Java DataBase Connectivity (JDBC)

By,

Hitha Paulson

Assistant Professor, Dept. of Computer Science

LF College, Guruvayoor

JDBC

- JDBC is an interface between Java and Database
- JDBC receives queries from Java Application program and communicate with Database
- All the communications are in the form of SQL commands
- JDBC is reponsible for
  - Open a Connection
  - Communicate with database
  - Execute SQL statements
  - Retrive query results

Open DataBase Connectivity

- Standard designed by Microsoft to interact with databases
- ODBC is packed with many features and extending support for all type of databases
- ODBC provides multiple mechanism for performing single task and number of data handling capabilities

JDBC vs ODBC

- JDBC and ODBC are X/OPEN call level interface for SQL
- JDBC is not a derivative of ODBC
- JDBC is compact and simple
- JDBC is meant for simple access to the database and difficult task at least make possible

JDBC Drivers

- JDBC driver is responsible for making connection with different databases
- It is also translating the queries received from Application and submit into database
- A reverse translation is also required to perform by the Driver
- JDBC Driver speaks JAVA to Application and native language to database
- JDBC Drivers are exists for almost all databases
- Appropriate driver will load for requied database

JDBC – ODBC Bridge

- It is a JDBC driver designed to let Java application communicate with database via an underlying ODBC driver
- It is called Type I JDBC connector
- It can be used with multiple databases and is vendor independent
- This type JDBC driver speaks only to ODBC driver, hence works for Databases supported by ODBC
- One more added layer is used and hence more complex and slower than JDBC drivers

Native-API-Partly-Java Driver

- It make use of local native libraries to communicate with database
- Vendor specific Call Level Interface(CLI) installed locally are used by this type driver
- CLI libraries are actually communicate with database
- Application level requests are translated into equallent native method call of CLI
- Faster than Type I driver

JDBC-Net-All-Java Driver

- Type III Driver, uses the CLI libraries locates in a remote server
- Type III driver has two major components
  - An All-Java portion that can download to the client
  - Server portion containing both Java and Native methods
- All communication between Application and Database is 100% Java to Java
- This type of driver is also depending on CLI calls, which is installed on Server
- Type III can be used in Internet, since no direct access to CLI libraries
- Type III Network protocol is not standardized

Native-Protocol-All-Java Driver

- 100% java specific drivers
- No intermediate translation is required
- But all vendor specific driver cannot released by Java
- Java Applets are now free from Acces restrictions

JDBC Implementation

- Seven Steps
  - Import java.sql package
  - Load and register the driver
  - Establish a connection to the database server
  - Create a statement
  - Execute the statement
  - Retrive the result
  - Close the statement and connection

Load and Register Driver

Class.forName("Driver ClassName");

Eg:1 Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”)

2 Class.forName("com.mysql.jdbc.Driver");

Note: Calling the Class.forName automatically creates an instance of a driver and registers it with the DriverManager, so you don't need to create an instance of the class

Establish Connection

Connection conn=DriverManager.getConnection("URL");

The drivers loaded recognizes, the JDBC URL in DriverManager.getConnection, that driver establishes a connection to the DBMS specified in the JDBC URL.
The DriverManager class,manages all of the details of establishing the connection

The connection returned by the method DriverManager.getConnection is an open connection you can use to create JDBC statements that pass your SQL statements to the DBMS.

DriverManager.getConnection(“jdbc:mysql://172.16.5.27/campusdb”,”mca”,”mca”);

Managing Statement

createStatement() of **Connection** class is used to make object of **Statements**.

Eg: Statement stat=con.createStatement();

Statement object can call **executeQuery**(“SQL Command”) to execute a select statement.

Use **executeUpdate**(“SQL Command”) to execute any data updation commands

Resultset

An executeQuery() method retrives the selected records as an object of ResultSet class

It stores data in tabular format. Rowid and ColID can be used to identify each data.

Rows are records of table and columns are fields of table

A cursor is attached to fetch data from any row

Metadata

- Information that describes the structure and properties of your data
- Two types of Metadata
  - Resultset Metadata: Information about the data contained in a Resultset, such as column name, number of columns and column data types
  - Database Metadata: Information about database, such as supported functions, username, current transaction isolation level