RDM Embedded 10.1

JDBC User Guide

Trademarks

Contents

Introduction

This section is not designed to be a complete JDBC tutorial. If you need more information about using JDBC you might be interested in the following online tutorials that are more in-depth than the information presented here:

- JDBC Basics: A tutorial from Sun covering beginner topics in JDBC
- JDBC Short Course: A more in-depth tutorial from Sun and JGuru

Driver Information

The JDBC driver for RDM Embedded is a JDBC 4.0 driver, requiring Java 6 or later. It is contained in the `rdmejdbc10.jar` file. When accessing data via JNI (where available), the `rdmejdbcjni10` shared library is also required. The driver has two modes that can be used. In the first mode, the JDBC driver is directly linked in with RDM Embedded and accessed via JNI. In the second mode, the JDBC driver communicates via TCP/IP with an rsqlserver. The advantage of the first mode is that the calls are all local, going through JNI – not a TCP/IP connection. The advantage of the second mode is only Java is needed on the application side – the **rsqlserver** component can run on a server somewhere else – but TCP/IP is required to communicate, potentially impacting performance.

Because the JDBC driver for RDM Embedded is a JDBC 4.0 driver which supports the Java Service Provider mechanism, no work is required by the application to load the driver. Simply including the `rdmejdbc10.jar` file in the CLASSPATH will cause the driver to register itself with the JDBC DriverManager.

Connection URL

The connection URL for the JDBC driver has the following format.

url	= jdbc:raima:rdme://<conninfo>[?<properties>]
conninfo	= <location>/<dblist>
location	= <hostlist> | local
hostlist	= <host>[:<port>][,<host>[:<port>]]…
host	= <identifier> | <ip_address>
dblist	= <dbname>[@<tfs>][,<dbname>[@<tfs>]]
tfs	= <host>[:<port>]
prop- erties	= auto commit=[on|off] | transaction isolation=[read_uncommitted|read_committed|repeatable_read|s- erializable] | open mode=[s|x|r] | document root=<path> | date separator=[-|/] | date format=[yyyymmdd|mmddyyyy|ddmmyyyy] | lock timeout=<number> | read only transactions=[on|off]

The connection URL must start with "`jdbc:raima:rdme://`". After that comes the location information. For JNI connections, this must be "local". For TCP/IP connections this can be a list of at least one hostname and port combinations. If more than one is specified, they will be attempted in order, with the first successful connection being used. For each, a hostname (or IP address) is required and the port is optional. If no port is supplied, the default port of 21555 will be used.

Following the connection information is an optional list of databases. Each database that is specified will be opened automatically after connecting. If the TFSR system is currently being used, you can specify a TFS to connect to (using the <dbname>@<tfs>) as documented elsewhere. It TFST or TFSS is used and a TFS is specified (i.e. if the '@' is included), an error will be generated.

Following the database list is an optional property list. The current properties supported are the following.

auto commit	this controls whether insert, update, and delete statements are automatically committed or whether explicit transactions are required. This parameter can be 'on' or 'off'. The default is 'on'.
transaction isolation	this controls the default transaction isolation mode. This can be one of the following isolation modes: 'read_uncommitted', 'read_committed', 'repeatable_read', and 'serializable'. The default is 'serializable'.
open mode	this tells the system the default open mode to use when opening the databases specified in the database list. Options are 's' for shared mode, 'x' for exclusive mode, and 'r' for read-only mode. The default is 's'.
document root	for TFST or TFSS, this allows the document root to be specified. For TFSR this parameter is ignored.
date separator	Species what separator to use between the year, month, and day values in a date. Options are '-' and '/'. The default is '-'.
date format	Specifies what order the year, month, and day values are returned in dates. Options are YYYYMMDD, MMDDYYYY, and DDMMYYYY. The default is YYYYMMDD.
lock timeout	Specifies the timeout to use for locking in seconds. The default is 10 seconds.
read only transactions	Specifies whether locking or read only transactions will be used for queries. The default is off.

Examples

```
Connection conn = DriverManager.getConnection("jdbc:raima:rdme://local/db1");
```

This syntax creates a local (via JNI) connection and attempts to open 'db1'. To use a local connection the rdmejdbcjni10 shared library must be available. If the database open fails the connection fails and an SQLException is thrown.

```
Connection conn = DriverManager.getConnection("jdbc:raima:rdme://localhost/db1");
```

This syntax creates a remote (via TCP/IP) connection to an rsqlserver running on the same machine (defaulting to port 21555) and attempts to open 'db1'. If the database open fails, the connection fails and an SQLException is thrown.

```
Connection conn = DriverManager.getConnection("jdbc:raima:rdme://local?auto com-
mit=off");
```

This syntax creates a local connection, doesn't specify any database to open and disables auto commit mode.

```
Properties props = new Properties();
props.setProperty("auto commit", "off");
props.setProperty("read only transactions", "on");
```

```
Connection conn = DriverManager.getConnection("jdbc:raima:rdme"//myserver:1000",
props);
```

This syntax creates a remote connection to an **rsqlserver** running on 'myserver' at port 1000, doesn't specify any database to open and disables auto commit and turns on read only transactions.

After a connection is established standard JDBC calls can be used to get access to data, change data and all other operations.

Unsupported JDBC Features

The following JDBC features are not supported by RDM Embedded.

The SQL XML type. The SQL engine in RDM Embedded has no XML type, so the JDBC driver does not support it. This includes the following methods which throw a SQLException.

- Connection.createSQLXML()
- PreparedStatement.setSQLXML(int, SQLXML)
- CallableStatement.setSQLXML(String, SQLXML)
- ResultSet.getSQLXML(int)
- ResultSet.getSQLXML(String)

The SQL Array type. The SQL engine in RDM Embedded has no Array type, so the JDBC driver does not support it. This includes the following methods which throw a SQLException.

- Connection.createArrayOf(String, Object[])
- PreparedStatement.setArray(int, Array)
- ResultSet.getArray(int)
- ResultSet.getArray(String)

The SQL Struct type. The SQL engine in RDM Embedded has no Struct type, so the JDBC driver does not support it. This includes the following method which throws a SQLException.

- Connection.createStruct(String, Object[])

The SQL Ref type. The SQL engine in RDM Embedded has no Ref type, so the JDBC driver does not support it. This includes the following methods which throw a SQLException.

- PreparedStatement.setRef(int, Ref)
- ResultSet.getRef(int)
- ResultSet.getRef(String)

The SQL ROWID type. Although the SQL engine in RDM Embedded does have a rowid concept it is limited, so the JDBC driver limits what can be done. The following methods throw SQLExceptions.

- PreparedStatement.setRowId(int, RowId)
- CallableStatement.setRowId(String, RowId)

The SQL URL type. The SQL engine in RDM Embedded has no URL type, so the JDBC driver does not support it. This includes the following methods which throw a SQLException.

- PreparedStatement.setURL(int, URL)
- CallableStatement.setURL(String, URL)

Modifiable result sets. The SQL engine in RDM Embedded does not allow a result set to be modified. The following methods will throw a SQLException.

- ResultSet.deleteRow()
- ResultSet.insertRow()

- ResultSet.updateClob(String, Reader)
- ResultSet.updateClob(String, Reader, long)
- ResultSet.updateDate(int, Date)
- ResultSet.updateDate(String, Date)
- ResultSet.updateDouble(int, double)
- ResultSet.updateDouble(String, double)
- ResultSet.updateFloat(int, float)
- ResultSet.updateFloat(String, float)
- ResultSet.updateInt(int, int)
- ResultSet.updateInt(String, int)
- ResultSet.updateLong(int, long)
- ResultSet.updateLong(String, long)
- ResultSet.updateNCharacterStream(int, Reader)
- ResultSet.updateNCharacterStream(int, Reader, long)
- ResultSet.updateNCharacterStream(String, Reader)
- ResultSet.updateNCharacterStream(String, Reader, long)
- ResultSet.updateNClob(int, NClob)
- ResultSet.updateNClob(int, Reader)
- ResultSet.updateNClob(int, Reader, long)
- ResultSet.updateNClob(String, NClob)
- ResultSet.updateNClob(String, Reader)
- ResultSet.updateNClob(String, Reader, long)
- ResultSet.updateNString(int, String)
- ResultSet.updateNString(String, String)
- ResultSet.updateNull(int)
- ResultSet.updateNull(String)
- ResultSet.updateObject(int, Object)
- ResultSet.updateObject(int, Object, int)
- ResultSet.updateObject(String, Object)
- ResultSet.updateObject(String, Object, int)
- ResultSet.updateRef(int, Ref)
- ResultSet.updateRef(String, Ref)
- ResultSet.updateRow()
- ResultSet.updateRowId(int, RowId)
- ResultSet.updateRowId(String, RowId)
- ResultSet.updateShort(int, short)
- ResultSet.updateShort(String, short)
- ResultSet.updateSQLXML(int, SQLXML)
- ResultSet.updateSQLXML(String, SQLXML)

- ResultSet.updateString(int, String)
- ResultSet.updateString(String, String)
- ResultSet.updateTime(int, Time)
- ResultSet.updateTime(String, Time)
- ResultSet.updateTimestamp(int, Time)
- ResultSet.updateTimestamp(String, Time)

Output parameters. The SQL engine in RDM Embedded only supports input parameters to procdures, so the following functions are not supported by the JDBC driver and will throw a SQLException.

- CallableStatement.getArray(int)
- CallableStatement.getArray(String)
- CallableStatement.getBigDecimal(int)
- CallableStatement.getBigDecimal(String)
- CallableStatement.getBlob(int)
- CallableStatement.getBlob(String)
- CallableStatement.getBoolean(int)
- CallableStatement.getBoolean(String)
- CallableStatement.getByte(int)
- CallableStatement.getByte(String)
- CallableStatement.getBytes(int)
- CallableStatement.getBytes(String)
- CallableStatement.getCharacterStream(int)
- CallableStatement.getCharacterStream(String)
- CallableStatement.getClob(int)
- CallableStatement.getClob(String)
- CallableStatement.getDate(int)
- CallableStatement.getDate(int, Calendar)
- CallableStatement.getDate(String)
- CallableStatement.getDate(String, Calendar)
- CallableStatement.getDouble(int)
- CallableStatement.getDouble (String)
- CallableStatement.getFloat(int)
- CallableStatement.getFloat (String)
- CallableStatement.getInt(int)
- CallableStatement.getInt (String)
- CallableStatement.getLong(int)
- CallableStatement.getLong (String)
- CallableStatement.getNCharacterStream(int)
- CallableStatement.getNCharacterStream(String)
- CallableStatement.getNClob(int)

- CallableStatement.getNClob(String)
- CallableStatement.getNString(int)
- CallableStatement.getNString (String)
- CallableStatement.getObject(int)
- CallableStatement.getObject(int, Map)
- CallableStatement.getObject (String)
- CallableStatement.getObject (String, Map)
- CallableStatement.getRef(int)
- CallableStatement.getRef(String)
- CallableStatement.getRowId(int)
- CallableStatement.getRowId (String)
- CallableStatement.getShort(int)
- CallableStatement.getShort (String)
- CallableStatement.getSQLXML(int)
- CallableStatement.getSQLXML (String)
- CallableStatement.getString(int)
- CallableStatement.getString (String)
- CallableStatement.getTime(int)
- CallableStatement.getTime (int, Calendar)
- CallableStatement.getTime (String)
- CallableStatement.getTime (String, Calendar)
- CallableStatement.getTimestamp(int)
- CallableStatement.getTimestamp (int, Calendar)
- CallableStatement.getTimestamp (String)
- CallableStatement.getTimestamp (String, Calendar)
- CallableStatement.getURL(int)
- CallableStatement.getURL (String)
- CallableStatement.registerOutParameter(int, int)
- CallableStatement.registerOutParameter(int, int, int)
- CallableStatement.registerOutParameter(int, int, String)
- CallableStatement.registerOutParameter(String, int)
- CallableStatement.registerOutParameter(String, int, int)
- CallableStatement.registerOutParameter(String, int, String)
- CallableStatement.wasNull()

Scrollable cursors. The SQL engine only supports static, forward-only cursors. Attempting to set the resultSetType to anything other than ResultSet.TYPE_FORWARD_ONLY or attempting to move in any other direction than forwards throws a SQLException. This affects the following methods.

- Connection.createStatement(int, int)
- Connection.createStatement(int, int, int)

- Connection.prepareCall(String, int, int)
- Connection.prepareCall(String, int, int, int)
- Connection.prepareStatement(String, int, int)
- Connection.prepareStatement(String, int, int, int)
- Statement.setFetchDirection(int)
- ResultSet.last()
- ResultSet.previous()
- ResultSet.relative()

Updatable cursors. The SQL engine does allow updates to the current results in one statement, but it requires a second statement to do it. The JDBC method which occurs in a single statement is not supported. Attempting to set the resultSetConcurrency to anything other than ResultSet.CONCUR_READ_ONLY throws a SQLException. This affects the following methods.

- Connection.createStatement(int, int)
- Connection.createStatement(int, int, int)
- Connection.prepareCall(String, int, int)
- Connection.prepareCall(String, int, int, int)
- Connection.prepareStatement(String, int, int)
- Connection.prepareStatement(String, int, int, int)

Cursors held over a commit. All SQL result sets are closed by a commit. Attempting to set the resultSetHoldability to anything other than ResultSet.CLOSE_CURSORS_AT_COMMIT throws a SQLException. This affects the following methods.

- Connection.createStatement(int, int, int)
- Connection.prepareCall(String, int, int, int)
- Connection.prepareStatement(String, int, int, int)
- Connection.setHoldability(int)

Auto generated keys. The SQL engine does not support the returning of auto generated columns/keys. Attempting to specify that auto generated columns/keys should be returned will throw a SQLException. This affects the following methods

- Connection.prepareStatement(String, int)
- Connection.prepareStatement(String, int[])
- Connection.prepareStatement(String, String[])
- Statement.execute(String, int)
- Statement.execute(String, int[])
- Statement.execute(String, String[])
- Statement.executeUpdate(String, int)
- Statement. executeUpdate (String, int[])
- Statement. executeUpdate (String, String[])
- Statement.getGeneratedKeys()

Setting a database to read only mode after it has been open. The SQL engine allows databases to be opened in read only mode (see the connection attributes), but if they are already open in shared or exclusive mode they can't be changed to read only mode. The reverse is also true - a database open in read only mode cannot be changed out of read only mode. Attempting to switch the database mode to or from read only mode will throw a SQLException. This affects the following method.

- Connection.setReadOnly(boolean)

Column and table privileges. The SQL engine does not support user accounts so privileges have no meaning. Attempting to get privilege information will throw a SQLException. This affects the following methods.

- DatabaseMetaData.getColumnPrivileges()
- DatabaseMetaData.getTablePrivileges()

Function information. The SQL engine does not provide the ability to enumerate system functions. Attempting to get function information will throw a SQLException. This affects the following methods.

- DatabaseMetaData.getFunctions()
- DatabaseMetaData.getFunctionColumns()

JDBC deprecated functions. The following are deprecated in JDBC 4.0 and thus are unsupported by the RDM Embedded JDBC driver and will throw a SQLException.

- PreparedStatement.setUnicodeStream(int, InputStream, int)
- CallableStatement.getBigDecimal(int, int)
- ResultSet.getBigDecimal(int, int)
- ResultSet.getBigDecimal(String, int)
- ResultSet.getUnicodeStream(int)
- ResultSet.getUnicodeStream(String)