TEST 3 REVIEW

DATABASE REVIEW

- Four basic things you can do to an existing table CRUD
  - Create: `INSERT` statement
  - Retrieve: `SELECT` statement
  - Update: `UPDATE` statement
  - Destroy: `DELETE` statement
- You can use clauses to narrow/format your result sets or the records to retrieve/modify

SELECT SQL Statements

- SELECT statements work on existing records in a database
- Example:

```
SELECT * FROM movies;
```

- For readability, should be more specific

```
SELECT movie_num, title, actor,
year FROM movies;
```

SELECT Statement Aliases

- For joining multiple tables in a single query it is sometimes easier (clearer) to give table names aliases:
- E.g.:

```
SELECT movies.title, movies.year, actors.name
FROM movies, actors
WHERE movies.actor = actors.id
ORDER BY movies.year ASC
```

- Could be:

```
SELECT m.title, m.year, a.name
FROM movies m, actors a
WHERE m.actor = a.id
ORDER BY m.year ASC
```

- In this case, a and m become aliases for the tables actors and movies respectively

SELECT Statement Qualifiers

- You can narrow down your results by using various qualifiers
  - `WHERE` *`column_name`* `LOGIC_OPERATOR` *`value`*
- The logic operators are the same as programming:
  - `<>, >=, =, <=, <, >`
    - N.B. single equal sign is the logic comparator
    - And <> to check inequaliy (not the !=)

Insert SQL Statements

- INSERT statements create records in a database
- Example:

```
INSERT INTO movies VALUES(21, 'Casino
  Royale', 'Daniel Craig', 2006);
```

- Should be more specific

```
INSERT INTO movies (movie_num, title, actor,
year) VALUES(21, 'Casino Royale', 'Daniel
Craig', 2006);
```

- N.B. strings are delimited by single quotes ('), not double quotes

SELECT Statement Sorting

- You can sort your select result set with the "ORDER BY" clause
  - ORDER BY *column_name directional_qualifier*
- The directional qualifier can be:
  - ASC for ascending (default)

Update SQL Statements

- UPDATE statements modify existing records in a database, and uses same clauses as SELECT statements
- Example:

```
UPDATE movies SET year = 2003
  WHERE title = 'Die Another Day';
```

- Be aware you can update multiple records with one UPDATE command (if not careful)

DELETE SQL Statements

- DELETE statements remove existing records in a database, and uses same clauses as SELECT statements
- Example:

```
DELETE FROM movies WHERE title = 'Die Another Day'
```

- Be aware you can DELETE multiple records with one DELETE command (if not careful)
- e.g.

```
DELETE FROM movies;
```

  - Removes all records from the movies table (but does not remove the table, you must use a DROP statement to delete)

Creating a New PostGreSQL DB

- Now that a database exists for you, and you have been given ownership, when you log onto the server with PuTTy, if you type: `psql userid_db`
  - You will be prompted for your password
  - After entering it, the system will take you into your database, where you can perform SQL commands (prompt will be => instead of the # or $ OS prompt)
- To change your password at the sql prompt => type:

```
ALTER USER your_user_id WITH ENCRYPTED PASSWORD 'your_new_password';
```

Allowing Data Access to Users

- When a *.sql script is run by a PostgreSQL user, the user owns the table and the data in the table
- If a different user needs to access the data they must be given permission explicitly
  - E.g. in this class your instructor needs to see your tables/data
- The command to given certain access levels on your table to anpther user is: `GRANT`

SQL Comments

- SQL scripts support comments in two formats:
  - Single line comments start with -- (two hyphens)

```
-- this is a single line SQL comment
```

  - Multi-line comments are the same as c-style:

```
/*
    This is a  multi-line
    SQL comment
*/
```

Removing a Database

- If you create a database (misnamed or unwanted), to remove it type:

```
dropdb nogood_db
```

  where nogood_db is the unused/unwanted database
- **NOTE: do NOT execute this command on your `lastnamefirstinitial_db` database, you are able to remove your db, but your user does not have permission to create a new one**

GRANTing Privileges

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

- For this course add the following line in your SQL build script

```
GRANT ALL ON table_name TO faculty;
```

- Example:

```
DROP TABLE actors;
CREATE TABLE actors(
      id INTEGER,
      name CHAR(20)
);
GRANT ALL ON actors TO faculty;
INSERT INTO actors(id, name) VALUES…
```

Creating a New PostGreSQL DB

- For this course (WEBD2201) you will not need to create your own database (it is created for you)
- From the command line the command following command was executed:

```
createdb userid_db
```

  - Where userid is your user id (i.e. pufferd)
  - **When the database was successfully created the system displays a "`CREATE DATABASE`" message**
- Alternately, you can run:

```
CREATE DATABASE userid_db
```

from inside a database

Running SQL Script from the command line

- For large scale applications, it is advantageous to set up "build" scripts that drop/create databases as required
- Easier to implement changes than doing everything manually
- At the command prompt type:

```
psql -d userid_db -f script_file.sql
```

  - Where `userid_db` is the database to be modified and `script_file.sql` is the SQL file with the commands to be executed (must be co-located in the current directory, or else fully qualify the address)

PostGreSQL Meta-commands

- There are several commands that are defined for PostGreSQL that allow users some short-cuts to administer databases or runs scripts
- Some common and useful ones that can be executed, at the PostGreSQL => prompt type:
  - \q will quit or exit the database, takes you back to the OS
  - \d will "dump" (quickly preview) the database's content
  - \d *table_name* will dump a specific table's info
  - \i *db_script.sql* allows you to run a script from outside the db prompt
    NOTE: this will default go to the directory the user was in when they connected to the database, to use a file from a different directory the file path must be fully qualified:
    \i */var/www/users/webd2201/user_id/sql/db_script.sql*

PHP Database commands

PHP PostGreSQL Database Commands

pg_connect()

- Performs the all important step of connecting to a database
- Returns a PostGreSQL connection object that is used by subsequent functions
- Example

```
$conn = pg_connect("host=127.0.0.1 dbname=DB_NAME user=USERID password=PASSWORD");
```

  - 127.0.0.1refers to the computer running the PHP page
    - NOTE: localhost should work as well, but the opentech server does not like this
  - dbname is the database you are connecting to (in our case userid_db)
  - user is the postGreSQL user you are connecting as
  - password is the user's PostGreSQL password
- `$conn` is a connection object that will be used below (does NOT have to be called connection)

pg_fetch_result()

- Used to retrieve any information contained in a result set from a SELECT statement
- Requires a result set argument, and a reference to a record and a column
- Example:

```
$title = pg_fetch_result($results, 0, 0);//$results was
                                          //created by
                                          //the pg_query()
$year = pg_fetch_result($results, 0, 1);
```

- This is putting the first record's (0th) first column (0th) into a variable named $title, and the first record's second column (1st) into a variable named $year

PHP Functions that Work With PostGreSQL

- PHP has the ability to connect to many types of Database software
- most popularly MySQL, but also PostGreSQL (both shareware)
- This course is using PostGreSQL as it is a more powerful tool than MySQL (closer to Oracle/SQL Server than MS Access)
- The Good news: very limited number of PHP functions are required
- Note: all of the functions shown have to exist in PHP `<?php ?>` tags

pg_query()

- Used to run SQL statements against your database
- N.B. this includes INSERT, UPDATE and DELETE statements (not just SELECT queries)
- Will return a result set if a SELECT statement is given
- Example:

```
$sql = "SELECT title, year FROM movies";
$results = pg_query($conn, $sql); //$conn was
                                  //created with
                                  //the pg_connect()
```

- `$results` is a result set that contains the title and year of any records in the movies table

pg_fetch_result() (cont'd)

- Another syntax that you can use is the database table's field name versus the index of the field in the SQL SELECT statement. Example:

```
$title = pg_fetch_result($results, 0, "title");
                          //$results was
                          //created by
                          //the pg_query()
$year = pg_fetch_result($results, 0, "year");
```

- This is putting the first record's (0th) "title" column into a variable named `$title`, and the first record's "year" column into a variable named `$year`
- This syntax sometimes makes it easier to see what the code is doing (less cryptic/confusing than a zero'ed index "array")

Basic DB Processing Sequence

- In order to access information from any database, you must always perform the following steps:
  - first one has to connect to the database
  - Then run SQL commands against tables in the database
  - Process any information that is returned from the database (if there is any)
- To complete all of this for this course we only need to use the following PHP provided functions:
  - `pg_connect()`
  - `pg_query()`
  - `pg_num_rows()`
  - `pg_fetch_result()`

pg_num_rows()

- Used to determine if any records were returned in a result set
- Used to decide if any further processing required
- Example:

```
$records = pg_num_rows($results); //$results was
                                  //created by
                                  //the pg_query()
```

- `$records` is the total number of records in the result, will usually be 0 or above (integer values as it is a count)
- Can throw a `-1`, if there was an error (usually occurs if you do not pass a result set argument)

Realistic pg_fetch_result()

- Usually (though not always) a SQL SELECT statement will return multiple records

The following gives a way to process them all:

```
$conn = pg_connect("host=127.0.0.1 dbname=DB_NAME
                    user=USERID password=PASSWORD");
$sql = "SELECT title, year FROM movies";
$results = pg_query($conn, $sql);
$records = pg_num_rows($results);
for($i = 0; $i < $records; $i++){
  $title = pg_fetch_result($results, $i, "title");
  $year = pg_fetch_result($results, $i, "year");
  echo "<p>The movie " . $title . " was released in " . $year ."</p>";
}
```

- The for loop starts at the 0th record and goes to the (n-

PHP functions

`int time();`

- https://www.php.net/manual/en/function.time.php
- Takes no arguments
- Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).
- Current time comes from the computer that PHP is running on

e.g. time() today ~45-50 years after 1970-1-1

returns approx. 1.5 billion

A week from now would be:

```
$weekFromNow = time( ) + 60*60*24*7;
```

UNIX Type Operating Systems

UNIX is a family of multitasking, multiuser computer operating systems (O/S):

- developed in the 1970s
- immensely popular, but some basic design flaws (some importantly dealing with security)

With the advances in networking in the last couple decades (i.e. the World Wide Web) several other O/S were created to address vulnerabilities.

Several have been created, and the majority were designed to look/feel like UNIX, ergo "UNIX Type" Operating Systems

The course web server is currently running Ubuntu, a popular version of Linux, arguably the most popular "UNIX Type" O/S

Therefore, in this class students will need some basic knowledge of UNIX commands

File Permission Commands

- Change mode (permissions) of file/directory
- using '`ls -l`', 10 fields of information are shown
- ex: `drwxr-xr--`
- first position: 'd' (directory) or '-' (file)
- next three: user permissions:
  - 'r' – Read permissions
  - 'w' – Write permissions
  - 'x' – eXecute permissions
- next three: group permissions
- last three: world/other permissions

File Permission Commands

- For this course your files and directories permissions should be: 0750
  - For directories this corresponds to `drwxr-x---`
    - Which means you (the owner) can do anything to the folder, and that the group (in our case www-data aka Apache) can list the files and access the directory
  - For files this corresponds to `-rwxr-x---`
    - Which means you (the owner) can do anything to all of your files, and that the group can read the content of the files and is allowed to process (i.e. execute) your * php pages
- To set this permission to all files/folders in your work directory, through PuTTy (Note the '-R' flag tells the OS to set all files/folders and subfolders permissions recursively):

```
cd /var/www/html/webd2201/userid
chmod -R 0750 *
```

PHP Functions

`string date ( string $format [, int $timestamp ] )`

- https://www.php.net/manual/en/function.date.php

Returns a string formatted according to the given format string using the given integer *timestamp* or the current time if no timestamp is given. In other words, *timestamp* is optional and defaults to the value of time().

```
$sql = "UPDATE users SET last_access = '". date("Y-m-d",
  time()) . "' WHERE id = '".$login."'";
```

Creates:

```
UPDATE users SET last_access = '2018-03-02' WHERE id = 'jdoe'
```

- Another useful function of the date(); function is you can grab specific fields off of the optional timestamp

e.g. `<?php echo date('Y'); ?>`

NOTE: capital Y will display the current year as a 4-digit number

File System Commands

- cd – change directory
  cd *file_directory* (go into file directory)
- ls – list files
  ls –l (list files with details/long listing)
- mv – move files
- cp – copy file
- rm – remove file
- rmdir – remove directory
- mkdir – make/create directory
- passwd – change your password
- pwd – present-working-directory
- man – manual or help for the command

File Permission Commands

- permissions are represented as octal numbers
- `rwxrwxrwx = 111 111 111 = 777`
- `rwxr-x-r-- = 111 101 100 = 754`
- `rw------- = 110 000 000 = 600`

Format

`chmod <mode> <file>`

Example:

`chmod 750 index.html`

- Give yourself (the owner) all permissions, group read and execute permission, the world has no access on the file `index.html`

File Permission Commands

- Or you can use WinSCP
  - Right-mouse click on file or folder and set appropriate permissions

Login Functionality

```
<?php    //embed in a page with a two input box (named id and pass) form
   require "./includes/functions.php";
   $login = $_POST['id'];
   $password = $_POST['pass'];
   $conn = db_connect();
   $sql = "SELECT first_name, last_name, email_address, last_access FROM
      users WHERE id = '".$login."' AND password= '".$password."'";
   $results = pg_query($conn, $sql);
   if(pg_num_rows($results)){  //not zero means something was found
        //user found, use pg_fetch_result to pull user specific info to display
   }else{
        //user not found, check for just login id
        $sql = "SELECT * FROM  users WHERE id = '".$login."'";
        $results = pg_query($conn, $sql);
         if(!pg_num_rows($results)){  //user not found, empty $login to unstick it
                $login = ""; //when echo'ed in the form
        }
   }
?>
```

File Permissions

- Operating Systems support placing permissions on who can and cannot access files and directories
- Windows
  - You right-mouse click and select the file/folders properties
- UNIX/Linux
  - You must set file permissions manually

File Permission Commands

What do the permissions mean for files and folders/directories?

Access Type	File	Folder/ Directory
Read	A user/group with read permission can open the file to view its contents	A user/group with read permission on a folder can list the folders contents using the `ls` command (used in combination with the eXecute permission below)
Write	A user/group with write permission can open the file and change or overwrite its contents	A user/group with write permission on a folder can: make new files/folders; rename existing files/folders; and, delete existing files/folders in the folder.
Xecute	A user/group with eXecute permission can	A user/group with execute permission on a folder can make the folder it

chgrp Commands

- If you were to navigate into your website directory on the web server and run (assuming you have a file named `file.php`):

```
ls -l file.php
```

- The output would be similar to the following

```
-rwxr-x--- 1 usrid www-data 452 Apr 5 05:52 file.php
```

- This indicates that the user with id `usrid` is the owner (with full permissions) and that the group `www-data` has read and execute permissions for the file.
- If the group is anything other than `www-data`, your browser will give a "forbidden error" for the page you are trying to access. If so you should log into PuTTy, navigate (i.e. `cd`) into your web site folder and run the command:

```
chgrp www-data file.php
```

PHP Email Validation

- The majority of scripting or programming languages require that you code a "email validator"
  - Something that accepts a String argument and return a boolean depending on whether the string passed conform to the valid email rules
- PHP, as its sole purpose is for use on the web, provides a predefined funtion:

```
filter_var()
```

filter_var()

- The `filter_var()` function can "filter" out several things
- For the purposes of this course, it shall be used to "filter" invalid email address strings
- The syntax is:

```
filter_var($a_string_arg, FILTER_VALIDATE_EMAIL);
```

- This will either return a boolean false (if the string argument is an invalid email), or the string argument itself (if the string is a valid email address)

NOTE: the `FILTER_VALIDATE_EMAIL` is a pre-defined constant, that tells the function what it is filtering

- See: http://ca.php.net/manual/en/function.filter-var.php

Simple filter_var() Example

```
<?php
  //comes from http://www.w3schools.com/PHP/func_filter_var.asp
  if(!filter_var("someone@example.com",
                          FILTER_VALIDATE_EMAIL))
    {
      echo("E-mail is not valid");
    }
    else
    {
      echo("E-mail is valid");
    }
?>
```

File/Page Redirection

- There are situations on the Web when you need to automatically redirect a user/visitor to another web page
  - Change in URL
  - Page done processing
  - Unauthorized attempt to access protected pages
- Due to these situations, PHP provides a predefined function that redirects a page:

```
header();
```

header()

This function can send any raw HTTP header
- This includes handling page exceptions

Most popular use is to redirect a user from the executing page to a new page/location

NOTE: that the **`header()`** function will fail if you have already called an echo before the `header("Location:…");` call
- Error returned is: **Warning**: Cannot modify header information - headers already sent by… followed by a file name and line number the echo occurred

See: http://ca3.php.net/manual/en/function.header.php

header() Error Work Around

- The work around to the redirect error is to empty the buffer (of http header data), to do this in the *header.php* file, at the very top of the page type the following:

```
<?php
  require "./includes/functions.php";
          //already there from assignment
  ob_start();
?>
```

- `ob_start();` //turns on output buffering
- Then when you make the `header()` function re-direct call:

```
<?php
  header("Location:lab9_login.php");
  ob_flush();
?>
```

- `ob_flush();` // flush (send) the output buffer

Simple header() Example

```
<?php
  // this comes from:
  http://www.w3schools.com/php/func_http_header.asp
  header('Location: http://www.example.com');
          //absolute addresses work but…
  header('Location: index.php');
          //so do relative addresses
?>
```

If you ever get the error when a page is executing the header function:

Warning: Cannot modify header information - headers already sent

Means the page has already started HTML output above the call or a PHP echo was already executed

Realistic header() Example

```
<?php
  require("./includes/function.php");
                    //need file for db stuff
  ob_start();
  include("./header.php");
  $errors = "";      //initialize empty string for errors
  //Do some error checking, concatenate error messages
  //to $errors variable (no echoes!)
  if(strlen($errors) == 0){
       //$errors is still empty, which means there are no //errors
  after error checking
       //do database stuff like INSERT a new record
       header("Location:lab9.php");
       ob_flush();
```