "a@gmail.com");

preparedStatement.setString(4, "India");

preparedStatement.setString(5, "secret");

preparedStatement.addBatch();

preparedStatement.setInt(1, 21);

preparedStatement.setString(2, "b");

preparedStatement.setString(3, "b@gmail.com");

preparedStatement.setString(4, "India");

preparedStatement.setString(5, "secret");

preparedStatement.addBatch();

preparedStatement.setInt(1, 22);

preparedStatement.setString(2, "c");

preparedStatement.setString(3, "c@gmail.com");

preparedStatement.setString(4, "India");

preparedStatement.setString(5, "secret");

preparedStatement.addBatch();

preparedStatement.setInt(1, 23);

preparedStatement.setString(2, "d");

preparedStatement.setString(3, "d@gmail.com");

preparedStatement.setString(4, "India");

preparedStatement.setString(5, "secret");

preparedStatement.addBatch();

int[] updateCounts = preparedStatement.executeBatch();

System.out.println(Arrays.toString(updateCounts));

connection.commit();

connection.setAutoCommit(true);

} catch (BatchUpdateException batchUpdateException) {

printBatchUpdateException(batchUpdateException);

} catch (SQLException e) {

printSQLException(e);

}

}

public static void printBatchUpdateException(BatchUpdateException b) {

System.err.println("----BatchUpdateException----");

System.err.println("SQLState: " + b.getSQLState());

System.err.println("Message: " + b.getMessage());

System.err.println("Vendor: " + b.getErrorCode());

System.err.print("Update counts: ");

int[] updateCounts = b.getUpdateCounts();

for (int i = 0; i < updateCounts.length; i++) {

System.err.print(updateCounts[i] + " ");

}

}

}

Output:

[1, 1, 1, 1]

## JDBC Batch Update Example using PreparedStatement

Let's develop an example to have parameterized batch update, as shown in the following code fragment:

package com.javaguides.jdbc.batch;

import java.sql.BatchUpdateException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.Arrays;

/\*\*

\* Update Batch operation using PreparedStatement Interface

\* @author Ramesh Fadatare

\*

\*/

public class BatchUpdateExample {

public static void main(String[] args) {

parameterizedBatchUpdate();

}

public static void printSQLException(SQLException ex) {

for (Throwable e: ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

private static void parameterizedBatchUpdate() {

String UPDATE\_USERS\_SQL = "update users set name = ? where id = ?;";

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "root");

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(UPDATE\_USERS\_SQL)) {

connection.setAutoCommit(false);

preparedStatement.setString(1, "A");

preparedStatement.setInt(2, 1);

preparedStatement.addBatch();

preparedStatement.setString(1, "B");

preparedStatement.setInt(2, 2);

preparedStatement.addBatch();

preparedStatement.setString(1, "C");

preparedStatement.setInt(2, 3);

preparedStatement.addBatch();

preparedStatement.setString(1, "D");

preparedStatement.setInt(2, 4);

preparedStatement.addBatch();

int[] updateCounts = preparedStatement.executeBatch();

System.out.println(Arrays.toString(updateCounts));

connection.commit();

connection.setAutoCommit(true);

} catch (BatchUpdateException batchUpdateException) {

printBatchUpdateException(batchUpdateException);

} catch (SQLException e) {

printSQLException(e);

}

}

public static void printBatchUpdateException(BatchUpdateException b) {

System.err.println("----BatchUpdateException----");

System.err.println("SQLState: " + b.getSQLState());

System.err.println("Message: " + b.getMessage());

System.err.println("Vendor: " + b.getErrorCode());

System.err.print("Update counts: ");

int[] updateCounts = b.getUpdateCounts();

for (int i = 0; i < updateCounts.length; i++) {

System.err.print(updateCounts[i] + " ");

}

}

}

Output:

[1, 1, 1, 1]

## ResultSetMetaData Interface Commonly used methods

1. *String getCatalogName(int column)* - Gets the designated column's table's catalog name.
2. *String getColumnClassName(int column)* - Returns the fully-qualified name of the Java class whose instances are manufactured if the method ResultSet.getObject is called to retrieve a value from the column.
3. *int getColumnCount()* - Returns the number of columns in this ResultSet object.
4. *String getColumnName(int column)* - Get the designated column's name.
5. *int getColumnType(int column)* - Retrieves the designated column's SQL type.
6. *String getColumnTypeName(int column)* - Retrieves the designated column's database-specific type name.
7. *int getPrecision(int column)* - Get the designated column's specified column size.
8. *int getScale(int column)* - Gets the designated column's number of digits to right of the decimal point.
9. *String getSchemaName(int column)* - Get the designated column's table's schema.
10. *String getTableName(int column)* - Gets the designated column's table name.

## ResultSetMetaData Interface Examples

This example demonstrates below commonly used APIs of *ResultSetMetaData* interface:

package com.javaguides.jdbc.databasemetadata;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.ResultSetMetaData;

import java.sql.SQLException;

import java.sql.Statement;

/\*\*

\*

\* @author Ramesh Fadatare

\*

\*/

public class ResuleSetMetaDataExample {

private static final String QUERY = "select id,name,email,country,password from Users";

public static void main(String[] args) {

// using try-with-resources to avoid closing resources (boiler plate code)

// Step 1: Establishing a Connection

try (Connection connection = DriverManager

.getConnection("jdbc:mysql://localhost:3306/mysql\_database?useSSL=false", "root", "root");

// Step 2:Create a statement using connection object

Statement stmt = connection.createStatement();

// Step 3: Execute the query or update query

ResultSet rs = stmt.executeQuery(QUERY)) {

ResultSetMetaData resultSetMetaData = rs.getMetaData();

System.out.println("1. Column count in users table :: " + resultSetMetaData.getColumnCount());

System.out.println("2. First column name in users table :: " + resultSetMetaData.getColumnName(1));

System.out.println("3. Database name of users table' column id :: " + resultSetMetaData.getCatalogName(1));;

System.out.println("4. Data type of column id :: " + resultSetMetaData.getColumnTypeName(1));

System.out.println("5. Get table name of column id :: " + resultSetMetaData.getTableName(1));

} catch (SQLException e) {

printSQLException(e);

}

// Step 4: try-with-resource statement will auto close the connection.

}

public static void printSQLException(SQLException ex) {

for (Throwable e: ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

}

Output:

Column count in users table :: 5

First column name in users table :: id

Database name of users table' column id :: mysql\_database

Data type of column id :: INT

Get table name of column id :: users