

HW3 WebDB

For this assignment you will construct a simple client/server web database. It supports browsing and editing on the client side for a database stored on a centralized server. This is actually a pretty useful tool to have around if you need something that can be both read and written by a bunch of people spread all over the place. There's some Perl code and other materials to get you started in the homework directory. HW3 is due midnight ending Tue May 21st.

Part A vs. Part B

Part A of the assignment builds all the basic web database functionality. Part B will add in a couple advanced features. As usual, P/NC students may work in teams of two, and only need to do Part A, though the screen shots show the Part B interface.

WebDB Home Page

In the simplest case, WebDB is invoked with a single "db" binding to specify the filename to open — e.g. "webdb.pl?db=movies.db". In that case, WebDB returns its simple two part home page. The top section contains controls for the basic operations, and the bottom section shows rows from the database in a big HTML table...

Your solution should have the same structure and capabilities shown here, but it's ok if your HTML has a different esthetic -- different table border style, etc..

CS193i Movies

The CS193i movie database.

All 3 records...

Title	Staring	Description	
Alien	Yaphet Kotto, Sigourney Weaver, Harry Dean Stanton	Awesome scary space movie, especially considering it was made in 1979.	<input type="button" value="Edit"/> Mon May 13 10:45:46 2002 nick@stanford.edu
Midnight Run	Yaphet Kotto, Robert Deniro	Great comedy about a bounty hunter.	<input type="button" value="Edit"/> Mon May 13 10:45:51 2002 nick@stanford.edu
Sense and Sensibility	Emma Thompson, Hugh Grant	Romantic comedy -- best adaption of all the Jane Austen novels.	<input type="button" value="Edit"/> Mon May 13 10:46:03 2002 nick@stanford.edu

3 records

There is one column in the table for each column in the database file. There is an extra column on the right hand side with a little "Edit" button for each row. (Part b functionality: the edit field also shows the last-mod time for that row and the IP address of the client who edited it.)

Show All vs. Search

The "Show All Records" and "Search" buttons do practically the same thing. Show All Records yields the home page show above. The Search button yields the home page, except it only shows the rows that contain the substring typed

into the search text field. The substring search should not be case-sensitive, and (Part B) may also search the time and IP. If the user types something in the search field and then clicks the Show All Records, button, the Show All should take precedence. Here the user has typed the string "kot" into the text field and clicked the Search button yielding the two movies starring Yaphet Kotto...

The screenshot shows a Netscape browser window titled "Netscape: CS193i Movies". The address bar contains the URL: `http://cgi-courses.stanford.edu/~nick/cgi-bin/webdb.pl?db=movies.db&target=kot`. The page content includes a search interface with buttons for "Show All Records", "Search", and "Add a New Record". Below the search interface, a table displays the search results for the target 'kot'.

Records containing the target 'kot'...

Title	Starring	Description	
Alien	Yaphet Kotto, Sigourney Weaver, Harry Dean Stanton	Awesome scary space movie, especially considering it was made in 1979.	<input type="button" value="Edit"/> Mon May 13 10:45:46 2002 nick3@stanford.edu
Midnight Run	Yaphet Kotto, Robert Deniro	Great comedy about a bounty hunter.	<input type="button" value="Edit"/> Mon May 13 10:45:51 2002 nick3@stanford.edu

2 records

The little preface sentence before the table is slightly different for the two cases: it is "All n records" for Show All, while it is "Records containing the target ' $target$ '" for Search. If the target string is the empty string or it is not present, the home page should do a Show All. For example, the very first screenshot on page 2 shows the CGI being called with no bindings specifying a search, so it defaults to a Show All. At the bottom of the table there is a count of the number of record

shown. Consider writing the code so the pluralization of the word "record" is correct.

3. Edit

Selecting the Edit button on a record "checks out" the current contents of that record into an to an edit form that has a 60 column <textarea> for each field in the database labeled with that Field's name. The check-out operation should not modify the database file. (Part B -- The previous last-mod time and address for the page are mentioned at the bottom of the page).

Netscape: CS193i Movies

Back Forward Reload Home Search Netscape Images Print Security Shop Sto

Location: <http://cgi-courses.stanford.edu/~nick/cgi-bin/webdb.pl?startedit=Edit&id=1&db=movies.db&target=>

CS193i Movies

The CS193i movie database.

Show All Records Search Add a New Record

Save This Record Delete This Record Cancel Edit

Title	Alien
Starring	Yaphet Kotto, Sigourney Weaver, Harry Dean Stanton
Description	Awesome scary space movie, especially considering it was made in 1979.

Last edited at 'Mon May 13 10:45:46 2002' by 'nick3.stanford.edu'

Save Button

After editing the fields, the Save button tries to write the edited values back into the appropriate row in the database back on the server. If the save is successful, the Save basically goes back to the home page, except there's a little "Record saved" confirmation sentence before the table. In the following example, the user edited the description for the movie Alien to include the phrase "Go Sigourney!" and clicked Save, leading to this screen, which acknowledges the save, followed by a show-all...

CS193i Movies

The CS193i movie database.

Record saved

All 3 records...

Title	Staring	Description	
Alien	Yaphet Kotto, Sigourney Weaver, Harry Dean Stanton	Awesome scary space movie, especially considering it was made in 1979. Go Sigourney!	<input type="button" value="Edit"/> Mon May 13 10:55:52 2002 nick3.stanford.edu
Midnight Run	Yaphet Kotto, Robert Deniro	Great comedy about a bounty hunter.	<input type="button" value="Edit"/> Mon May 13 10:45:51 2002 nick3.stanford.edu
Sense and Sensibility	Emma Thompson, Hugh Grant	Romantic comedy -- best adaption of all the Jane Austen novels.	<input type="button" value="Edit"/> Mon May 13 10:46:03 2002 nick3.stanford.edu

3 records

If you look very carefully, you can see that before the edit, the mod-time for that row was 10:45, but afterwards it was 10:55. The delete button removes the given row from the database and then goes to the home page just like save, but with

the message "Record deleted". The Cancel Edit button goes back to the home page without any special message and without doing anything to the database.

The Prodigal Row Edit

Consider the following scenario: Alice hits edit on row 0 (Alien), and starts typing something. Bob hits edit on row 1 (Midnight Run) and starts typing. Then Alice decides she hates Alien, and so she deletes the Alien record — now Midnight Run is in row 0 and Sense and Sensibility is row 1. What happens when Bob hits save to save his Midnight Run edits? Midnight Run used to be in row 1, but now it's in row 0. An edit will need to remember which record it is editing by something other than the record's row position in the database.

Row ID Numbers

Our solution will be to add an extra "id" column to the database. Each row will get an id number when it is created, and the id number of a row will never change. An edit will remember which row it is editing by id number. When the save happens, the correct row can be found to update by its id number.

The id numbers do not show up in the user interface at all; the user need not be aware of them. However, they must be part of the implementation for the edit/save/delete system to work correctly. With the id numbers in use, multiple people can be editing, adding, and deleting rows on the database at the same time, and it all works so long as multiple people do not operate on one row at the same time.

File Format

Though it does not show up in the user interface, we're going to specify a particular file format for WebDB so we can move files between implementations. Each WebDB file should have a ".db" filename extension. The .db file should be in the same directory as webdb.pl.

Meta Items

The first few lines of the file should contain meta-information about the database, one item per line. There are three required items

"title" is the title string used as the title of the page and repeated in an `<h1>` at the top of the page.

"description" is the HTML text at the top of the home page.

"id" is the next available id number. It will be the id number of the next record added to the database.

Each item should be formatted like an HTTP header line — *name:value-text*. The name will be made of letters and numbers and will be lowercase. The data can be any text other than newline. The items may occur in any order, and there may be more than 3 of them (your code should just ignore items it does not know about). Read the items into a hash table where they can be retrieved by name. The list of meta-items will be ended by a blank line. So the top of the movies.db looks like:

```
title:CS193i Movies
description:The CS193i movie database.
id:5
```

Fields And Data

After the blank line should be a single line with the field names separated by tabs. After that should come the data lines, with the elements separated by tabs. Each element can be any amount of text so long as it does not contain tab or newline. A line may have fewer elements than the overall number of columns in the database, in which case the missing columns are assumed to be from the right hand side. Here's what the whole movies.db looks like...

```
title: CS193i Movies
description: The CS193i movie database.
id: 5

id      Title           Starring      Description
1       Alien            Yaphet..     Awesome..
3       Midnight..      Yaphet..     Great..
4       Sense..         Emma..       Romantic..
```

There used to be rows with id numbers 0 and 2, but at some point they got deleted. The id numbers just increase as rows are added -- we never go back and re-use old id numbers.

Generality

You'll do most of your testing with the movie database. However, your code should work for any database file -- any number of fields, any labels for those fields. See the dog-walking database in the deliverables section for an example of a different database.

Save Error Cases

There are a couple error cases that can come up when an edit or delete operation tries to save back to the database. The problems occur because some other user has deleted the relevant record out from under the edit. Our strategy will be to proceed as best we can, but give the user a little feedback about what happened...

Delete but the record does not exist: the feedback should say "Tried to delete record, but it had already been deleted."

Save but the record does not exist. This can happen if someone else deletes a record you are editing. The save operation should effectively add the record back into the database. It may use its old id number, or it can get a new id. Modify the usual save feedback with a prefix that says ("Someone deleted the record, but your save operation has put it back.")

Other Error Cases

There are a few other miscellaneous error cases where WebDB should return a short error message. There is some code to help with these routine cases in the starter file.

- `ReportError("No databases specified");` If there is no "db" binding, just exit with this error.
- `ReportError("Filename must not contain any funny chars, but got '$dbFile');` Exit if the filename has funny characters in it.
- `ReportError("Could not open '$filePrefix$name' for reading");`
- `ReportError("Could not open '$filePrefix$name' for writing");`
- `if ((-s "$filePrefix$name") > 500 * 1024) { ReportError("File writing error");}` Before opening the file for writing, check its size. If it is already 500k, refuse to write the change. This protects us from someone crashing our machine by filling the disk up through WebDB.

There are other assorted error conditions that can arise — missing bindings, unparseable files, etc. You should do some sort of reporting for those cases to help your own debugging sanity, but we will not require or test it.

Part B

Part B adds a couple advanced features onto the basic WebDB. It's probably best to get the basic WebDB working and tested before adding in the Part B features.

1. Time and IP addr

Each row should store the local server time that it was last written (saved) and the IP addr of the client that did it. This information should all be stored in the "id" column of the database separated by commas. So the contents of the first column of the Alien record is just "1" for part A, but becomes "1,Mon May 13 18:27:02 2002,24.142.60.125" for Part B. If the time and ip are not present in the id column of a particular row, just treat them as the empty string for that row. When that row is saved, add in the time and client ip (not the time that the editing began, the time of the save). The Perl function `localtime()` will give you the current time as a string (Perl gotcha: you must assign `localtime()` like this: `$str = localtime()`, not like this: `my($str) = localtime()`; -- the "my" variant is an array context which causes `localtime()` to return an array of the time components.) The DNS name of the client may be available in the environment variable `$ENV{'REMOTE_HOST'}`. If `REMOTE_HOST` is not defined, use the IP address of the client from the environment variable `$ENV{'REMOTE_ADDR'}`. With the CGI perl module, these values are available from the CGI object like this: `$query->remote_host`). The id, time, and ip can be searched with the search button along with the rest of the row data.

2. Edit Collision

Suppose Alice edits a row. And then Bob starts editing that same row. Then Alice saves. Then Bob saves — collision! Bob's changes will overwrite Alice's changes. Fortunately you know the edit time of every row, so you can detect the collision. When checking out an edit, note the mod-time of the row. When saving back, check if the mod-time of the row is the same as when the edit began. At save time, if the mod-time has changed, then someone else has edited the row. The solution is to return a new edit page with the values drawn from both the values seen in the database and the values that were trying to be saved. The user will have to reconcile the two versions and save the result.

For the following screenshot, I've added a "Rocky" row. Alice and Bob start editing the row (use multiple browser windows). Alice changes the title to "Rocky II". Her save succeeds. Then, Bob tries to change the title to "Rocky III". His save gets kicked back to the edit screen. It puts her version (the saved one) first followed by his...

Netscape: CS193i Movies

Back Forward Reload Home Search Netscape Images Print Security Shop Stop

Location: <http://cgi.stanford.edu/~nick/cgi-bin/webdb.pl?db=b.db&id=6&target=&time=We> What's Related

Page One Plus CNN - Weather - Yahoo! - News WWebster Dicti Headline N

CS193i Movies

The CS193i movie database.

Show All Records Search Add a New Record

Collision!

Tried to save the record, but someone else edited that record in the meantime. Please reconcile the two versions below together and save the result. Each field shows the other person's version followed by your version.

Save This Record Delete This Record Cancel Edit

Title	Rocky II Rocky III
Staring	Sly Stallone Sly Stallone
Description	Better than you would think. Better than you would think.

Last edited at 'Wed May 10 21:10:49 2000' by '24.142.60.125'

The "Staring" and "Description" sections are just duplicates since Alice and Bob left those unchanged. It's Bob's problem to edit this to something sensible and save it (i.e. we just recognize the collision – we make the user actually solve it).

Implementation Ideas

Here are some assorted implementation ideas to help you out...

Run From Command Line

Run your CGI from the command line like this...

```
% webdb.pl "db=a.db&target=foo"          ## print to screen
% webdb.pl "db=a.db&target=foo" | more  ## print by screenfull
% webdb.pl "db=a.db&target=foo" > tmp   ## print to tmp file
```

It's much easier to debug this way — you can see various Perl error messages. Those don't work when using the web server. Only run it through the web server when it appears to work from the command line. Also, use "View Source" in the browser to look at the HTML you are generating.

#!/usr/bin/perl -w

Usually the `-w` option for Perl is an excellent way to let the interpreter find some bugs for you. For HW3, that is mostly still true. However, it is possible in some cases that the `-w` errors will confuse the web server, so you may need to turn off `-w`. You can always use `-w` for your command line testing. Also, the leland webserver may put some extra debugging information at the very top of the HTML page -- ignore it.

Undefined Values

If there is no binding for a name, such as "foo", then `$query->param("foo")` will be `undef`. Using such a value can generate a warning. Test for `undef` with the `defined()` function.

See Your Bindings

As a feature of the CGI module, the following line will print out all the available bindings...

```
print $query->dump();
```

File Prefix

The web server only lets you open files by their full path name, such as `"/afs/ir.stanford.edu/users/n/i/nick/cgi-bin/movies.db"`. However, the `db` binding is expressed as a simple filename `"movies.db"`. Therefore, we use a global `$filePrefix` to encode the `cgi-directory`, and then prepend that to any filename when opening...

```
## change the following to point to your /u/s/username
$filePrefix = /afs/ir.stanford.edu/users/n/i/nick/cgi-bin/";

## to open $fname...
open(F, "$filePrefix$fname");
```

Simple Security

We will disallow filenames that have funny characters in them like "/" to prevent bad guys from using webdb to see random files in the filesystem.. We will also disallow file writing once the file on disk is 500k in size to prevent bad guys from filling up our disk through WebDB. There's code for these two cases in the starter file.

Reading / Writing

When first called, WebDB should read the whole db file into memory. For many operations, that's all that's necessary. For save and delete, WebDB should change the in-memory data structure, open the file a second time for writing, and write out the new version. This will overwrite the old version on disk. A file is opened for writing by prefixing it with a '>'.

For an industrial strength solution, we would need to put in file-locking so that simultaneous CGIs will not interfere with each other. In Unix, this is done with the flock() function. However for HW3 we will ignore this problem. As a practical matter, CGI read-write conflicts will be very rare, since the read-modify-write cycle of the CGI is so fast.

Binding Test Order

1. Notice the edit cases first — starting an edit operation or saving the result of one.
2. If the user clicked the show-all button, do a show-all and ignore the search text.
3. Otherwise, if the search-text is non-empty, do a search, otherwise default to doing a show all.

Array / Split

The database can store the data as an array of lines of text. In the table-printing code, use `@line = split(/\t/, $line)` to make a temporary array out of each tab delimited text line. Then use a foreach to iterate through the elements.

Array Element Delete

The following statement will delete the element at index \$i from an the @db array...

```
splICE(@db, $i, 1);
```

Row Cleanup

Before saving the user's text into the database file, it's vital to remove any tabs or newlines which could confuse the database storage with a substitution such as `"s/\s+//g"` which changes all whitespace chars to simple spaces.

Since we are embedding the database text in an HTML table for output, it's a good idea to remove tags which could confuse the table or the HTML output

scheme. Use this function to tidy up the text before printing it into the HTML table. The text should be saved to the database file in its not-cleaned-up form.

```
sub CleanForTable {
    my($string) = @_ ;
    $string =~ s/<(\/?)(form|table|script|html|body|head|tr|td|input).*?>//ig ;
    return $string ;
}
```

quotemeta

You can use the `quotemeta()` function on a string if you are about to use it as a target in a regular expression, and want to suppress the specialness of characters like "*" and "(" . This is not required; we will not use weird target strings in testing.

New Record vs. Edit Record

These two cases are virtually the same. Use a special id number, such as -1, to denote a record that should be added as new to the database instead of modifying an existing row.

FindByID

Write a `FindByID` subroutine that finds the row number for a record given its id number.

Smaller Font

(Part 2) To make the time and ip appear a little smaller on the home page, nest them in a ` ... ` tags (that's a "-2"). This starts with the user's font pref, and then switches to a smaller version of it. This is the portable way to fiddle with fonts but still work with the user's preferences. Don't specify fixed font sizes.

Time Compares

(Part B) At first I thought I would need to parse the time strings into their parts so I could compare the two times. But actually you can just compare them as strings with "eq" and no parsing — much easier. Just make sure you parse the time out of column 0 cleanly without any spaces or commas to mess up the string compare.

Your `ExtractTimeAndIP` functionality should degrade gracefully if the time and ip are not present in a record -- just return them as the empty string. That way, you can run with your Part-A .db file and it doesn't crash as you add the Part-B functionality using the same .db file.

Sorting

The DB output looks better if the rows are sorted. The following code is included in the starter file, so you can sort the records before output.

```
# Reduce a text record to the part to consider when sorting:
# just the first non-id field, lowercase, with extraneous characters removed.
sub CleanWord {
```

```

my($word) = @_ ;
$word =~ /(.*)\t(.*)\t(\t|\n)/;
$word = $2; # take the 2nd field (the first is the ID)
$word =~ tr/A-Z/a-z/; # change to lowercase
$word =~ s/^(a|an|the)\W//; # remove leading a/an/the
$word =~ s/[^a-z0-9]//g; # remove all weird chars
return $word;
}

# Compare two text records using the cleaned up form of their
# fields.
sub CleanWordCompare {
    return (CleanWord($a) cmp CleanWord($b));
}

## this expression returns the @db array sorted
(sort CleanWordCompare @db)

```

Logistics

Call your solution webdb.pl. Begin a session with a single db=filename.db binding to open a .db file in the same directory as the CGI.

The leland people have CGI servers set up for use by leland account holders. The one for CS193i is called "cgi-courses.stanford.edu". See the "leland cgi" link off the course page for instructions on setting up the leland CGI environment. The leland support page for CGI is <http://www.stanford.edu/leland/cgi/>. Here are the basic setup steps...

1. Create a cgi-bin directory of your home directory. Change its permissions as follows (there's a dot after the "sa"), and **username** is your leland username...

```

mkdir cgi-bin
cd cgi-bin
fs sa . username.cgi rlidwka

```

2. Set the permissions on your .pl to be "700" with the following. This should set execute permission on your file. Use ls -l to check.

```

elaine39:~/cgi-bin> chmod 700 webdb.pl
elaine39:~/cgi-bin> ls -l webdb.pl
-rwx----- 1 nick      38          11352 May 16 10:47 webdb.pl

```

3. Your CGI should be accessible as <http://cgi-courses.stanford.edu/~username/cgi-bin/dumpenv.pl?a=b>

Troubleshooting

What if you try your CGI, but get back an error message...

1. What does the error message say? Some of them are quite cryptic, but sometimes it gives an actual useful description about the problem. We will keep a listing of the common error cases on the course CGI page.
2. Command line? Try running from the command line -- does it produce the right output without any warnings? Most often, this step will reveal the problem.
3. Does the `dumpenv.pl` script work? Are you able to run `...~username/cgi-bin/dumpenv.pl` -- if that doesn't work, then there's actually something wrong with the setup of your CGI account or the CGI server. See the CS193i CGI page.

Dog Walking Database

You can most of your testing with the movie database. However, you must also submit a dog-walking database called "dog.db". The idea is that the database represents the dog walking schedule of three friends. Each day, somebody needs to walk the dog. Use the following structure: one "name" column followed by 7 columns -- one for each day of the week -- mon, tue, wed, thu, fri, sat, sun. Three rows "Alice", "Bob" and "Carl". Put an "x" in a cell to represent that the person is walking the dog that day, and otherwise leave the cell blank. Put in random data so the dog is getting walked every day. The point is to demonstrate that your code is general enough to deal with databases with different numbers of columns and different labels.

Deliverables

Turn in your `cgi-bin` directory with your `webdb.pl` in it and a Readme as usual. We should be able to run your `cgi` over the web and from the command line. There should be two data files in your `cgi-bin` directory -- `movies.db` and `dog.db`. The databases should have the structure of the one used in this handout, although you can edit the movie database to include data of your choice.