Project 1 Tips (Fall 2014)

Table of Contents

Common Bugs

Using two-table Columns on a single mega-table for two table select

If you're using the technique suggested in the skeleton/video where you create Columns whose constructors are provided with two Tables (i.e. with possible values for _table of 0 or 1), then it is important that you provide them with rows corresponding to BOTH of these original tables when filtering.

In other words, if you're trying to do this, but you've built a giant table consisting of all pairwise combinations and are thus only giving a single row to `new Row(List<Column>, row ...)`, then it won't work. If you're using this technique, you don't need to explicitly build the big pairwise combination table (and indeed doing so will considerably complicate things).

This section will be updated with bugs as we go through bug-submits.

More Tips

javadoc

Use the javadoc command to generate HTML documentation for the db61b class. The example below creates documentation and puts it in the ~/p1docs folder on your machine. The javadoc command should be installed on your machine if you have a Java compiler.

```
  $ javadoc -private db61b -d ~/p1docs
```

Make sure to include the `-private` flag, which will ensure that private, protected, and package protected methods are include in the documentation.

After running this command, open `~/p1docs/index.html` in your web browser.

Mergeskel

mergeskel was described in project 0's README file. The documentation is repeated below for your convenience:

```
MERGING CHANGES:
----------------

If and when we publish new versions of the skeleton, you may want to
incorporate those changes in your project.  This is known as "merging".
Basically, it works like this:

   0. Commit your current files (as with 'hw commit'). NEVER start
      merging files until you have done this successfully!!!
   1. Compute the set of differences between the version of the skeleton
      from which your current code comes and the current version of the
      skeleton.
   2. Try to apply these differences to your current code.
   3. Where the changes in the skeleton overlap your changes, there are
      "merge conflicts".  Edit the files containing these conflicts and
      resolve the conflict as appropriate.
   4. When done, mark the files as no longer being conflicted.
   5. Commit the new version of your files.  Again, NEVER do
      additional editing until you have successfully committed.

This process is so common that all widely used version-control systems
support it with one or more commands.  In Subversion, this is the 'svn
merge' command.  Rather than have you confront this directly, we've
introduced the command 'mergeskel' to do steps 1 and 2 and
'resolveskel' to do step 4.  So,

   A. Put yourself in your project working directory first.
   B. Commit any changes (hw commit)
   C. Run mergeskel
   D. Edit out any conflicts (hw status will tell you which files
      have conflicts).
   E. Run resolveskel, which tells Subversion that the conflicts are
      fixed (it otherwise will not let you commit.)
   F. Commit the results.

If you have files lying around with names like "FOO.merge-right.r1234", you
haven't resolved something.

The rest of this section explains the full version of what's really
going on (you can skip it, but it's good to know).  We store copies of
all versions of the skeleton in staff repository files called (on the
instructional machines)

       $STAFFREPOS/tags/projN-V

where projN is the project (e.g., proj1) and V is a version number
(starting at 0).  The current version of the skeleton is in

       $STAFFREPOS/projN

The file staff-version in your project directory, if present, contains the
the tag name of that skeleton (if it isn't present, the tag name is projN-0,
as in proj1-0).

In a project working directory, say proj1, after committing all your
current files, you (or mergeskel) merge in a new version of the
skeleton (on the instructional machines) by first figuring out (using
staff-version, if present) what was the last version you updated to.
Suppose this is proj1-0.  Now you enter the commands

        svn merge --accept=postpone $STAFFREPOS/tags/proj1-0 $STAFFREPOS/proj1

This produces some progress messages, possibly including some messages
about conflicts.  If there are no conflicts, you can simply commit your files
and you are done.

Otherwise, each file with conflicts will have sections like this

    <<<<<<< .working
         System.out.println ("Welcome to my project");
         initialize ();
    =======
         initialize (args);
    >>>>>>> .merge-right.r1009

The part before the ======= was in your file to start with.  The part after
======= is from the updated version of the skeleton.  Choose which you
want to use, or what combination you want to use and edit accordingly.
Then remove the marker lines (<<<<, >>>>, ====) and save.  Do this for
all conflicted files.  Finally, you (or resolveskel) can run the command

    svn resolved --accept working <FILE>

for each <FILE> you have resolved in this way.  If you've done this
right, 'hw status' will no longer show conflicts (just modified
files).  Now commit your work and you're done.
```

FAQ and Common Misconceptions

Q: How do I read from a .db file if it's in the testing folder? You can use the same trick that is used by `make check`. From your testing folder, run your code with `java -cp ../:$CLASSPATH db61b.Main`.

Q: How do I use the debugger? I'm getting a NoClassDefFoundError. The gjdb command has a -classpath argument, similar to the -cp argument we pass to the java interpreter. For example, to run my basic test from the testing directory, I do the following:

```
jug@Hvlargs-MacBook-Pro : ~/Dropbox/61b/projects/p1/db61b
$ gjdb -classpath ../:$CLASSPATH db61b.CoreTests
gjdb version 6.2.7 (for JDK 1.7.0_25)
  Please send all questions, comments, and bug reports
  to Hilfinger@cs.berkeley.edu.
Initializing...
[-] run
java db61b.CoreTests
Program Started.
[josh, 61249]
Time: 0.014
Ran 3 tests. All passed.

The application exited
```

This section will be updated as questions recur.