

## Developing Applications using the Derby Plug-ins

### Lab Instructions

**Goal:** Use the Derby plug-ins to write a simple embedded and network server application.

In this lab you will use the tools provided with the Derby plug-ins to create a simple database schema in the Java perspective and a stand-alone application which accesses a Derby database via the embedded driver and/or the network server. Additionally, you will create and use three stored procedures that access the Derby database.

**Intended Audience:** This lab is intended for both experienced and new users to Eclipse and those new to the Derby plug-ins. Some of the Eclipse basics are described, but may be skipped if the student is primarily focused on learning how to use the Derby plug-ins.

### High Level Tasks Accomplished in this Lab

Create a database and two tables using the jay\_tables.sql file. Data is also inserted into the tables.

Create a public java class which contains two static methods which will be called as SQL stored procedures on the two tables.

Write and execute the SQL to create the stored procedures which calls the static methods in the java class.

Test the stored procedures by using the SQL 'call' command.

Write a stand alone application which uses the stored procedures. The stand alone application should accept command line or console input.

### Detailed Instructions

These instructions are detailed, but do not necessarily provide each step of a task. Part of the goal of the lab is to become familiar with the Eclipse Help document, the Derby Plug-ins User Guide. Use this help document to assist with completing any task associated with using the plug-ins like starting and stopping the network server or running SQL scripts.

1. Unzip the file *derby\_plugins\_lab.zip* into a convenient location on the host where Eclipse is installed.
2. After unzipping this file the following directory structure and files are present:

Directory	Files	Description
.	derby_plugins_lab.sxw	This file in OpenOffice format.

	derby_plugins_lab.pdf	This file in PDF format.
sql	execute_sp.sql	A sample sql script to execute the stored procedures.
	jay_tables.sql	The sql script to create the necessary jay_species and jay_attributes tables and populate them with data.
	stored_procedures.sql	The create procedure sql commands.
java	Jays.java	A possible solution to the stand alone Java application.
	JaySpecies.java	The public Java class that contains static methods which implement the logic of the stored procedures.

3. If Eclipse 3.1M6 or higher is not already installed, install it now. Install both the 10.1.1 Core and the 1.1.0 UI and Help plug-ins if they have not already been installed. Refer to the document Using the 10.1 Core and 1.1 UI Derby plug-ins at [http://db.apache.org/derby/integrate/plugin\\_howto.html](http://db.apache.org/derby/integrate/plugin_howto.html) if you need help.
4. Launch Eclipse and bring up the Help for the Derby plug-ins. **Help >> Help Contents>> Derby Plug-in User Guide.**
5. Task: Create a database containing the *jay\_species* and *jay\_attributes* tables.
  - a) Create a java project and name it **DerbyLab**.
  - b) Add the Apache Derby Nature to the **DerbyLab** project.
  - c) Import the *jay\_tables.sql* file (including the sql directory) from the directory where you unzipped the *derby\_plugins\_lab.zip* file. To import the file: Right click the DerbyLab project, select Import, then File System, click the Next button. Browse to the directory where you unzipped the lab zip file. Once this is done the directory will appear as a folder icon in the left frame of the Import window. Expand this folder and check only the sql directory (the parent directory will have a grey check next to it, but don't directly check this directory too.) The 'Into Folder' text field should say '**DerbyLab**'. The radio button, 'Create selected folders only' should be the only one selected. Now click 'Finish'. If the import is successful your DerbyLab project folder should have an sql directory under it along with the three sql files.
  - d) Start the Derby network server by right-clicking the **DerbyLab** project and selecting **Apache Derby >> Start Derby Network Server**. Confirm the pop-up box stating the server is being started.
  - e) Now run the *jay\_tables.sql* file via ij as a script, but first open it. Open the sql

folder in the Package Explorer view and double click the jay\_tables.sql file to see the contents of it. Right click the *jay\_tables.sql* file from the Package Explorer View and select **Apache Derby >> Run SQL Script using ij**. If the script runs successfully the console view will show output from running the script, with the output from the select statements.

- 6) Create a public java class called *JaySpecies2.java* containing three static methods which use PreparedStatements.
  - a) Create a java class by right-clicking the **DerbyLab** project and selecting **New >> Class**. Follow the wizard to create the class, placing the class in a package, for instance, **org.apache.derby.plugin.lab**.
  - b) Name the first static method **selectLatinCommonName**. This method takes the genus name as input, and the common and latin names of the genus are returned as a ResultSet. The signature for this java method looks like this:

***public static void selectLatinCommonName(String genus,  
ResultSet[] common\_latin) throws SQLException***

The full code for the first static method is shown below:

```
public static void selectLatinCommonName(String genus,  
    ResultSet[] common_latin) throws SQLException  
{  
  
    Connection conn = DriverManager  
        .getConnection("jdbc:default:connection");  
  
    String select = "select common_name, latin_name from  
        APP.jay_species where genus = ?";  
  
    PreparedStatement prepStmt = conn.prepareStatement(select);  
    prepStmt.setString(1, genus);  
    common_latin[0] = prepStmt.executeQuery();  
  
    conn.close();  
}
```

- c) Given the common\_name as input, all attributes from the *jay\_attributes* table are returned as a ResultSet. The java signature for the static method to be used for this stored procedure should look like this:

***public static void selectAttributes(String common\_name,  
 ResultSet[] jay\_attributes) throws SQLException***

- d) Given a minimum and maximum value for the wing span as input return the genus,

latin\_name, common\_name, wing\_span, length, weight, range and voice of all jays which have wing spans greater than or equal to the minimum wing span and less than or equal to the maximum wing span. The java signature for the static method is this:

```
public static void wingSpan(java.math.BigDecimal minWingSpan,  
    java.math.BigDecimal maxWingSpan,  
    ResultSet[] jay_attributes) throws SQLException
```

- e) Make sure the class compiles correctly.
  - f) Optional: Import the java class, ***JaySpecies.java*** to see the solution and not write the code yourself.
- 7) Either create a new file called ***stored\_procedures2.sql*** and write the Stored Procedures which call the static methods you just created or open the ***stored\_procedures.sql*** file you imported earlier.
- a) An example of the sql to create the stored procedure named selectName which calls the static method ***selectLatinCommonName*** shown above is listed below:

```
create procedure selectNames(IN genus VARCHAR(30)  
    parameter style java dynamic result sets 1 language java  
    external name  
    'org.apache.derby.plugin.lab.JaySpecies.selectLatinCommonName'  
    READS SQL DATA;
```

- b) Create all three sql procedures to correspond with the three static methods.
  - c) Run the create procedure sql commands to create them in the database using ij.
  - d) To run the commands as an entire script in ij the first statement in the script needs to connect to the ***jaysDB*** database. Make sure the network server is started if you are connecting using the Derby Client driver.
- 8) Either create a new file called ***execute\_sp2.sql*** which calls each stored procedure you created above with some sample data from the ***jay\_attributes*** or ***jay\_species*** tables or open the ***execute\_sq.sql*** file you imported earlier.
- a) An example of calling the 'selectNames' stored procedure is:

```
call selectNames('Cyanocitta');
```

- b) Make sure to test all three of the procedures before continuing to ensure they work correctly.
- 9) Create a stand-alone Java class (it includes a main method) called ***Jays2.java*** which either allows for command line arguments or reads input from the console which calls all of the stored procedures correctly given various input. Alternately import ***Jays.java*** from the lab zip file into your project to see a possible solution.
- a) An example of a class that reads command line arguments could be invoked like

this:

***java Jays names Cyanocitta***

The results would be the output of the selectLatinCommonName static method with Cyanocitta as the input.

- b) Another example of calling a stored procedure is:

***java Jays attributes Western Scrub-Jay***

The results would show all attributes for the bird with the common name of Western Scrub-Jay

- c) A final example would be to create a class that reads from standard input as a console program. An example of how this works is shown below:

**Usage:**

**names <genus\_name>,  
attributes <common\_name>,  
wingspan <min\_wingspan> <max\_wingspan>,  
quit**

**>> names Nucifraga**

**Common name: Clark's Nutcracker, Latin name: Nucifraga columbiana**

**Usage:**

**names <genus\_name>,attributes <common\_name>,  
wingspan <min\_wingspan> <max\_wingspan>,  
quit**

**>> attributes Clark's Nutcracker**

**Common name: Clark's Nutcracker, Wingspan in: 24.00, Length in: 12.0,  
Weight oz: 4.6, Range: southwest, northwest,  
Voice: Varied; common call a long, harsh, slightly rising shraaaaaa.**

**Usage:**

**names <genus\_name>,  
attributes <common\_name>,  
wingspan <min\_wingspan> <max\_wingspan>,  
quit**

**>> wingspan 19.0 25.0**

**Genus: Cyanocitta, Latin name: Cyanocitta stelleri**

**Common name: Stellar's Jay, Wingspan in: 19.00, Length in: 11.5, Weight oz:  
3.7, Range: pacific, queen charlotte islands,  
Voice: Varied; most common a very harsh, unmusical, descending shaaaar**

**Genus: Nucifraga, Latin name: Nucifraga columbiana**  
**Common name: Clark's Nutcracker, Wingspan in: 24.00, Length in: 12.0,**  
**Weight oz: 4.6, Range: southwest, northwest,**  
**Voice: Varied; common call a long, harsh, slightly rising shraaaaaa.**

**Genus: Pica, Latin name: Pica nuttalli**  
**Common name: Yellow-billed Magpie, Wingspan in: 24.00, Length in: 16.5,**  
**Weight oz: 5.0, Range: california coast, valley,**  
**Voice: Chatter call reportedly higher-pitched and clearer than Black-billed.**