Borrui Data

RapidsDB Unified JDBC Driver

Version 2.5

Table of Contents

1. Changes From Prior Versions 3
   - 1.1 Changes from Version 2.3 3
   - 1.2 Changes from Version 2.2 3
   - 1.3 Changes from Version 2.1 3
   - 1.4 Changes from Version 2.0 3
   - 1.5 Changes from Version 1.2 3
   - 1.6 Changes from Version 1.1 3
   - 1.7 Changes from Version 1.0 3
2. Introduction 4
3. Architecture 4
   - 3.1 Interface to RapidsDB Cluster 4
   - 3.2 Interface to External Data Sources 5
4. Configuring RapidsDB for JDBC 6
5. Using the Unified JDBC Driver to Access RapidsDB 7
   - 5.1 Driver Requirements 7
   - 5.2 Basic Connection URL 7
   - 5.3 Connection URL Options 7
     - 5.3.1 Specifying the Host and Port Numbers of the RapidDB Nodes to Connect to 7
     - 5.3.2 Specifying the Default Catalog and Schema 8
   - 5.4 Specifying a Native Connection at Startup 9
   - 5.5 Prepared Statement Support 10
   - 5.6 Unsupported JDBC Features 11
   - 5.7 Java Data Types Retrievable Through the JDBC Driver 14
   - 5.8 SQL to Java Data Type Conversions Supported in the JDBC Driver 15
   - 5.9 Sample JDBC Application 18
6. Using the Unified JDBC Driver to Access External Data Sources Directly 20
   - 6.1 Overview 20
   - 6.2 Opening a Connection to an External Data Source 20
   - 6.3 Usage Notes 21
   - 6.4 Example 21
7. Open-source Tools 21
   - 7.1 JMeter (http://jmeter.apache.org/) 21
   - 7.2 SQuirreL (http://squirrel-sql.sourceforge.net/) 22
   - 7.3 DBVisualizer (https://www.dbvis.com/) 25

8. Debugging 30
8.1 JDBC Driver 30
8.2 RapidsDB 32

1. Changes From Prior Versions

1.1 Changes from Version 2.4

- Switched the MySQL JDBC Driver to the version 5 JDBC Driver
- This version is released as part of the R3.4.2 release

1.2 Changes from Version 2.3

- Included JDBC Drivers for MySQL, Oracle, Postgres, Greenplum and DB2 as part of the Unified JDBC Driver jar file, which means that the user does not have to include JDBC Drivers in the class path for the Unified JDBC Driver when using a native connection to those databases.

1.2 Changes from Version 2.2

- Fixed a problem where the Unified JDBC Driver was not correctly loading the Greenplum JDBC Driver

1.3 Changes from Version 2.1

- Added support for the “connector” property as part of the connection url. The “connector” property allows the user to specify that the Unified JDBC Driver should open a native connection to the underlying database associated with the specified connector when starting up. This is equivalent to the user sending a “USE CONNECTOR <connector>;” command to the Unified JDBC Driver.
- Added support for the “USE CONNECTOR” command to be sent using any of the JDBC statement methods: execute, executeQuery, or executeUpdate.

1.4 Changes from Version 2.0

- Fixed a problem where the Unified JDBC Driver was incorrectly stripping statement prefixes from statements.

1.5 Changes from Version 1.2

- This is the first release of the Unified JDBC Driver which supports the ability to send JDBC requests directly to external data sources, bypassing RapidsDB. The Unified JDBC Driver continues to support access to the RapidsDB cluster.
- Version 2.0 is released as part of the R3.4.1 release

1.6 Changes from Version 1.1

- The RapidsDB JDBC Driver now supports prepared statements
- Version 1.2 is released as part of the RapidsDB 3.4 Release

1.7 Changes from Version 1.0

- The RapidsDB JDBC Driver supports Connection Balancing. This feature allows the user to spread the JDBC connections over multiple nodes in the RapidsDB Cluster. With Connection Balancing the user will be able to provide a list of DNS-resolvable host names or ip addresses (for nodes in the RapidsDB cluster) as part of the JDBC connection url, and the JDBC Driver will then round-robin connection requests over the specified set of hosts.
- The RapidsDB JDBC Driver now supports Statement.setMaxRows() to limit the number of rows returned to the application. As a result of this change, tools such as SQuirreL and DBVisualizer

(see section 6) will now correctly have the number of rows returned to the application limited based on the row limits set by the tool.

- Version 1.1 is released as part of the RapidsDB 3.1 Release.

2. Introduction

The RapidsDB Unified JDBC Driver is a type 4 JDBC Driver that provides a programmatic interface for Java applications to RapidsDB. A type 4 JDBC Driver is written entirely in Java and communicates with the database system using the database system's own network protocol. Because of this, the driver is platform independent; once compiled, the driver can be used on any system.

This document describes how to install and use the RapidsDB Unified JDBC Driver, it is not intended to be a guide for programming with JDBC. For more information on JDBC the user should refer to the standard JDBC API documentation.

3. Architecture

3.1 Interface to RapidsDB Cluster

Figure 1 below shows the architecture of the RapidsDB Unified JDBC Driver and how it interfaces to RapidsDB:

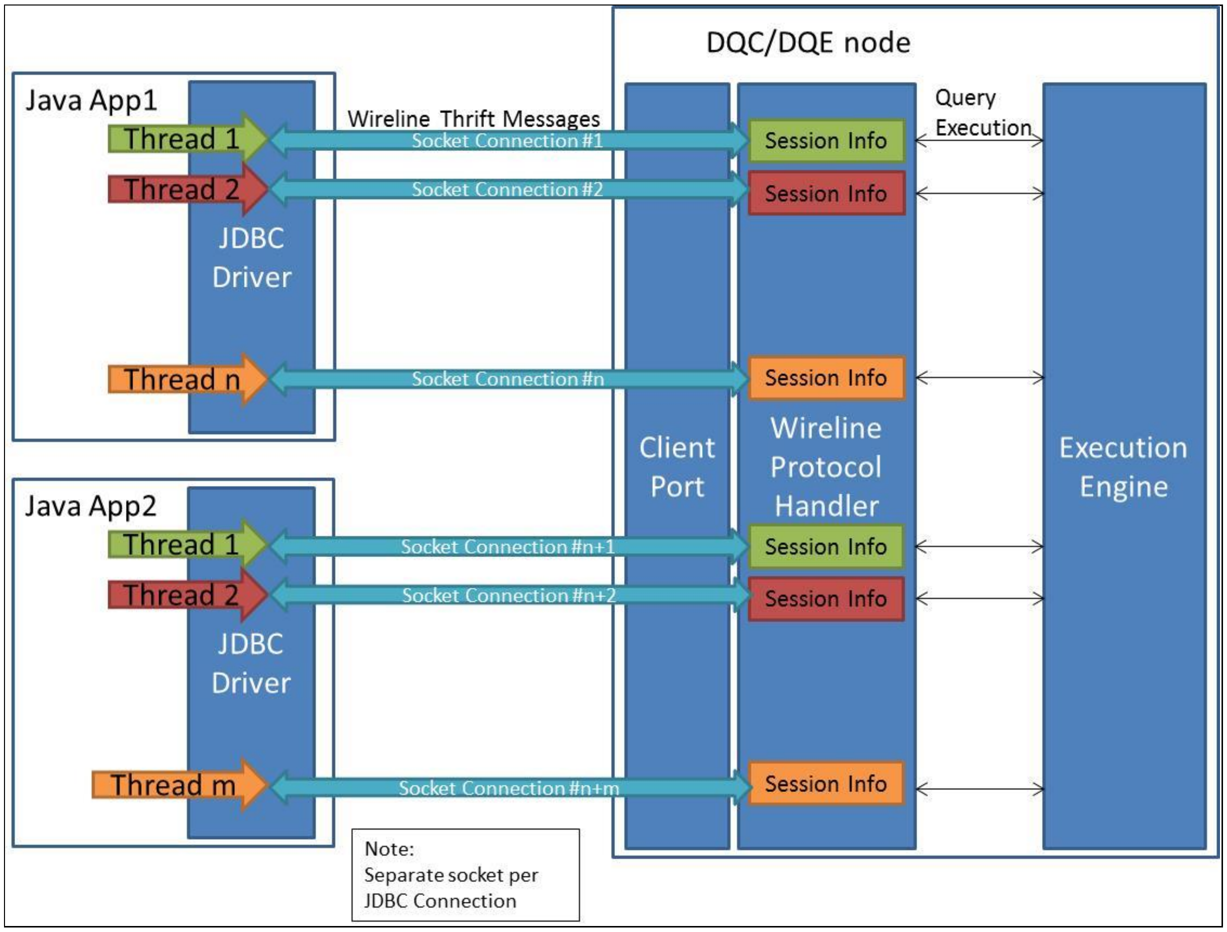


Figure 1. RapidsDB JDBC Architecture

The JDBC Driver communicates with a node in the RapidsDB cluster (can be a DQC or DQE node) using a Thrift-based messaging protocol (referred to as the RapidsDB Wireline Protocol). Within RapidsDB there is a component called the Wireline Protocol Handler that is responsible for managing the Thrift-based messaging interface. The Wireline Protocol Handler maintains session information for each active JDBC Connection. The JDBC Driver can be configured on the client (see 4.2) to use any of the nodes in the RapidsDB cluster, and there can be multiple JDBC Drivers communicating with different nodes in the RapidsDB cluster.

3.2 Interface to External Data Sources

Figure 2 below shows how the RapidsDB Unified JDBC Driver communicates directly with external data sources:

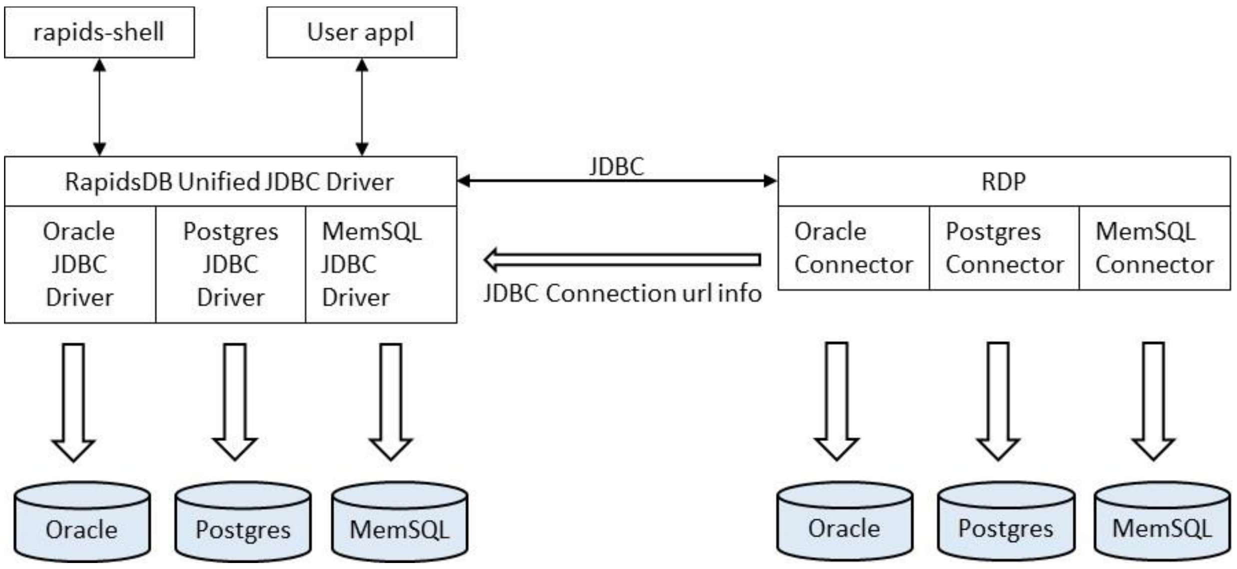


Figure 2. Architecture with External Data Sources

When communicating directly with an external data source, the Unified JDBC Driver will retrieve the connection url from the RapidsDB Connector definition, and it will then use the connection url associated with that Connector to establish a direct connection to the external data source.

4. Configuring RapidsDB for JDBC

The JDBC Driver can communicate with any node in the RapidsDB cluster. The parameter named "clientPort" in the RapidsDB cluster.config file allows the user to specify which port should be used for communication with the JDBC Driver. The default port number is 4333. Below is a sample section from the cluster.config file showing the "clientPort" parameter:

```
{
 "commonNodeConfig": {
  "enabled                      : true,
  "role"                        : "DQE",
  "clientPort"                  : 4333,
  "clusterPort"                 : 4334,
  “seEnabled”                  : false,
  “seArgs”                      : “”,
  "sshUsername"                 : "rapids",
  "sshPathToIdentityFile"       : "~/.ssh/id_rsa",
  "installationDir"             : "/opt/rdp",
  "workingDir"                  : "/opt/rdp/current",
  "startupCommand"              : "sh ./startDqx.sh",
  "shutdownCommand"             : "sh ./stopDqx.sh"
 },
```

5. Using the Unified JDBC Driver to Access RapidsDB

5.1 Driver Requirements

The driver has been tested under Java runtime 8. It has not certified for use under prior versions of Java.

The driver will emit a warning message from SLF4J if there is no logging framework supplied by the client application. The JDBC driver complies with the SLF4J logging API, but it is up to the end-user to provide a logging framework compatible to SLF4J.

5.2 Basic Connection URL

Every JDBC driver requires a connection URL that not only identifies it as a connection to a RapidsDB system, but can also configure that connection and set any default values it may use. All RapidsDB JDBC drivers must use a connection URL that begins with:

Basic RapidsDB Connection URL
jdbc:rdp:

Connection URLs are case sensitive. The "rdp" part of the URL indicates that this is a connection to a RapidsDB database. It will be recognised by the RapidsDB Unified JDBC Driver and ignored by JDBC drivers to other systems (e.g., the Postgres JDBC driver). When used, the above URL will tell the RapidsDB Unified JDBC Driver to connect to a RapidsDB node on localhost:4333.

The JDBC driver has a number of optional fields that can be specified to configure how the JDBC driver connects and interacts with the RapidsDB server. These include:

- Setting a list of one or more host and port numbers of the nodes in the RapidsDB Cluster that the JDBC Driver can connect to.
- Setting the default catalog and/or schema.

These fields need to be set in a specific order and with specific syntax. This is documented below.

5.3 Connection URL Options

Full RapidsDB Connection URL
jdbc:rdp:[//<host>[:<port>][,<host>[:<port>]…][/<catalog>[/<schema>]][?connector=<connector_name]

5.3.1 Specifying the Host and Port Numbers of the RapidDB Nodes to Connect to

The RapidsDB Unified JDBC Driver can connect to any DQC or DQE node on the port number configured by the clientPort parameter for that RapidsDB node (see 3 above). When not specified, the JDBC driver will connect to localhost:4333 by default. The user can provide a comma-separated list of host and port numbers, and the JDBC Driver will round-robin connection requests across the specified set of hosts. The use of multiple nodes allows for connection balancing across the specified set of hosts.

Specifying the list of host and port numbers must come immediately after the basic URL given above. The host and port list is delineated by a double forwardslash ("//") before the hostname.

The hostname can be a DNS-resolvable name of a host, or it may also be an IP address. Both IPv4 and IPv6 addresses are valid, however an IPv6 address must be enclosed within square brackets because it uses colons to delimit octets and the square brackets prevent the IPv6 address from being confused with the port number.

The port number is separated from the hostname by a colon (":"). Specifying the port number is optional.

Some valid examples of setting the hostname and port include:

```
jdbc:rdp://localhost
jdbc:rdp://localhost:4333
jdbc:rdp://127.0.0.1:4333                                        # uses an IPv4 loopback address
jdbc:rdp://[::1]:4333                                            # uses an IPv6 loopback address
jdbc:rdp://[2001:0db8:85a3:0000:0000:8a2e:0370:7334]:4333         # uses a specific IPv6 address
jdbc:rdp://192.168.10.10, 192.168.10.11, 192.168.10.12           # connection balancing over 3 hosts
                                                                   using the default port number
```

5.3.2 Specifying the Default Catalog and Schema

When using the JDBC driver, RapidsDB supports database sessions and therefore some degree of statefulness. This allows a user to set the default catalog and/or schema, which will then change the behavior of the system when subsequent queries are planned and executed. Setting the default catalog and schema can be achieved either by specifying them as part of the JDBC connection URL, or by sending the following explicit commands to RapidsDB:

- SET CATALOG catalogName;
- SET SCHEMA [catalogName .] schemaName;

When setting the schema, the catalog must either have been previously set, or the catalog name must be specified as part of the SET SCHEMA command.

The value of the current catalog and schema can be retrieved by sending the following query via JDBC:

```
        SELECT CURRENT_CATALOG, CURRENT_SCHEMA FROM RAPIDS.SYSTEM.TABLES LIMIT 1;
```

Setting a default catalog or schema will change the way that RapidsDB resolves unqualified tables (table names that do not have the catalog and schema specified). Without a default catalog or schema set, RapidsDB will search every catalog and schema for a table matching the table name in the query. However with a default catalog and/or schema set, any unqualified table name will be converted into a qualified table name with the default catalog and schema prepended before it. For example, if the default catalog was RAPIDS and the default schema was SYSTEM and the query was SELECT * FROM TABLES; then the unqualified table name TABLES would first be translated into the qualified name RAPIDS.SYSTEM.TABLES.

After a default catalog or schema has been set, it is possible to go back to the original behavior of the system by setting the current catalog and/or schema to null. e.g., SET CATALOG NULL;.

Instead of executing these SET commands individually, one can specify the default catalog and schema in the JDBC connection URL, and the JDBC driver will set these defaults for you automatically when the connection is established.

Setting the default catalog/schema is specified in a hierarchical fashion using a single forward slash ('/') to delineate these parameters. This grouping of parameters occurs after the hostname/port group (if set).

In this field, it is possible to set the following combinations of defaults:

1. The default catalog.
2. The default catalog and schema

The ordering of fields in the above list is significant.

Some examples of setting these defaults can be seen below:

With the host/port being specified:

```
jdbc:rdp://localhost:4333/RAPIDS              # sets the default catalog to RAPIDS
jdbc:rdp://localhost:4333/RAPIDS/SYSTEM    # sets the default catalog to RAPIDS and schema to SYSTEM
jdbc:rdp://192.168.10.10, 192.168.10.11/RAPIDS/SYSTEM  # set default catalog and schema with
                                                          multiple hosts
```

Without the host/port being specified:

```
jdbc:rdp:/RAPIDS                              # sets the default catalog to RAPIDS
jdbc:rdp:/RAPIDS/SYSTEM                       # sets the default catalog to RAPIDS and schema to SYSTEM
```

5.4 Specifying a Native Connection at Startup

The user can set the “connector” property to instruct the Unified JDBC Driver to open a connection to an external data source when starting up rather than opening a connection to the RapidsDB cluster. This is equivalent to the user sending a “USE CONNECTOR” command (see 6.1) after the Unified JDBC Driver has started up and connected to the RapidsDB Cluster.

Below are some example urls where the Unified JDBC Driver would open a connection to external data source associated with the RapidsDB Connector named “PG1”:

```
jdbc:rdp:?connector=PG1
```

The following example sets up multiple hosts when connecting to the RapidsDB Cluster, with the default catalog as “HADOOP” and the default schema as “PUBLIC” when connected to the RapidsDB Cluster:

```
jdbc:rdp://192.168.10.10, 192.168.10.11/HADOOP/PUBLIC?connector=PG1
```

The following example sets up the host ip address and port number when connecting to the RapidsDB Cluster, with the default catalog as “HADOOP” and the default schema as “PUBLIC” when connected to the RapidsDB Cluster:

```
jdbc:rdp://192.168.10.10:54333/HADOOP/PUBLIC?connector=PG1
```

Notes:

1. When specifying the "connector" property, the Unified JDBC Driver must still be able to connect to the RapidsDB Cluster in order to access the definition for the specified RapidsDB Connector.
2. When the "connector" property is set, all commands will be sent directly to the associated data store, bypassing RapidsDB. In order to have commands sent to RapidsDB the user must execute the command "use connector rapids;". The result of executing this command is that the default catalog and schema for the RapidsDB Cluster will be set
3. For native connections to any database other than MySQL (includes MemSQL), Postgres, Greenplum, Oracle or DB2, the location for the JDBC Driver jar file for that database must be included in the class path for the application using the Unified JDBC Driver. In the case of the rapids-shell, the JDBC Driver file must be located in the drivers directory for the rapids-shell.

5.5 Prepared Statement Support

The RapidsDB Unified JDBC Driver supports creating prepared statements from the JDBC connection using the standard Prepared Statement interface (e.g., `connection.prepareStatement("SELECT * FROM t WHERE col1 = ?;")`). The Driver supports all methods of the PreparedStatement interface except for the following:

- getParameterMetaData()
- setBytes(int, byte[])
- setBinaryStream(int, InputStream, int)
- setBinaryStream(int, InputStream)
- setBinaryStream(int, InputStream, long)
- setBlob(int, Blob)
- setBlob(int, InputStream)
- setBlob(int, InputStream, long)
- setNCharacterStream(int, Reader)
- setNCharacterStream(int, Reader, long)
- setNClob(int, NClob)
- setNClob(int, Reader)
- setNClob(int, Reader, long)
- setNString(int, String)
- setObject(in, Object, SQLType)
- setObject(in, Object, SQLType, int)
- setRowId(int, RowId)
- setSQLXML(int, SQLXML)
- setTime(int, Time)
- setTime(int, Time, Calendar)
- setUnicodeStream(int, InputStream, int)
- closeInCompletion()
- isCloseOnCompletion()
- getLargeUpdateCount()
- setLargeMaxRows(long)
- getLargeMaxRows()

- executeLargeBatch()
- executeLargeUpdate(String)
- executeLargeUpdate(String, int)
- executeLargeUpdate(String, int[])
- executeLargeUpdate(String, String[])

As for regular JDBC Statements, the RapidsDB Driver only supports TYPE_SCROLL_INSENSITIVE for the ResultSet. The RapidsDB Driver only supports forward iteration of ResultSets objects. (see item 23 in Unsupported JDBC Features section).

5.6 Unsupported JDBC Features

As of version 1.2, the following features are not supported:

	Interface	**Topic**	**Description**
1	General	Error Codes	Exceptions thrown by the JDBC driver may not have SQLState error codes or RapidsDB error codes.
2	Connection	Auto-commit transaction	Non-auto-commit transactions are not supported since RapidsDB does not support them.
4	Connection	Callable Statements	Callable statements are not supported since RapidsDB does not support them.
5	Connection	Rollbacks	Transaction rollbacks are not supported since RapidsDB does not support non-auto-commit transactions.
6	Connection	Savepoints	Transaction savepoints are not supported since RapidsDB does not support them.
7	Connection	Network Timeouts	Setting a network timeout with connection.setNetworkTimeout() is not yet supported.
8	Connection	Connection pooling	Connection pooling within the JDBC driver is not supported.
9	Statement	Multiple Statements	Multiple statements separated by semicolons and executed in a single call to execute() are not yet supported, however explicit batching of non-SELECT statements via statement.addBatch() and statement.executeBatch() is supported.
10	Statement	Update counts	Update counts from executing non-select statements are currently either set to 0, or -1 in the case of a ResultSet being returned. This is because RapidsDB does not currently return update counts when non-select statements are executed.

11	Statement	closeOnCompletion	Statement.setCloseOnCompletion() is not supported.
12	Statement	Pooling statements	Statement pooling via statement.setPoolable() is not supported.
13	Statement	Auto-generated keys	Execution and retrieval of auto-generated keys is not supported (e.g., via statement.execute(String sql, String[] columns), since RapidsDB does not yet support retrieval of auto-generated keys.
14	Statement	Multiple ResultSets	Calling Statement.getMoreResults() is not yet supported.
15	Statement	Query Timeout	Setting a query timeout via Statement.setQueryTimeout() is not yet supported.
16	Statement	Warnings	The JDBC driver will not return warnings since RapidsDB currently does not support warnings.
17	Statement	Escape Processing	Escape processing within the JDBC driver is generally not supported.
18	Statement	Fetch Direction	The JDBC driver only supports a forward fetch direction of ResultSets.
19	Statement	Fetch Size	The fetch size set for a JDBC statement is not currently used by the driver.
20	Statement	Max Field Size	The driver currently does not adhere to any maximum field size set via Statement.setMaxFieldSize().
21	ResultSet	ResultSet Updatability	ResultSets only support read-only operations and are not updatable. Operations such as insertRow(), deleteRow(), updateRow() and update<*datatype*>() related operations are not supported.
22	ResultSet	ResultSet Holdability	By default, ResultSets are kept open over commits. They do not support "close at commit" semantics since RapidsDB does not support this.
23	ResultSet	ResultSet Type	ResultSets currently only support forward-only iterating, even though Statement.getResultSetType() returns TYPE_SCROLL_INSENSITIVE. This is because this type of ResultSet is required for JMeter, however ResultSets do not yet support backwards scrolling, or absolute or relative cursor positioning via methods like

			absolute()
			afterLast()
			beforeFirst()
			relative()
			first()
			last()
			previous()

24	ResultSet	ResultSet Sensitivity	The JDBC driver only supports ResultSets that are insensitive to database changes. As such, related operations such as refreshRow() are not supported.
25	ResultSet	Data Types	The JDBC driver does not support all possible data types defined in the ResultSet interface definition. Refer to the table below for supported and unsupported data types.
26	ResultSet	Data Type Conversions	The JDBC driver does not support all possible conversions from SQL data types to Java data types. Refer to the table below for supported and unsupported data type conversions.

27	DatabaseMetaData	Unsupported Methods	The following methods are not supported:
			
			getBestRowIdentifier()
			getColumnPrivileges()
			getCrossReference()
			getExportKeys()
			getFunctionColumns()
			getFunctions()
			getImportedKeys()
			getIndexInfo()
			getProcedureColumns()
			getProcedures()
			getSQLKeywords()

28	DatabaseMetaData	Methods Returning Empty Results or Empty ResultSet	The following methods are not fully implemented and return empty ResultSets:
			
			getPrimaryKeys()
			getStringFunctions()
			getSuperTables()
			getSuperTypes()
			getSystemFunctions()
			getTablePrivileges()

			getTimeDateFunctions()
			getUDTs()
			getVersionColumns()

29	ResultSetMetaData	Column Name And Label	RapidsDB and the JDBC driver currently do not distinguish between column names and column labels. If an alias is given for a column then that alias will be used as both the column name and the column label.
30	ResultSetMetaData	Originating Table, Schema And Catalog	RapidsDB and the JDBC driver currently do not provide the catalog name, schema name or table name that each column of a ResultSet came from. Instead, an empty string is returned.

5.7 Java Data Types Retrievable Through the JDBC Driver

Data Type	Supported?	Notes
getArray()	No	Not supported in RapidsDB.
getAsciiStream()	Yes	
getBigDecimal()	Yes	
getBinaryStream()	Yes	Binary data types are not supported in RapidsDB.
getBlob()	Yes	Binary data types are not supported in RapidsDB.
getBoolean()	Yes	
getByte()	Yes	RapidsDB returns all integers with 64 bit precision.
getBytes()	Yes	Binary data types are not supported in RapidsDB.
getCharacterStream()	Yes	
getClob()	No	
getDate()	Yes	RapidsDB currently only supports timestamps, but these can be retrieved as dates.
getDouble()	Yes	
getFloat()	Yes	RapidsDB returns all floating point numbers with double length precision.
getInt()	Yes	RapidsDB returns all integers with 64 bit precision.

getLong()	Yes	
getNCharacterStream()	No	RapidsDB does not support the NVARCHAR or NCHAR data types.
getNClob()	No	RapidsDB does not support the NVARCHAR or NCHAR data types.
getNString()	No	RapidsDB does not support the NVARCHAR or NCHAR data types.
getObject()	Yes	
getRef()	No	RapidsDB does not support REFs.
getRowId()	No	RapidsDB does not support RowIds.
getShort()	Yes	
getShort()	Yes	
getSQLXML()	No	
getString()	Yes	
getTime()	Yes	RapidsDB currently only supports timestamps, but these can be retrieved as times.
getTimestamp()	Yes	RapidsDB supports timestamps with nanosecond precision, however the JDBC interface only supports precision to the millisecond level.
getUnicodeStream()	No	
getURL()	No	

5.8 SQL to Java Data Type Conversions Supported in the JDBC Driver

JDBC Method	**SQL Data Type**								**Notes**
	BOOLEAN	**INTEGER**	**DECIMAL**	**FLOAT**	**TIMESTAMP**	**VARCHAR**	**INTERVAL**	**VARBINARY**	
getArray()									Data type not supported.

getAsciiStream()	X	X	X	X	X	X	X		
getBigDecimal()	X	X	X	X		X			Conversion from VARCHAR supported if it only contains a numeric value.
getBinaryStream()						X			
getBlob()						X			
getBoolean()	X	X	X	X		X			Numerics: A value of 0 is converted to FALSE. Everything else is TRUE. VARCHAR: Conversion is supported if the string is a case insensitive form of "true" or "false".
getByte()	X	X	X	X		X			Conversion from VARCHAR supported if it only contains a numeric value. It is yet to be determined whether overflowing values should return a modulus or throw an exception.
getBytes()						X			
getCharacterStream()	X	X	X	X	X	X	X		

getClob()									Data type not supported.
getDate()					X	X			Conversion from VARCHAR support if it only contains a valid date string.
getDouble( )	X	X	X	X		X			Conversion from VARCHAR support if it only contains a valid numeric value.
getFloat()	X	X	X	X		X			Conversion from VARCHAR support if it only contains a valid numeric value.
getInt()	X	X	X	X		X			Conversion from VARCHAR support if it only contains a valid numeric value.
getLong()	X	X	X	X		X			Conversion from VARCHAR support if it only contains a valid numeric value.
getNChara cterStream ()									Data type not supported.
getNClob()									Data type not supported.
getNString( )									Data type not supported.
getObject()	X	X	X	X	X	X	X		
getRef()									Data type not supported.
getRowId()									Data type not supported.

getShort()	X	X	X	X		X			Conversion from VARCHAR support if it only contains a valid numeric value.
getSQLXML()									Data type not supported.
getString()	X	X	X	X	X	X	X		
getTime()					X	X			Conversion from VARCHAR support if it only contains a valid time string.
getTimestamp()					X	X			Conversion from VARCHAR support if it only contains a valid timestamp string.
getURL()									Data type not supported.
getUnicode()									Data type not supported.
getShort()	X	X	X	X		X			Conversion from VARCHAR support if it only contains a valid numeric value.

5.9 Sample JDBC Application

```
// Copyright (c) 2018 Boray Data Co. Ltd.  All rights reserved.

// Instructions:
// * Make the directories [some_dir]/com/rapidsdata
// * Copy this content into a file at [some_dir]/com/rapidsdata/JdbcSample.java
//
// From [some_dir]:
// To compile:  javac com/rapidsdata/JdbcSample.java
// To execute:  java -cp .:/path/to/rapids-jdbc-1.0.0.jar JdbcSample
//
```

```
// This example assumes an empty table exists called TABLE_A1
// with columns (A VARCHAR(128), B INTEGER),
// and that the JDBC driver can connect to a node listening on localhost:4333.
//
//
// This application will insert a row of data into the TABLE_A1 table and then
// read it back out. The returned data should be (A=RAPIDS DB, B=88888).
//
// To avoid a warning from SLF4J about logging defaulting to a no-operation,
// simply add a compatible logging framework to the class path when running
// the sample (e.g. any one of slf4j-nop.jar, slf4j-simple.jar, slf4j-log4j12.jar,
// slf4j-jdk14.jar or logback-classic.jar).


package com.rapidsdata;

import java.sql.*;

public class JdbcSample
{
  public static boolean doSample()
  {
    boolean bRet = false;
    String url = "jdbc:rdp://localhost:4333";
    String userName = "";
    String password = "";
    Connection conn = null;
    Statement stmt = null;
    try {
      // Connect to Rapids
      conn = DriverManager.getConnection(url, userName, password);

      // Get a Statement object from the connection
      stmt = conn.createStatement();

      // TABLE_A1 has two columns, A as VARCHAR and B as BIG INTEGER
      // Insert a row in the table
      stmt.executeUpdate("INSERT INTO TABLE_A1 VALUES ('RAPIDS DB', 88888);");

      // Execute an SQL statement "Select...".
      ResultSet rs = stmt.executeQuery("SELECT A, B FROM TABLE_A1;");

      ResultSetMetaData metaData = rs.getMetaData();
      int numberOfColumns = metaData.getColumnCount();
      while (rs.next()) {
        for (int i = 1; i <= numberOfColumns; i++) {
          System.out.print(String.format("%s=", metaData.getColumnName(i)));
          if (metaData.getColumnTypeName(i).equalsIgnoreCase("VARCHAR"))
```

```
            System.out.print(String.format("%s, ", rs.getString(i)));
          else if (metaData.getColumnTypeName(i).equalsIgnoreCase("INTEGER"))
            System.out.print(String.format("%d, ", rs.getInt(i)));
          else if (metaData.getColumnTypeName(i).equalsIgnoreCase("BIGINT"))
            System.out.print(rs.getLong(i) + ", ");
        } // end for-loop
        System.out.println(" ");
      } // end while-loop
      rs.close();
      stmt.close();
      conn.close();
      bRet = true;

    } catch (SQLException se) {
      System.out.println(se.toString());
    }
    return bRet;
  }

  public static void main(String[] args)
  {
    doSample();
  }
}
```

6. Using the Unified JDBC Driver to Access External Data Sources Directly

6.1 Overview

The RapidsDB Unified JDBC Driver allows the user application to establish a native connection directly to any external data source that can be accessed via a RapidsDB Connector using the native JDBC Driver for that external data source. When communicating directly with an external data source, the Unified JDBC Driver will retrieve the connection url for the external data source from the RapidsDB Connector definition for that data source, and it will then use that connection url to establish the direct connection to the external data source. After establishing the connection to the external data source, the RapidsDB Unified JDBC Driver will operate in pass-through mode and send all requests directly to the external data source via the native JDBC Driver, and as such it will expose all of the features and capabilities supported by the native JDBC Driver.

6.2 Opening a Connection to an External Data Source

To open a connection to an external data source the user application must execute the following SQL statement:

```
USE CONNECTOR <Connector Name>;
```

The RapidsDB Unified JDBC Driver will retrieve the JDBC connection url for the native JDBC Driver from the Connector definition in RapidsDB. The RapidsDB Unified JDBC Driver will then use that connection url to open a connection to the external data source. After successfully opening the connection to the target data source, the RapidsDB Unified JDBC Driver will route all subsequent commands directly to the JDBC connection just opened, bypassing RapidsDB completely.

6.3 Usage Notes

1. For native connections to any database other than MySQL (includes MemSQL), Postgres, Greenplum, Oracle or DB2, the location for the JDBC Driver jar file for that database must be included in the class path for the application using the Unified JDBC Driver. In the case of the rapids-shell, the JDBC Driver file must be located in the drivers directory for the rapids-shell.
2. The RapidsDB Unified JDBC Driver requires that the RapidsDB Cluster is available in order for the RapidsDB Unified JDBC Driver to access the Connector information associated with any external data source.
3. To switch back to using RapidsDB as the target data source, the following command must be sent:

```
USE CONNECTOR rapids
```

6.4 Example

The following example assumes that a connection has already been established with the RapidsDB Cluster. This example uses the following Connector to a Postres database:

```
CREATE CONNECTOR PG2 TYPE POSTGRES WITH
CONNECTIONSTRING='jdbc:postgresql://boray03:5432/tpch', USER='postgres', PASSWORD='postgres'
NODE BORAY03 NODE BORAY04 CATALOG * SCHEMA * TABLE * ;
```

1. Open the connection to Postgres:

```
// Execute the Use Connector command to open connection to Postgres database
ResultSet rs = stmt.executeQuery("USE CONNECTOR PG2;");
```

2. All subsequent requests sent directly to Postgres using the Postgres JDBC Driver, and as such the features supported are limited to those supported by the Postgres JDBC Driver.

3. Re-establish connection to RapidsDB Cluster:

```
// Execute the Use Connector command to open connection to Postgres database
ResultSet rs = stmt.executeQuery("USE CONNECTOR rapids;");
```

4. All subsequent requests sent to the RapidsDB Cluster, and as such the features supported are limited to this discussed in section 5 of this document.

7. Open-source Tools

7.1 JMeter (http://jmeter.apache.org/)

To set up a JMeter test plan using JDBC, please refer to steps documented at this location: http://jmeter.apache.org/usermanual/build-db-test-plan.html

To configure the RapidsDB Unified JDBC Driver follow these steps:

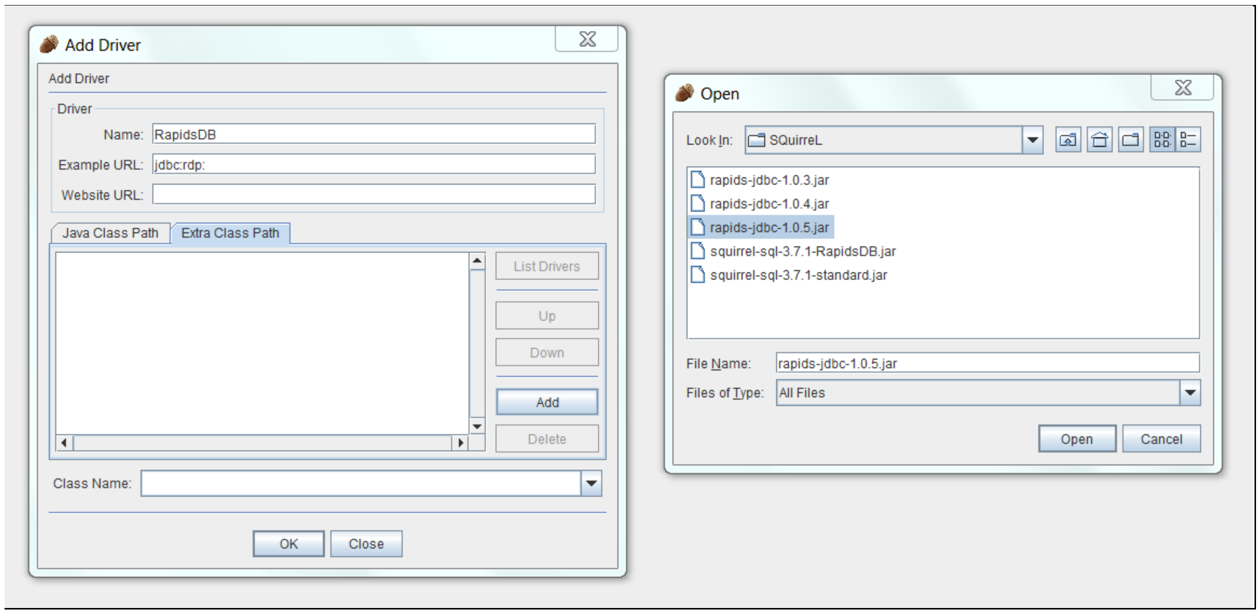
- Under menu "Edit", select "Add" then "Config Element".
- Then select “JDBC Connection Configuration”.
- At “Database URL:” enter "jdbc:rdp://<address>:<port>"

  Where <address> is the ip address for the RapidsDB node to connect to
  <port> is the port number being used to communicate with the RapidDB node (default is 4333)
- At “JDBC Driver Class”, enter com.rapidsdata.jdbcdriver.Driver
- At “Username:” enter "root" No password

7.2 SQuirreL (http://squirrel-sql.sourceforge.net/)

The first step is to add the RapidsDB Unified JDBC Driver:

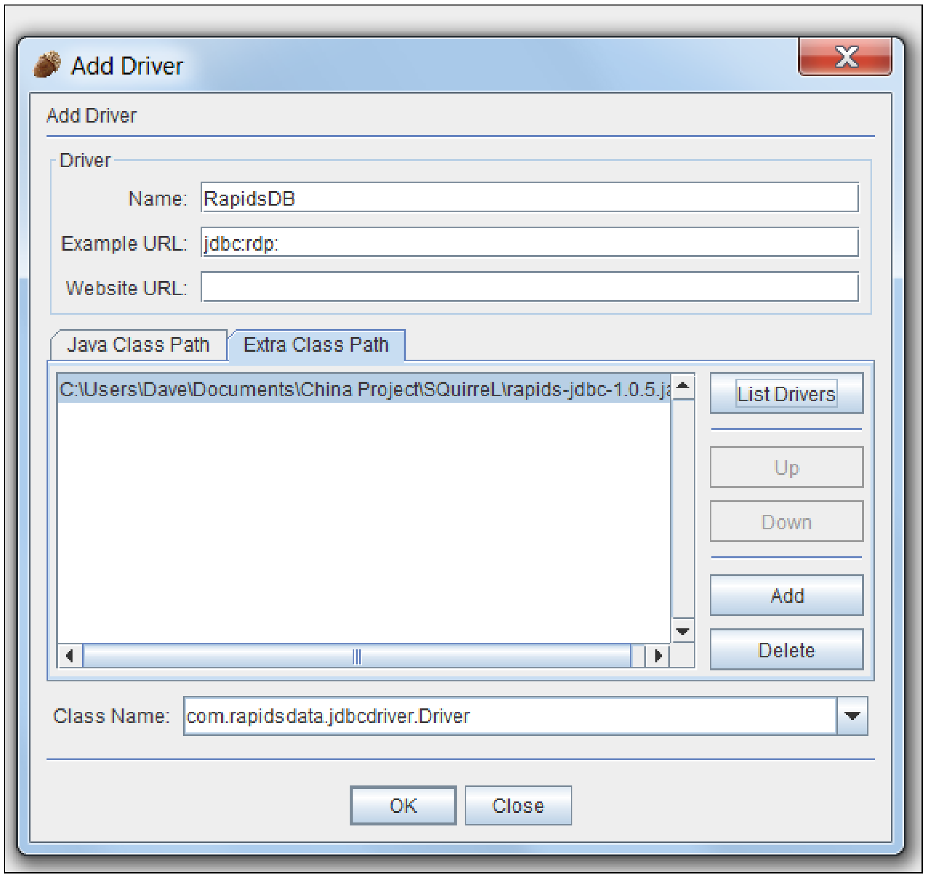
- From the menu "Drivers", select "New Driver"
- Enter driver name e.g. "RapidsDB"; at the "Example URL:" field, enter "jdbc:rdp:"
- Select the tab labeled as "Extra Class Path", select "Add" then enter the path where the driver is (e.g. C:\Users\Dave\rapids\JDBC) and press "Open". See Figure 2 below.


Add Driver
Driver
Name: RapidsDB
Example URL: jdbc:rdp:
Website URL:
Class Name:

Open
Look In: SQuirreL
rapids-jdbc-1.0.3.jar
rapids-jdbc-1.0.4.jar
rapids-jdbc-1.0.5.jar
squirrel-sql-3.7.1-RapidsDB.jar
squirrel-sql-3.7.1-standard.jar
File Name: rapids-jdbc-1.0.5.jar
Files of Type: All Files

Figure 2. SQuirreL – Adding a Driver (1)

- Then press the "List Drivers" button, the string value "com.rapidsdata.jdbcdriver.Driver" should show up at the "Class Name" field. See Figure 3 below.



Name: RapidsDB

Example URL: jdbc:rdp:

Website URL:

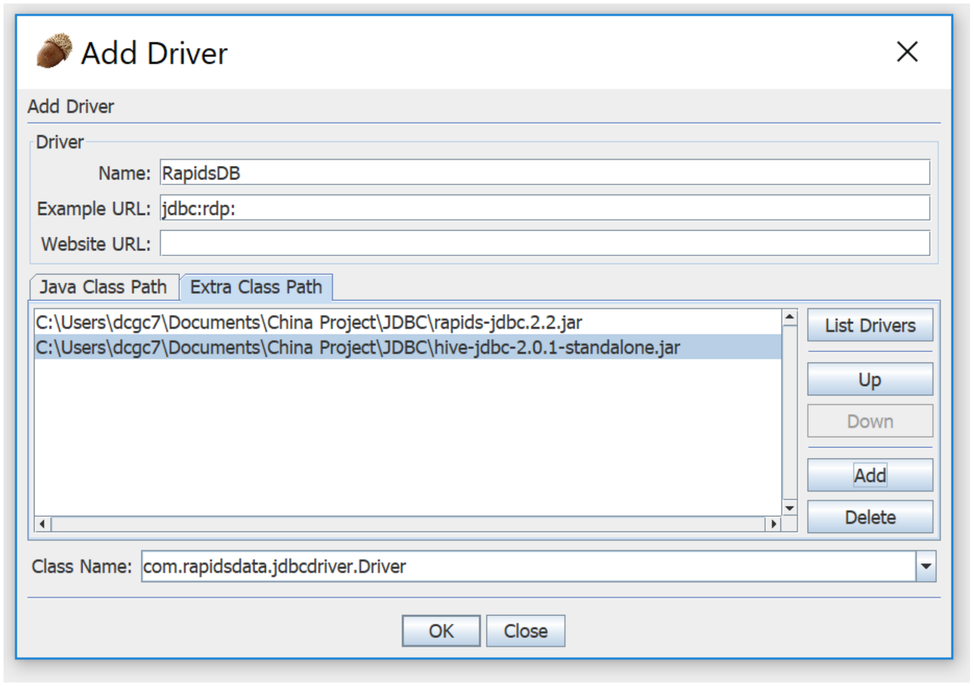
C:\Users\Dave\Documents\China Project\SQuirreL\rapids-jdbc-1.0.5.ja

Class Name: com.rapidsdata.jdbcdriver.Driver

Figure 3. SquirreL – Adding a Driver (2)

Notes:

1. If the connection url to be used for the Alias (see next step) includes the ?connector=<connector> option to establish a native connection (see 5.4) to another database, or if a "use connector <connector>" command is going to be used to open a native connection (see 6.), then if the native connection uses a JDBC Driver other than one of the following: MySQL, Postgres, Greenplum, Oracle or DB2, then the path to that JDBC Driver must also be included in the Extra Class Path setting. The screen below shows an example of including the JDBC Driver for Hive:


Add Driver

Driver

Name: RapidsDB

Example URL: jdbc:rdp:

Website URL:

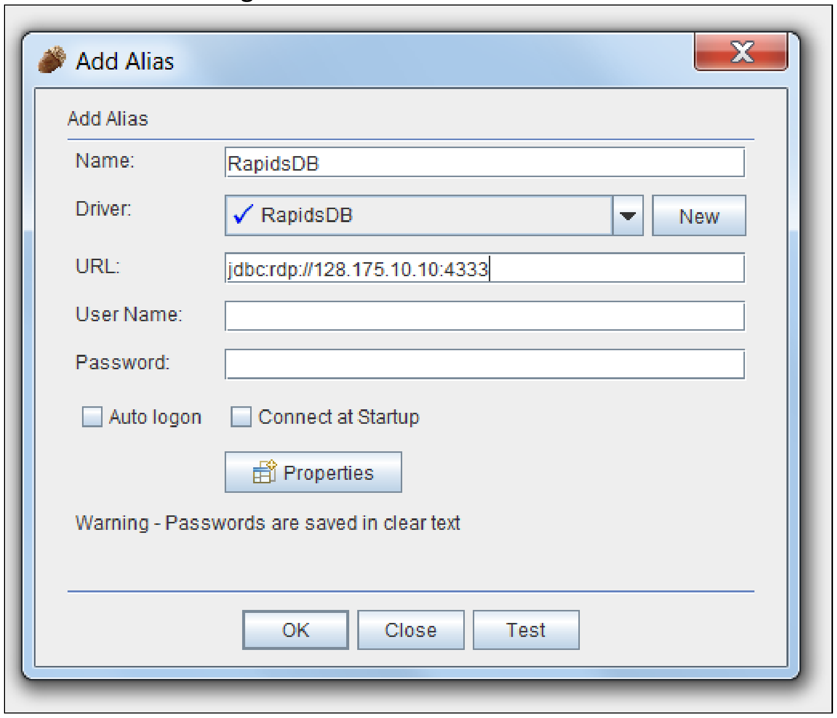
Java Class Path | Extra Class Path

C:\Users\dcgc7\Documents\China Project\JDBC\rapids-jdbc.2.2.jar
C:\Users\dcgc7\Documents\China Project\JDBC\hive-jdbc-2.0.1-standalone.jar

Class Name: com.rapidsdata.jdbcdriver.Driver

The next step is to set up a connection to RapidsDB:

- From the menu "Aliases", select "New Alias" and enter the name for the Alias, e.g. "RapidsDB".
- At the "Driver:" dropdown, select the driver created from the above step, e.g. "RapidsDB"
- The "URL: should have "jdbc:rdp:", if not, enter "jdbc:rdp:" Then add "//<address>:<port>"
- Leave the User blank. See Figure 4 below.


Add Alias

Name: RapidsDB

Driver: ✓ RapidsDB

URL: jdbc:rdp://128.175.10.10:4333

User Name:

Password:

☐ Auto logon ☐ Connect at Startup

Warning - Passwords are saved in clear text

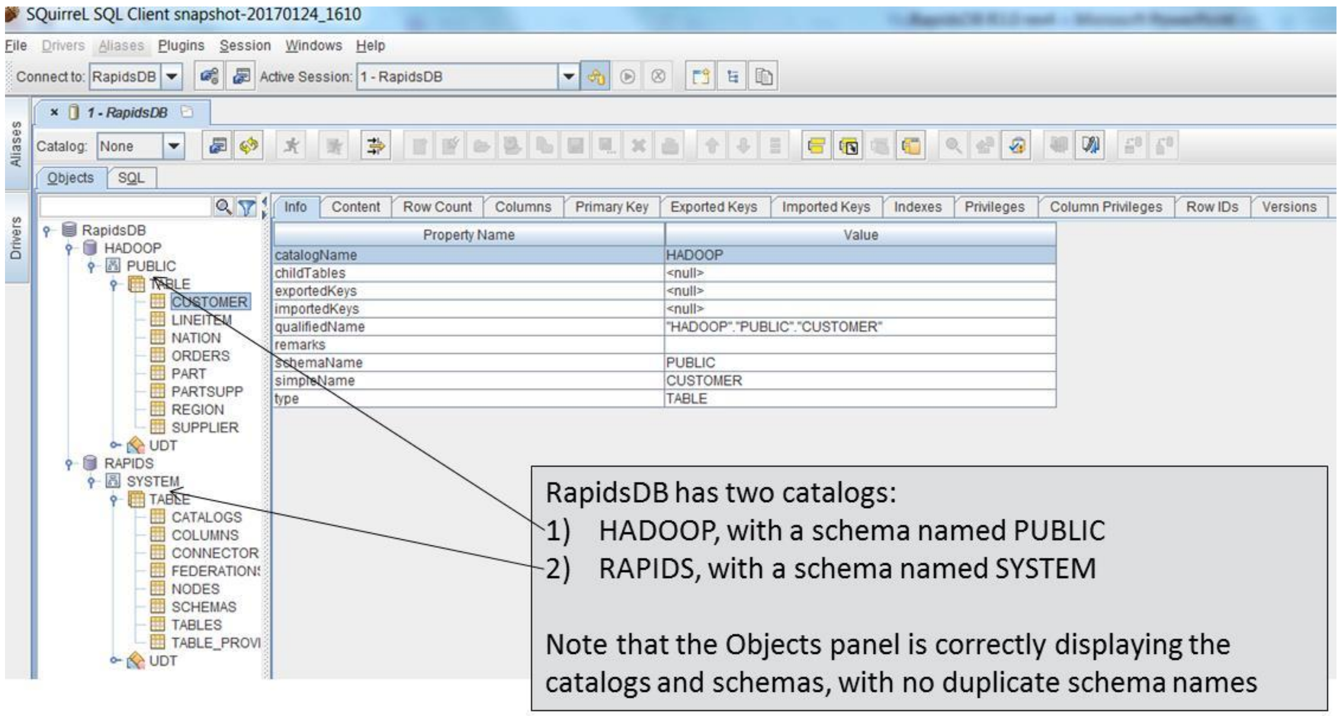
Figure 4. SQuirreL – Adding an Alias

- Click "Test", then click "Connect", to verify it can connect to the RapidsDB node.

- At the main screen, now the "Aliases" tab should show the entry "RapidsDB".
- Double-click on "RapidsDB" and click "Connect" at the popup screen, now it should connect to the RapidsDB node and shows the catalog information. See Figure 5 below.

NOTES:

1. The standard version of SQuirreL does not handle systems such as RapidsDB that have multiple catalogs each with their own schema. When displaying the metadata for RapidsDB, the standard version of SQuirreL will correctly display all of the catalogs, but for each catalog SQuirreL will show all of the schemas from all of the catalogs. If a schema from another catalog is clicked on and then the TABLE icon is clicked on, nothing will be shown for the tables in that schema. This can be confusing to the user, and for this reason, a special version of SQuirreL has been created for RapidsDB that addresses this issue and displays the catalog and schema information correctly. Contact your RapidsDB representative to obtain this modified version of SQuirreL. Figure 5 below shows an example of the RapidsDB metadata information correctly displayed using the modified version of SQuirreL.



Property Name	Value
catalogName	HADOOP
childTables	<null>
exportedKeys	<null>
importedKeys	<null>
qualifiedName	"HADOOP"."PUBLIC"."CUSTOMER"
remarks	
schemaName	PUBLIC
simpleName	CUSTOMER
type	TABLE

RapidsDB has two catalogs:
1) HADOOP, with a schema named PUBLIC
2) RAPIDS, with a schema named SYSTEM

Note that the Objects panel is correctly displaying the catalogs and schemas, with no duplicate schema names

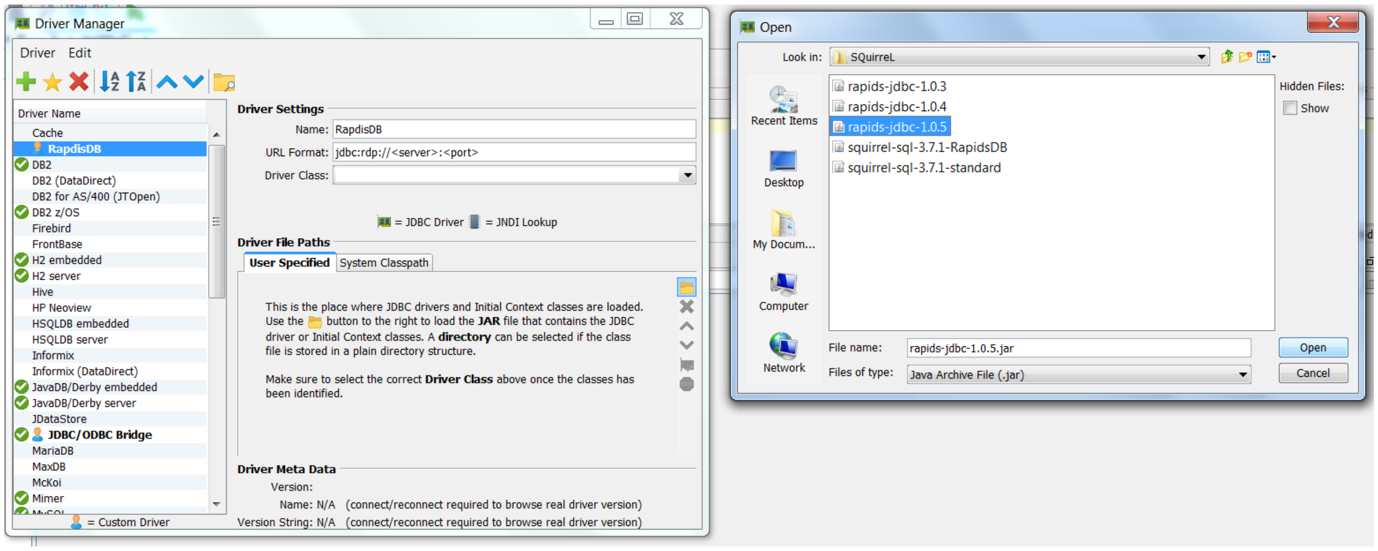
Figure 5. Modified Squirrel Display of RapidsDB Metadata.

7.3 DBVisualizer (https://www.dbvis.com/)

The first step is to create a Driver for RapidsDB:

Under the menu Tools/Driver Manager/Driver/Create Driver

- At “Name:” enter a name for the RapidsDB Unified JDBC Driver, e.g. “RapidsDB”.
- At “URL Format:” enter "jdbc:rdp://<server>:<port>"
- At “Driver File Paths”, click on User Specified, and then click on the folder icon on the right-hand side and navigate to the folder where the RapidsDB Unified JDBC Driver.jar file can be found and click on the jar file and then click “Open”. See Figure 6 below.



Name: RapdisDB
URL Format: jdbc:rdp://<server>:<port>
Driver Class:

This is the place where JDBC drivers and Initial Context classes are loaded. Use the button to the right to load the **JAR** file that contains the JDBC driver or Initial Context classes. A **directory** can be selected if the class file is stored in a plain directory structure.

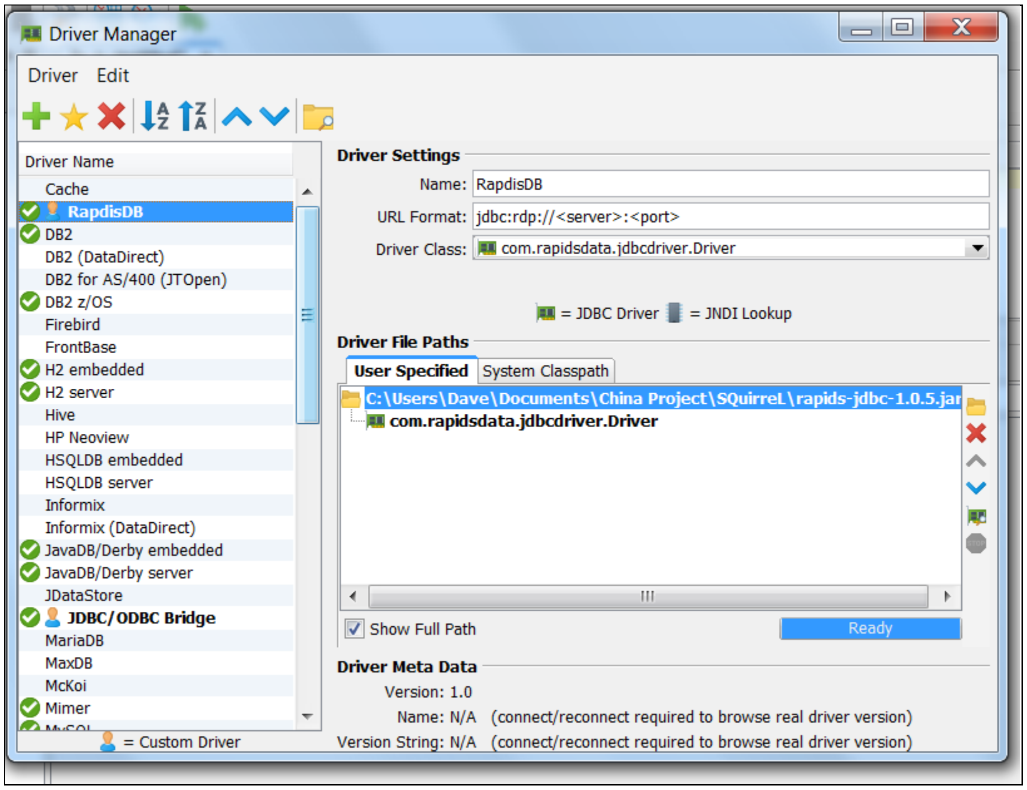
Make sure to select the correct **Driver Class** above once the classes has been identified.

Version:
Name: N/A (connect/reconnect required to browse real driver version)
Version String: N/A (connect/reconnect required to browse real driver version)

File name: rapids-jdbc-1.0.5.jar
Files of type: Java Archive File (.jar)

Figure 6. DBVisualizer – Adding a Driver(1)

- After clicking open the Driver Class filed in the Driver Settings should be filled in and the Ready button should be shown (see Figure 7 below). The Driver has been added at this point and the Driver Manager box can be closed.



Driver Settings

Name: RapdisDB

URL Format: jdbc:rdp://<server>:<port>

Driver Class: com.rapidsdata.jdbcdriver.Driver

Driver File Paths

C:\Users\Dave\Documents\China Project\SQuirreL\rapids-jdbc-1.0.5.jar

com.rapidsdata.jdbcdriver.Driver

Driver Meta Data

Version: 1.0

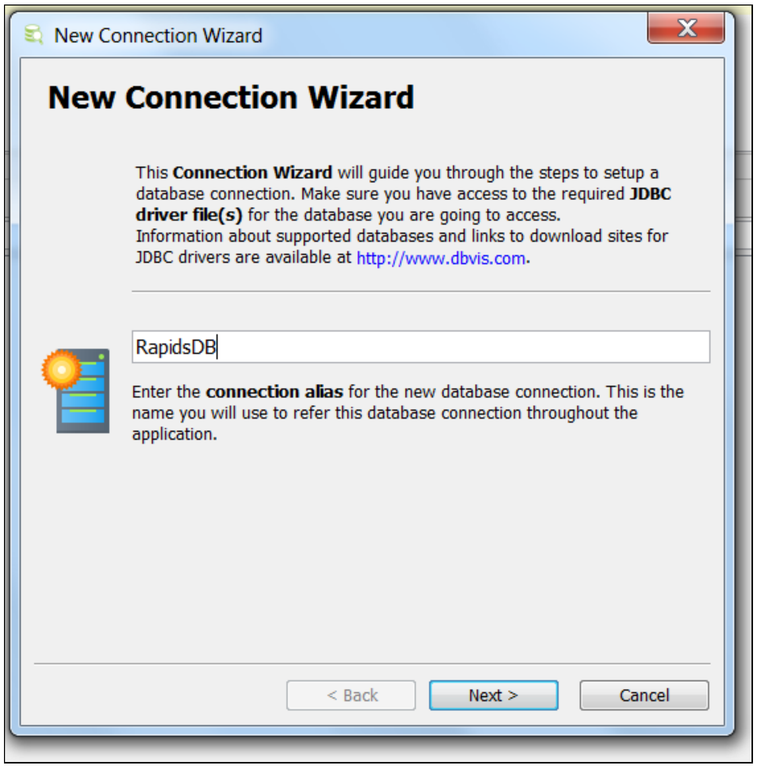
Name: N/A (connect/reconnect required to browse real driver version)

Version String: N/A (connect/reconnect required to browse real driver version)

Figure 7. DBVisualizer – Adding a Driver (2)

The second step is to create a database connection to RapidsDB:
Under the menu Database

- Select “Create Database Connection” and select “Use Wizard” and then enter a connection name e.g. “Rapids”, press “Next”. See Figure 8 below:



New Connection Wizard

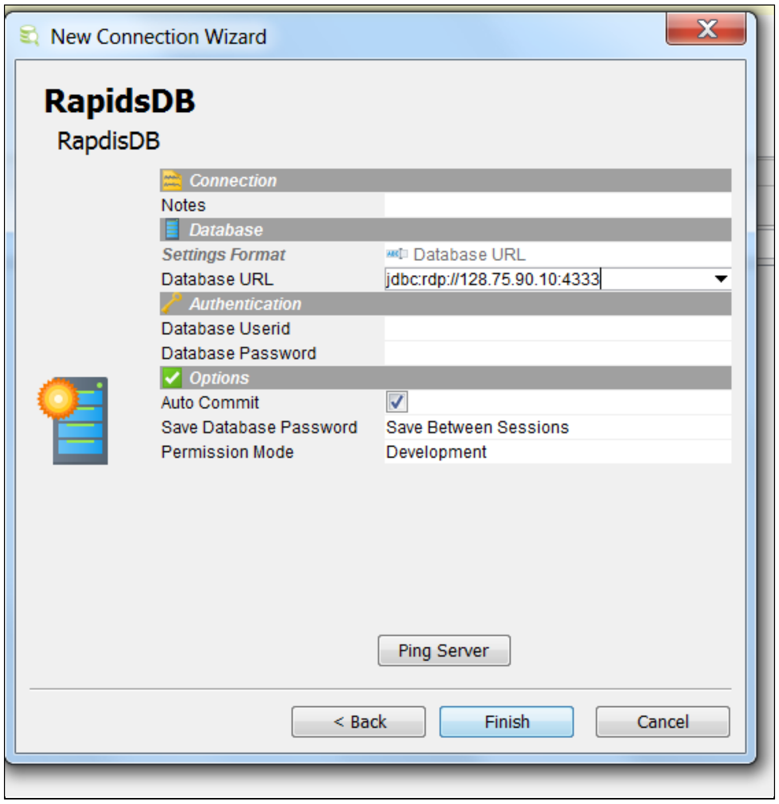
This **Connection Wizard** will guide you through the steps to setup a database connection. Make sure you have access to the required **JDBC driver file(s)** for the database you are going to access.
Information about supported databases and links to download sites for JDBC drivers are available at http://www.dbvis.com.

RapidsDB

Enter the **connection alias** for the new database connection. This is the name you will use to refer this database connection throughout the application.

Figure 8. DBVisualizer – Connection Wizard (1)

- At "Select Database Driver" dropdown, select "RapidsDB" or the name that you entered at the creation step above, click "Next".
- At the connection screen "Database URL", enter "jdbc:rdp://<address>:<port>"
- Leave everything else blank, click "Finish". See Figure 9 below.



RapidsDB

RapdisDB

Connection	
Notes	
Database	
Settings Format	Database URL
Database URL	jdbc:rdp://128.75.90.10:4333
Authentication	
Database Userid	
Database Password	
Options	
Auto Commit	✓
Save Database Password	Save Between Sessions
Permission Mode	Development

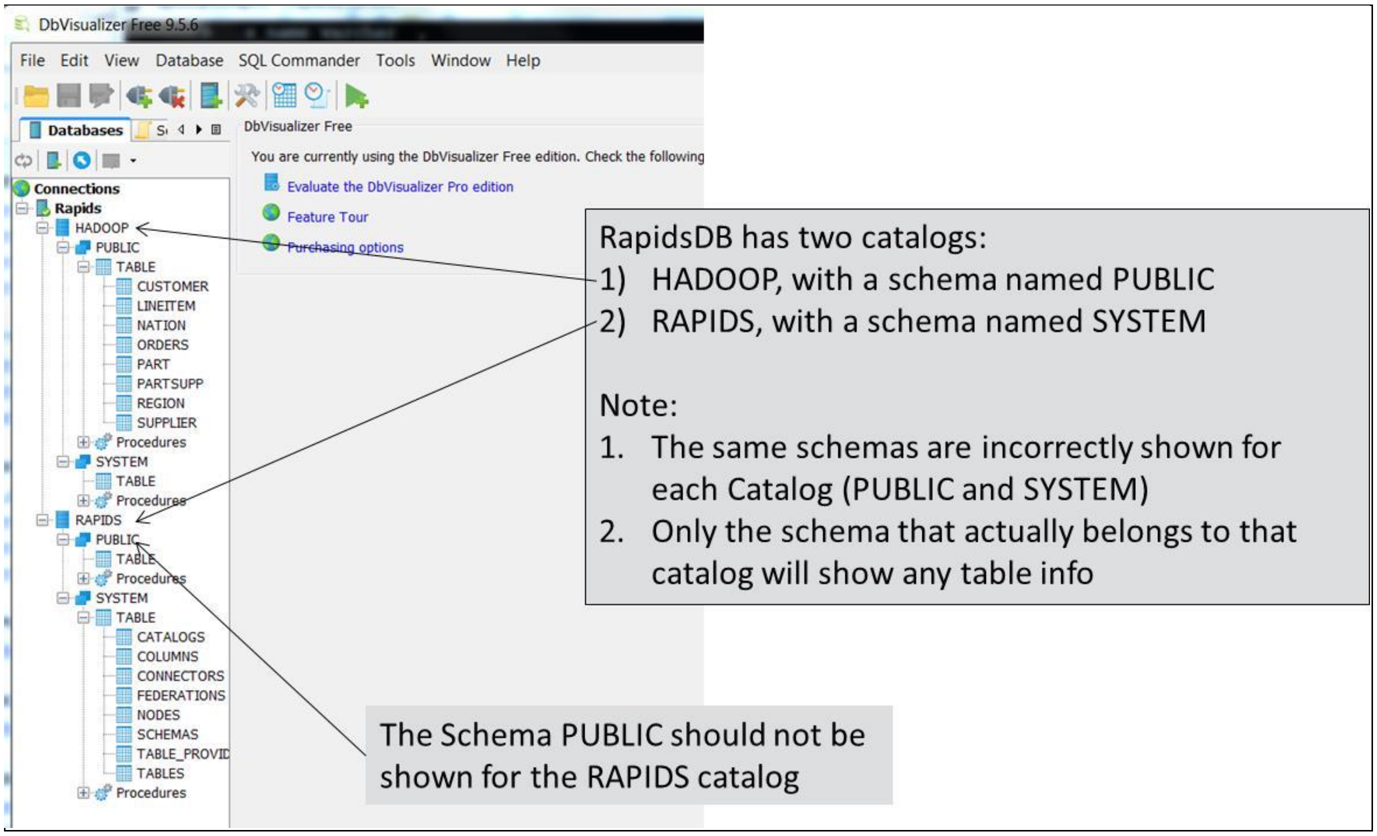
Figure 9. DBVisualizer – Connection Wizard (2)

Connect to RapidsDB

- Select the connection name created above, e.g. "RapidsDB", right-click and select "Connect".

NOTE:

1. DBVisualizer does not handle systems such as RapidsDB that have multiple catalogs each with their own schema. When displaying the metadata for RapidsDB, DBVisualizer will correctly display all of the catalogs, but for each catalog DBVisualizer will show all of the schemas from all of the catalogs. If a schema from another catalog is clicked on and then the TABLE icon is clicked on, nothing will be shown for the tables for that schema. Figure 10 below shows an example of this problem.



RapidsDB has two catalogs:
1) HADOOP, with a schema named PUBLIC
2) RAPIDS, with a schema named SYSTEM

Note:
1. The same schemas are incorrectly shown for each Catalog (PUBLIC and SYSTEM)
2. Only the schema that actually belongs to that catalog will show any table info

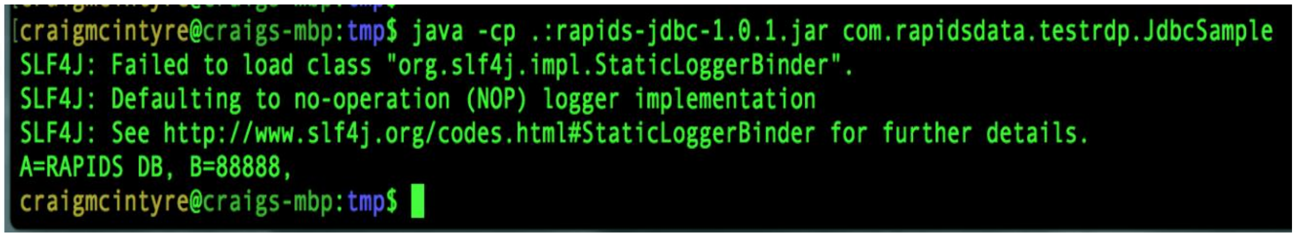
The Schema PUBLIC should not be shown for the RAPIDS catalog

Figure 10. DBVisualizer Sample Screen

8. Debugging

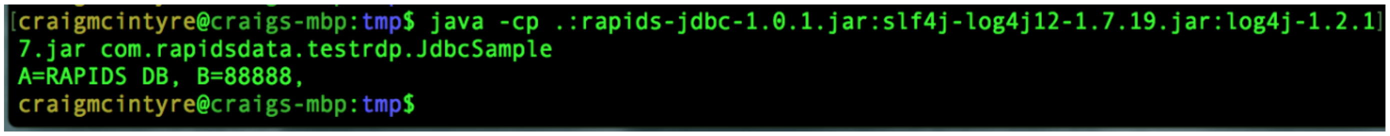
8.1 JDBC Driver

The JDBC driver does not contain a logging framework, as this would impose a constraint on any client that uses it. Instead, the driver uses the SLF4J logging API, which has bindings with many popular logging frameworks. So it is up to the client application that is using the JDBC driver to provide a logging framework that can bind to SLF4J. Failure to do so will result in some warning messages appearing on the console about the lack of a logging framework being supplied:



```
[craigmcintyre@craigs-mbp:tmp$ java -cp .:rapids-jdbc-1.0.1.jar com.rapidsdata.testrdp.JdbcSample
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
A=RAPIDS DB, B=88888,
craigmcintyre@craigs-mbp:tmp$
```

And here is an example where the client application has provided a jar file for binding SLF4J to Log4J, as well as a jar file for the implementation of Log4J. This enables all the logging that occurs within the JDBC driver to be directed through the Log4J framework:


```
craigmcintyre@craigs-mbp:tmp$ java -cp .:rapids-jdbc-1.0.1.jar:slf4j-log4j12-1.7.19.jar:log4j-1.2.1
7.jar com.rapidsdata.testrdp.JdbcSample
A=RAPIDS DB, B=88888,
craigmcintyre@craigs-mbp:tmp$
```

Once a logging framework is in place, the driver can be debugged by setting the log level to TRACE or DEBUG (where TRACE is an even more verbose level than DEBUG) for all the classes in the com.rapidsdata.jdbcdriver namespace. How this is done will vary greatly with the logging framework used, as well as the user's preferred way of configuring the logging framework.

As one possible example, if the user is using Log4J then the JDBC driver can be debugged by providing a file, log4j.properties, on the application's classpath with the following contents:

Example log4j.properties file with tracing enabled for the RapidsDB Unified JDBC Driver:

```
# By default, log INFO or higher to console
log4j.rootLogger=INFO,consoleLogger

# Output ALL logging messages from the RapidsDB JDBC driver
log4j.logger.com.rapidsdata.jdbcdriver=TRACE

# A logger for sending messages to the console
log4j.appender.consoleLogger=org.apache.log4j.ConsoleAppender
log4j.appender.consoleLogger.layout.ConversionPattern=%d{ISO8601} [%-22c{2}] %-5p: %m%n
log4j.appender.consoleLogger.layout=org.apache.log4j.EnhancedPatternLayout
```

Below is some sample debug output from the JDBC driver:

```
craigmcintyre@craigs-mbp:tmp$ java -cp .:rapids-jdbc-1.0.1.jar:slf4j-log4j12-1.7.19.jar:log4j-1.2.17.jar
com.rapidsdata.testrdp.JdbcSample
2016-12-30 19:37:41,900 [aj.NamedThreads       ] INFO : jcabi-aspects 0.22.5/4a18718 started new
daemon thread jcabi-loggable for watching of @Loggable annotated methods
2016-12-30 19:37:41,911 [jdbcdriver.Driver     ] TRACE: #getVersion(): entered
2016-12-30 19:37:41,917 [jdbcdriver.Driver     ] TRACE: #getVersion(): '1.0.1' in 1.99ms
2016-12-30 19:37:41,918 [jdbcdriver.Driver     ] TRACE: #connect('jdbc:rdp://localhost:4333', '{user=,
password=}'): entered
2016-12-30 19:37:41,952 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): entered
2016-12-30 19:37:41,953 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): 1 in 550.04µs
2016-12-30 19:37:41,970 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): entered
2016-12-30 19:37:41,971 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): 1 in 109.03µs
2016-12-30 19:37:42,044 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): entered
2016-12-30 19:37:42,045 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): 1 in 255.97µs
2016-12-30 19:37:42,045 [jdbcdriver.JdbcConnection] INFO : serverName:localhost cat:null
databaseName: user: password:
2016-12-30 19:37:42,233 [jdbcdriver.JdbcConnection] TRACE: #setFederation('DEFAULTFED'): entered
2016-12-30 19:37:42,234 [jdbcdriver.JdbcConnection] TRACE: #setFederation('DEFAULTFED'): in
493.89µs
```

```
2016-12-30 19:37:42,235 [jdbcdriver.Driver     ] TRACE: #connect('jdbc:rdp://localhost:4333', '{user=, 
password=}'): com.rapidsdata.jdbcdriver.JdbcConnection@2f686d1f in 316.73ms
2016-12-30 19:37:42,235 [jdbcdriver.JdbcConnection] TRACE: #createStatement(): entered
2016-12-30 19:37:42,236 [jdbcdriver.JdbcConnection] TRACE: #createStatement(1004, 1007): entered
2016-12-30 19:37:42,248 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): entered
2016-12-30 19:37:42,249 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): 1 in 137.39µs
2016-12-30 19:37:42,249 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): entered
2016-12-30 19:37:42,250 [jdbcdriver.JdbcConnection] TRACE: #getServerType(): 1 in 78.14µs
2016-12-30 19:37:42,253 [jdbcdriver.JdbcConnection] TRACE: #createStatement(1004, 1007): 
com.rapidsdata.jdbcdriver.JdbcStatement@3b0143d3 in 14.30ms
2016-12-30 19:37:42,253 [jdbcdriver.JdbcConnection] TRACE: #createStatement(): 
com.rapidsdata.jdbcdriver.JdbcStatement@3b0143d3 in 17.35ms
2016-12-30 19:37:42,254 [jdbcdriver.JdbcStatement] TRACE: #executeUpdate('INSERT INTO TABLE_A1 
VALUES ('RAPIDS DB', 88888);'): entered
2016-12-30 19:37:42,254 [jdbcdriver.JdbcStatement] TRACE: #executeUpdate('INSERT INTO TABLE_A1 
VALUES ('RAPIDS DB', 88888);', 2): entered
2016-12-30 19:37:42,254 [jdbcdriver.JdbcStatement] TRACE: #isClosed(): entered
2016-12-30 19:37:42,255 [jdbcdriver.JdbcStatement] TRACE: #isClosed(): false in 78.50µs
```

8.2 RapidsDB

At times, it may be necessary to debug the messages that are being sent and received across the wire line protocol from the RapidsDB server. Logging can be turned on that prints the messages sent and received by the server as well as their contents.

To enable this logging, edit the file cfg/log4j.dqx.properties on the RapidDB node that the JDBC driver has connected to and add the following line:

```
log4j.logger.com.rapidsdata.wirelineprotocol=TRACE
```

Then restart the RapidsDB cluster. The log messages should be written to the logfile specified in the log4j.dqx.properties file (typically dqx.log).

To disable logging, either remove the above line or change the log level on the above line from TRACE to INFO.

Below is some sample output generated from the logging:

```
2017-01-25 16:48:50,664 [messaging.BaseProtocolHandler] TRACE: Session (null): Read message header 
(type = 1 (ProtocolCompatibleRequest), payload length = 10 bytes) from client at /10.0.8.1:55504.
2017-01-25 16:48:50,666 [messaging.BaseProtocolHandler] TRACE: Session (null): Read and decoded 
message type 1 (ProtocolCompatibleRequest) from the client at /10.0.8.1:55504.
Message fields:  ProtocolCompatibleRequest(clientProtocolVersion:1, clientDescription:login)
2017-01-25 16:48:50,728 [messaging.BaseProtocolHandler] TRACE: Session (null): Sent message type 2 
(ProtocolCompatibleResponse) from server to client at /10.0.8.1:55504
.
```

```
 Message fields:  ProtocolCompatibleResponse(protocolVersion:1, serverSoftwareVersion:3.0-POC-14-
g57bc291, serverBuildInfo:Build information:
  version    3.0-POC-14-g57bc291
  from branch (no
  by         rapids
  on         localhost.localdomain
  at         Tue Jan 24 11:04:42 PST 2017
)
2017-01-25 16:48:51,159 [messaging.BaseProtocolHandler] TRACE: Session 1: Read message header
(type = 21 (ClearTextTunnelRequest), payload length = 1 bytes) from client at /10.0.8.1:55504.
2017-01-25 16:48:51,161 [messaging.BaseProtocolHandler] TRACE: Session 1: Read and decoded
message type 21 (ClearTextTunnelRequest) from the client at /10.0.8.1:55504.
Message fields:  ClearTextTunnelRequest()
2017-01-25 16:48:51,162 [messaging.BaseProtocolHandler] TRACE: Session 1: Sent message type 22
(TunnelSupportedResponse) from server to client at /10.0.8.1:55504.
 Message fields:  TunnelSupportedResponse()
2017-01-25 16:48:51,162 [messaging.ProtocolHandlerV1] DEBUG: Tunnel established for the client at
/10.0.8.1:55504
2017-01-25 16:48:51,590 [messaging.BaseProtocolHandler] TRACE: Session 1: Read message header
(type = 23 (AuthenticationRequest), payload length = 7 bytes) from client at /10.0.8.1:55504.
2017-01-25 16:48:51,592 [messaging.BaseProtocolHandler] TRACE: Session 1: Read and decoded
message type 23 (AuthenticationRequest) from the client at /10.0.8.1:55504.
Message fields:  AuthenticationRequest(username:root)
2017-01-25 16:48:51,593 [messaging.BaseProtocolHandler] TRACE: Session 1: Sent message type 26
(AuthenticationOkResponse) from server to client at /10.0.8.1:55504.
 Message fields:  AuthenticationOkResponse()
2017-01-25 16:48:51,593 [messaging.ProtocolHandlerV1] DEBUG: Authentication succeeded for the
client at /10.0.8.1:55504
```