**PHP for MySQL Library**

**PDO Objects**

* Create a PDO Object (Entire database.php file below)
  + $db is the PDO object
  + <?php

//might want a login view file to POST the username and password to this page

$dsn = ‘mysql:host=*localhost:8080*;dbname=*database\_name*’; // use host name and db name

$username = ‘*username*’;

$password = ‘*password’*;

try {

$db = new PDO($dsn, $username, $password);

$success\_message = ‘You are connected to the database!’;

} catch (PDOException $e) {

$error\_message = $e->getMessage();

include(‘database\_error.php’);

exit();

}

?>

* The ‘database\_error.php’ file (a VIEW file)
  + <!DOCTYPE html>

<html>

<!-- the head section -->

<head>

<title>Database Error</title>

<link rel=”stylesheet” type=”text/css” href=”*filename.css*”>

</head>

<!-- the body section -->

<body>

<main>

<h1>Database Error</h1>

<p>There was a problem connecting to the database.</p>

<p>Error message: <?php echo $error\_message; ?></p>

</main>

</body>

</html>

* **Prepared Statements**
  + When calling prepare() method, you return a PDOStatement Object
    - lastInsertId(); gets the ID that was automatically generated for the row
  + Methods of the PDOStatement Object
    - bindValue(‘:*sqlVarName*’, *$phpVarName*);
      * Named parameters
        + Binds the specified value to the parameter in the prepared statement
      * Unnamed parameters
        + Use ? instead of *:sqlVarName* in the qry, integers used with the bindValue method instead of *:sqlVarName* (first ? is 1, second ? is 2…)
    - execute();
    - fetchAll(); // returns all of the rows of the result set
      * Returns numbered and associative indexes, not useful unless you change settings
      * fetchAll(PDO::FETCH\_GROUP|PDO::FETCH\_ASSOC);
        + Will return with table primary key as array key
        + Each primary key value will have an array with the rest of the table values associated with it
        + Use array\_map(‘reset’, *$array\_name*); to remove extra numbered index values and return your associative array
        + Store the array keys (linked to each row) in a variable $PrimaryKey = array\_keys(*$array\_name*);
        + See section below on full prepared query and functions
    - fetch(); // returns the next row or NULL if out of rows or none returned
      * Common to use fetch(); then use a while loop to process (uses less memory)
        + $results = $statement->fetch();

while ($results != NULL) {

code to process

$results = $statement->fetch(); // get the next row

}

* + - rowCount(); //returns the number of rows affected by the last statement
    - closeCursor(); //closes the cursor and frees the connection to the database for the next query
* Error modes for the PDO (query errors)
  + ERRMODE\_SILENT (default)
    - Sets an error but does not emit php warning or throw exception
    - errorCode() and errorInfo() methods can retrieve the error from the PDO or statement object
    - Must actively check for errors
  + ERRMODE\_WARNING
    - PDO sets the error and doesn’t throw an exception
    - It does emit a php warning message
    - Useful for testing/debugging
  + ERRMODE\_EXCEPTION
    - Sets error and throws a PDOException object
    - Very useful for testing/debugging
  + Typically set back to SILENT (default) after debugging
  + Setting the error mode for a PDO object
    - When debugging, use try/catch statements to catch these errors
    - Use constructor of the PDO class when creating the PDO object
      * $options = array(PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION);

try { $db = new PDO($dsn, $un, $pw, $options); } catch { …

* + - Use setAttribute() method
      * $db->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);
      * I like this one better, easier to remove after debugging
* Passing variables to SQL code within a php file (see bindVaue method)
  + Prepared statements (best method)
    - Adding $var1 and $var2 obtained in php file to a query
    - $query = ‘SELECT \* FROM tableName

WHERE columnName1 = :var1

AND columnName2 = :var2’

$statement = $db->prepare($query);

$statement->bindValue(‘:var1’, $var1);

$statement->bindValue(‘:var2’, $var2);

$statement->execute();

$*returnVarName* = $statement->fetchAll(); // fetchAll() not so great without params

$statement->closeCursor();

* + Escape characters using quote() method of the PDO Object (not as good as prepared statements)
    - $escaped\_var = $db->quote($var);
    - $query = “SELECT \* FROM tableName WHERE columnName = $escaped\_var”;
    - Can use to INSERT or UPDATE data too (must escape before inserting in database by doing this or by using prepared statemens (better method))
* Using the query method for SELECT queries (not as good as using prepared statements)
  + $query = ‘QUERY PARAMS HERE’;

$var = $db->query($query);

* Using the exec method for INSERT, UPDATE, and DELETE queries (not as good as prepared statements)
  + First escape any variables using quote() method
  + $query = “INSERT INTO tableName

(column1, column2…)

VALUES ($escaped\_var1, $escaped\_var2…)”;

$insert\_count = $db->exec($query); // $insert\_count contains the number of inserted rows

* + $query = “UPDATE tableName

SET columnName = $escaped\_var

WHERE columnName = $escaped\_var2”;

$update\_count = $db->exec($query); // $update\_count contains the number of updated rows

* + $query = “DELETE FROM tableName

WHERE columnName = $escaped\_var”;

$delete\_count = $db->exec($query); // $delete\_count contains the number of deleted rows

**Query Examples Using Prepared Statements**

* If using functions, must first access the global database (global $db;) within the function
* Database table structure for these examples:
  + Sampling table with info on when, where, and conditions a sample was collected
  + Samplers table with info on who took samples including contact info
  + Flounders table with data on the specimens collected
  + Species table with data on each species
* SELECT query from one table, selecting all columns
  + Obtain $start\_date and $end\_date from user inputs on web form
  + $query = ‘SELECT \* FROM Sampling

WHERE sampleDate >= :startDate

AND sampleDate <= :endDate’;

$statement = $db->prepare($query);

$statement->bindValue(‘:startDate’, $start\_date);

$statement->bindValue(‘:endDate’, $end\_date);

$statement->execute();

$result\_i = $statement->fetchAll(PDO::FETCH\_GROUP|PDO::FETCH\_ASSOC);

$result = array\_map(‘reset’, $result\_i); // need more processing to display $result in table

$statement->closeCursor();

return $result; // necessary if query is within a function

* SELECT query from three tables, selecting specific columns
  + $query = ‘SELECT location, date, specimenNum, totalLength, speciesName

FROM Sampling s

INNER JOIN Flounders f

ON s.sampleCode = f.sampleCode

INNER JOIN Species sp

ON f.speciesCode = sp.speciesCode

ORDER BY date’;

$statement = $db->prepare($query);

$statement->execute();

$result\_i = $statement->fetchAll(PDO::FETCH\_GROUP|PDO::FETCH\_ASSOC);

$result = array\_map(‘reset’, $result\_i); // needs more processing

$statement->closeCursor();

return $result; // necessary if query is within a function

* INSERT query to add data to a table
  + $query = ‘INSERT INTO sampling

(sampleCode, sampler, date, location, latDeg, longDeg, bottomType, bottomTemp, depth)

VALUES

(:sampleCode, :sampler, :date, :location, :latDeg, :longDeg, :bottomType, :bottomTemp, :depth)’;

$statement = $db->prepare($query);

$statement->bindValue(‘:sampleCode’, $sampleCode);

$statement->bindValue(‘:sampler’, $sampler);

$statement->bindValue(‘:date’, $date); // should process/format date first

$statement->bindValue(‘:location’, $location);

$statement->bindValue(‘:latDeg’, $latDeg);

$statement->bindValue(‘:longDeg’, $longDeg);

$statement->bindValue(‘:bottomType’, $bottomType);

$statement->bindValue(‘:bottomTemp’, $bottomTemp);

$statement->bindValue(‘:depth’, $depth);

$statement->execute();

$statement->closeCursor();

$sample\_id = $db->lastInsertID();

return $sample\_id; // will return autogen ID assigned by MySQL for the INSERT query

* UPDATE query to modify data in a table
  + $query = ‘UPDATE Species

SET speciesName = :speciesName,

gillRakers = :gillRakers,

family = :family

WHERE speciesCode = :speciesCode’;

$statement = $db->prepare($query);

$statement->bindValue(‘:speciesName’, $speciesName);

$statement->bindValue(‘:gillRakers’, $gillRakers);

$statement->bindValue(‘:family’, $family);

$statement->bindValue(‘:speciesCode’, $speciesCode);

$row\_count = $statement->execute(); // affected # of rows

$statement->closeCursor();

return $row\_count;

* DELETE query to delete a row in a table
  + $query = ‘DELETE FROM samplers

WHERE sampler = :samplerName’;

$statement = $db->prepare($query);

$statment->bindValue(‘:samplerName’, $samplerName);

$row\_count = $statement->execute();

$statement->closeCursor();

return $row\_count;

* Functions needed to display $result or $data in a table using array returned from

$data = fetchAll(PDO::FETCH\_GROUP|PDO::FETCH\_ASSOC);

* + *$arr\_keys* = array\_keys(*$arr\_name*); // will store the primary key for each row in an array
  + $num\_rows = count(*$array\_keys*); // tells you the # of primary keys (rows) returned by the query
  + $*key\_values* = $data[*$arr\_keys*[0]]; // delivers the column names in an array as keys for one row
  + $column\_headers = array\_keys(*$key\_values*); // gets all of your column names except the first one
    - array\_unshift($column\_headers, ‘*ColumnOneName*’); // adds column one name to your list
  + Displaying the column headers
    - <tr>foreach with <th></th> </tr>
  + Displaying the data
    - for and and nested foreach
  + Full functions from flounder database pasted below:
    - function get\_all\_sampling\_data() {
    - global $db;
    - $query = 'SELECT \* FROM samplingdata
    - ORDER BY SampleID';
    - try {
    - $statement = $db->prepare($query);
    - $statement->execute();
    - // fetchAll, using tbl primary key as array key
    - $data\_i = $statement->fetchAll(PDO::FETCH\_GROUP|PDO::FETCH\_ASSOC);
    - $data = array\_map('reset', $data\_i); // strip out numbered index
    - $statement->closeCursor();
    - return $data;
    - } catch (PDOException $e) {
    - display\_error($e->getMessage());
    - }
    - }
    - // get sample IDs from returned query
    - function get\_SampleIDs($data) {
    - $SampleIDs = array\_keys($data);
    - return $SampleIDs;
    - }
    - // get number of rows from returned query for format table function
    - function get\_num\_rows($SampleIDs) {
    - $num\_rows = count($SampleIDs);
    - return $num\_rows;
    - }
    - // format key values to get column headers
    - function key\_values($data, $SampleIDs) {
    - $key\_values = $data[$SampleIDs[0]];
    - return $key\_values;
    - }
    - // get column headers from returned query
    - function get\_column\_headers($key\_values) {
    - $column\_headers = array\_keys($key\_values);
    - array\_unshift($column\_headers, 'SampleID');
    - return $column\_headers;
    - }
    - // display column headers for table (wrap in <th> tags using foreach
    - function display\_column\_headers($column\_headers) {
    - $tag\_th = 'th';
    - $before\_th = '<' . $tag\_th . '>';
    - $after\_th = '</' . $tag\_th . '>';
    - $tag\_tr = 'tr';
    - $before\_tr = '<' . $tag\_tr . '>';
    - $after\_tr = '</' . $tag\_tr . '>';
    - echo $before\_tr;
    - foreach ($column\_headers as $column\_header) {
    - $ch = $before\_th . $column\_header . $after\_th;
    - echo $ch;
    - }
    - echo $after\_tr;
    - }
    - // format data for table body
    - function format\_table($data, $num\_rows, $SampleIDs) {
    - $tag\_tr = 'tr';
    - $before\_tr = '<' . $tag\_tr . '>';
    - $after\_tr = '</' . $tag\_tr . '>';
    - $tag\_td = 'td';
    - $before\_td = '<' . $tag\_td . '>';
    - $after\_td = '</' . $tag\_td . '>';
    - for ($i = 0; $i < $num\_rows; $i++) {
    - $td\_a = $before\_tr . $before\_td . $SampleIDs[$i] . $after\_td;
    - $cells = $data[$SampleIDs[$i]];
    - echo $td\_a;
    - foreach ($cells as $cell) {
    - $entry = $before\_td . $cell . $after\_td;
    - echo $entry;
    - }
    - echo $after\_tr;
    - }
    - }
    - In view file:
      * <table>
      * <thead><?php echo display\_column\_headers($column\_headers); ?></thead>
      * <tbody><?php echo format\_table($data, $num\_rows, $SampleIDs); ?></tbody>
      * </table>

**MVC Model Elements**

* Utility Files
  + General purpose files used throughout a website
  + util/main.php file

<?php

$doc\_root = filter\_input(INPUT\_SERVER, ‘DOCUMENT\_ROOT’);

// Gets the document root of the Apache server (usually htdocs dir)

$uri = filter\_input(INPUT\_SERVER, ‘REQUEST\_URI’);

$dirs = explode(‘/’, $uri);

$app\_path = ‘/’ . $dirs[1] . ‘/’ . $dirs[2] . ‘/’;

// this code assumes the app directory is two directories down from htdocs dir

set\_include\_path($doc\_root . $app\_path);

?>

* View
  + Header files
    - <!DOCTYPE html>

<html>

<!-- the head section -->

<head>

<title>App Title Name</title>

<link rel=”stylesheet” type=”text/css”

href=”<?php echo $app\_path ?>main.css” />

</head>

<!-- the body section -->

<body>

<header>

<h1>Main Application Title</h1>

</header>

* + Footer files (can include more here if desired)
    - <footer>

<p class=”copyright”>

&copy; <?php echo date(“Y”); ?> Copyright Owner</p>

</footer>

</body>

</html>

* + Sidebars (sidebar.php file that only contains sidebar stuff)
    - <aside>

<h2>Links</h2> // first list contains general links

<ul>

<li>

<a href=”<?php echo $app\_path; ?>”>Home</a>

</li>

</ul>

<h2>Header for your list of database link items (db categories)</h2>

<ul>

<li>

<a href=”<?php echo $app\_path . ‘*directory’* . // optional dir

‘?action=*action\_name*’ . // pass action var &/or other var

‘$amp;*var\_name*=’ . $*array[‘index’]*; ?>”>

<?php echo $*array[‘value’]*; ?> // link display from array

</a>

</li>

<?php endforeach; ?>

<li>&nbsp;</li>

</ul>

</aside>

* + Including subviews within a view page
    - <?php include ‘../../view/header.php’; ?>

<?php include ‘../../view/sidebar.php’: ?>

<section>

<!-- main view content here for this page -->

</section>

<?php include ‘../../view/footer.php’;

* + Dropdown lists
    - <select name=”*form\_field\_name*”>

<!-- displays database fields for the dropdown list -->

<?php foreach(*$items* as *$item*) : ?>

<option value=”<?php echo *$array[‘index’]*; ?>

<?php echo *$array[‘index’]*; ?>

</option>

<?php endforeach; ?>

</select><br>

* + Images (in an images folder)
    - Set $image\_path and $image\_alt variables
      * $image\_filename = $related\_var\_name . ‘.png’; // use system for naming images
      * $image\_path = $app\_path . ‘images/’ . $image\_filename;
      * $image\_alt = ‘Image filename: ‘ . $image\_filename;
    - <img src=”<?php echo $image\_path; ?>”

alt=”<?php echo $image\_alt; ?>”>

**User Authentication**

* Form-Based Authentication
  + Most common for production websites
  + Gives much control (you design the form/page)
  + Most work to create
  + Sends info to server as plain text (need SSL to encrypt)
* Basic Authentication
  + The browser displays the dialog box asking for username and password
  + Browser determines appearance of the form
  + Not common for production websites
  + Sends info to server as plain text (need SSL to encrypt)
* Digest Authentication
  + Works like basic, but the data is encrypted prior to sending to the server
  + Less secure than Basic with SSL
  + Least common type
* Hashing a password to encrypt it
  + Use password\_hash(); function
  + $password = ‘pa$$word’;

$hash = password\_hash($password, PASSWORD\_DEFAULT);

// can set PASSWORD\_BCRYPT to always use the bcrypt algorithm (right now it’s the default, that may change)

* Setup for implementing Form Based Authentication
  + Create a table for storing usernames and passwords in the database
  + Suggested fields
    - adminID (autonumber) (primary key)
    - emailAddress
    - password (255 chars is best practice)
    - firstName
    - lastName
  + Create admin\_db.php (model) file
    - <?php

// function to add users to the database

function add\_admin($email, $password, $level) {

global $db;

$hash = password\_hash($password, PASSWORD\_DEFAULT);

$query = ‘INSERT INTO *passwordTableName* (emailAddress, password, level)

VALUES (:email, :password, :level)’;

$statement = $db->prepare($query);

$statement->bindValue(‘:email’, $email);

$statement->bindValue(‘:password’, $hash);

$statement->bindValue(‘:level’, $level); // set user level (admin or user)

$statement->execute();

$statement->closeCursor();

}

// function to verify email and passwords against the database

function is\_valid\_admin\_login($email, $password) {

global $db;

$query = ‘SELECT password FROM *passwordTableName*

WHERE emailAddress = :email’;

$statement = $db->prepare($query);

$statement->bindValue(‘:email’, $email);

$statement->execute();

$row = $statement->fetch();

$statement->closeCursor();

$hash = $row[‘password’];

return password\_verify($password, $hash);

}

?>

* + Controller section for protected pages
    - <?php

// start session management and include necessary functions

session\_start();

require\_once(‘model/database.php’);

require\_once(‘model/admin\_db.php’);

// get the action to perform

$action = filter\_input(INPUT\_POST, ‘action’);

if ($action == NULL) {

$action = filter\_input(INPUT\_GET, ‘action’);

if ($action == NULL) {

$action = ‘*default\_action*’;

}

}

// If the user isn’t logged in, force the user to login

if (!isset($\_SESSION[‘is\_valid\_admin’])) { // could use diff than ‘is\_valid\_admin’

$action = ‘login’;

}

// Perform the specified action

switch($action) {

case ‘login’:

$email = filter\_input(INPUT\_POST, ‘email’);

$password = filter\_input(INPUT\_POST, ‘password’);

if (is\_valid\_admin\_login($email, $password)) {

$\_SESSION[‘is\_valid\_admin’] = true; // again with variable name

include(‘view/*mainViewPage*.php’);

} else {

$login\_message = ‘You must login to view this page.’;

include(‘view/login.php’);

}

break;

\\ other cases here that require you to be logged in

case ‘logout’:

$\_SESSION = array(); // clear all session data

session\_destroy(); // clean up the session ID

$login\_message = ‘You have been logged out.’;

include(‘view/login.php’);

break;

}

?>

* + Utility file that forces a valid admin user (util/valid\_admin.php) // filename for example
    - <?php

// make sure the user is logged in as administrator (or user)

if (!isset($\_SESSION[‘is\_valid\_admin’])) { // again with variable name

header(“Location: .” ); // if not logged in, redirects to the controller

}

?>

* + Code that’s included at the top of the login page
    - <?php require\_once(‘util/secure\_conn.php’); ?> // require a secure connection
  + Code that’s included at the top of other protected pages
    - <?php require\_once(‘util/secure\_conn.php’);

require\_once(‘util/valid\_admin.php’); ?> // example filename

* Setup for implementing Basic Authentication
  + Typically no logout button on view pages
  + See Murach p. 713

**Secure Connections**

* See p. 687 Murach for more details
* Redirecting to a secure connection
  + If a user requests http, this will redirect to https
  + include this file (util/secure\_conn.php)
    - <?php

if (!https) {

$host = filter\_input(INPUT\_SERVER, ‘HTTP\_HOST’);

$uri = filter\_input(INPUT\_SERVER, ‘REQUEST\_URI’);

$url = ‘https://’ . $host . $uri;

header(“Location: “ . $uri);

exit();

}

?>

**Working with Files**

* Get directory listings
  + is\_file($path); // returns true if $path exists and is a file
  + is\_dir($path); // returns true if $path eists and is a directory
  + file\_exists($path); // returns true if $path exists and is either a file or a directory
  + getcwd(); // returns a string that specifies the current working directory
    - Can use to append file names, or move up/down directories
  + Windows uses \ to separate directory levels, while Mac/Linux use /
    - Use DIRECTORY\_SEPARATOR to always insert the correct one regardless of OS
  + scandir($path); // returns an array containing a list of the files and directories in $path (returns false if not a valid dir)
  + Code to display all items in a directory or all files in a directory is listed on p. 745 Murach.
* Reading entire files
  + file($name); // returns an array with each element containing one line from the file
  + file\_get\_contents($name); // returns the contents of the file as a string
  + readfile($name); // reads a file and echoes it to the web page
* Reading/writing part of a file
  + Opening/closing files
    - fopen($path, $mode);
      * modes
        + ‘rb’ // opens the file for reading (returns false if doesn’t exist)
        + ‘wb’ // opens the file for writing, erases all data first if exists, creates the file if it doesn’t
        + ‘ab’ // opens the file for writing, new data is appended to any existing data, file is created if doesn’t exist
        + ‘xb’ // creates a new file for writing, returns false if the file already exists
    - feof($file); // returns true when the end of the specified file is reached
    - fclose($file); // closes the specified file
  + Reading/writing opened files
    - fread($file, $length); // reads up to the specified number of bytes from the specified file
    - fgets($file); // reads a line from the specified file
    - fwrite($file, $data); // writes the specified string data to the specified file (must add any new line characters you want
* Writing entire files
  + file\_put\_contents($name, $data); // writes the specified data string to the specified filename
    - Can use implode(); to convert array to a string then use file\_put\_contents to write an array of lines
* **Reading and Writing CSV Files**
  + If a field within a record contains a comma, double quote, or line break, the field must be surrounded by double quotes
  + Reading
    - fgetcsv($file); // reads one line of csv and returns them in an array
  + Writing
    - fputcsv($file, $array); // writies the specified array to the specified file as a line of csv
* **Writing a CSV file from a returned query**
  + $file = fopen(*$path*, ‘wb’); // opens a file for writing, will erase any data if there, or will create if doesn’t exist
    - // use ‘ab’ to append data to a file, or use ‘xb’ to create a new file that can’t overwrite an old file (must use a new file name)
    - $path could be ‘download.csv’
  + Need a nested array (array that contains all the data to be written, each line is an array)
  + foreach ($data as $datum) {

fputcsv($file, $datum);

}

* + fclose($file);

**Examples from flounderdb**

* + Executing queries and formatting data
    - function scdnr\_sampling\_data() {
    - global $db;
    - $query = 'SELECT \* FROM samplingdata
    - WHERE Sampler = "SCDNR"
    - ORDER BY SampleID';
    - try {
    - $statement = $db->prepare($query);
    - $statement->execute();
    - // fetchAll, using tbl primary key as array key
    - $data\_i = $statement->fetchAll(PDO::FETCH\_GROUP|PDO::FETCH\_ASSOC);
    - $statement->closeCursor();
    - // format the sampling data
    - $data = format\_sampling\_data($data\_i);
    - return $data;
    - } catch (PDOException $e) {
    - display\_error($e->getMessage());
    - }
    - }
    - function format\_sampling\_data($data\_i) {
    - $SampleIDs = get\_SampleIDs($data\_i);
    - $data = [];
    - // extract sampling data into an array
    - foreach ($SampleIDs as $SampleID) {
    - $trips = $data\_i[$SampleID];
    - $num\_trips = count($trips);
    - for ($i = 0; $i < $num\_trips; $i++) {
    - array\_push($data, $trips[$i]);
    - }
    - }
    - // set SampleID as first column
    - for ($i = 0; $i < count($data); $i++) {
    - $data[$i] = array('SampleID' => $SampleIDs[$i]) + $data[$i];
    - }
    - return $data;
    - }
    - function format\_flounder\_data($data\_i) {
    - $SampleIDs = get\_sampleIDs($data\_i);
    - //$num\_sampleids = count($SampleIDs);
    - $flounder = [];
    - $data = [];
    - // stores all data with numbered index as rows
    - foreach ($SampleIDs as $SampleID) {
    - $flounder = $data\_i[$SampleID];
    - $num\_flounder = count($flounder);
    - for ($i = 0; $i < $num\_flounder; $i++) {
    - array\_push($data, $flounder[$i]);
    - }
    - }
    - return $data;
    - }
    - function get\_all\_flounder\_data() {
    - global $db;
    - $query = 'SELECT \* FROM samplingdata s
    - INNER JOIN flounders f
    - ON s.SampleID = f.SampleID
    - INNER JOIN species sp
    - ON f.SpCode = sp.SpCode
    - ORDER BY f.SampleID';
    - try {
    - $statement = $db->prepare($query);
    - $statement->execute();
    - // fetchAll, using tbl primary key as array key
    - $data\_i = $statement->fetchAll(PDO::FETCH\_GROUP|PDO::FETCH\_ASSOC);
    - $statement->closeCursor();
    - // format flounder data
    - $data = format\_flounder\_data($data\_i);
    - return $data;
    - } catch (PDOException $e) {
    - display\_error($e->getMessage());
    - }
    - }
    - function format\_csv\_data($data, $num\_rows, $SampleIDs, $column\_headers) {
    - $array = [];
    - $array[0] = $column\_headers;
    - for ($i = 0; $i < $num\_rows; $i++) {
    - $rows = $data[$SampleIDs[$i]];
    - $array[] = $rows;
    - }
    - return $array;
    - }
    - function write\_csv($data) {
    - $file = fopen('download.csv', 'wb');
    - foreach ($data as $datum) {
    - fputcsv($file, $datum);
    - }
    - fclose($file);
    - }
    - from index.php
    - case: ‘download\_data’:
    - $file\_name = 'download.csv';
    - // output headers so the file is downloaded
    - header("Content-Type: text/csv");
    - header("Content-Disposition: attachment; filename=$file\_name");
    - // disable caching
    - header("Cache-Control: no-cache, no-store, must-revalidate"); // HTTP 1.1
    - header("Pragma: no-cache"); // HTTP 1.0
    - header("Expires: 0"); // Proxies
    - // download file
    - readfile($file\_name);
    - break;
    - Not used code, but will immediately create output.csv file from a query
    - function outputCSV($data, $file\_name = 'download.csv') {
    - // output headers so the file is downloaded
    - header("Content-Type: text/csv");
    - header("Content-Disposition: attachment; filename=$file\_name");
    - // disable caching
    - header("Cache-Control: no-cache, no-store, must-revalidate"); // HTTP 1.1
    - header("Pragma: no-cache"); // HTTP 1.0
    - header("Expires: 0"); // Proxies
    - // start the output
    - $output = fopen("php://output", "w");
    - // loop through data array
    - foreach ($data as $datum) {
    - fputcsv($output, $datum);
    - }
    - //close file
    - fclose($output);
    - }