Java JDBC Tutorial

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

* JDBC-ODBC Bridge Driver,
* Native Driver,
* Network Protocol Driver, and
* Thin Driver

We have discussed the above four drivers in the next chapter.

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

x

* Driver interface
* Connection interface
* Statement interface
* PreparedStatement interface
* CallableStatement interface
* ResultSet interface
* ResultSetMetaData interface
* DatabaseMetaData interface
* RowSet interface

A list of popular *classes* of JDBC API are given below:

* DriverManager class
* Blob class
* Clob class
* Types class

Why Should We Use JDBC

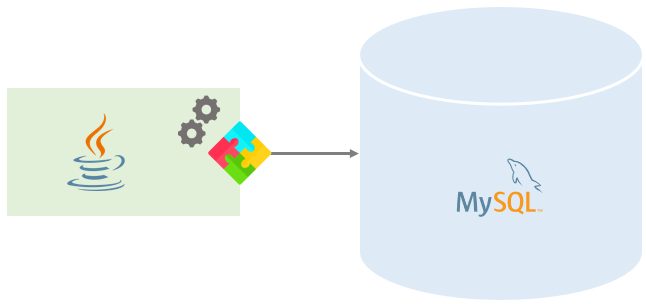
Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

**MYSQL JDBC :**

Java JDBC API provides a standard interface to interact with any relational databases. In this MySQL JDBC tutorial section, we will show you how to use JDBC to interact with MySQL databases. You will learn how to [use JDBC API to connect to MySQL using MySQL Connector/J driver](https://www.mysqltutorial.org/connecting-to-mysql-using-jdbc-driver/), [execute SQL statements in Java program](https://www.mysqltutorial.org/querying-data-from-mysql-using-jdbc/) and process the results, how to[call stored procedures](https://www.mysqltutorial.org/calling-mysql-stored-procedures-from-jdbc/) and [work with BLOB](https://www.mysqltutorial.org/mysql-jdbc-blob) using JDBC API.



**Summary**: in this tutorial, we will give you a very brief overview of JDBC so that you can use it for interacting with MySQL databases.

JDBC API provides a standard interface for interacting with any relational database management systems (RDBMS). JDBC API consists of the following main components:

1. JDBC Driver
2. Connection
3. Statement
4. ResultSet

Let’s take a look at each component in more detail.

## **JDBC Driver**

A JDBC driver is set of Java classes that implement JDBC interfaces for interacting with a specific database. Almost all database vendors such as MySQL, Oracle, Microsoft SQL Server, provide JDBC drivers. For example, MySQL provides a JDBC driver called MySQL Connection/J that allows you to work with MySQL database through a standard JDBC API.

There are three types of JDBC drivers including JDBC-native API Driver, JDBC-net Driver, and JDBC Driver.

We will discuss about the JDBC driver, for more detailed information on the other driver type, you can check it out the [JDBC driver](http://en.wikipedia.org/wiki/JDBC_driver).

JDBC Driver is written in pure Java. It translates JDBC calls into MySQL specific calls and sends the calls directly to a specific database. To use a JDBC driver, you need to include the driver JAR file with your application. MySQL Connector/J is the JDBC driver.

## **Connection**

The first and most important component of JDBC is the Connection object. In a Java application, you first load a JDBC driver and then [establish a connection to the database](https://www.mysqltutorial.org/connecting-to-mysql-using-jdbc-driver/). Through the Connection object, you can interact with the database e.g., creating a Statement to execute SQL queries against tables. You can open more than one connection to a database at a time.

## **Statement**

To execute a SQL query e.g., [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx), [INSERT](https://www.mysqltutorial.org/mysql-insert-statement.aspx), [UPDATE](https://www.mysqltutorial.org/mysql-update-data.aspx), [DELETE](https://www.mysqltutorial.org/mysql-delete-statement.aspx), etc., you use a Statement object. You create the Statement object through the Connection object. JDBC provides several types of statements for different purposes such as PreparedStatement , CallableStatement . We will cover the details of each object in the next tutorials.

## **ResultSet**

After [querying data from the database](https://www.mysqltutorial.org/querying-data-from-mysql-using-jdbc/), you get a ResultSet object. The ResultSet object provides a set of API that allows you to traverse result of the query.

The typical flow of using JDBC is as follows:

1. First, load the JDBC driver and create a connection to the database.
2. Then, create a Statement and [execute the query to get a ResultSet](https://www.mysqltutorial.org/querying-data-from-mysql-using-jdbc/).
3. Next, traverse and process the ResultSet .
4. Close the ResultSet , Statement , and Connection .



In this tutorial, we have introduced you to some basic information on JDBC and its main components: JDBC Driver, Connection, Statement, and ResultSet.

# Connecting to MySQL Using JDBC Driver

In this tutorial, you will learn how to connect to MySQL database using JDBC Connection object.

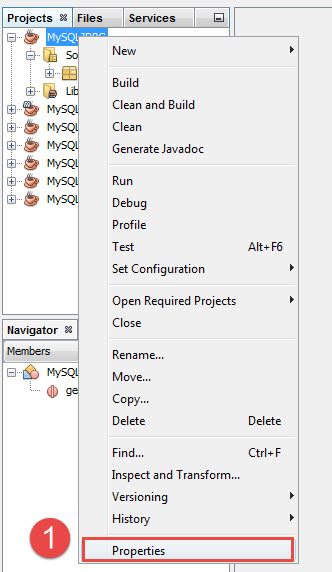
To connect to MySQL database from a Java program, you need to do the following steps:

1. Load the MySQL Connector/J into your program.
2. Create a new Connection object from the DriverManager class. Then you can use this Connection object to execute queries.

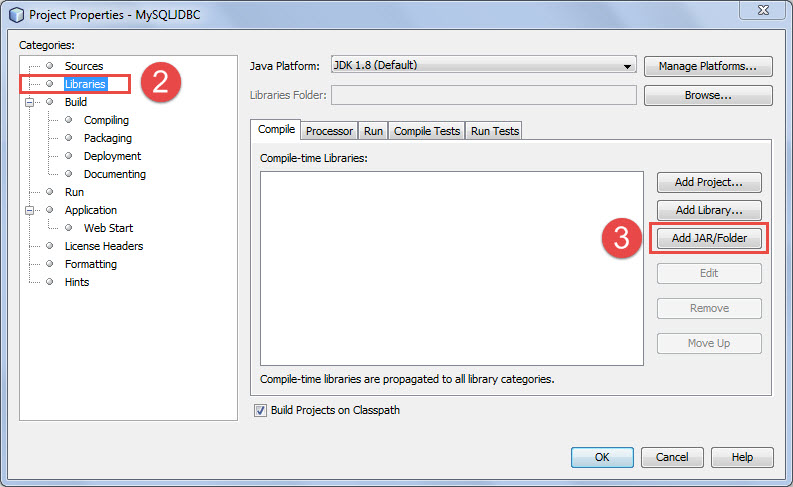
## **Loading MySQL Connector/J into your program**

To load MySQL Connector/J into your program you follow three steps below:

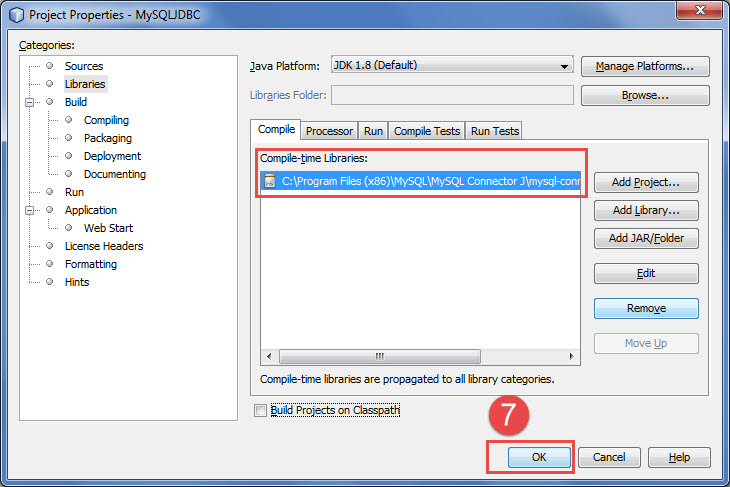
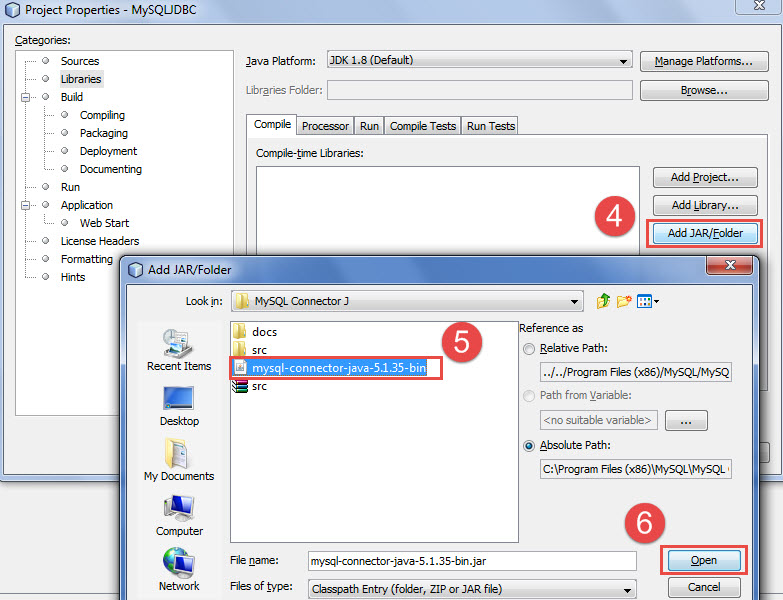
First, in NetBeans IDE, from project name, right mouse click and choose properties menu item. The project properties dialog will appear.



Second, on the left hand side of the project properties dialog, from the Categories section, choose Libraries item.



Third, click on the Add JAR folder button, browse to the location where you installed MySQL Connector/J, and choose the JAR file as screenshot below; after that click OK button.



## **Connecting to MySQL database**

First, you need to import three classes: SQLException, DriverManager, and Connection from the java.sql.\* package.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

Code language: Java (java)

Second, you call the getConnection() method of the DriverManager class to get the Connection object. There are three parameters you need to pass to the getConnection() method:

1. url: the database URL in the form jdbc:subprotocol:subname. For MySQL, you use the jdbc:mysql://localhost:3306/mysqljdbc i.e., you are connecting to the MySQL with server name localhost, port 3006, and database mysqljdbc.
2. user: the database user that will be used to connect to MySQL.
3. password: the password of the database user.

Connection conn = null;

try {

// db parameters

String url = "jdbc:mysql://localhost:3306/mysqljdbc";

String user = "root";

String password = "secret";

// create a connection to the database

conn = DriverManager.getConnection(url, user, password);

// more processing here

// ...

} catch(SQLException e) {

System.out.println(e.getMessage());

} finally {

try{

if(conn ! null)

conn.close()

}catch(SQLException ex){

System.out.println(ex.getMessage())

}

}

Code language: Java (java)

When connecting to MySQL, anything could happens e.g., database server is not available, wrong user name or password, etc. in such cases, JDBC throws a SQLException . Therefore, when you create a Connection object, you should always put it inside a try catch block. Also you should always close the database connection once you complete interacting with database by calling close() method of the Connection object.

From Java 7, there is another nice statement called [try-with-resources](https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html) that allows you to simplify the code above as follows:

// db parameters

String url = "jdbc:mysql://localhost:3306/mysqljdbc";

String user = "root";

String password = "secret";

Connection conn = null;

try(conn = DriverManager.getConnection(url, user, password);) {

// processing here

} catch(SQLException e) {

System.out.println(e.getMessage());

}

Code language: Java (java)

It is automatically calls the close() method of the Connection object once program finishes. As you can see it’s cleaner and more elegant. However…

It is not secure as well as flexible when you hard coded the database parameters inside the code like above. In case you change the database server or password; you have to change the code, compile it again, which is not a good design.

To avoid hard coding all the database parameters in the code, you can use a Java properties file to store them. In case of changes, you just need to change them in the properties file and you don’t have to recompile the code.

Let’s take a look at the properties file named db.properties:

# MySQL DB parameters

user=root

password=secret

url=jdbc:mysql://localhost:3306/mysqljdbc

Code language: PHP (php)

You can rewrite the code for creating a Connection object with parameters from a properties file as follows:

Connection conn = null;

try(FileInputStream f = new FileInputStream("db.properties")) {

// load the properties file

Properties pros = new Properties();

pros.load(f);

// assign db parameters

String url = pros.getProperty("url");

String user = pros.getProperty("user");

String password = pros.getProperty("password");

// create a connection to the database

conn = DriverManager.getConnection(url, user, password);

} catch(IOException e) {

System.out.println(e.getMessage());

} finally {

try{

if(conn != null)

conn.close();

}catch(SQLException ex){

System.out.println(ex.getMessage());

}

}

Code language: Java (java)

For each interaction with MySQL database, you need to create a new connection. You would have the same piece of code for doing this in all places. Rather than doing this, you can create a new class for handing connection creation:

package org.mysqltutorial;

import java.io.FileInputStream;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.Properties;

/\*\*

\*

\* @author mysqltutorial.org

\*/

public class MySQLJDBCUtil {

/\*\*

\* Get database connection

\*

\* @return a Connection object

\* @throws SQLException

\*/

public static Connection getConnection() throws SQLException {

Connection conn = null;

try (FileInputStream f = new FileInputStream("db.properties")) {

// load the properties file

Properties pros = new Properties();

pros.load(f);

// assign db parameters

String url = pros.getProperty("url");

String user = pros.getProperty("user");

String password = pros.getProperty("password");

// create a connection to the database

conn = DriverManager.getConnection(url, user, password);

} catch (IOException e) {

System.out.println(e.getMessage());

}

return conn;

}

}

Code language: Java (java)

From next tutorial, we will use this MySQLJDBCUtil class for creating a new connection to MySQL as follows:

package org.mysqltutorial;

import java.sql.Connection;

import java.sql.SQLException;

/\*\*

\*

\* @author mysqltutorial.org

\*/

public class Main {

public static void main(String[] args) {

// create a new connection from MySQLJDBCUtil

try (Connection conn = MySQLJDBCUtil.getConnection()) {

// print out a message

System.out.println(String.format("Connected to database %s "

+ "successfully.", conn.getCatalog()));

} catch (SQLException ex) {

System.out.println(ex.getMessage());

}

}

}

Code language: Java (java)

In this tutorial, we have shown you step by step how to connect to MySQL using JDBC Connection object and use properties file to store database parameters. At the end of the tutorial, we developed a utility class that you can reuse it every time you create a connection to the database.