Review on Java Database Connectivity

Ms. Poonam Walimbe

Lecturer, Department of Computer Engineering, V.P.M's Polytechnic, Thane, Maharashtra

Abstract - *Java Database with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from 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.*

***Key Words*: JDBC, JDBC API, JVM, Database, SQL, Servlet, JSP**

1. INTRODUCTION

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executable, such as –

- Java Applications
- Java Applets
- Java Servlets
- Java Server Pages (JSPs)
- Enterprise JavaBeans (EJBs).

All of these different executable are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

1.1 Creating JDBC Application

There are following six steps involved in building a JDBC application –

• Import the packages: Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using import java.sql.* will suffice.

• Register the JDBC driver: Requires that you initialize a driver so you can open a communication channel with the database.

• Open a connection: Requires using the Driver Manager. Get Connection () method to create a connection object, which represents a physical connection with the database.

• Execute a query: Requires using an object of type Statement for building and submitting an SQL statement to the database.

• Extract data from result set: Requires that you use the appropriate Result Set.getXXX() method to retrieve the data from the result set.

• Clean up the environment: Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

1.2 Common JDBC Components

The JDBC API provides the following interfaces and classes:

• Driver Manager: This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol.

The first driver that recognizes a certain sub protocol 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 Driver Manager objects, which manages 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.

• Result Set: 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.

• SQL Exception: This class handles any errors that occur in a database application.

1.3 JDBC Connection:

The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps:

1. Import JDBC Packages: Add import statements to your Java program to import required classes in your Java code.

2. Register JDBC Driver: This step causes the JVM to load the desired driver implementation into memory so it can fulfill your JDBC requests.

3. Database URL Formulation: This is to create a properly formatted address that points to the database to which you wish to connect.

4. Create Connection Object: Finally, code a call to the Driver Manager object's get Connection ( ) method to establish actual database connection.

5. Close Connections : Finally, we have to explicitly close the connections that we have opened.

2. JDBC Architecture:

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 :

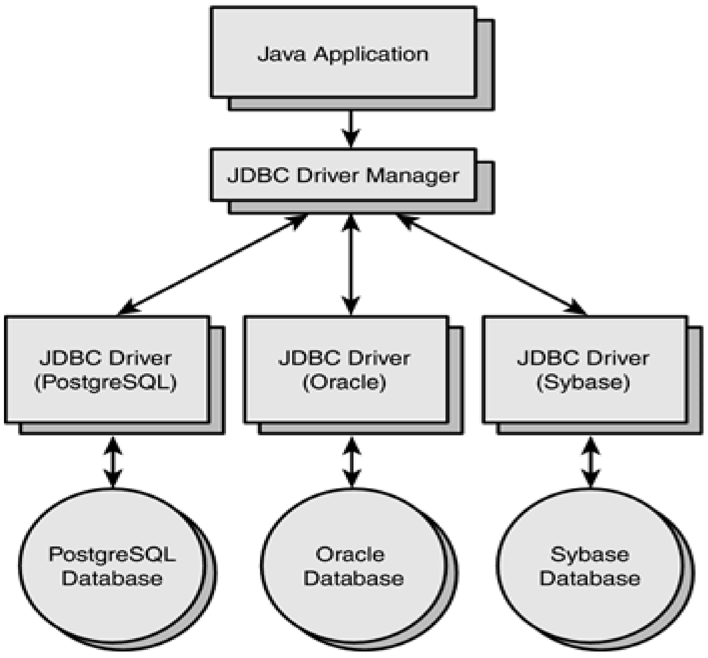

Fig -1: JDBC Architecture overview

3. CONCLUSIONS

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.

REFERENCES

[1]http://en.wikipedia.org/wiki/Java_Database_Connectivity

[2]http://searchoracle.techtarget.com/definition/JavaDatabase-Connectivity

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

[4]http://www.tutorialspoint.com/jdbc/jdbcintroduction.htm

[5]http://en.wikipedia.org/wiki/JDBC_driver

[6]http://www.tutorialspoint.com/jdbc/jdbcdrivertypes.htm