Basics of JDBC

Overview

- What is JDBC?
- Why need an application other than DBMS?
- How JDBC can help?
- JDBC Steps
- Transactions in JDBC
- JDBC Drivers
- Summary

What is JDBC?

- JDBC is: a Sun trademark
  - is often taken to stand for Java Database Connectivity.
  - is a Java API for connecting programs written in Java to the data in relational databases.
  - consists of a set of classes and interfaces written in the Java programming language.
  - provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API.
  - The standard defined by Sun Microsystems, allowing individual providers to implement and extend the standard with their own JDBC drivers.

JDBC

- Java is very standardized, but there are many versions of SQL
- JDBC is a means of accessing SQL databases from Java
  - JDBC is a standardized API for use by Java programs
  - JDBC is also a specification for how third-party vendors should write database drivers to access specific SQL versions
- JDBC:
  - establishes a connection with a database
  - sends SQL statements
  - processes the results.

The need of an application

- Databases offer structured storing, easy retrieval and processing of data
- The maintenance, however, can be tedious

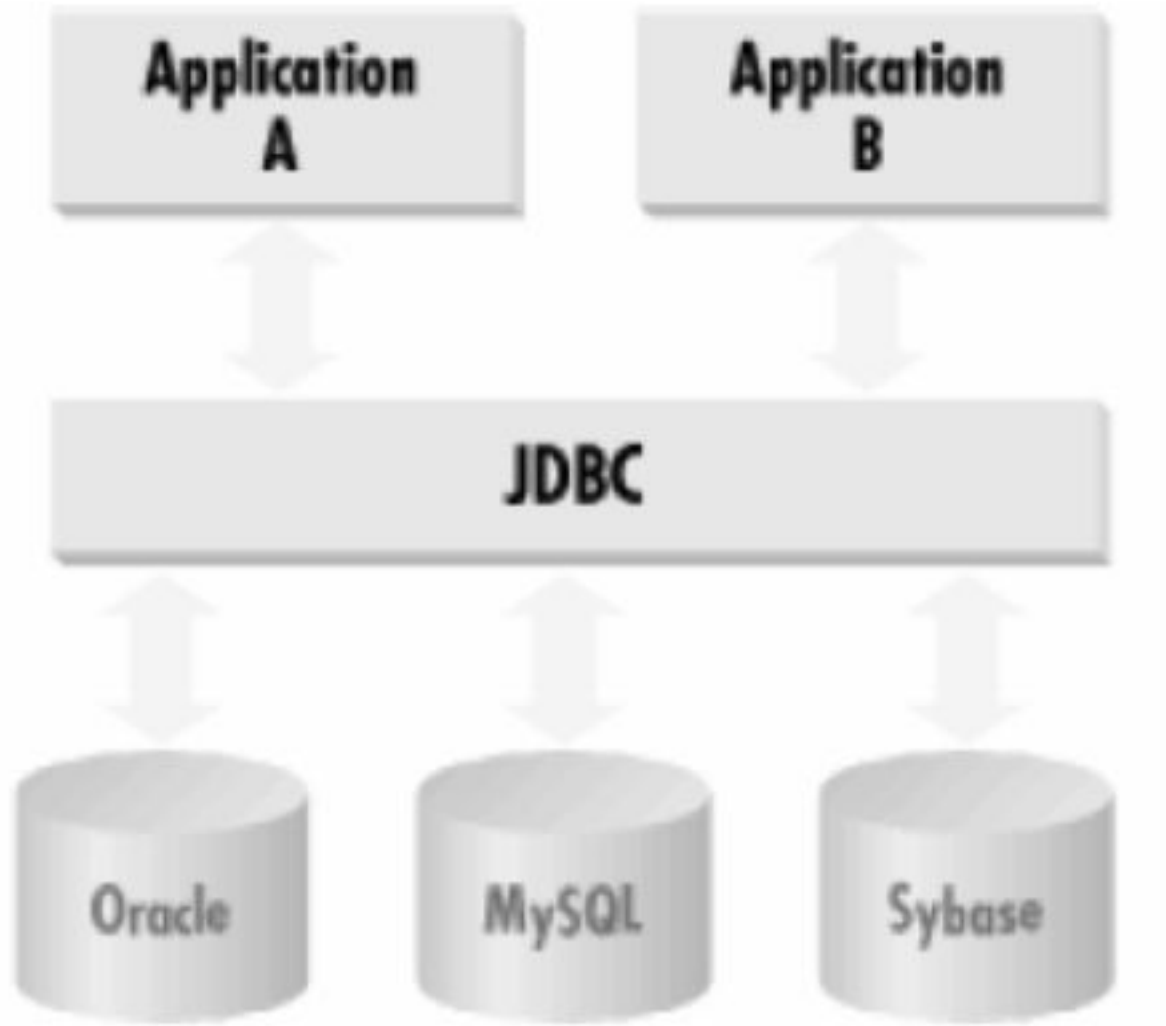
  Suppose you manage a database store on an SQL server. Then you need to have a knowledge on SQL in order to retrieve or update data. This will not be that pleasant for you. Will it?

- An application in between can do the job for you

  The application will deal with the database while you interact with the user-friendly GUI

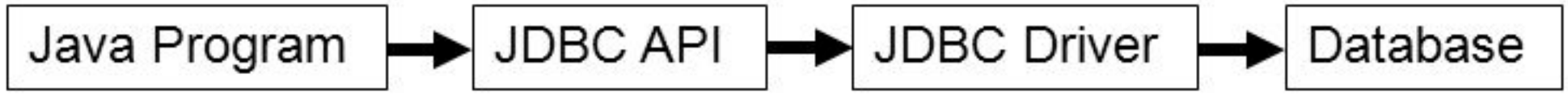
JDBC

- Java Database Connectivity
- Provides a Java API to access SQL databases



The JDBC API

- The JDBC API contains methods to communicate with DBMS or RDBMS
- The JDBC API uses the JDBC driver to carry out it tasks



The process of accessing the database

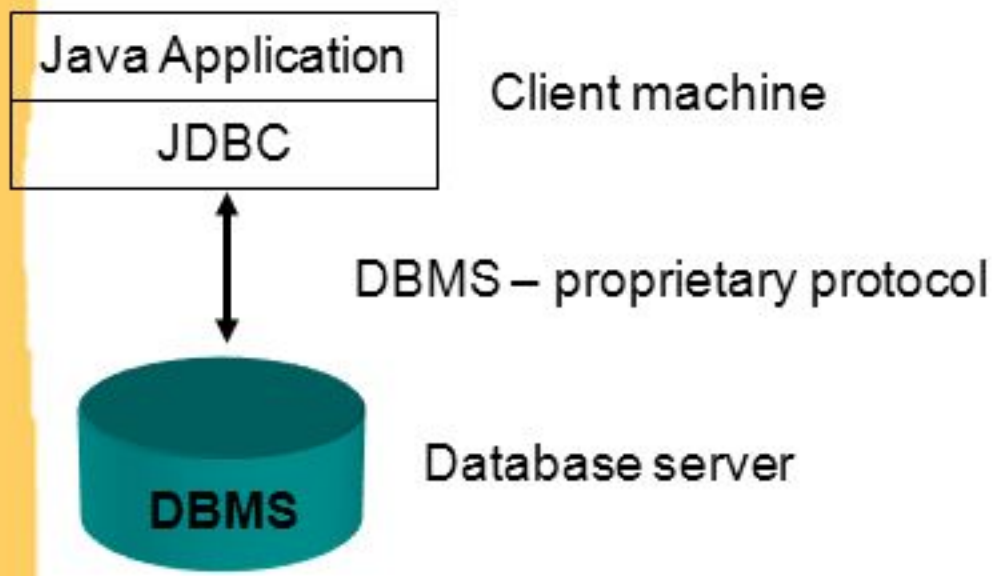
- The JDBC API & Driver enables a Java application to:
  1. Establish a connection with a data source
  2. Send queries and update statements to the data source
  3. Process the results

JDBC API

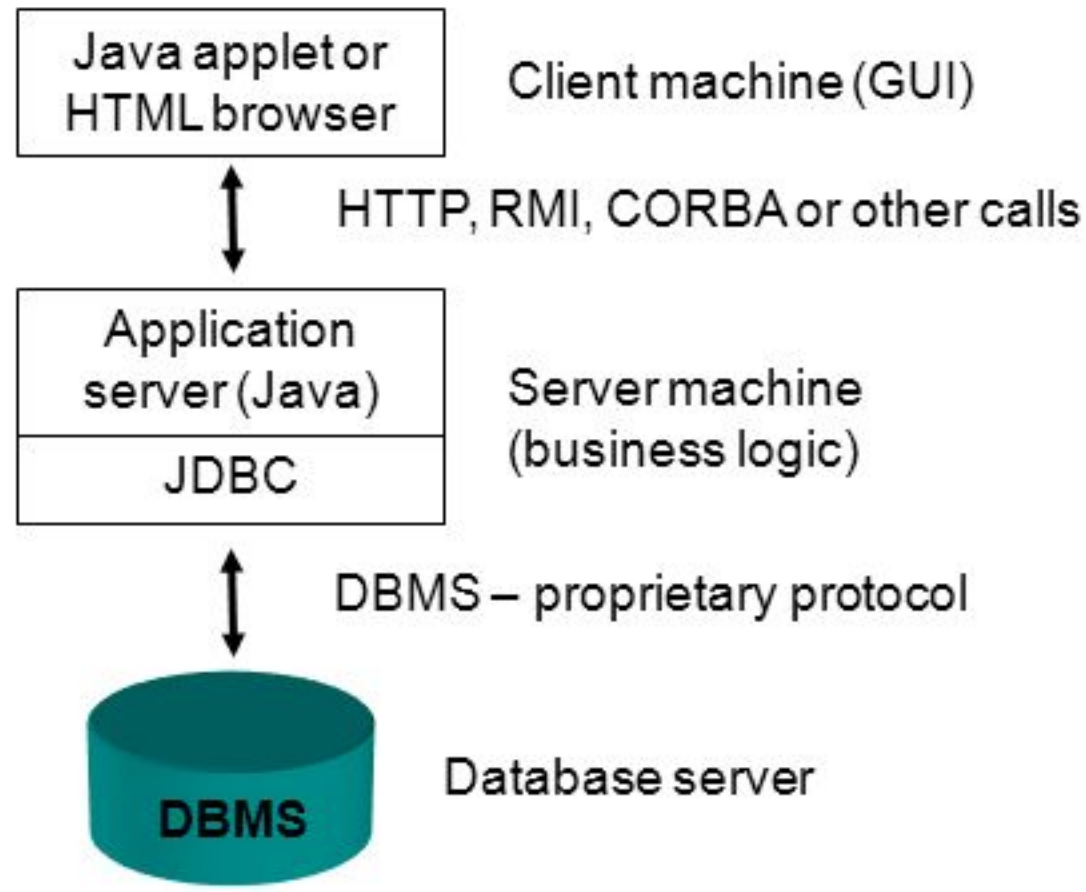
- The JDBC API supports both two-tier and three-tier models for database access.
- Two-tier model -- a Java applet or application interacts directly with the database.
- Three-tier model -- introduces a middle-level server for execution of business logic:
  - the middle tier to maintain control over data access.
  - the user can employ an easy-to-use higher-level API which is translated by the middle tier into the appropriate low-level calls.

The JDBC API

- The JDBC API supports both two-tier and three-tier models



Two-tier model



Three-tier model

JDBC Classes

- java.sql.
  - DriverManager
    - getConnection
  - Connection
    - createStatement
  - Statement
    - execute, executeQuery, executeBatch, executeUpdate
  - ResultSet
    - next, getString, getInt, getDate, getMetaData
  - ResultSetMetadata
    - getColumnCount, getColumnName, getColumnType

JDBC Steps

1. Instantiate proper driver
2. Open connection to database
3. Connect to database
4. Query database (or insert/update/delete)
5. Process the result
6. Close connection to database

1. Instantiate Driver

- Class.forName("driver class")
- Driver class is vendor dependent, e.g.,
  - sun.jdbc.odbc.JdbcOdbcDriver
    - JDBC-ODBC bridge used to access ODBC Sources
  - com.mysql.jdbc.Driver
    - driver to access MySQL database
  - com.sybase.jdbc2.jdbc.SybDriver
    - driver to access Sybase database

2. Open Connection

- DriverManager.getConnection(url)

  or

- DriverManager.getConnection(url, user, pwd) return Connection

- URL is jdbc:<subprotocol>:<subname>
  - e.g., jdbc:odbc:someDB

- jdbc:<subprotocol>://<host>:<port>/<db>
  - jdbc:mysql://localhost:3306/testDB

3. Connect to database

- Load JDBC driver
  - Class.forName("com.mysql.jdbc.Driver").newInstance();

- Make connection
  - Connection conn = DriverManager.getConnection(url);

- URL
  - Format: jdbc:<subprotocol>://<host>:<port>/<db>
  - jdbc:mysql://localhost:3306/testDB

4. Query database

a. Create statement

- Statement stmt = conn.createStatement();
- stmt object sends SQL commands to database
- Methods
  - executeQuery() for SELECT statements
  - executeUpdate() for INSERT, UPDATE, DELETE, statements

b. Send SQL statements

- stmt.executeQuery("SELECT …");
- stmt.executeUpdate("INSERT …");

5. Process results

- Result of a SELECT statement (rows/columns) returned as a ResultSet object
  - ResultSet rs = stmt.executeQuery("SELECT * FROM users");
- Step through each row in the result
  - rs.next()
- Get column values in a row
  - String userid = rs.getString("userid");
  - int type = rs.getInt("type");

users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0
John	John	Smith	pass	1

Print the users table

```
ResultSet rs = stmt.executeQuery("SELECT * FROM users");

while (rs.next()) {
    String userid = rs.getString(1);
    String firstname = rs.getString("firstname");
    String lastname = rs.getString("lastname");
    String password = rs.getString(4);
    int type = rs.getInt("type");
    System.out.println(userid + " " + firstname + " " +
       lastname + " " + password + " " + type);
}
```

users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0
John	John	Smith	pass	1

Add a row to the users table

```
String str = "INSERT INTO users VALUES('Bob', 'Bob', 'King',
                'cat', 0)";

//Returns number of rows in table
int rows = stmt.executeUpdate(str);
```

users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0

6. Closing connections

- Close the ResultSet object
  - rs.close();

- Close the Statement object
  - stmt.close();

- Close the connection
  - conn.close();

```
import java.sql.*;
public class Tester {
   public static void main(String[] args) {
      try {
         // Load JDBC driver
         Class.forName("com.mysql.jdbc.Driver");
         // Make connection
         String url = "jdbc:mysql://localhost:3306/testDB";
         Connection conn = DriverManager.getConnection(url, "root", "rev");
         // Create statement
         Statement stmt = conn.createStatement();
         // Print the users table
         ResultSet rs = stmt.executeQuery("SELECT * FROM users");
         while (rs.next()) {
               ...
         }
         // Cleanup
         rs.close(); stmt.close(); conn.close();
         }
      catch (Exception e) {
         System.out.println("exception " + e);
         }
}
```

Transactions

- Currently every executeUpdate() is "finalized" right away
- Sometimes want to a set of updates to all fail or all succeed
  - E.g. add to Appointments and Bookings tables
  - Treat both inserts as one transaction
- Transaction
  - Used to group several SQL statements together
  - Either all succeed or all fail

Transactions

- Commit
  - Execute all statements as one unit
  - "Finalize" updates
- Rollback
  - Abort transaction
  - All uncommited statements are discarded
  - Revert database to original state

Transactions in JDBC

- Disable auto-commit for the connection
  - conn.setAutoCommit(false);
- Call necessary executeUpdate() statements
- Commit or rollback
  - conn.commit();
  - conn.rollback();

JDBC Driver

- There are four flavors of JDBC

  JDBC-ODBC Bridge (Type 1 driver)

  Native-API/partly-Java driver (Type 2 driver)

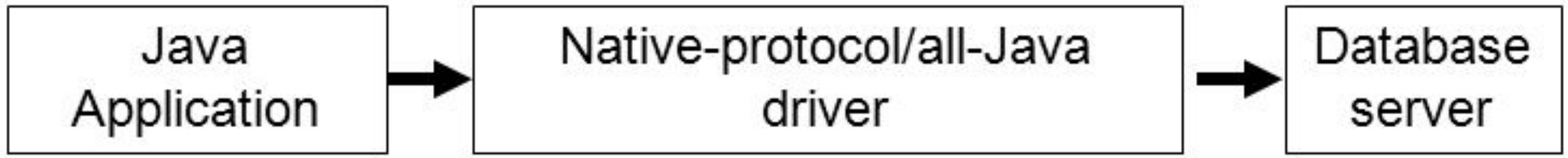
  Net-protocol/all-Java driver (Type 3 driver)

  Native-protocol/all-Java driver (Type 4 driver)

JDBC Driver

- We will use the,

  Native-protocol/all-Java driver



Functionality of Type 4 driver

On The Job

- Required software

  Java-2 JVM s including JDK – 1.4x or newer to compile

  MySQL server version 4.1 or newer

  MySQL Connector/J

Note: *We have used MySQL as the database server and MySQL Connector/J as the database specific JDBC driver*

Example

```
import java.sql.*;
import java.io.*;

class  JDBC
{
      public static void main(String[] args)
      {
            try
            {
                  Connection conn;
                  Statement stmt;
                  String Query;
                  ResultSet rs;

                   Class.forName("com.mysql.jdbc.Driver");
conn=DriverManager.getConnection("jdbc:mysql://localhost/game","root","passwo
rd");
                  stmt=conn.createStatement();
```

```
  Query="select * from table1";
rs=stmt.executeQuery(Query);
while(rs.next())
{               String s = rs.getString(1);
                System.out.println(s);
}
rs.close();
                conn.close();
                }
                catch(SQLException sqle)
                {
                        System.out.println(sqle);
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

Loading the Driver

a) Import statements

```
import java.sql.*;
```

b) Loading the database driver

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

- We load the driver class by calling Class.forName() with the Driver class name as an argument
- If the class name is jdbc.DriverXYZ , you would load the driver with the following line of code: Class.forName("jdbc.DriverXYZ");
- Once loaded, the Driver class creates an instance of itself.