**Java Database Connectivity**

Table of Contents

[JDBC 1](#_Toc68566597)

[What is JDBC? 1](#_Toc68566598)

[What is a Driver? 1](#_Toc68566599)

[Types of JDBC Drivers 1](#_Toc68566600)

[Type-1 Driver 2](#_Toc68566601)

[Type-2 Driver 2](#_Toc68566602)

[Type-3 Driver 3](#_Toc68566603)

[Type-4 Driver 4](#_Toc68566604)

[Questions and Answers 4](#_Toc68566605)

[Question 2: What factors decide the choice of drivers? 5](#_Toc68566606)

[Answer: There are 2 factors that decide the choice of drivers: portability and performance. 5](#_Toc68566607)

[Question 3: Which driver is least used? 5](#_Toc68566608)

[JDBC Architecture 5](#_Toc68566609)

[Operations With DB 7](#_Toc68566610)

[JDBC 4.3 API 7](#_Toc68566611)

[java.sql package 7](#_Toc68566612)

# What is JDBC?

* JDBC stands for **J**ava **D**ata**B**ase **C**onnectivity.
* JDBC is a **Java API** used to connect and execute queries with the database.
* JDBC API consists of a set of **classes**, **interfaces** and **methods** to work with databases
* JDBC can be used to interact with every type of RDBMS such as MySQL, Oracle, Apache Derby, MongoDB, PostgreSQL, Microsoft SQL Server etc.
* It is a **part of JavaSE** (Java Platform, Standard Edition).
* JDBC API uses **JDBC drivers** to connect with the database.
* The current version of JDBC is **4.3**.

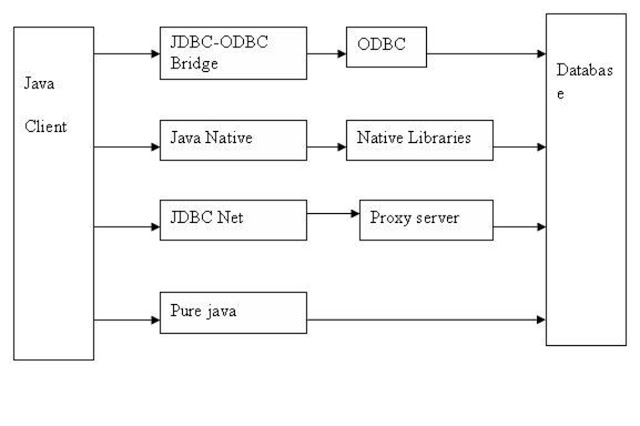
## What is a Driver?

A driver is nothing but a piece of software required to connect to a database from Java program. JDBC drivers are client side adapters (**installed on the client machine**, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.

Types of JDBC Drivers

There are four types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver



Pictorial representation of JDBC Drivers

### Type-1 Driver

* This is the oldest JDBC driver, mostly used to connect database like MS Access from Microsoft Windows operating system.
* Type 1 JDBC driver actually translates JDBC calls into ODBC (Object Database connectivity) calls, which in turn connects to database.
* It converts the JDBC method calls into ODBC function calls.

#### Pros:

* Any database that provides an ODBC driver can be accessed

Cons:

* Features are limited and restricted to what ODBC driver is capable of
* Platform dependent as it uses ODBC which in turn uses **native O/S libraries**
* ODBC driver must be installed on client machine
* **Limited portability** as **ODBC driver is platform dependent** & may not be available for all platforms
* **Poor Performance** because of several layers of translation that take place before the program connects to database
* It is now **obsolete** and only used for development and testing.
* It has been removed from JDK 8 (1.8)

### Type-2 Driver

* This was the second JDBC driver introduced by Java after Type 1, is hence it known as type 2.
* In this driver, performance was improved by **reducing communication layer**.
* Instead of talking to ODBC driver, **JDBC driver directly talks to DB** client using **native API**.
* That's why it’s also known as **native API or partly Java driver**
* Type 2 drivers use the client side libraries of the database.
* The driver converts JDBC method calls into native database API calls.

#### Pros:

* Faster than JDBC-ODBC bridge as there is no conversion like ODBC involved
* Since it required native API to connect to DB client it is also less portable and platform dependent.

#### Cons:

* Client side libraries needs to be installed on client machine
* Driver is platform dependent
* Not all database vendors provide client side libraries
* Performance of type 2 driver is slightly better than type 1 JDBC driver.

### Type-3 Driver

* This was the third JDBC driver introduced by Java, hence known as type 3.
* Type 3 driver makes use of middle tier between the Java programs and the database.
* Middle tier is an **application server** that converts JDBC calls into vendor-specific database calls.
* It was very different than type 1 and type 2 JDBC driver in sense that **it was completely written in Java** as opposed to previous two drivers which were not written in Java.
* That's why this is also known as all **Java driver**.
* This driver uses **3 tier approach i.e. client (java program), server and database**.
* So you have a Java client talking to a Java server and Java Server talking to database.
* Java client and server talk to each other using net protocol hence this type of JDBC driver is also known as **Net protocol JDBC driver**.
* This driver **never gained popularity** because database vendor was reluctant to rewrite their existing native library which was mainly in C and C++

#### Pros:

* No need to install any client side libraries on client machine
* Middleware application server can provide additional functionalities
* Database independence

#### Cons:

* Requires middleware specific configurations and coding
* May add extra latency as it goes through middleware server

### Type-4 Driver

* Type 4 drivers are also called **Pure Java Driver**.
* This is the driver you are most likely using to connect to modern database like Oracle, [SQL Server](http://javarevisited.blogspot.sg/2015/07/javalangclassnotfoundexception-com.microsoft.sqlserver.jdbc.SQLServerDriver.html), MySQL, SQLLite and [PostgreSQL](http://javarevisited.blogspot.sg/2015/06/org.postgresql.Driver-javalangclassnotfoundexception.html).
* This driver is implemented in Java and directly speaks to database using its native protocol.
* It converts JDBC calls directly into vendor-specific database protocol.
* This driver includes all database calls in **one JAR file**, which makes it very easy to use.
* All you need to do to connect a database from Java program is to include JAR file of relevant JDBC driver.
* Because of **light weight**, this is also known as **thin JDBC driver**.
* Since this driver is also written in **pure Java**, **it is portable across all platforms**, which means you can use same JAR file to connect to MySQL even if your Java program is running on Windows, Linux or Solaris.
* Performance of this type of JDBC driver is also best among all of them because **database vendor liked this type** and all enhancements they make they also port for type 4 drivers.

#### Pros:

* Written completely in Java hence platform independent
* Provides better performance than Type 1 and 2 drivers as there is no protocol specific conversion is required
* Better than Type 3 drivers as it doesn’t need additional middleware application servers
* Connects directly to database drivers without going through any other layer

#### Cons:

* Drivers are database specific

### Questions and Answers

**Question 1:** Which driver should we use?

Answer:

* A Type 4 driver is preferred if Java application is accessing any 1 database such as Oracle, Sybase etc.
* In case multiple databases are accessed then a Type 3 driver would be preferable.
* Type 2 drivers are recommended, if Type 3 or 4 drivers are not available for the database.
* Type 1 drivers are not recommended for production deployment.

### Question 2: What factors decide the choice of drivers?

### Answer: There are 2 factors that decide the choice of drivers: portability and performance.

### Question 3: Which driver is least used?

**Answer:** Type 1 JDBC driver is the poorest in terms of portability and performance. **It is no longer used.**

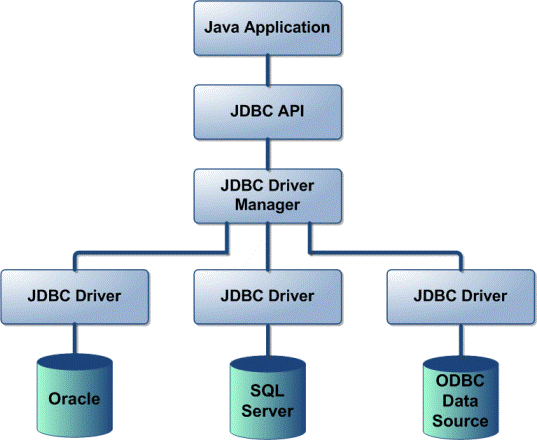
**Question 4:** Which is the best driver?

**Answer:** Type 4 JDBC driver is highly portable and gives the best performance.

**Question 5:** Which driver do we use to connect and transact with MySQL DB?

**Answer:** We use Type 4 driver

## JDBC Architecture



1. **Java Application** is any application that likes to connect and transact with any database.
2. **JDBC API**
   1. It provides the application-to-DB Connection
   2. It provides the driver manager (**java.sql.DriverManager**)
   3. It uses the database specific driver to connect to heterogeneous databases.
3. **Driver Manager**
   1. The JDBC driver manager ensures that the **correct driver is used** to access each data source
   2. The driver manager is capable of supporting **multiple concurrent drivers** connected to **multiple heterogeneous databases**.
   3. One application can connect to different databases simultaneously.
4. **JDBC driver API** supports the JDBC Database connection.
   1. **Database vendors** provide the **JDBC drivers.**
   2. For example: MySQL vendor provides “**mysql-connector-java-8.0.19**” jar file that contains “**com.mysql.cj.jdbc.Driver**”
5. **Databases**
   1. **A Java application** can connect and transact with **multiple databases** simultaneously or one at a time.
   2. The vendors provide their specific drivers
   3. The Driver Manager takes care of all the drivers

## Operations With DB

The following are the key operations we do with a database frequently.

1. Connect to DataBase
2. Execute Queries
   1. **C**reate/Insert Data
   2. **R**etrieve Data
   3. **U**pdate Data
   4. **D**elete Data
3. Close Connections/Resources

## JDBC 4.3 API

JDBC 4.0 API is mainly divided into two package

* java.sql
* javax.sql

### java.sql package

This package include classes and interface to perform almost all JDBC operation such as creating and executing SQL Queries.

#### Important classes and interface of java.sql package

|  |  |
| --- | --- |
| **classes/interface** | **Description** |
| java.sql.BLOB | Provide support for BLOB(Binary Large Object) SQL type. |
| java.sql.Connection | Creates a connection with specific database |
| java.sql.CallableStatement | Execute stored procedures |
| java.sql.CLOB | Provide support for CLOB(Character Large Object) SQL type. |
| java.sql.Date | Provide support for Date SQL type. |
| java.sql.Driver | Create an instance of a driver with the DriverManager. |
| java.sql.DriverManager | This class manages database drivers. |
| java.sql.PreparedStatement | Used to create and execute parameterized query. |
| java.sql.ResultSet | It is an interface that provide methods to access the result row-by-row. |
| java.sql.Savepoint | Specify savepoint in transaction. |
| java.sql.SQLException | Encapsulate all JDBC related exception. |
| java.sql.Statement | This interface is used to execute SQL statements. |