Lesson Number 4

Name:

Input Validation

Description:

GROUP ACTIVITY - PHP Quick Review

5 minutes with a partner to jot down (without looking at notes) what you remember from last week.

ACTIVITY - add_song.php & artist_songs.php

- add a song but ensure we add the artist id for that song
- create a view that shows all the songs for one artist
- use the HTTP request method GET and the corresponding link through artists.php (link the artist name)

ACTIVITY - .user.ini & Introducing try/catch()

Adding Error Output to Azure

- create a .user.ini file in the site root folder (httpdoc or www)
- populate it with the following

```
display_errors=On
```

Introducing try/catch()

- known as an exception
- used to catch errors produced by code so they can be handled
    1. **try** the code that could possibly thrown an error
    2. **catch** the error that is thrown
        - you can catch multiple errors by using multiple catch blocks
        - you can throw custom errors using the keyword throw
    3. PDO uses the exception PDOException when it encounters an error
- IE:

```
try {
  echo "I'm intentionally throwing an exception.";
  throw new Exception("Here it is.");
} catch ( Exception $e ) {
  echo '<pre>', var_dump($e), '</pre>';
}
```

Client-Side Validation & Server-Side Validation

Client-Side Validation

- Why do we validate client-side?
    - user experience
    - to avoid unnecessary errors
    - to encourage specific formats
- HTML 5 Validation
    - we can use required to ensure a field is populated before being submitted
    - we can control data format by providing interactive HTML widgets that collect data in a specific format
        - we can also place limitations on data such as minimum and maximum values
    - we can provide placeholders for examples of data
    - we can use HTML5 input patterns with regular expression to create custom input types
- JavaScript Validation

- we can fine tune our requirements, such as neediing one field or another not both
- we can compare fields such as emails and passwords
- we can provide feedback to the user to give them visual validation
- we can provide custom logic to a field allowing us greater control over the inputted data
  - ie: we can force the selection of two or more checkboxes

Server-Side Validation

- Why do we validate server-site?
  - to avoid insecure data
  - to maintain data integrity
  - to ensure data format
- Is bindParam() enough for server-side validation?
  - this won't prevent against XSS (cross-site attacks) as it doesn't prevent client-side scripts from being placed in the database
  - it can't maintain data format
- Server-Side Validation
  - we can check if a required input was empty
  - we can validate the data is in the correct format
  - we can check that the form submitted is from our site
  - we can ensure no data being passed is client-side code or HTML
  - we can verify if the data was requested/submitted via AJAX
  - we can verify the required response (HTML, JSON, XML, etc...)

ACTIVITY - Updating Our SQL

NOTE: Remove all duplicates from your tables before applying the below entries!

```
ALTER TABLE artists ADD UNIQUE (name);
ALTER TABLE songs ADD UNIQUE unique_index(artist_id, title);
```

ACTIVITY - Client Side Validation

new_artists.php

new_song.php

Common Server-Side Validation/Sanitation Checks

Validation

- empty()
  - true if any of the following values are found:
    - *""* (an empty string)
    - *0* (0 as an integer)
    - *0.0* (0 as a float)
    - *"0"* (0 as a string)
    - `NULL`
    - `FALSE`
    - *array()* (an empty array)
    - *$var;* (a variable declared, but without a value)
- isset()
  - true if any of the following values are met:
    - not **NULL**
    - not **FALSE**
    - $var exists
- filter_var()

ID	Name	Options	Flags	Description
`FILTER_VALIDATE_BOOLEAN`	"boolean"	`default`	`FILTER_NULL_ON_FAILURE`	Returns `TRUE` for "1", "true", "on" and "yes". Returns `FALSE` otherwise. If `FILTER_NULL_ON_FAILURE` is set, `FALSE` is

				returned only for "0", "false", "off", "no", and "", and `NULL` is returned for all non-boolean values.
`FILTER_VALIDATE_EMAIL`	"validate_email"	`default`		Validates whether the value is a valid e-mail address. In general, this validates e-mail addresses against the syntax in RFC 822, with the exceptions that comments and whitespace folding are not supported.
`FILTER_VALIDATE_FLOAT`	"float"	`default`, `decimal`	`FILTER_FLAG_ALLOW_THOUSAND`	Validates value as float, and converts to float on success.
`FILTER_VALIDATE_INT`	"int"	`default`, `min_range`, `max_range`	`FILTER_FLAG_ALLOW_OCTAL`, `FILTER_FLAG_ALLOW_HEX`	Validates value as integer, optionally from the specified range, and converts to int on success.
`FILTER_VALIDATE_IP`	"validate_ip"	`default`	`FILTER_FLAG_IPV4`, `FILTER_FLAG_IPV6`, `FILTER_FLAG_NO_PRIV_RANGE`, `FILTER_FLAG_NO_RES_RANGE`	Validates value as IP address, optionally only IPv4 or IPv6 or not from private or reserved ranges.
`FILTER_VALIDATE_MAC`	"validate_mac_address"	`default`		Validates value as MAC address.
`FILTER_VALIDATE_REGEXP`	"validate_regexp"	`default`, `regexp`		Validates value against `regexp`, a[Perl-compatible](#) regular expression.
`FILTER_VALIDATE_URL`	"validate_url"	`default`	`FILTER_FLAG_PATH_REQUIRED`, `FILTER_FLAG_QUERY_REQUIRED`	Validates value as URL (according to[» http://www.faqs.org/rfcs/rfc2396](#)), optionally with required components. Beware a valid URL may not specify the HTTP protocol*http://* so further validation may be required to determine the URL uses an expected protocol, e.g. *ssh://* or*mailto:*. Note that the function will only find ASCII URLs to be valid; internationalized domain names (containing non-ASCII characters) will fail.

- comparison operators

Example	Name	Result
$a == $b	Equal	`TRUE` if *$a* is equal to *$b* after type juggling.
$a === $b	Identical	`TRUE` if *$a* is equal to *$b*, and they are of the same type.
$a != $b	Not equal	`TRUE` if *$a* is not equal to *$b* after type juggling.
$a <> $b	Not equal	`TRUE` if *$a* is not equal to *$b* after type juggling.
$a !== $b	Not identical	`TRUE` if *$a* is not equal to *$b*, or they are not of the same type.
$a < $b	Less than	`TRUE` if *$a* is strictly less than *$b*.
$a > $b	Greater than	`TRUE` if *$a* is strictly greater than *$b*.
$a <= $b	Less than or equal to	`TRUE` if *$a* is less than or equal to *$b*.
$a >= $b	Greater than or equal to	`TRUE` if *$a* is greater than or equal to *$b*.
$a <=> $b	Spaceship	An integer less than, equal to, or greater than zero when *$a* is respectively less than, equal to, or greater than *$b*. Available as of PHP 7.

$a ?? $b ?? $c	Null coalescing	The first operand from left to right that exists and is not `NULL`. `NULL` if no values are defined and not `NULL`. Available as of PHP 7.

Sanitation

- strip_tags()
  - removes all HTML and PHP tags from a string
  - you can whitelist allowable tags
- addslashes()
  - escapes quote characters and the forward and backward slash characters
- htmlspecialchars()
  - converts HTML characters into their character codes
    - these include <, >, &, "", ''
- filter_var()

ID	Name	Flags	Description
`FILTER_SANITIZE_EMAIL`	"email"		Remove all characters except letters, digits and *!#$%&'*+-=?^_`{|}~@.[]*.
`FILTER_SANITIZE_ENCODED`	"encoded"	`FILTER_FLAG_STRIP_LOW`, `FILTER_FLAG_STRIP_HIGH`, `FILTER_FLAG_ENCODE_LOW`, `FILTER_FLAG_ENCODE_HIGH`	URL-encode string, optionally strip or encode special characters.
`FILTER_SANITIZE_MAGIC_QUOTES`	"magic_quotes"		Apply addslashes().
`FILTER_SANITIZE_NUMBER_FLOAT`	"number_float"	`FILTER_FLAG_ALLOW_FRACTION`, `FILTER_FLAG_ALLOW_THOUSAND`, `FILTER_FLAG_ALLOW_SCIENTIFIC`	Remove all characters except digits, +-and optionally *.,eE*.
`FILTER_SANITIZE_NUMBER_INT`	"number_int"		Remove all characters except digits, plus and minus sign.
`FILTER_SANITIZE_SPECIAL_CHARS`	"special_chars"	`FILTER_FLAG_STRIP_LOW`, `FILTER_FLAG_STRIP_HIGH`, `FILTER_FLAG_ENCODE_HIGH`	HTML-escape *'"<>&* and characters with ASCII value less than 32, optionally strip or encode other special characters.
`FILTER_SANITIZE_FULL_SPECIAL_CHARS`	"full_special_chars"	`FILTER_FLAG_NO_ENCODE_QUOTES`,	Equivalent to calling htmlspecialchars()with `ENT_QUOTES` set. Encoding quotes can be disabled by setting `FILTER_FLAG_NO_ENCODE_QUOTES`. Likehtmlspecialchars(), this filter is aware of the default_charset and if a sequence of bytes is detected that makes up an invalid character in the current character set then the entire string is rejected resulting in a 0-length string. When using this filter as a default filter, see the warning below about setting the default flags to 0.
`FILTER_SANITIZE_STRING`	"string"	`FILTER_FLAG_NO_ENCODE_QUOTES`, `FILTER_FLAG_STRIP_LOW`, `FILTER_FLAG_STRIP_HIGH`, `FILTER_FLAG_ENCODE_LOW`, `FILTER_FLAG_ENCODE_HIGH`, `FILTER_FLAG_ENCODE_AMP`	Strip tags, optionally strip or encode special characters.
`FILTER_SANITIZE_STRIPPED`	"stripped"		Alias of "string" filter.
`FILTER_SANITIZE_URL`	"url"		Remove all characters except letters, digits and *$-_.+!*'(),{}|\\^~[]`<>#%";/?:@&=*.
`FILTER_UNSAFE_RAW`	"unsafe_raw"	`FILTER_FLAG_STRIP_LOW`, `FILTER_FLAG_STRIP_HIGH`, `FILTER_FLAG_ENCODE_LOW`, `FILTER_FLAG_ENCODE_HIGH`, `FILTER_FLAG_ENCODE_AMP`	Do nothing, optionally strip or encode special characters. This filter is also aliased to `FILTER_DEFAULT`.

MySQL Validation

MySQL Unique & try/catch()

- you can define columns in MySQL as being unique
  - won't allow duplicated data
  - if you define multiple columns it will use the multiple values as a combination comparison
  - maintains data integrity
    - you don't want multiple artists with the same name as associating the songs become more difficult
- using the PHP try/catch method, you can avoid your code 'blowing up'
  - you can view the specific error codes
  - the error code for duplication is 1062

```
              try {
                // prepare the SQL statment
                $sth = $dbh->prepare($sql);

                // bind the values
                $sth->bindParam(':artist_id', $artist_id);
                $sth->bindParam(':title', $title, PDO::PARAM_STR,
100);
                $sth->bindParam(':length', $length,
PDO::PARAM_STR);

                // execute the SQL
                $sth->execute();

                // close the DB connection
                $dbh = null;

                $_SESSION['success'] = "Song was added
successfully";
                header( 'Location: new_song.php' );
              } catch ( PDOException $e ) {
                // verify that the artist name isn't a duplicate
                if ( $e->errorInfo[1] == 1062 ) {
                  // redirect to our error page
                  header( 'Location: new_song_error.php' );
                }
              }
```

  - the first step is to attempt our code
  - next we catch any errors thrown by PDO
  - we check to see if the error is caused by duplication
  - if it is, we direct them to an error page

ACTIVITY - Server-Side Validation

add_artist.php
add_song.php

ACTIVITY - PRG (Post, Redirect, Get)

new_artist.php -> add_artist.php -> artists.php
new_song.php -> add_song.php -> new_song.php

PROJECT 01 - Discussion

Objective

- demonstrate the first two parts of CRUD

    - CREATE
    - READ
- demonstrate the understanding of one-to-many relationship in an application environment
- demonstrate validating and sanitizing user input
- demonstrate working with POST and GET HTTP methods

Example Project

http://georgian.shaunmckinnon.ca/project-01/new_category.php

Project Parts

1. MySQL
    1. Tables
        - a parent table containing 3+ data columns, and an unique ID column that is auto-incremented
            - ie: Actors, Categories, Subjects
        - a child table containing 3+ data columns, and an unique ID column that is auto-incremented
            - ie: Movies, Products, Classes
    2. Example Data
        - the parent table must contain 5+ rows of example data
        - the child table must contain 2+ rows foreach parent table relationship
            - if the parent table has 5 rows, the child table will have 10
    3. Foreign Keys
        - a foreign key must be created in the child table referencing the parent table's ID column
        - make sure the ON DELETE CASCADE and ON UPDATE CASCADE to ensure data integrity
    4. Unique Column(s)
        - your parent table must have unique values (to follow the rules of normalization)
            - when you create your tables, you should follow the following syntax example

```
# parent table example
CREATE TABLE parent (
  id int(11) NOT NULL AUTO_INCREMENT,
  column_one varchar(50) NOT NULL,
  column_two varchar(100) DEFAULT NULL,
  column_three datetime DEFAULT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY unique_column (column_one)
);

# child table example
CREATE TABLE child (
  id int(11) NOT NULL AUTO_INCREMENT,
  column_one varchar(100) NOT NULL,
  column_two time DEFAULT NULL,
  parent_id int(11) NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY unique_columns (parent_id,
column_one),
  FOREIGN KEY (parent_id) REFERENCES
parent (id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

            - unique columns will help you maintain data integrity
2. PHP
    1. CREATE
        - a form for collecting user input for the parent table
            - must contain 3+ HTML form elements to represent the 3+ data columns in the database
            - the elements must be the most practical elements for the type of data you are collecting
        - a form for collecting user input for the child table

        - must contain 1 dropdown column populated with data from the parent table
            - must contain the ID as the value, and the name as the label
        - must contain 2+ HTML form elements for the remaining 2+ data columns in the database
    - the form must contain the two attributes action and method
    - both forms require a php page for processing the form and inserting the data into the database
        - each processing page must abide by the following:
            - a working connection to the database
            - binded parameters (bindParam)
            - a redirect to either a confirmation page, or to another page that contains either an error message or a success message

2. READ
    - an HTML page containing an HTML table for the parent table data
        - the table must have a header row labelling each column
        - it is required that a php script be used to select all the rows of data from the parent table in the database
        - a foreach loop is required to loop through the rows and output them within the HTML table
        - each parent row must contain a link to its child data
    - a linked HTML page, from the parent HTML table, that shows the following:
        - a section showing each column entry from the selected parent table row
        - t is required that a php script be used to select all the rows of data from the parent table in the database using the parent table id as a condition
        - an HTML table containing a header row with the child table column labels
        - a foreach loop is required to loop through the rows and output them within the HTML table
3. Navigation
    - the navigation must contain a link to the following pages
        - new_parent.php
        - new_child.php
        - parents.php
    - in the parents.php page you will have a link to parent_children.php, so you do not require a children.php page
4. Validate
    - you must have client-side validation for both the new_parent.php page and the new_child.php page
        - you may use HTML 5 validation or JavaScript validation (such as Parsley JS)
    - you must have server-side validation
        - you must validate against empty for required fields
        - you must sanitize strings to avoid the injection of client-side scripting
        - you must validate the data format of URLs, emails, telephone numbers, or any other required formats
        - you must validate using the try/catch method against duplicate entries
    - you must send the user to an error page or back to the form with a session message if the form hasn't been filled out properly
5. Interface
    - you must utilize Bootstrap, Foundation, or your own custom UI CSS so your page is organized and styled
6. Submission
    - you must provide a compressed version of your project uploaded to BlackBoard
    - you must provide a link to a working version of your project

Rubric

Criteria	Bonus	
Tables	**10** (5.68%) a parent table containing 3+ data columns, and an unique ID column that is auto-incremented ie: Actors, Categories, Subjects a child table containing 3+ data columns, and an unique ID column that is auto-	**1** (.57%) If you provide more than 3 columns per table

	incremented ie: Movies, Products, Classes	
Example Data	**5** (2.84%) the parent table must contain 5+ rows of example data the child table must contain 2+ rows foreach parent table relationship if the parent table has 5 rows, the child table will have 10	**1** (.57%) if you provide 10 or more parent row examples (with the corresponding children rows)
Foreign Keys	**5** (2.84%) a foreign key must be created in the child table referencing the parent table's ID column make sure the ON DELETE CASCADE and ON UPDATE CASCADE to ensure data integrity	**0** (0%)
Unique Column(s)	**5** (2.84%) your parent table must have unique values (to follow the rules of normalization) when you create your tables, you should follow the following syntax example unique columns will help you maintain data integrity NOTE: you need this column to do the validation check in the processing script for the parent table	**2** (1.14%) If you create unique columns in your child table to prevent duplication
PHP CREATE form for collecting user input for the parent table	**10** (5.68%) must contain 3+ HTML form elements to represent the 3+ data columns in the database the elements must be the most practical elements for the type of data you are collecting must contain the two attributes action and method must contain client-side validation	**0** (0%)
PHP CREATE a form for child table	**10** (5.68%) must contain 3+ HTML form elements to represent the 3+ data columns in the database the elements must be the most practical elements for the type of data you are collecting (the dynamic dropdown counts as one of the required elements) must contain client-side validation	**0** (0%)
PHP CREATE a dynamic dropdown element for child form	**10** (5.68%) must have a valid working database connection must select all of the rows from the database must use a php loop to output data from the results must use the id from the current row as the value must use the main column from the current row as the label	**0** (0%)
PHP CREATE parent processing page	**20** (11.36%) must have a working connection must use binded parameters (bindParam) must redirect to either a confirmation page, or to another page that contains a	**0** (0%)

	success message passed using the session super global must contain server-side validation (see validation requirements)	
PHP CREATE child processing page	**20** (11.36%) must have a working connection must use binded parameters (bindParam) must redirect to either a confirmation page, or to another page that contains a success message passed using the session super global must contain server-side validation (see validation requirements)	**0** (0%)
PHP processing pages validation	**20** (11.36%) must validate against empty required fields must validate data formats such as urls, emails, and custom patterns must validate against duplicate entries within the parent table (plausibly the child table) user must be sent to an error page if there is an error, or be returned to the form with an error message passed through the session super global	**0** (0%)
PHP form page validation	**5** (2.84%) must use HTML 5 or JavaScript validation	**0** (0%)
PHP READ parent table	**15** (8.52%) must contain an HTML table must have a header row labelling each column must contain a php block connecting to the database, selecting all the rows from the parent table and storing them in an array must contain a foreach loop looping through each element of the array and outputting them into HTML table rows each row must contain a link to its child data	**0** (0%)
PHP READ a parent's children table	**20** (11.36%) must contain a php block connecting to the database, selecting the row specific to the selected parent, and all the children corresponding to that parent a section showing all the details about the parent an HTML table showing all the children rows corresponding to the parent must have a header row labelling each column must contain a foreach loop looping through each element of the array and outputting them into HTML table rows	**0** (0%)
Navigation	**6** (3.41%) Must have navigation linking to the following pages: new_parent.php new_child.php parents.php NOTE: replace parent with the name of your	**1** (.57%) if you style the navigation with hover effects and active link logic

	parent table	
User Interface	**5** (2.84%) must be styled you may use Twitter Bootstrap or Foundation you may use custom CSS or any other UI framework	**1** (.57%) If it is responsive and can work functionally on a mobile device
Submission	**10** (5.68%) provide a link to a working copy (8 points) provide a compressed version of the project (2 points)	**5** (2.84%) IF you provide one of the following functionality options: 1) if an error occurs on form submission, you return the user to the form and repopulate the form with their original submitted values 2) if you create a connect.php file, a header.php file, and require() those within your pages to avoid duplicated code 3) if you use 5 undiscussed PHP library functions, such as array_map(), glob(), uniqid(), json_encode/decode(), count(), explode(), implode(), str_replace(), substr(), preg_replace(), preg_match(), and many others (see documentation) NOTE: you must indicate in your submission text that you used a unique function and point to where I can find it (file name, line number)