Project: Code Snippets

Table of Contents

In the following exercise, you will develop a database for your code snippets. Apart from the practice, you may find this database a useful resource for your development.

To begin with, you will need to create a new database called "snippets" and a new table, also called "snippets".

You can use the following structure for your table[1]:

```
CREATE  TABLE snippets (
  id INT NOT NULL AUTO_INCREMENT,
  collection INT NULL DEFAULT NULL,
  owner INT NULL DEFAULT NULL,
  title VARCHAR(48) NOT NULL,
  content TEXT NULL DEFAULT NULL,
  entered DATETIME NULL,
  modified DATETIME NULL,
  PRIMARY KEY (id)
) ENGINE = InnoDB;
```

The fields are as follows:

`id`	The autoincremented primary key
`title`	A descriptive title
`content`	The text of the snippet
`entered`	The date the snippet was created
`modified`	The date the snippet was last update

The two fields, `collection` and `owner` will not be used at this stage, but will allow a future refinement involving multiple collections and contributors.

[1]This SQL statement is for MYSQL. For testing purposes, a file called `snippets.sqlite` is included, which allows you use the built-in SQLite database.

The Page

The index.php page contains mainly a single form with multiple submit buttons.

```
<form action="" method="post">
   <div id="titles">
      <select name="snippets" size="16">
         <option value="0">New Snippet</option>
         <!-- Snippet Titles -->
      </select>
      <button type="submit" name="select">Select Snippet</button>
   </div>

   <div id="editnote">
      <p><input type="text" name="id" value="<?php print $id; ?>" /></p>
      <p><label for="addtitle">Title:</label><br/>
         <input type="text" name="title" id="title" class="text" value="..." /></p>
      <p><label for="content">Content:</label><br/>
         <textarea name="content" id="content" cols="20" rows="12">...</textarea></p>
      <div id="control">

         <!-- if new -->
            <p><button type="submit" name="insert">Add</button></p>
         <!-- else -->
            <p><button type="submit" name="update">Update</button>
                  <button name="delete">Delete</button></p>
         <!-- end if -->
      </div>
   </div>
</form>
```

The select element will contain a list of existing snippet titles to select one to be edited or deleted. Its first element (New Snippet) will be used to insert a new snippet.

The text box and text area will contain the data from an existing snippet for editing or reviewing.

The control area will contain either an insert button, if a new one is selected, or update and delete buttons if an existing one is selected.

Connecting to the Database

Create a file called `db.php`, and save it into your includes folder.

The connection script is a standard connection using PDO. We will use the variable `$pdo` in future scripts.

Enter the following[2]:

```
<?php
   $database='snippets';
   $user='snippets';       //   or whatever
   $password='password';   //   or whatever
   //$dsn="sqlite:includes/$database.sqlite";
   $dsn="mysql:host=localhost;dbname=$database";
   try {
      //$pdo = new PDO($dsn);
      $pdo = new PDO($dsn,$user,$password);
   } catch(PDOException $e) {
      die ("Cannot connect to database $database");
   }
?>
```

At the top of your `index.php` file, include the following:

```
<?php
   require_once('includes/db.php');
?>
```

[2]This code is for a connection to MYSQL. If you are using the sample `snippets.sqlite` database, you can use the commented code instead (the `$dsn` and `$pdo` assignment statements). Don't forget to comment out the MYSQL versions.

Adding a Record

The form contains a button named “insert”. Currently the update and delete buttons are also visible, but later we will choose between them.

To test whether the form has been submitted via the insert button, test for its existence in the $_POST array.

```
<?php
  require_once('includes/db.php');

  if(isset($_POST['insert'])) {

  }
?>
```

We don’t care what its actual value is, just whether it has been set.

First, we will read in the text fields. Just in case, we will also trim them, removing spaces at the beginning or end:

```
if(isset($_POST['insert'])) {
  $title=trim($_POST['title']);
  $content=trim($_POST['content']);
}
```

(At this point, we should check whether they have been filled, but we will deal with that later).

Because the data comes from the user, it should be treated with care. All outside data should be regarded as a potential SQL injection attack, but is easily managed using prepared statements.

The SQL for inserting the data is:

```
$sql='INSERT INTO snippets(title,content) VALUES(?,?);';
```

The question mark place holders will be replaced with data after the statement has been prepared, preventing the data from being misinterpreted as additional SQL.

We will now prepare the statement:

```
$pdoStatement=$pdo->prepare($sql);
```

Although you can bind the data separately to the statement at this point, it is much simpler to bind the data when you execute the prepared statement. This is done by passing an array of values:

```
$pdoStatement->execute(array($title,$content));
```

Remember that the data must be in an array (even if there is only one value), every question mark must have a corresponding value, and that the values must be in the correct order.

This will give us:

```
    if(isset($_POST['insert'])) {
        $title=trim($_POST['title']);
        $content=trim($_POST['content']);

        $sql='INSERT INTO snippets(title,content) VALUES(?,?);';
        $pdoStatement=$pdo->prepare($sql);
        $pdoStatement->execute(array($title,$content));

}
```

For convenience, we will want a copy of the id of the new record. The id will be used for display later. However, since the id is and auto incremented value, we will need to get the value from the database. This is given by the `lastInsertId()` method of PDO:

```
$id=$pdo->lastInsertId();
```

The finished code reads:

```
if(isset($_POST['insert'])) {
    $title=trim($_POST['title']);
    $content=trim($_POST['content']);
    $sql='INSERT INTO snippets(title,content) VALUES(?,?);';
    $pdoStatement=$pdo->prepare($sql);
    $pdoStatement->execute(array($title,$content));
    $id=$pdo->lastInsertId();
}
```

Populating the List

Now that we have something in the table, we will want to see it.

We will create a variable called `$snippets`. This will contain the option elements to go into the select element in the form.

An option element has this form:

```
<option value="...">...</option>
```

The text content of the element is what the user sees. The value of the element is what will be posted. In our case, the text will be the title of the snippet, while the value will be the id.

To begin with, this array will be an empty array.

Towards the end of the PHP block (after the processing of the form data), add the following:

```
<?php
   …
   $snippets=array();
?>
```

Now, we will read the table for a list of ids and titles.

```
$sql='SELECT id,title FROM snippets';
```

Since we are not working with user data, we can query the database without having to prepare a statement first:

```
$pdo->query($sql)
```

This will give us a result set (which is technically a `PDOStatement`), which we can assign to a variable:

```
$results=$pdo->query($sql);
```

The results can then be iterated like an array:

```
foreach($results as $row) {

}
```

More simply, you do not have to use an intervening variable, and you can iterate through the query result set directly:

```
foreach($pdo->query($sql) as $row) {

}
```

Each `$row` variable will be an array with the data. Although the array also contains numbered elements with the same data, we will be using the associative elements:

```
foreach($pdo->query($sql) as $row) {
     $snippets[]="<option value=\"$row[id]\" $selected>$row[title]</option>";
}
```

Since the associative elements are placed inside the string, *you do not quote the keys*, even though they are text keys!

The `$snippets[]=` notation means that a new value will be added to the end of the array. This is also known as pushing a value onto the array.

When the array is complete, we will convert into a string, since it is this string which must be printed into the HTML.

The `implode()` function (also known as `join()`) takes two parameters: the glue (what comes between the values), and the array.

```
$snippets=implode('',$array);
```

In this case, we want nothing between the values (in HTML the only thing permissible between option elements is space).

Note that by re-using the `$snippets` variable, we have replace an array with a string. However, we are still using the variable for the same purpose.

The completed code should look like this:

```
$snippets=array();
$sql='SELECT id,title FROM snippets';
foreach($pdo->query($sql) as $row) {
   $selected= $id==$row['id'] ? ' selected="selected"' : '';
   $snippets[]="<option value=\"$row[id]\" $selected>$row[title]</option>";
}
$snippets=implode('',$snippets);
```

Displaying an Existing Snippet

Although part of the same form as the rest, the select element has a button closely associated with it.

Later we will include some JavaScript which will replace this button with an action. This is not required, especially if JavaScript has been turned off, so the JavaScript will only be applied if available.

We will put the code before the other input handling code, so we can use the results in later code if we need.

The select element is named "snippets". To check whether one has been selected, we will check the post array. If one has been selected, we will set the id:

```
<?php
   require_once(...);

   if(isset($_POST['snippets'])) {
      $id=...;
   }
   …
?>
```

Remember that the first option, New Snippet, has a value of 0. Since the snippet ids start at 1, this will (later) be use to indicate that we want a new record.

As a zero, 0, may also have been selected, we really want to test whether the value is non-zero, rather than whether it has been set:

```
if($_POST['snippets']) ...
```

However, if no value has been selected at all, the above would generate an error, so we add the error suppression operator (`@`):

```
if(@$_POST['snippets']) {
   $id=...;
}
```

The error suppression operator ignores the potential error, and can be used in cases where "no value" is an acceptable alternative.

The above could be interpreted as "if snippets, where it exists, is non-zero, then …".

If none has been selected, then we will set the `$id` to 0:

```
if(@$_POST['snippets']) {

   }
else $id=0;
```

Although the data should have come from a form, and we have constructed this data ourselves, it is still best convert the value to an integer:

```
if(@$_POST['snippets']) {
   $id=intval($_POST['snippets']);

}
```

The `intval()` function will convert as much as possible to an integer, stopping at the first invalid character. If there is no valid character, the result will be 0.

The SQL statement will select the title and content from the snippets table matching the id. For good measure, we will also count the results.

Since the id is unique, you will never get more than one row, and a corresponding count of 0. An invalid id would result in an empty row. However, if you include a count, you will get one row again, with a count of 0, and NULL for the remaining fields. This will simplify testing.

```
if(isset($_POST['snippets'])) {
   $id=intval($_POST['snippets']);
   $sql="SELECT count(*), title, content FROM snippets WHERE id=$id;";

}
```

It is perfectly safe to include the user data into this SQL statement, since the `intval()` function has converted into an integer, making it incapable of being interpreted as additional SQL.

Since the SQL is safe, we can run the query directly:

```
$pdo->query($sql)
```

Since there will only be exactly one row, we can fetch it immediately:

```
$pdo->query($sql)->fetch();
```

The result, as usual, will be an array of values. This time, it is more convenient to use the numbered keys, since we can use them in a list():

```
list($count,$title,$content)=$pdo->query($sql)->fetch();
```

The `list()` operator (this is not technically a function, though it looks like one) will copy the numbered elements of an array into the listed variables. It is a convenient way of working with multiple variables.

In the event of an invalid id, we will have a `$count` of 0, and NULL for the `$title` and `$content`, which we will deal with presently.

If the count is 0, we will set the id to 0, which will also indicate a new entry:

```
if(!$count) $id=0;
```

This will give us:

```
if(isset($_POST['snippets'])) {
   $id=intval($_POST['snippets']);
   $sql="SELECT count(*), title, content FROM snippets WHERE id=$id;";
   list($count,$title,$content)=$pdo->query($sql)->fetch();
   if(!$count) $id=0;
}
```

If a valid snippet has not been selected, and hence the id is 0, we will set the title and content to empty strings:

```
else $id=0;
if(!$id) $title=$content='';
```

Note that is possible to assign the empty string to both variables in one statement.

Displaying the Content on the Form

Having read the title and content, or set them to empty strings, we will display them on the form.

For a text box, and input element, you display the old or default value in the value attribute:

```
<input type="text" name="..." value="..." />
```

In this case we will insert a PHP print statement of the variable. To be safe we will also include the error suppression operator, in case the value was not set:

```
<input type="text" name="title" value="<?php print @$title; ?>" />
```

For a text area, the old or default value is between the opoening and closing tags:

```
<textarea name="...">...</textarea>
```

In this case we will use:

```
<textarea name="content"><?php print @$content; ?></textarea>
```

Finally, though this will not be displayed, we will need to store the id. This will be used when later we update or delete the record, and so the record will need to be identified.

To store a value in a form without user intervention, we use a hidden field. This field is not displayed on the page (though it is still visible in the page source).

```
<input type="hidden" name="..." value="..." />
```

Generally you would hard code the value, since the user is unable to enter the value otherwise. A JavaScript might also be used to set the value.

In our case, we will set the hidden field for the id:

```
<input type="hidden" name="id" value="<?php print @$id; ?>" />
```

The Submit Buttons

Although a form may have many submit buttons, you should still hide the buttons which are not relevant, or which might interfere with you application.

In this case, we will display an insert button if the record is a new record, or the update and delete buttons if the record is an existing record.

The simplest way to distinguish between a new record and an existing record is with the id. A value of 0 will indicate a new record, while a non-0 id will indicate an existing record.

We have a `div` (`control`) which contains the various buttons.

PHP allows you to intersperse PHP blocks with ordinary HTML (or other text). Remember, that what is not inside a PHP block is regarded as PHP output.

A useful structure is:

```
<?php if(...) { ?>
  <!-- PlanA -->
<?php } else { ?>
  <!-- PlanB -->
<?php } ?>
```

If the condition is true, then the PlanA text will be output. Otherwise the PlanB text will be output.

For this project, we will test whether the id is 0 (new record). If so, we will display the insert button. Otherwise we will display the update and delete buttons.

Enter the following:

```
<?php if(...) { ?>
  <!-- PlanA -->
<?php } else { ?>
  <!-- PlanB -->
<?php } ?>
```

Replace the condition with:

```
<?php if(!$id) { ?>
```

This will be true if `$id` is not something, that is, it is zero.

Replace PlanA with the insert button, and PlanB with the update and delete buttons.

The code should read:

```
<div id="control">
  <?php if(!$id) { ?>
    <p><button type="submit" name="insert">Add</button></p>
  <?php } else { ?>
    <p><button type="submit" name="update">Update</button>
      <button name="delete">Delete</button></p>
  <?php } ?>
</div>
```

Updating Records

Now that we can add and display records, we will look at changing them.

The key to updating is to have an id to identify the record being updated. This id comes from the selected snippet earlier, copied into the hidden id field.

To check whether a record is being updated, we check its submit button. This is logically placed after the test for insert, thought it doesn't have to be:

```
if(isset($_POST['insert'])) {
   ...
}
elseif(isset($_POST['update'])) {

}
```

Note that as you can only press one submit button, you do not need the `elseif` clause - you could simply have used another `if`. The `elseif` is only required when both conditions might be true, but you only want one to be executed. However, the use of `elseif` here helps to group the tests together.

Just as with the insert script, we will read in the title and content from the POST array. In addition, we will also read in the id, which comes from the hidden field previously set. This value will be processed through `intval()`:

```
elseif(isset($_POST['update'])) {
   $id=intval($_POST['id']);
   $title=trim($_POST['title']);
   $content=trim($_POST['content']);
}
```

As before, we should really check whether the title and content have been supplied.

To update a record, we use the SQL UPDATE command.

```
UPDATE ... SET ...=..., ...=...;
```

The UPDATE command is potentially disastrous to a database, since it will change all of the records unless you qualify it with a WHERE clause.

```
UPDATE ... SET ...=..., ...=... WHERE ...=...;
```

In this case we will set new values for the title and the content, and use the WHERE clause to select for the record's id:

```
UPDATE snippets SET title=...,content=... WHERE id=...;
```

Since we will be adding user data, we will need to prepare the statement first, and execute it with data later. The values will be set with question mark place holders:

```
elseif(isset($_POST['update'])) {
   …
   $sql='UPDATE snippets SET title=?,content=? WHERE id=?;';
   $pdoStatement=$pdo->prepare($sql);

}
```

Finally, we execute the command with data:

```
elseif(isset($_POST['update'])) {
   …

   $pdoStatement->execute(array($title,$content,$id));
}
```

The finished script should look like this:

```
elseif(isset($_POST['update'])) {
   $id=intval($_POST['id']);
   $title=trim($_POST['title']);
   $content=trim($_POST['content']);
   $sql='UPDATE snippets SET title=?,content=? WHERE id=?;';
   $pdoStatement=$pdo->prepare($sql) or die('oops');
   $pdoStatement->execute(array($title,$content,$id));
}
```

Deleting Records

Now we can delete records using the SQL DELETE command. Although this command is also potentially disastrous for the same reasons, some would argue that it is better to lose your data than to have bad data.

We will add another `elseif` block underneath the others. As before, the `elseif` is not strictly necessary, since we can only press one submit button.

```
elseif(isset($_POST['delete'])) {

}
```

Again, as before, we will extract the id of the record:

```
elseif(isset($_POST['delete'])) {
   $id=intval($_POST['id']);

}
```

To delete a record, we use the SQL DELETE command:

```
DELETE FROM ...       DELETE FROM ... WHERE ...
```

The first form will delete all records. The second will delete only selected records.

Since the id has been processed through `intval()`, it cannot contain any harmful SQL. This means it is safe to include it directly into the SQL string, and to execute it directly:

```
$sql="DELETE FROM snippets WHERE id=$id;";
$pdo->exec($sql);
```

Note that unlike SELECT statements, use the `exec()` method of PDO, not the query() method.

Finally, we reset the id, title and content:

```
$id=0;  $title=$content='';
```

The final code should look like this:

```
elseif(isset($_POST['delete'])) {
   $id=intval($_POST['id']);
   $sql="DELETE FROM snippets WHERE id=$id;";
   $pdo->exec($sql);
   $id=0;
   $title=$content='';
}
```

Finishing Off

The project will now work and can be used for storing your code snippets or other useful notes. Hwever there are a few futures which might make your project more useful.

Data Validation

When we submit or update a snippet, you will need to check whether you have a title, and depending on your policy, content (you may wish to allow empty content if you plan on filling it later).

To begin with, we will have a variable called $error, and set it to an empty string. This variable is best set just before the code which checks the insert and update operations:

```
$error='';
if(isset($_POST['insert'])) {

}
elseif(isset($_POST['update'])) {

}
```

After the `title` and `content` variables have been set in the `insert` code, you will need to test whether they both have a non-empty value. The if structure should embrace the remaining code:

```
if(isset($_POST['insert'])) {
   $title=trim($_POST['title']);
   $content=trim($_POST['content']);
   if($title && $content) {
      $sql='INSERT INTO snippets(title,content) VALUES(?,?);';
      $pdoStatement=$pdo->prepare($sql) or die('oops');
      $pdoStatement->execute(array($title,$content));
      $id=$pdo->lastInsertId();
   }
}
```

Note that a convenient shorthand to test whether a variable has an empty value is simply to test the variable itself. A non-empty string evaluates as `true`, while an empty string evaluates as `false`.

Note also the use of the and operator (&&).

If either the title or content are empty, the else clause will assign an error message:

```
if($title && $content) {

} else $error='<p class="error">Please complete the Title & Contents</p>';
```

The same needs to be done for the `update` code.

Finally, the error message needs to be displayed. In the form, just below the paragraph with the hidden form element, add the following PHP block:

```
<p><input type="hidden" name="id" value="<?php print $id; ?>" /></p>
<?php print $error; ?>
```

Since the $error variable will have been set to an empty string or to an error paragraph, the print statement will safely print the results.

Highlighting the Current Record

When you select a snippet title and submit it, the newly refreshed page will not highlight the selected title in the list.

To highlight a selected option in HTML, you can use the selected attribute:

```
<option value="..." selected="selected">...</option>
```

When we generate the list of options, we will insert the selected attribute when (or if) the id of the current item matches the id selected with the previous submit. One simple way of inserting an occasional attribute is to define a variable, and set it using the conditional (ternary) operator:

```
$selected = test ? planA : planB;
```

The conditional operator uses either the first or the second value, depending on whether the test evaluates as true.

The `id` selected will be in the variable `$i`, which will be `0` if no item has been selected. We will compare this against the current row's id:

```
$id==$row['id']
```

If true, we will use the string `' selected="selected"'`; otherwise we will use the empty string `''`. Note the space at the beginning of the first string. This makes it easier to insert into the HTML which requires a space between attributes.

```
$selected = $id==$row['id'] ? ' selected="selected"' : '';
```

It remains only to insert this string into the code for the option:

```
$snippets[]="<option value=\"$row[id]\" $selected>$row[title]</option>";
```

The resulting code should look like this:

```
foreach($pdo->query($sql) as $row) {
   $selected= $id==$row['id'] ? ' selected="selected"' : '';
   $snippets[]="<option value=\"$row[id]\" $selected>$row[title]</option>";
}
```

Auto Submit

Currently, the snippets list comes with a submit button. Using JavaScript, you can have the snippets list auto-submit. The following JavaScript will enable auto-submit, as well as hide the submit button:

```
<script type="text/javascript">
   window.onload=function() {
      document.forms[0].select.style.visibility='hidden';
      document.forms[0].snippets.onchange=function() {document.forms[0].submit(); };
   }
</script>
```