

# Database (SQL) Realm

{scrollbar}

In this section we will focus on the use a database for verifying and retrieving user names and passwords.

For this example we created a new database called **SecurityDatabase** using the built-in Derby database. The following steps summarize the procedure performed to create the database and tables, load some sample data and create the connection pool. Detailed instructions on how to define database connection pools are described in the [Configuring database pools](#) section.

## Create database and load sample data

- In the **Console Navigation** menu on the left click on **Database Manager**.
- Enter **SecurityDatabase** in the **Create DB:** field and click **Create**.
- Select the **SecurityDatabase** database from the **Use DB:** pull-down menu, enter the following commands and click **Run SQL**.

```
create table users
(username varchar(15),
password varchar(15));
create table groups
(username varchar(15),
groupname varchar(15));
insert into users values('userone','p1');
insert into users values('usertwo','p2');
insert into users values('userthree','p3');
insert into groups values('userone','admin');
insert into groups values('usertwo','admin');
insert into groups values('userthree','user');
```

## Create connection pool

- In the **Console Navigation** menu on the left click on **Database Pools**.
- Click on **Using the Geronimo database pool wizard**.
- Enter **SecurityDatabase** as the database pool name.
- Select **Derby embedded** from the database pool type pull-down menu and click **Next**.
- Verify the JDBC driver class is **org.apache.derby.jdbc.EmbeddedDriver**.
- From the Driver Jar pull-down menu select **org.apache.derby/derby/10.1.1.0/jar**.
- Leave **blank** the connection user name and password.
- Enter **SecurityDatabase** as the database name and click **Next**.
- Click **Test Connection**.
- Click **Deploy**.

## Add a new security realm

To create a new security realm click on **Add new security realm** from the **Security Realms** portlet.

**Security Realms** [view]

Create Security Realm -- Step 1: Select Name and Type

Name of Security Realm:

A name that is different than the name for any other security realms in the server (no spaces in the name please). Other components will use this name to refer to the security realm.

Realm Type:

The type of login module used as the master for this security realm. Select "Other" for manual configuration options including custom login modules and realms that use multiple login modules to populate user principals.

[Cancel](#)

Enter **derby\_security\_realm** in the **Name of Security Realm:** field and select **Database (SQL) Realm** from the **Realm type:** pull-down menu and click **Next**.

The following screen configures the login module. The first two field you need to fill will potentially vary from one database type to another. In this case we are using the embedded Derby database so the User and Group select SQL should read as follows:

**User SELECT SQL:** select username, password from **APP**.users where username=?

**Group SELECT SQL:** select username, groupname from **APP**.groups where username=?

Note that **APP** is the default schema for the embedded Derby database and it needs to precede the table in the SQL statement. These statements are likely to be different from one database to another, for instance this procedure was also tested with DB2, the SQL statements used were:

**User SELECT SQL:** select username, password from users where username=?  
**Group SELECT SQL:** select username, groupname from groups where username=?

Once you entered the SQL statements for retrieving users and groups you need to select from the **Database Pool** pull-down menu the database connection pool you created in the previous step. Add the required values as shown below and click **Next**.

**Database Pool:** SecurityDatabase

**JDBC Driver Class:** org.apache.derby.jdbc.EmbeddedDriver

**Driver JAR:** org.apache.derby/derby/10.1.1.0/jar

**JDBC URL:** jdbc:derby:SecurityDatabase

**Security Realms** [view]

Create Security Realm -- Step 2: Configure Login Module

User  
SELECT SQL:

select username, password from APP.users where username=?

A SQL statement to load user/password information. It should return 2 columns, the first holding a username and the second holding a password. The statement may use the PreparedStatement syntax of ? for a parameter, in which case the username will be set for every parameter. A typical setting would be SELECT username, password FROM app\_users WHERE username=?

Group  
SELECT SQL:

select username, groupname from APP.groups where username=?

A SQL statement to load group information for a user. It should return 2 columns, the first holding a username and the second holding a group name. The statement may use the PreparedStatement syntax of ? for a parameter, in which case the username will be set for every parameter. A typical setting would be SELECT username, group\_name FROM user\_groups WHERE username=? or for a more normalized schema, SELECT u.username, g.name FROM app\_users u, groups g, user\_groups ug WHERE ug.user\_id=users.id AND ug.group\_id=g.id AND u.username=?  
*A SQL security realm must either have a database pool or JDBC connectivity settings to connect to the database. Please select EITHER the database pool, OR the rest of the JDBC settings.*

Database Pool

SecurityDatabase

A database pool that the login module will use to connect to the database. If this is specified, none of the rest of the settings after this are necessary.

JDBC Driver Class

org.apache.derby.jdbc.EmbeddedDriver

The fully-qualified JDBC driver class name. This driver must be located in the JAR specified in the next field.

Driver JAR:

org.apache.derby/derby/10.1.1.0/jar

The JAR holding the selected JDBC driver. Should be installed under Geronimo/repository/ to appear in this list.

JDBC URL

jdbc:derby:SecurityDatabase

The JDBC URL that specifies the details of the database to connect to. This has a different form for each JDBC driver.

JDBC Username

The username used to connect to the database

JDBC Password

The password used to connect to the database

Next

[Cancel](#)

The following step will allow you to enable auditing for monitoring the login attempts via this realm. In this step you can also configure the account lockout based on the number of failed logging attempts withing a specified timeframe. The last option in this step, **Store Password**, when enabled will allow the realm to store the user password in a private credential in the subject.

Security Realms

[view]

Create Security Realm -- Step 3: Advanced Configuration

Enable Auditing:

☒

Log File:

var/log/derby\_security\_realm.log

If enabled, every login attempt will be recorded to the specified file. The path should be relative to the Geronimo home directory (a typical value would be var/log/login-attempts.log).

Enable Lockout:

☒

Lock a user after

3

failures within

10

seconds

and keep the account locked for

60

seconds.

If enabled, a certain number of failed logins in a particular time frame will cause a user's account to be locked for a certain period of time. This is a defense against brute force account cracking attacks.

Store Password:

☐

If enabled, the realm will store each user's password in a private credential in the Subject. This will allow access to the password later after the login process has completed. This is not normally required.

Test a Login

Skip Test and Deploy

Skip Test and Show Plan

[Cancel](#)

At this point you have configured this new security realm, the next step is to test it and then deploy it. Click on **Test a Login**.

Enter a valid user name and password to be retrieved from the database and click **Next**.

Security Realms

[view]

Create Security Realm -- Step 4: Test Login

From here you can enter a username and password for the main login module in the realm, and see if the login is successful and which Principals are generated for the user. This is meant to be an indication of whether the settings for the main login module are correct. It does not invoke advanced features such as auditing or lockout.

Username:

userone

The username to use to log in to the realm.

Password:

jack

The password to use to log in to the realm.

Next

[Cancel](#)

You should receive a confirmation message that the login succeeded, click on **Deploy Realm** to load this configuration to the server.

Security Realms

[view]

Create Security Realm -- Step 5: Login Results

Test Results:

Login succeeded with 2 principals

Principals:

userone org.apache.geronimo.security.realm.providers.GeronimoUserPrincipal

admin org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal

Test Again

Edit Realm

Show Plan

Deploy Realm

[Cancel](#)

Now you have a new, fully configured, security realm that retrieves user names and passwords from the built in Derby database.

The following example shows the deployment plan for this security realm. As an alternative to the Geronimo Administration Console, you can save this example to a file (i.e. derby\_security\_realm.xml) and deploy it with the [Deployer tool](#) by running the following command:

```
<geronimo_home>\bin\deploy --user system --password manager deploy <realm_path>\derby_security_realm.xml
```

```
xmlsolidderby_security_realm <module xmlns="http://geronimo.apache.org/xml/ns/deployment-1.1"> <environment> <moduleId> <groupId>console</groupId> <artifactId>realm-derby_security_realm</artifactId> <version>1.0</version> <type>car</type> </moduleId> <dependencies> <dependency> <groupId>geronimo</groupId> <artifactId>j2ee-security</artifactId> <type>car</type> </dependency> <dependency> <groupId>org.apache.derby</groupId> <artifactId>derby</artifactId> <version>10.1.1.0</version> <type>jar</type> </dependency> </dependencies> </environment> <gbean name="derby_security_realm" class="org.apache.geronimo.security.realm.GenericSecurityRealm"> <attribute name="realmName">derby_security_realm</attribute> <reference name="ServerInfo"> <name>ServerInfo</name> </reference> <reference name="LoginService"> <name>JaasLoginService</name> </reference> <xml-reference name="LoginModuleConfiguration"> <log:login-config xmlns:log="http://geronimo.apache.org/xml/ns/loginconfig-1.1"> <log:login-module control-flag="REQUIRED" server-side="true" wrap-principals="false"> <log:login-domain-name>derby_security_realm</log:login-domain-
```

```
name> <log:login-module-class>org.apache.geronimo.security.realm.providers.SQLLoginModule</log:login-module-class> <log:option name="userSelect"
>select username, password from APP.users where username=?</log:option> <log:option name="jdbcDriver">org.apache.derby.jdbc.EmbeddedDriver<
/!log:option> <log:option name="groupSelect">select username, groupname from APP.groups where username=?</log:option> <log:option name="
jdbcURL">jdbc:derby:SecurityDatabase</log:option> </log:login-module> <log:login-module control-flag="OPTIONAL" server-side="true" wrap-principals="
false"> <log:login-domain-name>derby_security_realm-Audit</log:login-domain-name> <log:login-module-class>org.apache.geronimo.security.realm.
providers.FileAuditLoginModule</log:login-module-class> <log:option name="file">var/log/derby_security_realm.log</log:option> </log:login-module> <log:
login-module control-flag="REQUISITE" server-side="true" wrap-principals="false"> <log:login-domain-name>derby_security_realm-Lockout</log:login-
domain-name> <log:login-module-class>org.apache.geronimo.security.realm.providers.RepeatedFailureLockoutLoginModule</log:login-module-class>
<log:option name="lockoutDurationSecs">60</log:option> <log:option name="failurePeriodSecs">10</log:option> <log:option name="failureCount">3</log:
option> </log:login-module> </log:login-config> </xml-reference> </gbean> </module>
```