Java DataBase Connectivity (JDBC)
J2EE application model

- J2EE is a multitiered distributed application model
  - client machines
  - the J2EE server machine
  - the database or legacy machines at the back end
JDBC API

- JDBC is an interface which allows Java code to execute SQL statements inside relational databases.
The JDBC-ODBC Bridge

- ODBC (Open Database Connectivity) is a Microsoft standard from the mid 1990’s.

- It is an API that allows C/C++ programs to execute SQL inside databases

- ODBC is supported by many products.
The JDBC-ODBC Bridge (Contd.)

- The JDBC-ODBC bridge allows Java code to use the C/C++ interface of ODBC
  - it means that JDBC can access many different database products

- The layers of translation (Java --> C --> SQL) can slow down execution.
The JDBC-ODBC Bridge (Contd.)

- The JDBC-ODBC bridge comes *free* with the J2SE:
  - called `sun.jdbc.odbc.JdbcOdbcDriver`

- The ODBC driver for Microsoft Access comes with MS Office
  - so it is easy to connect Java and Access
JDBC Pseudo Code

- All JDBC programs do the following:
  - Step 1) load the JDBC driver
  - Step 2) Specify the name and location of the database being used
  - Step 3) Connect to the database with a `Connection` object
  - Step 4) Execute a SQL query using a `Statement` object
  - Step 5) Get the results in a `ResultSet` object
  - Step 6) Finish by closing the `ResultSet`, `Statement` and `Connection` objects
Set up a database server (Oracle, MySQL, pointbase)

Get a JDBC driver
- set CLASSPATH for driver lib
  - Set classpath in windows, control panel->system->advanced->environment variable
  - Set classpath in Solaris, set CLASSPATH to driver jar file

Import the library
- import java.sql.*;

Specify the URL to database server
- String url = "jdbc:pointbase://127.0.0.1/test"

Load the JDBC driver
- Class.forName("com.pointbase.jdbc.jdbcUniversalDriver");

Connect to database server
- Connection con = DriverManager.getConnection(url, "dbUser", "dbPass");

Create SQL Statement
- stmt = con.createStatement();

Execute SQL
- stmt.executeUpdate("insert into COFFEES " + "values('Colombian', 00101, 7.99, 0, 0)");
- ResultSet rs = stmt.executeQuery(query);
import java.sql.*;

public class SqlTest
{
    public static void main(String[] args)
    {
        try
        {
            // Step 1: Make a connection
            // Load the driver
            Class.forName("com.pointbase.jdbc.jdbcUniversalDriver");

            // Get a connection using this driver
            String url = "jdbc:pointbase://localhost/cs595";
            String dbUser = "PBPUBLIC";
            String dbPassword = "PBPUBLIC";

            Connection con = DriverManager.getConnection(url, dbUser, dbPassword);
        }
    }
}
Statement stmt = con.createStatement();
String sql= "select * from Traps";

ResultSet rs = stmt.executeQuery(sql);

String name;
double val;
date = new java.sql.Date();
while (rs.next())
{
    name = rs.getString("TrapName");
    val = rs.getDouble("TrapValue");
    date = rs.getDate("TrapDate");
    System.out.println("name = " + name + " Value = " + val + " Date = " + date);
}

stmt.close();
con.close();

} catch(ClassNotFoundException ex1)
{
    System.out.println(ex1);
}
} catch(SQLException ex2)
{
    System.out.println(ex2);
}
JDBC Diagram

DriverManager creates Connection creates Statement creates ResultSet

Driver makes link to driver

SQL data

data

SQL data
Load Driver

- DriverManager is responsible for establishing the connection to the database through the driver.

- e.g.

  ```java
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection conn = DriverManager.getConnection(url);
  ```
Specify the URL to database server

- The name and location of the database is given as a URL
  - the details of the URL vary depending on the type of database that is being used
Database URL

```
jdbc:pointbase://host.domain.com:9092/data/file
```

- The comms protocol
- The machine holding the database.
- Database port
- The path to the database on the machine

**E.g.**  
```
jdbc:pointbase://localhost/myDB
```
The **Statement object** provides a workspace where SQL queries can be created, executed, and results collected.

**e.g.**

```java
Statement st =
    conn.createStatement();
ResultSet rs = st.executeQuery(
    " select * from Authors" );
sts.close();
```
ResultSet Object

- Stores the results of a SQL query.

- A ResultSet object is similar to a ‘table’ of answers, which can be examined by moving a ‘pointer’ (cursor).
Accessing a ResultSet

- **Cursor operations:**
  - `first()`, `last()`, `next()`, `previous()`, etc.

- **Typical code:**
  ```java
  while( rs.next() ) {
    // process the row;
  }
  ```
Accessing a ResultSet (Contd.)

- The ResultSet class contains many methods for accessing the value of a column of the current row
  - can use the column name or position
  - e.g. get the value in the lastName column:
    ```java
    rs.getString("lastName")
    or rs.getString(2)
    ```
The ‘tricky’ aspect is that the values are SQL data, and so must be converted to Java types/objects.

There are many methods for accessing/converting the data, e.g.

- `getString()`, `getDate()`, `getInt()`, `getFloat()`, `getObject()`
Meta Data

- Meta data is the information *about* the database:
  - e.g. the number of columns, the types of the columns
  - meta data is the *schema* information

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Course</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>007</td>
<td>James Bond</td>
<td>Shooting</td>
<td>99</td>
</tr>
<tr>
<td>008</td>
<td>Aj. Andrew</td>
<td>Kung Fu</td>
<td>1</td>
</tr>
</tbody>
</table>
Accessing Meta Data

- The `getMetaData()` method can be used on a `ResultSet` object to create its metadata object.

  e.g.

  ```java
  ResultSetMetaData md = rs.getMetaData();
  ```
Using Meta Data

```java
int numCols = md.getColumnCount();

for (int i = 0; i <= numCols; i++) {
    if (md.getColumnType(i) == Types.CHAR) {
        System.out.println(md.getColumnName(i));
    }
}
```
Connection pooling is a technique that was pioneered by database vendors to allow multiple clients to share a cached set of connection objects that provide access to a database resource.

Connection pools minimize the opening and closing of connections.
JDBC in J2EE

- Step 1: Start Sun Application Server PE 8
- Step 2: Start PointBase
- Step 3: Use J2EE admin to create connection pool
- Step 4: Use J2EE admin to create JDBC data source
- Step 5: import java.sql.*;
- Step 6: get Context
- Step 7: look up data source with JNDI
- Step 8: Execute SQL and process result
Start Application Server & PointBase
Create Connection Pool Using Admin GUI
Create Data Source Using Admin GUI

[Image of Sun Java System Application Server Admin Console]

- Resources
  - JDBC
    - JNDI Name: TrapDB
    - Pool Name: PointBasePool
    - Description: Use the Connection Pools page to create new pools
    - Status: Enabled
Example: JDBC Using JNDI & Connection Pools

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
import java.io.*;
import java.util.*;
import java.util.*;

public class SqlServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException
    {
        res.setContentType("text/plain");
```
Example: JDBC Using JNDI & Connection Pools (Contd.)

```java
try {
    PrintWriter pw = res.getWriter();
    String dbName = "java:comp/env/jdbc/TrapDB";
    InitialContext ic = new InitialContext();
    DataSource ds = (DataSource) ic.lookup(dbName);
    Connection con = ds.getConnection();
    Statement stmt = con.createStatement();
    String sql = "select * from Traps";
    ResultSet rs = stmt.executeQuery(sql);
    String name;
    double val;
    java.sql.Date date;
    while (rs.next()) {
        name = rs.getString("TrapName");
        val = rs.getDouble("TrapValue");
        date = rs.getDate("TrapDate");
        pw.println("name = " + name + " Value = " + val + " Date = " + date);
    }
}
```
stmt.close();

}  
catch(SQLException ex2)  
{
    System.out.println(ex2);
}
catch(IOException ex3)  
{
    System.out.println(ex3);
}
catch(Exception ex4)  
{
    System.out.println(ex4);
}
Reference

- Database and Enterprise Web Application Development in J2EE, Xiachuan Yi, Computer Science Department, University of Georgia.