JDBC Architecture

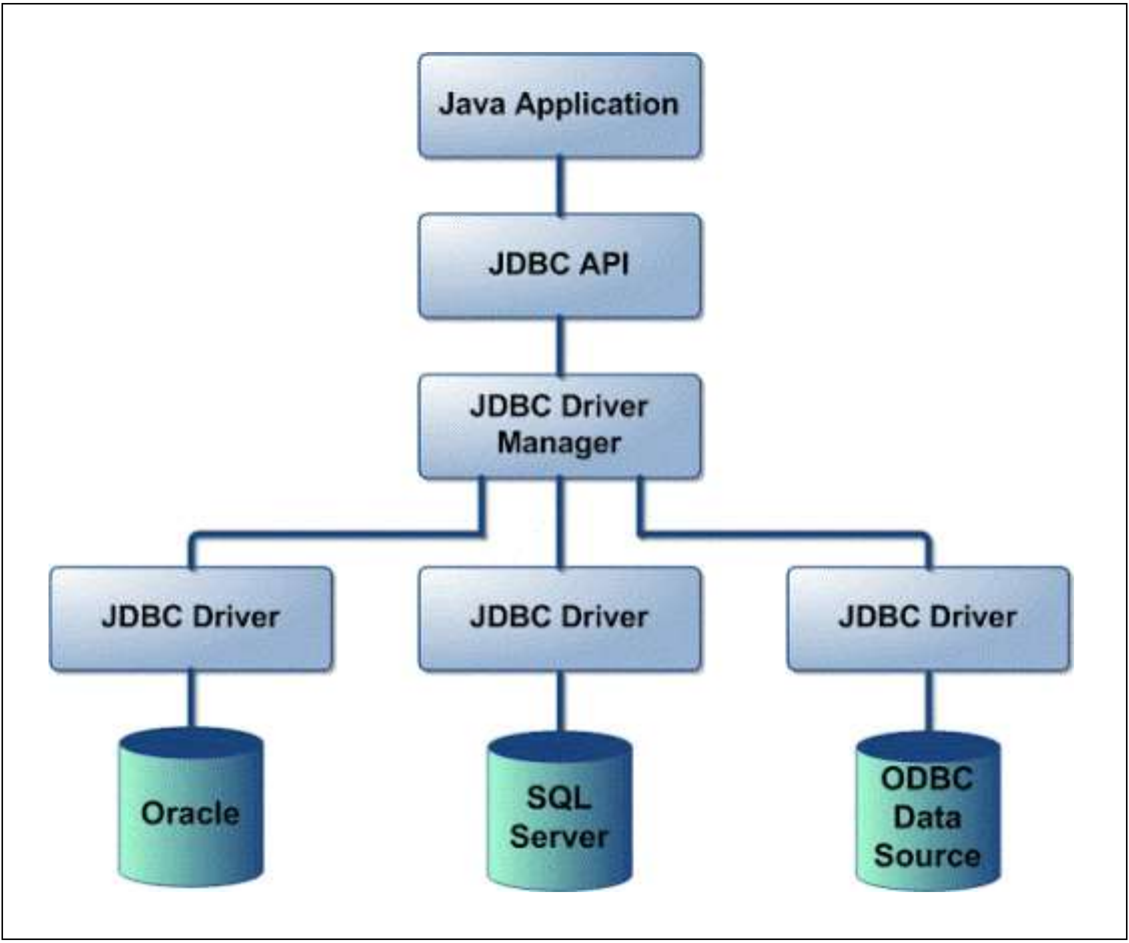
JDBC or Java Database Connectivity is a specification from Sun microsystems that provides a standard abstraction (that is API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standard. It is used to write programs required to access databases. JDBC along with the database driver is capable of accessing databases and spreadsheets. JDBC is an API (Application programming interface) which is used in java programming to interact with databases. The classes and interfaces of JDBC allows application to send request made by users to the specified database. Applications that are created using the JAVA need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity which can be achieved by using the ODBC (Open database connectivity) driver. This driver is used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql and SQL server database.

Architecture of JDBC

The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

- JDBC API: This provides the application-to-JDBC Manager connection.
- JDBC Driver API: This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases. Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application.



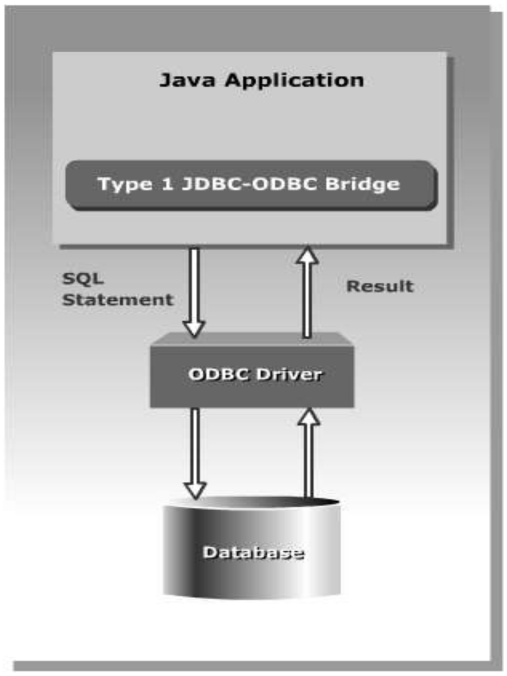
JDBC drivers

JDBC drivers are client-side adapters that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

- Type-1 driver or JDBC-ODBC Bridge driver
- Type-2 driver or Native-API Partly-Java driver (partially java driver)
- Type-3 driver or JDBC-Net Pure-Java driver (fully java driver)
- Type-4 driver or Native-Protocol Pure-Java driver or Thin driver

JDBC-ODBC Bridge driver

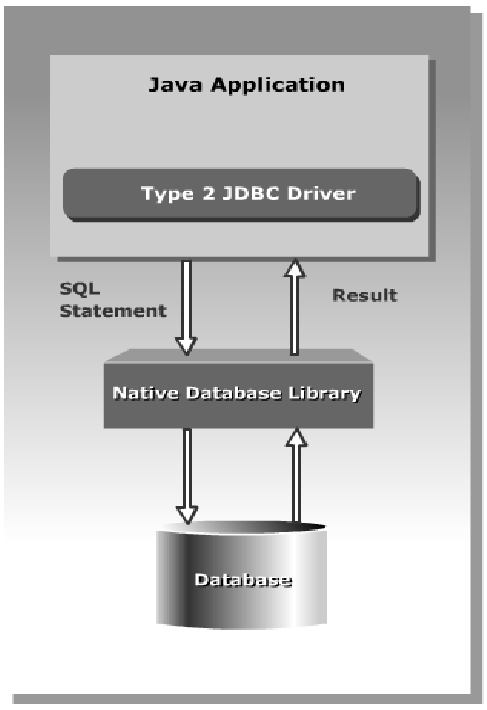
The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases. As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured. The ODBC bridge driver is needed to be installed in individual client machines.



Type-1 driver isn't written in java, that's why it isn't a portable driver. This driver software is built-in with JDK so no need to install separately. It is a database independent driver.

Native-API Partly-Java driver

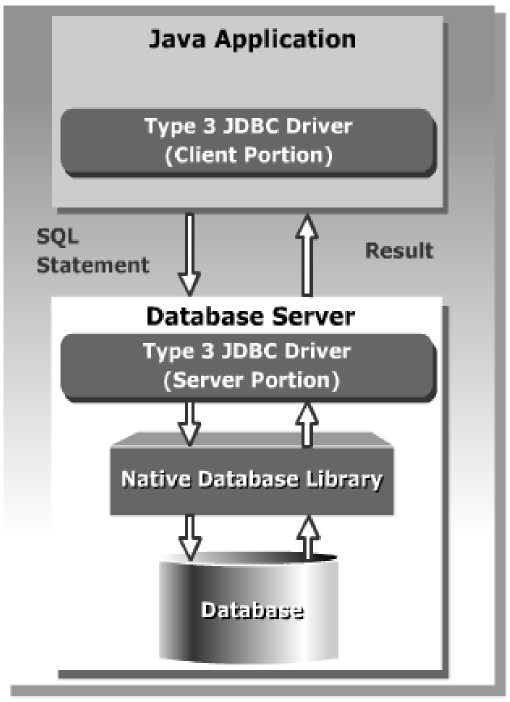
The Native API driver uses the client-side libraries of the database.



The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java. In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver. Driver needs to be installed separately in individual client machines. The Vendor client library needs to be installed on client machine. Type-2 driver isn't written in java, that's why it isn't a portable driver. It is a database dependent driver.

JDBC-Net Pure-Java driver

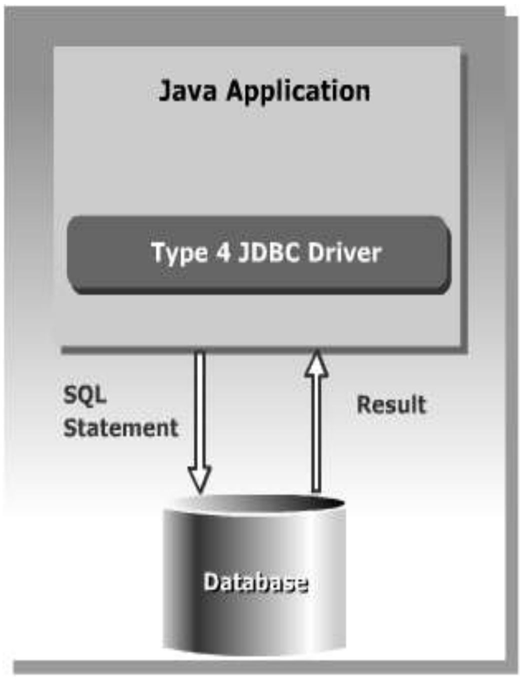
The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation. Type-3 drivers are fully written in Java, hence they are portable drivers.



No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc. Network support is required on client machine. Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier. Switch facility to switch over from one database to another database.

Native-Protocol Pure-Java driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language. It does not require any native database library that is why it is also known as Thin Driver.



This driver does not require any native library and Middleware server, so no client-side or server-side installation. It is fully written in Java language, hence they are portable drivers.