JDBC and Oracle Rdb – New Features

Wolfgang Kobarg-Sachsse

Oracle Support Services – Oracle Rdb Support

ORACLE

Overview

- History of the JDBC Driver for Rdb
- Functionality of the driver until version 7.1.2.1
- New Features that were built in into version 7.1.3

ORACLE

History

- Before the Rdb JDBC driver was released, the only possibility to connect from Java to an Rdb database was to use an Oracle JDBC driver and prepare the Rdb database for SQL*Net for Rdb – a lot of overhead.

- First Beta in 2002

- First production version 7.1.1 released in May 2003

- Version 7.1.2 released in December 2003

- Version 7.1.2.1 released in March 2004

- Version 7.1.3 in Beta since September 2004 (may be released as production in the moment as we speak here)

ORACLE

Usage

We know about more than 40 companies worldwide which are using the Rdb JDBC driver.

(Unfortunately only 8 of them registered for the use of the version 7.1.3 Beta)

ORACLE

Functionality of the driver before version 7.1.3

- Two derivates (Native / Thin)
- Thin Pool Server
- Thin Server Controller

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

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

Thin Server Controller

The Rdb JDBC Thin Server controller allows 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 –
      rdb$jdbc_home:rdbthinsrv.jar -controlpass mypassword
```

Then the Rdb JDBC Thin Server controller can be started:

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar
```

ORACLE

New Features of Version 7.1.3

- Installation is more convenient
- XML-formatted configuration files
- Better server control using SQL/Services
- A new multi-process multithreaded thin server process
- Use of Secure Socket Layer (SSL)
- Improved server control using enhanced command line interface
- KANJI and other ASIAN character set support.

ORACLE

Installation (1)

```
$ product install RDBJDBC71

  1 - ORCL VMS RDBJDBC71 V7.1-31B53F      Layered Product
  2 - Exit

The Oracle Rdb Native JDBC Drivers have been successfully installed in :

DISK$VMS073-1:[SYS0.SYSCOMMON.rdb$jdbc.0701-31B53F]

To help you setup the required logical names, a file named
  RDBJDBC_STARTUP.COM has been added to this installation directory
```

ORACLE

Installation (2)

Logical names defined in RDBJDBC_STARTUP.COM:

$ DEFINE/SYSTEM/NOLOG RDB$JDBC_HOME DISK$VMS073-1:[SYS0.SYSCOMMON.rdb$jdbc.0701-31B53F]

$ DEFINE/SYSTEM/NOLOG RDB$JDBC_LOGS DISK$VMS073-1:[SYS0.SYSCOMMON.rdb$jdbc.logs]

$ DEFINE/SYSTEM/NOLOG RDB$JDBC_COM DISK$VMS073-1:[SYS0.SYSCOMMON.rdb$jdbc.com]

$ DEFINE/SYSTEM/NOLOG RDBJDBCSHR RDB$JDBC_HOME:RDBJDBCSHR71.EXE

$ DEFINE/SYSTEM/NOLOG RDBJDBCMPSHR RDB$JDBC_HOME:RDBJDBCMPSHR71.EXE

$ DEFINE/SYSTEM/NOLOG RDBJDBCEXEC RDB$JDBC_HOME:RDBJDBCEXEC71.EXE

ORACLE

XML-Files (1)

```
<?xml version = '1.0'?>
<!-- Configuration file for Rdb Thin JDBC Drivers and Servers -->
<config>
   <!-- SERVERS -->
   <servers>
      <!-- DEFAULT server characteristics-->
      <server
         name="DEFAULT"
...
      />
      <server
...
      />
…
   </servers>
</config>
```

ORACLE

XML-Files (2)

```
<!-- DEFAULT server characteristics-->
    <server
        name="DEFAULT"
        type="RdbThinSrv"
        url="//localhost:1880/"
        maxClients="-1"
        srv.bindTimeout="0"
        srv.idleTimeout="0"
        srv.mcBasePort="5520"
        srv.mcGroupIP="239.192.1.10"
        autoStart="false"
        controlUser="jdbc_user"
        controlPass="0x811B15F866179583EB3C96751585B843"
        cfg="rdb$jdbc_com:sqlsrv_jdbc_server_cfg.xml"
        srv.startup="rdb$jdbc_home:rdbjdbc_startsrv.com"
       srv.onStartCmd= "@rdb$jdbc_com:rdbjdbc_sqs_onstartup.com"
    />
```

ORACLE

XML-Files (3)

controlUser="jdbc_user"

controlPass="0x811B15F866179583EB3C96751585B843"

Password obfuscation in server configuration files:

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar
rdbthincontrol> digest jdbc_user
 digest : 0x811B15F866179583EB3C96751585B843
rdbthincontrol>
```

ORACLE

XML-Files (4)

```
 <server
        name="SQSrjs1"
        type="RdbThinSrv"
        url="//localhost:1891/"
/>
     <server
        name="SQSrjs2"
        type="RdbThinSrv"
        url="//localhost:1892/"
/>
     <server
        name="SQSrjs3"
        type="RdbThinSrv"
        url="//localhost:1893/"
/>
```

ORACLE

XML-Files (5)

```
<!-- Pool Server -->
    <server
       name="SQS1880"
       type="RdbThinSrvPool"
       url="//localhost:1880/" >
       <pooledServer name="SQSrjs1"/>
       <pooledServer name="SQSrjs2"/>
       <pooledServer name="SQSrjs3"/>
    </server>
```

ORACLE

XML-Files (6)

Assume that the name of the XML-formatted configuration file is rdbjdbccfg.xml. The -cfg switch on the command line allows you then to specify the file specification of this file.

```
$java -jar rdbthinsrv.jar -cfg rdbjdbccfg.xml
```

or

```
$java -jar rdbthinsrvpool.jar -cfg rdbjdbccfg.xml
```

ORACLE

JDBC Dispatcher (1)

```
$ MCR SQLSRV_MANAGE71
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER POOL_DISP NETWORK_PORT TCPIP PORT_ID 1880 PROTOCOL  JDBC;
SQLSRV> show dispatcher pool_disp;
Dispatcher POOL_DISP
   State:                INACTIVE
   Autostart:            on
   Max connects:         100 clients
   Idle User Timeout:    <none>
   Max client buffer size:  5000 bytes
   Network Ports:                        (State)   (Protocol)
    TCP/IP  port    1880                  Unknown   JDBC clients
   Log path:          SYS$MANAGER:
   Dump path:         SYS$MANAGER:
```

ORACLE

JDBC Dispatcher (2)

Attention:

SQL/Services 7.1.6 is required.

But this version is still in Beta.

One of the reasons:

The JDBC protocol is unknown to the SQL/Services Manager GUI Tool and prevents to use it!

ORACLE

JDBC Dispatcher (3)

Before the JDBC dispatcher can be started we need two files:

- an XML formatted configuration file
- an on-startup command file

The name of the on-startup command file is up to you, you specify it in your XML formatted configuration file:

srv.onStartCmd= "@rdb$jdbc_com:rdbjdbc_sqs_onstartup.com"

$ type rdb$jdbc_com:RDBJDBC_SQS_ONSTARTUP.COM
$@sys$library:rdb$setver 7.1
$@sys$manager:java$141_setup fast

ORACLE

JDBC Dispatcher (4)

During the installation of the driver the file SQLSRV_JDBC_SERVER_STARTUP71.COM will be created in the SYS$COMMON:[SYSMGR] directory. The JDBC dispatcher starts this command procedure and knows from it which XML formatted file shall be used.

```
$   nam    = f$logical("RDB$JDBC_SQSNAM_"port"")
$   if nam.eqs."" then nam :="SQS"port""
...
$   cfg    = f$logical("RDB$JDBC_SQSCFG_"port"")
$   if cfg.eqs."" then cfg = f$search("rdb$jdbc_com:"nam'_CFG.XML")
$   if cfg.eqs."" then cfg =
      f$search("rdb$jdbc_com:SQLSRV_JDBC_SERVER_CFG.XML")
$   if cfg.eqs."" then cfg = f$search("rdb$jdbc_com:rdbjdbccfg.XML")
```

ORACLE

JDBC Dispatcher (5)

```
$ MCR SQLSRV_MANAGE71
SQLSRV> CONNECT SERVER;
SQLSRV> START DISPATCHER POOL_DISP;
SQLSRV> show dispatcher pool_disp;
Dispatcher POOL_DISP
   State:                RUNNING
   Autostart:            on
   Max connects:         100 clients
   Idle User Timeout:    <none>
   Max client buffer size:  5000 bytes
   Network Ports:                        (State)   (Protocol)
    TCP/IP  port    1880                  Inactive  JDBC clients
   Log path:          SYS$MANAGER:
   Dump path:         SYS$MANAGER:
   Log File:          SYS$SYSROOT:[SYSMGR]SQS_RTVMS4_POOL_DISP08F71.LOG;
   Dump File:         SYS$SYSROOT:[SYSMGR]SQS_RTVMS4_POOL_DISP08F.DMP;
```

ORACLE

JDBC Dispatcher (6)

```
$ show system/process=sqs*
OpenVMS V7.3-1  on node RTVMS4   4-APR-2005 03:45:01.47  Uptime  4 18:19:41
  Pid    Process Name    State  Pri      I/O       CPU       Page flts  Pages
21C0199A SQS1880         HIB      5    10324   0 00:00:23.97      3529   2796 M
21C0199C SQSRJS3         HIB      6     2135   0 00:00:03.14      3320   2562 M
21C0199E SQSRJS2         HIB      6     2135   0 00:00:03.05      3250   2595 M
21C019A0 SQSRJS1         HIB      6     2135   0 00:00:03.21      3209   2603 M
```

An application can now use the Thin Server Pool process through port 1880.

ORACLE

JDBC Dispatcher (7)

You can examine some log files (if it fails to start the dispatcher, or if it was successful to start the dispatcher, then some log files can be examined after shutdown the dispatcher to flush the content to the log files).

```
$ dir sys$manager:sqs*.log

 SQS_RTVMS4_JDBC_DISP08H71.LOG;4

$ dir rdb$jdbc_logs

SQSRJS1.LOG;5      SQSRJS2.LOG;4      SQSRJS3.LOG;4
SRVLOG_SQS1880.LOG;4               SRVLOG_SQSRJS1.LOG;28
SRVLOG_SQSRJS2.LOG;21              SRVLOG_SQSRJS3.LOG;21
SRVOUT_SQSRJS1.LOG;19              SRVOUT_SQSRJS2.LOG;21
SRVOUT_SQSRJS3.LOG;21
```

ORACLE

Multi-process Thin Server (1)

Very simple example: two threads that are doing the same update through the same thin server process.

```
public static void main(String[] args) {
    TestD d1 = new TestD(1701);
    // Ok if different servers, Deadlock if same server.
    TestD d2 = new TestD(1701);
    d1.start();
    try {
      Thread.sleep(1000);
    } catch (InterruptedException ie) {
    }
    d2.start();
  }
```

ORACLE

Multi-process Thin Server (2)

Result:

```
$ java "TestD"
Start
Start
1701 : create connection
1701 : create connection
1701 : Start transaction
1701 : Start transaction
1701 : update
1701 : commit
java.sql.SQLException: in <rdbjdbcsrv:execute_immediate>
%RDB-E-DEADLOCK, request failed due to resource deadlock
```

ORACLE

Multi-process Thin Server (3)

Output of the same program, but this time with a multi-process thin server:

```
$ java "TestD"
Start
Start
1701 : create connection
1701 : create connection
1701 : Start transaction
1701 : Start transaction
1701 : update
1701 : commit
1701 : update
1701 : commit
$
```

ORACLE

Multi-process Thin Server (4)

What is the difference?

This is the start command in the first case:

```
$spawn/nowait/proc=Rdb_Thin java -jar rdb$jdbc_home:rdbthinsrv.jar
```

And this is the start command in the second case:

```
$spawn/nowait/proc=Rdb_Thin_MP java -jar rdb$jdbc_home:rdbthinsrv.jar -type
      "RdbThinSrvMP"
```

ORACLE

Multi-process Thin Server (5)

What is a multi-process thin server?

"The Oracle Rdb thin multi-process server is a server-side component that Processes requests from the Oracle Rdb thin driver using small memory footprint sub-processes to carry out the requested operations on the Rdb database."

"The majority of the server operations are carried out in a client thread context within the main server process, handing off control to the client's allocated sub-process only when direct Rdb database operations are required. Each client has its own OpenVMS sub-process, thus concurrency is improved, as the server does not need to synchronize database operations."

ORACLE

Multi-process Thin Server (6)

How is this shown up?

Simple thin server:

```
$ show system/process=rdb*
OpenVMS V7.3-1  on node RTVMS4   4-APR-2005 08:16:03.94  Uptime  4 22:50:43
  Pid     Process Name    State  Pri      I/O       CPU       Page flts  Pages
21C01B40 RDB_THIN        CUR  1   6     2314   0 00:00:03.65     12127   3940 MS
```

Multi-process thin server:

```
$ show system/process=rdb*
OpenVMS V7.3-1  on node RTVMS4   4-APR-2005 08:20:00.23  Uptime  4 22:54:40
  Pid     Process Name    State  Pri      I/O       CPU       Page flts  Pages
21C01B48 RDB_THIN_MP     HIB      6     2422   0 00:00:03.36     11264   3073 MS
21C01B49 RDBTHIN00000001 LEF      6      585   0 00:00:00.28      1183   1235  S
21C01B4A RDBTHIN00000002 LEF      5      573   0 00:00:00.24      1170   1238  S
```

ORACLE

Multi-process Thin Server (7)

Three processes instead of one, how should that be better (in terms of performance)?

Let‘s listen to the developer of the driver:

“The server and the executor talk to each other via global memory so that should be fast. I do not think that this executor startup cost would be a problem as I would expect that any serious users of the JDBC drivers will probably be using connection pooling anyway so there will only be a small overhead for the executor startup on the initialization of the pooled connection.“

ORACLE

Secure Socket Layer (1)

SSL may be used by the Rdb Native Thin server and thin clients for communication over TCP/IP. SSL allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

Before trying to use SSL with the Rdb Native Drivers, you should familiarize yourself with general JAVA security and SSL concepts. Please refer to your JAVA documentation for general information on SSL and JAVA Security.

ORACLE

Secure Socket Layer (2)

We know already these three server “types“:

- RdbThinSrv
- RdbThinSrvMP
- RdbThinSrvPool

And with the suffix “SSL“ they are conform to the SSL:

- RdbThinSrvSSL
- RdbThinSrvMPSSL
- RdbThinSrvPoolSSL

ORACLE

Secure Socket Layer (3)

Here is an example of an entry in an XML-formatted configuartion file:

```
<server
        name="MYSSL"
        type="RdbThinSrvSSL"
        ssl.default="false"
        ssl.context="TLS"
        ssl.keyManagerFactory="SunX509"
        ssl.keyStoreType="jks"
        ssl.keyStore="rdbjdbcsrv.kst"
        ssl.keyStorePassword="CHANGETHIS"
        ssl.trustStore="rdbjdbcsrv.kst"
        ssl.trustStorePassword="CHANGETHIS"
/>
```

ORACLE

Secure Socket Layer (4)

You can‘t mix SSL and non-SSL server processes:

If a pool server is SSL-enabled, for security reasons it will only communicate with pooled servers within its pool that are also SSL-enabled. Non-SSL-enabled pooled servers within the pool will be ignored and not considered candidates for re-direction of connection requests.

ORACLE

Thin Server Controller Enhancements (1)

The Thin Server Controller command line is started with

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar
rdbthincontrol>
```

We know already

```
rdbthincontrol> digest password
```

The following command shows all servers, e.g.:

```
rdbthincontrol> poll
Polling servers ...
srvMPforRdb(1) //138.1.190.14:1705/ (0x21C02E9E<566242974>)
srv1forRdb(0) //138.1.190.14:1701/ (0x21C0229C<566239900>)
srv2forRdb(0) //138.1.190.14:1708/ (0x21C0229D<566239901>)
PoolforRdb(4) //138.1.190.14:1702/ (0x21C0229F<566239903>)
```

ORACLE

Thin Server Controller Enhancements (2)

To do more on the servers it is necessary to connect to one of the servers, using the control user and password, e.g.:

```
rdbthincontrol> connect //localhost:1701/ control_user control_user
rdbthincontrol> show server 1701
RDB$NODE               : 138.1.190.14
RDB$PORT               : 1701
RDB$STATUS             : Idle
RDB$SERVER_NAME        : srv1forRdb
RDB$SERVER_TYPE        : RdbThinSrv
RDB$SERVER_VERSION     : V7.1-300 20050315 B53F
RDB$SERVER_SHR_VERSION : V7.1-300 20050315 B53F
RDB$SERVER_PID         : 0x21C02EAC(566242988)
RDB$ALLOWS_ANON        : false
RDB$ALLOWS_BYPASS      : false
RDB$NUMBER_OF_CLIENTS  : 0
RDB$MAX_CLIENTS        : 1
RDB$TRACE_LEVEL        : 0
RDB$LOG_FILE           : rdb$jdbc_logs:srv1forRdb.log
RDB$RESTRICT_ACCESS    : false
```

ORACLE

Thin Server Controller Enhancements (3)

Other commands beside the “show server“ command:

close server

Closing a server sets the maxClients to 0 thus preventing any further connections to be made. Already established connections are not affected.

open server

You may issue an open command to re-open a closed server, which will reestablish the maxClient value for the server back to the value it was prior to closing.

ORACLE

Thin Server Controller Enhancements (4)

stop server

Stopping a server will forcibly terminate all database connections on that server and does not wait for client transaction completion. Consider using the close server command first, to stop further client connections and then stop server later when no clients are bound.

Example:

```
rdbthincontrol> stop server 1708
Successfully stopped Rdb Thin Server : srv2forRdb (//localhost:1708/)
```

ORACLE

Thin Server Controller Enhancements (5)

start server

Example:

```
rdbthincontrol> start server 1708
Starting server ...
.

RDB$NODE            : 138.1.190.14
RDB$PORT            : 1708
RDB$STATUS          : Idle
RDB$SERVER_NAME     : srv2forRdb
RDB$SERVER_TYPE     : RdbThinSrv
...
```

ORACLE

Thin Server Controller Enhancements (6)

show clients

Information about clients within active thin servers may be displayed using the thin controller.

stop clients

Clients within active thin servers may be stopped using the thin controller. Stopping a client will forcibly terminate all database connections on that server for that client and does not wait for client transaction completion.

ORACLE

Thin Server Controller Enhancements (7)

The “show“ and the “stop“ commands were already available in version 7.1.2.1. The enhancement on these commands in 7.1.3 is a greater flexibility, e.g. on “stop clients“:

stop active clients
stop all clients
stop active clients <name>
stop all clients <name>
stop active clients in <database_spec>
stop all clients in <database_spec>
stop clients
stop clients in <database_spec>
stop client <client_id>

ORACLE

Summary

Mainly we discussed

- how to use an SQL/Services JDBC Dispatcher to control the thin server processes,

- the new multithreaded thin server process.

Beside this we saw

- how XML-formatted configuration files are used,

- that a Secure Socket Layer (SSL) can be configured,

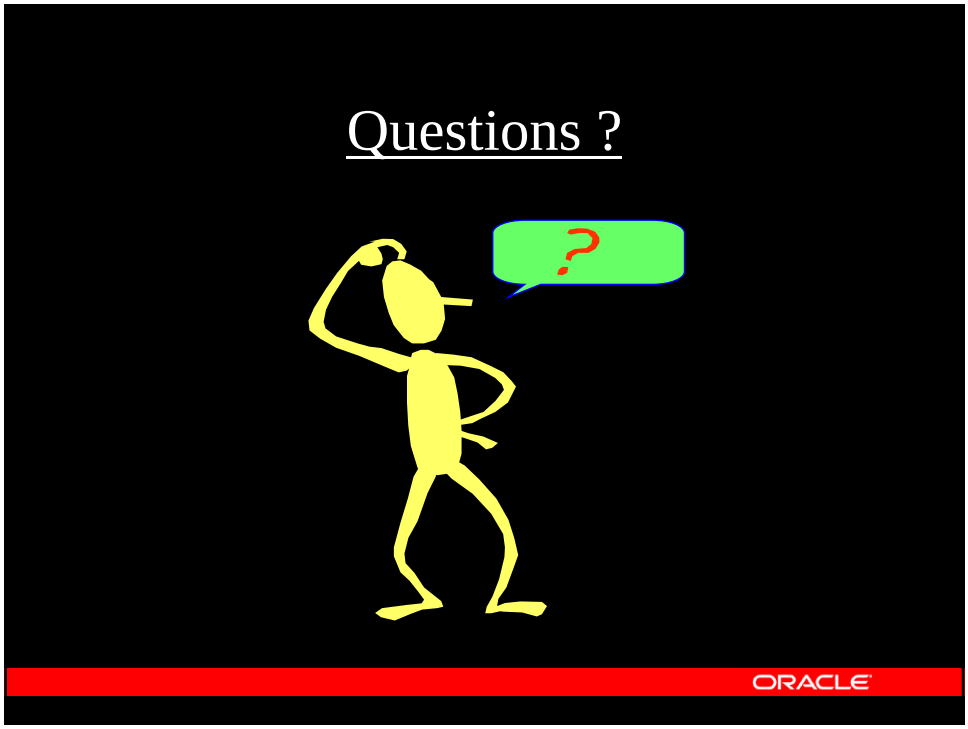
- how to use the thin server controller using the command line interface.

ORACLE

Information

- User Guide (included in the kit).
- 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