**FOUR LEVELS OF JDBC DRIVERS(PROS AND CONS)**

There are 4 different types of Drivers available in JDBC. They are classified based on the technique which is used to access a Database.

They are as follows:

* Type I : JDBC- ODBC Bridge
* Type II: Native APT- Partly Java Driver
* Type III: Network Protocol- Fully Java Driver
* Type IV: Thin Driver- Fully Java Driver

Type I: JDBC- ODBC Bridge

* In this type of Driver, JDBC – ODBC Bridge act as an interface between client and DB server. When a user uses Java application to send requests to the database using JDBC – ODBC Bridge, it first converts the JDBC API to ODBC API and then sends it to the database. When the result is received from DB, it is sent to ODBC API and then to JDBC API.
* This Driver is platform-dependent because it uses ODBC which depends on the native library of the OS. In this Type, JDBC – ODBC driver should be installed in each client system and the database must support the ODBC driver.
* When it is no matter about the installation in the client machine, we can use this driver. It is easy to use but it gives low performance due to the conversion of the JDBC method calls to the ODBC method calls.

Note:

* + It is available in JDK 1.2
  + Java 8 doesn’t support this type of driver. Oracle recommends that the user makes use of the JDBC drivers provided by their Database vendors.

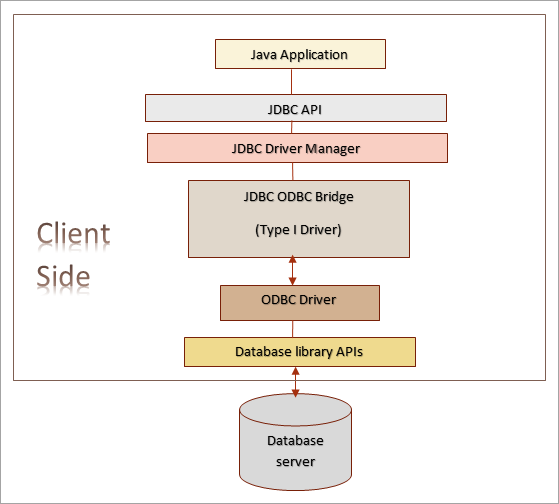


Figure: Type I: JDBC – ODBC Bridge Driver

Type II: Native API – Partially Java Driver

* It is similar to Type I Driver. Here, the ODBC part is replaced with native code in Type II Driver. This native code part is targeted at a specific database product. It uses the libraries of the client-side of the database. This Driver converts the JDBC method calls to native calls of the database native API.
* When the database gets the requests from the user, the requests are processed and sent back with the results in the native format which are to be converted to JDBC format and give it to the Java application.
* It was instantly adopted by the DB vendors because it was quick and inexpensive to implement. It is also in the native format of the DB.
* This type of driver gives faster response and performance than the Type I driver.

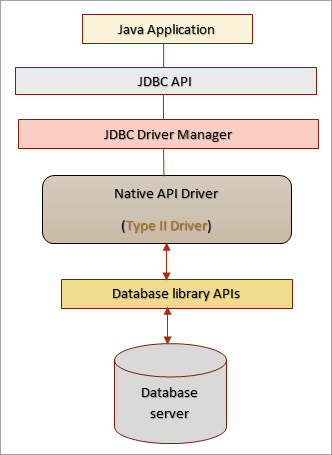


Figure: Type II: Native APT – Partly Java Driver

Type III: Network Protocol

* The type III driver is fully written in Java. It is like a 3-tier approach to access the database. It sends the JDBC method calls to an intermediate server. On behalf of the JDBC, the intermediate server communicates with the database. The Application server (intermediate or middle – tier) converts the JDBC calls either directly or indirectly to the vendor-specific Database protocol.
* This approach does not increase the architecture efficiency and it is also costly, due to this most of the database vendors don’t prefer this driver. Since the application server is used, you need to have good knowledge about the application server to use this approach efficiently.

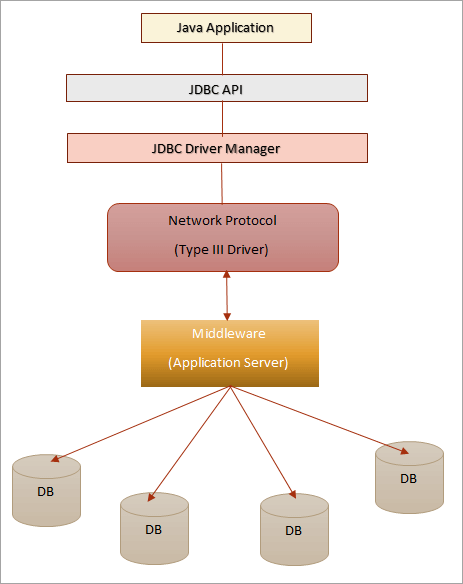


Figure: Type III: Network Protocol – Fully Java Driver

Type IV: Thin Driver

* Type IV driver is directly implemented that converts JDBC calls directly into vendor-specific Database protocol. Today, most of the JDBC Drivers are type IV drivers. It is written fully in Java and thus it is platform-independent. It is installed inside the JVM (Java Virtual Machine) of the client, so you don’t have to install any software on the client or server-side. This driver architecture has all the logic to communicate directly with the DB in a single driver.
* It provides better performance than the other type of drivers. It allows for easy deployment. Nowadays, this type of driver is developed by the database vendor itself so that programmers can use it directly without dependence on other sources.

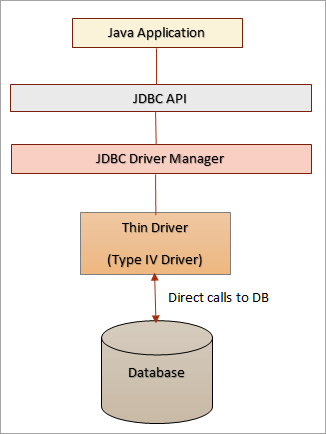


Figure: Type IV: Thin Driver – Fully Java Drive

The below are the drivers supports to specific database:

|  |  |  |
| --- | --- | --- |
| Database  (SQL/NoSQL) | JDBC Driver Provider Name | JAR File Name |
| MySQL | Open source community | mysql-connector-java-VERSION.jar |
| Oracle | Oracle Corporation | ojdbcX.jar |
| SQL Server | Microsoft Corporation | sqljdbc41.jar,  sqljdbc42.jar |
| Postgre SQL | PostgreSQL Global Development Group | postgresql-VERSION.jar |
| SQLite | Xerial.org | sqlite-jdbc-VERSION.jar |
| MS Access | UCanAccess.com | ucanaccess-VERSION.jar |
| Mongo DB |  | unityjdbc.jar and mongo-java-driver-2.12.2.jar. |

Difference between Two Tier and Three Tier Architecture

|  |  |
| --- | --- |
| Two Tier Architecture | Three Tier Architecture |
| Two-tier Architecture | Three-tier Architecture |
| * It helps Java application to directly connect with the database. * It needs a JDBC driver for the communication with a particular DB. * The user sends the requests to DB and receives the response directly without any mediator except JDBC Driver. * The database, either in the same machine or on a remote machine is connected via a network. * It can be called as a client-server architecture. | * There is no direct communication between the user and the database. * The user sends the request to the middle tier (Application Server) from which the request is again sent to Database. * Then the database processes the request and sends the result to the middle-tier from which the user receives the result/ response. |