Training Resources

Making Oracle and SQLJ Work For You

Presented to:
RMOUG Training Days 2003

John King

King Training Resources

6341 South Williams Street

Littleton, CO 80121-2627 USA

www.kingtraining.com

800.252.0652 or 303.798.5727

 Copyright @ 2003, John Jay King

Training Resources

Session Objectives

- Understand how to use SQLJ to access Oracle data
- Know the different types of Oracle connections supported
- Be able to query data from Oracle using SQLJ
- Use an Oracle stored procedure via SQLJ

Copyright @ 2003, John Jay King

Major Keywords

- Connection
- Statement
- Result Set
- Thin Client
- OCI Client

Copyright @ 2003, John Jay King

Training Resources

SQLJ Process

SQLJ Source
(.sqlj suffix)

SQLJ
Translator

Serialized
Profile
(.ser suffix)

Java Source
(.java suffix)

Java
Compiler

Bytecode
(.class suffix)

Java
Package
Libraries

.class files
.jar files
.war files
.ear files
.zip files

Java Virtual Machine
(JVM)

Copyright @ 2003, John Jay King

Oracle and Java

- Oracle9i and Oracle8i provide a Java Virtual Machine (JVM) built in to the database
- Oracle provides JDBC drivers to allow Java to use the database
- Oracle provides an SQLJ translator allowing the use of embedded SQL in Java
- JDeveloper is an Integrated Development Environment (IDE) that may be used to create Java programs using JDBC or SQLJ
- Common Object Request Broker Architecture (CORBA) support
- Oracle Application Server (Oracle9i AS)
- Support for JavaBeans and EJB

Copyright @ 2003, John Jay King

Training Resources

Database Connectivity with JDBC

- JDBC is a tool used to access SQL databases from Java
- Oracle JDBC drivers support JDBC 1.22 fully
- All of JDBC 2.0 and parts of JDBC 3.0 (the current release) are supported by Oracle9i JDBC or by Oracle extensions
- JDBC allows programmers to:
  - Connect to a database
  - Query and Update a database
  - Perform PL/SQL and call stored Procedures/Functions
- JDBC programs are database vendor and platform independent (mostly) unless Oracle-specific features are used

Copyright @ 2003, John Jay King

Training Resources

JDBC Drivers

- Oracle support three types of JDBC drivers:
  - Oracle Thin JDBC driver
    - Does not require Oracle client
    - Creates its own Oracle Net/Net8/SQL*Net connection
    - Downloadable (about 900k)
    - Used for Applets
  - Oracle OCI JDBC driver
    - Requires Oracle Client
    - Uses existing Oracle Net/Net8/SQL*Net connection
    - Used for Applications on client and middle-tier applications
  - Oracle Server JDBC driver (aka KPRB driver)
    - Server-side only
    - Used for server-side JDBC, Java stored procedures, and Enterprise Java Beans (EJBs)
    - Supports communication between PL/SQL and Java

Copyright @ 2003, John Jay King

Which Driver for Me?

- Use the Thin JDBC driver for:
  - Applets
  - Most applications
  - TCP/IP connections only
- Use the OCI JDBC driver for:
  - Applications requiring the best performance?
  - Connections not using TCP/IP
- Use the Server-side JDBC driver for:
  - Accessing other databases from within the database (requires Oracle 8.1.7 or later)
- Use the Server-side Internal JDBC driver for:
  - Programs running in the server, accessing the server (requires Oracle 8.1.5 or later)

Copyright @ 2003, John Jay King

What Database Version?

- Thin JDBC drivers and OCI JDBC drivers work with:
  - Oracle 9.2.x
  - Oracle 9.0.x
  - Oracle 8.1.x
  - Oracle 8.0.x (no support for objects)
  - Oracle 7.x (no support for objects or LOBs)
- Oracle Server JDBC Internal drivers became available with Oracle 8.1.5 but can access data in Oracle 8.1.4
- Oracle Server-side Thin driver became available in 8.1.7

Copyright @ 2003, John Jay King

Oracle JDBC Features

- Oracle's JDBC drivers add several features to the standard Java JDBC
  - JDBC 1.22 compliant, supports most JDBC 2.0 and parts of JDBC 3.0
  - Supports object-relational data
  - Supports LOB data
  - Provides Oracle-specific performance features
  - Allows use of PL/SQL and Java stored procedures
  - Supports all Oracle character sets
- Once a connection has been established, JDBC works the same regardless of driver being used

Copyright @ 2003, John Jay King

JDBC Drivers

- JDBC drivers for Oracle must be available for compilation and testing, they come in two files:
  - classes111.zip   Java 1.1
  - classes12.zip   Java 1.2, 1.3, 1.4
  - ojdbc14.jar   Java 1.4
  - Two additional sets of drivers may be added for installations using Oracle NLS features:
  - nls_charset11.zip   Java 1.1.x NLS characters
  - nsl_charset12.zip   Java 1.2, 1.3, 1.4 NLS characters
- Make sure that ONLY ONE SET of these zip/jar files is in your CLASSPATH
- Zip files may be found in directory:

  <ORACLE_HOME>/jdbc/lib

Copyright @ 2003, John Jay King

King Training Resources

Using JDBC

1. Import java.sql.* and other needed packages
2. Load/register the JDBC driver
3. Connect to the database using JDBC
4. Create a statement object
5. Execute SQL statements and process results
6. Close the result set
   (technically not required, but safest to avoid memory leak issues)
7. Close the statement
8. Close the connection
   (disconnect from database)

Copyright @ 2003, John Jay King

King Training Resources

Load/register JDBC driver

- This is one of the two parts of JDBC that is most impacted by switching database vendors
- Make sure that the JDBC driver has been made available to the JVM (in the classpath)
- Make sure the most current driver is used
- Connect using one of two mechanisms:

```
DriverManager.registerDriver(
    new oracle.jdbc.driver.OracleDriver());
```

or

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```


Copyright @ 2003, John Jay King

Training Resources

Connect to the Database

- Connections are the other part of JDBC that is vendor-dependent, Oracle’s connection string varies depending upon the driver in use
- A DBA's (DataBase Administrator's) help may be required to properly format the connect string and identify the appropriate settings
- It is a bad idea to hard-code userid and password information in code as shown in these examples, be sure to prompt the current user for the necessary information
- Be sure to close the connection, this is code that is often placed in a finally block

Copyright @ 2003, John Jay King

King Training Resources

Connection Object

- Thin client connection
  - jdbc:oracle:thin:@**hostcomputer**:**port**:**oraclesid**

```
String url = "jdbc:oracle:thin:@tecra:1521:tec817";
String uID = "scott";          // prompt user for this
String uPswd = "tiger";        // prompt user for this
Connection conn =
      DriverManager.getConnection(url,uID,uPswd);
```

- OCI client connection
  - jdbc:oracle:oci8:@**oraclesid**

```
String url = "jdbc:oracle:oci8:@tec817";
String uID = "scott";          // prompt user for this
String uPswd = "tiger";        // prompt user for this
Connection conn =
      DriverManager.getConnection(url,uId,uPswd);
```


Copyright @ 2003, John Jay King

Training Resources

Internal Connection

- Internal database connections used by Java Stored Procedures are simpler:

```
Conn = DriverManager.getConnection("jdbc:default:connection:");
```


Copyright @ 2003, John Jay King

Connection Pools

- Creating and destroying database connections is one of the most expensive operations JDBC performs
- Performance may sometimes be improved dramatically by using “Connection Pools”
- Connection Pools represent a set of existing database connections that application programs share, eliminating the cost of create/destroy each time a program touches the database
- Creating and managing Connection Pools is often the province of the DBA staff, though, applications may create them manually

Copyright @ 2003, John Jay King

Training Resources

More on Connection Pools

- With Connection Pools the Application Server opens a given number of Connections to the database (dependent upon system configuration) and Java programs share them via ConnectionPool
- Caching the Connection objects provides significant performance improvements
- Connection Pools take advantage of JDBC 2.0's JNDI (Java Naming and Directory Interface) technology using ConnectionPoolDataSource, DataSource, and PooledConnection interfaces

Copyright @ 2003, John Jay King

Training Resources

Using Connection Pools

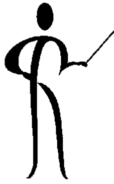
- As with normal Connections, Connection Pool implementations tend to be vendor-specific; many people find it useful to use a “Connection Factory” as part of framework to make the database vendor transparent to the Java programs
- The use of a “Connection Factory” encapsulates (hides) the complexity of the pool connection and looks to the JDBC/SQLJ developer like just another database connection
- A common mistake made by developers is failing to close() the pooled connection, closing the connection makes the pooled connection available for reuse (close() is polymorphic!)

Copyright @ 2003, John Jay King

Connect Using Pool

```
try {
    // Retrieve the DataSource using logical JNDI name
    ctx = new InitialContext(env);
    ds = (DataSource) ctx.lookup("jdbc/mysample");
}
catch (NamingException eName) {
    System.out.println("Error in lookup jdbc/mysample");
    System.exit(0);
}
try {
   conn = ds.getConnection(); // NOT DriverManager
   // ** database access code goes here
}
catch (SQLException e) {
   // SQL exception code
}
```

Create a Statement Object



- The Statement object contains the SQL statement to be executed
- For maximum portability use generic SQL
- For maximum performance/flexibility, it might be necessary to use Oracle-specific SQL and OracleStatement object
- Be sure to close the Statement object, this is code often found in a finally block

Copyright @ 2003, John Jay King

Statement Object

- Standard JDBC Statement:

```
Statement stmt = conn.createStatement ();
```

- Oracle-specific Statement:

```
OracleStatement stmt = conn.createStatement ();
```


Copyright @ 2003, John Jay King

PreparedStatement Object

- Standard JDBC Statements:

```
PreparedStatement stmt =
      conn.prepareStatement("update emp set sal = ? "
                          +  " where empno = ? ");
stmt.setBigDecimal(1,1234.56);
stmt.setInt(2,7788);
nbrRows = stmt.executeUpdate();
```

- Oracle-specific Statements:

```
PreparedStatement stmt =
      conn.prepareStatement("update emp set sal = ? "
                          +  " where empno = ? ");
((OraclePreparedStatement)stmt).setDouble(1,1234.56);
((OraclePreparedStatement)stmt).setInt(2,7788);
nbrRows = stmt.executeUpdate();
```


Copyright @ 2003, John Jay King

Result Set

- Immediately after a query, the result set pointer points one record before the first row returned, use the next method to get the first record
- It is not normally necessary to manually close the Result Set, it will be closed automatically when the Statement it is associated with is closed or re-executed
- Due to memory-leak issues, it may be safest to close the Result Set

Copyright @ 2003, John Jay King

Execute SQL Statements

- To execute a query:

```
String mySql = "SELECT ENAME,JOB,DEPTNO, "
             + " to_char(HIREDATE,'MM/DD/YYYY') HIREDATE "
             + " FROM EMP "
             + " WHERE DEPTNO = '" + myParm + "'"
             + " ORDER BY ENAME";
ResultSet rs = stmt.executeQuery(mySQL);
```

- To execute an update or stored procedure call:

```
String mySQL = "UPDATE EMP SET SAL = SAL * 1.06 "
                    + " WHERE ROWID = '" + saveRowId + "'";
int    nbrRows = stmt.executeUpdate(mySql);
```


Copyright @ 2003, John Jay King

Training Resources

ROWID

- When updating or deleting rows previously displayed, be sure to store and use the Oracle ROWID for the column
- ROWID is the fastest possible access to a given row
- Don't show the ROWID to the user… (pretty scary looking…)
- Be careful! Oracle will reuse rowids when rows are deleted and new rows are inserted

(Probably best not to use rowid if Updates, Deletes, and Inserts are all possible)

Copyright @ 2003, John Jay King

King Training Resources

Process Result Sets

- Data is processed a row at time in the ResultSet using the next method:

```
while (rs.next()) { /** process SQL row **/ }
```

- Retrieve values one column at a time by name, or, by position in the SQL statement:

```
String myEmpno = rs.getString("EMPNO");
```

```
String myEmpno = rs.getString(1);
```


Copyright @ 2003, John Jay King

King Training Resources

Result Set Processing

- Some other methods available for result set processing include:

Method	Description
– afterLast	Moves cursor to the end of ResultSet
– beforeFirst	Moves cursor to the front of ResultSet
– clearWarnings	Clears all warnings on ResultSet
– close	Releases ResultSet object's database and JDBC resources immediately
– deleteRow	Deletes the current row
– first	Moves cursor to first row in ResultSet
– getWarnings	Returns the first warning reported
– insertRow	Inserts a row into this ResultSet
– last	Moves cursor to last row in ResultSet
– next	Moves the cursor down one row
– previous	Moves the cursor up one row

Copyright @ 2003, John Jay King

Result Set Testing

- Some ResultSet methods useful for testing:
  - rowDeleted
  - rowInserted
  - rowUpdated
  - wasNull
- Test wasNull for ***each*** column value that might be null

```
myEmpno = rs.getInt("EMPNO");
if (rs.wasNull())
   myEmpno = 0000;
myEname = rs.getString("ENAME");
if (rs.wasNull())
   myEname = " ";
```


Copyright @ 2003, John Jay King

Result Set Values

- Some standard methods used to process column values by name or by position:
  - getBigDecimal
  - getBlob
  - getBoolean
  - getByte
  - getClob
  - getDate
  - getDouble
  - getFloat
  - getInt
  - getLong
  - getObject
  - getShort
  - getStatement
  - getString

Copyright @ 2003, John Jay King

Training Resources

OracleResultSet Extensions

- Oracle extensions if using OracleResultSet instead of ResultSet:
  - getArray
  - getBfile
  - getBlob
  - getClob
  - getNumber
  - getOracleObject
  - getRaw
  - getRef
  - getRowid
  - getStruct

Copyright @ 2003, John Jay King

JDBC Cleanup

- It is a good idea to put the Statement close and Connection close in a finally block

```
finally {
    try {
        rs.close();         // Close result set
        stmt.close();       // Close statement
        conn.close();       // Close connection
    }
    catch (SQLException e) {
        // code for error closing things
    }
}
// More precise to have try-catch for
// each "close"
```


Copyright @ 2003, John Jay King

Batch Processing

- Sometimes a program will be doing a large number of Insert, Update, or Delete statements
- Normally, JDBC performs an automatic commit after each SQL statement’s execution
- This can be controlled using methods from the Connection object

Copyright @ 2003, John Jay King

Training Resources

Batch Processing Syntax

- To turn off autocommit

```
conn.setAutoCommit(false);
```

- To add commands to the batch

```
stmt.addBatch(insertUpdateDeleteStmt);
```

- To execute a batch-style command

```
int [] updateCounts = stmt.executeBatch();
```

- To manually commit or rollback

```
conn.rollback(); // conn.commit();
```


Copyright @ 2003, John Jay King

Using Stored Procedures

- Using Stored Procedures is a simply a matter of knowing the procedure/function name, parameters, and return type (for functions)
- CallableStatement objects or OracleCallableStatement objects are used instead of the normal Statement objects
- Input parameters are set using setString, setArray, setAsciiStream, setBigDecimal, setBinaryStream, setBlob, setBoolean, setByte, setBytes, setCharacterStream, setClob, setDate, setDate, setDouble, setFloat, setInt, setLong, setNull, setObject, setRef, setShort, setString, setTime, setTimestamp, and setUnicodeStream
- Output parameters are defined using RegisterOutParameter
- Statement executeUpdate() is used
- Get methods are used to retrieve output values

Copyright @ 2003, John Jay King

Calling a Procedure

- Note use of substitution/bind variables (OK with any SQL!)
- Variables are referenced via relative position in statement

```
 // Following is Oracle-specific
CallableStatement stmt1
   = conn.prepareCall("begin addem(?,?,?); end;");
// Portable Stored Procedure call follows
// CallableStatement stmt1
//    = conn.prepareCall("{CALL addem(?,?,?)}");
 stmt1.setString(1,"123");
 stmt1.setString(2,"456");
 stmt1.registerOutParameter(3,Types.VARCHAR);
 stmt1.executeUpdate();
 System.out.println("Value returned is "
   + stmt1.getString(3));
```


Copyright @ 2003, John Jay King

Calling a Function

```
CallableStatement stmt2
  = conn.prepareCall("begin ? := times_2(?); end;");
stmt2.registerOutParameter(1,Types.NUMERIC);
stmt2.setString(2,"1234.56");
stmt2.executeUpdate();
java.math.BigDecimal outVal = stmt2.getBigDecimal(1);
System.out.println("Value returned is " + outVal);
```


Copyright @ 2003, John Jay King

Training Resources

CallableStatement Methods

- getBigDecimal
- getBoolean
- getByte
- getBytes
- getDate
- getDouble
- getFloat
- getInt
- getLong
- getObject
- getShort
- getString
- getTime
- getTimeStamp
- registerOutParameter
- wasNull

Copyright @ 2003, John Jay King

OracleCallableStatement Extensions

- getArray
- getBfile
- getBlob
- getClob
- getCursor
- getCustomDatum
- getNumber
- getOracleObject
- getRaw
- getRef
- getRowid
- getStruct

Copyright @ 2003, John Jay King

King Training Resources

Good, Better, Best?

- JDBC's autocommit (the default) may be a poor performer when many SQL statements are involved
  - Controlling commit/rollback manually will probably improve performance
  - If many Insert/Update/Delete operations are involved, Batched SQL might improve performance more
  - If many Insert/Update/Delete operations are involved, a Stored Procedure might work best
  - If a statement is executed more than once, a Prepared Statement with bind variables is usually better (bind variables)

Copyright @ 2003, John Jay King

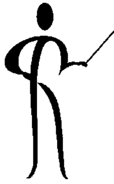
Training Resources

What is SQLJ?

- SQLJ is "Pro*Java" in appearance and functionality
  - Relatively normal SQL statements are coded
  - A pre-processor converts the SQL to two files:
    - xxx.sqlj SQLJ source code
    - xxx.ser SQLJ profile
  - Tends to be a better performer for "static" SQL
- SQLJ is mentioned here because it is a viable alternative to JDBC in many cases

Copyright @ 2003, John Jay King

What Does SQLJ Look Like?



- SQLJ clauses begin with a pound (#) sign and include standard looking SQL statements
- The statement below inserts a row into the EMP table passing three column values

```
#sql {INSERT INTO EMP (EMPNO, LASTNAME, SALARY)
      VALUES( :empId, :lastName, :empPay) };
```

  - Remember how complex JDBC calls with replaceable values were?
  - SQLJ is far simpler to code
  - Interestingly, most Java programmers seem to prefer the JDBC code

Copyright @ 2003, John Jay King

Training Resources

Oracle Documentation

- Oracle9i/8i CORBA Developer's Guide and Reference
- Oracle9i/8i Enterprise JavaBeans Developer's Guide and Reference
- Oracle9i/8i Java Developer's Guide
- Oracle9i/8i Java Stored Procedures Developer's Guide
- Oracle9i/8i Java Tools Reference
- Oracle9i/8i JDBC Developer's Guide and Reference
- Oracle9i/8i JPublisher User's Guide
- Oracle9i/8i Oracle Servlet Engine Release Notes
- Oracle9i/8i Oracle Servlet Engine User's Guide
- Oracle9i/8i SQLJ Developer's Guide and Reference
- Oracle9i/8i Supplied Java Packages Reference
- Oracle JavaServer Pages Developer's Guide and Reference
- Javadoc for Oracle JDBC: <oraclehome>/jdbc/doc/javadoc.zip
- Lots of papers and examples: http://technet.oracle.com

Copyright @ 2003, John Jay King

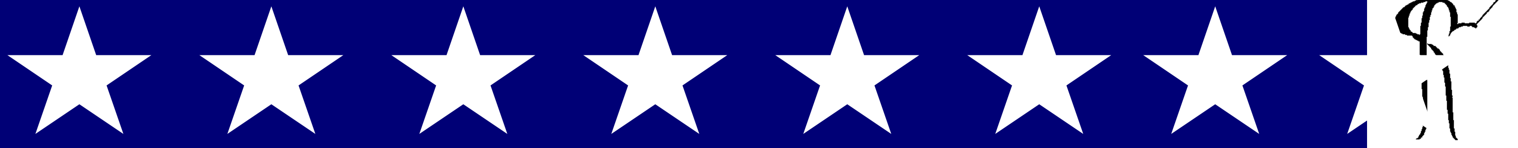
Training Resources

Wrapping it all Up

- JDBC access allows full use of Oracle from Java programs
- Oracle-specific features improve performance at the cost of portability
- Review of the steps:

1. Load/register the JDBC driver
2. Connect to the database
3. Create a statement object
4. Execute SQL statements and process results
5. Close the result set (if used), statement, and connection
   (probably from a finally block)

Copyright @ 2003, John Jay King



Mark your calendars for the

Spring 2003 Atlantic Oracle Training Conference!

May 8–9, 2003

At the new
Washington Convention Center
Washington, D.C.

To contact the author:

John King

King Training Resources

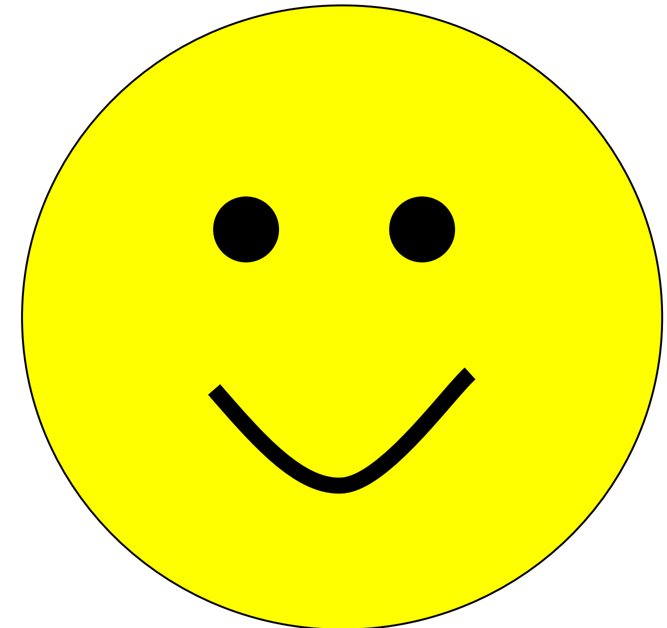
6341 South Williams Street

Littleton, CO 80121-2627 USA

1.800.252.0652 - 1.303.798.5727

Email: john@kingtraining.com

Paper & Sample Code: www.kingtraining.com



Thanks for your attention!

Copyright @ 2003, John Jay King