

# Frequently Asked Questions

## Table of contents

1 Questions.....	2
1.1 1. Dependency FAQs.....	2
1.2 2. Configuring HA-JDBC.....	2
1.3 3. Using HA-JDBC.....	3
1.4 4. About HA-JDBC.....	6

## Questions

### 1. Dependency FAQs

#### 1.1. JGroups FAQ

<http://www.jgroups.org/javagroupsnew/docs/faq.html>

#### 1.2. SLF4J FAQ

<http://slf4j.org/faq.html>

#### 1.3. Quartz FAQ

<http://www.opensymphony.com/quartz/wikidocs/FAQ.html>

### 2. Configuring HA-JDBC

#### 2.1. How do I configure HA-JDBC to notify me when a database is deactivated?

Use your logging facility. HA-JDBC generates an ERROR level log message when it automatically deactivates a database. If you use Log4J, configure an SMTP appender for the "net.sf.hajdbc" logger.

If you use java.util.logging, configure an [SMTPHandler](#) for the "net.sf.hajdbc" package.

#### 2.2. I need to pass additional parameters to my JDBC driver. How can I specify these in my HA-JDBC configuration?

The database element may contain any number of property elements. HA-JDBC will pass these properties through to the Driver.connect(String url, Properties info) method of the underlying JDBC driver.

e.g.

```
<cluster id="..." balancer="..." default-sync="...">
  <database id="...">
    <driver>org.postgresql.Driver</driver>
    <url>jdbc:postgresql:database</url>
    <property name="ssl">true</property>
    <user>postgres</user>
    <password>XXXX</password>
  </database>
```

```
</cluster>
```

Alternatively, many JDBC drivers accept properties appended directly to the url.

e.g.

```
<cluster id="..." balancer="..." default-sync="...">
  <database id="...">
    <driver>org.postgresql.Driver</driver>
    <url>jdbc:postgresql:database?ssl=true</url>
    <user>postgres</user>
    <password>XXXX</password>
  </database>
</cluster>
```

### 2.3. How can I specify JNDI environment properties to my HA-JDBC DataSource configuration?

The datasource element may contain any number of property elements. HA-JDBC will pass these properties into the InitialContext(Hashtable env) constructor.

e.g.

```
<cluster id="..." balancer="..." default-sync="...">
  <datasource id="...">
    <name>jdbc/database</name>
    <property
name=" java.naming.factory.initial">org.jnp.interfaces.NamingContextFactory</property>
    <property
name=" java.naming.factory.url.pkgs">org.jboss.naming:org.jnp.interfaces</property>
  </datasource>
</cluster>
```

Alternatively, these properties may be specified in an `jndi.properties` application resource file. Details [here](#).

## 3. Using HA-JDBC

### 3.1. I get a `java.lang.OutOfMemoryError` when synchronizing a failed cluster node. Why?

This can occur if your database contains a table with more rows than can be fetched into memory at once. Both the differential and full synchronization strategies have a `fetchSize` property that control the number of rows that are fetched at a time.

e.g.

```
<sync id="diff"
class="net.sf.hajdbc.sync.DifferentialSynchronizationStrategy">
  <property name="fetchSize">1000</property>
</sync>
```

Each strategy also has a `maxBatchSize` property to control the number of statements to execute at a time.

e.g.

```
<sync id="full" class="net.sf.hajdbc.sync.FullSynchronizationStrategy">
  <property name="maxBatchSize">50</property>
</sync>
```

### 3.2. How can HA-JDBC be leveraged to improve database-driven HTTP Session failover?

Several session replication methods are described in an [article](#) posted to TheServerSide.com. Figure 7 illustrates the database persistence approach. When describing the disadvantages of this approach, the article fails to mention that the session database is a single point of failure in this design. HTTP sessions will survive the failure of an application server node, but failure of the session database spells doom for the application.

However, this can easily be remedied with redundant session databases using the HA-JDBC driver in distributable mode.

e.g.

For Tomcat, following the configuration instructions for setting up a Persistent Manager using a JDBC store [here](#). Simply use HA-JDBC's driver and url in place of your database's JDBC driver and url for the `driverName` and `connectionURL` properties of your store configuration.

### 3.3. Can I use HA-JDBC with Tomcat 5.0?

Yes, but first you will need to upgrade the JMX implementation used by Tomcat (found in `$CATALINA_HOME/bin/jmx.jar`). Tomcat 5.0 ships with MX4J 1.1.1 which only implements JMX 1.1. Because HA-JDBC requires JMX 1.2, you will need to upgrade this file to MX4J 2.0 or greater.

### 3.4. HA-JDBC remembers which databases were inactive even after I restart my JVM. Where is this state recorded and how do I clear it?

HA-JDBC uses the Java preferences API to persist the local database cluster state. The default storage mechanism varies depending on your operating system.

On Unix-like systems, the cluster state will be stored on the file system within the user's home directory:

```
~/ . java / . userPrefs / net / sf / ha jdbc / local / prefs . xml
```

On Windows systems, the cluster state will be stored in the registry:

```
HKEY_CURRENT_USER \ Software \ JavaSoft \ Prefs \ net \ sf \ ha jdbc \ local \ cluster-name
```

### **3.5. Why does my application need to specify a username and password when making database connections through HA-JDBC when the necessary authentication information is already specified in its configuration file?**

The username and password set in the configuration file are only used by HA-JDBC when it needs to connect to a database independently of your application. e.g. during synchronization, DatabaseCluster.isAlive() calls, etc. When your application connects to the database, the authentication information specified is passed through to the underlying driver.

The purpose of this authentication pass-through is to retain the flexibility your application had without HA-JDBC to use an appropriate database user (with an appropriate set of permissions) for a given task. Traditionally, applications will use a database user with select/insert/update/delete privileges only. Some application may not need to write information to a database and can use a database user with select privileges only. The database user that is used by HA-JDBC should be more of a root/superuser, with specific abilities to drop and create indexes and foreign keys (used during synchronization).

Since the same password will be passed through to each database, the user used by your application must exist and have the same password on each database in your cluster. The usernames/passwords used by HA-JDBC (as specified in the configuration file), however, do not need to be the same on each database.

### **3.6. Using version 1.4 or 1.5 of Sun's Java implementation, HA-JDBC mysteriously deadlocks the first time DriverManager.getConnection() is called. Why?**

In Java 1.5 and earlier, all of the methods on `java.sql.DriverManager` are synchronized. Your initial call to `DriverManager.getConnection(...)` triggers the startup of HA-JDBC. As HA-JDBC initializes, it will eventually call `DriverManager.getDriver(...)` for each database url in your cluster. The calls to `DriverManager.getDriver(...)` occur in a different thread from your application thread, hence the deadlock.

There are at least 2 known workarounds for this problem:

1. Upgrade to Java 1.6. After forever insisting that the blanket synchronization of

- `java.sql.DriverManager` was [not a defect](#), Sun *silently* fixed this in Java 1.6.
- Rather than use `DriverManager.getConnection(...)` to obtain connections, use `DriverManager.getDriver(...).connect(...)` instead. This will circumvent the deadlock since `DriverManager.getDriver(...)` will not trigger HA-JDBC startup and `Driver.connect(...)` is not synchronized.

## 4. About HA-JDBC

### 4.1. How does HA-JDBC compare to Sequoia?

Both HA-JDBC and Sequoia attempt to solve the same problem (i.e. eliminating the database as a single point of failure), but have different approaches.

	HA-JDBC	Sequoia
Architecture	<ol style="list-style-type: none"> <li>HA-JDBC driver delegates JDBC methods directly to the underlying JDBC drivers.</li> <li>Cluster details are stored in distributed cache on client.</li> <li>Leverages underlying database for request scheduling.</li> <li>High-availability is inherent in symmetric design.</li> </ol>	<ol style="list-style-type: none"> <li>Sequoia's JDBC driver delegates query execution to a remote controller process. Controller then delegates queries to the underlying JDBC driver.</li> <li>Cluster details are known only to controller.</li> <li>The controller's request scheduler strategy is responsible for determining execution order.</li> <li>Controller introduces new single point-of-failure. Workaround is to set up a failover controller process. Details <a href="#">here</a>.</li> </ol>
Cluster topography	<ol style="list-style-type: none"> <li>Databases within a cluster must be homogenous.</li> <li>Supports "mirroring" only.</li> </ol>	<ol style="list-style-type: none"> <li>Supports heterogenous database clusters - requires that SQL queries be translated to appropriate SQL dialect.</li> <li>Supports RAIDb-0, RAIDb-1, and RAIDb-2 configurations (i.e. mirroring (replication), striping (partitioning), and partial mirroring/striping, respectively).</li> </ol>
Failed node recovery	<ol style="list-style-type: none"> <li>No recovery log is maintained.</li> <li>Synchronization achieved through various brute force</li> </ol>	<ol style="list-style-type: none"> <li>Controller uses internal database recovery log to restore state of a reactivated database.</li> </ol>

Performance  Cluster administration  Provides connection pooling  ResultSet caching ability  JDBC 2.0 feature support	<p>strategies.</p> <ol style="list-style-type: none"> <li>Inefficient hot synchronization capability tolerated at the savings of performance during normal usage.</li> <li>Includes the ability to automatically reactivate failed database nodes according to a schedule.</li> </ol>	<ol style="list-style-type: none"> <li>Synchronization is done by executing SQL statements since last checkpoint.</li> <li>Efficient hot synchronization capability achieved at the cost of recovery log overhead.</li> <li>Failed database nodes can only be re-activated manually.</li> </ol>
	<p>Faster reads under load, slightly slower writes than standard JDBC. Details <a href="#">here</a>.</p>	<p>Qualitatively slower than HA-JDBC since each request requires an additional network hop (i.e. driver to controller, controller to database). Details <a href="#">here</a>.</p>
	<p>Leverages JConsole from Sun's JDK 1.5+ or any other 3rd party JMX console for cluster administration.</p>	<p>Provides custom JMX-based command-line administration console.</p>
	<p>No - many open source solutions already exist and can be used in conjunction with HA-JDBC:</p> <ul style="list-style-type: none"> <li><a href="#">c3p0</a></li> <li><a href="#">Proxool</a></li> <li><a href="#">Commons DBCP</a></li> <li><a href="#">XAPool</a></li> <li><a href="#">Primrose</a></li> </ul>	<p>Yes - implemented in controller</p>
	<p>No - Transparent result set caching is available through <a href="#">IronEye Cache</a>.</p>	<p>Yes - implemented in controller</p>
	<p>Full support</p>	<p>Lacks support for:</p> <ul style="list-style-type: none"> <li>Database-compiled PreparedStatements</li> <li>CallableStatements with OUT parameters</li> <li>True large object support - Blob and Clob are simulated with encoded byte[] and String, respectively</li> <li>True binary/character stream</li> </ul>

JDBC 3.0 support		<p>support - simulated with encoded byte[] and String, respectively</p> <ul style="list-style-type: none"> <li>• Array and Ref types</li> <li>• Custom type mapping</li> <li>• Block fetched scrollable ResultSets</li> <li>• Statement execution cancellation</li> </ul>
	Full support	<p>Lacks support for:</p> <ul style="list-style-type: none"> <li>• Transactional Savepoints</li> <li>• XADataSource and XAConnection</li> <li>• PreparedStatement pooling</li> <li>• Retrieval of auto-generated keys</li> <li>• ParameterMetaData</li> <li>• ResultSet holdability support</li> <li>• Queries/Store procedures that return multiple ResultSets</li> <li>• Updatable Blobs and Clobs</li> </ul>
JDBC 4.0 support	Full support	None

**Table 1: Feature comparison**