© 2014 IJIRT | Volume 1 Issue 5 | ISSN : 2349-6002

WHY USE JDBC

Tushar Wason, Anubhav Chandra
Student, Department of Information Technology,
Dronacharya College of Engineering, Gurgaon, Hr, India

***Abstract*—The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases – SQL databases and other tabular data sources, such as spreadsheets or flat files. The JDBC API provides a call-level API for SQL-based database access.**

JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data. With a JDBC technology-enabled driver, you can connect all corporate data even in a heterogeneous environment.

***Index Trems*- JDBC, API, Drivers**

I. INTRODUCTION

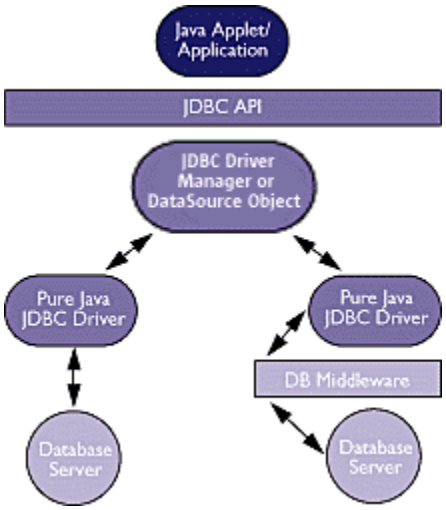
The Internet has spurred the invention of several new technologies in client/server computing—the most recent of which is Java. Java is two-dimensional: It's a programming language and also a client/server system in which programs are automatically downloaded and run on the local machine (instead of the server machine). The wide embrace of Java has prompted its quick development. Java includes Java compilers, interpreters, tools, libraries, and integrated development environments (IDEs). Javasoft is leading the way in the development of libraries to extend the functionality and usability of Java as a serious platform for creating applications. One of these libraries, called Application Programming Interfaces (APIs), is the Java Database Connectivity API, or JDBC. Its primary purpose is to intimately tie connectivity to databases with the Java language. We'll discuss the reasoning behind the JDBC in this chapter, as well as the design of the JDBC and its associated API. The Internet, or better yet, the technologies used in the operation of the Internet, are tied into the design of the JDBC. The other dominant design basis for the JDBC is the database standard known as SQL. Hence, the JDBC is a fusion of three discrete computer areas: Java, Internet technology, and SQL. With the growing implementation of these Internet technologies in "closed" networks, called intranets, the time was right for the development of Java-based enterprise APIs. In this book, intranet and Internet are both used to describe the software technology behind the network, such as the World Wide Web.

II. JDBC API OVERVIEW

The JDBC API makes it possible to do three things:
Establish a connection with a database or access any tabular data source
Send SQL statements
Process the results

III. JDBC ARCHITECTURE

The JDBC API contains two major sets of interfaces: the first is the JDBC API for application writers, and the second is the lower-level JDBC driver API for driver writers. JDBC technology drivers fit into one of four categories. Applications and applets can access databases via the JDBC API using pure Java JDBC technology-based drivers, as shown in this figure:



IV. WHAT IS JDBC DRIVER ?

JDBC drivers implement the defined interfaces in the JDBC API for interacting with your database server.

For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java. The *Java.sql* package that ships with JDK contains various classes with their behaviours defined and their actual implementaions are done in third-party drivers. Third party vendors implements the *java.sql.Driver* interface in their database driver.

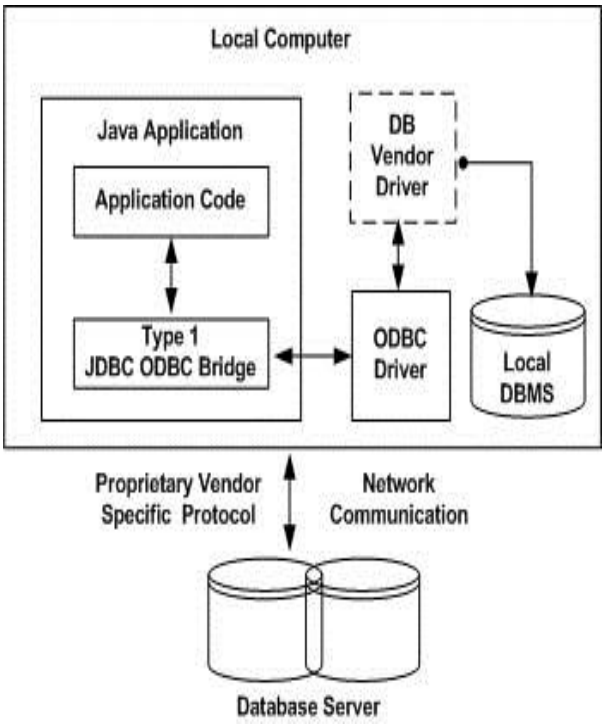
V. JDBC DRIVERS TYPES

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below:

Type 1: JDBC-ODBC Bridge Driver:

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC requires configuring on your system a Data Source Name (DSN) that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

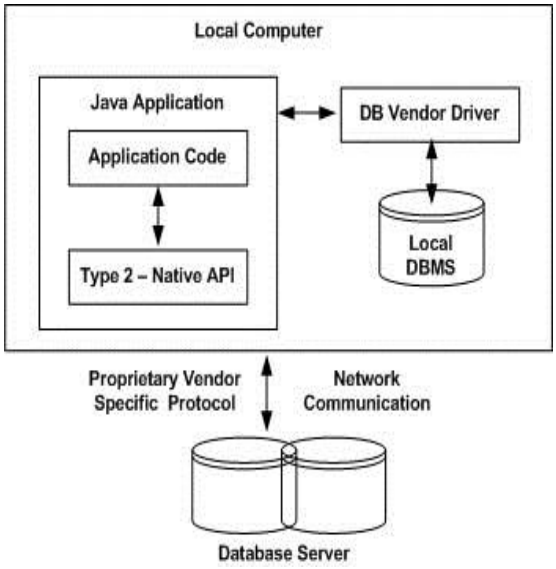


The JDBC-ODBC bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API:

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls which are unique to the database. These drivers typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge, the vendor-specific driver must be installed on each client machine.

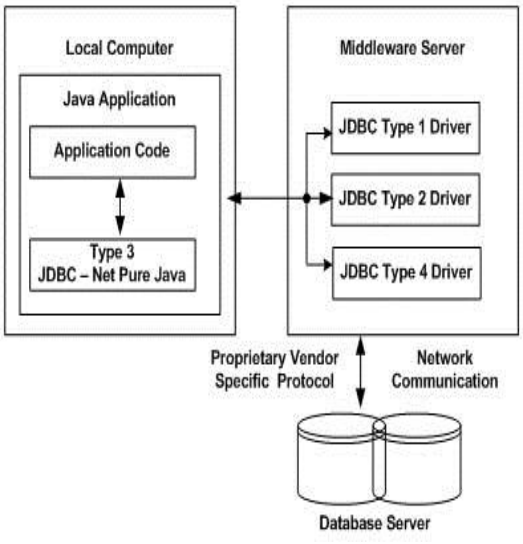
If we change the Database we have to change the native API as it is specific to a database and they are mostly obsolete now but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.



The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 3: JDBC-Net pure Java:

In a Type 3 driver, a three-tier approach is used to accessing databases. The JDBC clients use standard network sockets to communicate with an middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

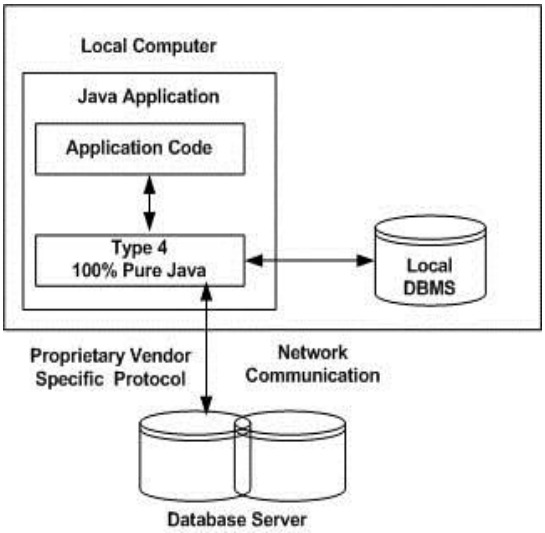


You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Type 4: 100% pure Java:

In a Type 4 driver, a pure Java-based driver that communicates directly with vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

VI. WHICH DRIVER SHOULD BE USED?

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.Type 2 drivers are useful in situations where a type 3 or type 4 driver is not available yet for your database.The type 1 driver is not considered a deployment-level driver and is typically used for development and testing purposes only.

Key Features

Full Access to Metadata
The JDBC API provides metadata access that enables the development of sophisticated applications that need to understand the underlying facilities and capabilities of a specific database connection.

No Installation
A pure JDBC technology-based driver does not require special installation; it is automatically downloaded as part of the applet that makes the JDBC calls.

Database Connection Identified by URL
JDBC technology exploits the advantages of Internet-standard URLs to identify database connections. The JDBC API includes an even better way to identify and connect to a data source, using a DataSource object, that makes code even more portable and easier to maintain.

VII. Advantages of JDBC Technology

Leverage Existing Enterprise Data
With JDBC technology, businesses are not locked in any proprietary architecture, and can continue to use their installed databases and access information easily -- even if it is stored on different database management systems.

Simplified Enterprise Development
The combination of the Java API and the JDBC API makes application development easy and economical. JDBC hides the complexity of many data access tasks, doing most of the "heavy lifting"for the programmer behind the scenes. The JDBC API is simple to learn, easy to deploy, and inexpensive to maintain.

Zero Configuration for Network Computers
With the JDBC API, no configuration is required on the client side. With a driver written in the Java programming language, all the information needed to make a connection is completely defined by the JDBC URL or by a DataSource object registered with a Java Naming and Directory Interface (JNDI) naming service. Zero configuration for clients supports the network computing paradigm and centralizes software maintenance.

REFERENCE

[1]http://www.oracle.com/technetwork/java/overview-141217.html
[2]http://www.studymode.com/course-notes/Jdbc-Notes-1064864.html
[3]http://en.wikipedia.org/wiki/Java_Database_Connectivity