www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2. Issue 10 October 2013 Page No. 2979-2984

JAVA DATABASE CONNECTIVITY (JDBC) - DATA ACCESS TECHNOLOGY

Aditi Khazanchi, Akshay Kanwar, Lovenish Saluja

ABSTRACT

A serious problem facing many organizations today is the need to use information from multiple data sources that have been developed separately. To solve this problem, Java Database Connectivity came into existence. JDBC helps us to connect to a database and execute SQL statements against a database. JDBC API provides set of interfaces and there are different implementations respective to different databases. This paper emphasis on its history and implementation, architecture and JDBC drivers.

INTRODUCTION

JDBC is a Java-based data access technology from Oracle Corporation. This technology is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC (Java Database Connectivity) is a standard API for accessing relational databases from a Java program. This interface makes it easy to access a database because it provides an abstract layer that hides the low-level details, such as managing sockets. It also provides for interoperability and portability since it allows a single application to access multiple database management systems simultaneously. For example, a single application can query and manipulate a database in Oracle and a database in DB2. JDBC actually has two levels of interface. In addition to the main interface, there is also an API from a JDBC "manager" that in turn communicates with individual database product "drivers," the JDBC-ODBC Bridge if necessary, and a JDBC network driver when the Java program is running in a network environment (that is, accessing a remote database).

When accessing a remote database, JDBC takes advantage of the Internet's file addressing scheme and a file name looks much like a Web page address (or Uniform Resource Locator). For example, a Java SQL statement might identify the database as :

jdbc:odbc://www.somecompany.com:400/databasefile

Advantages of JDBC:

- Can read any database if proper drivers are installed.
- Creates XML structure of data from database automatically
- No content conversion required
- Query and Stored procedure supported.
- Can be used for both Synchronous and Asynchronous processing.
- Supports modules

EVOLUTION

History and Implementation

Sun Microsystems released JDBC as part of JDK 1.1 on February 19, 1997.It has since formed part of the Java Standard Edition. The JDBC classes are contained in the Java package java.sql and javax.sql. Starting with version 3.1, JDBC has been developed under the Java Community Process. JSR 54 specifies JDBC 3.0 (included in J2SE 1.4), JSR 114 specifies the JDBC Rowset additions, and JSR 221 is the specification of JDBC 4.0 (included in Java SE 6).The latest version,

JDBC 4.1, is specified by a maintenance release of JSR 221[3] and is included in Java SE 7.

Functionality

JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. These may be update statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT. Additionally, stored procedures may be invoked through a JDBC connection. JDBC represents statements using one of the following classes:

Statement – the statement is sent to the database server each and every time.

PreparedStatement – the statement is cached and then the execution path is pre-determined on the database server allowing it to be executed multiple times in an efficient manner.

CallableStatement – used for executing stored procedures on the database.

Update statements such as INSERT, UPDATE and DELETE return an update count that indicates how many rows were affected in the database. These statements do not return any other information.Query statements return a JDBC row result set. The row result set is used to walk over the result set. Individual columns in a row are retrieved either by name or by column number. There may be any number of rows in the result set. The row result set has metadata that describes the names of the columns and their types.

There is an extension to the basic JDBC API in the javax.sql.

JDBC connections are often managed via a connection pool rather than obtained directly from the driver. Examples of connection pools include BoneCP, C3P0 and DBCP.

COMPONENTS OF JDBC

The JDBC API provides the following interfaces and classes:

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication subprotocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manage objects of this type. It also abstracts the details associated with working with Driver objects
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

ARCHITECTURE

The JDBC API supports both two-tier and three-tier processing models for database access.

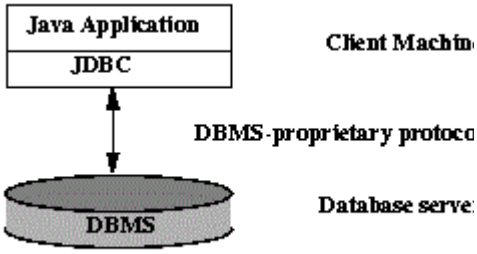

Fig: **Two-tier Architecture for Data Access.**

In the two-tier model, a Java application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

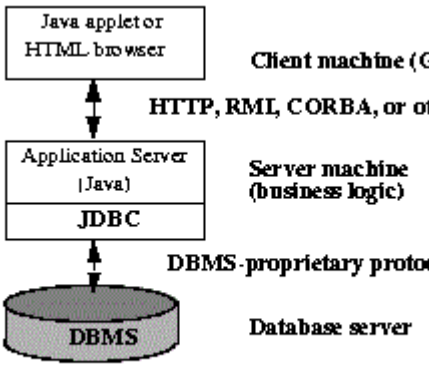

Fig: **Three-tier Architecture for Data Access.**

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

5 JDBC DRIVERS

A JDBC driver is a software component enabling a Java application to interact with database. 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-

- Type 1 Driver - JDBC-ODBC Bridge
- Type 2 Driver - Native-API Driver
- Type 3 Driver - Network-Protocol Driver(MiddleWare Driver)
- Type 4 Driver - Database-Protocol Driver(Pure Java Driver)

5.1 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.
If a driver 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.
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.

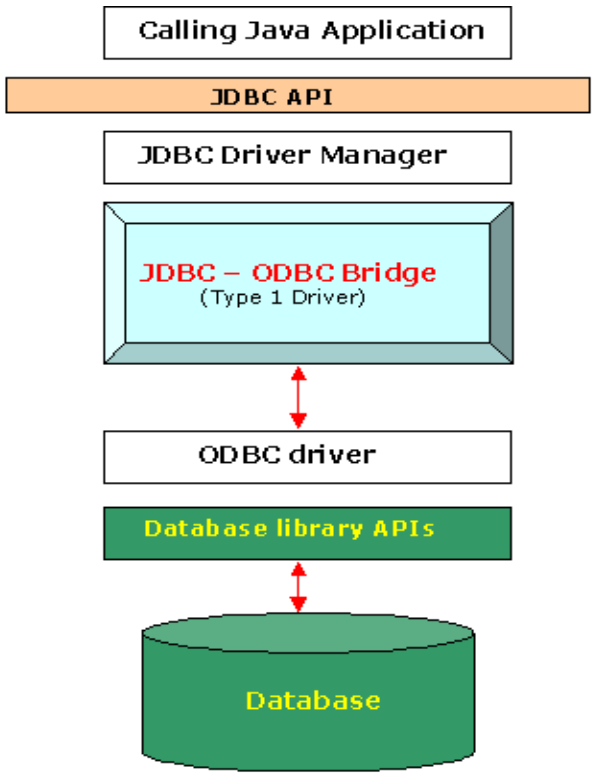

Fig. Schematic of the JDBC-ODBC Bridge

5.2 Type 2 Driver - Native-API Driver

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.

1. **Advantages**: As there is no implementation of jdbc-odbc Bridge, its considerably faster than a type 1 driver.
2. **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

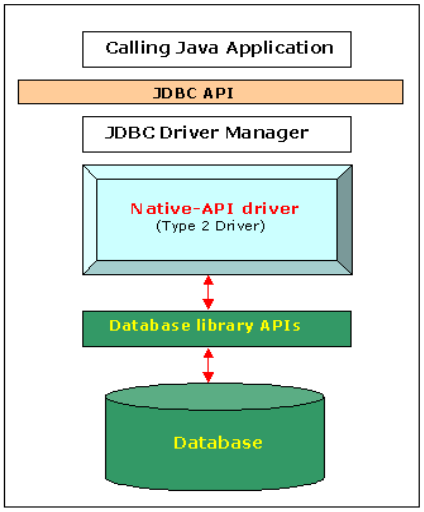

Fig. Schematic of the Native API driver

5.3 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.

5.3.1 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.

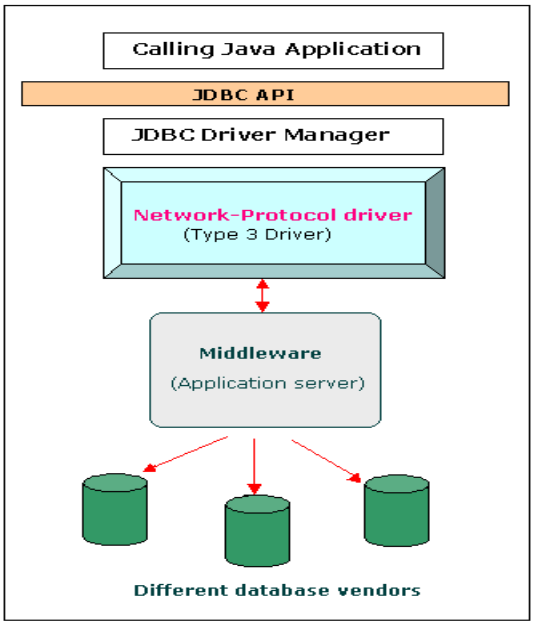

Fig. Schematic of the Network Protocol driver

5.4 Type 4 Driver - Database-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.

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.

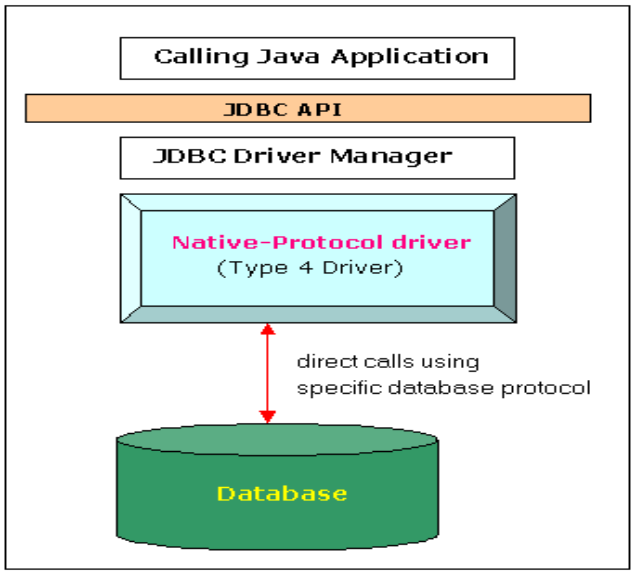

Fig. Schematic of the Native-Protocol driver

CONCLUSION

We presented the JDBC, an API(application programming interface) for java that allows the Java programmer to access the Database. The JDBC API consists of a numbers of classes and interfaces, written in java programming language, they provides a numbers of methods for updating and querying a data in a database. It is a relational database oriented driver. It allows the java application to reuse database connection the connection that has been created already instead of creating a new connection every time. Creating and destroying a database connection is very costly, therefore this feature is very important for java application.

REFERENCES

http://www.cs.ubc.ca/~ramesh/cpsc304/tutorial/JDBC/jdbc1.html

http://en.wikipedia.org/wiki/Java_Database_Connectivity

http://searchoracle.techtarget.com/definition/Java-Database-Connectivity

http://docs.oracle.com/javase/tutorial/jdbc/overview/architecture.html

http://www.tutorialspoint.com/jdbc/jdbc-introduction.htm

http://en.wikipedia.org/wiki/JDBC_driver

http://www.tutorialspoint.com/jdbc/jdbc-driver-types.htm