JDBC and Oracle Rdb

Wolfgang Kobarg-Sachsse

Oracle Support Services – Oracle Rdb Support

ORACLE

Overview

- What is JDBC ?
- About JAVA
- Oracle JDBC Driver
- Native JDBC Driver for Rdb
  - ...

ORACLE

What is JDBC

- JDBC defines a Java based API allowing applications to connect/query/update relational databases
- Consider it the “ODBC“ for Java
- Allows you to:
  - Make a connection to relational databases
  - Send SQL statements to the relational database system
  - Receive results back from the relational database

ORACLE

About Java

Java 2 Software Development Kit:

Actual version: 1.4.2_02

Location: http://java.sun.com/j2se/1.4/

Windows, *Linux* and *Solaris* only

For *OpenVMS* the location is here:

http://h18012.www1.hp.com/java/download/index.html

(The actual version on OpenVMS is 1.4.2-1)

ORACLE

Using Oracle‘s JDBC Driver

Requirements:

- JDK (or JRE)
- Oracle JDBC Driver (which one?)
- SQL/Services
- Rdb database must be prepared for
  SQL*Net for Rdb

Please read Metalink note #198416.1

ORACLE

Which Oracle JDBC Driver?

Oracle9i Release 2 (9.2.0.3 and 9.2.0.1)

Oracle9i Release 1 (9.0.1.4 and 9.0.1)

Oracle8i Release 2 (8.1.7)

Location: OTN, of course
http://technet.oracle.com/software/tech/java/sqlj_jdbc/content.html

ORACLE

Disadvantage - Conclusion

The OCI layer is an overhead which slows down performance.

Conclusion for Oracle was to develop a Native JDBC Driver for Rdb, which has been released in May 2003.

Since then about 30 different customers opened more than 110 Service Requests concerning the JDBC Driver for Rdb.

ORACLE

Native JDBC Driver for Rdb

- Requirements
- Two derivates
- Installation
- Connection and two simple examples
- Troubleshooting (Tracing)
- Thin Server Options and Controlling
- Handle Blobs (example)
- Pool Server
- Outstanding problems

ORACLE

Requirements

- On the server:
  - OpenVMS for ALPHA minimum version V7.2
  - JAVA (JDK) for OpenVMS system minimum version V1.2.2
  - Oracle Rdb release 7.1.0.4 as a minimum.
- On the client:
  - JAVA (the Java versions on the server and on the client may be different)

ORACLE

Two Derivates

In reality, the Native JDBC Driver for Rdb consists of **two** different drivers:

- Oracle Rdb Native Driver:

  The Oracle Rdb native driver is a Type II driver for use with client-server Java applications.

  This driver runs only on OpenVMS and is not suitable for applets.

- Oracle Rdb Thin Driver

  The Oracle Rdb thin driver is a 100 percent pure Java, Type IV driver.

  Because it is written entirely in Java, this driver is platform-independent. It does not require any additional Oracle software on the client side.

ORACLE

Installation (1)

The actual version is called 7.1.2.1.

The Oracle Rdb JDBC Driver kit is downloadable from Metalink (Patchset 3510504). The current filename is:

ORCL-VMS-RDBJDBC71-V0701-2B3BM-1.PCSI

The Rdb Native JDBC Drivers kit uses Polycenter to simplify the installation of the product.

$ product install rdbjdbc71/source=MY_DIR

The following product has been selected:

ORCL VMS RDBJDBC71 V7.1-2B3BM Layered Product

ORACLE

Installation (2)

The installation procedure will copy all the kit files to the appropriate Rdb JDBC product directory within sys$common:[rdb$jdbc] directory:

Directory SYS$COMMON:[RDB$JDBC.0701-21B432]

RDBJDBCCHECKUP.CLASS;1 RDBJDBCCHECKUP.JAVA;1
RDBJDBCSHR71.EXE;1 RDBJDBC_FAQ.HTML;1 RDBJDBC_FAQ.TXT;1
RDBJDBC_RELNOTES.HTML;1 RDBJDBC_RELNOTES.TXT;1
RDBNATIVE.JAR;1 RDBTHIN.JAR;1 RDBTHINCONTROL.JAR;1
RDBTHINSRV.JAR;1 RDBTHINSRVPOOL.JAR;1

Total of 12 files.

ORACLE

Installation (3)

The installation kit is comprised of the following files:

RDBJDBCCHECKUP.CLASS	Used to verify the installation of the kit
RDBJDBCCHECKUP.JAVA	Used to verify the installation of the kit
RDBJDBCSHR71.EXE	Shared image required for Oracle Rdb database access
RDBNATIVE.JAR	Java Jar file containing the classes for the Oracle Rdb Native driver
RDBTHIN.JAR	*Java Jar file containing the classes for the Oracle Rdb Thin driver*
RDBTHINSRV.JAR	Java Jar file containing the classes for the Oracle Rdb Thin server
RDBTHINSRVPOOL.JAR	Java Jar file containing the classes for the Oracle Rdb Thin pool server
RDBTHINCONTROL.JAR	Java Jar file containing the classes for the Oracle Rdb Thin controller
RDBJDBC_FAQ.*	Frequently asked questions
RDBJDBC_RELNOTES.*	Driver release notes

ORACLE

Installation (4)

Important: All *.JAR files have to reside on the server, except the RDBTHIN.JAR file.

This file must be copied to the client. It is the client part of the Rdb JDBC Thin Driver.

Exception: Server and client are identical, but there is no practical use of this. The Rdb JDBC Native Driver is faster than the Rdb JDBC Thin Driver.

ORACLE

Connection(1)

Preparation:

Define the system logical name RDBJDBCSHR to point to the shared image RDBJDBCSHR71.EXE.

$ define/system RDBJDBCSHR –

SYS$COMMON:[RDB$JDBC.0701-2B3BM]RDBJDBCSHR71.EXE

Include the rdbnative.jar file in your Java CLASSPATH by using either the logical name JAVA$CLASSPATH or the -classpath option on the Java command line.

$ define MY_CLASSES SYS$COMMON:[RDB$JDBC.0701-2B3BM]

$ define JAVA$CLASSPATH [],MY_CLASSES:RDBNATIVE.JAR

(If you plan to use the Rdb JDBC Thin Driver on your Alpha, then include also the rdbthin.jar file in your classpath definition.)

ORACLE

Connection(2)

If you want to use the Rdb JDBC Thin Driver, then you need these two additional steps:

Start Java on the server:

$ @sys$startup:JAVA$nnn_SETUP.COM

where nnn is the version of Java that is installed on your Alpha

Start the Thin Server process:

$ SPAWN/NOWAIT/PROC=RDB_THIN java -jar MY_CLASSES:rdbthinsrv.jar

For a production environment it is more comfortable to start the Thin Server process as a detached process.

ORACLE

Connection – Example (1)

The kit contains the RDBJDBCCHECKUP class. You can use it to test your connection. The Release Notes (included in the kit) describe how to do it.

Here is a simpler example, using the Rdb JDBC Thin Driver.

Therefore we need the file RDBTHIN.JAR on a client (which is a Windows PC in this example). Let the file be in D:\RDBJDBC. We further need the classpath on the PC. Via the advanced system properties you can add and modify the environment variables.

ORACLE

Connection – Example (2)

Here is the example code:

```
import java.sql.*;
public class RdbExample {
   public static void main(String args[])
    { try
      {
      Class.forName ("oracle.rdb.jdbc.rdbThin.Driver");
         String URL = "jdbc:rdbThin://mynodename:1701/path_to_my_database";
         String Username = "myusername";
         String Password = "mypassword";
         Connection MyConn = DriverManager.getConnection(URL,Username,Password);
         MyConn.close();
     } catch (Exception e)
      {e.printStackTrace();}
  } }
```

ORACLE

Connection – Example (3)

Here is another example code:

```
import java.sql.*;
public class RdbExample2 {
   public static void main(String args[])
    { try
      {
    Class.forName ("oracle.rdb.jdbc.rdbThin.Driver");
    String URL = "jdbc:rdbThin://mynodename:1701/path_to_my_database";
    String Username = "myusername";
    String Password = "mypassword";
    Connection MyConn = DriverManager.getConnection(URL,Username,Password);
        // Get metadata information
        DatabaseMetaData dbmd = MyConn.getMetaData();
        System.out.println("DB Name = " + dbmd.getDatabaseProductName());
        System.out.println("DB Version = " + dbmd.getDatabaseProductVersion());
        System.out.println("JDBC driver version is " + dbmd.getDriverVersion());
...
```

ORACLE

Connection – Example (4)

```
        // Create a new table
        Statement stmt = MyConn.createStatement() ;
        stmt.executeUpdate( "Create table MYTABLE " +
                    "  ( COL1 char(3) " +
                    "  , COL2 integer)" );
        System.out.println("Table MYTABLE created.");

        // Simple insert
        stmt.executeUpdate("Insert into MYTABLE values ('AAA',1)");
        System.out.println("First record inserted.");
...
```

ORACLE

Connection – Example (5)

```
// Different insert
ResultSet rs = stmt.executeQuery("Select * from MYTABLE");
rs.moveToInsertRow();
rs.updateString("COL1", "CCC");
rs.updateInt("COL2",3);
rs.insertRow();
rs.updateString("COL1", "BBB");
rs.updateInt("COL2",2);
rs.insertRow();
rs.close();
System.out.println("The following two records inserted.");
...
```

ORACLE

Connection – Example (6)

```
System.out.println("Now the sorted result will follow.");
// Show the result
rs = stmt.executeQuery("select col1,col2 " +
  "from MYTABLE order by col2");
rs.next();
do {
   System.out.println(rs.getString(1) +  "   " + rs.getInt(2));
   } while(rs.next());
rs.close();
...
```

ORACLE

Connection – Example (7)

```
            // Drop the table
            stmt.executeUpdate( "Drop table MYTABLE");
            System.out.println("Table MYTABLE dropped.");

            stmt.close();
            MyConn.close();
        } catch (Exception e)
        {
           e.printStackTrace();
        }
    }
}
```

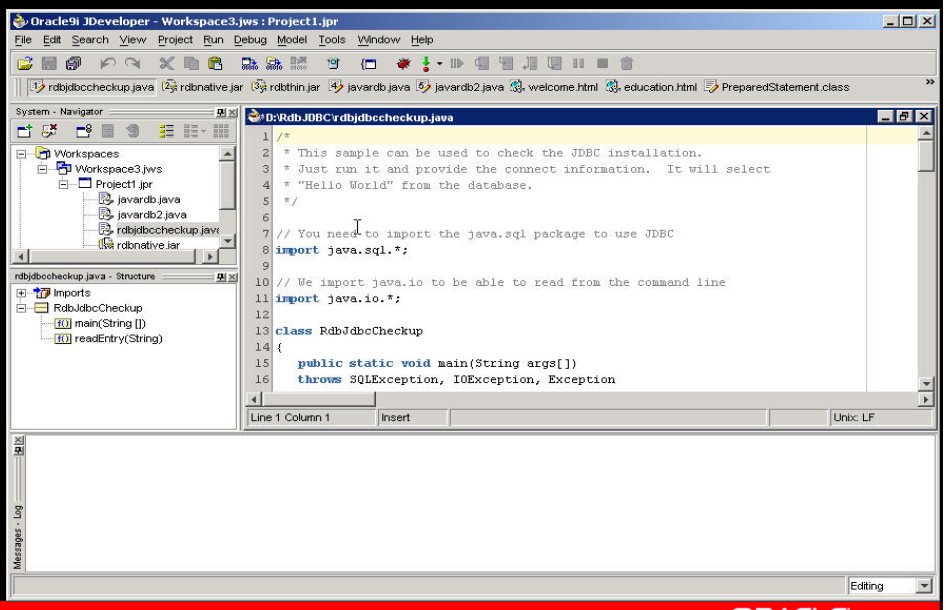
ORACLE

Connection – Example (8)

```
D:\Java\Examples>javac RdbExample2.java
D:\Java\Examples>java -cp .;"d:\rdbjdbc\rdbthin.jar" RdbExample2
DB Name = Oracle Rdb
DB Version = V7.1-231
JDBC driver version is V7.1-210
Table MYTABLE created.
First record inserted.
The following two records inserted.
Now the sorted result will follow.
AAA   1
BBB   2
CCC   3
Table MYTABLE dropped.
```

ORACLE

JDeveloper: Just a cool tool to make the life easier...



Troubleshooting

At first:

How to find out which version of the Rdb JDBC Driver is installed?

```
$ pipe anal/image RDBJDBCSHR | -
    search sys$input "image file ident"/win=(0,4)
```

Output (for example):

```
        image file identification: "RDBJDBC V7.1-21"
        image file build identification: ""
        link date/time:  2-MAR-2004 18:39:49.64
        linker identification: "A11-50"
```

ORACLE

Troubleshooting – Tracing (1)

Trace provides tracing of method calls within the Native JDBC drivers.

Tracing can be activated at start time of the Thin Server process, e.g.:

```
$ SPAWN/NOWAIT/PROC=RDB_THIN java -jar -
      MY_CLASSES:rdbthinsrv.jar -tracelevel –1
```

If the output should be written into a file, then the /OUTPUT=filename option should be specified with the SPAWN command (resp. if the Thin Server process is started as a detached job).

ORACLE

Troubleshooting – Tracing (2)

The value –1 is just an example:

Bit	Hexadecimal Value	Decimal Value	Traces
0	0x00000001	1	standard JDBC methods
1	0x00000002	2	standard JDBC class create/finalize
2	0x00000004	4	SQL statements
4	0x00000010	16	non-standard JDBC methods
5	0x00000020	32	non-standard JDBC class create/finalize
6	0x00000040	64	garbage collection
8	0x00000100	256	Rdb JNI calls
9	0x00000200	512	network sends
10	0x00000400	1024	server actions
29	0x20000000	536870912	memory information
30	0x40000000	1073741824	provides more details on certain flags
(ALL)	0xFFFFFFFF	-1	trace everything

ORACLE

Troubleshooting – Tracing (3)

But what to do if the Rdb JDBC Native Driver is used because then the Thin Server process must not be started?

Then tracing is an option at the URL:

```
String URL = "jdbc:rdbNative:path_to_my_database@tracelevel=-1";
String Username = "myusername";
String Password = "mypassword";
Connection MyConn = DriverManager.getConnection(URL,Username,Password);
```

ORACLE

Troubleshooting – Tracing (4)

The trace output can be rather long and is not always easy to interpret, but one fact is easy to check:

```
> main ThinConnect@3.setTraceLevel msg : rdbNativeInstance=20030508
> main ThinConnect@3.setTraceLevel msg : rdbServerInstance=20030508
```

If these two dates are not identical, then this is most likely the reason for your trouble.

ORACLE

Thin Server Options

We have already seen that we can add an option when we start the Thin Server process (-tracelevel –1). There are more:

```
$ java -jar rdbthinsrv.jar
        [-anonymous]
        [-b -buffersize <send_buf_size>]
        [-bypass]
        [-cfg -configfile <configuration_filename>]
        [-controlpass <control_password>]
        [-fs -fetchsize <default_fetch_size>]
        [-lockwait <lock_wait>]
        [-maxclients <maximum_number_of_clients>]
        [-maxtries <maximum_number_of_lock_attempts>]
        [-p -port <port_num>]
        [-pw -password <default_user_password>]
        [-tl -tracelevel <trace_level>]
        [-u -user <default_user_name>]
```

ORACLE

Thin Server Controller (1)

You remember this file:

RDBTHINCONTROL.JAR Java Jar file containing the classes for the Oracle Rdb Thin controller

The Rdb JDBC Thin Server controller allows basic management of Rdb JDBC Thin Servers. The Rdb JDBC Thin Server must be started with the controlpass option:

```
$ SPAWN/NOWAIT/PROC=RDB_THIN java -jar –
      MY_CLASSES:rdbthinsrv.jar -controlpass mypassword
```

Then the Rdb JDBC Thin Server controller can be started:

```
$ java -jar my_classes:rdbthincontrol.jar
```

ORACLE

Thin Server Controller (2)

Example:

```
$ java -jar my_classes:rdbthincontrol.jar
rdbthincontrol> connect //localhost:1701/ myusername mypassword
rdbthincontrol> show server
RDB$NODE                : localhost
RDB$PORT                : 1701
RDB$STATUS              : Available
RDB$SERVER_TYPE         : RdbThinSrv
RDB$SERVER_VERSION      : V7.1-210 20040303 B433
RDB$SERVER_SHR_VERSION : V7.1-210 20040303 B433
RDB$SERVER_PID          : 0x202015f4
RDB$ALLOWS_ANON         : false
RDB$ALLOWS_BYPASS       : false
RDB$NUMBER_OF_CLIENTS   : 1
RDB$MAX_CLIENTS         : 0
rdbthincontrol>
```

ORACLE

Thin Server Controller (3)

Possible options for the Rdb JDBC Thin Server controller:

```
$ java -jar rdbthincontrol.jar
      [-cfg -configfile <configuration_filename>]
      [-fs -fetchsize <fetch_size>]
      [-n -node <node>]
      [-oem]
      [-p -port <port_num>]
      [-pw -password <password>]
      [-showclients]
      [-showserver]
      [-srvargs <server_arguments>]
      [-stopclient <client_id>]
      [-stopserver]
      [-tl -tracelevel <trace_level>]
      [-u -user <user_name>]
```

ORACLE

Handle Blobs (1)

Insert a text blob:

```
String blobVal;
...
        // Insert the row and add the BLOB data
        // Prepare and execute the insert statement
        pstmt = conn.prepareStatement(
          "insert into BLOBTABLE ( KEYCOL, BLOBCOL ) " +
          "  values ( ?, ? ) " );
        pstmt.setString( 1, keyVal );
        InputStream bs = new ByteArrayInputStream(blobVal.getBytes());
        pstmt.setBinaryStream( 2, bs,blobVal.length() );
        pstmt.execute();
        pstmt.close();
```

(From Metalink note 254991.1 to handle text blobs.)

ORACLE

Handle Blobs (2)

Retrieve a text blob:

```
        ResultSet rs = stmt.executeQuery("select * from blobtable");
        while (rs.next())
        {
         oracle.rdb.jdbc.common.Blob bl = (oracle.rdb.jdbc.common.Blob)rs.getBlob(2);
         bl.setSegSeparator("\n");
         byte[] bytes = bl.getBytes(1,9999);
         String st1 = new String(bytes);
         System.out.println(rs.getString(1)+"\n" + st1);
         }
```

(From Metalink note 254991.1 to handle text blobs.)

The new public method setSegSeparator has been added to oracle.jdbc.rdb.common.Blob to enable limited formatting of data returned from Oracle Rdb segmented strings.

ORACLE

Handle Blobs (3)

Insert a binary blob:

```
FileInputStream in_upd = new FileInputStream("Myphoto_in.jpg" );
keyVal = "1";
pstmt = conn.prepareStatement(
  "insert into blobtable (KEYCOL,BLOBCOL) values (?,?)");
pstmt.setString( 1, keyVal );
pstmt.setBinaryStream(2, (InputStream)in_upd, in_upd.available());
in_upd.close();
pstmt.execute();
pstmt.close();
```

ORACLE

Handle Blobs (4)

Retrieve a binary blob:

```
ResultSet rs = stmt.executeQuery("select * from blobtable");
while (rs.next())
{
 oracle.rdb.jdbc.common.Blob bl = (oracle.rdb.jdbc.common.Blob)rs.getBlob(2);
 FileOutputStream out_file = new FileOutputStream("Myphoto_out.jpg" );
 byte[] bytes = bl.getBytes(1,9999999);
 out_file.write(bytes);
 out_file.close();
 }
```

ORACLE

Pool Server (1)

You remember this file:

RDBTHINSRVPOOL.JAR Java Jar file containing the classes for the Oracle Rdb Thin pool server

The Oracle Rdb thin pool server provides server-side redirection to Oracle Rdb Thin Servers.

The Oracle Rdb thin pool server enables load balancing of several Oracle Rdb Thin Servers.

Using the pool server you can designate a single PORT id which can be used by your client side applications to connect to the next available thin server. The pool server selects an available thin server from a table of candidate servers in a round-robin fashion.

ORACLE

Pool Server (2)

Possible options for the Rdb JDBC Thin Server controller:

```
$ java -jar rdbthinsrvpool.jar
        [-cfg -configfile <configuration_filename>]
        [-controlpass <control_password>]
        [-node<n> <node>]
        [-poolserver]
        [-poolsize <pool_size>]
        [-p -port <port_num>]
        [-port<n> <port_num>]
        [-tl -tracelevel <trace_level>]
```

As the number of arguments can be quite long the easiest way is to use configuration files.

ORACLE

Pool Server (3)

Example:

Start two Thin Server processes:

$ spawn/nowait/proc=Rdb_Thin_1 java -jar my_classes:rdbthinsrv.jar -port 1701

%DCL-S-SPAWNED, process RDB_THIN_1 spawned

$ spawn/nowait/proc=Rdb_Thin_2 java -jar my_classes:rdbthinsrv.jar -port 1710

%DCL-S-SPAWNED, process RDB_THIN_2 spawned

ORACLE

Pool Server (4)

Create a configuration file (e.g. myconfig.cfg):

port=1702

poolserver=true

poolsize=2

node1=mynode

port1=1701

node2=mynode

port2=1710

And start the Oracle Rdb thin pool server:

$ spawn/nowait/proc=rdb_thin_pool java -jar my_classes:rdbthinsrvpool.jar –

–cfg myconfig.cfg

ORACLE

Pool Server (5)

Remark:

Do not confuse the Rdb JDBC Pool Server with Connection Pooling.

Sun‘s connection pooling is included in Java‘s extension class javax.sql.

Connection pooling has been written as part of a new JAR rdbext.jar that will contain a number of the classes required for javax.sql (JDBC Extensions), but it is still in the process of being tested and it will be released in a future version of the driver.

ORACLE

Outstanding Problems (1)

Multi-Threading:

Java developers are often thinking in multiple threads.

But it is a fact that Rdb is not multithreaded !

That means that multiple threads in Java do work – but they are handled sequentially.

Rdb Development is looking at ways of providing multiple low overhead small footprint executors, so we can have what would appear to be multithreading at the database level.

ORACLE

Outstanding Problems (2)

SQL/Services:

The advantage of the Native Rdb JDBC Driver is that it does not need the SQL/Services (as using the Oracle JDBC drivers with Rdb is doing).

For some customers this is a disadvantage because of two reasons:
- They prefer relying on the stability and customization possibilities to manage services, and they would like to do that also for JDBC.
- Java installation on the Server is essential to use the Native Rdb JDBC driver. Using SQL/Services can avoid to have Java installed.

There is an enhancement request under investigation to use the SQL/Services together with the Native Rdb JDBC driver.

ORACLE

Summary

- Short introduction to Oracle JDBC drivers
- More deep introduction to the new Native Rdb JDBC Drivers
  - Installation
  - Examples (Connection, Blobs)
  - Troubleshooting
  - Tools (Thin Server Controlling, Pool Server)
  - Future Features

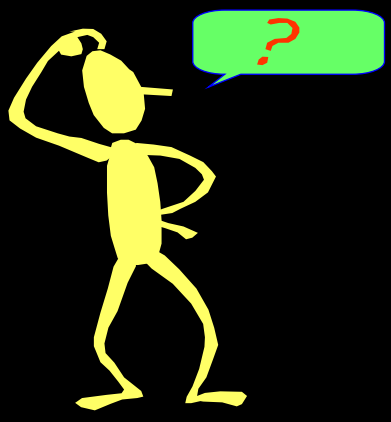
ORACLE

Information

- Release Notes (included in the kit).
- FAQ (included in the kit).
- Oracle® Rdb JDBC Driver F.A.Q. – Metalink note 255447.1
  (this note contains links to some other notes and examples.)
- E-mail to
  - Jim.Murray@oracle.com
  - Dr.Wolfgang.Kobarg-Sachsse@oracle.com

ORACLE

Questions ?



ORACLE