C-JDBC User's Guide

Emmanuel Cecchet

Julie Marguerite

Mathieu Peltier

Nicolas Modrzyk

Version 2.0.2

Copyright © 2002, 2003, 2004, 2005 French National Institute For Research In Computer Science And Control (INRIA)Emic Networks

Java, and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Table of Contents

1. Getting Started 4

1.1. What is C-JDBC? 4
1.2. What do I need to use C-JDBC? 4
1.3. Why should I use C-JDBC? 4
1.4. How does it work? 4
1.5. What does it cost? 5
1.6. What kind of modifications are needed? 6

2. Getting the Sofware 6

3. Installation 6

3.1. C-JDBC Controller 6
3.1.1. Using the Java graphical installer 6
3.1.2. Using the binary distribution 7
3.2. C-JDBC Driver 8
3.3. C-JDBC out of the box 8

4. C-JDBC Driver 8

4.1. Overview 9
4.2. Loading the Driver 9
4.3. C-JDBC JDBC URL 9
4.3.1. URL options 9
4.4. Proxying mode 11
4.5. Getting a connection using a data source 11
4.6. Stored procedures 13
4.7. Blobs: Binary Large Objects 13
4.8. Clobs: Character Large Objects 14
4.9. ResultSet streaming 15
4.10. Current Limitations 15

5. Configuring C-JDBC with 3rd party software 16
5.1. Forenotes on configuring C-JDBC with your application 16
5.2. Configuring C-JDBC with Jakarta Tomcat 16
5.3. Configuring C-JDBC with JOnAS 16
5.4. Configuring C-JDBC with JBoss 16
5.5. Configuring C-JDBC with BEA Weblogic Server 7.x/8.x 17
5.6. Configuring C-JDBC with Hibernate 17
5.7. Using sequences with Hibernate, C-JDBC and PostgreSQL 18
6. C-JDBC controller 18
6.1. Design Overview 18
6.2. Starting the Controller 19
6.3. Writing the controller configuration file 20
6.3.1. Controller Parameters 20
6.3.2. Internationalization 21
6.3.3. Report 21
6.3.4. JMX 22
6.3.5. Virtual Database 23
6.3.6. Security 23
6.4. Configuring the Log 25
6.5. Recovery Log 26
6.5.1. A practical example 27
6.5.2. Understanding checkpoints 27
6.5.3. A fault tolerant Recovery Log 27
6.6. Controller replication 28
6.7. Current Limitations 29
7. Administration console 29
7.1. Jmx Notifications List 29
7.2. Starting the Administration Console 30
7.3. Console Quickstart 31
7.4. Console Main Menu 33
7.5. Administrator Menu 34
7.5.1. Administrator Standard Commands 34
7.5.2. Administrator Expert Commands 35
7.6. Automated Backup With Jmx 36
7.7. Recoverying from a failed controller in distributed mode 37
8. Virtual Database Console Menu 37
9. Monitoring Console Menu 38
10. RAIDb Basics 38
10.1. RAIDb Definition 39
10.2. RAIDb-0 39
10.3. RAIDb-1 39
10.4. RAIDb-2 39
10.5. Nested RAIDb Levels 39
11. Virtual database configuration 40
11.1. Writing a Virtual Database Configuration File 40
11.2. Virtual Database 42
11.2.1. Distribution 43
11.2.2. Monitoring 44
11.3. Backup Manager 45
11.4. Authentication Manager 45

11.5. Database Backend 46
11.5.1. Rewriting requests on backends 47
11.5.2. Database Schema Definition 48
11.5.3. Connection Manager 50
11.6. Request Manager 51
11.6.1. Macros Handler 52
11.6.2. Request Scheduler 53
11.6.3. Request Cache 53
11.6.4. Load Balancer 57
11.6.5. Recovery Log 61
11.7. SSL Configuration 65
11.7.1. Controller 65
11.7.2. Console / Jmx Clients 66
11.7.3. Driver 66
11.7.4. Certificates (public and private keys) 66
11.8. Configuration Examples 67
12. Request Player 67
12.1. Recording a request trace 67
12.2. Replaying a trace file 68
12.3. requestplayer.properties 68
13. Glossary 69
14. About C-JDBC 69
14.1. License 70
14.2. Web Site 70
14.3. Wiki 70
14.4. Mailing Lists 70
14.5. Reporting a Bug 70
14.6. Getting Involved 71
14.7. About INRIA 71
14.8. About ObjectWeb 71

1. Getting Started

1.1. What is C-JDBC?

C-JDBC is a database cluster middleware that allows any Java™ application (standalone application, servlet or EJB™ container, ...) to transparently access a cluster of databases through JDBC™. You do not have to modify client applications, application servers or database server software. You just have to ensure that all database accesses are performed through C-JDBC.

C-JDBC is a *free*, *open source* project of the ObjectWeb Consortium (http://www.objectweb.org/). It is licensed under the GNU Lesser General Public License (http://www.gnu.org/copyleft/lesser.html) (LGPL).

1.2. What do I need to use C-JDBC?

In order to use C-JDBC, you will need:

- a client application that accesses a database through JDBC,
- a JDK™ 1.3 (or greater) compliant Java Virtual Machine™ (JVM)[1],
- a database with a JDBC driver (type 1, 2, 3 or 4) or an ODBC driver used with the JDBC-ODBC bridge.
- a network supporting TCP/IP communications between your cluster nodes.

> **Note:** If your client application uses ODBC, it is possible to use an ODBC-JDBC bridge such as the unixODBC (http://www.unixodbc.org/) provided by Easysoft.

1.3. Why should I use C-JDBC?

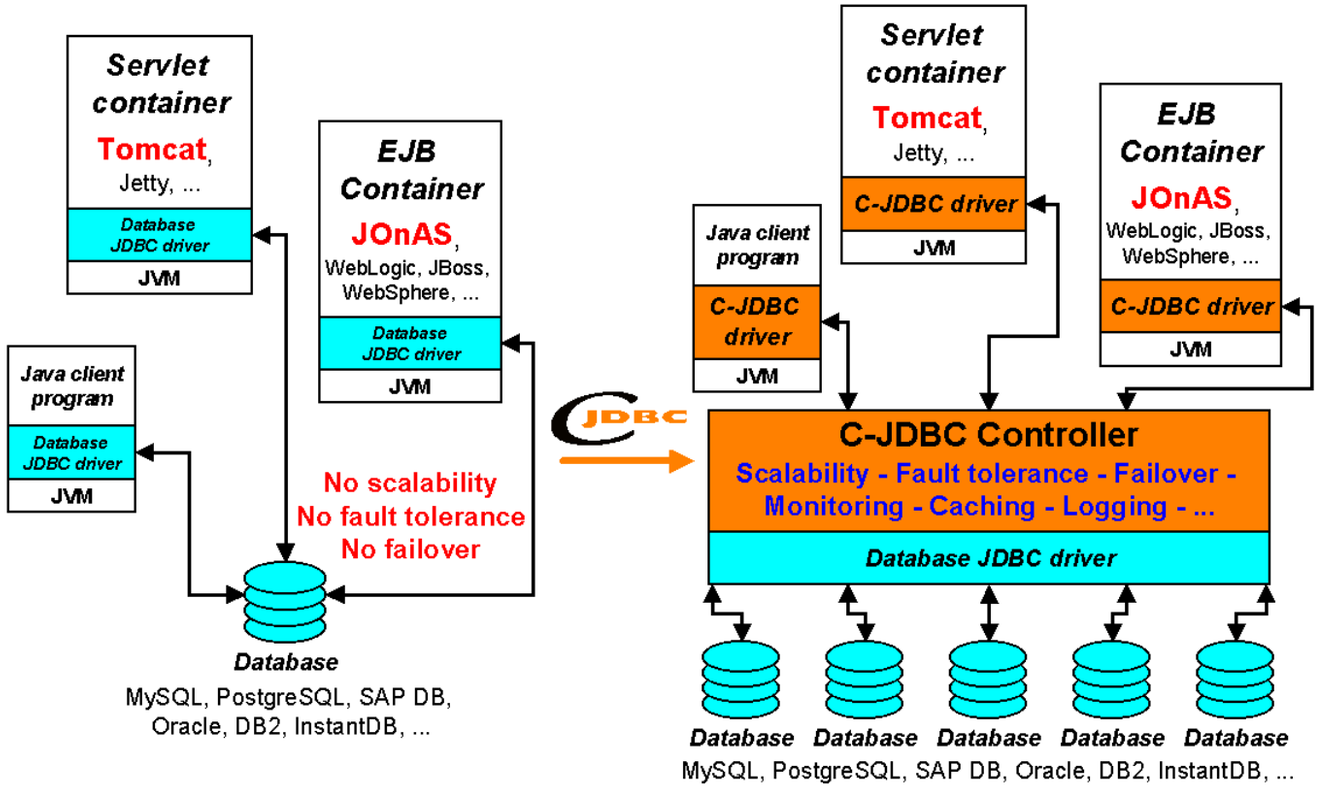
You have a Java application or a Java-based application server that accesses one or several databases. The database tier becomes the bottleneck of your application or it is a single point of failure or both. C-JDBC can help you resolve these problems by providing:

- performance scalability by adding database nodes and balacing the load among these nodes.
- high availability of the database tier, i.e. C-JDBC tolerates database crashes and offers transparent failover using database replication techniques.
- improved performance with fine grain query caching and transparent connection pooling.
- SQL traffic logging for performance monitoring and analysis.
- support for clusters of heterogenous database engines.

1.4. How does it work?

C-JDBC provides a flexible architecture that allows you to achieve scalability, high availability and failover with your database tier. C-JDBC implements the concept of RAIDb: *Redundant Array of Inexpensive Databases* (see Section 10). The database is distributed and replicated among several nodes and C-JDBC load balances the queries between these nodes.

Figure 1. C-JDBC principle



C-JDBC provides a generic JDBC driver to be used by the clients (see Section 4). This driver forwards the SQL requests to the C-JDBC controller (see Section 6) that balances them on a cluster of databases (reads are load balanced and writes are broadcasted). C-JDBC can be used with any RDBMS (Relational DataBase Management System) providing a JDBC driver, that is to say almost all existing open source and commercial databases. Figure 1 gives an overview of the C-JDBC principle.

C-JDBC allows to build any cluster configuration including mixing database engines from different vendors. The main features provided by C-JDBC are performance scalability, fault tolerance and high availability. Additional features such as monitoring, logging, SQL requests caching are provided as well.

The architecture is widely open to allow anyone to plug custom requests schedulers, load balancers, connection managers, caching policies, ...

1.5. What does it cost?

From a software point of view, C-JDBC is an open-source software licensed under LGPL which means that it is free of charge for any usage (personal or commercial). If you are using commercial RDBMS (such as Oracle, DB2, ...), you will have to buy extra licenses for the nodes where you install replicas of the database. But you can possibly use open-source databases to host replicas of your main database.

You need to buy extra machines if you want more performance and more fault tolerance. C-JDBC has been designed to work with standard off-the-shelf workstations because it primarily targets low cost open-source solutions but it can work as well with large SMP machines. A standard Ethernet network is sufficient to achieve good performance.

1.6. What kind of modifications are needed?

You *do not have to change anything* to your application or your database.

You only have to update the JDBC driver configuration used by your application (usually it is just a configuration file update) and to setup a C-JDBC configuration file (see Section 11).

2. Getting the Sofware

The binary distribution of C-JDBC can be downloaded from C-JDBC's Web site (http://c-jdbc.objectweb.org/). It mainly contains the JAR files for the C-JDBC driver and controller and also the documentation and other tools such as the C-JDBC administration console.

> **Note:** A source distribution of C-JDBC is also available. The whole code base can also be downloaded through an anonymous CVS server[2]. For more information, please refer to C-JDBC Developper's Guide. Most users will only need the binary distribution.

The following formats are available (where `x.y` is the C-JDBC release number):

- `c-jdbc-x.y-bin-installer.jar`: Java graphical installer (powered by IzPack (http://www.izforge.com/izpack/)).
- `c-jdbc-x.y-bin.tar.gz`: binary distribution for the Unix platforms users.
- `c-jdbc-x.y-bin.zip`: binary distribution for the Windows platforms users.
- `rpm` packages are also available from JPackage (http://www.jpackage.org/).

We strongly advice to use the Java installer package since it automatically configures the scripts to suit your system configuration.

> **Note:** All distributions contain the user documentation.

3. Installation

3.1. C-JDBC Controller

3.1.1. Using the Java graphical installer

The easiest way to install C-JDBC is to use the Java graphical installer. A Java Virtual Machine is of course needed in this case.

- Unix users can simply launch the installation program by typing:

```
bash> java -jar c-jdbc-x.y.bin-installer.jar
```

- Windows users can use the same command or just double-clik on the JAR installation file if your JRE has been properly installed.

3.1.2. Using the binary distribution

If you want to use the other distribution formats (for example if you have not installed a JVM or if you can not launch a graphical application), you have to uncompress the downloaded file in the directory of your choice, and then set the `CJDBC_HOME` environment variable.

> **Note:** If you are using the Java installer, you do not need to set any environment variable since the installer customizes the scripts with the installation path.

To set the `CJDBC_HOME` environment variable, you can proceed as follows:

- Unix users can proceed as follows:

```
bash> mkdir -p /usr/local/c-jdbc
bash> cd /usr/local/c-jdbc
bash> tar xfz /path-to-c-jdbc-bin-dist/c-jdbc-x.y-bin.tar.gz
bash> export CJDBC_HOME=/usr/local/c-jdbc
```

> **Note:** In this example, we assume you install C-JDBC in the `/usr/local/c-jdbc` directory.

  You can modify your shell configuration file (`.bashrc`, `.cshrc`, ...) to set the environment variable permanently.

- Windows users have to use an utility such as WinZip (http://www.winzip.com/) to extract the files from the archive. Then, to set the `CJDB_HOME` variable, do the following according to your Windows version:
  - *Windows 95 or 98*: you must insert the following line in the `AUTOEXEC.BAT` file:

    ```
    set CJDBC_HOME="C:\Program Files\C-JDBC"
    ```

  - *Windows Me*: go to the "Start Menu", then choose "Programs", "Accessories", "System Tools" and "System Information". A window titled "Microsoft Help and Support" should appear. Select the "Tools" menu, and choose the "System Configuration Utility". Go to the "Environment" and click on the "New" button. Enter `CJDBC_HOME` in the "Variable Name" field and `"C:\Program Files\C-JDBC"` in "Variable Value". Once you have changed and saved the value, you will be prompted for reboot.
  - *Windows NT*: go to the "Start Menu", then choose "Settings", "Control Panel" and select "System". Select the "Environment" tab and click on the "New" button. Enter `CJDBC_HOME` in the "Variable Name" field and `"C:\Program Files\C-JDBC"` in "Variable Value".
  - *Windows 2000*: go to the "Start Menu", then choose "Settings", "Control Panel" and select "System". Select the "Advanced" tab and click on the "New" button. Enter `CJDBC_HOME` in the "Variable Name" field and `"C:\Program Files\C-JDBC"` in "Variable Value".
  - *Windows XP*: go to the "Start Menu", then double click on "System". In the "System Control Panel" select the "Advanced" tab and push the `Environment Variables` button. Click on the "New" button for

"System Variables". Enter `CJDBC_HOME` in the "Variable Name" field and `"C:\Program Files\C-JDBC"` in "Variable Value".

Note: In this example, we assume you install C-JDBC in the `C:\Program Files\C-JDBC` directory.

Note: Do not forget the quotes in the CJDBC_HOME environment variable definition else the starting scripts will fail with paths including spaces.

3.2. C-JDBC Driver

Once you have installed the C-JDBC controller, you will find the driver JAR file in the `drivers/` directory of the controller installation location.

To install the C-JDBC driver, you just have to add the `c-jdbc-driver.jar` file to the client application classpath. This driver replaces the database native driver in the client application. The database native driver will be used by the C-JDBC controller to access your database. Therefore, the C-JDBC driver and controller can be seen as a proxy between your application and your database native driver.

3.3. C-JDBC out of the box

Since version 1.0b13, you have access to a RAIDb-1 configuration of HyperSonic SQL databases, just by launching the **demo-raidb1.sh** or **demo-raidb1.bat** file from the `demo` directory in your C-JDBC installation.

This is especially useful if you are new to clustering, or new to C-JDBC. The setup used is as follows:

- 2 HyperSonic SQL databases are started on two different ports (9001 and 9002)
- An extra HyperSonic SQL database is started on port 9003 to be used as the recovery log database
- The C-JDBC controller is configured to load automatically a virtual database containing those two HyperSonic SQL backends. The controller startup configuration file is found in `CJDBC_HOME/config/controller/controller-raidb1.xml` and the virtual database configuration file is `CJDBC_HOME/config/virtualdatabase/hsqldb-raidb1.xml`.
- Once the RAIDb-1 configuration is loaded, you can connect to C-JDBC using Squirrel, a graphical SQL console bundled with C-JDBC. You can start Squirrel by using **squirrel.sh** or **squirrel.bat**.

Squirrel shows the C-JDBC driver and HSQL database driver, and links to databases. You can click on them to view the different data. The login to use for C-JDBC is `user` with an empty password. The login for both HSQL databases is `test` with an empty password.

4. C-JDBC Driver

4.1. Overview

The C-JDBC driver is a generic JDBC driver that is designed to replace any database specific JDBC driver that could be used by a client. The client only has to know on which node the C-JDBC controller is running and the name of the database to access. The C-JDBC driver implements most of the JDBC 2.0 interface and some functionalities from JDBC 3.0 such as the support for autogenerated keys.

Users reported successful usage of C-JDBC with the following RDBMS: Oracle®, PostgreSQL, MySQL, Apache Derby, IBM DB2®, Sybase®, SAP DB (MySQL MaxDB), HyperSonic SQL, Firebird, MS SQL Server and InstantDB.

4.2. Loading the Driver

The C-JDBC driver can be loaded as any standard JDBC driver from the client program using:

```
Class.forName("org.objectweb.cjdbc.driver.Driver");
```

Note: The `c-jdbc-driver.jar` file must be in the client classpath else the driver will fail to load.

4.3. C-JDBC JDBC URL

The JDBC URL expected for the use with C-JDBC is the following:
`jdbc:cjdbc://host1:port1,host2:port2/database`.

`host` is the machine name (or IP address) where the C-JDBC controller is running, `port` is the port the controller is listening for client connections.

At least one host must be specified but a list of comma separated hosts can be specified. If several hosts are given, one is picked up randomly from the list. If the currently selected controller fails, another one is automatically picked up from the list.

The port is optional is the URL and the default port number is 25322 (C-JDBC on the phone!) if it is omitted. Those two examples are equivalent:

```
DriverManager.getConnection("jdbc:cjdbc://localhost/tpcw");
DriverManager.getConnection("jdbc:cjdbc://localhost:25322/tpcw");
```

Examples using two controllers for fault tolerance:

```
DriverManager.getConnection("jdbc:cjdbc://c1.objectweb.org,c2.objectweb.org/tpcw");
DriverManager.getConnection("jdbc:cjdbc://localhost,remote.objectweb.org:2048/tpcw");
DriverManager.getConnection("jdbc:cjdbc://smpnode.com:25322,smpnode.com:1098/tpcw");
```

4.3.1. URL options

The C-JDBC driver accepts additional options to override the default behavior of the driver. The options are appended at the end of the C-JDBC URL after a question mark followed by a list of ampersands separated options. Here is an example:

```
DriverManager.getConnection("jdbc:cjdbc://host/db?user=me&password=secret")
```

Another option is to use semicolons to delimit the start of options and options themselves. Example:

```
DriverManager.getConnection("jdbc:cjdbc://host/db;user=me;password=secret")
```

The recognized options are:

- `booleanTrue`: String value to use in PreparedStatement.setBoolean(true), default is '1'.
- `booleanFalse`: String value to use in PreparedStatement.setBoolean(false), default is '0'.
- `connectionPooling`: By default the C-JDBC driver does transparent connection pooling on your behalf meaning that when connection.close() is called, the connection is not physically closed but rather put in a pool for reuse within the next 5 seconds. Set this to false if you do not want the driver to perform transparent connection pooling.
- `debugLevel`: Debug level that can be set to 'debug', 'info' or 'off' to display driver related information on the standard output. Default is off.
- `driverProcessed`: can be set to 'true or 'false', see Proxying mode below.
- `escapeBackslash`: Set this to false if you don't want to escape backslashes when performing escape processing of PreparedStatements, default is true.
- `escapeSingleQuote`: Set this to false if you don't want to escape single quotes (') when performing escape processing of PreparedStatements, default is true
- `escapeCharacter`: Character to prepend and append to the String values when performing escape processing of PreparedStatements, default is a single quote.
- `user`: user login
- `password`: user password
- `preferredController`: defines the strategy to use to choose a preferred controller to connect to.

  `jdbc:cjdbc://node1,node2,node3/myDB?preferredController=ordered` : Always connect to node1, and if not available then try to node2 and finally if none are available try node3.

  `jdbc:cjdbc://node1,node2,node3/myDB?preferredController=random`: Pickup a controller node randomly (default strategy)

  `jdbc:cjdbc://node1,node2:25343,node3/myDB?preferredController=node2:25343,node3` : Round-robin between node2 and node3, fallback to node1 if none of node2 and node3 is available.

  `jdbc:cjdbc://node1,node2,node3/myDB?preferredController=roundRobin`: Round robin starting with first node in URL.

- `retryIntervalInMs`: once a controller has died, the driver will try to reconnect to this controller every retryIntervalInMs to see if the backend is back online. The default is 5000 (5 seconds).

4.4. Proxying mode

By default, the C-JDBC driver interprets the PreparedStatement locally and forwards a pre-processed statement. The C-JDBC controller executes directly these statements on the backends as statements without recreating a whole PreparedStatement and re-calling all setXXX() methods on the PreparedStatement. The default setting is the one that consumes the less resources and carries the minimum information over the network.

However, it might happen that this behavior is not desired or that the C-JDBC driver interpretation is unsuitable for some data types or database specific syntax. Therefore, it is possible to make the driver act as a real proxy that will forward all setXXX() calls to the database native driver. This will usually result in slightly lower performance but better portability.

The proxying mode can be enabled for a connection by setting a specific variable named `driverProcessed` to false (default value if omitted is true). To enable PreparedStatement proxying in C-JDBC use a connection URL like this:

```
DriverManager.getConnection("jdbc:cjdbc://host/db?driverProcessed=false")
```

Note: since any optional blob encoding is performed by the driver, blob encoding is disabled by `driverProcessed=false`. If you ever encoded blobs, you CANNOT switch to `driverProcessed=false` anymore.

4.5. Getting a connection using a data source

Another way to use the C-JDBC driver is to use its `DataSource` implementation. Data sources have been introduced in JDBC 2.0 Standard Extension API and are also a part of JDBC 3.0. They use the Java Naming and Directory Interface (JNDI) to break the application dependence on the JDBC driver configuration (i.e., driver class name, machine name, port number, etc.). With a data source, the only thing an application has to know is the name assigned to the `DataSource` object in the `jdbc` naming subcontext of the JNDI namespace.

The example below registers a data source object with a JNDI naming service. It is typically used by an application server.

```
import org.objectweb.cjdbc.driver.DataSource;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
...
private final static String NAME = "jdbc/c-jdbc";
private final static String URL = "jdbc:cjdbc://localhost:25322/mysql";

// Initializing data source
DataSource ds = new DataSource();
ds.setUrl(URL);

// Get initial context
Context ctx;
```

```
try {
  ctx = new InitialContext();
} catch (javax.naming.NamingException _e) {
  ... // Naming exception
}

// Bind data source to a JNDI name
try {
  ctx.bind(NAME, ds);
} catch (javax.naming.NamingException _e) {
  ... // Naming exception
}
```

The `org.objectweb.cjdbc.driver.DataSource` class implements the `javax.sql.DataSource` JDBC 3.0 interface. The `setUrl` line initializes the data source properties (the URL in this case). The data source object is bound to a logical JNDI name by calling `ctx.bind()`. In the example above, the JNDI name specifies a "jdbc" subcontext and a "c-jdbc" logical name within this subcontext.

Once a data source object is registered to JNDI, it can be used by an application. The example below gets the data source using the JNDI naming service. Such a piece of code is typically a part of an application that uses JDBC.

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import java.sql.Connection;
import javax.sql.DataSource;
...
private final static String NAME = "jdbc/c-jdbc";

// Lookup for the data source object
try {
  Context ctx = new InitialContext();
  Object obj = ctx.lookup(NAME);
  if (null == obj) {
    ... // Something wrong: NAME not found
  }
  ctx.close( );
} catch (javax.naming.NamingException _e) {
  ... // Naming exception
}

// Get a new JDBC connection
try {
  DataSource ds = (DataSource) obj;
  Connection conn = ds.getConnection("user", "c-jdbc");
  ... // Use of the connection retrieved
  ...
} catch (SQLException _e) {
  ... // SQL exception
}
```

The `ctx.lookup()` line in the example uses the retrieved initial JNDI naming context to do a lookup using the data source logical name. The method returns a reference to a Java object which is then narrowed to a `javax.sql.DataSource` object. Such an object can be then used to open a new JDBC connection by invoking one of its `getConnection()` methods. The application code is completely independent of the driver details,

such as the `Driver` class name, URL, etc. (the user name and password used by the connection can be also set by the application server - look at the C-JDBC javadoc documentation for more details). The only information a JDBC application has to know is the logical name of the data source object to use.

> **Note:** The URL used for the C-JDBC data source is the same as for the `Driver` decribed in the previous section.

4.6. Stored procedures

Stored procedures are supported by C-JDBC since version 1.0b6. Note that C-JDBC only support calls in the form **{call <procedure-name>[<arg1>,<arg2>, ...]}** but does not support **{? = call <procedure-name>[<arg1>,<arg2>, ...]}**.

A call to a stored procedure is systematically broadcasted to all backends since there is no way to know if the stored procedure will update the database or not. Therefore, the query cache (see Section 11.6.3), is completely flushed on every stored procedure call. To prevent cache flushing, the user can force the connection to read-only before calling the stored procedure. But never set a connection to read-only when calling a stored procedure that updates the database. If C-JDBC detects a read-only connection, it will not flush the cache. However, the call will still be broadcasted to all nodes resulting in duplicated jobs on each backend. Here is an example on how to prevent cache flushing when calling a stored procedure that does only read-only:

```
...
CallableStatement cs = connection.prepareCall("{call myproc(?)}");
cs.setString(1,"parameter1");
// Force no cache flush
connection.setReadOny(true);
// Call the stored procedure without flushing the cache ...
ResultSet rs = cs.executeQuery();
```

In the case of horizontal scalability, only read-only stored procedures are not broadbasted. All other stored procedures returning an int or a ResultSet are executed by all backends at all controllers.

> **Note:** It is not allowed to set a connection to read-only in the middle of a transaction. If you need to set a connection to read-only, you must do so before starting the transaction.

4.7. Blobs: Binary Large Objects

Binary large objects can now be stored using the C-JDBC driver since 1.0b10. Data is encoded into hexadecimal to be portable accross database engines.

FIXME: this whole section is outdated and should be re-written. Blob encoding is now configured in controller's DTD and set to "none" by default. It requires `driverProcessed=true`, which is the default.

- The column type used to store large objects with `MySQL` is `text`.
- The column type used to store large objects with `PostgreSQL` is `bytea`.

You should not have to change your code for storing blobs into your database, but previous blobs have to be converted to their hexadecimal form. You can use Octopus (http://octopus.objectweb.org) to perform this transformation.

Please refer to the following lines of code for storing and retrieving of large objects:

```
// In the code below:
// The signature of the readBinary method is:
// byte[] readBinary(File file) throws IOException
// it just read a file, and convert its content into an array of bytes

// Store file in database
File fis = new File(storeFile);
query = "insert into ... values(...,?)";
ps1 = con.prepareStatement(query);
if (callBlobMethods)
{
  org.objectweb.cjdbc.driver.Blob bob =
    new org.objectweb.cjdbc.driver.Blob(readBinary(fis));
  ps1.setBlob(1, bob);
}
else
{
  ps1.setBytes(1, readBinary(fis));
}
ps1.executeUpdate();
// Read File from database
query = "select * from ... where id=...";
ps1 = con.prepareStatement(query);
ResultSet rs = ps1.executeQuery();
rs.first();
byte[] lisette;
if (callBlobMethods)
{
  Blob blisette = rs.getBlob("blobcolumnname");
  lisette = blisette.getBytes((long) 1, (int) blisette.length());
}
else
{
  lisette = rs.getBytes("blobcolumnname");
}
```

4.8. Clobs: Character Large Objects

CLOB is a built-in type that stores a Character Large Object as a column value in a row of a database table. By default drivers implement Clob using an SQL locator (CLOB), which means that a Clob object contains a logical pointer to the SQL CLOB data rather than the data itself. A Clob object is valid for the duration of the transaction in which it was created.

Clobs in C-JDBC are handled like strings. You can refer to the section of code below to make good usage of clobs. This code is part of the C-JDBC test suite.

```
String clob = "I am a clob";
ps = con.prepareStatement("insert into ... values(...,?)");
ps.setString(1, clob);
```

```
ps.executeUpdate();

// Test retrieval
String ret;
ps = con.prepareStatement("Select * from ... where id=...");
rs = ps.executeQuery();
rs.first();
clob = rs.getClob("name");
ret = clob.getSubString((long) 0, (int) clob.length());
```

4.9. ResultSet streaming

In its default mode, when a query is executed on a backend, C-JDBC makes a copy of the backend's native ResultSet into a C-JDBC serializable ResultSet. If the result contains many rows or very large objects, the controller might run out of memory when trying to copy the whole ResultSet.

Since C-JDBC 1.0rc6, it is possible to fetch ResultSets by blocks using the Statement.setFetchSize(int rows) method. In this case, the ResultSet will be copied by block of rows and returned when needed by the client. Note that the current implemtation only allows to fetch forward streamable ResultSet, which basically means that you are only allowed to call ResultSet.next() on a streamable ResultSet.

C-JDBC will try to call setFetchSize() on the backend's driver to let the backend driver also perform the necessary optimizations. However, some driver requires a prior call to setCursorName() in which case you will also have to call setCursorName() on C-JDBC to pass it to the backend's driver.

A typical usage of the ResultSet streaming feature is as follows:

```
...
Connection con = getCJDBCConnection();
con.setAutocommit(false);
Statement s = con.createStatement();
s.setCursorName("cursor name");
s.setFetchSize(10);
rs = s.executeQuery(sql);
while (rs.next())
{ // Every 10 calls, C-JDBC will transfer a new block of rows
  XXX o = rs.getXXX("some column name");
}
...
con.commit();
```

Note: Streamable ResultSets are not cacheable. The result cache automatically detects this kind of ResultSet and does not keep them in the cache. However, as database specific ResultSets are copied into C-JDBC ResultSets, the memory footprint of the fetched blocks will be twice the one obtained without C-JDBC. If you have memory restrictions, you can reduce your fetch size by half to reduce the memory footprint of streamed ResultSets.

Streamable ResultSets do not work properly in autocommit mode as the connection used for retrieving the ResultSet is handed back to the pool. The workaround is to always encapsulate the query in a transaction. Note that databases such as PostgreSQL do not support streamable ResultSets in autocommit mode as well.

4.10. Current Limitations

The C-JDBC driver currently does not support the following features:

- java.sql.Array and java.sql.Ref types,
- Custom type mapping using java.sql.Connection.setTypeMap(java.util.Map map),
- `XAConnections` (look at the XAPool project (http://xapool.experlog.com) for XA support with C-JDBC),
- CallableStatements with OUT parameters,
- Streamable ResultSets do not work in autocommit mode.

5. Configuring C-JDBC with 3rd party software

5.1. Forenotes on configuring C-JDBC with your application

If the application you are using C-JDBC with requires a mapper, the best thing to do is to configure the mapping to be that of C-JDBC's underlying databases. For example, if you were using JBoss with PostgreSQL , then using C-JDBC on top of the PostgreSQL backends with JBoss would imply to still use the mapping for PostgreSQL while plugging the application server to C-JDBC (using C-JDBC's driver and C-JDBC's url).

5.2. Configuring C-JDBC with Jakarta Tomcat

Copy the `c-jdbc-driver.jar` file to the `lib` directory of your web application (for example: `$TOMCAT_HOME/webapps/mywebapp/WEB-INF/lib`).

There are many ways to obtain connections from a Tomcat application. Just ensure that you are using `org.objectweb.cjdbc.driver.Driver` as the driver class name and that the JDBC URL is a C-JDBC URL (see Section 4.3).

5.3. Configuring C-JDBC with JOnAS

The `c-jdbc-driver.jar` file must be found in the JOnAS CLASSPATH.

Here is an example of a `cjdbc.properties` file to store in JONAS 3.x `conf` directory (use the `config` directory for JOnAS 2.x):

```
###################### C-JDBC DataSource configuration example #
datasource.name      jdbc_1
datasource.url       jdbc:cjdbc://someMachine/someDatabase
datasource.classname org.objectweb.cjdbc.driver.Driver
datasource.username  your-username
datasource.password  your-password
```

5.4. Configuring C-JDBC with JBoss

Copy the `c-jdbc-driver.jar` file to `$JBOSS_DIST/server/default/lib` for JBoss 3.x or to `$JBOSS_DIST/jboss/lib/ext` for JBoss 2.x.

Here is an example of a datasource configuration file to be used with JBoss:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ===================================================================== -->
<!--                                                                       -->
<!-- JBoss Server Configuration                                            -->
<!--                                                                       -->
<!-- ===================================================================== -->

<!-- ===================================================================== -->
<!-- Datasource config for C-JDBC                                          -->
<!-- ===================================================================== -->
<datasources>
  <local-tx-datasource>
    <jndi-name>cjdbc-DS</jndi-name>
    <connection-url>jdbc:cjdbc://localhost:25322/lscluster</connection-url>
    <driver-class>org.objectweb.cjdbc.driver.Driver</driver-class>
    <user-name>user</user-name>
    <password>tagada</password>
  </local-tx-datasource>
</datasources>
```

5.5. Configuring C-JDBC with BEA Weblogic Server 7.x/8.x

Place the `c-jdbc-driver.jar` file in the classpath of the Weblogic Server.

Here is an example of a connection pool configuration for use with Weblogic:

```
<JDBCConnectionPool
    DriverName="org.objectweb.cjdbc.driver.Driver"
    InitialCapacity="1"  MaxCapacity="15"
    Name="cjdbcPool" Properties="user=username;password=password"
    ShrinkingEnabled="true" SupportsLocalTransaction="true"
    Targets="wlservername" URL="jdbc:cjdbc://192.168.0.1/vdb"
    XAPreparedStatementCacheSize="0"/>
```

Next, create the required `TXDataSources`:

```
<JDBCTxDataSource EnableTwoPhaseCommit="true"
  JNDIName="cjdbc-DS" Name="C-JDBC TX Data Source"
  PoolName="cjdbcPool" RowPrefetchEnabled="true" Targets="wlservername"/>
```

5.6. Configuring C-JDBC with Hibernate

C-JDBC just has to be defined as any JDBC driver in Hibernate, leaving the syntax set to the proper database. Here is a configuration example to use Hibernate with a C-JDBC cluster made of Sybase backends:

```
## C-JDBC
hibernate.dialect                 net.sf.hibernate.dialect.SybaseDialect
hibernate.connection.driver_class org.objectweb.cjdbc.driver.Driver
hibernate.connection.username     user
hibernate.connection.password     pass
hibernate.connection.url          jdbc:cjdbc://localhost:25322/test
```

5.7. Using sequences with Hibernate, C-JDBC and PostgreSQL

Our Hibernate dialect is as follows:

```
import net.sf.hibernate.dialect.PostgreSQLDialect;
public class CJDBCPostgreSQLDialect extends PostgreSQLDialect
{
  public String getSequenceNextValString(String sequenceName)
  {
    return "{call nextval('"+sequenceName+"')}";
  }
}
```

We simply extend the default PostgreSQL Dialect and override the getSequenceNextValString() method and tell it to use "{call ..." so that all the sequences in the cluster get incremented.

We then changed our Hibernate conf file to user to our custom dialect instead of net.sf.hibernate.dialect.PostgreSQLDialect.

6. C-JDBC controller

6.1. Design Overview

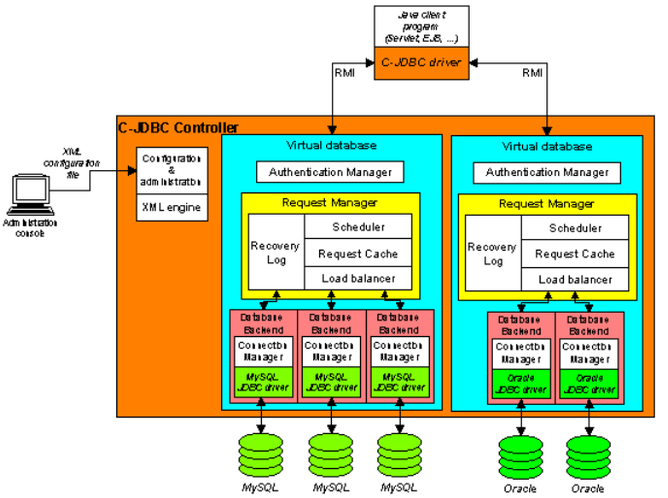
The C-JDBC controller is made of several components as shown in Figure 2. The controller hosts *virtual databases*. A *virtual database* gives the illusion of a single database to the user. It exports the same database name and login/password as those used in the client application. Therefore the client application can run unmodified with C-JDBC.

When the client application connects to the database using an URL like `jdbc:cjdbc://host:25322/myDB`, the C-JDBC driver tries to connect to a C-JDBC controller running on port `25322` on node `host`. Once the connection is established the login and password are sent with the `myDB` database name to be checked by the controller.

A virtual database contains the following components:

- *authentication manager*: it matches the virtual database login/password (provided by the application to the C-JDBC driver) with the real login/password to use on each backend. The authentication manager is only involved at connection establishment time.
- *backup manager*: manages a list of generic or database specific Backupers that are in charge of performing database dump and restore operation. Backupers should also take careof transferring dumps from one controller to another.

Figure 2. C-JDBC controller design overview



- *request manager*: it handles the requests coming from a connection with a C-JDBC driver. It is composed of several components:
  - *scheduler*: it is responsible for scheduling the requests. Each RAIDb level has its own scheduler.
  - *request caches*: these are optional components that can cache query parsing, the result set and result metadata of queries.
  - *load balancer*: it balances the load on the underlying backends according to the chosen RAIDb level configuration.
  - *recovery log*: it handles checkpoints and allows backends to dynamically recover from a failure or to be dynamically added to a running cluster.

- *database backend*: it represents the real database backend running the RDBMS engine. A *connection manager* mainly provides connection pooling on top of the database JDBC native driver.

Each virtual database and its components are configured using an XML configuration file that is sent from the administration console to the C-JDBC controller.

> **Note:** A research report details RAIDb and C-JDBC implementation (http://c-jdbc.objectweb.org/current/doc/RR-C-JDBC.pdf). Other documents and presentations about C-JDBC can be found in the documentation section of the web site (http://c-jdbc.objectweb.org/doc).

6.2. Starting the Controller

The `bin` directory of the C-JDBC distribution contains the scripts to start the controller. Unix users must start the controller with **controller.sh** whereas Windows users will use **controller.bat**.

Since C-JDBC Controller version 1.0b11, the controller start is tuned via a configuration file, called `controller.xml`, included under the `config/controller` directory of your C-JDBC installation. A simple configuration file looks like this:

A standard C-JDBC Controller configuration file looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE C-JDBC-CONTROLLER PUBLIC "-//ObjectWeb//DTD C-JDBC-CONTROLLER 2.0.2//EN"  "http://c-jdbc.objectweb.
<C-JDBC-CONTROLLER>
<Controller port="25322">
    <Report hideSensitiveData="true" generateOnFatal="true"/>
    <JmxSettings>
      <RmiJmxAdaptor/>
    </JmxSettings>
  </Controller>
</C-JDBC-CONTROLLER>
```

You can specify at startup a different file than `config/controller/controller.xml`. This is useful if you have to startup many identical controllers from the network. You can then use the command **controller.sh -f filename** on Unix machines or **controller.bat -f filename** on windows.

For more information you can refer to the controller-configuration.xml example in the example directory of c-jdbc.

Next section describes how to write a controller configuration file.

6.3. Writing the controller configuration file

The controller is entirely configurable via an xml file, by default it is `controller.xml` located in the `config/controller` of the C-JDBC installation. This section details how to write such a file.

6.3.1. Controller Parameters

The root element of the controller configuration is defined as follows

```
<!ELEMENT Controller (Internationalization?, Report?, JmxSettings?,
                                  VirtualDatabase*, SecuritySettings?)>
<!ATTLIST Controller
  port             CDATA "25322"
  ipAddress        CDATA "127.0.0.1"
  backlogSize      CDATA "10"
>
```

All sub-elements of Controller are defined in the next sections. Here is a brief overview of each of them:

- `Internationalization`: defines the language setting for C-JDBC console and error messages.
- `Report`: if this option is enabled, C-JDBC can automatically generate a report on fatal errors or shutdown. If you experience any problem with C-JDBC, you can directly send the report on the mailing list to get a quick diagnostic of what happened.
- `JmxSettings`: JMX is the technology used for management and monitoring in C-JDBC. These functionalities can be accessed through HTTP with an internet browser or through the RMI connector used by the C-JDBC console.
- `VirtualDatabase`: Defines a virtual database to load automatically at controller startup given a reference to its configuration file.
- `SecuritySettings`: Allows to filter accesses to a controller based on access lists.

The attributes of a Controller element are defined as follows:

- `port`: the port number on which clients (C-JDBC drivers) will connect. The default port number is *25322*.

  Note: A port number below 1024 will require running the controller with privileged rights (root user under Unix).

- `ipAddress`: This can be defined to bind a specific IP address in case of a host with multiple IP addresses. This can be ignored if there is only one IP address available and will be replaced by `127.0.0.1`.
- `backlogSize`: the server socket backlog size (number of connections that can wait in the accept queue before the system returns "connection refused" to the client). Default is 10. Tune this value according to your operating system, but the default value should be fine for most settings.

If your machine has multiple network adapters, you can for the C-JDBC Controller to bind a specific IP address like this:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE C-JDBC-CONTROLLER PUBLIC "-//ObjectWeb//DTD C-JDBC-CONTROLLER 2.0.2//EN"  "http://c-jdbc.objectweb.
<C-JDBC-CONTROLLER>
<Controller port="25322" ipAddress="192.168.0.1">
<JmxSettings enabled="false"/>
</Controller>
</C-JDBC-CONTROLLER>
```

6.3.2. Internationalization

You can use this element to override the default locale retrieved by java. English is the only language looked at at the moment.

```
    <!ELEMENT Internationalization EMPTY>
    <!ATTLIST Internationalization language (en|fr|it|jp) "en">
```

6.3.3. Report

A report can be define in case you want to get a trace of what happened during the execution of the controller. If this element is included in the `controller.xml` report is enabled and will output a report, under certain conditions, in a file named `c-jdbc.report`.

```
<!ELEMENT Report EMPTY>
<!ATTLIST Report
     hideSensitiveData  (true|false) "true"
     generateOnShutdown (true|false) "true"
     generateOnFatal    (true|false) "true"
     enableFileLogging  (true|false) "true"
     reportLocation     CDATA        #IMPLIED
>
```

- `hideSensitiveData`: will replace passwords with '*****'.
- `generateOnShutdown`: tells the controller to generate a report when it has received a shutdown command.
- `generateOnFatal`: tells the controller to generate a report when it cannot recover from an error.
- `enableFileLogging`: logs all the console output into a file and include this file into the report.
- `reportLocation`: specify the path where to create the report, default is `CJDBC_HOME/log` directory.

6.3.4. JMX

JMX is used to remotely administrate the controller. You can use the bundled C-JDBC console or use your own code to access JMX MBeans via the protocol adaptor. C-JDBC proposes both the RMI and HTTP adaptors of the MX4J (http://mx4j.sourceforge.net/) JMX server. You can override the default port numbers for each adaptor if they conflict with another application that is already using them (i.e. another C-JDBC controller on the same machine).

```
<!ELEMENT JmxSettings (HttpJmxAdaptor?, RmiJmxAdaptor?)>
<!ELEMENT HttpJmxAdaptor EMPTY>
<!ATTLIST HttpJmxAdaptor
  port CDATA "8090"
>

 <!ELEMENT RmiJmxAdaptor (SSL?)>
 <!ATTLIST RmiJmxAdaptor
   port         CDATA        "1090"
   username     CDATA        #IMPLIED
   password     CDATA        #IMPLIED
 >


 <!ELEMENT SSL EMPTY>
 <!ATTLIST SSL
   keyStore CDATA        #REQUIRED
   keyStorePassword CDATA        #REQUIRED
   keyStoreKeyPassword CDATA        #IMPLIED
   isClientAuthNeeded (true|false) "false"
   trustStore CDATA        #IMPLIED
   trustStorePassword CDATA        #IMPLIED
 >
```

Configure ssl for encryption and/or authentication.

- `keyStore`: The file where the keys are stored
- `keyStorePassword`: the password to the keyStore
- `keyStoreKeyPassword`: the password to the key, if none is specified the same password as for the store is used
- `isClientAuthNeeded`: if set to false ssl is used for encryption, the server is only accepting trusted clients (the client certificate has to be in the trusted store)
- `trustStore`: the file where the trusted certificates are stored, if none is specified the same store as for the key is used

- `trustStorePassword`: the password to the trustStore, if none is specified the same password as for the keyStore is used

You have to enable the RMI adaptor if you want to use the C-JDBC console to administrate the controller remotely. To enable the RMI JMX adaptor, use this setting:

```
     <JmxSettings>
       <RmiJmxAdaptor/>
 </JmxSettings>
```

6.3.5. Virtual Database

This element specifies virtual databases to load at controller startup.

```
<!ELEMENT VirtualDatabase EMPTY>
<!ATTLIST VirtualDatabase
    configFile          CDATA #REQUIRED
    virtualDatabaseName CDATA #REQUIRED
    autoEnableBackends  (true | false | force) "true"
    checkpointName      CDATA ""
>
```

- `configFile`: The path to the virtual database configuration file. See Section 11 to learn how to write a virtual database configuration file.
- `virtualDatabaseName`: The name of the virtual database since the configuration file can contain multiple virtual database definitions.
- `autoEnableBackends`: set to true by default to reenable backends from their last known state as stored during last shutdown. If backends where not properly shutdown, nothing will happen. You can specify false to let the backends in disabled state at startup. The force option should only be used if you know exactly what you are doing and override backend status by providing a new checkpoint. *Warning!* Use this setting carefully as it might break your database consistency if you do not provide a valid checkpoint.Force is considered the same as true if no recovery log has been defined.
- `checkpointName`: the checkpoint name to use with the recovery log to enable backend from a known coherent state. If the checkpoint is omitted, the last known checkpoint is used.

Example:

```
<VirtualDatabase configFile="/databases/MySQLDb.xml" virtualDatabaseName="rubis" autoEnableBackends="true"/>
```

This will enable a virtual database named `rubis` taken from a configuration file named `/databases/MySQLDb.xml` and will enable all backends of the database from the last known checkpoint.

6.3.6. Security

Security settings define the policy to adopt for some functionalities that may compromise the security of the controller. These settings depends on your environment and can be relaxed if you are running in a secure network. The less security settings you have, the faster the controller will run. A SecuritySettings element is defined as follows:

```
<!ELEMENT SecuritySettings (Jar?, Shutdown?, Accept?, Block?)>
<!ATTLIST SecuritySettings
  defaultConnect (true|false) "true"
>
```

`defaultConnect`: is used to allow (true) or refuse (false) connections to the controller. This default setting can be then be tuned with access lists defined in `Accept` and `Block` elements (see below).

Additional database drivers can be uploaded dynamically to the controller. As the controller has no way to check if this is a real JDBC driver or some malicious code hidden a JDBC driver interface, you have to be very careful if you enable this option and anybody can connect from anywhere to your controller.

```
<!ELEMENT Jar EMPTY>
<!ATTLIST Jar
allowAdditionalDriver (true|false) "true"
>
```

The Shutdown element defines how the controller can be terminated - in order to shutdown the controller properly, we have to use the console. Specify if the controller should consider shutdown command received by one or the other, and if this command can only be received from localhost or not. A default configuration would be:

```
<Shutdown>
  <Client allow="true" onlyLocalhost="true"/>
  <Console allow="true" onlyLocalhost="true"/>
</Shutdown>
```

This prevents unwanted and unauthorized shutdown calls from remote hosts. Only somebody logged locally on the machine can request a shutdown of the controller. Here is the full description for details:

```
<!ELEMENT Shutdown (Client?,Console?)>

<!ELEMENT Client EMPTY>
<!ATTLIST Client
    allow         (true|false) "true"
    onlyLocalhost (true|false) "true"
>
<!ELEMENT Console EMPTY>
<!ATTLIST Console
    allow (true|false) "true"
    onlyLocalhost (true|false) "true"
>
```

You can control who can connect to the controller by setting access lists based on IP addresses to accept or block. `defaultConnect` is set in `SecuritySettings` defined above. Default is to accept all connections if no security manager is enabled.

```
<!ELEMENT Accept (Hostname|IpAddress|IpRange)*>
<!ELEMENT Block (Hostname|IpAddress|IpRange)*>

<!ELEMENT Hostname EMPTY>
<!ATTLIST Hostname
     value CDATA #REQUIRED
```

```
>
```

IpAddress value is an IPv4 address (ex:192.168.1.12):

```
<!ELEMENT IpAddress EMPTY>
<!ATTLIST IpAddress
     value CDATA #REQUIRED
>
```

IpRange value is based on IPv4 addresses and has the following form: 192.168.1.*.

```
<!ELEMENT IpRange EMPTY>
<!ATTLIST IpRange
     value CDATA #REQUIRED
>
```

Here is a full security configuration example:

```
<SecuritySettings defaultConnect="false">
  <Jar allowAdditionalDriver="true"/>
  <Shutdown>
    <Client allow="true" onlyLocalhost="true"/>
    <Console allow="false"/>
  </Shutdown>
  <Accept>
    <IpRange value="192.168.*.*"/>
  </Accept>
</SecuritySettings>
```

This setting accepts driver connections only from machines having an IP address starting with 192.168, allows loading of additional drivers via the console, refuses shutdown from the console, but allows it from the local machine.

6.4. Configuring the Log

C-JDBC uses the Log4j (http://jakarta.apache.org/log4j/) logging framework. The `log4j.properties` configuration file is located in the `/c-jdbc/config` directory of your installation. Here is a brief description of the loggers available in the configuration file:

- `log4j.logger.org.objectweb.cjdbc.core.controller`: Controller related activities mainly for bootstrap and virtual database adding/removal operations.
- `log4j.logger.org.objectweb.cjdbc.controller.xml.Handler`: XML configuration file parsing and handling.
- `log4j.logger.org.objectweb.cjdbc.controller.VirtualDatabase`: Virtual database related operations. A specific `log4j.logger.org.objectweb.cjdbc.controller.VirtualDatabase.virtualDatabaseName` logger is automatically created for each virtual database. This allows to tune different logging levels for each virtual database.

- `log4j.logger.org.objectweb.cjdbc.controller.VirtualDatabase.request`: Log the incoming requests and transactions in files that can be replayed by the Request Player tool provided with C-JDBC.
- `log4j.logger.org.objectweb.cjdbc.controller.distributedvirtualdatabase.request` : Log distributed request execution when using horizontal scalability (a.k.a. controller replication).
- `log4j.logger.org.objectweb.cjdbc.controller.backup` : Log backup manager and backuper related activities from dump/restore operations.
- `log4j.logger.org.objectweb.cjdbc.controller.VirtualDatabaseServerThread`: The server thread accepts client connections and manages the worker threads.
- `log4j.logger.org.objectweb.cjdbc.controller.VirtualDatabaseWorkerThread`: Each worker thread handle a session with a client C-JDBC driver.
- `log4j.logger.org.objectweb.cjdbc.controller.RequestManager`: Log the request flows between the different Request Manager components (scheduler, cache, load balancer, recovery log).
- `log4j.logger.org.objectweb.cjdbc.controller.scheduler`: Log the request ordering and synchronization performed by the scheduler.
- `log4j.logger.org.objectweb.cjdbc.controller.cache`: SQL Query cache related activities.
- `log4j.logger.org.objectweb.cjdbc.controller.loadbalancer`: Log how requests are balanced on the backends.
- `log4j.logger.org.objectweb.cjdbc.controller.connection`: Connection pooling related information.
- `log4j.logger.org.objectweb.cjdbc.controller.recoverylog`: C-JDBC Recovery Log information.
- `log4j.logger.org.objectweb.cjdbc.controller.console.jmx`: JMX management system logging.
- `log4j.logger.org.objectweb.tribe.channels`: Tribe low level group communication channel.
- `log4j.logger.org.objectweb.tribe.gms`: Tribe Group Membership Service (GMS).
- `log4j.logger.org.objectweb.tribe.discovery`: Tribe Discovery Service (used by GMS).
- `og4j.logger.org.objectweb.tribe.blocks.multicastadapter`: Tribe Multicast Dispatcher building block for application level message handling.

6.5. Recovery Log

When you want to add a database to your cluster, you do not want to stop the system, replicate the current database state to the new database (that may take a long while) and then restart the system. The Recovery Log helps you in the process of dynamically adding a new backend (or recovering a previously failed backend) without stopping the system.

The Recovery Log records the write operations and transactions that are performed by the C-JDBC controller between checkpoints. A checkpoint is just a logical index in the log that reflect the recovery log state at a given time. As of C-JDBC 2.0, checkpoints are automatically managed by the controller and are generated when needed on behalf of the administrator when a backend is disabled or enter a backup phase. When re-enabling the backend, the Recovery Log replays all write queries and transactions that the backend missed during the time it was offline and it comes back to the enabled state once it is synchronized with the other nodes.

> **Note:** Since version 2.0, the backup infrastructure has completely changed and is based on Backupers. We provide a generic Backuper based on Enhydra Octopus (http://octopus.enhydra.org/) to copy, backup and restore content of backends through JDBC. Even if Octopus is supposed to handle most common

databases, it might fail for some specific databases or data types. In that case, we strongly recommend to use or implement a database specific Backuper.

6.5.1. A practical example

Your Web site is running with a single database and you want to use C-JDBC with three nodes using full replication (RAIDb-1). You have two new backends ready to be installed. You can start the C-JDBC console and connect to the controller. Start the administration module by connecting to the virtual database. Type: **backup <backend name> <dump name> <backuper name> <path to backup directory>**. If you want to use Octopus you will use a command line like **backup node1 dump1 Octopus /var/backups**. During the backup, the update requests are logged in the recovery log, so no update is lost. If the backend was in the enabled state when backup was initiated, it will automatically replay the recovery log to resynchronize itself and return to the enabled state.

To restore the dump on another backend, just type **restore <newbackend> <dumpname>** and the appropriate backuper (Octopus in our previous example) will be used to restore the dump. After restoring the dump, you can enable the backend at any time so that the recovery log replays all the missing requests since the dump was taken.

Here is the set of commands to use in the C-JDBC console if node1 is your existing backend and you want to dynamically add node2 and node3:

```
backup node1 initial_dump Octopus /var/backups
restore node2 initial_dump
restore node3 initial_dump
enable node2
enable node3
```

Note: Note that these steps can be automated by scriptin the console.

If a node crashes, use the administration console to restore the dump on the node using the restore command. Once the dump is restored, re-enable the backend from the stored checkpoint and the Recovery Log will automatically replay all the write queries to rebuild a consistent database state on the node.

To prevent the recovery log from being too large, you can periodically perform backup operations. This will also lower the recovery time since the part of the log to replay will be smaller. You can delete older dumps and logs if you do not need them anymore.

6.5.2. Understanding checkpoints

A checkpoint is a reference used by the recovery log to replay missing requests. If a backend is disabled from the console for maintenance, the controller will automatically create a checkpoint (prior to v2.0, the checkpoint name had to be provided manually through the console). Once the backend is enabled again, the controller retrieves its last known checkpoint from the recovery log and replays all the requests that the disabled backend missed since it was disabled. A checkpoint is nothing more than a reference in time.

6.5.3. A fault tolerant Recovery Log

As the C-JDBC recovery log can be stored in a database providing a JDBC driver, it is possible to make the recovery log fault tolerant by redirecting it to a C-JDBC controller (even self) that will distribute and replicate the log content on several backends.

The JDBC Recovery Log configuration is detailed in Section 11.6.5.

6.6. Controller replication

To prevent the C-JDBC controller from being a single point of failure, C-JDBC provides controller replication also called horizontal scalability. A virtual database can be replicated in several controllers that can be added dynamically at runtime. Controllers use the JGroups group communication middleware to synchronize updates in a distributed way. The JGroups stack configuration is found in `config/jgroups.xml` and should not be altered unless you specifically know what you are doing. Keep in mind that total order reliable multicast is needed to ensure proper synchonization of the controllers. More information about JGroups can be found on the JGroups web site (http://www.jgroups.org). Note that JGroups requires proper network settings, here are a few guidelines:

- a default route must be defined (check with /sbin/route under Linux) for the network adapter which is bound by JGroups (usually eth0). If such route does not exist, either the group communication initialization will block or controllers will not be able to see each other even on the local host. If you don't have any default entry in your routing table you can use a command like '/sbin/route add default eth0' to define this default route.
- issues have been reported with DHCP that can either block (under Windows) or just fail to properly set a default route and leads to the issue reported above. We strongly discourage the use of DHCP, you should use fixed IP addresses instead.
- name resolution should be properly set so that the IP address/machine name matching works both ways. Often improper /etc/hosts or DNS configuration leads to group communication initialization problems. In particular, under Linux, the IP address associated to the name returned by the 'hostname' command must not resolve to 127.0.0.1 else controllers will not see each other.

In order for a virtual database to be replicated, you must define a `Distribution` element in the virtual database configuration file (see Section 11.2.1). There are several constraints for different controllers to replicate a virtual database:

- give the list of all controllers that you plan to use for replication of your virtual database in the C-JDBC driver URL. Even if all controllers are not online at all times, the driver will automatically detect the alive controllers: jdbc:cjdbc://node1,node2,node3,node4/myDB
- the virtual database must have the same name and use the same groupName (in the Distribution element).
- each controller must have its own set of backends and no backends should be shared between controllers (C-JDBC checks the database URLs, having different backend names is not sufficient).
- each controller must have its own recovery log, recovery logs cannot be shared. It is possible for a controller not to have a recovery log but this controller will have no recovery capabilities.
- the authentication managers must support the same logins.
- schedulers and load balancers must implement the same RAIDb configuration.
- database schemas (if defined) must be compatible according to the RAIDb level you are using.

> **Note:** As backends cannot be shared between controllers, it is not possible to use a SingleDB load balancer with controller replication. If each controller only has a single database backend attached to it, then you must use a RAIDb-1 configuration since in fact you have 2 replicated backends in the cluster.

Several configuration file examples are available in the `doc/examples/HorizontalScalability` directory of your C-JDBC distribution.

Note: You can find more information in the document titled "C-JDBC Horizontal Scalability - A controller replication user guide" available from the C-JDBC web site.

6.7. Current Limitations

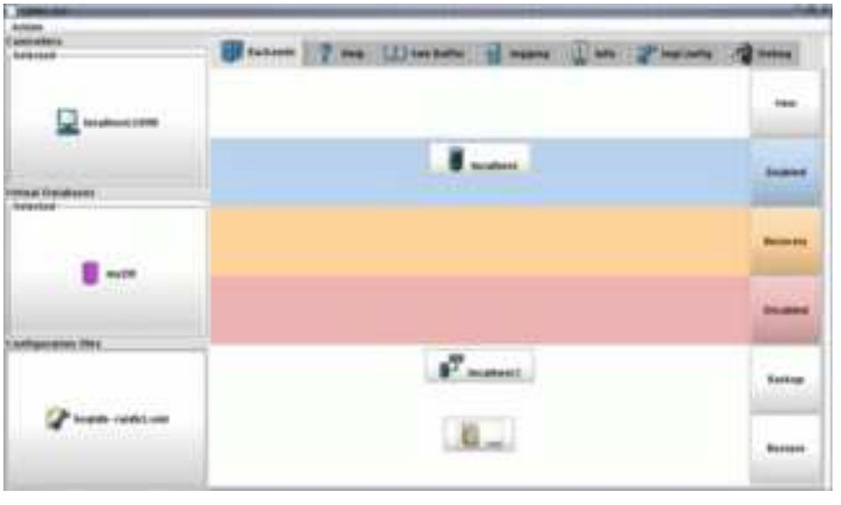
The C-JDBC controller in its 2.0.2 release has the following limitations:

- GRANT/REVOKE commands will be sent to the database engines but this will not add or remove users from the virtual database authentication manager.
- network partition/reconciliation is not supported,
- distributed joins are not supported which means that you must ensure that every query can be executed by at least a single backend,
- `RAIDb-1ec` and `RAIDb-2ec` levels are not supported,

7. Administration console

The C-JDBC administration console is now based on JMX technologies. The text mode console is a JMX client based on the standard RMI connector for JMX but you can also use a generic a JMX administration console through HTTP from any web browser to see all the MBeans registered in the cjdbc domain. The graphical console is not fully supported and only the text console is actively maintained.

You can start the graphic interface using the console.sh/.bat script. If your environment does not support graphic interface, it will automatically revert to the text console.



Note: *This section is under huge rewriting and will be put up to date soon. Please, do come back and check this soon.*

7.1. Jmx Notifications List

Here is a list of the JMX remote notifications generated by C-JDBC.

- `cjdbc.controller.virtualdatabases.removed` a virtual database has been removed.
- `cjdbc.controller.virtualdatabase.added` a virtual database has been added to the controller
- `cjdbc.virtualdatabase.dump.list` the list of dump files has been updated
- `cjdbc.virtualdatabase.backend.added` a backend has been added to the virtual database
- `cjdbc.distributed.controller.added` a controller has joined the group
- `cjdbc.virtualdatabase.backend.disabled` a backend has been disabled
- `cjdbc.virtualdatabase.backend.enabled` a backend has been enabled
- `cjdbc.virtualdatabase.backend.recovering` a backend is recovering a dump file
- `cjdbc.virtualdatabase.backend.recovery.failed` Recovery of a dump file failed
- `cjdbc.virtualdatabase.backend.replaying.failed` Recovery log replay failed
- `cjdbc.virtualdatabase.backend.backingup` a backend is backing up
- `cjdbc.virtualdatabase.backend.enable.write` a backend is now write enabled
- `cjdbc.virtualdatabase.backend.removed` a backend has been removed from the virtual database
- `cjdbc.virtualdatabase.backend.disabling` a backend is now in state disabling (finishing pending transactions and pending requests)
- `cjdbc.virtualdatabase.backend.unknown` The backend state has been completely lost. Recovery needed
- `cjdbc.virtualdatabase.backend.replaying` a backend is replaying requests from the recovery log

7.2. Starting the Administration Console

The `bin` directory of the C-JDBC distribution contains the scripts to start the console. Unix users must start the console with **console.sh -t** whereas Windows users have to start **console.bat -t**.

The console script accepts several options:

- `-d` or `--debug`: show stack trace when error occurs.
- `-f` or `--file`: Use a given file as the source of commands instead of reading commands interactively.
- `-h` or `--help`: displays usage information.
- `-i` or `--ip`: IP address of the host name where the JMX Server hosting the controller is running (the default is '0.0.0.0').
- `-p` or `--port`: JMX/RMI port number of (the default is 1090).
- `-s` or `--secret`: Password for JMX connection.
- `-u` or `--username`: username for JMX connection.
- `-v` or `--version`: displays version information.
- `-t` or `--text`: force the console to start in text mode. By default, it will try to start in graphic mode

For example, **console.sh -t -i 192.168.0.1 -p 1234** will connect the console to the controller using the RMI JMX adaptor listening on port 1234 on `192.168.0.1`.

The console has an online help that is accessible by typing **help** at any time.

7.3. Console Quickstart

Here is a quick description of the steps needed to make a controller ready to serve requests:

1. Start the controller using **controller.sh** or **controller.bat** (see Section 6.2).
2. Start the console using **console.sh -t** or **console.bat -t** (see Section 7.2).
3. Load a configuration file using **load <complete-path>/config.xml**. The controller configuration files are described in Section 11.
4. Connect to the virtual database with the administrator login using the **admin** command (see example below).
5. Enable all backends using the **enableAll** command.
6. Come back to the main menu using the **quit** command.
7. Check the configuration using the **getInfo** command.

Here is an example of a controller configuration and startup:

```
[emmanuel@gre-home bin]$ console.sh -t
Launching the C-JDBC controller console
Initializing Controller module...
Initializing VirtualDatabase Administration module...
Initializing Monitoring module...
Initializing SQL Console module...
C-JDBC driver (v. 2.0) successfully loaded.

gre-home:1090 >help
Commands available for the Controller module are:
admin <virtualdatabase name>
   Administrate a virtual database
connect controller <controller hostname> <jmx port>
   Connect to a C-JDBC controller
drop virtualdatabase <virtualdatabase name>
   Drop a virtual database from the controller
help
   Print this help message
history [<commandIndex>]
   Display history of commands for this module
load virtualdatabase config <virtualdatabase xml file>
   Send a virtual database XML configuration file to the controller and load it
monitor <virtualdatabase name>
   Monitor a virtual database
quit
   Quit this console
reload logging configuration
   Refresh the trace system by reloading the logging configuration file
save configuration
   Save the current configuration of the virtual databases as an XML file
show controller config
   Show Controller configuration
show logging config
   Show logging configuration and the most recent traces
show virtualdatabases
```

```
   Show the names of the virtual databases for this controller
shutdown [mode]
   Shutdown the controller and all its virtual databases. Mode parameter must be:
        1 -- wait for all client connections to be closed, does not work with a connection pool
        2 -- mode safe, default value, waits for all current transactions to complete
        3 -- mode force, immediate shutdown without consistency: recovery will be needed on restart
sql client <c-jdbc url>
   Open a SQL client console for the virtual database specified by the C-JDBC URL
upload driver <driver file>
   Upload a driver to the controller

gre-home:1090 > <userinput>show virtualdatabases</userinput>
myDB
gre-home:1090 > <userinput>admin myDB</userinput>
Virtual database Administrator Login > <userinput>admin</userinput>
Virtual database Administrator Password > <userinput>*****</userinput>
Ready to administrate virtual database myDB
myDB(admin) > help
Commands available for the VirtualDatabase Administration module are:
backup <backend name> <dump name> <backuper name> <path> [<tables>]
   Backup a backend into a dump file and associate a checkpoint with this dump
delete dump <dump name>
   Delete a dump
disable <backend name | *>
   Disable the specified backend and automatically set a checkpoint
        * means that all backends of this virtual database must be disabled
enable <backend name | *>
   Enable the specified backend
        * means that all backends of this virtual database must be enabled
expert <on|off>
   Switch to expert mode (commands for advanced users are available)
help
   Print this help message
history [<commandIndex>]
   Display history of commands for this module
quit
   Quit this console
restore <backend name> <dump name> [<tables>]
   Starts the recovery process of the given backend for a given dump name
show backend <backend name | *>
   Show information on backend of this virtual database
        * means to show information for all the backends of this virtual database
show backends
   Show the names of the backends of this virtual database on the current controller
show backupers
   Show the backupers available for backup
show controllers
   Show the names of the controllers hosting this virtual database
show dumps
   Show all dumps available for database recovery
show virtualdatabase config
   Show the XML configuration of the virtual database
transfer dump <dump name> <controller name> [nocopy]
   Make a dump available for restore on another controller.
Optional 'nocopy' (default: false) flag specifies not to copy the dump.

myDB(admin) > <userinput>show backend *</userinput>
```

```
+--------------------------+----------------------------------+
| Backend Name             | localhost                        |
| Driver                   | org.hsqldb.jdbcDriver            |
| URL                      | jdbc:hsqldb:hsql://localhost:9001 |
| Active transactions      | 0                                |
| Pending Requests         | 0                                |
| Read Enabled             | true                             |
| Write Enabled            | true                             |
| Is Initialized           | true                             |
| Static Schema            | false                            |
| Connection Managers      | 1                                |
| Total Active Connections | 5                                |
| Total Requests           | 0                                |
| Total Transactions       | 0                                |
| Last known checkpoint    | <unknown>                        |
+--------------------------+----------------------------------+
| Backend Name             | localhost2                       |
| Driver                   | org.hsqldb.jdbcDriver            |
| URL                      | jdbc:hsqldb:hsql://localhost:9002 |
| Active transactions      | 0                                |
| Pending Requests         | 0                                |
| Read Enabled             | true                             |
| Write Enabled            | true                             |
| Is Initialized           | true                             |
| Static Schema            | false                            |
| Connection Managers      | 1                                |
| Total Active Connections | 5                                |
| Total Requests           | 0                                |
| Total Transactions       | 0                                |
| Last known checkpoint    | <unknown>                        |
+--------------------------+----------------------------------+
```

7.4. Console Main Menu

The graphical version of the console provides a shell-like history (more precisely a tcsh-like behavior). You can recall a previous command by using the arrow keys (up and down) to browse the history. If you prefix a command by `!`, the console will browse the history and complete the command with the latest command in the history starting with the command prefix (completion occurs when you press the tab key). In the graphical version, you can also access all the commands of the different module using the right button of the mouse.

> **Note:** All the commands issued can also be recalled using the history menu in the contextual menu that appears on a right-button click.

Commands available from the console main menu are:

- **admin <virtualdatabase name>**: Administrate a virtual database
- **connect controller <controller hostname> <jmx port>**: connect to a C-JDBC controller
- **drop virtualdatabase <virtualdatabase name>**: Drop a virtual database from the controller
- **help**: Print this help message

- **history [<commandIndex>]**: Display history of commands for this module
- **load virtualdatabase config <virtualdatabase xml file>**: Send a virtual database XML configuration file to the controller and load it
- **monitor <virtualdatabase name>**: Monitor a virtual database
- **quit**: Quit this console
- **reload logging configuration**: Refresh the trace system by reloading the logging configuration file
- **save configuration**: Save the current configuration of the virtual databases as an XML file
- **show controller config**: Show Controller configuration
- **show logging config**: Show logging configuration and the most recent traces
- **show virtualdatabases**: Show the names of the virtual databases for this controller
- **shutdown [`mode`]**: shutdown the controller and all its virtual databases.

  Three shutdown modes are provided. If not specified, the default mode is the shutdown mode immediate.

  - Shutdown mode wait (mode `1`): wait for all client connections to be closed, does not work if the client uses a connection pool with persistent connections.
  - Shutdown mode safe (mode `2`): default value, waits for all current transactions to complete before shutting down. transaction and shutdown.
  - Shutdown mode force (mode `3`): does not wait for transactions completion and kill all connections. Backends are disabled without consistency and a full recovery will be needed on restart.

  E.g: **shudown 2**.

- **sql client <c-jdbc url>**: Open a SQL client console for the virtual database specified by the C-JDBC URL
- **upload driver <driver file>**: Upload a driver to the controller

7.5. Administrator Menu

Once the configuration file has been loaded on the controller, all backends are in the disabled state. You must enable them all or one by one to allow them to execute requests. C-JDBC does not check that database contents are synchronized and you must ensure that all backends are in a coherent state prior to starting the controller. To ensure that backends remain synchronized on startup, you must use checkpoints (see Section 6.5.2).

If you properly shutdown the controller using the wait or safe mode, database backend states are properly recorded and their state is automatically restored when they are enabled.

7.5.1. Administrator Standard Commands

Standard commands available from the console administrator menu are:

- **backup <backend name> <dump name> <backuper name> <path> [<tables>]**: Backup a backend into a dump file and associate a checkpoint with this dump. Note that the console will ask for a login and password to connect to the backend to backup. This is specific to the Backuper that you are using but this should usually be a valid login/password on the database engine that you are backuping. The login must be granted access on all tables from the controller node.
- **delete dump <dump name>**: Delete a dump

- **disable <backend name | *> <checkpoint>**: Disable the specified backend and store the given checkpoint (* means that all backends of this virtual database must be disabled)
- **enable <backend name | *>**: Enable the specified backend from its last known checkpoint (* means that all backends of this virtual database must be enabled)
- **expert <on|off>**: Switch to expert mode (commands for advanced users are available)
- **help**: Print this help message
- **history [<commandIndex>]**: Display history of commands for this module
- **quit**: Quit this console
- **restore <backend name> <dump name> [<tables>]**: Starts the recovery process of the given backend using the given dump name. Note that the console will ask for a login and password to connect to the backend to restore This is specific to the Backuper that you are using but this should usually be a valid login/password (real login in the C-JDBC terminology) on the database engine that you are restoring. Note that this login must be granted the right to create new databases and tables.
- **show backend <backend name | *>**: Show information on backend of this virtual database (* means to show information for all the backends of this virtual database)
- **show backends**: Show the names of the backends of this virtual database on the current controller
- **show backupers**: Show the backupers available for backup
- **show controllers**: Show the names of the controllers hosting this virtual database
- **show dumps**: Show all dumps available for database recovery
- **show virtualdatabase config**: Show the XML configuration of the virtual database
- **transfer dump <dump name> <controller name>**: Transfer a dump from the current controller to another controller. An example is **transfer dump dump1 controller2.emic.com:1090**

7.5.2. Administrator Expert Commands

Expert commands are not available by default. use the command expert on to make them available.

- **clone backend config <backend from> <backend to> <url> [driverPath=<value>] [driver=<value>] [connectionTestStatement=<value>]**: Clone the configuration of a backend in the current virtual database (this virtually allows to add a new backend)
- **disable read <backend name>**: Disable read requests on a backend
- **enable read <backend name>**: Enable read requests on a backend
- **force checkpoint <backend name> <checkpoint name>**: Force the last know checkpoint of a disabled backend
- **force disable <backend name | *>**: Force the disabling of a backend without storing any checkpoints. The backend will not be in a consistent state after this operation! (* means that all backends of this virtual database must be disabled by force)
- **force enable <backend name | *>**: Force the enabling of a backend without checking for checkpoints. This command can break the cluster consistency, only use it if you know what you are doing! (* means that all backends of this virtual database must be enabled by force)
- **force path <dump name> <new path>**: Update the path of the dump
- **get backend schema <backend name> <file name>**: Display backend schema or save it to a file

- **purge log <checkpoint name>**: Purge the recovery log upto specified checkpoint. All the entries of the recovery log prior to that checkpoint will be deleted.
- **restore log <dump name> <controller name>**: Copy the local recovery log from the specified checkpoint onto the specified remote controller. All previous recovery log content on the remote controller will be erased.
- **show checkpoints**: Show all checkpoints available in the recovery log.
- **transfer backend <backend name> <controller jmx address>**: Transfer a backend from a controller to an other controller

7.6. Automated Backup With Jmx

Marc Wick has given an example of a cron file to do a daily backup using the jmx connector in C-JDBC. The complete sources can be found in the example file:DBBackup.java in the jmx directory of the examples.

```
JMXServiceURL address = new JMXServiceURL("rmi", host, 0, "/jndi/jrmp");

Map environment = new HashMap();
environment.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.rmi.registry.RegistryContextFactory");
environment.put(Context.PROVIDER_URL, "rmi://" + host + ":" + port);
environment.put(JMXConnector.CREDENTIALS, PasswordAuthenticator
    .createCredentials("jmxuser", "jmxpassword"));

JMXConnector connector = JMXConnectorFactory.connect(address, environment);
ObjectName db = JmxConstants.getVirtualDbObjectName("databaseName");

...

MBeanServerConnection delegateConnection = connector
    .getMBeanServerConnection(subj);

// we create a proxy to the virtual database
VirtualDatabaseMBean proxy = (VirtualDatabaseMBean) MBeanServerInvocationHandler
    .newProxyInstance(delegateConnection, db, VirtualDatabaseMBean.class,
        false);

SimpleDateFormat fmt = new SimpleDateFormat("yyyy_MM_dd");
String checkpointName = fmt.format(new Date());

// we disable the backend and set a checkpoint
proxy.disableBackendForCheckpoint("node1", checkpointName);

// we call the database specific backup tool for the backup
runDatabaseBackupTool();

// we enable the backend again
proxy.enableBackend("node1");
```

The runDatabaseBackupTool() method is completely open and can call any external program (like pg_dump, mysql_dump...)

> **Note:** This method does not use octopus and as a consequence, the generated backup cannot be restored on a a different database vendor than the one it was issued from. As a great benefit though, the backup process will gain in speed, and the metadata will be completely conformed to that database vendor.

7.7. Recoverying from a failed controller in distributed mode

In a distributed controller configuration, when a controller goes down, here is the list of action to take to recover the failed controller:

- If the controller does not have any dump available, connect to a controller that has database dumps and use the **transfer dump** command to copy the dump to the recovering controller.
- During its failure, the recovery log of the controller missed queries that were executed by the cluster and it is therefore necessary to re-synchronize its recovery log. This can be achieve using the **recover log** from the same controller you used to transfer the dump.
- Once the previous operations are completed, you can safely restore the dump on the backends attached to the controller. Then, enabling the backends will resynchronize them with the other nodes of the cluster.

8. Virtual Database Console Menu

C-JDBC is now bundled with a graphical SQL console called Squirrel that you can launch fron the `bin` directory of the C-JDBC installation, using either `squirrel.sh` or `squirrel.bat`.You can also directly issue SQL statements from the virtual database console menu. The other commands available from the virtual database console menu are:

- **begin**: Start a transaction
- **commit**: Commit a transaction
- **fetchsize <x>**: Set the ResultSet fetch size to x rows per block
- **help**: Print this help message
- **history [<commandIndex>]**: Display history of commands for this module
- **load <file name>**: Execute all SQL statements contained in file
- **maxrows <x>**: Limits the maximum number of rows to get from the database to x
- **quit**: Quit this console
- **rollback [<savepoint name>]**: Rollback a transaction (to an optional savepoint)
- **savepoint <savepoint name>**: Create a savepoint for the current transaction
- **setisolation <x>**: Set the connection transaction isolation level to x
  - 0 - TRANSACTION_NONE
  - 1 - TRANSACTION_READ_UNCOMMITTED
  - 2 - TRANSACTION_READ_COMMITTED
  - 4 - TRANSACTION_REPEATABLE_READ
  - 8 - TRANSACTION_SERIALIZABLE
- **show tables**: Display all the tables of this virtual database

- **timeout <x>**: Set the query timeout to x seconds (default is 60 seconds)
- **{call proc_name(?,?,...)}**: Call a stored procedure

Here is an example of a session with the virtual database console:

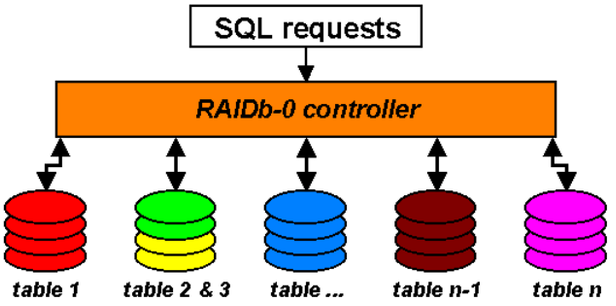
```
localhost:1090 > sql client jdbc:cjdbc://localhost/myDB
 > Login     : user
 > Password  : ******
Connected to jdbc:cjdbc://localhost/myDB
jdbc:cjdbc://localhost/myDB (user) > begin
Transaction started
jdbc:cjdbc://localhost/myDB (user) > select * from regions
... result to be displayed here ...
jdbc:cjdbc://localhost/myDB (user) > commit
jdbc:cjdbc://localhost/myDB (user) > quit
```

9. Monitoring Console Menu

Once you have typed **monitor** in the main console you have now access to the monitor console. You can use **settarget `database`** to monitor all the resources of a specific virtual database. The available commands in the monitor console are:

- **settarget `databasename`**: monitor a specific database. If no target is specified, monitors the controller.
- **help**: print this help message.
- **quit**: return to the main menu.
- **showstats**: display sql statistics in a new frame.
- **showclients**: display client information in a new frame.
- **showcache**: display cache content in a new frame.
- **showcachestats**: display cache stats content in a new frame.
- **showbackends**: display backends data in a new frame.
- **showcontroller**: show the overall controller load.
- **showdatabases**: show the status of the different datases.
- **showscheduler**: show the status of the scheduler.
- **closegraph `index`**: close the graph at the specified index.
- **trace `controller|databases|cache|scheduler|backends`**: Start tracing a graph with live info from the specified target.
- **stoptrace**: stop all graph updaters.

Figure 3. RAIDb-0 example



10. RAIDb Basics

10.1. RAIDb Definition

RAIDb stands for *Redundant Array of Inexpensive Databases*. This acronym has been used in reference to the RAID (*Redundant Array of Inexpensive Disks*) concept that achieves scalability and high availability of disk subsystems at a low cost. RAIDb aims at providing better performance and fault tolerance than a single database by combining multiple inexpensive database instances into an array of databases.

One of the goals of RAIDb is to hide the distribution complexity and to provide the database clients with the view of a single database. As for RAID, a controller sits in front of the underlying resources. The clients send their requests to the RAIDb controller that balances them among the set of RDBMS backends.

10.2. RAIDb-0

RAIDb-0 consists in *partitioning* the database tables among the database backend nodes. A table itself cannot be partitionned but the different tables can be distributed on different backend nodes. RAIDb-0 requires at least two database backends, provides moderate performance scalability but does not offer fault tolerance. Figure 3 shows an example of a RAIDb-0 configuration.

10.3. RAIDb-1

RAIDb-1 offers a *full mirroring* or *full replication* of the database on the backends. It offers the best fault tolerance scheme since the system is still available with only one backend. On the minus side, there is no speedup on writes (`UPDATE`, `INSERT`, `DELETE` requests) since they have to be broadcasted to all nodes. Figure 4 shows an example of a RAIDb-1 configuration.

10.4. RAIDb-2

RAIDb-2 is a tradeoff between RAIDb-0 and RAIDb-1. It provides partial replication to tune the degree of replication of each database table to obtain the best read/write throughput. RAIDb-2 requires that each database table is available on at least two nodes. Figure 5 shows an example of a RAIDb-2 configuration.

Figure 4. RAIDb-1 example

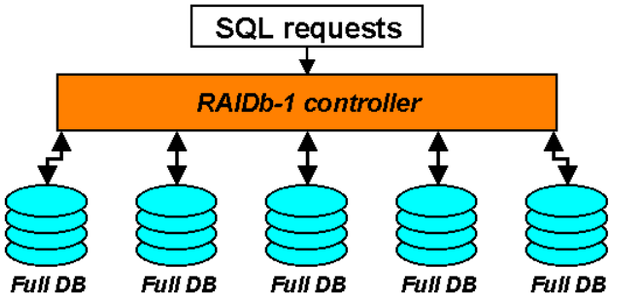
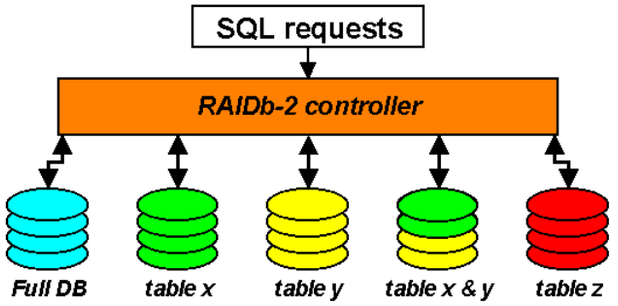


Figure 5. RAIDb-2 example



10.5. Nested RAIDb Levels

It is possible to compose several RAIDb levels to build large scale configurations or meet specific needs. The next example is a RAIDb-1-0 configuration where a top level RAIDb-1 controller dispatches the requests to three full databases implemented with a RAIDb-0 controller. Figure 6 shows an example of a RAIDb-1-0 configuration.

This last example (Figure 7) shows a RAIDb-0-1 composition. The top level is a RAIDb-0 controller and fault tolerance is achieved on each partition using a RAIDb-1 controller.

11. Virtual database configuration

11.1. Writing a Virtual Database Configuration File

The C-JDBC controller configuration file must conform to the C-JDBC DTD that can be found in the `xml` directory of the C-JDBC distribution. The DTD is extensively documented and the most up-to-date information will be found in the `xml/c-jdbc-x.y.dtd` file. Several configuration file examples are available in the `doc/examples` directory.

Here is an example of how a minimal C-JDBC configuration file should look like:

```
<?xml version="1.0" encoding="UTF8"?>
```

Figure 6. RAIDb-1-0 example

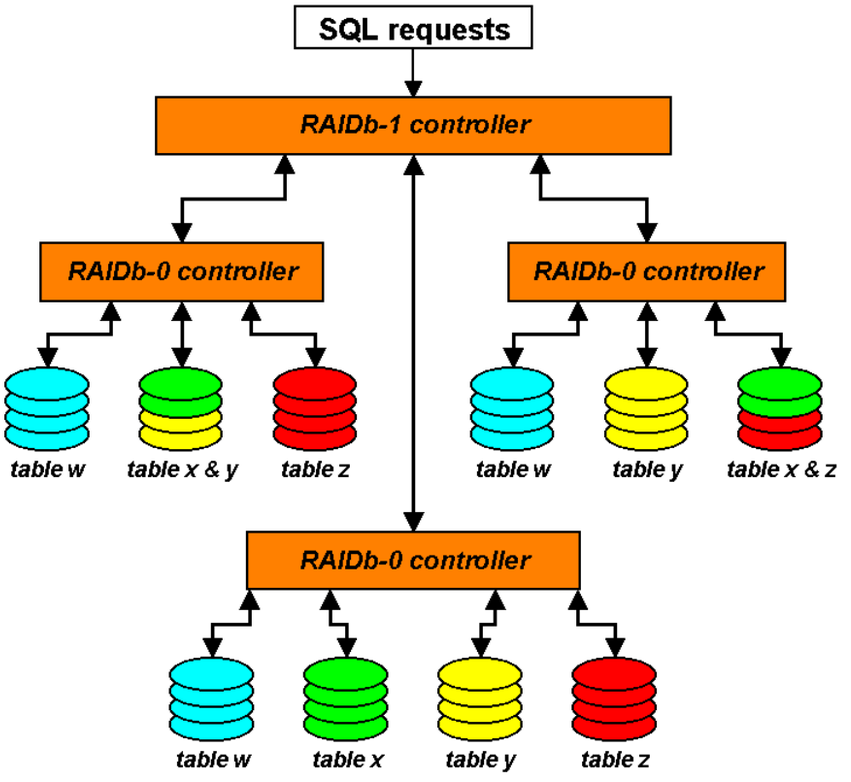
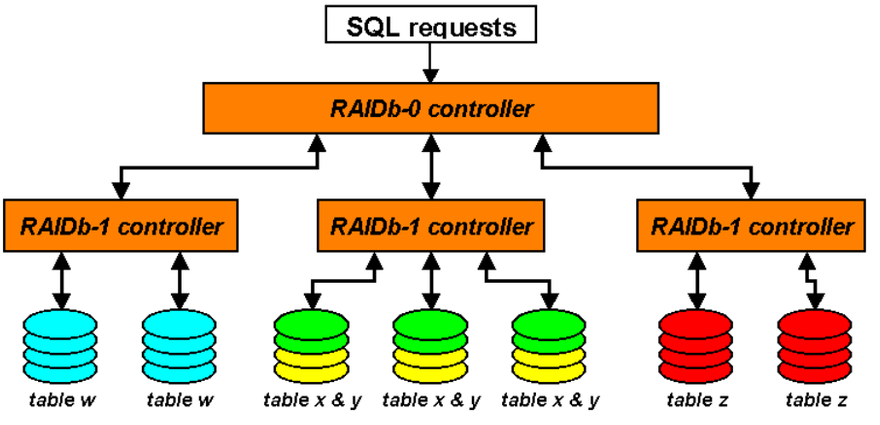


Figure 7. RAIDb-0-1 example



```
<!DOCTYPE C-JDBC PUBLIC "-//ObjectWeb//DTD C-JDBC 1.0//EN"
                        "http://www.objectweb.org/c-jdbc/dtds/c-jdbc-2.0.2.dtd">

<C-JDBC> <VirtualDatabase name="vdbName">
<AuthenticationManager> ...  </AuthenticationManager>

    <DatabaseBackend name="node1" driver="com.myDriver.class"
                     url="jdbc:protocol://host/myDB" connectionTestStatement="select 1">
       ...
    </DatabaseBackend>

    <RequestManager>
      <RequestScheduler>
         ...
      </RequestScheduler>

      <LoadBalancer>
         ...
      </LoadBalancer>
    </RequestManager>
  </VirtualDatabase>
</C-JDBC>
```

The next sections describes the different elements composing an XML configuration file.

11.2. Virtual Database

A virtual database element is defined as follows:

```
<!ELEMENT VirtualDatabase (Distribution?, Monitoring?, Backup?,AuthenticationManager, DatabaseBackend+, Reque
<!ATTLIST VirtualDatabase
       name               CDATA #REQUIRED
       maxNbOfConnections CDATA #IMPLIED
       poolThreads        (true | false) "true"
       minNbOfThreads     CDATA #IMPLIED
       maxNbOfThreads     CDATA #IMPLIED
       maxThreadIdleTime  CDATA #IMPLIED
       sqlDumpLength      CDATA "40"
>
```

A virtual database is the database exposed to the user. It contains:

- a set of real database backends,
- an authentication manager that matches the virtual database and real backends login/password,
- a request manager that defines the behavior of the controller for this virtual database,

Here is a brief description of each virtual database attribute:

- `name`: name of the virtual database to be used in the JDBC URL (`jdbc:cjdbc://host/VirtualDatabaseName`).

- `maxNbOfConnections`: maximum number of concurrent connections accepted for this virtual database. The controller stops accepting client connections when `maxNbOfConnections` concurrent connections are running. Default is 0 (no limit).
- `poolThreads`: if `false`, one thread is created for each connection and dies when the connection closes. If set to `true`, threads are created on-demand and kept in a pool to be reused to serve multiple connections. Default is `true`.
- `minNbOfThreads`: minimum number of threads to keep in the pool (if `poolThreads` is set to `true`). Default is 0.
- `maxNbOfThreads`: maximum number of threads in the pool (if `poolThreads` is set to `true`). Default is 0 (no limit).
- `maxThreadIdleTime`: maximum time in seconds a thread can remain idle before being removed from the pool. Default is 60 seconds (a thread that has not serve any request in the past 60 seconds will be killed).
- `sqlDumpLength`: maximum number of characters of a SQL statement to display in traces and exception messages. 0 means no limit and the full statement is inserted in the message (be careful especially if you are using large objects. Default is 40.

11.2.1. Distribution

A Distribution element defines the virtual database distribution rules so that the virtual database can be shared by several C-JDBC controllers (feature called horizontal scalability). A Distribution element is defined as follows:

- `groupName`: group name to be used by the JGroups communication layer. If no name is provided, the virtual database name is used instead.
- `macroClock`: if a request contains a date macro that needs to be replaced, this attribute defines how the current time is determined. If macroClock is set to none, then each controller uses its local timestamp for all of its backends. If macroClock is set to local, then the macros are replaced using the local time before being sent to remote controllers. Note that in all cases, it is preferable to have controllers synchronized using NTP or an equivalent mechanism.
- `castTimeout`: is the timeout to use when sending request to the group.

```
<!ELEMENT Distribution>
<!ATTLIST Distribution
    groupName  CDATA #IMPLIED
macroClock (none | local) "local"
castTimeout CDATA "0"
>
```

`groupName` defines the group name to be used by the JGroups communication layer. If no name is provided, the virtual database name is used as the default group name.

> **Note:** The JGroups stack configuration is defined in `config/jgroups.xml`. Refer to the JGroups documentation if you want to alter the JGroups configuration.

> **Note:** When a controller fails, all backends attached to it are automatically disabled. A full recovery process is then needed for these nodes. To learn more about this issue, read the horizontal scalability design document (http://c-jdbc.objectweb.org/current/doc/C-JDBC_horizontal_scalability.pdf).

11.2.2. Monitoring

> **Note:** CAREFUL! Monitoring can possibly lead to a memory leak and should only be used on a short period of time There is also a JMX method on the VirtualDatabaseMBean to set this on and off while online:
>
> ```
> void setMonitoringToActive(boolean active) throws VirtualDatabaseException;
> ```

Monitoring provides a generic section for different monitoring modules. At the moment, only "SQLMonitoring" is provided.

```
<!ELEMENT Monitoring (SQLMonitoring*)>

<!ELEMENT SQLMonitoring (SQLMonitoringRule*)>
<!ATTLIST SQLMonitoring
    defaultMonitoring (on | off) "on"
>
```

SQL Monitoring provides statistics (count, error, cache hits, timing) for SQL queries. It is possible to define rules to turn monitoring on or off for specific query patterns.

`defaultMonitoring`: defines the default rule if a request should be monitored (on) or not (off) if no specific rule matches the request.

```
<!ELEMENT SQLMonitoringRule EMPTY>
<!ATTLIST SQLMonitoringRule
    queryPattern    CDATA #REQUIRED
    caseSensitive   (true | false) "false"
    applyToSkeleton (true | false) "false"
    monitoring      (on | off) "on"
>
```

A SQLMonitoringRule Defines a specific monitoring rule for all queries that match the given pattern.

- `queryPattern`: a regular expression understood by the Jakarta Regexp API. For more information on Regexp format, go to Jakarta Regexp web site (http://jakarta.apache.org/regexp).
- `caseSensitive`: true if the pattern matching must be case sensitive.
- `applytoSkeleton`: true if the pattern must apply to the query skeleton (found in PrepareStatement), false if the instantiated query should be used. Example: - *skeleton*: `SELECT * FROM t WHERE x=?` - *instantiated query*: `SELECT * FROM t WHERE x=12`
- `monitoring`: on to activate the monitoring for this rule, off to disable it.

Examples:

- `<SQLMonitoring queryPattern="^delete" monitorRequest="off"/>` will turn monitoring off for all delete queries.
- `<SQLMonitoring queryPattern="select * from users *" monitorRequest="on"/>` will turn monitoring on for all select queries on the users table.

> **Note:** *!Warning!* This is different from `<SQLMonitoring queryPattern="select \* from users *" monitorRequest="on"/>` which turns monitoring on for the "select * from users ..." kind of queries.

11.3. Backup Manager

A Backup Manager defines a number of Backuper in charge of performing backup/restore operations on backends. This element is defined as follows:

```
<!ELEMENT Backup (Backuper+)>

<!ELEMENT Backuper EMPTY>
<!ATTLIST Backuper
    backuperName CDATA #REQUIRED
    className    CDATA #REQUIRED
    options      CDATA #IMPLIED
>
```

A Backuper is defined by a logical backuperName used by the administration console when performing a backup operation. The className specifies the implementation of the Backuper. Backuper specific options can be provided as well (this can be the path to a properties files or a set of attributes). Check your Backuper documentation for its specific options.

Here is an example to use the Octopus Backuper for a virtual database:

```
<Backup>
  <Backuper backuperName="Octopus" className="org.objectweb.cjdbc.controller.backup.OctopusBackuper"/>
</Backup>
```

> **Note:** Octopus does not have access to the C-JDBC classloader for drivers and therefore it needs database drivers to be accessible from the controller classpath. A good solution is to unjar the drivers in the drivers/ directory of C-JDBC.
>
> Octopus dumps are by default stored in a compressed .zip format.

11.4. Authentication Manager

An authentication manager element is defined as follows:

```
<!ELEMENT AuthenticationManager (Admin+, VirtualUsers)>

<!ELEMENT Admin (User+)>

<!ELEMENT User EMPTY>
<!ATTLIST User
    username CDATA #REQUIRED
    password CDATA #REQUIRED
>

<!ELEMENT VirtualUsers (VirtualLogin+)>
```

```
<!ELEMENT VirtualLogin (TrustedLogin*)>
<!ATTLIST VirtualLogin
    vLogin    CDATA #REQUIRED
    vPassword CDATA #REQUIRED
>
<!ELEMENT TrustedLogin EMPTY>
```

An authentication manager defines:

1. an administrator login to be used by the console to access the virtual database administration menu (see Section 7.5) that allows enabling the backends,
2. "virtual logins" that are used by the client application and that are mapped to "real logins" for each backend.
3. "trusted logins" will be used in the future to allow reusing other form of authentication from within C-JDBC.

Here is an example of an authentication manager definition:

```
<AuthenticationManager>
  <Admin>
   <User username="admin" password="adminPwd"/>
  </Admin>
  <VirtualUsers>
    <VirtualLogin vLogin="user1" vPassword="userPwd1"/>
    <VirtualLogin vLogin="user2" vPassword=""/>
  </VirtualUsers>
</AuthenticationManager>
```

In this example, the virtual database has one administrator. The admin can use the login/password "admin/adminPwd" to log in the console.

Two virtual logins are defined: `user1` and `user2` with `userPwd1` and no password, respectively. These logins are those used by the client application and given to the C-JDBC driver.

The connection manager to use with each of the virtual has to be defined in the DatabaseBackend section. Each DatabaseBackend has to define a pool connection manager for each of the virtual user specified here.

11.5. Database Backend

Each database backend must be given a unique name (it is a logical name but it is convenient to use the same name as the real machine name). The database schema is automatically gathered from the backend when it is added to the virtual database. However, you can specify a static database schema (refer to Section 11.5.2) to be used instead. Finally, a specific connection manager (see Section 11.5.3) defines the connection pooling strategy for each virtual login on each backend.

A database backend element is defined as follows:

```
<!ELEMENT DatabaseBackend (DatabaseSchema?, RewritingRule*, ConnectionManager+)>
<!ATTLIST DatabaseBackend
    name                    CDATA #REQUIRED
    driver                  CDATA #REQUIRED
    driverPath              CDATA #IMPLIED
    url                     CDATA #REQUIRED
    connectionTestStatement CDATA #REQUIRED
>
```

Here is a brief description of database backend attributes:

- `name`: the unique logical name identifying this backend.
- `driver`: the database native JDBC driver class name.
- `driverPath`: name of the directory or jar file containing the native driver files. If driverPath is omitted, the driver must be in the `drivers/` directory. If several driver jar files are in the same directory, the first jar file containing the class name specified in the `driver` attribute is used. Note that drivers are loaded in separate classloaders which allows you to use different versions of the same driver on different backends just by specifying the right jar file.
- `url`: the JDBC URL to connect to this database backend.
- `connectionTestStatement`: SQL statement to send on a connection to check if the connection is still valid. This is used when C-JDBC suspects a connection to be broken after the failure of a request. This statement should not update the database because if the connection is still valid the database state should remain the same. Here are the settings for the most popular databases:
  - for MySQL use **select 1**
  - for PostgreSQL use **select now()**.
  - for Apache Derby use **values 1**.
  - for HSQL use **call now()**.
  - for SAP DB (MySQL MaxDB) use **select count(*) from versions**.
  - for Oracle use **select * from dual**.
  - for Firebird use **select * from rdb$types**.
  - for InstantDB use **set date format "yyyy/mm/dd"**.
  - for Interbase use **select * from rdb$types**.
  - for Microsoft SQL server 2000 **select 1**.

Here is a complete example of a database backend element including its connection manager definition:

```
<DatabaseBackend name="node1" driver="org.gjt.mm.mysql.Driver"
      url="jdbc:mysql://node1.objectweb.org/rubis" connectionTestStatement="select 1">
  <ConnectionManager vLogin="user1" rLogin="ruser1" rPassword="rpass1">
    <SimpleConnectionManager/>
  </ConnectionManager>
  <ConnectionManager vLogin="user2" rLogin="ruser1" rPassword="rpass1">
    <VariablePoolConnectionManager initPoolSize="10"
                                   minPoolSize="5"
                                   maxPoolSize="100"/>
  </ConnectionManager>
</DatabaseBackend>
```

11.5.1. Rewriting requests on backends

If your cluster is made of database engines from different vendors, client requests might not be understood by all database backends. If your application was written for PostgreSQL and you want to add MySQL backends, some request might have to be adapted to execute correctly on MySQL. You can specify rules to rewrite queries on the

fly on a specific backend. A RewritingRule element defines how a query matching a given pattern should be rewritten.

```
<!ELEMENT RewritingRule EMPTY>
<!ATTLIST RewritingRule
    queryPattern  CDATA #REQUIRED
    rewrite       CDATA #REQUIRED
    matchingType  (simple | pattern) "simple"
    caseSensitive (true | false) "false"
    stopOnMatch   (true | false) "false"
>
```

- `queryPattern`: SQL query pattern to match.
- `rewrite`: rewritten SQL query.
- `matchingType`: `simple`: means that the first occurrence of queryPattern in the request will be replaced by the string specified in rewrite. `pattern`: uses a pattern based match/replace. A pattern uses `?x` where x is a logical number assigned to the pattern. Example: `select ?1 from ?2 where x=?3`.
- `caseSensitive`: true if matching must be case sensitive.
- `stopOnMatch`: rules are applied in the order they are defined. If one rule matches and stopOnMatch is set to true, next rules are ignored. If stopOnMatch is set to false, if another rule matches the rewritten query, the query will be rewritten again.

Examples:

```
 <RewritingRule queryPattern="from user" rewrite="from "user;""
    matchingType="simple"/>
```

will rewrite the query `select * from user where x=y` as `select * from "user" where x=y`.

```
 <RewritingRule queryPattern="select * from t where x=?1"
    rewrite="select x from y where y=?1" matchingType="pattern"/>
```

will rewrite the query `select * from t where x=435` to `select x from y where y=435`

```
 <RewritingRule queryPattern="?1 LIMIT ?2,?3" rewrite="?1 LIMIT ?3,?2"
    matchingType="pattern"/>
```

will rewrite the query `select * from t limit 10,20` to `select * from t limit 20,10`

11.5.2. Database Schema Definition

DatabaseSchema groups static and dynamic definitions for gathering, constructing and validating the in-memory schema used for load balacing and caching.

A Database schema is defined as follow

```
<!ELEMENT DatabaseSchema (DatabaseStaticSchema?)>
<!ATTLIST DatabaseSchema
 dynamicPrecision (static|table|column|procedures|all) "all"
 gatherSystemTables (true | false) "false"
 schemaName CDATA #IMPLIED
>
```

- `dynamicSchemaPrecision`: if set to `static`, the controller will not check schemas and stored procedures, it will entirely rely on the statically defined schema. If set to something else than "static" it will get information from the backend to check validity of static schema at given level. `table` level will check for table names only, `column` level will check for column names, `procedures` will gather all executable stored procedures. All, includes all information that can be collected.
- `gatherSystemTables`: true if system tables and views should be retrieved, false otherwise (default).
- `schemaName`: if no schemaName is specified all objects visible to the user are gathered, otherwise only the objects belonging to the specified schema are used.

> **Note:** Default option for constructing the schema is to collect all information, even if a static schema is defined especially for checking validity of input. This can be really slow if the database has quite a number of stored procedures defined.

A static database schema can be defined to override the schema automatically gathered by the controller. However, the schema must remain compatible with the schema gathered from the backend.

A database schema element is defined as follows:

```
<!ELEMENT DatabaseStaticSchema (DatabaseProcedure*,DatabaseTable+)>

<!ELEMENT DatabaseProcedure (DatabaseProcedureParameter*)>
<!ATTLIST DatabaseProcedure
     name       CDATA #REQUIRED
     returnType (resultUnknown | noResult | returnsResult) "resultUnknown"
>

<!ELEMENT DatabaseTable (DatabaseColumn+)>
<!ATTLIST DatabaseTable
    tableName   CDATA #REQUIRED
    nbOfColumns CDATA #REQUIRED
>

<!ELEMENT DatabaseColumn EMPTY>
<!ATTLIST DatabaseColumn
    columnName CDATA #REQUIRED
    isUnique   (true | false) "false"
>
```

The `isUnique` attribute should be set to `true` if the column has a `UNIQUE` constraint. This is the case for primary keys (composed primary keys are not yet supported). This affects only cache behavior and select statements parsing.

Here is an example of a database schema definition:

```
<DatabaseStaticSchema>
   <DatabaseTable tableName="users" nbOfColumns="10">
      <DatabaseColumn columnName="id" isUnique="true"/>
      <DatabaseColumn columnName="firstname" isUnique="false"/>
      <DatabaseColumn columnName="lastname" isUnique="false"/>
      <DatabaseColumn columnName="nickname" isUnique="false"/>
```

```
      <DatabaseColumn columnName="password" isUnique="false"/>
      <DatabaseColumn columnName="email" isUnique="false"/>
      <DatabaseColumn columnName="rating" isUnique="false"/>
      <DatabaseColumn columnName="balance" isUnique="false"/>
      <DatabaseColumn columnName="creation_date" isUnique="false"/>
      <DatabaseColumn columnName="region" isUnique="false"/>
   </DatabaseTable>

   <DatabaseTable tableName="regions" nbOfColumns="2">
     <DatabaseColumn columnName="id" isUnique="true"/>
     <DatabaseColumn columnName="name" isUnique="false"/>
   </DatabaseTable>
</DatabaseStaticSchema>
```

11.5.3. Connection Manager

One connection manager must be defined for each virtual login the backend belongs to. The user/password combination defined in the `RealLogin` element for this vLogin/Backend is used to create the connection manager. An example of a connection manager definition is available in Section 11.5.

C-JDBC offers several connection managers that are described hereafter:

- `SimpleConnectionManager`: basic connection manager that opens a new connection on each request and closes it at the end. It is useful if the underlying driver already implements connection pooling for example.
- `FailFastPoolConnectionManager`: offers connection pooling and fails fast when the pool is empty. `poolSize` is the size of the pool.

  All connections are initialized at startup time and if the pool size is too large it is adjusted to the largest number of connections available. Once the pool is empty, `null` is returned instead of a connection. Therefore incoming requests will fail until at least one connection is freed. No system overload should occur with this connection manager, but if the pool size is too small, many requests will fail.

- `RandomWaitPoolConnectionManager`: provides connection pooling and wait when the pool is empty until a connection is freed. This connection manager accepts the following attributes:
  - `poolSize`: this is the size of the pool.
  - `timeout`: this is the maximum time in seconds to wait for a connection to be freed. Default is 0 and means no timeout, that is to say that we wait until one connection is freed.

  All connections are initialized at startup time and if the pool size is too large it is adjusted to the largest number of connections available. Once the pool is empty, the requests wait until a connection is freed or the specified timeout has elapsed. The FIFO[3] order of connection request is not ensured by this connection manager since it relies on the Java wait/notify mechanism.

- `VariablePoolConnectionManager`: provides connection pooling with a dynamically adjustable pool size. This connection manager accepts the following attributes:
  - `initPoolSize`: initial pool size to be intialized at startup.
  - `minPoolSize`: minimum number of connections to keep in the pool. Default is equal to `initPoolSize`.
  - `maxPoolSize`: maximum number of connections in this pool. Default is 0 and means no limit.

- `idleTimeout`: time in seconds a connection can stay idle before being released (removed from the pool). Default is 0 and means that once allocated, connections are never released.
- `waitTimeout`: this is the maximum time in seconds to wait for a connection to be freed. Default is 0 and means no timeout, that is to say that we wait until one connection is freed.

The connection manager element complete definition is as follows:

```
<!ELEMENT ConnectionManager (SimpleConnectionManager |
                             FailFastPoolConnectionManager |
                             RandomWaitPoolConnectionManager |
                             VariablePoolConnectionManager)>
<!ATTLIST ConnectionManager
    Login CDATA #REQUIRED
>

<!ELEMENT SimpleConnectionManager EMPTY>

<!ELEMENT FailFastPoolConnectionManager EMPTY>
<!ATTLIST FailFastPoolConnectionManager
    poolSize CDATA #REQUIRED
>

<!ELEMENT RandomWaitPoolConnectionManager EMPTY>
<!ATTLIST RandomWaitPoolConnectionManager
    poolSize CDATA #REQUIRED
    timeout  CDATA #IMPLIED
>

<!ELEMENT VariablePoolConnectionManager EMPTY>
<!ATTLIST VariablePoolConnectionManager
    initPoolSize CDATA #REQUIRED
    minPoolSize  CDATA #IMPLIED
    maxPoolSize  CDATA #IMPLIED
    idleTimeout  CDATA #IMPLIED
    waitTimeout  CDATA #IMPLIED
>
```

11.6. Request Manager

The request manager is composed of a scheduler (see Section 11.6.2), an optional query cache (see Section 11.6.3), a load balancer (see Section 11.6.4) and an optional recovery log (see Section 11.6.5).

If requests need to be parsed, it can be done sequentially when needed (`backgroundParsing` is set to `false` which is the default value) or forced to be performed in background by a separate thread (it means a new thread is created for each request that need to be parsed).

Parsing is by default case insensitive (`caseSensitiveParsing` is set to `false`) which means that table and column names will be matched to the database schema without checking the case. If you want to enforce the parsing to be case sensitive and reject queries that do not use the same case for table and column names as the ones fetched from the database, set `caseSensitiveParsing` to `true`.

A timeout in seconds can be defined for begin/commit/rollback operations. If no value is given, the default timeout is set to 60 seconds. Warning! A value of 0 means no timeout and waits forever until completion.

The request manager element definition is as follows:

```
<!ELEMENT RequestManager (RequestScheduler, RequestCache?, LoadBalancer, RecoveryLog?)>
<!ATTLIST RequestManager
    backgroundParsing    (true | false) #IMPLIED
    caseSensitiveParsing (true | false) #IMPLIED
    beginTimeout      CDATA #IMPLIED
    commitTimeout     CDATA #IMPLIED
    rollbackTimeout   CDATA #IMPLIED
>
```

11.6.1. Macros Handler

C-JDBC can interpret and replace on-the-fly macros with a value computed by the controller (the RequestManager in fact). This prevents different backends to generate different values when interpreting the macros which could result in data inconsitencies. The supported macros are the following:

- `rand`: RAND() can be replaced with an int, long, float or double value.

all the date macros (now, currentDate, currentTime, timeOfDay and currentTimestamp) can be replaced by one of the following:

- `off`: do not replace the macro

  `date`: java.sql.Date.toString() build from current time at controller (example: 2001-02-17)

  `time`: java.sql.Time.toString() build from current time at controller (example: 19:07:32).

  `timestamp`: java.sql.Timestamp.toString() build from current time at controller (example: 2001-02-17 19:07:32-05).

- `timeResolution`:defines the timer precision to use when rewriting a query that contains a date macro. Default is 0 millisecond which is the highest precision. A value of 1000 corresponds to a 1 second precision, 60000 to a 1 minute precision and so on.

The MacroHandling element definition is as follows:

```
<!ELEMENT MacroHandling EMPTY>
<!ATTLIST MacroHandling
        rand (off | int | long | float | double) "float"
        now  (off | date | time | timestamp) "timestamp"
        currentDate (off | date | time | timestamp) "date"
        currentTime (off | date | time | timestamp) "time"
        timeOfDay (off | date | time | timestamp) "timestamp"
        currentTimestamp (off | date | time | timestamp) "timestamp"
        timeResolution CDATA "0"
>
```

Note: A default Macrohandling element is instanciated and used if nothing is specified in the configuration file.

11.6.2. Request Scheduler

The request scheduler is responsible for scheduling the requests and ensuring a serializable execution order. Different schedulers are provided for each RAIDb level (see Section 10). Optimized schedulers are also provided for use with a single database backend (SingleDB configuration).

The request scheduler element definition is as follows:

```
<!ELEMENT RequestScheduler (SingleDBScheduler | RAIDb-0Scheduler |
                            RAIDb-1Scheduler | RAIDb-2Scheduler)>

<!ELEMENT SingleDBScheduler EMPTY>
<!ATTLIST SingleDBScheduler
    level (passThrough | pessimisticTransaction) #REQUIRED
>

<!ELEMENT RAIDb-0Scheduler EMPTY>
<!ATTLIST RAIDb-0Scheduler
    level (passThrough | pessimisticTransaction) #REQUIRED
>

<!ELEMENT RAIDb-1Scheduler EMPTY>
<!ATTLIST RAIDb-1Scheduler
    level (passThrough | query | optimisticQuery | optimisticTransaction |
           pessimisticTransaction) #REQUIRED
>

<!ELEMENT RAIDb-2Scheduler EMPTY>
<!ATTLIST RAIDb-2Scheduler
    level (passThrough | query | pessimisticTransaction) #REQUIRED
>
```

Here is a brief definition of the meaning of each scheduler:

- `passThrough`: queries are just assigned a unique idnetifier and forwarded as is to the load balancer letting each database perform the scheduling and the locking. Therefore you will obtain the locking granularity provided by the database which should be row-level locking. The load balancer will just ensure that the writes are sent in the same order to all backends.

  `query`: the query level scheduler is the most basic scheduler that ignores transactions and schedule requests as they come. Reads can execute concurrently but a write blocks every other request execution until it completes. This scheduling might lead to database deadlocks under moderate load. Only use this scheduler if the requests are already scheduled upfront by the application.

- `optimisticTransaction`: this is an optimistic transactional level scheduler that takes care of scheduling transactions in an optimistic way (writes on different tables can occur in parallel). An optimistic scheduler does not provide deadlock detection to rollback transactions that are deadlocking, but lets the database resolve the deadlocks and rollback the appropriate transactions.
- `pessimisticTransaction`: this is a pessimistic transactional level scheduler that schedules transactions in a safe way (without possible deadlocks) but providing less parallelism for writes compared to optimistic scheduling (this is only sensitive on write heavy workloads). This is the safest scheduler and it is recommended for most configurations.

11.6.3. Request Cache

A Request Cache can be composed of different caches that differ in the type of data they cache:

- `MetadataCache`: this cache improves the ResultSet creation time by keeping the various field information with their metadata. It is strongly encouraged to use this cache that reduces both cpu and memory usage.
- `ParsingCache`: allows to parse a request only once for all its executions. This reduces the cpu load on the controller.
- `ResultCache`: this cache keeps the results associated to a given request. Cache entries can be invalidated according to various policies. This cache reduces the load on the database backends.

A RequestCache element is defined as follows:

```
<!ELEMENT RequestCache (MetadataCache?, ParsingCache?, ResultCache?)>
```

11.6.3.1. Metadata Cache

The MetadataCache caches ResultSet metadata and fields meta information associated to a query execution so that each time a query is executed, we don't have to gather all metadata from the underlying driver and we can build C-JDBC ResultSet much more efficiently both in terms of speed and memory usage. Note that if you use PreparedStatements, the query skeleton is used for matching the cache instead of the instanciated query.

Example: `SELECT * FROM t WHERE x=?` hits on the same cache entry for all queries of this form for any value of x.

A MetadataCache element is defined as follows:

```
<!ELEMENT MetadataCache EMPTY>
<!ATTLIST MetadataCache
    maxNbOfMetadata CDATA "10000"
    maxNbOfField    CDATA "0"
>
```

`maxNbOfMetadata`: maximum number of metadata entries to keep in the cache (default is 10000 and 0 means unlimited)

`maxNbOfField`: maximum number of field entries to keep in the cache (0 means unlimited and is the default setting).

11.6.3.2. Parsing Cache

Parsing requests is a resource consuming process. The ParsingCache caches the result of the parsing processing so that a request is only parsed once for all its executions. If you are using PreparedStatements, the ParsingCache can store the query skeleton meaning that the cached parsing will match any instances of the skeleton.

Example: `SELECT * FROM t WHERE x=?` will be parsed only once for any value of x.

A MetadataCache element is defined as follows:

```
<!ELEMENT ParsingCache EMPTY>
<!ATTLIST ParsingCache
    backgroundParsing (true | false) "false"
    maxNbOfEntries    CDATA "5000"
>
```

Request parsing can be done sequentially when needed (backgroundParsing set to false which is the default value) or forced to be performed in background by a separate thread (it means a new thread is created for each request that need to be parsed).

`maxNbOfEntries`: Defines the maximum number of entries to keep in the cache. The cache uses a LRU (Least Recently Used) replacing policy meaning that the oldest entries from the cache are removed when it is full. Default is 0 and means no limit on cache size.

11.6.3.3. Result Cache

The ResultCache caches results of queries. A query and its ResultSet are stored in the cache so that if the sames query is executed, the ResultSet stored in the cache is returned.

ResultCacheRule elements define the cache coherency and policy. Default cache behavior is eager consistency for all queries (ResultSet returned by the cache are always coherent and up-to-date). See below (ResultCacheRule element) to relax the cache consistency to achieve better performance. Note that ResultSet caching is disabled if no result cache element is found in the configuration file

If two exact same requests are to be executed at the same time, only one is executed and the second one waits until the completion of the first one (this is the default `pendingTimeout` value which is 0). To prevent the second request from waiting forever, a `pendingTimeout` value in seconds can be defined for the waiting request. If the timeout expires, the request is executed in parallel with the first one.

A result cache element is described as follows:

```
<!ELEMENT ResultCache (DefaultResultCacheRule?, ResultCacheRule*)>
<!ATTLIST ResultCache
    granularity   (database | table | column | columnUnique) "database"
    maxNbOfEntries CDATA "100000"
    pendingTimeout CDATA "0"
>
```

The result cache `granularity` defines how entries are removed from the cache. `database` flush the whole cache on every write access. This is the default cache setting. `table` and `column` provide table-based and column-based invalidations, respectively. `columnUnique` can optimize requests that select a unique primary key (useful with EJB entity beans).

You can specify the maximum number of entries (default is 100000) to limit the cache size. This is obviously not as efficient as setting a cache size, but in the latter case we would have to spend a lot of time computing size of result sets from queries (Java does not provide a sizeof operator!). We offer size display in bytes when viewing the cache from the console though.

Finer grain tuning of the cache is based on rules matching query pattern. A queryPattern are regular expressions to match according to Jakarta Regexp (see their web site (http://jakarta.apache.org/regexp) for more information). A default cache rule defines the policy if no other rule matches:

```
<!ELEMENT DefaultResultCacheRule (NoCaching | EagerCaching | RelaxedCaching)>
<!ATTLIST DefaultResultCacheRule
    timestampResolution CDATA "1000"
>

<!ELEMENT ResultCacheRule (NoCaching | EagerCaching | RelaxedCaching)>
<!ATTLIST ResultCacheRule
    queryPattern        CDATA #REQUIRED
    caseSensitive       (true | false) "false"
    applyToSkeleton     (true | false) "false"
```

```
    timestampResolution CDATA "1000"
>
```

- `queryPattern`: the regular expression to match.
- `caseSensitive`: true if the pattern matching must be case sensitive
- `applytoSkeleton`: true if the pattern must apply to the query skeleton (found in PrepareStatement), false if the instantiated query should be used. Example: skeleton is `SELECT * FROM t WHERE x=?` and instantiated query is `SELECT * FROM t WHERE x=12`.
- `timestampResolution`: If a query contains a NOW() macro, it is replaced with the current date on the controller. timestampResolution indicates the resolution (in milliseconds) to use when replacing NOW() with the current date. If the resolution is below 1 second (value <1000ms), the request is never kept in the cache because there is almost no chance that the same request will come with the same timestamp. Note that this timestamp is for the cache only and you can use a greater resolution for the load balancer (see below).

  Note: If timestampResolution is set to 60000, every execution of a query like `SELECT * FROM x WHERE date=NOW()` will be replaced with the same value for 1 minute (i.e. `SELECT * FROM x WHERE date="2012-11-15 08:03:00.000"`) and therefore the cache entry may be hit for 1 minute.

To define a default rule that disable caching use:

```
<DefaultResultCacheRule>
  <NoCaching/>
</DefaultResultCacheRule>
```

If no default rule is provided, the following default rule is assumed:

```
<ResultCacheRule queryPattern="default" timestampResolution="1000">
  <EagerCaching/>
</ResultCacheRule>
```

Each cache rule can have a different caching behavior. The available behavior are the following:

```
<!ELEMENT NoCaching EMPTY>
<!ELEMENT EagerCaching EMPTY>
<!ELEMENT RelaxedCaching EMPTY>
<!ATTLIST RelaxedCaching
     timeout        CDATA "60"
     keepIfNotDirty (true | false) "true"
>
```

- `NoCaching` means we do not put the match in the cache
- `EagerCaching` means that all entries in the cache are always coherent and any update query (insert,delete,update,...) will automatically invalidate the corresponding entry in the cache. This was the previous cache behavior for all queries
- `RelaxedCaching` means that a `timeout` is set for this entry and the entry is kept in the cache until the timeout expires. When the timeout expires, if no write has modified the corresponding result and `keepIfNotDirty` is set to true, the entry is kept in the cache and the timeout is rearmed (reset) with its initial value.

Note: RelaxedCaching may provide stale data. The timeout defines the maximum staleness of a cache entry. It means that the cache may return an entry that is out of date.

Here is a cache rule example:

```
<ResultCacheRule queryPattern="select ? from b where id=?" applyToSkeleton="true">
  <RelaxedCaching timeOut="6000" keepIfNotDirty="true"/>
</ResultCacheRule>
```

11.6.4. Load Balancer

The load balancer defines the way requests will be distributed among the backends according to a RAIDb level (see Section 10). The following load balancers are available:

- `SingleDB`: load balancer for a single database backend instance. This is only available if you use a single controller.
- `ParallelDB`: load balancer to use with a parallel database such as Oracle Parallel Server or Middle-R. Both read and write are load balanced on the backends, letting the parallel database replicating writes.
- `RAIDb-0`: full database partitioning (no table can be replicated) with an optional policy specifying where new tables are created.
- `RAIDb-1`: full database mirroring (all tables are replicated everywhere) with an optional policy specifying how distributed queries (writes/commit/rollback) completion is handled (when the first, a majority or all backends complete).
- `RAIDb-1ec`: full database mirroring (like RAIDb-1) with error checking for byzantine failure detection.
- `RAIDb-2`: partial replication (each table must be at least replicated once) with optional policies for new table creation (like RAIDb-0) and distributed queries completion (like RAIDb-1).
- `RAIDb-2ec`: partial replication (like RAIDb-2) with error checking for byzantine failure detection.

The load balancer element definition is as follows:

```
<!ELEMENT LoadBalancer (SingleDB | ParallelDB | RAIDb-0 | RAIDb-1 | RAIDb-1ec | RAIDb-2 | RAIDb-2ec)>
```

11.6.4.1. `SingleDB` load balancer

The `SingleDB` load balancer does not need any specific parameter. The definition of the `SingleDB` element is as follows:

```
 <!ELEMENT SingleDB EMPTY>
```

11.6.4.2. `ParallelDB` load balancer

The `ParallelDB` load balancer must be used with a SingleDB request scheduler. This load balancer provides two implementations: `ParallelDB-RoundRobin` and `ParallelDB-LeastPendingRequestsFirst` providing round robin and least pending request first load balancing policies, respectively. ParallelDB load balancers are designed to provide load balancing and failover on top of parallel databases such as Oracle Parallel Server or Middle-R. It means that read and write requuest are just sent to one alive backends, the parallel database being responsible for maintaining the consistency between the backends. The definition of the `ParallelDB` element is as follows:

```
<!ELEMENT ParallelDB (ParallelDB-RoundRobin | ParallelDB-LeastPendingRequestsFirst)>

<!ELEMENT ParallelDB-RoundRobin EMPTY>
<!ELEMENT ParallelDB-LeastPendingRequestsFirst EMPTY>
```

No specific settings are required for these load balancers. They do not require request parsing which means that requests are just forwarded as is to the backends (rewriting rules are still applied but no automatic transformation is performed).

11.6.4.3. `RAIDb-0` load balancer

The `RAIDb-0` load balancer accepts a policy to specify where new tables are created. The definition of the `RAIDb-0` element is as follows:

```
<!ELEMENT RAIDb-0 (MacroHandling?,CreateTable*)>

<!ELEMENT CreateTable (BackendName*)>
<!ATTLIST CreateTable
    tableName     CDATA #IMPLIED
    policy        (random | roundRobin | all) #REQUIRED
    numberOfNodes CDATA #REQUIRED
>

<!-- BackendName simply identifies a backend by its logical name -->
<!ELEMENT BackendName EMPTY>
<!ATTLIST BackendName
    name CDATA #REQUIRED
>
```

If `MacroHandling` is omitted, a default MacroHandling element is added.

`CreateTable` defines the policy to adopt when creating a new table. This policy is based on the given list of `BackendName` nodes (which might be a subset of the complete set of backends). If the backend list is omitted, then all enabled backends are taken at decision time. The attributes have the following meaning:

- `numberOfNodes` represents the number of backends to pickup from the `BackendName` list to apply the policy (it must be set to 1 for `RAIDb-0` load balancers and can never be greater than the number of nodes declared in the `BackendName` list).
- `policy` works as follows:
  - `random`: `numberOfNodes` backends are picked up randomly from the `BackendName` list and the table is created on these nodes.
  - `roundRobin`: `numberOfNodes` backends are picked up from the `BackendName` list using a round-robin algorithm and the table is created on these nodes.
  - `all`: the table is created on *all* nodes in the `BackendName` list (`numberOfNodes` is ignored).

Here is an example of a `RAIDb-0` controller with three nodes where new tables are created randomly on the first two nodes:

```
...
<DatabaseBackend name="node1" ...
<DatabaseBackend name="node2" ...
<DatabaseBackend name="node3" ...
```

```
...

<LoadBalancer>
  <RAIDb-0>
    <CreateTable policy="random" numberOfNodes="1">
      <BackendName name="node1" />
      <BackendName name="node2" />
    </CreateTable>
  </RAIDb-0>
</LoadBalancer>
```

11.6.4.4. `RAIDb-1:full mirroring` load balancer

A RAIDb-1 load balancer is defined as follows:

```
<!ELEMENT RAIDb-1 (WaitForCompletion?, MacroHandling?, (RAIDb-1-RoundRobin |
           RAIDb-1-WeightedRoundRobin | RAIDb-1-LeastPendingRequestsFirst))>

<!ELEMENT RAIDb-1-RoundRobin EMPTY>
<!ELEMENT RAIDb-1-WeightedRoundRobin (BackendWeight)>
<!ELEMENT RAIDb-1-LeastPendingRequestsFirst EMPTY>

<!ELEMENT WaitForCompletion EMPTY>
<!ATTLIST WaitForCompletion
    policy (first | majority | all) "first"
>

<!ELEMENT BackendWeight EMPTY>
<!ATTLIST BackendWeight
    name   CDATA #REQUIRED
    weight CDATA #REQUIRED
>
```

If `WaitForCompletion` is omitted, the default behaviour is to return the result as soon as one backend has completed.

If MacroHandling is omitted, a default MacroHandling element is added.

The `RAIDb-1` load balancer accepts a policy to specify distributed queries completion. Several load balancing policies are proposed:

- `RoundRobin`: simple round-robin load balancing. The first request is sent to the first node, the second request to the second node, etc... Once a request has been sent to the last backend, the next request is sent to the first backend and so on.
- `WeightedRoundRobin`: same as round-robin but a weight is associated to each backend. A backend that has a weight of 2 will get two times more requests than a backend with a backend with a weight of 1.
- `LeastPendingRequestsFirst`: the request is sent to the backend that has the least pending requests to execute (that can be considered as the shortest pending request queue).

The definition of the `RAIDb-1` element is as follows:

`WaitForCompletion` defines the policy to adopt when waiting for the completion of a request. Policy works as follows:

- `first`: returns the result as soon as one node has completed.
- `majority`: returns the result as soon as a majority of nodes (n/2+1) has completed.
- `all`: waits for all nodes to complete before returning the result to the client.

11.6.4.5. `RAIDb-1ec` load balancer

The RAIDb-1 with error checking must provide an error checking policy (defined below). The optional `WaitForCompletion` policy only concern write requests (`INSERT`, `DELETE`, `UPDATE`, commit, ...).

> **Note:** `RAIDb-1ec` is not operational in C-JDBC v1.0alpha.

The definition of the `RAIDb-1ec` element is as follows:

```
<!ELEMENT RAIDb-1ec (WaitForCompletion?, ErrorChecking, (RAIDb-1ec-RoundRobin |
                     RAIDb-1ec-WeightedRoundRobin))>
<!ATTLIST RAIDb-1ec
    nbOfConcurrentReads CDATA #REQUIRED
>

<!ELEMENT RAIDb-1ec-RoundRobin EMPTY>
<!ELEMENT RAIDb-1ec-WeightedRoundRobin (BackendWeight)>

<!ELEMENT ErrorChecking EMPTY>
<!ATTLIST ErrorChecking
    policy        (random | roundRobin | all) #REQUIRED
    numberOfNodes CDATA #REQUIRED
>
```

Error checking policy (for RAIDb-1ec and RAIDb2-ec). Error checking is used to detect byzantine failures of nodes. It means detecting when a node sends funny results in a non-deterministic way. Error checking allows read queries to be sent to more than one database, and the results are compared. A majority of nodes must agree on the result that will be sent to the client. Error checking policies are defined as follows:

- `random`: `numberOfNodes` backends are picked up randomly; the read request is sent to these backends and results are compared.
- `roundRobin`: `numberOfNodes` backends are picked up using a round-robin algorithm ; the read request is send to these backends and results are compared.
- `all`: the request is sent to *all* nodes (`numberOfNodes` is ignored) and the results compared.

`numberOfNodes` must be greater or equal to 3.

11.6.4.6. `RAIDb-2 : distributed mirroring` load balancer

The definition of the `RAIDb-2` element is as follows:

```
<!ELEMENT RAIDb-2 (CreateTable*, WaitForCompletion?, MacroHandling?, (RAIDb-2-RoundRobin |
 RAIDb-2-WeightedRoundRobin | RAIDb-2-LeastPendingRequestsFirst))>


<!ELEMENT RAIDb-2-RoundRobin EMPTY>
```

```
<!ELEMENT RAIDb-2-WeightedRoundRobin (BackendWeight)>
<!ELEMENT RAIDb-2-LeastPendingRequestsFirst EMPTY>
```

If MacroHandling is omitted, a default MacroHandling element is added.

The `RAIDb-2` load balancer accepts a policy to specify where new tables are created and how distributed queries completion should be handled. Several load balancing policies are proposed:

- `RoundRobin`: simple round-robin load balancing. The first request is sent to the first node, the second request to the second node, etc... Once a request has been sent to the last backend, the next request is sent to the first backend and so on.
- `WeightedRoundRobin`: same as round-robin but a weight is associated to each backend. A backend that has a weight of 2 will get two times more requests than a backend with a backend with a weight of 1.
- `LeastPendingRequestsFirst`: the request is sent to the backend that has the least pending requests to execute (that can be considered as the shortest pending request queue).

The `CreateTable` element definition is defined in Section 11.6.4.3.

The `WaitForCompletion` element definition is defined in Section 11.6.4.4.

11.6.4.7. `RAIDb-2ec` load balancer

The RAIDb-2 with error checking must provide an error checking policy as in `RAIDb-1ec` (see Section 11.6.4.5). The other elements are similar to those defined for `RAIDb-2` controller (see Section 11.6.4.6).

> **Note:** `RAIDb-2ec` is not operational in C-JDBC v1.0alpha.

The definition of the `RAIDb-2ec` element is as follows:

```
<!ELEMENT RAIDb-2ec (CreateTable*, WaitForCompletion?, ErrorChecking,
                     (RAIDb-2ec-RoundRobin | RAIDb-2ec-WeightedRoundRobin))>
<!ATTLIST RAIDb-2ec
    nbOfConcurrentReads CDATA #REQUIRED
>

<!ELEMENT RAIDb-2ec-RoundRobin EMPTY>
<!ELEMENT RAIDb-2ec-WeightedRoundRobin (BackendWeight)>
```

11.6.5. Recovery Log

The C-JDBC Recovery Log stores write queries and transactions between logical checkpoints defined by the user. The log can be only be stored in a database (or cluster of databases) using a `JDBCRecoveryLog` element.

The definition of a `RecoveryLog` element is as follows:

```
<!ELEMENT RecoveryLog (JDBCRecoveryLog)>
```

11.6.5.1. JDBCRecoverylog

The JDBCRecoveryLog stores the recovery information in a database. To access this database, you must provide the driver class name to load (driver), an optional jar file or directory where to find the class to load (driverPath), the JDBC url to access the database as well as a valid login/password.

A timeout in seconds can be defined for the sql requests. If no value is given, the default timeout is set to 60 seconds. Warning! 0 means no timeout and wait forever until completion.

recoveryBatchSize is used to speedup the recovery process and allow several queries to be accumulated into a batch on the recovering backend. Increasing this value beyond a certain limit will not increase performance and will consume a significant amount of memory. Default is 10 and minimum is 1.

The recovery information is stored in 3 tables defined in the RecoveryLogTable, CheckpointTable and BackendLogTable elements.

The definition of a `JDBCRecoveryLog` element is as follows:

```
<!ELEMENT JDBCRecoveryLog (RecoveryLogTable, CheckpointTable, BackendTable)>

<!ATTLIST JDBCRecoveryLog
    driver            CDATA #REQUIRED
    driverPath        CDATA #IMPLIED
    url               CDATA #REQUIRED
    login             CDATA #REQUIRED
    password          CDATA #REQUIRED
    requestTimeout    CDATA "60"
    recoveryBatchSize CDATA "10"
>
<!ELEMENT RecoveryLogTable EMPTY>
<!ATTLIST RecoveryLogTable
    createTable              CDATA "CREATE TABLE"
    tableName                CDATA "logtable"
    idColumnType             CDATA "BIGINT NOT NULL UNIQUE"
    vloginColumnType         CDATA "VARCHAR(20) NOT NULL"
    sqlColumnName            CDATA "sql"
    sqlColumnType            CDATA "TEXT NOT NULL"
    transactionIdColumnType  CDATA "BIGINT NOT NULL"
    extraStatementDefinition CDATA ""
>

<!ELEMENT CheckpointTable EMPTY>
<!ATTLIST CheckpointTable
    createTable              CDATA "CREATE TABLE"
    tableName                CDATA "checkpointtable"
    checkpointNameColumnType CDATA "VARCHAR(127) NOT NULL"
    requestIdColumnType      CDATA "BIGINT"
    extraStatementDefinition CDATA ",PRIMARY KEY (name)"
>

<!ELEMENT BackendTable EMPTY>
<!ATTLIST BackendTable
    createTable                  CDATA "CREATE TABLE"
    tableName                    CDATA "backendtable"
    databaseNameColumnType       CDATA "VARCHAR(50) NOT NULL"
    backendNameColumnType        CDATA "VARCHAR(50) NOT NULL"
    backendStateColumnType       CDATA "INTEGER"
    checkpointNameColumnType     CDATA "VARCHAR(127) NOT NULL"
    extraStatementDefinition     CDATA ""
```

```
>

<!ELEMENT DumpTable EMPTY>
<!ATTLIST DumpTable
    createTable              CDATA "CREATE TABLE"
    tableName                CDATA "dumptable"
    dumpNameColumnType       CDATA "TEXT NOT NULL"
    dumpDateColumnType       CDATA "TEXT NOT NULL"
    dumpPathColumnType       CDATA "TEXT NOT NULL"
    dumpFormatColumnType     CDATA "TEXT NOT NULL"
    checkpointNameColumnType CDATA "TEXT NOT NULL"
    backendNameColumnType    CDATA "TEXT NOT NULL"
    tablesColumnType         CDATA "TEXT NOT NULL"
    extraStatementDefinition CDATA ""
>
```

The `JDBCRecoveryLog` element requires the following attributes:

- `driver`: the driver class name
- `url`: the JDBC URL to access the database
- `login`: the user login to connect to the database
- `password`: the user password to connect to the database
- `requestTimeout`: optional timeout request in second that will be used to replay the log queries. Default timeout is 60 seconds and 0 means no timeout (wait forever until a request complete).
- `recoveryBatchSize`: used to speedup the recovery process and allow several queries to be accumulated into a batch on the recovering backend. Increasing this value beyond a certain limit will not increase performance and will consume a significant amount of memory. Default is 10 and minimum is 1.

The `RecoveryLogTable` defines how the JDBCRecoveryLog log table is created. The log table name (`tableName`) must conform to the syntax of a database table name. The log table stores a unique request id (`id`), the virtual login (`vlogin`)to use to execute the sql statement (`sql`) in the given transaction (`transactionId`). The statement used by the JDBCRecoveryLog to create the log table uses the RecoveryLogTable attributes as follows:

```
CREATE TABLE tableName (
  id             idColumnType,
  vlogin         vloginColumnType,
  sql            sqlColumnType,
  transaction_id transactionIdColumnType
  extraStatementDefinition)
```

If all default values are used, the log table is created using the following statement:

```
CREATE TABLE logTable (
  id             INTEGER NOT NULL UNIQUE,
  vlogin         VARCHAR (20) NOT NULL,
  sql            TEXT NOT NULL,
  transaction_id INTEGER NOT NULL
  )
```

The `CheckpointTable` stores the checkpoint name and the corresponding index in the recovery log table. The statement used by the JDBCRecoveryLog to create the checkpoint table uses the CheckpointTable attributes as follows:

```
CREATE TABLE tableName (
  name       checkpointNameColumnType,
  request_id requestIdColumnType
  extraStatementDefinition)
```

If all default values are used, the log table is created using the following statement:

```
CREATE TABLE checkpointTable (
  name       VARCHAR (20) NOT NULL,
  request_id INTEGER,
  PRIMARY KEY(name))
```

The `BackendLogTable` stores the states of the different backends of a virtual database. It stores the name of the backend, the database it belongs to and the last known checkpoint of a backend when the backend is disabled, and the state the backend was in when the database was last shutdown. If all default values are used, the log table is created using the following statement:

```
CREATE TABLE backendTable (
  databaseName VARCHAR(50) NOT NULL,
  backendName  VARCHAR(50) NOT NULL,
  backendState INTEGER,
  checkpointName VARCHAR(50) NOT NULL
  )
```

Here is an example on how to define a JDBCRecoveryLog to work with a HSQL database:

```
<RecoveryLog>
   <JDBCRecoveryLog driver="org.hsqldb.jdbcDriver" url="jdbc:hsqldb:hsql://localhost" login="sa" password="">
      <RecoveryLogTable
         tableName="recovery"
         idColumnType="INTEGER NOT NULL"
         sqlColumnType="VARCHAR NOT NULL"
        extraStatementDefinition=",PRIMARY KEY (id)"/>
      <CheckpointTable tableName="checkpoint"/>
  <BackendLogTable tableName="backendTable"/>
   </JDBCRecoveryLog>
</RecoveryLog>
```

The `DumpTable` stores the dump names and associated meta-data such as the corresponding checkpoint name. The statement used by the JDBCRecoveryLog to create the dump table uses the DumpTable attributes as follows:

```
createTable tableName (
  dump_name       dumpNameColumnType,
  dump_date       dumpDateColumnType,
  dump_path       dumpPathColumnType,
  dump_format     dumpTypeColumnType,
  checkpoint_name checkpointNameColumnType,
  backend_name    backendNameColumnType,
  tables          tablesColumnType
```

```
    extraStatementDefinition)
```

dump_name is the dump logical name, dump_date the date at which the backup was started, dump_path the path where the dump can be found, dump_format an implementation specific text form that specifies the method used for the dump, checkpoint_name is the name of the checkpoint associated to this dump, tables is the list of tables that are contained in this dump (* means all tables). If all default values are used, the log table is created using the following statement:

```
CREATE TABLE DumpTable (
  dump_name       TEXT NOT NULL,
  dump_date       DATE,
  dump_path       TEXT NOT NULL,
  dump_format     TEXT NOT NULL,
  checkpoint_name TEXT NOT NULL,
  backend_name    TEXT NOT NULL,
  tables          TEXT NOT NULL
  )
```

11.7. SSL Configuration

SSL may be used for encryption as well as authentication for all connections to cjdbc.

SSL support for c-jdbc is based on the Java Secure Socket Extension (JSSE). JSSE has been integrated into the Java 2 SDK, Standard Edition, v 1.4. For java 1.3 it can be installed as an optional package. (available at http://java.sun.com/products/jsse/)

11.7.1. Controller

On the controller side ssl can be configured for all jmx connections and the virtual database accessed via the cjdbc driver with the xml element SSL in the controller configuration :

```
<!ELEMENT SSL EMPTY>
<!ATTLIST SSL
    keyStore              CDATA        #IMPLIED
    keyStorePassword      CDATA        #IMPLIED
    keyStoreKeyPassword   CDATA        #IMPLIED
    isClientAuthNeeded    (true|false) "false"
    trustStore            CDATA        #IMPLIED
    trustStorePassword    CDATA        #IMPLIED
>
```

- keyStore: The file where the keys are stored
- keyStorePassword: the password to the keyStore
- keyStoreKeyPassword: the password to the private key, if none is specified the same password as for the store is used

- isClientAuthNeeded: if set to false ssl is used for encryption only, it true set to true then the server is only accepting trusted clients (the client certificate has to be in the trusted store)
- trustStore: the file where the trusted certificates are stored, if none is specified the same store as for the key is used
- trustStorePassword: the password to the trustStore, if none is specified the same password as for the keyStore is used

11.7.2. Console / Jmx Clients

The console and other jmx clients are configured with the use of java properties :

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword

Example : -Djavax.net.ssl.trustStore=client.keystore -Djavax.net.ssl.trustStorePassword=clientpassword

11.7.3. Driver

SSL on the driver side is configured with java properties

- cjdbc.ssl.enabled=true
- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword

Example : -Djavax.net.ssl.trustStore=client.keystore -Djavax.net.ssl.trustStorePassword=clientpassword

11.7.4. Certificates (public and private keys)

You may create your certificates with the keytool (part of jsse)

1. Create a self-signed server and a self-signed client key each in its own keystore

```
$> keytool -genkey -v -keyalg RSA -keystore server.keystore -dname "CN=Server, OU=Bar, O=Foo, L=Some, ST=W
$> keytool -genkey -v -keyalg RSA -keystore client.keystore -dname "CN=Client, OU=Bar, O=Foo, L=Some, ST=W
```

2. Export the server's and the client's public keys from their respective keystores

```
$> keytool -export -rfc -keystore server.keystore -alias mykey -file server.public-key
$> keytool -export -rfc -keystore client.keystore -alias mykey -file client.public-key
```

3. Import the client's public key to the server's keystore, and vice-versa:

```
$> keytool -import -alias client -keystore server.keystore -file client.public-key
$> keytool -import -alias server -keystore client.keystore -file server.public-key
```

11.8. Configuration Examples

Configuration files examples are available in the C-JDBC distribution in the `/c-jdbc/doc/examples` directory.

Here is a brief overview of each example content:

- `Cache`: gives various configuration examples on how to use the cache.
- `Derby` : contains examples for the Apache Derby database including the ones used in the ApacheCon demos.
- `HorizontalScalbility` : provides configuration files to create a distributed virtual database on 2 controllers. One file should be loaded on each of the two controllers.
- `LinuxService` and `SuSE` : contains examples to run C-JDBC controller as a Linux service.
- `SingleDB`: a C-JDBC configuration with a unique MySQL backend.
- `RAIDb-0`: C-JDBC configuration examples for RAIDb-0.
  - `RAIDb-0.xml`: a simple 2 nodes RAIDb-0 configuration.
  - `RAIDb-0-schema.xml`: a 2 nodes RAIDb-0 configuration using a static database schema definition matching the RUBiS benchmark database schema.
- `RAIDb-1`: contains various RAIDb-1 configuration examples.
- `RAIDb-2`: contains various RAIDb-2 configuration examples.
- `RecoveryLog`: gives an example of a fault tolerant recovery log.

12. Request Player

C-JDBC comes with a tool called 'Request Player' that allows to replay queries that have been recorded using the logging facility of the controller. This is useful for both debugging and performance tuning.

12.1. Recording a request trace

There is a specific logger in the `log4j.properties` configuration file is located in the `/c-jdbc/config` directory of your installation. To learn more about configuring the log report to Section 6.4. To turn on the request tracing, update your `log4j.properties` with the following logger declaration:

```
# To trace requests #
log4j.logger.org.objectweb.cjdbc.controller.VirtualDatabase.request=INFO, Requests
log4j.additivity.org.objectweb.cjdbc.controller.VirtualDatabase.request=false
```

The trace file is stored as defined in the logger definition. Here is the default definition that is shipped with C-JDBC:

```
log4j.appender.Requests=org.apache.log4j.RollingFileAppender
log4j.appender.Requests.File=request.log
log4j.appender.Requests.MaxFileSize=100MB
log4j.appender.Requests.MaxBackupIndex=5
log4j.appender.Requests.layout=org.apache.log4j.PatternLayout
log4j.appender.Requests.layout.ConversionPattern=%d{ABSOLUTE} %c{1} %m\n
```

You can set the trace file name (you can also provide a path) in the `log4j.appender.Requests.File` property. The `log4j.appender.Requests.MaxFileSize` property defines the maximum trace file size. Finally, `log4j.appender.Requests.MaxBackupIndex` defines the number of trace files that will be generated. For example, in the above configuration, the trace will be made of 5 files of 100MB.

The file format expected by the request player is as follows:

```
date virtualDatabaseName requestType transactionId SQL
```

`requestType` is `B` for begin, `S` for select statements, `W` for write statements (insert, update, delete, create, drop), `C` for commit and `R` for rollback.

Here is an example of a trace of transaction n?27562:

```
10:34:22,775 tpcw B 27562
10:34:22,776 tpcw S 27562 select count(*) from shopping_cart_line where scl_sc_id = 424
10:34:22,778 tpcw S 27562 select i_related1 from item where i_id = 5759
10:34:22,779 tpcw S 27562 select scl_qty from shopping_cart_line where scl_sc_id = 424 and scl_i_id = 4903
10:34:22,781 tpcw W 27562 insert into shopping_cart_line (scl_sc_id, scl_qty, scl_i_id) values (424,1,4903)
10:34:22,782 tpcw W 27562 update shopping_cart set sc_time = now() where sc_id = 424
10:34:22,783 tpcw S 27562 select * from shopping_cart_line, item where scl_i_id = item.i_id and scl_sc_id = 4
10:34:22,787 tpcw C 27562
```

12.2. Replaying a trace file

The Request Player allows you to replay a trace file conforming to the format described in the previous section. The Request Player behavior is defined in a property file. The default property file is `requestplayer.properties` located in the `/c-jdbc/config` directory of your installation. The format of this file is described in Section 12.3.

The `bin` directory of the C-JDBC distribution contains the scripts to start the Request Player. Unix users must start the controller with **requestplayer.sh** whereas Windows users will use **requestplayer.bat**. These scripts accepts the following options:

- `-h` or `--help` displays a help message.
- `-f` or `--file` followed by the property file name. If this option is omitted, the default file is `requestplayer.properties` located in the `/c-jdbc/config` directory of your installation.
- `-v` or `--version` displays version information.

Table 1. List of acronyms used in this document

C-JDBC	Clustered Java DataBase Connectivity
CVS	Concurrent Versions System
INRIA	French National Institute for Research in Computer Science and Control
JDBC	Java DataBase Connectivity (not officially recognized as such)
JMX	Java Management eXtensions
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LGPL	GNU Lesser General Public License
RAIDb	Redundant Array of Inexpensive Databases
RDBMS	Relational DataBase Management System
RMI	Remote Method Invocation
SQL	Standard Query Language

12.3. requestplayer.properties

The Request Player properties file defines the following properties:

- `db_driver`: Database driver class (an example is `org.objectweb.cjdbc.driver.Driver`).
- `db_url`: Database JDBC URL (an example is `jdbc:cjdbc://localhost/test`).
- `db_username`: the login to use to connect to the database.
- `db_password`: the password to use to connect to the database.
- `trace_file`: the full path and name of the request trace file to replay (an example is `/tmp/request.log`).
- `nb_requests`: the number of requests to replay or 0 if the full trace file has to be played. Note that once the specified number of requests has been reached, all opened transactions are played by the player until they finish. Therefore, there might be more requests executed than the number specified in this property.
- `nb_clients`: number of emulated clients that will issue the requests in parallel (number of threads in the player).
- `timeout`: request timeout in seconds, a value of 0 meaning no timeout.
- `connection_type`: any value in `standard`, `fixed` or `pooling`. If `standard` is chosen, a new connection is used for each transaction or for each non transactionnal request to execute. If `fixed` is chosen, one connection is dedicated to each client thread for the whole run. If `pooling` is set, connection pooling is used.
- `poolsize`: size of the pool if `connection_type` has been set to `pooling`.

13. Glossary

Table 1 summarizes all the acronyms used in this document.

14. About C-JDBC

14.1. License

C-JDBC is free software. You can redistribute it and/or modify it under the terms of the GNU Lesser General Public License (http://www.gnu.org/copyleft/lesser.html) (LGPL) as published by the Free Software Foundation (http://www.fsf.org/); either version 2.1 of the License, or any later version.

C-JDBC is copyrighted by the French National Institute For Research In Computer Science And Control (http://www.inria.fr/) (INRIA) and Emic Networks.

14.2. Web Site

The C-JDBC project is hosted by ObjectWeb at the following URL: http://c-jdbc.objectweb.org/. To facilitate the development, a C-JDBC project has also be created on the ObjectWeb Forge (http://forge.objectweb.org/) facility (the ObjectWeb Consortium's own installation of GForge). The main project page can be found at: http://forge.objectweb.org/projects/c-jdbc/.

14.3. Wiki

There is also a Wiki for the C-JDBC project at the following URL: http://wiki.objectweb.org/c-jdbc/. You can share ideas and information there. You can also comment on the design of the code, bring new concepts and ideas ...

14.4. Mailing Lists

Two mailing lists are currently available for C-JDBC. Both lists are archived for public review at the C-JDBC's Web site (http://c-jdbc.objectweb.org/).

- `<c-jdbc@objectweb.org>` is the user mailing list. It is the source to get the latest information about C-JDBC, send your feedback and get support from the C-JDBC community.
- `<c-jdbc-commits@objectweb.org>` is a developper mailing list that reports every commit in the C-JDBC CVS repository.

Feedback is crucial to improve C-JDBC. Please send us your comments or any other form of input to: `<c-jdbc@objectweb.org>`.

14.5. Reporting a Bug

The ObjectWeb Forge C-JDBC project page (http://forge.objectweb.org/projects/c-jdbc/) provides support for bug tracking. We strongly encourage you to use the automatic Report feature (see Section 6.3.3) that provides all the details we usually need to figure out what happened. If you cannot use this feature, please include the following information when reporting a bug (when applicable):

- The C-JDBC driver and controller version.
- The XML file you used to configure the C-JDBC controller.

- JDK vendor and version (example: Sun JDK 1.3.1_06). If you use different JDK for driver and controller, please give as much detail as possible.
- OS vendor and version (examples: Linux 2.4.19 or Windows XP®). If you use different operating systems for clients, controllers and backends, give the appropriate information.
- Database backend version and driver (example: MySQL 4.0.8 Linux with mm.mysql driver 2.0.14).
- Detailed error description with possibly the exception stack trace or a logging trace with debugging enabled.

14.6. Getting Involved

C-JDBC is an open source project and welcomes external contributions. Please read the C-JDBC Developper's Guide and join us!

Basically, any feature you need but you do not find implemented in C-JDBC may become a contribution topic. Simply send your ideas, documents and developments (if any) to the `<c-jdbc@objectweb.org>` mailing list. You may also wish to get involved in an already undertaken work. The list of hot topics is available at the C-JDBC's ObjectWeb Forge site (http://forge.objectweb.org/projects/c-jdbc/). Please use also the ObjectWeb Forge tools for feature requests and bug reports/fixes.

You can finally subscribe to the c-jdbc-commits (http://www.objectweb.org/wws/admin/c-jdbc-commits) mailing list if you want to receive notifications of the CVS changes.

14.7. About INRIA

INRIA (http://www.inria.fr/) is the French National Institute for Research in Computer Science and Control. The Sardes project (http://sardes.inrialpes.fr/) at INRIA Rhones-Alpes has defined the RAIDb concept and leads the C-JDBC project developments.

14.8. About ObjectWeb

The goal of the ObjectWeb Consortium (http://www.objectweb.org/) is the development of open source distributed middleware, in the form of adaptable and flexible components. ObjectWeb components range from specific software frameworks and protocols to integrated platforms. More information on ObjectWeb and its projects is available at the ObjectWeb's Web site.

Notes

1. C-JDBC may work with older JVM version, but hasn't been tested.
2. CVS stands for *Concurrent Versions System* and is a popular version control system.
3. First In First Out.