JDBC driver

A **JDBC driver** is a software component enabling a Java application to interact with a database.[1] JDBC drivers are analogous to ODBC drivers, ADO.NET data providers, and OLE DB providers.

To connect with individual databases, JDBC (the Java Database Connectivity API) requires drivers for each database. The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

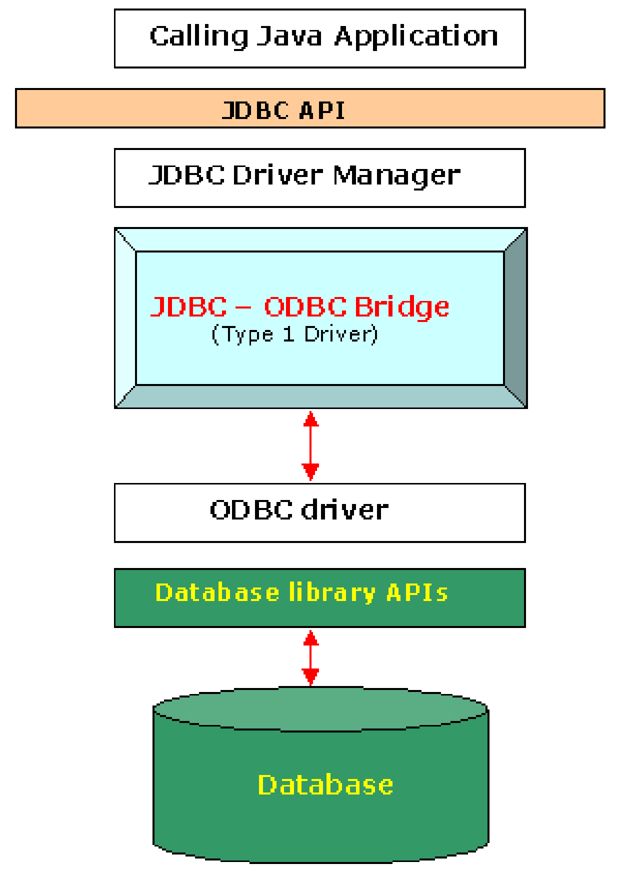
JDBC technology drivers fit into one of four categories.[2]

Type 1 Driver - JDBC-ODBC bridge

The JDBC type 1 driver, also known as the **JDBC-ODBC bridge**, is a database driver implementation that employs the ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls.

The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the underlying operating system the JVM is running upon. Also, use of this driver leads to other installation dependencies; for example, ODBC must be installed on the computer having the driver and the database must support an ODBC driver. The use of this driver is discouraged if the alternative of a pure-Java driver is available. The other implication is that any application using a type 1 driver is non-portable given the binding between the driver and platform. This technology isn't suitable for a high-transaction environment. Type 1 drivers also don't support the complete Java command set and are limited by the functionality of the ODBC driver.

Sun provides a JDBC-ODBC Bridge driver: sun.jdbc.odbc.JdbcOdbcDriver. This driver is native code and not Java, and is closed source.



Schematic of the JDBC-ODBC bridge

If a has been written so that loading it causes an instance to be created and also calls DriverManager.registerDriver with that instance as the parameter (as it should do), then it is in the DriverManager's list of drivers and available for creating a connection.

It may sometimes be the case that more than one JDBC driver is capable of connecting to a given URL. For example, when connecting to a given remote database, it might be possible to use a JDBC-ODBC bridge driver, a JDBC-to-generic-network-protocol driver, or a driver supplied by the database vendor. In such cases, the order in which the drivers are tested is significant because the DriverManager will use the first driver it finds that can successfully connect to the given URL.

First the DriverManager tries to use each driver in the order it was registered. (The drivers listed in jdbc.drivers are always registered first.) It will skip any drivers that are untrusted code unless they have been loaded from the same source as the code that is trying to open the connection.

It tests the drivers by calling the method Driver.connect on each one in turn, passing them the URL that the user originally passed to the method DriverManager.getConnection. The first driver that recognizes the URL makes the connection.

Advantages

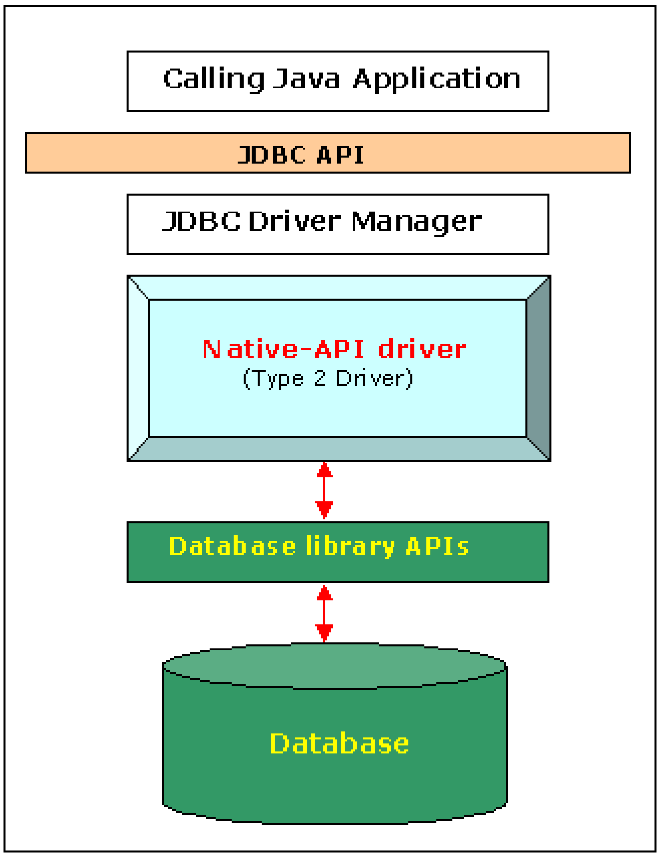
Almost any database for which ODBC driver is installed, can be accessed.

Disadvantages

- Performance overhead since the calls have to go through the jdbc Overhead bridge to the ODBC driver, then to the native db connectivity interface (thus may be slower than other types of drivers).
- The ODBC driver needs to be installed on the client machine.
- Not suitable for applets, because the ODBC driver needs to be installed on the client.

Type 2 Driver - Native-API Driver

The JDBC type 2 driver, also known as the **Native-API driver**, is a database driver implementation that uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API.



Schematic of the Native API driver

Advantages

- As there is no implementation of jdbc-odbc bridge, its considerably faster than a type 1 driver.

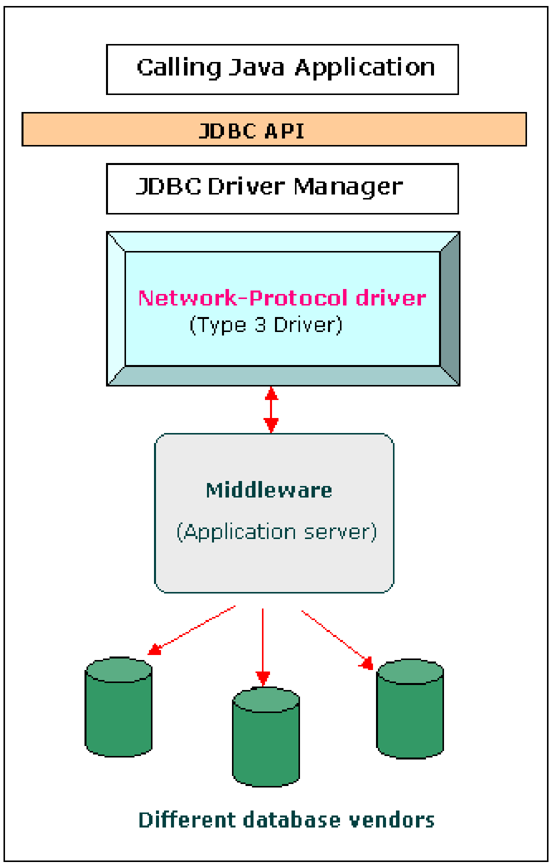
Disadvantages

- The vendor client library needs to be installed on the client machine.
- Not all databases have a client side library
- This driver is platform dependent
- This driver supports all java applications except Applets

Type 3 Driver - Network-Protocol Driver(MiddleWare Driver)

The JDBC type 3 driver, also known as the Pure Java Driver for Database **Middleware**, is a database driver implementation which makes use of a middle tier between the calling program and the database. The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol.

This differs from the type 4 driver in that the protocol conversion logic resides not at the client, but in the middle-tier. Like type 4 drivers, the type 3 driver is written entirely in Java. The same driver can be used for multiple databases. It depends on the number of databases the middleware has been configured to support. The type 3 driver is platform-independent as the platform-related differences are taken care of by the middleware. Also, making use of the middleware provides additional advantages of security and firewall access.



Schematic of the Network Protocol driver

Functions

- Sends JDBC API calls to a middle-tier net server that translates the calls into the DBMS-specific network protocol. The translated calls are then sent to a particular DBMS.
- Follows a three tier communication approach.
- Can interface to multiple databases - Not vendor specific.
- The JDBC Client driver written in java, communicates with a middleware-net-server using a database independent protocol, and then this net server translates this request into database commands for that database.
- Thus the client driver to middleware communication is database independent.

Advantages

- Since the communication between client and the middleware server is database independent, there is no need for the database vendor library on the client. The client need not be changed for a new database.
- The middleware server (which can be a full fledged J2EE Application server) can provide typical middleware services like caching (of connections, query results, etc.), load balancing, logging, and auditing.
- A single driver can handle any database, provided the middleware supports it.
- E.g.:-IDA Server

Disadvantages

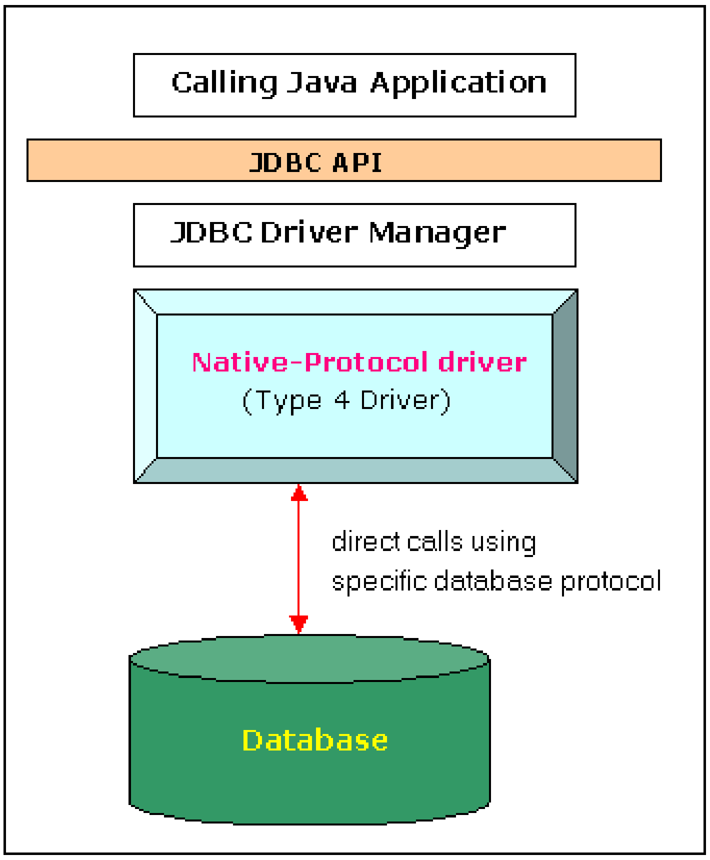
- Requires database-specific coding to be done in the middle tier.
- The middleware layer added may result in additional latency, but is typically overcome by using better middleware services.

Type 4 Driver - Native-Protocol Driver(Pure Java Driver)

The JDBC type 4 driver, also known as the Direct to Database **Pure Java Driver**, is a database driver implementation that converts JDBC calls directly into a vendor-specific database protocol.

Written completely in Java, type 4 drivers are thus platform independent. They install inside the Java Virtual Machine of the client. This provides better performance than the type 1 and type 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls. Unlike the type 3 drivers, it does not need associated software to work.

As the database protocol is vendor specific, the JDBC client requires separate drivers, usually vendor supplied, to connect to different types of databases. This type includes, for example, the widely used Oracle thin driver.



Schematic of the Native-Protocol driver

Advantages

- Completely implemented in Java to achieve platform independence.
- These drivers don't translate the requests into an intermediary format (such as ODBC).
- The client application connects directly to the database server. No translation or middleware layers are used, improving performance.
- The JVM can manage all aspects of the application-to-database connection; this can facilitate debugging.

Disadvantages

- Drivers are database dependent, as different database vendors use wildly different (and usually proprietary) network protocols.

List of JDBC Drivers

- List of jdbc vendors registered with Oracle [3]
- List of drivers registered with Oracle [4]
- Open Source Performance Benchmark [5]

References

[1] "Java SE Technologies - Database" (http://java.sun.com/javase/technologies/database/)
[2] Sun JDBC Overview (http://www.oracle.com/technetwork/java/overview-141217.html)
[3] http://www.oracle.com/technetwork/java/index-136695.html
[4] http://devapp.sun.com/product/jdbc/drivers
[5] http://www.polepos.org/

Article Sources and Contributors

JDBC driver *Source*: http://en.wikipedia.org/w/index.php?oldid=555733383 *Contributors*: 2001:470:1F0B:447:48B4:8635:8EDC:79B3, Alainr345, Alphachimp, Apavlo, Barkeep, Cmdrjameson, Crohnie, Danim, Denisarona, Dinesh92kumar, Doug Bell, DragonLord, Dukey42, Forage, Frank McLean, Fritzpoll, Fæ, Hermandr, Hk.agarwal, I dream of horses, IGeMiNix, Jandalhandler, Jay, Jk2q3jrklse, John of Reading, Joke dst, Josepant, JubalHarshaw, Kadishmal, Kai Ojima, Kiran joseph, Klausness, Lam Kin Keung, MacTed, McSly, Mike Rosoft, MikeLynch, Minkythecat, NubKnacker, Papa November, Pearle, Praetor alpha, Quietgenie, RadiantRay, SJK, Sleske, Spearhead, Sreenivasa Reddy Mulinti, Stevage, Subterrane, The Thing That Should Not Be, TucsonDavid, Vaibsrock, VictorianMutant, Vilerage, Vivek kodira, Vrenator, Wavelength, Welsh, Widr, Yaksha, 231 anonymous edits

Image Sources, Licenses and Contributors

Image:JDBC driver.png *Source*: http://en.wikipedia.org/w/index.php?title=File:JDBC_driver.png *License*: GNU Free Documentation License *Contributors*: Joey-das-WBF, LoStrangolatore, WikipediaMaster

Image:Native API driver.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Native_API_driver.png *License*: GNU Free Documentation License *Contributors*: Original uploader was Jay at en.wikipedia

Image:Network Protocol driver.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Network_Protocol_driver.png *License*: GNU Free Documentation License *Contributors*: Jay. Original uploader was Jay at en.wikipedia

Image:Native Protocol driver.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Native_Protocol_driver.png *License*: GNU Free Documentation License *Contributors*: Jay at en.wikipedia

License