

<p>Download derby and unzip to your hard drive</p>	<p>http://db.apache.org/derby/derby_downloads.html</p> <p>1) I downloaded the 10.4.2.0 version 2) Opened the file in AIZip and unzipped it to the c:\ directory 3) renamed the directory where it unzipped to c:\derby</p> <pre>C:\derby>dir Volume in drive C is C_Drive Volume Serial Number is D2F3-B15C Directory of C:\derby 11/06/2008 02:01 AM <DIR> . 11/06/2008 02:01 AM <DIR> .. 11/06/2008 12:50 AM <DIR> bin 11/06/2008 12:50 AM <DIR> demo 11/06/2008 02:06 AM 495 derby.log 11/06/2008 12:50 AM <DIR> docs 03/10/2008 02:11 PM 5,513 index.html 11/06/2008 12:50 AM <DIR> javadoc 08/06/2008 12:55 PM 33,734 KEYS 11/06/2008 12:50 AM <DIR> lib 08/06/2008 12:55 PM 11,560 LICENSE 11/06/2008 02:06 AM <DIR> MyDbTest 08/06/2008 12:55 PM 1,414 NOTICE 08/26/2008 06:30 AM 24,215 RELEASE-NOTES.html 11/06/2008 01:54 AM 390 setDerby.bat 11/06/2008 01:55 AM 211 setDerby.txt 11/06/2008 12:50 AM <DIR> test 8 File(s) 77,532 bytes 9 Dir(s) 40,893,104,128 bytes free C:\derby></pre>
<p>Configure embedded derby on your system</p>	<pre>C:\derby>dir Volume in drive C is C_Drive Volume Serial Number is D2F3-B15C Directory of C:\derby 11/06/2008 02:01 AM <DIR> . 11/06/2008 02:01 AM <DIR> .. 11/06/2008 12:50 AM <DIR> bin 11/06/2008 12:50 AM <DIR> demo 11/06/2008 02:06 AM 495 derby.log 11/06/2008 12:50 AM <DIR> docs 03/10/2008 02:11 PM 5,513 index.html 11/06/2008 12:50 AM <DIR> javadoc 08/06/2008 12:55 PM 33,734 KEYS 11/06/2008 12:50 AM <DIR> lib 08/06/2008 12:55 PM 11,560 LICENSE 11/06/2008 02:06 AM <DIR> MyDbTest 08/06/2008 12:55 PM 1,414 NOTICE 08/26/2008 06:30 AM 24,215 RELEASE-NOTES.html 11/06/2008 01:54 AM 390 setDerby.bat 11/06/2008 01:55 AM 211 setDerby.txt 11/06/2008 12:50 AM <DIR> test 8 File(s) 77,532 bytes 9 Dir(s) 40,893,104,128 bytes free C:\derby>set DERBY_INSTALL=C:\derby C:\derby>set CLASSPATH=%CLASSPATH%;%DERBY_INSTALL%\lib\derby.jar;%DERBY_INSTALL%\lib\derbytools.jar;. C:\derby>cd bin This bat file completes the same thing you just did manually above C:\derby\bin>setEmbeddedCP.bat</pre>

	<pre> C:\derby\bin>SET DERBY_HOME=C:\derby C:\derby\bin>set CLASSPATH=C:\derby\lib\derby.jar;C:\derby\lib\derbytools.jar; C:\Program Files\IBM\Java50\jre\lib\ext\QTJava.zip;C:\Java;C:\derby\lib\derby.j r;C:\derby\lib\derbytools.jar; This command verifies the install of derby the output will look similar to what I have below but not exact C:\derby\bin>java org.apache.derby.tools.sysinfo ----- Java Information ----- Java Version: 1.6.0_07 Java Vendor: Sun Microsystems Inc. Java home: C:\Program Files\Java\jre1.6.0_07 Java classpath: C:\derby\lib\derby.jar;C:\derby\lib\derbytools.jar;.;C:\Progra Files\IBM\Java50\jre\lib\ext\QTJava.zip;C:\Java;C:\derby\lib\derby.jar;C:\derb \lib\derbytools.jar; OS name: Windows XP OS architecture: x86 OS version: 5.1 Java user name: meveret Java user home: C:\Documents and Settings\Administrator Java user dir: C:\derby\bin java.specification.name: Java Platform API Specification java.specification.version: 1.6 ----- Derby Information ----- JRE - JDBC: Java SE 6 - JDBC 4.0 [C:\derby\lib\derby.jar] 10.4.2.0 - (689064) [C:\derby\lib\derbytools.jar] 10.4.2.0 - (689064) ----- ----- Locale Information ----- Current Locale : [English/United States [en_US]] Found support for locale: [cs] version: 10.4.2.0 - (689064) Found support for locale: [de_DE] version: 10.4.2.0 - (689064) Found support for locale: [es] version: 10.4.2.0 - (689064) Found support for locale: [fr] version: 10.4.2.0 - (689064) Found support for locale: [hu] version: 10.4.2.0 - (689064) Found support for locale: [it] version: 10.4.2.0 - (689064) Found support for locale: [ja_JP] version: 10.4.2.0 - (689064) Found support for locale: [ko_KR] version: 10.4.2.0 - (689064) Found support for locale: [pl] version: 10.4.2.0 - (689064) Found support for locale: [pt_BR] version: 10.4.2.0 - (689064) Found support for locale: [ru] version: 10.4.2.0 - (689064) Found support for locale: [zh_CN] version: 10.4.2.0 - (689064) Found support for locale: [zh_TW] version: 10.4.2.0 - (689064) ----- </pre>
<p>If you get the exception in this row of the table go back a step and setup your environment again, your config is not correct.</p>	<pre> C:\>cd derby C:\derby>dir KEYS RELEASE-NOTES.html docs setDerby.bat LICENSE bin index.html setDerby.txt MyDbTest demo javadoc test NOTICE derby.log lib C:\derby>cd demo C:\derby\demo>cd simple The system cannot find the path specified. </pre>

	<pre> C:\derby\demo>dir README.txt databases programs templates C:\derby\demo>cd programs C:\derby\demo\programs>ls csfull.css nserverdemo toursdb demo.html readme.html vtis derbylogo128_bluebg.png scores workingwithderby localcal simple navbar.html simplemobile C:\derby\demo\programs>cd simple If there is no class file compile SimpleApp.java C:\derby\demo\programs\simple>ls SimpleApp.class SimpleApp.java derby.properties example.html C:\derby\demo\programs\simple>javac SimpleApp.java Note: SimpleApp.java uses unchecked or unsafe operations. Note: Recompile with -Xlint:unchecked for details. C:\derby\demo\programs\simple>java SimpleApp SimpleApp starting in embedded mode Unable to load the JDBC driver org.apache.derby.jdbc.EmbeddedDriver Please check your CLASSPATH. java.lang.ClassNotFoundException: org.apache.derby.jdbc.EmbeddedDriver at java.net.URLClassLoader\$1.run(Unknown Source) at java.security.AccessController.doPrivileged(Native Method) at java.net.URLClassLoader.findClass(Unknown Source) at java.lang.ClassLoader.loadClass(Unknown Source) at sun.misc.Launcher\$AppClassLoader.loadClass(Unknown Source) at java.lang.ClassLoader.loadClass(Unknown Source) at java.lang.ClassLoader.loadClassInternal(Unknown Source) at java.lang.Class.forName0(Native Method) at java.lang.Class.forName(Unknown Source) at SimpleApp.loadDriver(SimpleApp.java:417) at SimpleApp.go(SimpleApp.java:128) at SimpleApp.main(SimpleApp.java:96) ----- SQLException ----- SQL State: 08001 Error Code: 0 Message: No suitable driver found for jdbc:derby:derbyDB;create=true SimpleApp finished C:\derby\demo\programs\simple>cd .. C:\derby\demo\programs>cd .. C:\derby\demo>cd .. </pre>
<p>Working execution of the sample application</p>	<pre> C:\derby\bin>cd .. C:\derby>ls KEYS RELEASE-NOTES.html docs setDerby.bat LICENSE bin index.html setDerby.txt MyDbTest demo javadoc test NOTICE derby.log lib C:\derby>cd demo C:\derby\demo>ls README.txt databases programs templates C:\derby\demo>cd programs C:\derby\demo\programs>cd simple </pre>

	<pre>C:\derby\demo\programs\simple>ls SimpleApp.class SimpleApp.java derby.properties example.html C:\derby\demo\programs\simple>java SimpleApp SimpleApp starting in embedded mode Loaded the appropriate driver Connected to and created database derbyDB Created table location Inserted 1956 Webster Inserted 1910 Union Updated 1956 Webster to 180 Grand Updated 180 Grand to 300 Lakeshore Verified the rows Dropped table location Committed the transaction Derby shut down normally SimpleApp finished</pre>
Java source	<pre>/* Derby - Class SimpleApp Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to You under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. */ import java.sql.Connection; import java.sql.DriverManager; import java.sql.PreparedStatement; import java.sql.ResultSet; import java.sql.SQLException; import java.sql.Statement; import java.util.ArrayList; import java.util.Properties; /** * <p> * This sample program is a minimal Java application showing JDBC access to a * Derby database.</p> * <p> * Instructions for how to run this program are * given in example.html, by default located in the * same directory as this source file (\$DERBY_HOME/demo/programs/simple/).</p> * <p> * Derby applications can run against Derby running in an embedded * or a client/server framework.</p> * <p> * When Derby runs in an embedded framework, the JDBC application and Derby * run in the same Java Virtual Machine (JVM). The application * starts up the Derby engine.</p> * <p> * When Derby runs in a client/server framework, the application runs in a * different JVM from Derby. The application only needs to load the client * driver, and the connectivity framework (in this case the Derby Network * Server) provides network connections.</p> */ public class SimpleApp { /* the default framework is embedded*/ private String framework = "embedded";</pre>

```

private String driver = "org.apache.derby.jdbc.EmbeddedDriver";
private String protocol = "jdbc:derby:";

/**
 * <p>
 * Starts the demo by creating a new instance of this class and running
 * the <code>go()</code> method.</p>
 * <p>
 * When you run this application, you may give one of the following
 * arguments:
 * <ul>
 * <li><code>embedded</code> - default, if none specified. Will use
 * Derby's embedded driver. This driver is included in the derby.jar
 * file.</li>
 * <li><code>derbyclient</code> - will use the Derby client driver to
 * access the Derby Network Server. This driver is included in the
 * derbyclient.jar file.</li>
 * <li><code>jccjdbcclient</code> - will use the DB2 Universal JDBC
 * network client driver, also known as JCC, to access the Network
 * Server. This driver is not part of the Derby distribution.</li>
 * </ul>
 * <p>
 * When you are using a client/server framework, the network server must
 * already be running when trying to obtain client connections to Derby.
 * This demo program will try to connect to a network server on this
 * host (the localhost), see the <code>protocol</code> instance variable.
 * </p>
 * <p>
 * When running this demo, you must include the correct driver in the
 * classpath of the JVM. See <a href="example.html">example.html</a> for
 * details.
 * </p>
 * @param args This program accepts one optional argument specifying which
 * connection framework (JDBC driver) to use (see above). The default
 * is to use the embedded JDBC driver.
 */
public static void main(String[] args)
{
    new SimpleApp().go(args);
    System.out.println("SimpleApp finished");
}

/**
 * <p>
 * Starts the actual demo activities. This includes loading the correct
 * JDBC driver, creating a database by making a connection to Derby,
 * creating a table in the database, and inserting, updating and retrieving
 * some data. Some of the retrieved data is then verified (compared) against
 * the expected results. Finally, the table is deleted and, if the embedded
 * framework is used, the database is shut down.</p>
 * <p>
 * Generally, when using a client/server framework, other clients may be
 * (or want to be) connected to the database, so you should be careful about
 * doing shutdown unless you know that no one else needs to access the
 * database until it is rebooted. That is why this demo will not shut down
 * the database unless it is running Derby embedded.</p>
 *
 * @param args - Optional argument specifying which framework or JDBC driver
 * to use to connect to Derby. Default is the embedded framework,
 * see the <code>main()</code> method for details.
 * @see #main(String[])
 */
void go(String[] args)
{
    /* parse the arguments to determine which framework is desired*/
    parseArguments(args);

    System.out.println("SimpleApp starting in " + framework + " mode");

    /* load the desired JDBC driver */

```

```

loadDriver();

/* We will be using Statement and PreparedStatement objects for
 * executing SQL. These objects, as well as Connections and ResultSets,
 * are resources that should be released explicitly after use, hence
 * the try-catch-finally pattern used below.
 * We are storing the Statement and Prepared statement object references
 * in an array list for convenience.
 */
Connection conn = null;
    /* This ArrayList usage may cause a warning when compiling this class
     * with a compiler for J2SE 5.0 or newer. We are not using generics
     * because we want the source to support J2SE 1.4.2 environments. */
ArrayList statements = new ArrayList(); // list of Statements, PreparedStatements
PreparedStatement psInsert = null;
PreparedStatement psUpdate = null;
Statement s = null;
ResultSet rs = null;
try
{
    Properties props = new Properties(); // connection properties
    // providing a user name and password is optional in the embedded
    // and derbyclient frameworks
    props.put("user", "user1");
    props.put("password", "user1");

    /* By default, the schema APP will be used when no username is
     * provided.
     * Otherwise, the schema name is the same as the user name (in this
     * case "user1" or USER1.)
     *
     * Note that user authentication is off by default, meaning that any
     * user can connect to your database using any password. To enable
     * authentication, see the Derby Developer's Guide.
     */

    String dbName = "derbyDB"; // the name of the database

    /*
     * This connection specifies create=true in the connection URL to
     * cause the database to be created when connecting for the first
     * time. To remove the database, remove the directory derbyDB (the
     * same as the database name) and its contents.
     *
     * The directory derbyDB will be created under the directory that
     * the system property derby.system.home points to, or the current
     * directory (user.dir) if derby.system.home is not set.
     */
    conn = DriverManager.getConnection(protocol + dbName
        + ";create=true", props);

    System.out.println("Connected to and created database " + dbName);

    // We want to control transactions manually. Autocommit is on by
    // default in JDBC.
    conn.setAutoCommit(false);

    /* Creating a statement object that we can use for running various
     * SQL statements commands against the database.*/
    s = conn.createStatement();
    statements.add(s);

    // We create a table...
    s.execute("create table location(num int, addr varchar(40))");
    System.out.println("Created table location");

    // and add a few rows...

    /* It is recommended to use PreparedStatements when you are
     * repeating execution of an SQL statement. PreparedStatements also

```

```

* allows you to parameterize variables. By using PreparedStatements
* you may increase performance (because the Derby engine does not
* have to recompile the SQL statement each time it is executed) and
* improve security (because of Java type checking).
*/
// parameter 1 is num (int), parameter 2 is addr (varchar)
psInsert = conn.prepareStatement(
    "insert into location values (?, ?)");
statements.add(psInsert);

psInsert.setInt(1, 1956);
psInsert.setString(2, "Webster St.");
psInsert.executeUpdate();
System.out.println("Inserted 1956 Webster");

psInsert.setInt(1, 1910);
psInsert.setString(2, "Union St.");
psInsert.executeUpdate();
System.out.println("Inserted 1910 Union");

// Let's update some rows as well...

// parameter 1 and 3 are num (int), parameter 2 is addr (varchar)
psUpdate = conn.prepareStatement(
    "update location set num=?, addr=? where num=?");
statements.add(psUpdate);

psUpdate.setInt(1, 180);
psUpdate.setString(2, "Grand Ave.");
psUpdate.setInt(3, 1956);
psUpdate.executeUpdate();
System.out.println("Updated 1956 Webster to 180 Grand");

psUpdate.setInt(1, 300);
psUpdate.setString(2, "Lakeshore Ave.");
psUpdate.setInt(3, 180);
psUpdate.executeUpdate();
System.out.println("Updated 180 Grand to 300 Lakeshore");

/*
   We select the rows and verify the results.
*/
rs = s.executeQuery(
    "SELECT num, addr FROM location ORDER BY num");

/* we expect the first returned column to be an integer (num),
 * and second to be a String (addr). Rows are sorted by street
 * number (num).
 *
 * Normally, it is best to use a pattern of
 * while(rs.next()) {
 *     // do something with the result set
 * }
 * to process all returned rows, but we are only expecting two rows
 * this time, and want the verification code to be easy to
 * comprehend, so we use a different pattern.
 */

int number; // street number retrieved from the database
boolean failure = false;
if (!rs.next())
{
    failure = true;
    reportFailure("No rows in ResultSet");
}

if ((number = rs.getInt(1)) != 300)
{
    failure = true;

```

```

reportFailure(
    "Wrong row returned, expected num=300, got " + number);
}

if (!rs.next())
{
    failure = true;
    reportFailure("Too few rows");
}

if ((number = rs.getInt(1)) != 1910)
{
    failure = true;
    reportFailure(
        "Wrong row returned, expected num=1910, got " + number);
}

if (rs.next())
{
    failure = true;
    reportFailure("Too many rows");
}

if (!failure) {
    System.out.println("Verified the rows");
}

// delete the table
s.execute("drop table location");
System.out.println("Dropped table location");

/*
 * We commit the transaction. Any changes will be persisted to
 * the database now.
 */
conn.commit();
System.out.println("Committed the transaction");

/*
 * In embedded mode, an application should shut down the database.
 * If the application fails to shut down the database,
 * Derby will not perform a checkpoint when the JVM shuts down.
 * This means that it will take longer to boot (connect to) the
 * database the next time, because Derby needs to perform a recovery
 * operation.
 *
 * It is also possible to shut down the Derby system/engine, which
 * automatically shuts down all booted databases.
 *
 * Explicitly shutting down the database or the Derby engine with
 * the connection URL is preferred. This style of shutdown will
 * always throw an SQLException.
 *
 * Not shutting down when in a client environment, see method
 * Javadoc.
 */

if (framework.equals("embedded"))
{
    try
    {
        // the shutdown=true attribute shuts down Derby
        DriverManager.getConnection("jdbc:derby::shutdown=true");

        // To shut down a specific database only, but keep the
        // engine running (for example for connecting to other
        // databases), specify a database in the connection URL:
        //DriverManager.getConnection("jdbc:derby:" + dbName + ";shutdown=true");
    }
    catch (SQLException se)

```

```

    {
        if (( (se.getErrorCode() == 50000)
            && ("XJ015".equals(se.getSQLState())) )) {
            // we got the expected exception
            System.out.println("Derby shut down normally");
            // Note that for single database shutdown, the expected
            // SQL state is "08006", and the error code is 45000.
        } else {
            // if the error code or SQLState is different, we have
            // an unexpected exception (shutdown failed)
            System.err.println("Derby did not shut down normally");
            printSQLException(se);
        }
    }
}
}
}
catch (SQLException sqle)
{
    printSQLException(sqle);
} finally {
    // release all open resources to avoid unnecessary memory usage

    // ResultSet
    try {
        if (rs != null) {
            rs.close();
            rs = null;
        }
    } catch (SQLException sqle) {
        printSQLException(sqle);
    }

    // Statements and PreparedStatements
    int i = 0;
    while (!statements.isEmpty()) {
        // PreparedStatement extend Statement
        Statement st = (Statement)statements.remove(i);
        try {
            if (st != null) {
                st.close();
                st = null;
            }
        } catch (SQLException sqle) {
            printSQLException(sqle);
        }
    }

    //Connection
    try {
        if (conn != null) {
            conn.close();
            conn = null;
        }
    } catch (SQLException sqle) {
        printSQLException(sqle);
    }
}
}

/**
 * Loads the appropriate JDBC driver for this environment/framework. For
 * example, if we are in an embedded environment, we load Derby's
 * embedded Driver, <code>org.apache.derby.jdbc.EmbeddedDriver</code>.
 */
private void loadDriver() {
    /**
     * The JDBC driver is loaded by loading its class.
     * If you are using JDBC 4.0 (Java SE 6) or newer, JDBC drivers may
     * be automatically loaded, making this code optional.
     */
}

```

```

* In an embedded environment, this will also start up the Derby
* engine (though not any databases), since it is not already
* running. In a client environment, the Derby engine is being run
* by the network server framework.
*
* In an embedded environment, any static Derby system properties
* must be set before loading the driver to take effect.
*/
try {
    Class.forName(driver).newInstance();
    System.out.println("Loaded the appropriate driver");
} catch (ClassNotFoundException cnfe) {
    System.err.println("\nUnable to load the JDBC driver " + driver);
    System.err.println("Please check your CLASSPATH.");
    cnfe.printStackTrace(System.err);
} catch (InstantiationException ie) {
    System.err.println(
        "\nUnable to instantiate the JDBC driver " + driver);
    ie.printStackTrace(System.err);
} catch (IllegalAccessException iae) {
    System.err.println(
        "\nNot allowed to access the JDBC driver " + driver);
    iae.printStackTrace(System.err);
}
}

/**
 * Reports a data verification failure to System.err with the given message.
 *
 * @param message A message describing what failed.
 */
private void reportFailure(String message) {
    System.err.println("\nData verification failed:");
    System.err.println("\t" + message);
}

/**
 * Prints details of an SQLException chain to <code>System.err</code>.
 * Details included are SQL State, Error code, Exception message.
 *
 * @param e the SQLException from which to print details.
 */
public static void printSQLException(SQLException e)
{
    // Unwraps the entire exception chain to unveil the real cause of the
    // Exception.
    while (e != null)
    {
        System.err.println("\n---- SQLException ----");
        System.err.println(" SQL State: " + e.getSQLState());
        System.err.println(" Error Code: " + e.getErrorCode());
        System.err.println(" Message: " + e.getMessage());
        // for stack traces, refer to derby.log or uncomment this:
        //e.printStackTrace(System.err);
        e = e.getNextException();
    }
}

/**
 * Parses the arguments given and sets the values of this class' instance
 * variables accordingly - that is which framework to use, the name of the
 * JDBC driver class, and which connection protocol protocol to use. The
 * protocol should be used as part of the JDBC URL when connecting to Derby.
 * <p>
 * If the argument is "embedded" or invalid, this method will not change
 * anything, meaning that the default values will be used.</p>
 * <p>
 * @param args JDBC connection framework, either "embedded", "derbyclient"
 * or "jccjdbcclient". Only the first argument will be considered,
 * the rest will be ignored.

```

```
*/
private void parseArguments(String[] args)
{
    if (args.length > 0) {
        if (args[0].equalsIgnoreCase("jccjdbcclient"))
        {
            framework = "jccjdbc";
            driver = "com.ibm.db2.jcc.DB2Driver";
            protocol = "jdbc:derby:net://localhost:1527/";
        }
        else if (args[0].equalsIgnoreCase("derbyclient"))
        {
            framework = "derbyclient";
            driver = "org.apache.derby.jdbc.ClientDriver";
            protocol = "jdbc:derby://localhost:1527/";
        }
    }
}
```