Week 9 Practical

CRUD

In programming, Create, Read, Update, and Destroy (CRUD) are the four basic functions of persistent storage. Persistent storage refers to the characteristic of state that outlives the process. We can make data outlive the process by storing it into a Database. We use CRUD in web development to build dynamic applications that allow users to create, modify, read, and delete content.

Create

To create new records (table row) in a database table, we use the INSERT statement. Here we'll be inserting four values into four specified columns into the specified table in our database.

INSERT INTO *tablename* (*col1*, *col2*, *col3*, *col4*) VALUES (*val1*, *val2*, *val3*, *val4*);

To insert rows with unique values in columns that aren't primary key values, we can use:

INSERT IGNORE INTO *tablename* (*col1*, *col2*, *col3*, *col4*) VALUES (*val1*, *val2*, *val3*, *val4*);

What IGNORE does is skips creating a new record if a column that was created to have UNIQUE values already has a record with an existing value. For example the ***users*** table was created to have the ***email*** column to be UNIQUE and has a record with ***example@gmail.com***. If we were to try to create a new record with the same email address our record would not be created because it already exists.

Read/Retrieve

Retrieve is often referred to as read, but makes more sense as we are selecting the data from our database to be retrieved and sent to our application. To retrieve records we use SELECT.

SELECT * FROM tablename

The statement above means we want to select all records and its column data from the targeted table. If we only wanted specific records we can use the WHERE clause.

SELECT * FROM *tablename* WHERE *col3* = *value*

Here we are now only selecting records where the third column contains a specified value. For example if we have a table called countries and wanted to select all records where the country name was 'Australia' we would use:

SELECT * FROM *countries* WHERE *name* = 'Australia'

Now if we only wanted the top 5 results we can tact on "LIMIT 5" at the end of the query.

SELECT * FROM *countries* WHERE *name* = 'Australia' LIMIT 5

And if we wanted to sort the records by a specific value we can use **ORDER BY *col* ASC** to order the rows in ascending order by the targeted column, or **ORDER BY *col* DESC** in descending order. For example, a table called ***comments*** has a column with the name ***votes*** with integer values. They are in no particular order, but we want to retrieve the data in ascending order, and return the top 50 results.

SELECT * FROM *comments* ORDER BY *votes* ASC LIMIT 50

So far we've been using the wildcard * to collect all data in the records. If we only wanted specified data we can specify what columns to select.

SELECT *col2, col4* FROM *tablename*

Here we'll be able to retrieve all data in the second and fourth columns in the targeted table.

Update

To update data in our records we use the UPDATE statement. You should always use a WHERE clause with your UPDATE statement, otherwise all records in your table will be affected.

Unlike SELECT, we cannot use the wildcard * to update all data in a record, so we need to specify the targeted columns. The basic syntax with a WHERE clause is as follows:

UPDATE *tablename* SET *col1* = *val1*, *col2* = *val2*, *col3* = *val3* WHERE *col2* = *value*;

The above would update the values inside of the first 3 columns of all records where the second column has the specified value.

Destroy

Destroying records is quite easy, but you can only destroy whole records and not just some data in the records. To destroy a record we use the DELETE statement. Be sure to use a WHERE clause otherwise all records will be affected, much like with UPDATE.

DELETE FROM *tablename* WHERE *col2* = *val2* AND *col5* = *val5*;

If you are not yet familiar with SQL statements then try executing some queries in your database manager. In phpmyadmin select your database then go to the 'SQL' tab. To get you started, in the text field type:

SELECT * FROM users

And click the Go button. You should see all user records if they exist.

Getting started

Download the folder blog from the repositiory and access it in the browser at http://localhost/blog. This will take you to the main page. For this practical you will be learning how to validate on the backend and using CRUD to build your dynamic site.

Go to http://localhost/blog/database.php. This script will create a new database with 4 tables. This week we will be using 2 tables: users and posts, so don't worry about the other 2.

Validating on the backend

The reason why we don't just rely on front-end validating is because it is not 100% secure. Web-savvy people can access the HTML and Javascript code through the browser and remove or alter the front-end validation to submit data that violates the rules put into place. However, they cannot access or alter the files on the server (unless they have access to it). We only use front-end validation for convenience for the user so they don't have to wait for a response, and back-end for security.

Error Messages

To display any caught validation errors we can store them into an array. The array $errors is passed by reference in the function **error_message** instead of passed as a value. To pass by reference we put an '**&**' symbol in front of the parameter. When a parameter is passed by reference, the caller and callee **use the same variable** for the parameter. This makes the change visible to the caller's variable. If we were to pass by value, then the caller and callee would have **two independent variables** with the same value, however the change would not be visible to the caller.

```php
$errors = [];
function error_message(&$errors, $message) {
    array_push($errors, $message);
}
```

The function **array_push** is used to insert new elements into the end of an array and takes two parameters: the parent array and the element to insert into the array.

Letters and Whitespace only

Here's an example on using the error_message function. The function **char_only** takes one parameter which should be a string and check to see if it only contains letters and whitespace. First we declare **$errors** as a **global** variable inside of the function to allow us to access the variable inside of the function. Next we use **preg_match** to check if the string as the correct characters. The function **preg_match** takes two parameters: a regex pattern and the string. If the condition is met then it will return **true**.

```php
function char_only($string) {
    global $errors;
    if (!preg_match("/^[a-zA-Z ]*$/", $string)) {
        error_message($errors, "Only letters and white space allowed");
    }
    return true;
}
```

In this example if the condition is false then we want to update the error array and place our error message.

Validating Email

PHP has a native function that allows us to validate email formats. The function **fliter_var** takes two parameters: the value to validate, and the type of validation. FILTER_VALIDATE_EMAIL tells the function that we want to validate the value against and email format. If it validates then it returns **true**.

```php
function validate_email($email) {
    global $errors;
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        error_message($errors, "Invalid email format");
    }
    return true;
}
```

Exercise 1 - Validating user registration

Using what you know now, use the two functions validate_email and char_only to validate the user registration. Add these functions in functions.inc and create 2 new functions called **validate_first_name** and **validate_last_name**. Inside of these functions you should be checking if the length is greater than or equal to 2 characters, less than or equal to 2 characters, and if they only have letters and whitespace.

If validation fails, then do not submit the data.

Exercise 2 – Auto-fill data on fail

When a submission fails due to validation then the user's input should be displayed in the input fields as it was before submission to prevent the user retyping everything.

To do this, add <?= $_POST['name'] ?? ''; ?> inside of the value attributes like so

```php
<div class="form-group">
    <label>First name</label>
    <input class="form-control" type="text" name="first_name" value="<?= $_POST['first_name'] ?? ''; ?>" />
</div>
```

This works the same way as a ternary operator, however it checks to see if the current value is set. The operator is read from left to right and you can add as many as you'd like, but will execute as soon as the first set value is found.

Add these to all input fields that require validation in your registeration form.

PDO – INSERTING AND SELECTING

The general syntax for using PDO is as follows:

```php
$query = "SELECT id, first_name, last_name FROM users WHERE email = :email";
// Prepare statement to prevent SQL injections
$stmt = $database->prepare($query);
// Bind parameters
$stmt->bindParam(':email', $email);
// Execute query
$stmt->execute();
// Check if there were any rows affected by the query
if ($stmt->rowCount() > 0) {
    // Fetch the affected row
    return $stmt->fetch();
} else {
    error_message($errors, "Email and password combination is incorrect");
    return false;
}
```

We first create a SQL query to process so we can create, read, update, or delete data from the database. We then pass the query through the prepare method to prevent SQL injections from occurring. Following that we need to bind parameters. With PDO, SQL parameter values are denoted as ":parameter". We can bind parameters one of two ways - using the **bindParam** method as shown above, or by using an associative array like so:

```php
$array = [
    ":first_name" => "John",
    ":email" => "test@example.com",
    ":date_time" => "12/10/2016"
];

$stmt->execute($array);
```

If you do use an associative array, then you will have to pass the array into the execute method.

$stmt->rowCount() returns the number of rows that were affected by statement. So if we inserted data and it was successful then it should return 1. If we selected everything from the table then it should return however many rows there are in the table.

For SELECT we need to actually fetch the data for use. We can do this by using fetch() and fetchAll(). Both will return an associative array where the keys are the names of the table columns.

```php
// For one row
$stmt->fetch();
// For multiple rows
$stmt->fetchAll();
```

Exercise 3 – Logging In

Create a new script inside of includes/scripts/login.inc to login as a user. You don't have to worry about storing the user ID or enforcing privileges this week. In your script you need to:

- Get the salt based of the email provided
- Encrypt the provided password the same way we did it in register.inc.
- Check to see if there is an existing record with the email "foobar@gmail.com" and password "password".
- Be sure to display the error message and prevent the user from redirecting to index.php if it fails.

Exercise 4 – Creating a new post

Now that you've had a bit of practice with PDO and validation, you are to now create a script that allows a user to submit a new post in post.php. The validation rules include:

- Title must not be left blank
- Title must be 100 characters at most
- Post must not be left blank
- Post must be 1000 characters at most

Check the post table to see what data you should be submitting. You can hardcode the user ID for this exercise.