



Working With Derby

Version 10.2

Derby Document build:
December 11, 2006, 7:06:09 AM (PST)

Contents

Copyright	3
Introduction and prerequisites	4
Activity overview	5
Activity 1: Run SQL using the Embedded driver	7
Creating the database and running SQL	7
Activity 2: Run SQL using the Client driver	10
Activity 3: Run a JDBC program using the Embedded driver	13
The WwdEmbedded program	14
Activity 4: Create and run a JDBC program using the Client driver and Network Server	18
What next with Derby	22

Copyright



Copyright 2006 The Apache Software Foundation or its licensors, as applicable.

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.

Introduction and prerequisites

Welcome to Derby! To help you get up and running with Derby as quickly as possible, this self-study guide highlights some of the more important features of Derby through a series of activities designed to demonstrate the use of Derby in embedded and client-server configurations. After performing these activities, you will find Derby to be an easy to use and fully functional RDBMS. This section provides a brief description of Derby, followed by information on the skills and software required to perform the activities presented in this document, and ends with a brief description of what is presented by each of the Working With Derby activities.

Derby is a full featured, open source relational database engine. It is written and implemented completely in Java, and provides users with a small footprint standards-based database engine that can be tightly embedded into any Java based solution. Derby ensures data integrity and provides sophisticated transaction support. In its default configuration there is no separate database server to be installed or maintained by the end user. For more information on Derby visit the Derby website at: <http://db.apache.org/derby>.

Performing the Working With Derby activities requires no prior knowledge of Java, JDBC or SQL. Each Activity Sequence section provides the complete command syntax needed to execute each operation on a Windows machine or in a UNIX/Linux Korn shell. This document demonstrates, but does not teach, the Java, JDBC and SQL presented, so readers wishing a deeper understanding of these topics will need additional reference materials.

Performing the Working With Derby activities does require that Java and Derby software be installed on the computer, and the ability to enter computer operating system commands. Specifically:

- A Java development kit version 1.3 or higher
- The binary (bin) installation of Apache Derby version 10.2
- A basic knowledge of the computer command line interface
 - how to start a command shell or window
 - how to navigate the filesystem hierarchy

If unsure about the Java or Derby environments installed, perform the following steps before attempting the subsequent activities:

1. Verify that the JAVA_HOME environment variable is set and points to a Java development kit version 1.3 or higher.

Open a command window and run the command `java -version` using the appropriate syntax for your system:

On Windows platforms:

```
%JAVA_HOME%\bin\java" -version
```

On UNIX Korn Shell platforms:

```
`${JAVA_HOME}/bin/java" -version
```

The output from the command will look something like this:

```
java version "1.4.2_04"  
Java(TM) 2 Runtime Environment, Standard Edition (build
```

```
1.4.2_04-b05)
Java HotSpot(TM) Client VM (build 1.4.2_04-b05, mixed mode)
```

The output you see may be different from what is shown here, because the `java -version` command outputs vendor-specific information. If the command produced an error or the version listed is not 1.3 or higher, please install a Java development kit before continuing.

2. Verify that the `DERBY_HOME` environment variable is set and points to the filesystem path to the root directory of the Derby 10.2 installation:

Open a command window and run the appropriate command for your system:

On Windows platforms:

```
echo %DERBY_HOME%
```

On UNIX Korn Shell platforms:

```
echo $DERBY_HOME
```

The output from the command will look something like this:

```
Windows: C:\derby
Unix/Linux: /opt/derby
```

If Derby is not installed or cannot be found, please install a copy now. The most recent version of Derby can be downloaded from: http://db.apache.org/derby/derby_downloads.html. From the Download page use the link to the 'Latest Official Release', then locate the `bin` distribution (e.g. `db-derby-<version>-bin.zip` and `db-derby-<version>-bin.tar.gz`). Download the appropriate file for your platform, '-bin.zip' for Windows or '-bin.tar.gz' for Unix/Linux, and unzip/untar this file. After decompressing the downloaded file, move the directory created to the filesystem path chosen to be the root directory of the Derby installation (e.g. move `db-derby-<version>-bin` `C:\derby`).

Anyone having problems with any aspect of these activities can gain assistance via e-mail by writing to `derby-user@db.apache.org`. The questions and feedback received will be used to make this document even more useful.

Activity overview

What you can expect to learn in each activity.

Activity 1: Use the Derby `ij` tool to load the Derby embedded driver and start the Derby database engine. Create the database `firstdb` and the table `FIRSTTABLE`. Use a few basic SQL commands to insert and select data. The Derby message log `derby.log` and the database directories and files are introduced.

Activity 2: Use Derby within a Client-Server configuration. Start the Derby Network Server which will embed the Derby engine. In a separate process use the Derby `ij` tool to load the Derby client driver and connect to the Server. Create a database called `seconddb` and the table `SECONDTABLE`. Use a few basic SQL commands to insert and select data.

Activity 3: Load the Derby database engine from a simple Java JDBC program. Use the embedded driver to create the database `jdbcDemoDB` and the `WISH_LIST` table.

Populate the table with text entered from the keyboard then view a list of the records in the table. Walk through the code to understand the basic structure of a JDBC program that accesses a Derby database. The `CLASSPATH` variable and connection URL attribute `;shutdown=true` are introduced.

Activity 4: Modify the Java JDBC program to load the client driver and connect to the Derby Network Server. Compile the altered program and test that it operates as it did in the previous activity.

Activity 1: Run SQL using the Embedded driver

In this section the Derby database engine is loaded by the Derby IJ tool. The Derby embedded driver is used to create and connect to the database firstdb. A few basic SQL commands are demonstrated.

In preparation for performing this activity the environment variable `DERBY_HOME` needs to be set and an empty working directory (`DERBYDBS`) created. The `DERBY_HOME` variable defines the filesystem path to the root directory of the Derby installation. The `DERBYDBS` directory is where the files created during the activities are stored.

The example commands defining `DERBY_HOME` use the fictitious filesystem paths `C:\derby` for Windows examples and `/opt/derby` for UNIX examples. Be sure to adjust these commands so `DERBY_HOME` indicates the location of the Derby installation on the system being used. The following activity sequence shows

1. Setting the `DERBY_HOME` environment variable.
2. Creating the `DERBYDBS` work directory
3. Changing the directory (`cd`) to the work directory
4. Copying the SQL scripts that create the tables and data for `toursdb` from the Derby `demo\programs\toursdb` subdirectory into the `DERBYDBS` directory.

Open a command window and perform the following activity sequence:

On Windows platforms:

```
set DERBY_HOME=C:\derby
md DERBYDBS
cd DERBYDBS
copy %DERBY_HOME%\demo\programs\toursdb\*.sql .
```

On UNIX Korn Shell platforms:

```
export DERBY_HOME=/opt/derby
mkdir DERBYDBS
cd DERBYDBS
cp $DERBY_HOME/demo/programs/toursdb/*.sql .
```

> **Important:** A command prompt will be displayed after each command is executed. If an error is displayed please double check the spelling and reenter the command.

Creating the database and running SQL

1. Run the Derby ij tool.

On Windows platforms:

```
java -jar %DERBY_HOME%\lib\derbyrun.jar ij
ij version 10.2
```

On UNIX Korn Shell platforms:

```
java -jar $DERBY_HOME/lib/derbyrun.jar ij
ij version 10.2
```

2. Create the database and open a connection to it using the embedded driver.

```
CONNECT 'jdbc:derby:firstdb;create=true';
```

3. Create a table with two columns using standard SQL.

```
CREATE TABLE FIRSTTABLE
  (ID INT PRIMARY KEY,
   NAME VARCHAR(12));

0 rows inserted/updated/deleted
```

4. Insert three records.

```
INSERT INTO FIRSTTABLE VALUES
  (10, 'TEN'), (20, 'TWENTY'), (30, 'THIRTY');

3 rows inserted/updated/deleted
```

5. Perform a simple select of all records in the table.

```
SELECT * FROM FIRSTTABLE;

  ID          |NAME
-----|-----
  10          |TEN
  20          |TWENTY
  30          |THIRTY

3 rows selected
```

6. Perform a qualified select of the record with column ID=20.

```
SELECT * FROM FIRSTTABLE
WHERE ID=20;

  ID          |NAME
-----|-----
  20          |TWENTY

1 row selected
```

7. Load the SQL script `ToursDB_schema.sql` to create the tables and other schema objects (this step is optional).

```
run 'ToursDB_schema.sql';

ij> CREATE TABLE AIRLINES
  ( AIRLINE CHAR(2) NOT NULL ,
    AIRLINE_FULL VARCHAR(24),
    BASIC_RATE DOUBLE PRECISION,
    . . .
0 rows inserted/updated/deleted
. . .
===> Other output messages not shown <=====
```

- a. Populate the tables with data by running the script `loadTables.sql`

```
run 'loadTables.sql';

ij> run 'loadCOUNTRIES.sql';
ij> insert into COUNTRIES values ( 'Afghanistan','AF','Asia');
1 row inserted/updated/deleted
ij> insert into COUNTRIES values ( 'Albania','AL','Europe');
1 row inserted/updated/deleted

===> Other output messages not shown <=====
```

8. Exit the ij tool.

```
exit;
```

9. Browse the key files created by the activity.

- The *derby.log* message and error log file. Under normal circumstances it will contain a set of startup messages and a shutdown message.

```
-----
2006-09-21 23:33:37.564 GMT:
  Booting Derby version The Apache Software Foundation
  - Apache Derby - 10.2.1.6 - (452058):
    instance c013800d-0109-7f82-e11f-000000119a68
    on database directory C:\DERBYDBS\FIRSTDB
  Database Class Loader started - derby.database.classpath=''

2006-09-21 23:44:13.178 GMT:
  Shutting down instance c013800d-0109-7f82-e11f-000000119a68
-----
```

- The *firstdb* database directory. Within the directory are the subdirectories *seg0* (containing the data files) and *log* (containing the transaction log files).

Description of connection command: connect

```
'jdbc:derby:firstdb;create=true';
```

connect - the ij command to establish a connection to a database

The Derby connection URL enclosed in single quotes:

- *jdbc:derby:* - JDBC protocol specification for the Derby driver.
- *firstdb* - the name of the database, this can be any string. Because no filepath is specified the database will be created in the default working directory (DERBYDBS).
- *;create=true* - The Derby *URL attribute* used to create databases. Derby does not have an SQL *create database* command.

; (semicolon) - the ij command terminator.

Activity 2: Run SQL using the Client driver

This activity uses Derby within a Client-Server configuration by using the Network Server. The `ij` tool is the client application that connects to the Derby Network Server. A database called `seconddb` is created and some basic SQL commands are executed.

This activity assumes that you know how to open a command shell, change directory (`cd`) to the `DERBYDBS` directory and set the `DERBY_HOME` environment variable.

Two command windows (labelled `Shell-1` and `Shell-2`) are used in this activity. `Shell-1` is used to start the Derby Network Server and display Network Server messages. `Shell-2` is used to establish a client connection to the Network Server using `ij` and then perform some basic SQL operations.

1. Open a command window that we'll call `Shell-1`. Change directory (`cd`) to the `DERBYDBS` directory and set the `DERBY_HOME` environment variable.
2. Start the Network Server.

On Windows platforms:

```
java -jar %DERBY_HOME%\lib\derbynet.jar start
Apache Derby Network Server - 10.2.1.6 - (452058) started and
ready to accept connections on port 1527 at 2006-09-22
00:08:30.049 GMT
```

On UNIX Korn Shell platforms:

```
java -jar $DERBY_HOME/lib/derbynet.jar start
Apache Derby Network Server - 10.2.1.6 - (452058) started and
ready to accept connections on port 1527 at 2006-09-22
00:08:30.049 GMT
```

A Network Server startup message is displayed in the `Shell-1` command window.

3. Open another command window that we'll call `Shell-2`. Change directory (`cd`) to the `DERBYDBS` directory and set the `DERBY_HOME` environment variable.
4. Start `ij`.

On Windows platforms:

```
java -jar %DERBY_HOME%\lib\derbyrun.jar ij
ij version 10.2
```

On UNIX Korn Shell platforms:

```
java -jar $DERBY_HOME/lib/derbyrun.jar ij
ij version 10.2
```

All subsequent commands are entered from the network client, and are therefore entered in the `Shell-2` command window.

5. Create and open a connection to the database using the client driver.

```
CONNECT 'jdbc:derby://localhost:1527/seconddb;create=true';
```

Remember: A client connection URL contains a hostname and a port number:
//localhost:1527/.

6. Create a table with two columns (ID and NAME) using SQL.

```
CREATE TABLE SECONDTABLE
  (ID INT PRIMARY KEY,
   NAME VARCHAR(14));
0 rows inserted/updated/deleted
```

7. Insert three records into the table.

```
INSERT INTO SECONDTABLE VALUES
  (100, 'ONE HUNDRED'), (200, 'TWO HUNDRED'), (300, 'THREE HUNDRED');
3 rows inserted/updated/deleted
```

8. Select all of the records in the table.

```
SELECT * FROM SECONDTABLE;

  ID          |NAME
-----|-----
  100         |ONE HUNDRED
  200         |TWO HUNDRED
  300         |THREE HUNDRED
3 rows selected
```

9. Select a subset of records from the table by qualifying the command.

```
ij> SELECT * FROM SECONDTABLE WHERE ID=200;

  ID          |NAME
-----|-----
  200         |TWO HUNDRED
1 row selected
```

10. Exit ij.

```
exit;
```

11. Shut down the Derby Network Server.

On Windows platforms:

```
java -jar %DERBY_HOME%\lib\derbynet.jar shutdown
Apache Derby Network Server - 10.2.1.6 - (452058) shutdown
at 2006-09-22 00:16:44.223 GMT
```

On UNIX Korn Shell platforms:

```
java -jar $DERBY_HOME/lib/derbynet.jar shutdown
Apache Derby Network Server - 10.2.1.6 - (452058) shutdown
at 2006-09-22 00:16:44.223 GMT
```

The server shutdown confirmation appears in both command windows.

Activity notes

The client connection URL contains network information (a hostname and portnumber) not found in the URL for an embedded connection. This information tells the client driver the "location" of the Network Server. The client driver sends requests to and receives responses from the Network Server.

In this activity the Derby database engine is embedded in the Network Server and returns data to the `ij` client (a client/server configuration). In contrast, establishing a connection using an embedded URL (one without `//localhost:1527/`) would have caused the Derby engine to be embedded in the `ij` application (an embedded configuration).

Network Server start up and shutdown messages are written to the `derby.log` log file along with the standard database engine messages. For example:

```
Apache Derby Network Server - 10.2.1.6 - (452058) started and ready to
accept connections on port 1527 at 2006-09-22 00:08:30.049 GMT
...( database engine messages not shown )...
Apache Derby Network Server - 10.2.1.6 - (452058) shutdown
at 2006-09-22 00:16:44.223 GMT
```

Though not demonstrated here, multiple client programs can connect to Network Server and access the database simultaneously in this configuration.

Activity 3: Run a JDBC program using the Embedded driver

This activity loads the Derby database engine using a simple Java JDBC program. JDBC is the Java Database Connectivity API and is also the native API for Derby. The program uses the embedded driver to create (if needed) and then connect to the `jdbcDemoDB` database. You can then populate a table within the database with text. The program demonstrates some basic JDBC processing along with related error handling.

This activity assumes that you have opened a command window, navigated to the `DERBYDBS` directory, and set the `DERBY_HOME` environment variable.

The `CLASSPATH` environment variable is used by Java to locate the binary files (jarfiles and classfiles) needed to run Derby and other Java applications. Before performing this activity, you need to set the `CLASSPATH` and compile the `WwdEmbedded.java` Java program.

1. Copy the program files into the `DERBYDBS` directory and set the `CLASSPATH`:

On Windows platforms:

```
copy %DERBY_HOME%\demo\programs\workingwithderby\* .
set CLASSPATH=%DERBY_HOME%\lib\derby.jar;
```

On UNIX Korn Shell platforms:

```
cp $DERBY_HOME/demo/programs/workingwithderby/* .
export CLASSPATH=$DERBY_HOME/lib/derby.jar:.
```

> Important: Include the dot (.) at the end of each command so that your current working directory is included in the `CLASSPATH` and the files are copied to the correct location.

2. Compile the `WwdEmbedded.java` program:

```
javac WwdEmbedded.java
```

> Important: Only a command prompt will be displayed if the compilation is successful. The binary file `WwdEmbedded.class` will be created. If an error is displayed please verify that the Java development kit is properly installed.

3. Run the program:

The `WwdEmbedded.java` program populates a table with wish list items. It is a simple Java program that prompts the User for text input (up to 32 characters), stores the text entered in a database table and then lists the items stored in the table. The program will continue to ask for wish list items until the word `exit` is entered or a problem is encountered. Some basic information on program progress is displayed at the beginning and the end of the program.

```
java WwdEmbedded
org.apache.derby.jdbc.EmbeddedDriver loaded.
Connected to database jdbcDemoDB
. . . . creating table WISH_LIST

Enter wish-list item (enter exit to end):
a peppermint stick

On 2006-09-21 15:11:50.412 I wished for a peppermint stick
```

```

Enter wish-list item (enter exit to end):
an all expenses paid vacation

On 2006-09-21 15:11:50.412 I wished for a peppermint stick
On 2006-09-21 15:12:47.024 I wished for an all expenses paid
vacation

Enter wish-list item (enter exit to end):
exit

Closed connection
Database shut down normally
Working With Derby JDBC program ending.

```

The WwdEmbedded program

This section describes the `WwdEmbedded.java` program, highlighting details specific to accessing a Derby database from a JDBC program.

Most of the code related to the database activities performed is included in this document but you may find it helpful to open the program file and follow along in a text viewer or editor. The *SECTION NAMES* referred to in this text can be found in the comments within the program code and serve as cross-reference points between this document and the Java program. The program utilizes routines from the `WwdUtils` class. The utility class code is not described here but is available for review in the file `WwdUtils.java`

Initialize the program

INITIALIZATION SECTION: The initial lines of code identify the Java libraries used in the program, then set up the Java class `WwdEmbedded` and the `main` method signature. Refer to a standard Java programming guide for information on these program constructs.

```

import java.sql.*;
public class WwdEmbedded
{
    public static void main(String[] args)
    {

```

Define key variables and Objects

DEFINE VARIABLES SECTION: The initial lines of the `main` method define the variables and Objects used in the program. This example uses variables to store the information needed to connect to the Derby database. Using variables for this information makes it easy to adapt the program to other configurations and other databases.

- *driver* - stores the name of the Derby embedded driver.
- *dbName* - stores the name of the database.
- *connectionURL* - stores the Derby connection URL that will be used to access the database.
- *createString* - stores the SQL CREATE statement for the *WISH_LIST* table .

```

String driver = "org.apache.derby.jdbc.EmbeddedDriver";
String dbName="jdbcDemoDB";
String connectionURL = "jdbc:derby:" + dbName + ";create=true";
String createString = "CREATE TABLE WISH_LIST "
    + "(WISH_ID INT NOT NULL GENERATED ALWAYS AS IDENTITY "
    + "... "
    + " WISH_ITEM VARCHAR(32) NOT NULL) " ;

```

Start the Derby engine

LOAD DRIVER SECTION: Loading the Derby embedded JDBC driver starts the Derby database engine. The `try` and `catch` block (the Java error handling construct) catches

the exceptions that may occur. A problem here is generally due to an incorrect CLASSPATH setting.

```
String driver = "org.apache.derby.jdbc.EmbeddedDriver";
...
try{
    Class.forName(driver);
} catch(java.lang.ClassNotFoundException e) {
    ...
}
```

Boot the database

BOOT DATABASE SECTION: The DriverManager class loads the database using the Derby connection URL stored in the variable `connectionURL`. This URL includes the parameter `;create=true` so the database will be created if it does not already exist. The primary `try` and `catch` block begins here. This construct handles errors for the database access code .

```
String connectionURL = "jdbc:derby:" + dbName + ";create=true";
...
try {
    conn = DriverManager.getConnection(connectionURL);
    ... <most of the program code is contained here>
} catch (Throwable e) {
    ...
}
```

Set up to execute SQL

INITIAL SQL SECTION: Program objects needed to perform subsequent SQL operations are initialized here and a check is made to see if the required data table exists.

The statement object `s` is initialized. If the utility method `WwdUtils.wwdChk4Table` does not find the `WISH_LIST` table it is created by executing the SQL stored in the variable `createString` via this statement object.

```
s = conn.createStatement();
if (! WwdUtils.wwdChk4Table(conn))
{
    System.out.println(" . . . creating table WISH_LIST");
    s.execute(createString);
}
```

The insert statement used to add data to the table is bound to the prepared statement object `psInsert`. The prepared statement uses the `?` parameter to represent the data that will be inserted by the user. The actual value that is inserted is `set` later in the code prior to executing the SQL. This is the most efficient way to execute SQL statements that will be used multiple times.

```
psInsert = conn.prepareStatement
    ("insert into WISH_LIST(WISH_ITEM) values (?);");
```

Interact with the Database

ADD / DISPLAY RECORD SECTION: This section uses the utility method `WwdUtils.getWishItem` to gather information from the User. It then utilizes the objects set up previously to insert the data into the `WISH_LIST` table and then display all records. A standard `do` loop causes the program to repeat this series of steps until `exit` is entered. The data related activities performed in this section are:

The `setString` method sets the substitution parameter of the `psInsert` object to the value entered by the User. Then `executeUpdate` is called to perform the database insert.

```
psInsert.setString(1,answer);
psInsert.executeUpdate();
```

The statement object `s` is used to select all the records in the `WISH_LIST` table and store them in the `ResultSet myWishes`.

```
myWishes = s.executeQuery("select ENTRY_DATE, WISH_ITEM
                           from WISH_LIST order by ENTRY_DATE");
```

The while loop reads each record in turn by calling the `next` method. The `getTimestamp` and `getString` methods return specific fields in the record in the proper format. The fields are displayed using rudimentary formatting.

```
while (myWishes.next())
{
    System.out.println("On " + myWishes.getTimestamp(1) +
                       " I wished for " + myWishes.getString(2));
}
```

Close the `ResultSet` to release the memory being used.

```
myWishes.close();
```

Shutdown the Database

DATABASE SHUTDOWN SECTION: When an application starts the Derby engine it should shutdown all databases prior to exiting. The attribute `;shutdown=true` in the Derby connection URL performs the shutdown. The shutdown process cleans up records in the transaction log to ensure a faster startup the next time the database is booted.

This section verifies that the embedded driver is being used then issues the shutdown command and catches the shutdown exception to confirm the database shutdown cleanly. The shutdown status is displayed before the program exits.

```
if (driver.equals("org.apache.derby.jdbc.EmbeddedDriver")) {
    boolean gotSQLException = false;
    try {
        DriverManager.getConnection("jdbc:derby:;shutdown=true");
    } catch (SQLException se) {
        if ( se.getSQLState().equals("XJ015") ) {
            gotSQLException = true;
        }
    }
    if (!gotSQLException) {
        System.out.println("Database did not shut down normally");
    } else {
        System.out.println("Database shut down normally");
    }
}
```

> Important: The `XJ015` error is the only exception thrown by Derby that signifies an operation succeeded. All other exceptions indicate an operation failed.

The `errorPrint` and `SQLExceptionPrint` methods

DERBY EXCEPTION REPORTING CLASSES: The two methods at the end of the file, `errorPrint` and `SQLExceptionPrint`, are generic exception reporting routines that can be used with any JDBC program. This type of exception handling is required because often multiple exceptions (`SQLException`) are chained together then thrown. A while loop is used to report on each error in the chain. These classes are used by calling the `errorPrint` method from the `catch` block of the code that accesses the database.

```

// Beginning of the primary catch block: uses errorPrint method
} catch (Throwable e) {
    /* Catch all exceptions and pass them to
    ** the exception reporting method */
    System.out.println(" . . . exception thrown:");
    errorPrint(e);
}

```

The `errorPrint` routine prints a stack trace for all exceptions except a `SQLException`. All `SQLException`s are passed to the `SQLExceptionPrint` method.

```

static void errorPrint(Throwable e) {
    if (e instanceof SQLException)
        SQLExceptionPrint((SQLException)e);
    else {
        System.out.println("A non SQL error occured.");
        e.printStackTrace();
    }
} // END errorPrint

```

The `SQLExceptionPrint` method iterates through each of the exceptions on the stack. For each error the codes, message then stacktrace are printed.

```

// Iterates through a stack of SQLExceptions
static void SQLExceptionPrint(SQLException sqle) {
    while (sqle != null) {
        System.out.println("\n---SQLException Caught---\n");
        System.out.println("SQLState: " + (sqle).getSQLState());
        System.out.println("Severity: " + (sqle).getErrorCode());
        System.out.println("Message: " + (sqle).getMessage());
        sqle.printStackTrace();
        sqle = sqle.getNextException();
    }
} // END SQLExceptionPrint

```

If you wish to see the output produced by this method enter a wish list item with more than 32 characters like: I wish to see a Java program fail.

Activity 4: Create and run a JDBC program using the Client driver and Network Server

This section demonstrates the ease with which a program that embeds Derby can be modified for a client/server implementation using the Derby Network Server. A Derby client program, `WwdClient.java`, is created by changing a few lines of the `WwdEmbedded.java` program. The client program can be run in multiple command shells allowing simultaneous update from two or more sources.

This activity assumes you have performed the preceding activities and so have a working directory called `DERBYDBS`, are familiar with setting the `DERBY_HOME` and `CLASSPATH` environment variables and have copies of the program files from the `$DERBY_HOME/demo/programs/workingwithderby/` directory. A basic knowledge of the `WwdEmbedded.java` program and experience starting and connecting to Network Server is helpful. You will need to use a text editor to create the `WwdClient.java` program.

Two command windows (`Server-Shell` and `Client-Shell`) are used in this activity. The `Server-Shell` is used to start the Derby Network Server and display Network Server messages. The `Client-Shell` is used to edit, compile and run the newly created `WwdClient.java` program. The `CLASSPATH` environment variable is set in `Client-Shell` to support the client JDBC program.

1. **Create the `WwdClient` program.**
 - a. Open a command window that we'll call the `Client-Shell`.
 - b. Change directory (`cd`) to the `DERBYDBS` directory.
 - c. Make a copy of the `WwdEmbedded.java` program called `WwdClient.java`.

On Windows platforms:

```
copy WwdEmbedded.java WwdClient.java
```

On UNIX Korn Shell platforms:

```
cp WwdEmbedded.java WwdClient.java
```

- d. Open the `WwdClient.java` file in your favorite text editor and update the class name to reflect the new filename:

```
Original declaration
public class WwdEmbedded

New declaration
public class WwdClient
```

- e. Edit the `DEFINE VARIABLES SECTION` of the program so the `driver` variable contains the name of the Derby Client Driver class and the `connectionURL` variable contains the hostname and a port number of the Network Server.

```
Original definitions
String driver = "org.apache.derby.jdbc.EmbeddedDriver";
String dbName="jdbcDemoDB";
String connectionURL = "jdbc:derby:" + dbName +
";create=true";

New definitions
String driver = "org.apache.derby.jdbc.ClientDriver";
...
String connectionURL = "jdbc:derby://localhost:1527/" +
```

```
dbName + ";create=true";
```

- f. Compile the application.

```
javac WwdClient.java
```

> Important: Only a command prompt will be displayed if the compilation is successful. The binary file `WwdClient.class` will be created. If a syntax error is displayed, modify the line indicated so it is identical to the example.

That's all there is to it.

2. Set up the client/server environment.

Before you run the `WwdClient` program, the Network Server needs to be started.

- a. Open a command window that we'll call the `Server-Shell`.
- b. Change directory (`cd`) to the `DERBYDBS` directory.
- c. Set the `DERBY_HOME` environment variable.
- d. Start the Network Server:

On Windows platforms:

```
java -jar %DERBY_HOME%\lib\derbynet.jar start
Apache Derby Network Server - 10.2.1.6 - (452058) started and
ready to accept connections on port 1527 at 2006-09-22
00:56:33.091 GMT
```

On UNIX Korn Shell platforms:

```
java -jar $DERBY_HOME/lib/derbynet.jar start
Apache Derby Network Server - 10.2.1.6 - (452058) started and
ready to accept connections on port 1527 at 2006-09-22
00:56:33.091 GMT
```

3. Run the client program.

- a. Return to the `Client-Shell` window.
- b. If it is not already defined, set the `DERBY_HOME` environment variable.
- c. Set the `CLASSPATH` environment variable to include the location of the file `derbyclient.jar`:

On Windows platforms:

```
set CLASSPATH=%DERBY_HOME%\lib\derbyclient.jar;.
```

On UNIX Korn Shell platforms:

```
export CLASSPATH=$DERBY_HOME/lib/derbyclient.jar:.
```

> Important: Include the dot (`.`) at the end of the command so that your current working directory is included in the `CLASSPATH`.

- d. Run the program:

```

java WwdClient

org.apache.derby.jdbc.ClientDriver loaded.
Connected to database jdbcDemoDB

Enter wish-list item (enter exit to end):
a sunny day

-----
On 2006-09-21 15:11:50.412 I wished for a peppermint stick
On 2006-09-21 15:12:47.024 I wished for an all expenses paid
vacation
On 2006-09-22 10:08:21.167 I wished for a sunny day
-----

Enter wish-list item (enter exit to end):
a new car

-----
On 2006-09-21 15:11:50.412 I wished for a peppermint stick
On 2006-09-21 15:12:47.024 I wished for an all expenses paid
vacation
On 2006-09-22 10:08:21.167 I wished for a sunny day
On 2006-09-22 10:08:33.665 I wished for a new car
-----

Enter wish-list item (enter exit to end):
exit

Closed connection
Working With Derby JDBC program ending.

```

4. Shut down the Network Server.

On Windows platforms:

```

java -jar %DERBY_HOME%\lib\derbynet.jar shutdown

Apache Derby Network Server - 10.2.1.6 - (452058) shutdown
at 2006-09-22 19:13:51.445 GMT

```

On UNIX Korn Shell platforms:

```

java -jar $DERBY_HOME/lib/derbynet.jar shutdown

Apache Derby Network Server - 10.2.1.6 - (452058) shutdown
at 2006-09-22 19:13:51.445 GMT

```

The server shutdown confirmation appears in both command windows.

Activity notes

In a client/server environment, the client program is often used from other computers on the network. Whenever a system accepts connections from other computers, there is a chance of abuse. To maintain security, the Derby Network Server defaults to accepting connections only from clients running on the local machine (`localhost`). Before this or any other Derby client program can access Network Server from another machine, additional steps should be taken to secure the Network Server environment. Once secured, the Network Server can be safely configured to accept connections from other machines. Refer to the *Network Server security* and *Running the Network Server under the security manager* sections of the *Derby Server and Administration Guide* for important information on securing the Network Server and enabling network connections.

With Network Server started, you can run the client program simultaneously in multiple windows. To demonstrate this, open two command windows and perform the substeps of the *Run the client program* step in each window. Both clients will operate without a problem. In contrast, it would not be possible for a program that uses the embedded driver (e.g. `WwdEmbedded`) to access the database until the database or the Network

Server is shut down.

You may have noticed that the client program does not shut down the database. This is because the database is a shared resource in a client/server environment and, in most cases, should only be shut down when the Server is shut down. If multiple clients are accessing the database and one shuts down the database, the remaining clients will encounter a failure the next time they attempt an SQL command.

Derby's two architectures have caused confusion for some new Derby users. They mistakenly think that embedded is a single user configuration. This is not true. The embedded driver supports multiple simultaneous connections, performs locking, and provides performance, integrity and recoverability. Any application using the embedded driver can open multiple Derby connections and then provide a means for multiple users to interact with the database on each connection. The Derby Network Server is an example of such an application.

What next with Derby

Congratulations on completing the activities in this workbook. You now have experience with using Derby in both the embedded and Client-Server architectures. With this basic knowledge you are ready to begin using Derby to address your own specific needs. We recommend visiting the Apache Derby website as your next step in learning about this lightweight and powerful tool.

Use this link: [WorkingWithDerby Resources page](http://wiki.apache.org/db-derby/WorkingWithDerby)

browser URL:

`http://wiki.apache.org/db-derby/WorkingWithDerby`

Activities Summary

We hope you have found these activities useful in understanding the steps needed to create and access Derby databases. Though simple to setup and use you will find that Derby has the features and reliability of much larger database systems. The examples presented here do not begin to scratch the surface of what can be done. Please take a few moments to become familiar with the many online resources available to Derby users and developers by browsing the Derby website at Apache. Whether you are performing a general evaluation of Derby or have a specific need to address, the above link is a good stepping stone to finding additional information of interest. The [Derby Quick Start page](#) is a good reference page organized by area of interest. You will find many content rich areas such as the *Derby Wiki* and the *Derby Users mailing list* available to you. If you are interested in how others are using Derby see the *Uses of Derby* page on the WIKI. This page contains informational links to development projects and products that use Derby. When you implement a system using Derby please add it to this list.