Homework Assignment 5

15-415 Database Applications
Carnegie Mellon University

March 28, 2003
Due: April 16, 2003 (10pm)

1 Introduction

By now you have worked extensively with Postgres and have become experts! This assignment will give you a glimpse in application development. An increasingly popular way to build interfaces to a database application is though the web. This has several obvious advantages: most important, any user with a web browser can access the application from anywhere[1]. Furthermore, almost all public websites (at least all interesting ones) have dynamic content and typically deploy some form of a DBMS somewhere along the line.

There are several options when it comes to choosing a way to implement a web frontend to a database backend; anything from plain old CGI binaries to unified database and web servers is out there! In this assignment, you will be using PHP. You can find more information on it at `http://www.php.net/`. In "prehistoric" times, during its early stages, PHP used to stand for "Pretty Home Pages" but it has come a very long way since then! Nowadays it is a popular and fairly efficient scripting language, dubbed by some as the opensource alternative to ColdFusion. If you have even minimal experience in other scripting languages (such as Perl, Python, etc), PHP should be very easy for you to pick up. Furthermore, we are giving you all the PHP code and HTML templates for the webpages; you only need to add the code that talks to the database and gets the results back.

The application you will be developing for this assignment is a simplified interface for University Center movies. The database contains movie information, showtimes and also allows users to register and post movie ratings and comments.

2 Installing the files

The distribution for this assignment can be found in `/usr0/dbcourse/hw5/hw5-pkg.tar.gz`. You should untar this into your `public_html` directory; everything will be placed in a subdirectory called `ucmovies`. Note that both your home and `public_html` directories should have at least execute permissions for others, so that the web server can read your files. You can do this with something like

```
$ chmod go+x $HOME; chmod go+x $HOME/public_html
```

It might be a good idea to give no more than execute permissions to others on your home directory (so they cannot list its contents).

[1] At least in theory; in practice, making sure that is the case is often a non-trivial task that involves dealing with "standards" and incompatibilities between browsers.

3 Database

In this assignment, you will have to run your own database server. Make sure to use the correct version of PostgreSQL (version 7.1.3, installed in `/usr/bin`). By now you know how to create a database. If you want to create a separate user, you can do it with something like

```
CREATE USER ucmovies WITH PASSWORD 'ucmovies';
```

from `psql`. Look at the SQL command reference for `CREATE USER`, `ALTER USER` and `GRANT` if you need more information. Alternatively you can use the command line utilities[2], e.g.,

```
$ createuser -d -A -P ucmovies
$ createdb -U ucmovies ucmovies
```

We are providing you with the schema in `db/schema.sql`, which contains the DDL statements necessary to create all the database tables for this assignment.

3.1 Database schema

The DDL statements to create all database tables and indices are contained in `db/schema.sql`. Assuming your database is called `ucmovies` and you are connecting to it as user `ucmovies`, you can create all tables using something like

```
$ psql -U ucmovies ucmovies < db/schema.sql
```

Please study this file carefully and particularly the primary and foreign keys and value constraints. We are using a fairly simple schema for this assignment; there are several simplifications you would probably want to do things differently for a full-blown application (say, something resembling IMDB or Yahoo! Movies), but this should be sufficient for this application.

The file `db/movies.sql` contains some initial data. After you create all the tables, you can use this to populate them with values. Again, you may do this with something like

```
$ psql -U ucmovies ucmovies < db/movies.sql
```

Note that the CMU activities board, as of this writing, has not yet published showtimes for future movies, so the showtimes contained are just guesses.

A final note about testing: you can use `db/schema.sql` as above to clear the database and recreate it with empty tables, then use `db/movies.sql` to repopulate them. This is a good idea, to clear all "junk" values that you ay insert while testing your application.

3.2 Sequence objects

Notice that some fields are declared as `SERIAL`. This is Postgres syntactic sugar; see the documentation on data types for details. In brief, a `SERIAL` field is just an integer, along with an implicitly declared sequence object that is used to provide a default value for that field. The sequence object is essentially a counter that can be atomically incremented and is used to guarantee you get a unique numerical identifier. The default initial value of any sequence is 1. See the SQL command reference for `CREATE SEQUENCE` if you want all the details; however, you will only need the `currval()` (and possibly `nextval()`) functions. See `db/movies.sql` for an example of using these; you should not need to do anything more with sequences.

[2] Make sure your `PGDATA` and `PGPORT` environment variables are properly set!

3.3 Connecting to the database server

The code necessary to connect to the database server is contained in the files `dbconst.php` and `dbconnect.php`. Both of these are (eventually) included in `db.php` (which is currently empty beyond that). Since we will need to test your code against our own database, you must keep these files. You are free to modify them for your database, but we will be using our own settings when we are testing your code. What this boils down to is: use the provided `db_connect()` function to get a database connection object and you should be fine!

Note that the current settings in `dbconnect.php` provide a hostname in the connection string. In order for this to work, your database server has to be listening for TCP connections. So, when starting postmaster, make sure you pass it the `-i` argument (or, equivalently, modify `postgresql.conf` in your data directory and set `tcpip_socket=true`).

Please modify `dbconst.php` according to the port number that has been assigned to you (the one in the `PGPORT` environment variable from assignment 0). Also, if you use different database and user names, make sure to change these as well.

3.4 PHP interface

The PHP provides a rich set of functions for several tasks; among those are functions to communicate with Postgres. For details, see `http://www.php.net/manual/en/ref.pgsql.php`. You will at least need to use `pg_exec()` and `pg_fetch_row()` (or equivalent functions), as well as `pg_numrows()` and `pg_cmdtuples()`. Please note that the version of PHP installed on the instructional machines is older than 4.2.0, so you should use the older naming conventions (see Table 2 in the above URL).

One final note on error reporting: PHP by default prints out warning messages in the HTML output; these include messages for key constraint violations. You do not have to turn these off, but if you find them annoying you can use the `error_reporting()` function. For example, you can call `error_reporting(E_ALL ^ (E_NOTICE | E_WARNING))`.

4 Web frontend

We have provided you with all the necessary code to generate HTML output. You only need to "fill in the blanks" and make the scripts actually communicate with the database to produce the output. In other words, you only need to worry about the *application logic* and not the *presentation*. PHP does not provide an explicit mechanism to separate the two (like, for example, Cocoon, for those who may have heard of it) and it is up to the programmer to organize the application appropriately.

There are comments in the code that should guide you in modifying the appropriate places in the code you have. **It is important that you do not change the HTML layout; we will be using an auto-grader that will parse your HTML output and may fail if you change the structure of your HTML output.** In most cases, your code just has to fill in the apropriate values of PHP variables already declared (with dummy, constant values as examples) in our code. In a few occasions (such as printing out user comments or search results) you need to produce some output per variable. **We have already declared functions that produce the appropriate HTML output. You should call these in a loop that iterates over result rows, at the locations marked in the code.**

All PHP files already include `db.php`. This is currently empty (except including the constants and functions required to connect to the database, as described in subsection 3.3). You can place any application logic functions in there. Although you do not have to write that much code, feel free to use more files if you want; just make sure to include them as necessary.

Also, all pages use the global variable `$errmsg` to report errors. This is set to an empty string by default. If an error occurs while you are processing the request, set that to an appropriate error message. The rest of the code should take care of reporting it to the user.

In the following sections we briefly describe the functionality you should implement for each page. A sample implementation will be running on one of the instructional machines (the URL will be posted only on the BBoard). You will have full access to it, but only from CMU IP addresses. Please keep in mind that its database may be periodically purged (so do not assume that usernames, etc that you register will be there permanently).

4.1 Main page

The main page should list the movies that are playing during the current week. For the purposes of this application, the week starts on Monday and ends on Sunday (inclusive); movies at the University Center are typically shown Wednesday thru Sunday. You will need to add code into `main.php`.

Your code should use `print_showtime()` to display the results. Although admittedly not the best user interfacce design, each movie may be listed several times if there are multiple shows scheduled. You should order the results by date/time.

In order to facilitate testing, the current date can be overriden (with a POST or GET request parameter we provide; if that is missing, then the actual current date is used). Therefore, *do not* use `CURRENT_DATE` (or equivalent functions); instead, you should use the `$curdate` variable in your PHP code. You may also want to use the variables `$lastMonday` and `$nextSunday`, which contain the date of the first and last day of the current week (based on `$curdate`). All dates are in the standard ISO format, i.e., `YYYY-MM-DD`. If you happen to need any fancier operations on dates, see `http://www.php.net/manual/en/ref.datetime.php` or the date/time functions in the PostgreSQL manuals (depending on whether you want to do things in PHP or in your SQL queries).

4.2 Movie information

This page displays information about a single movie, including an average user rating and user comments. You should add code into `show_movie.php`.

In particular, you will need to write queries to fetch the basic movie information (stored into `$title`, `$directors`, `$actors` and `$links`) and user ratings (in `$avg_rating` and `$num_votes`). Furthermore, you will need to fetch all user comments related to this movie; as explained, use `print_comment()` (one call per comment in the database) to produce the necessary HTML output.

4.3 Search

The application provides some rudimentary search capabilities; in particular, users may search for movies based on either show date or movie title. You will need to modify `search_date.php` and `search_title.php`. In both pages you should use `print_movie()` at the appropriate place to produce the necessary HTML for every match. Also, when searching by title, you should do a case-insensitive substring search; i.e., the expression in your SQL `WHERE` clause should look something like `m_title ILIKE '%${title}%'` (see ANSI regular expressions in the PostgreSQL manual for an explanation of this).

4.4 User registration

The application allows users to register. This allows them to post movie ratings and comments. You need to modify `register_user.php` for user registration. Your code should check that the passwords match. If they do, then it should insert the appropriate tuple in the database. Usernames have to be unique (see also the table definitions). Numerical user IDs are "automatically" assigned using the relevant sequence object.

Digression: Session management In order to make things simpler, both for you during development, as well as for us during testing, the application you're developing does not support sessions. In other words, there is *no* state associated with each user agent (i.e., browser) that connects to the web server; this would be

necessary if the site supported, for example, user logins. Instead, since a username and password are only required when submitting a rating or a comment, users are asked to type it in as necessary each time.

If you are curious about PHP support for sessions, you can look at `http://www.php.net/manual/en/ref.session.php`. PHP has support for non-persistent sessions. Session data is stored in memory, so it would be lost in the event of a server crash. Selected variables from a script's namespace can be saved and restored depending on a unique session ID. All this is handled by PHP for you. However, in this assignment you do *not* need to do any of this.

4.5 Comments and ratings

Registered users can post movie ratings and comments. You need to add code in `comment.php`, `comment_add.php`, `rating.php` and `rating_add.php`. If you look at the schema, you will notice that each user may post only one comment or rating per movie. Furthermore, comments may be modified[3]; the code for this should go in `comment_show.php` and `comment_edit.php`.

In particular, in both `comment.php` and `rating.php` all you need to do is retrieve the movie title (into `$title`) based on its numerical, unique ID. Of course, if the movie does not exist, you should return an error. Next, in both `comment_add.php` and `rating_add.php` you have to authenticate the user (i.e., check that the username/password combination exists in the database) and, if that is the case, insert the appropriate tuples in the database.

`Make sure you escape the comment text in your SQL query string`; for example, if the comment text contains quotes, these need to be escaped (*one* way to do this is with the PHP function `addslashes()`). This is true in general of all arguments you use in SQL query strings, but be particularly careful with the comment text field.

Finally, in `comment_show.php` you need to execute queries to fill in `$title` (as before) and `$userid` (based on the numerical `$uid`) as well as the comment data (`$ctitle` and `$comment`). In `comment_edit.php` you have to authenticate the user, make sure the ID of the authenticated user matches `$uid` and then do the necessary database update.

4.6 Management interface

The application provides a rudimentary management interface that allows insertion of movies, showtimes and movie URLs. Once again, in order to simplify things, there is no authentication for the management interface. This would be easy to add using either sessions (see subsection 4.4) or basic HTTP authorization (e.g., see `http://www.php.net/manual/en/features.http-auth.php` if you are curious). However, you should *not* do any of this.

Add movie The site manager can add new movies into the database. Your code for this goes into `manage_movie_add.php`. The variable `$title` contains the title of the new movie. The variables `$directors` and `$authors` are arrays and you need to insert one tuple per array element into the `directors` and `actors` tables, respectively. Essentially your queries should look like the ones in `db/movies.sql`; see the SQL statements there for an example of how to use the necessary sequence objects for the new movie ID.

However, there is one very important "detail." The entire operation **should execute as one (atomic) transaction**! In other words, if insertion of a movie fails for any reason, then the sequence should *not* be incremented and no `directors` or `actors` tuples should be inserted. See the SQL command reference for `BEGIN`, `COMMIT` and `ROLLBACK`. Furthermore, initiating an explicit transaction is also more efficient (as explained in the PostgreSQL documentation), although this is not the primary concern here.

A final note about `manage_movie.php`: this uses some dynamic HTML we have written to add more 'Director' and 'Cast' fields in the HTML form. It has been tested and works on Mozilla as well as Internet

[3]Ratings cannot be modified; doing this isn't conceptually any different from modifying comments. Feel free to add the functionality if you have the urge; however, we won't be checking for this.

explorer (version 5 or later). It is known *not* to work on Netscape Navigator (does anyone still use this?) and Opera. Feel free to modify it as necessary during testing, if your browser has problems. Our auto-grader will not be accessing this page, only `manage_movie_add.php`. In fact, it will only be accessing the pages we have explicitly told you to modify; you may do as you wish with the remaining pages.

Add link and showtime The site manager can also add showtimes and movie URLs. Your code should go in `manage_link.php`, `manage_link_add.php`, `manage_showtime.php` and `manage_showtime_add.php`. In particular, for both `manage_link.php` and `manage_showtime.php` you need to fetch a list of all movies in the database into the `$movies` associative array (see the dummy values we provide as an example). For the other two pages, you should simply do the appropriate database insertions.

5 Testing your website

We have put up a online testing site for you; again, the URL will be posted on the BBoard only. Please note that we may use a few more tests during grading (e.g., to check that you properly escape SQL query strings, etc) and/or a different dataset, but if your code passes these tests, you should be fine. The tests assume that your database is loaded with the sample data we have given you and *only* with that data. Therefore, **before testing you should purge and reload the database**[4], using `schema.sql` and then `movies.sql`. After you have done that, go to the testing website and submit the base URL of your site. This is where all your PHP files reside; for example, if your main page is accessible from `http://melpo.datanium.cs.cmu.edu/~spapadim/ucmovies/main.php`, the base URL is `http://melpo.datanium.cs.cmu.edu/~spapadim/ucmovies/`. **You have to make sure that your HTML output can be properly parsed by our testing scripts!** If you have used our HTML templates and output generating functions, you should have no problem. You can also compare your HTML output with that of our sample implementation, if necessary.

6 What to turn in

You will do this assignment individually. You should create a tarball of your `ucmovies` directory, e.g.,

```
$ cd $HOME/public_html
$ tar -zcvf hw5-solution-andrewid.tar.gz ucmovies
```

Do not forget to include the file `INFO` with your Andrew ID.

Given the recent problems with cross-realm authentication, you should email this file to the TAs. The lead TA for this assignment is Spiros Papadimitriou (`spapadim@cs.cmu.edu`). Do not assume that your files have been reecived unless you get back a confirmation. If necessary, please email one of the other two TAs. Your submission time will be the time that the email was *received* (based on the delivery path headers). Please do not submit the last minute!

[4] This means you cannot use our sample implementation, since its database will not necessarily be "clean."