Assignment 6: Web DB (version 2)
Scott D. Anderson
Due: Wednesday, May 9th

For this assignment, we will jointly build an Oracle database and individually write web forms and CGI/DBI scripts to access the data.

My idea for this assignment was inspired by the desire to have an online store, like Amazon.com and other e-businesses. The trouble is, we have nothing to sell. Then I remembered that my sister wanted to have a "baby gift registry," where she could list baby items that she needed, and friends and family could come and see what she needed and could volunteer to get particular items. They could "check off" items from the list, so as to avoid duplication. I think you've all seen that many retail stores (such as Macy's, Neiman-Marcus and even, believe it or not, Home Depot) allow people to register for gifts carried by that store.

1 The Database

I have created a user (cs349/buffy) in the Oracle database. This schema owns two tables:

Accts(aid: *number*,
email: *string*,
fullname: *string*,
password: *string*)

Gifts(aid: *number*,
gid *number*,
gift *string*,
needed *number*,
given *number*)

The "aid" field is a foreign key in the Gifts table, so that all the matching entries are gifts that the person in the Accts table has registered for. (Givers don't need to register.) The "gid" field uniquely identifies a gift; it's useful for interacting with the forms, but shouldn't be displayed to the user (just because it's ugly).

2 The Basic Idea

People will come to the site when they are getting married, having a child, or any other sort of event where others will give gifts. They can create an account for themselves and enter a list of gifts they would like. In most cases, they just want one of the gift (toaster) and other times they want a particular number (8 dessert plates). That's why the second table has columns for the number needed and the number given. (People coming to the site don't have to volunteer to buy all 8 dessert plates; they might sign up for only 4 or even 1 if the plates are really expensive. My mom once looked over a gift registry where one silver spoon was fifty bucks!)

3 To Do

Joint Work

So that we have some data to work with, I want each student to create an account in the database (this can be done before you have a working script, just by using `sqlplus`) and create a gift list. The account should use your real name and such; the gifts can be entirely fantasized.

To Build

Each student (or collaboration) must build this site, which must:

- Allow a user to create, edit and delete an account.
- Allow a registered user to create a gift list and add items to it.
- Passwords should be honored, so only a user who knows the password can modify an account or a gift list.
- Allow a visitor to display the gift list for a particular person. They have to know the name of the person they are shopping for; they can't just be browsing over stuff on strangers lists. (You may assume for the sake of this assignment that the name is unique.)
- Allow a visitor to display just the remaining items on the person's list.
- Allow a visitor to volunteer to get gifts on someone's list: that will increase the "given" field. This should be possible from either of the previous displays.

Coordination and Security

Since we all have to put our scripts in the `/home/httpd/cgi-bin` directory, we have to be careful not to be naming them the same thing. Therefore, please include your username in the name of your scripts, so Wendy Wellesley would name hers `wwellesl-foo`, `wwellesl-bar`, and so forth. Since your web pages will go in your own `public_html` directory, you can name those as you please.

Finally, a point I must reluctantly make, concerns the privacy of your work. Your work on this assignment must be your own, but testing your solution will require scripts and web pages to be globally readable. Thus, it's possible for someone to copy your work. I don't think anyone in this class would do that, but it only takes one student, overstressed and behind in her work, to spoil things. Therefore, I encourage you to do as much of the development as you reasonably can by running scripts from the command line and viewing pages locally (using the `file:` protocol instead of `http:`). I don't think there's any reason to get paranoid about it, but just be prudent.

Once you've done enough locally and you want to test your script using the server, you have to "upload" your script into the CGI directory and make it world readable and executable. Here's a little script you can use to do this:

```
#!/bin/bash
cp $1 /home/httpd/cgi-bin/
chmod a+rx /home/httpd/cgi-bin/$1
```

The $1 variable stands for the first command-line argument. I called my version of this "upload," so all I have to do is say `uploadfoo` and `foo` will be uploaded and made executable. Very convenient.

When you're not actively working on the assignment, leave the files unreadable by others. It's easy to do:

`chmod og-r wwellesl-*`

when you're done working and

`chmod og+r wwellesl-*`

when you begin work. You'll only be able to change the permissions on files you own, anyhow, so you could even use a more general pattern. You could also put this in a script, if you want. I called mine `lock` and `unlock`.

Note that CGI programs may compromise a computer's security, so most system administrators do not allow ordinary users to install scripts in `cgi-bin`. I'm allowing that for the purposes of this class, but I want you to be aware of this aspect of the technology.

Additional Work

None of the following need to be implemented, but I would like a paragraph or so of text saying what would be necessary to implement the following improvements to the site:

- People forget their passwords: how would you make it possible for the site to handle that in an automatic way?
- How could the site allow account holders to change their minds, removing items from the gift registry?
- How could the site automatically get rid of things that were out of date?
- How could the site keep track of who volunteered to get each gift? Can you think of any reason to do so?
- Discuss some other feature you think should be added and why. If possible (you may not know how), explain how it could be added.

4 Things to Know

Online SQL Manual

I've found the following web site to be very helpful:

`http://info-it.umsystem.edu/oradocs/doc/server/doc/SQL73/ch4a.htm`

Sequences

How can we generate unique AID and GID numbers? Oracle has the idea of a "sequence," which is a datatype that is like a ticket roll in a bakery: you grab one off the sequence and the next number pops up. This is done atomically, so we don't have to worry about scripts running simultaneously. (Consider the alternative: your script does a query to find the largest AID number, adds one to it, and inserts a new row using that AID number. Now consider having two such scripts run simultaneously.)

I've created sequences called `cs349.acct_seq` and `cs349.gift_seq`. To use them:

```
sqlplus> insert into cs349.Accts
values (cs349.acct_seq.nextval,'sue@aol.com','sue','silly');
```

You can also use `acct_seq.currval`, but consider the "race conditions" mentioned earlier. It's better to just find out the AID number, if you need it, by looking up the email address.

Outer Joins

A normal join finds all those rows (from the conceptual cross-product) where a key field matches. For example, if we join the Accts and Gifts tables, we find all the people and their gifts. But what about people who have not yet registered for any gifts? They get left out! A solution is called an "outer join." See page 149 of your book.

However, heaven forbid that Oracle should follow the 1992 SQL standard. Oracle has a different syntax for an outer join. To get a "left outer join" of the Accts and Gifts tables, you can do:

```
select * from Accts A, gifts G where A.aid=G.aid(+)
```

The `(+)` means that the `G.aid` is "optional" in the sense that if it's there it has to match, but if there are no matching rows from the Gifts table, the account will appear once, with nulls for the columns from the Gifts table.

This outer join query can be a convenient way to "dump" the database to see what you have so far.

5 Advice/Notes

- Throw the following incantation into your scripts, since the "nobody" user isn't getting the correct value for the `ORACLE_HOME` environment variable:

  ```
  $ENV{"ORACLE_HOME"} = "/home/oracle/OraHome1";
  ```

- Get scripts working in the shell first, so you can put in print statements and the like.

- Look at the error log (`/var/log/http/error_log`) if you get a script error. The errors are appended to the end of the file. You can quickly see the end of a file using the `tail` command. By default, it prints the last ten lines of a file. Check the man page to learn more.

  ```
  tail /var/log/http/error_log
  ```

- Print out SQL statements that you construct before you `prepare` or `do` them, since most errors are caused by syntax errors in your SQL statement, and it's easier to figure those out if you have a printout of the actual statement. I do something like this:

  ```
  $stmt = "insert into cs349.gifts values (" .
      $dbh->quote($aid).",".
        $dbh->quote($q->param("gift")).",".
          $dbh->quote($q->param("need")).",0)";
  print "statement is $stmt\n" if $debug;
  $dbh->do($stmt);
  ```

- If you know that a query will result in only one row, or you only care about the first row, you can skip the four-step ritual and do it in one step as follows:

  ```
  my @first = $dbh->selectrow_array($stmt);
  ```

- Print statements that precede the http headers are deadly once the script "goes live" on the web, because the first thing that the server must get is sequence of headers. So, print statements that are harmless in the shell will yield a 500 error on the web. That includes warning messages from Perl (*sigh*). Your best bet is just to make sure that the headers go out first. For example, you might put something like the following near the top of your script:

  ```
  print $q->header() if $debug;   # So print statements are after the headers
  ```

  This won't hurt if you're working in the shell, and at least your print statements should end up somewhere on your web page when you run the CGI script from the server.

- For parameters that aren't defined, `q->param("pname")` returns "undef," a Perl object that means undefined. You can test for it like:

  ```
  if( defined($q->param("debug")) && $q->param("debug") eq "on") {
     $debug = 1;
  }
  ```

- If there are nulls in the data returned from a query, Perl represents them as "undefs" in the array. There is code in the `list-emps-html` script that shows one way to deal with them.

- You can use hidden inputs to define parameters that you can't define in other ways.

- You can also tack inputs onto the URL, as follows:

  ```
  <a
  href="http://kefalonia.wellesley.edu/cgi-bin/wwellesl-script?parm1=val1&parm2=val2">
  click here
  </a>
  ```

  The downside of this is that your script may have to be adjusted. Contact me if you want to do this.

- If you get desperate about having print statements in a script run by the server, you can do the following:

  ```
  open LOG,">/tmp/log$$";
  chmod 0644, "/tmp/log$$";
  print LOG "Here's an error message\n";
  ```

  Note that $$ is a script variable that returns the current process ID, so using it is a common trick to give you a unique filename each time you run this script. You can figure out which one is your most recent by doing an `ls-lt` on the `/tmp` directory; the `-t` sorts by time.

- If you're tempted to write out an entire page that is fixed (such as an error message saying that the account or password is wrong), you can just redirect the browser to a different page, as follows:

  ```
  $q->redirect("bad-account-or-password.html");
  ```

  Of course, you have to write that page, but that's easy, compared to writing Perl.

- If you write out a form input using CGI.pm, it will take the "value" attribute from the `param` hash. For example, suppose we have the following code:

```
$q->param( email => 'fred@aol.com' );   # set email parameter
print $q->textfield( { -name => 'email', -value => 'wilma@yahoo.com' } );
```

  The code will cause the following to be printed:

```
<INPUT TYPE="text" NAME="email" VALUE="fred@aol.com">
```

  If you want it to have Wilma's email address, you either have to modify the parameter or add a `-override` key with a true value:

```
$q->param( name => 'fred' );
print $q->textfield( { -name => 'name', -value => 'wilma', -override => 1 } );
```

  The preceding yields

```
<INPUT TYPE="text" NAME="name" VALUE="wilma">
```

- Here's a useful subroutine to dump out the current table data:

```
sub print_tables {
   print "<table border=1>\n";
   $sth = $dbh->prepare
       ("select * from cs349.Accts A, cs349.gifts G where A.aid=G.aid(+)");
   $sth->execute();
   while ( my @row = $sth->fetchrow_array() ) {
     undef2null(@row);
     print "<tr><td>",join("</td><td>",@row),"</td></tr>\n";
   }
   print "</table>\n";
}
```

  Or, you may find the following code more intuitive:

```
sub print_tables {
  print "<table border=1>\n";
  $sth = $dbh->prepare
     ("select * from cs349.Accts A, cs349.gifts G where A.aid=G.aid(+)");
  $sth->execute();
  while ( my @row = $sth->fetchrow_array() ) {
    undef2null(@row);
    print $q->Tr($q->td(\@row));
  }
  print "</table>\n";
}
```

Setup

Here's how the tables were really set up:

```
connect creator/creator;

drop user cs349 cascade;
-- drop table cs349.gifts;
-- drop table cs349.accts;
-- drop sequence cs349.acct_seq
-- drop sequence cs349.gift_seq

create user cs349
identified by buffy
default tablespace users
temporary tablespace temp
quota 50M on users
profile default;

grant connect to cs349;

-- This lets us do things like acct_seq.nextval in an insertion

create sequence cs349.acct_seq;
create sequence cs349.gift_seq;

create table cs349.Accts(
       aid number not null,
       email varchar(40) not null,
       fullname varchar(50) not null,
       password varchar(20) not null,
primary key (aid),
unique (email));

create table cs349.Gifts(
       aid number,
       gid number not null,
       gift varchar(25),
       needed number,
       given  number,
primary key (aid,gift),
foreign key (aid) references cs349.Accts(aid)
on delete cascade);
```

List Employees

Here's an example script that queries the database and writes out a web page listing all the employees. It's in `/home/http/cgi-bin/list-emps-html`:

```
#!/usr/bin/perl -w

# From Programming the Perl DBI, chapter 4.  Connects to an Oracle
# database, fetches and prints all the users, and disconnects.  Uses the
# default automatic error checking.

# This version writes out html, so we can use it as a page-building script.

$ENV{"ORACLE_HOME"} = "/home/oracle/OraHome1/";

use strict;
use DBI;
use CGI;

my $q = new CGI;

my $dbh = DBI->connect( "dbi:Oracle:CS349.WORLD", "anderson", "scott" );

my $sth = $dbh->prepare( " select * from scott.emp " );

$sth->execute();

print $q->header ( -type => "text/html", -expires => "+30m" ),
  $q->start_html( -title => "Employees", -bgcolor => "white" );
print "<table>\n";
while( my @row = $sth->fetchrow_array() ) {
  undef2null(@row);
  #my $line = join(" ",@row);
  my $line = "<tr><td>".join("</td><td>",@row)."</td></tr>";
  print $line,"\n";
}
print "</table>\n";
print $q->end_html;

$dbh->disconnect();

exit;

sub undef2null {
  foreach ( @_ ) {
    if( !defined($_) ) {
      $_ = "null";
    }
  }
}
```