Types of Statements in JDBC

Difficulty Level : Medium Last Updated : 09 Jul, 2021

The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database. There are different types of statements that are used in JDBC as follows:

- Create Statement
- Prepared Statement
- Callable Statement

1. **Create a Statement:** From the connection interface, you can create the object for this interface. It is generally used for general–purpose access to databases and is useful while using static SQL statements at runtime.

Syntax:

```
Statement statement = connection.createStatement();
```

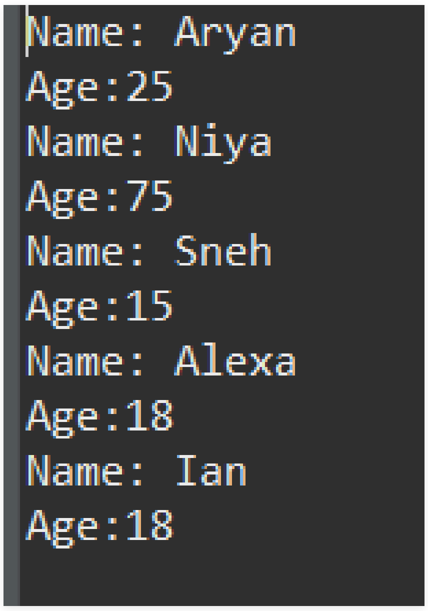
Implementation: Once the Statement object is created, there are three ways to execute it.

- ***boolean execute(String SQL):*** If the ResultSet object is retrieved, then it returns true else false is returned. Is used to execute SQL DDL statements or for dynamic SQL.
- **int executeUpdate(String SQL):** Returns number of rows that are affected by the execution of the statement, used when you need a number for INSERT, DELETE or UPDATE statements.
- ***ResultSet executeQuery(String SQL):*** Returns a ResultSet object. Used similarly as SELECT is used in SQL.

Example:

```
// Java Program illustrating Create Statement in JDBC

// Importing Database(SQL) classes
import java.sql.*;

// Class
class GFG {
    // Main driver method
    public static void main(String[] args)
    {

        // Try block to check if any exceptions occur
        try {

            // Step 2: Loading and registering drivers

            // Loading driver using forName() method
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Registering driver using DriverManager
            Connection con = DriverManager.getConnection(
                "jdbc:mysql:///world", "root", "12345");

            // Step 3: Create a statement
            Statement statement = con.createStatement();
            String sql = "select * from people";

            // Step 4: Execute the query
            ResultSet result = statement.executeQuery(sql);

            // Step 5: Process the results

            // Condition check using hasNext() method which
            // holds true till there is single element
            // remaining in List
            while (result.next()) {

                // Print name an age
                System.out.println(
                    "Name: " + result.getString("name"));
                System.out.println(
                    "Age:" + result.getString("age"));
            }
        }

        // Catching database exceptions if any
        catch (SQLException e) {

            // Print the exception
            System.out.println(e);
        }

        // Catching generic ClassNotFoundException if any
        catch (ClassNotFoundException e) {

            // Print and display the line number
            // where exception occurred
            e.printStackTrace();
        }
    }
}
```

Output: Name and age are as shown for random inputs



```
Name: Aryan
Age:25
Name: Niya
Age:75
Name: Sneh
Age:15
Name: Alexa
Age:18
Name: Ian
Age:18
```

2. Prepared Statement represents a recompiled SQL statement, that can be executed many times. This accepts parameterized SQL queries. In this, “?” is used instead of the parameter, one can pass the parameter dynamically by using the methods of PREPARED STATEMENT at run time.

Illustration:

Considering in the people database if there is a need to INSERT some values, SQL statements such as these are used:

```
INSERT INTO people VALUES ("Ayan",25);
INSERT INTO people VALUES("Kriya",32);
```

To do the same in Java, one may use Prepared Statements and set the values in the ? holders, setXXX() of a prepared statement is used as shown:

```
String query = "INSERT INTO people(name, age)VALUES(?, ?)";
Statement pstmt = con.prepareStatement(query);
pstmt.setString(1,"Ayan");
ptstmt.setInt(2,25);
// where pstmt is an object name
```

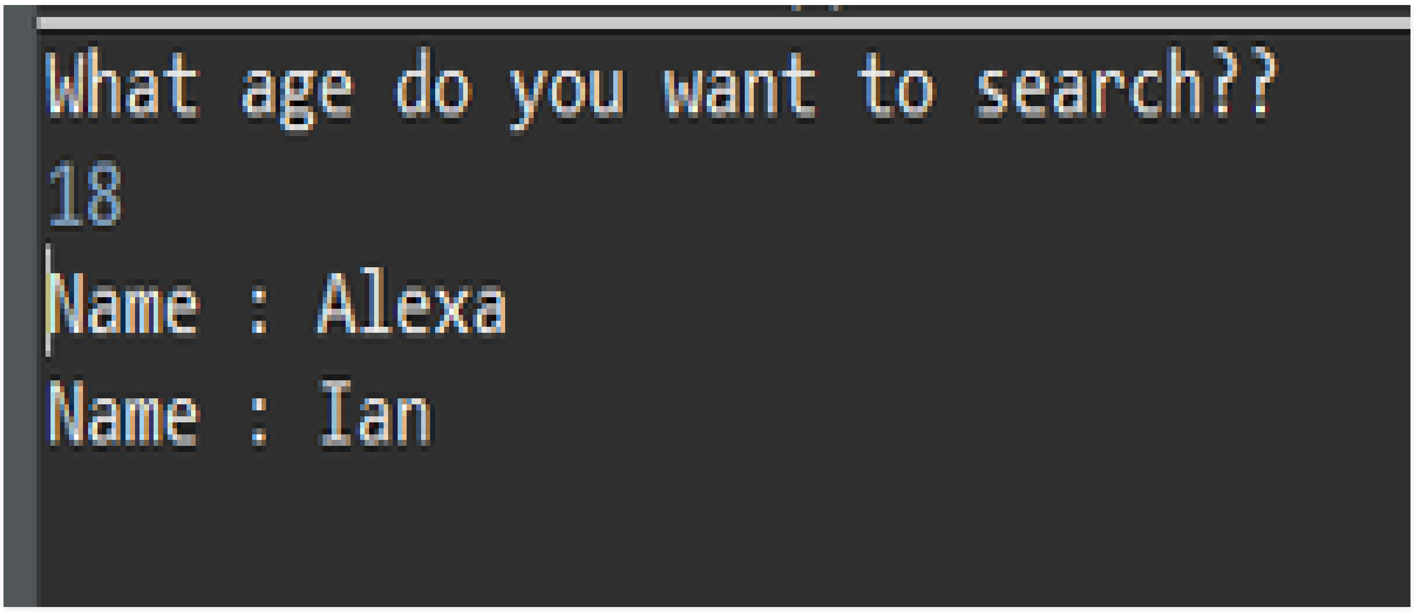
Implementation: Once the PreparedStatement object is created, there are three ways to execute it:

- ***execute():*** This returns a boolean value and executes a static SQL statement that is present in the prepared statement object.
- ***executeQuery():*** Returns a ResultSet from the current prepared statement.
- ***executeUpdate():*** Returns the number of rows affected by the DML statements such as INSERT, DELETE, and more that is present in the current Prepared Statement.

Example:

```
// Java Program illustrating Prepared Statement in JDBC

// Step 1: Importing DB(SQL here) classes
import java.sql.*;
// Importing Scanner class to
// take input from the user
import java.util.Scanner;

// Main clas
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // try block to check for exceptions
        try {

            // Step 2: Establish a connection

            // Step 3: Load and register drivers

            // Loading drivers using forName() method
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Scanner class to take input from user
            Scanner sc = new Scanner(System.in);

            // Display message for ease for user
            System.out.println(
                "What age do you want to search?? ");

            // Reading age an primitive datatype from user
            // using nextInt() method
            int age = sc.nextInt();

            // Registering drivers using DriverManager
            Connection con = DriverManager.getConnection(
                "jdbc:mysql:///world", "root", "12345");
            // Step 4: Create a statement
            PreparedStatement ps = con.prepareStatement(
                "select name from world.people where age = ?");

            // Step 5: Execute the query
            ps.setInt(1, age);
            ResultSet result = ps.executeQuery();

            // Step 6: Process the results

            // Condition check using next() method
            // to check for element
            while (result.next()) {

                // Print and display elements(Names)
                System.out.println("Name : "
                                   + result.getString(1));
            }

            // Step 7: Closing the connections
            // (Optional but it is recommended to do so)
        }

        // Catch block to handle database exceptions
        catch (SQLException e) {

            // Display the DB exception if any
            System.out.println(e);
        }

        // Catch block to handle class exceptions
        catch (ClassNotFoundException e) {

            // Print the line number where exception occurred
            // using printStackTrace() method if any
            e.printStackTrace();
        }
    }
}
```

Output:



```
What age do you want to search??
18
Name : Alexa
Name : Ian
```

3. Callable Statement are stored procedures which are a group of statements that we compile in the database for some task, they are beneficial when we are dealing with multiple tables with complex scenario & rather than sending multiple queries to the database, we can send the required data to the stored procedure & lower the logic executed in the database server itself. The Callable Statement interface provided by JDBC API helps in executing stored procedures.

Syntax: To prepare a CallableStatement

```
CallableStatement cstmt = con.prepareCall("{call Procedure_name(?, ?}");
```

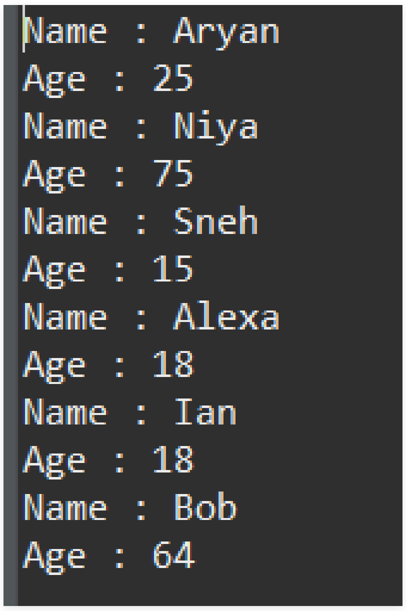
Implementation: Once the callable statement object is created

- ***execute()*** is used to perform the execution of the statement.

Example:

```
// Java Program illustrating Callable Statement in JDBC

// Step 1: Importing DB(SQL) classes
import java.sql.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Try block to check if any exceptions occurs
        try {

            // Step 2: Establish a connection

            // Step 3: Loading and registering drivers

            // Loading driver using forName() method
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Registering driver using DriverManager
            Connection con = DriverManager.getConnection(
                "jdbc:mysql:///world", "root", "12345");

            // Step 4: Create a statement
            Statement s = con.createStatement();

            // Step 5: Execute the query
            // select * from people

            CallableStatement cs
                = con.prepareCall("{call peopleinfo(?,?)}");
            cs.setString(1, "Bob");
            cs.setInt(2, 64);
            cs.execute();
            ResultSet result
                = s.executeQuery("select * from people");

            // Step 6: Process the results

            // Condition check using next() method
            // to check for element
            while (result.next()) {

                // Print and display elements (Name and Age)
                System.out.println("Name : "
                                   + result.getString(1));
                System.out.println("Age : "
                                   + result.getInt(2));
            }
        }

        // Catch statement for DB exceptions
        catch (SQLException e) {

            // Print the exception
            System.out.println(e);
        }

        // Catch block for generic class exceptions
        catch (ClassNotFoundException e) {

            // Print the line number where exception occurred
            e.printStackTrace();
        }
    }
}
```

Output:



```
Name : Aryan
Age : 25
Name : Niya
Age : 75
Name : Sneh
Age : 15
Name : Alexa
Age : 18
Name : Ian
Age : 18
Name : Bob
Age : 64
```