

# Processing SQL Statements with JDBC

In general, to process any SQL statement with JDBC, you follow these steps:

1. [Establishing a connection.](#)
2. [Create a statement.](#)
3. [Execute the query.](#)
4. [Process the `ResultSet` object.](#)
5. [Close the connection.](#)

This page uses the following method, `CoffeesTables.viewTable`, from the tutorial sample to demonstrate these steps. This method outputs the contents of the table `COFFEES`. This method will be discussed in more detail later in this tutorial:

```
public static void viewTable(Connection con, String dbName)
    throws SQLException {

    Statement stmt = null;
    String query = "select COF_NAME, SUP_ID, PRICE, " +
        "SALES, TOTAL " +
        "from " + dbName + ".COFFEES";

    try {
        stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            String coffeeName = rs.getString("COF_NAME");
            int supplierID = rs.getInt("SUP_ID");
            float price = rs.getFloat("PRICE");
            int sales = rs.getInt("SALES");
            int total = rs.getInt("TOTAL");
            System.out.println(coffeeName + "\t" + supplierID +
                "\t" + price + "\t" + sales +
                "\t" + total);
        }
    } catch (SQLException e) {
        JBDBCTutorialUtilities.printSQLException(e);
    } finally {
        if (stmt != null) { stmt.close(); }
    }
}
```

## Establishing Connections

First, establish a connection with the data source you want to use. A data source can be a DBMS, a legacy file system, or some other source of data with a corresponding JDBC driver. This connection is represented by a `Connection` object. See [Establishing a Connection](#) for more information.

## Creating Statements

A `Statement` is an interface that represents a SQL statement. You execute `Statement` objects, and they generate `ResultSet` objects, which is a table of data representing a database result set. You need a `Connection` object to create a `Statement` object.

For example, `CoffeesTables.viewTable` creates a `Statement` object with the following code:

```
stmt = con.createStatement();
```

There are three different kinds of statements:

- `Statement`: Used to implement simple SQL statements with no parameters.
- `PreparedStatement`: (Extends `Statement`.) Used for precompiling SQL statements that might contain input parameters. See [Using Prepared Statements](#) for more information.
- `CallableStatement`: (Extends `PreparedStatement`.) Used to execute stored procedures that may contain both input and output parameters. See [Stored Procedures](#) for more information.

## Executing Queries

To execute a query, call an `execute` method from `Statement` such as the following:

- `execute`: Returns `true` if the first object that the query returns is a `ResultSet` object. Use this method if the query could return one or more `ResultSet` objects. Retrieve the `ResultSet` objects returned from the query by repeatedly calling `Statement.getResultSet`.
- `executeQuery`: Returns one `ResultSet` object.
- `executeUpdate`: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using `INSERT`, `DELETE`, or `UPDATE` SQL statements.

For example, `CoffeesTables.viewTable` executed a `Statement` object with the following code:

```
ResultSet rs = stmt.executeQuery(query);
```

See [Retrieving and Modifying Values from Result Sets](#) for more information.

## Processing ResultSet Objects

You access the data in a `ResultSet` object through a cursor. Note that this cursor is not a database cursor. This cursor is a pointer that points to one row of data in the `ResultSet` object. Initially, the cursor is positioned before the first row. You call various methods defined in the `ResultSet` object to move the cursor.

For example, `CoffeesTables.viewTable` repeatedly calls the method `ResultSet.next` to move the cursor forward by one row. Every time it calls `next`, the method outputs the data in the row where the cursor is currently positioned:

```
try {
    stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    while (rs.next()) {
        String coffeeName = rs.getString("COF_NAME");
        int supplierID = rs.getInt("SUP_ID");
        float price = rs.getFloat("PRICE");
        int sales = rs.getInt("SALES");
        int total = rs.getInt("TOTAL");
        System.out.println(coffeeName + "\t" + supplierID +
                           "\t" + price + "\t" + sales +
                           "\t" + total);
    }
}
// ...
```

See [Retrieving and Modifying Values from Result Sets](#) for more information.

## Closing Connections

When you are finished using a `Statement`, call the method `Statement.close` to immediately release the resources it is using. When you call this method, its `ResultSet` objects are closed.

For example, the method `CoffeesTables.viewTable` ensures that the `Statement` object is closed at the end of the method, regardless of any `SQLException` objects thrown, by wrapping it in a `finally` block:

```
} finally {
    if (stmt != null) { stmt.close(); }
}
```

JDBC throws an `SQLException` when it encounters an error during an interaction with a data source. See [Handling SQL Exceptions](#) for more information.

In JDBC 4.1, which is available in Java SE release 7 and later, you can use a try-with-resources statement to automatically close `Connection`, `Statement`, and `ResultSet` objects, regardless of whether an `SQLException` has been thrown. An automatic resource statement consists of a `try` statement and one or more declared resources. For example, you can modify `CoffeesTables.viewTable` so that its `Statement` object closes automatically, as follows:

```
public static void viewTable(Connection con) throws SQLException {

    String query = "select COF_NAME, SUP_ID, PRICE, " +
                  "SALES, TOTAL " +
                  "from COFFEES";

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

        ResultSet rs = stmt.executeQuery(query);
```

```

        while (rs.next()) {
            String coffeeName = rs.getString("COF_NAME");
            int supplierID = rs.getInt("SUP_ID");
            float price = rs.getFloat("PRICE");
            int sales = rs.getInt("SALES");
            int total = rs.getInt("TOTAL");
            System.out.println(coffeeName + ", " + supplierID +
                               ", " + price + ", " + sales +
                               ", " + total);
        }
    } catch (SQLException e) {
        JDBCTutorialUtilities.printSQLException(e);
    }
}

```

The following statement is an `try-with-resources` statement, which declares one resource, `stmt`, that will be automatically closed when the `try` block terminates:

```

try (Statement stmt = con.createStatement()) {
    // ...
}

```

See [The `try-with-resources` Statement](#) in the [Essential Classes](#) trail for more information.