Everything you wanted to know about JDBC

Presented to the Calgary Oracle Users Group

October 23, 2003

by

Davis Swan

Davis@SQLMagic.com

Safe Harbour Statement

- Everything you hear here today may be inaccurate, incomplete, or just plain wrong
- This talk will NOT make you an expert in JDBC
- This talk will NOT get you a better job
- Yes, I am a contractor working for Shell, and yes, I do use JDBC often in that role
- If you screw something up at work and try to blame it on what you heard at this talk, you will be sorry
- Because my wife *is* a lawyer

Who am I?

- Started life in Calgary as a Geophysicist with Gulf Oil in 1978 but couldn't find oil - was moved in software development
- First database programming with S2K and ISAM in the early 1980's
- Got involved with database Web applications when we built an AFE/Invoice system with Server Side Javascript in 1997 with Burntsand Systems.
- Started Java/JDBC programming in 2000 and have written about 40,000 lines of code since then.

Who are you?

- Database Administrators
- Developers
- People that have written PL/SQL programs
  - Bonus - OO facilities such as VARRAY or TABLE objects
- People that have written Java programs
- People that have used JDBC
- People that only came for the cinnamon buns

Life before JDBC

- Before JDBC there was ODBC
  - Open DataBase Connectivity developed by Microsoft in the early 1990's
  - Primarily a client-server, file based system used to access local databases (MS Access, dBase, Excel files)
- Before there was ODBC there was OCI
  - First widely used call-level interface. Was used as the model for both ODBC and JDBC
- Before there was OCI all database access applications used precompilers
  - limited memory structures in 3GL
  - performance issues required separation of SQL syntax parsing and statement execution

JDBC - YAA *(yet another acronym)*

- What is JDBC
  - The acronym stands for Java DataBase Connectivity
  - It is a specification and an interface, not a piece of software
  - Implementations of the specification, referred to as JDBC drivers, are developed by 3rd parties, not Sun Microsystems
  - Multiple sources for the software produce inconsistent implementations of the specification
  - Many JDBC drivers are not free

JDBC - Raison d'être

- To provide database connectivity to Java programs in a standardized, rational fashion
- Can connect to multiple databases simultaneously, including different versions and different RDBMS
- Designed to be networked - connections are made to a Universal Resource Locator (URL)
- Supports the "write once, run anywhere" paradigm which is the main advantage of Java programs
- Uses native syntax of the underlying RDBMS (in effect, uses dynamic SQL all the time)

JDBC *vs.* ODBC

- ODBC Drivers are typically available only for Wintel platforms
- ODBC drivers are installed as Windows applications using the MS ODBC control panel
  - DLL conflicts, registry issues, administrator privileges
  - Every client machine must be touched (JDBC jar files can be stored on a network drive)
- ODBC methods not as consistent as JDBC
- JDBC-ODBC bridge - works well if clients are preconfigured and you cannot get a native JDBC driver

JDBC - The Good

- Write Once, Run Anywhere - the same physical file can be copied onto a different system and executes immediately
- Does not require the installation of any client software*
  - SQL*Net not required. As a result, JDBC enabled software can access Oracle from platforms not supported by Oracle
- Has full and direct access to the file system of the OS, so that files can be read and written to*
- JDBC enabled applets can be embedded within a web browser*
- Throws lots of exceptions making code easy to debug

*Requires a JRE, JDBC applets can only read files from the HTTP server,applet security will prevent network connections to database servers unless they are also running the web server, applications will not execute during power blackouts

JDBC - the Bad

- Driver registration
  - allows duplicate classes and methods by “registering” the driver
  - done slightly differently with JDBC drivers from different vendors
- Some drivers (Oracle in particular), require the presence of the physical driver file in order to compile!
- Does not insulate the developer from SQL syntax differences
- Can be slow if you keep opening and closing connections
- Throws lots of exceptions, which gets to be a real pain
  - Remember to try { try { try

```
    try
    {
/*
 *      Try and register the JDBC driver using several methods.
*/
      if ( dbType.equals("ORACLE") )
      {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      } else {
        Class.forName(tryDriverString);
      }
    } catch (Exception e ) {
      try
      {
        Class.forName(tryDriverString).newInstance();
      } catch (Exception e2 ) {
        System.out.println("Unable to register driver\n");
        return false;
      }
    }
```

JDBC - the Ugly

- The Java CLASSPATH - can get pretty hairy when you're implementing software that supports multiple RDBMS
- Set in different ways on different platforms.
- Set in different ways depending upon how the program is launched (command line, desktop icon)
- Mapping of variables to database column types can be tricky (compared to PL/SQL)
- MetaData - inconsistent implementation of classes and methods by developers of JDBC drivers.

JDBC - Setting the CLASSPATH

- Windows
  - In a Command Prompt Window type
    SET CLASSPATH=path1;path2;archive1.jar
    OR set a CLASSPATH variable in your SYSTEM Control Panel

- Unix using ksh
  - SET CLASSPATH=path1:path2:archive1.jar
    EXPORT CLASSPATH

- MacIntosh OS 9 - must be set with a GUI

- Java Applet
  - use the ARCHIVE=path1,path2,archive1.jar directive within the APPLET TAG

JDBC - Metadata

- DataBaseMetaData
  - 48 Fields, 117 Method
  - 23 methods return ResultSet Objects
  - The result is that no JDBC Driver developer provides complete implementations, no way of knowing which features have been implemented
- ResultSetMetaData
  - 3 Fields, 21 methods all of which return primitives
  - Generally more consistent implementations, but there are exceptions (e.g. getTableName produces NULL in Oracle)

JDBC - building “client side” Web pages

- This involves building a Java applet which uses JDBC to query and update a database directly.
- The applet is embedded in a web browser page.
- Applets are a good choice where connection speeds or server response are slow.
- By default a Java applet is very restricted in the types of activities it can do.
- The only simple choice in some cases (e.g. graphics)

JDBC - Java Applet Restrictions

- The default security model for Java applets prevents the applet from engaging in “dangerous” activities
  - applets can only read files from the Web server they are connected to at the time the applet starts execution
  - applets can only read ASCII files because the actual transfer of information takes place via HTTP.
  - applets cannot write files
  - applets cannot make a TCP/IP connection to a second server. Therefore, if your web server and your Oracle server are two different machines, applets will not work.

JDBC - Relaxing the applet security model

- If you have no control over client environments (for example, when dealing with retail clients), then
  - use “signed” Java applets - secure digital certificates can be purchased from signing authorities
  - the type of digital signature varies depending upon the browser and version.
  - The user will be prompted for permission when “dangerous” activities are attempted by the applet.
- When you can control (or at least influence) client configurations, then
  - install the Java plugin from Sun Microsystems
  - add a .security.policy file to each client machine

JDBC - Avoiding applet security issues

- Put a Web Server on your Oracle server
  - can be a special purpose, low usage server used only for database applets
- Make any files that you need to read visible from the Web server
- Give up on applets and switch your application to the server

JDBC - Building “server side” applications

- Users will have to endure server delay for almost every operation
- Developers can sometimes make things more tolerable by retrieving additional data and filtering on the client using Javascript functions
- Several robust technologies available
  - JSP - Java Server pages can use JDBC to combine database content into dynamic web pages
  - CGI - a very simple approach that can produce effective applications.

JDBC - RDBMS Portability

- The ability to create truly RDBMS independent applications is complicated by lack of standards in SQL syntax or “dialect”
- Column types are not standardized
  - VARCHAR2, TEXT, INT are examples of non-portable types
  - over-loading of XOPEN types
  - BLOB data access methods not standardized
- FUNCTIONS are not standardized
  - NVL vs. NULLIF
  - SYSDATE vs SYSDATE()
- String concatenation
  - || vs. CONCAT function

JDBC - the future

- Tied to the future of Java
  - Client side applets may not survive
  - MS trying to eliminate support for applets in IE
  - Sun Microsystems in financial difficulty
  - J2EE doing well, but .NET is a powerful competitor
- Supported by increasing interest in Open Source
  - JDBC drivers exist for all Open Source RDBMS
  - Free availability of Java makes it the language of choice for Open Source Developers
  - Major investments being made in Java by IBM, HP, Sony, Apple

JDBC - Conclusions

- JDBC provides fully functional database access which is truly independent of the hardware platform, OS, and RDBMS
- JDBC can be used in a client application or applet, or in server applications through the use of JSP or CGI
- Web browser applets which use JDBC are subject to security restrictions which make them difficult to implement
- JDBC does not insulate the developer from differences in SQL syntax.

Case Studies in Building Dynamic Web Pages using JDBC

Web pages made easy

- Primitive but effective approach is to put a tiny shell script in cgi-bin which launches PL/SQL or Java application
- Script hands off Web form parameters as a URL encoded string - parsing code for both PL/SL and Java available at SQLMagic.com
- PL/SQL or Java application generates HTML output including form parameters which allow the script to call itself.
- No need to learn perl or php - relatively easy to manage source and maintain code.

Web pages made easier

- Using Java gives developer access to both databases and OS resources such as E-mail, files and directories, process status
- Java is a very robust and well written programming language, with excellent error trapping/debugging
- Using Java will insulate you from a particular RBDMS
- Enterprise class systems can be built using Open Source web servers and database engines - no S/W licensing
- Java portability makes migration between hardware/OS platforms a non-issue (Linux, Mac, PSX?)

An example HTML file

(this will launch the simple CGI program shown on the next slide)

```
<FORM ACTION="http://161.19.112.1/cgi-bin/web_demo.cgi" METHOD="POST">
First Name: <INPUT TYPE="TEXT" NAME="FIRST_NAME" SIZE=20><BR>
Last Name: <INPUT TYPE="TEXT" NAME="LAST_NAME" SIZE=20><BR>
Email Address: <INPUT TYPE="TEXT" NAME="LAST_NAME" SIZE=20><BR>
<INPUT TYPE="SUBMIT" NAME="FUNCTION" VALUE="Update Information">
</FORM>
```

The Magical CGI (web_demo.cgi)

```
#!d:/zsh/zsh.exe
echo "Content-type: text/html\n\n"
read form_parm_input
form_parms=${form_parm_input##pattern=}
CLASSPATH=d:\java
export CLASSPATH
java WebJavaDemo "$form_parms"
exit
```

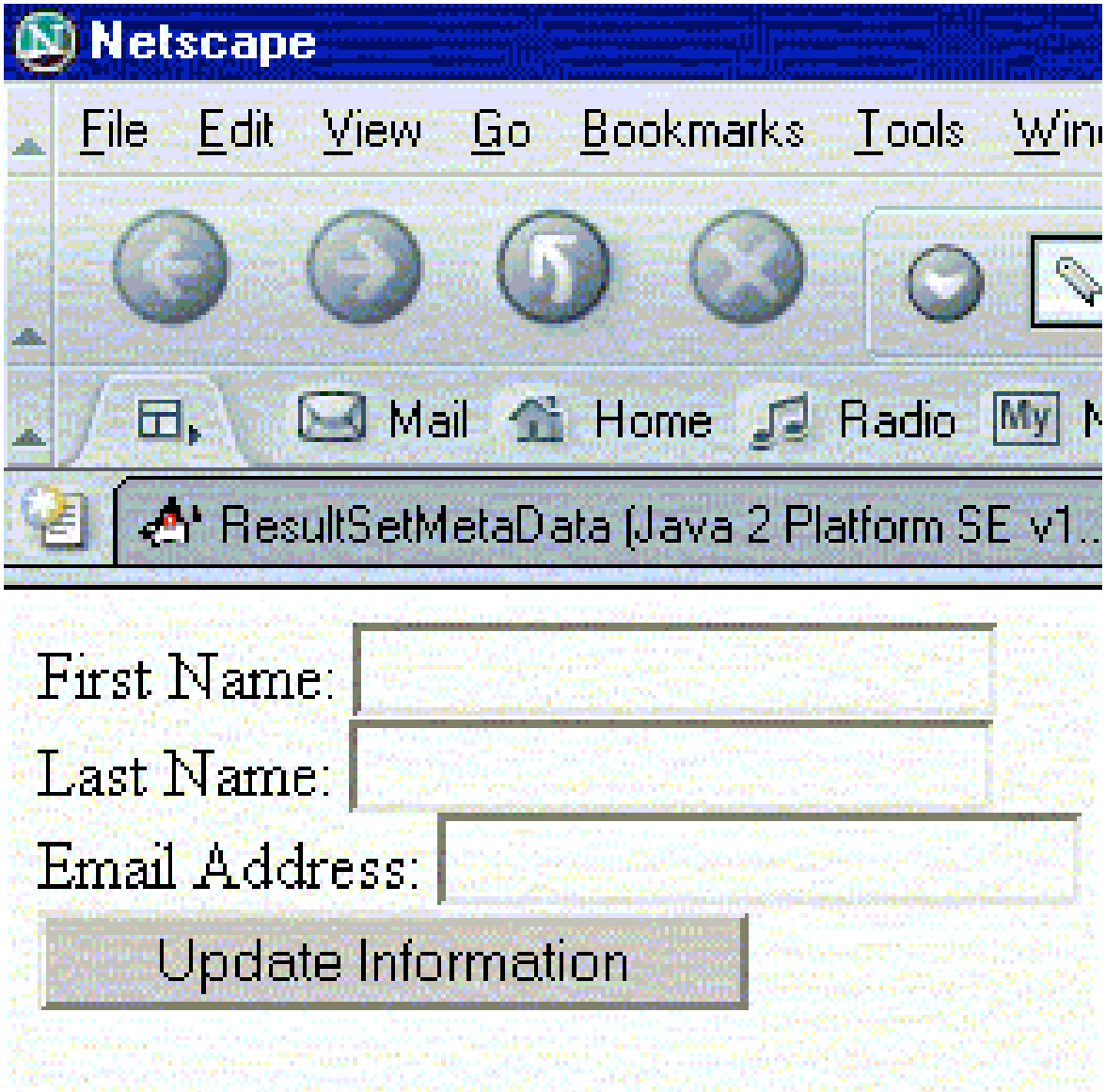
Example Java Program

```
import java.util.*;
public class WebJavaDemo
{
  public static void main(String[] args) throws Exception
  {
    StringTokenizer st;
    String firstName=(String)null,lastName=(String)null,
    email=(String)null,appFunction=(String)null,field,paramName;
    StringBuffer outstr = new StringBuffer(100);
    if ( args.length == 1 )
    {
      st = new StringTokenizer(args[0],"&");
      while (st.hasMoreTokens())
      {
        field = st.nextToken();
        paramName = field.substring(0,field.indexOf("="));
        if ( paramName.equals("FIRST_NAME") )
          firstName = field.substring(field.indexOf("=")+1);
```

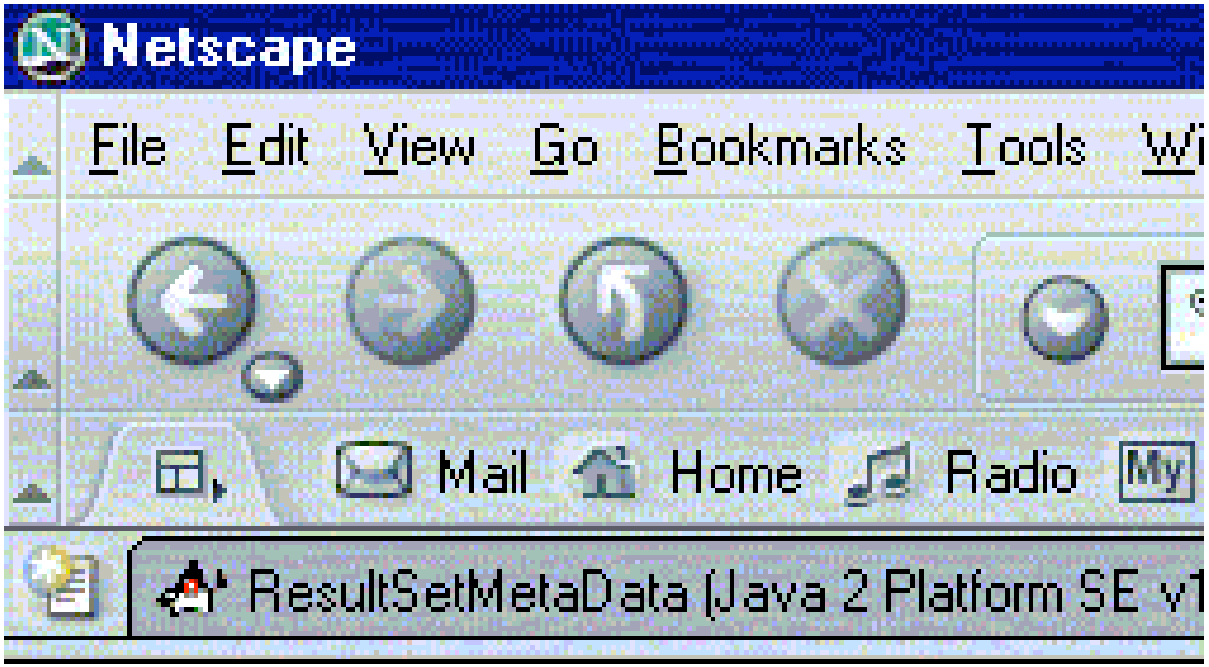
Example Java Program …*continued*

```
            else if ( paramName.equals("LAST_NAME") )
               lastName = field.substring(field.indexOf("=")+1);
            else if ( paramName.equals("EMAIL_ADDRESS") )
               email = field.substring(field.indexOf("=")+1);
            else if ( paramName.equals("FUNCTION") )
               appFunction = field.substring(field.indexOf("=")+1);
         }
         outstr.append("<TABLE><TR><TD>Name:<TD>" + firstName + " "
         + lastName + "<TR><TD>Email Address:<TD>" + email
         + "<TR><TD>Function:<TD>" + appFunction + "</TABLE>");
         System.out.println(outstr.toString());
      }
   }
}
```

Input Form


Netscape
File Edit View Go Bookmarks Tools Win
Mail Home Radio My
ResultSetMetaData [Java 2 Platform SE v1..
First Name:
Last Name:
Email Address:
Update Information

Output


Netscape
File Edit View Go Bookmarks Tools Wi
Mail Home Radio My
ResultSetMetaData (Java 2 Platform SE v1
Name: Davis Swan
Email Address: Davis@sqlmagic.com
Function: Update Information

The Magical CGI Using PL/SQL

(plus80.exe is the Personal Oracle version of SQL*Plus)

```
#!d:/zsh/zsh.exe
echo "Content-type: text/html\n\n"
read form_parm_input
form_parms=${form_parm_input##pattern=}
plus80.exe -SILENT budget/budget @web_demo "$form_parms"
exit
```

An example PL/SQL script

```
DECLARE
first_name VARCHAR2(20);
last_name VARCHAR2(20);
app_function VARCHAR2(40);
email VARCHAR2(60);
outstr VARCHAR2(255);
form_parms VARCHAR2(255);
BEGIN
   form_parms := '@1';
   first_name := URL_PARM(form_parms,'FIRST_NAME');
   last_name := URL_PARM(form_parms,'LAST_NAME');
   app_function := URL_PARM(form_parms,'FUNCTION');
   email := URL_PARM(form_parms,'EMAIL_ADDRESS');
   outstr := '<TABLE><TR><TD>Name:<TD>'||first_name||' '||last_name;
   outstr := outstr||'<TR><TD>Email Address:<TD>'||email;
   outstr := outstr||'<TR><TD>Function:<TD>'||app_function||'</TABLE>';
   DBMS_OUTPUT.PUT_LINE(outstr);
END;
```

JDBC - Demos using SuperSQL

- Simultaneous connection to two different databases.
- Connection to remote Oracle instance
- Reading OS files (start command)
- Writing OS files (spool command)
- SQL dialect translation