Create, Read, Update and Delete (CRUD)

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

English

0:09
So now, it's time for the payoff. You've been sort of working through this whole section on PHP, and SQL, and redirect, and all these things. And now, we're going to put it all together to build sort of the canonical basic application that does one of everything. We call this the CRUD application because we're going to create, we're going to read, we're going to update, and we're going to delete. Of course, we're going to select to do read, but CRUD sounds so much better. So we've actually done most of it. And so it's as much now just putting it all together and doing it right with POST redirect, and flash messages, and all those things. So that you'll just kind of see how we put this all together to do a basic single table application where you can insert data, delete data, update data, and delete data.

1:02
So the last time we kind of looked at our program, we had a view, the read part, we had add, and we had delete. We did not have update in it, right? But we all had this one page that did everything. And we weren't doing POST redirects or anything, okay? And so in general, you tend not in real applications to slam all this stuff into one big blobby user interface. So you sort of move this into multiple files. We call this refactoring.

1:33
Where you kind of got it all working, but it's sort of in the wrong place. And so we're going to keep it working, and we're going to move it around. And now that we can redirect back and forth, we can put make files that are sort of more simple and consistent and do one thing, okay? So we're refactoring our code, then we're going to add the edit, because we didn't have edit in any of these other things, so that's really what we're doing. So, what we're going to do is we're going to have a couple of files, four or five, our index file is going to be the main list of things. If you need a detail sort of, if there was too many columns in all of these little examples, I have too few columns. So we just have an index that shows everything. We're going to have a separate add file. We're going to have a separate delete file. And we're going to have a separate edit file. And so the kind of the starting point for all of these things is index.php. And we'll just put them in a directory, put them in a folder. So we'll have the CRUD folder, and we'll have add, delete, edit, and pdo.php, that's just to make our database connection. And we've been doing that now for long enough that hopefully that pdo for you has just been working fine for you, okay? So here is how our code is going to work, again, think of it as refactoring. We're going to have this List view, which is the index, it's the main view of things. It's going to have an Add button, an Edit button, and a Delete button. Now, these are not forms anymore, these are anchor tags, <a href..>'s. These are anchor tags, not forms, so they're going to go to new pages, okay? So the first thing we'll take a look at is how we do an Add New? So we're going to go now to add.php. And in add.php, the view part is going to just be a simple form. We have a Cancel button, which is just to go back to index.php. Or we type in Name, Email, and Password and then submit it. So let's take a look at the index.php. We're going to start with pdo.php to get our database set up. We're going to start the session. And then there is no other model up there other than those two things. And then the view starts, and we're going to put out the flash messages. And the flash messages are not the first time you come in. But when someone redirects to come into here, they will often set error or success. So it will be coming from somewhere. Coming from the add.php, delete.php, or edit.php, and we will get a message. So we just know that we'll might have to receive flash messages here. Now, I did sneak a little bit of select code down into my view that's kind of a no-no. I should have selected that above, but it will be okay. So in the view, we'll print out the table. We've done this before. We just print out the name, email, and the table rows, right? because these are just grids, row, end of row td /td, htmlentities we are good about that. And so now, what we're going to do, is in this last little action column, the last column, which is the action column. We're going to basically put an anchor tag to go to a href php, and a get parameter of user_id equals another primary key, the user_id, edit.php, and delete.php. So we just transfer to those, but pass in whichever row we're working with. So we're going to add a GET parameter underneath each of those little edit and delete buttons with the GET parameter.

4:42
Okay, and so if we look at how this ultimately, when we take a look at the source code of this, it looks like that, edit.php?user_id=1, a GET parameter, delete.php?user_id=1, a GET parameter. And this one of course is the primary key of that particular row in the database. So we know which one we're going to delete, or we know which one we're going to edit.

5:06
So now, let's take a look at the add code.

5:09
So the add code is model view controller. It's really model view controller, model and controller is up here and view is down here. It all starts out blank, you type in all the stuff and you type Add New, it comes back in with a POST. POST data will be checked for password. We're going to do an insert, we've seen this before. Some of the early stuff, I put debug statements in. But debug statements here are dangerous, because we're going to have to redirect after this POST, so you gotta be careful. It's often better to log stuff if you put it out, the redirect won't work. And you can put debug statements in there if you're having trouble with your code, it's just don't expect the redirect to work and it'll complain. It'll say headers have already been sent, don't call header, which is okay, because you're actually debugging the code, so be careful. We have no debug, Unless you're sending it to the log. If you send it to the log, that does not count as output. But this is stuff we've been doing before. We just created an INSERT statement. We do a prepare, so it handles these little placeholder guys. And then we do an execute where we map the placeholders to the data from the POST data. And then we set a success message, and then we redirect back to index.php, we're in add.php. add.php and then we go to index.php, after the successful post. So we don't come back to this thing after the successful POST, we go back to add.php after the successful POST, right? And so the pattern here is, we click Add New, we go into add.php, we type in the stuff. Replaced Add New, it POSTs in here, it adds the stuff in the database. And then it sends the session success variable and redirects back to index.php. And then index.php pulls this out of session, shows it once as flash with this little bit of code in index.php, and then gets rid of it, wipes it out. So if you hit refresh at this point, then you won't see that message the second time because that was a flash message. So you see how we wrote the code in index.php expecting to be redirected back to.

7:12
So we knew that add was coming back to us, and that add would send us a message in success. And so we just put those four lines up there. If anyone sends a message to us, we get a success.

7:24
So that's how add works.

7:26
Okay, so let's take a look at delete.php. So remember, delete.php, we link to delete.php and we give the primary key. So now, we have a GET request, okay? So when we come in first time, we're going to GET request, okay? So after we press the Delete button inside the index.php, we come here in a GET request. So none of this stuff is here, right? So it goes in, there's no POST data set. But now, we have a bit of the model code. We're still in the model now, right? The line between model and view is down here. We're still in the model. So the thing we're going to do in this model is we're going to actually select the data that's going to feed the display. We're also going to check to see If there is a bad value here because this could be like 400 and be wrong. So what we're going to do is take a SELECT name, user_id FROM users where user_id = this and then that's then we're going to take GET user_id and put that in there. And then we're do in a fetch. And if we get a row, that means there was a row, and then we have good data, and that data is an associative array of user_id and name in the $row variable. Or false, false means, we didn't get a row, it doesn't blow up. Remember, not getting a row on a SELECT statement is not a fatal error, it's exactly what you want, you're told you got nothing. Your SELECT WHERE clause didn't find anything. If this is WHERE 4000, you will get 0 rows, but it won't blow up. So it's not a failure to get no rows. But in this case, it's kind of a bug, which means we've been passed a bad value here. So what we're going to do is, we're going to say something nasty. We're going to say in the error in the session, the flash message we would like to have display is "Bad value for user_id". And then we're going to redirect back to index.php, which we'll see that and print out a red message. And then we return, right. Like I said, you'll do these three lines, putting a flash message in. Redirecting and then quitting, getting out of the script, because we don't want to fall through you, that's why you have a return, because we don't want to produce this output. Now, if the row was good, now we have we fall in, right. We fall in to view with $row, defined row name, htmlentities(), yey, so this row['name'], did it come from the user? In this code, it's actually coming from the database. But it originally came from the user. And so when we inserted in the add, we didn't call htmlentities(). Let me go back to that, make that point a little bit. So here, when Sarah put in her email, right, we did not put htmlentities() on this. The pattern is generally the case, that if there is data from the user going into the database, and then coming out of the database, we don't call htmlentities() here, we only call it when it comes out. Okay, so we don't put the data in, we put the raw data in double quotes, single quotes, everything and all. And at this point, we have just retrieved that same thing from the database. it's the raw data the user entered. We put it in raw and then we know it's dangerous, so we're going to say htmlentities. And the reason you do this is, you don't want to call htmlentities twice, you never want to do it. So you have to have a very strict policy that says, we're not going to call htmlentities before we put the data in the database, we're going to call htmlentities after we take the data out of the database.

10:53
So even though this came from the database, it originally came from the user, right? Sara typed that in, she typed her stuff in, we don't know if we can trust it or not. She typed it in,we put it in a database without touching it and then it came out of the database to us, and so we have to call htmlentities on it, and away you go. Now, the fact that we did a select is going to give us two things. First, it's going to give us to know whether or not get id is really a valid id or not. And second, it's going to give us the person name so we can a cute little confirmation dialogue. I mentioned before, there's a lot of user experience that always has delete confirmations, so at least you know what you're doing. And both is to avoid doing deletes on GETs, and to make sure you're deleting the thing that you want to do, and you're doing it in a POST. And so, we're going to have a Delete button here, and then we will have a hidden, let's hide that.

11:47
Right, we have a hidden variable here, and it's just the user's id which is just going to be 4. So that user_id is going to be 4, so that when we hit the submit button, that's going to come up here. And it's going to tell us if the Delete button was pressed and user_id is 4, and then it's going to run this delete code, which is the kind of code we've seen before. DELETE FROM users WHERE and then whatever that 4 number is going to be stuck in there. And then it runs this, we send a success message and we go back to index.php and we return.

12:16
So that's a delete, sweet?

12:21
So here's how that one works, right. We press the Delete button, I guess we're pressing Sarah's Delete buttons. We come into delete, we go here, we show it. And then we press the Delete button. This is a GET request, this is a POST request with the hidden data, this is the hidden, that's hidden, right? Hidden data, DELETE FROM users WHERE user_id = :zip, code, redirect, put the success message in, 'Record deleted', and then redirect index.php. And index.php, we got these lines of code that put this message up. And then wipe it out of the session, so it's a flash message. And that's how we go in a successful delete from delete.php through the model code doing the update to the database. And then the controller is deciding to send us back to index.php with a message. So that's read, insert, delete. The one thing we haven't done before is edit.

13:23
So this is edit.php. And this is how edit.php works. edit.php looks a lot like add. And as we get later, I just start writing code that has more if statements in it. But for now, we keep our edit separate from add.

13:37
because you really want to do update for edits, right, that kind of big difference is we use the UPDATE statement for edit because the record is already there UPDATE with the WHERE clause. And the UPDATE is quite different than INSERT, there's some cool things in SQL where you can say, INSERT blah, blah, blah, blah, blah, ON DUPLICATE KEY UPDATE, but we haven't quite set that up yet. So for now we're going to keep our edit and our add very separate. So here's edit.php. One thing is, you don't see it right here, you've got a GET parameter of user_id, okay? So you know which user you're supposed to edit. So edit is not all users, it's editing one user. So you come in with a GET request, and you have the user_id that you're supposed to do. You're going to open your database connections, start the session. We don't have any POST data the first time in, right? Now this is actually the code right here is from delete. It's from the delete, because both the edit and the delete have the problem, and you can put a function that do this and we will later. You both have the problem to check, I want to get the old data, and I want to make sure that the id is even right. I want to make sure that it's right. So we're going to do a SELECT, it's exactly the same thing. We're going to SELECT with a WHERE clause on that id, which is this GET parameter which comes in here. And if we get a row, we've got the old data, if we don't get a row, we set a sad face and send it back to index.php, with a flash, okay? So we'll come back to this in a second.

15:08
Let's go down to the view part that talks about how to produce that. So, I am going to basically create the variables $n $e and $p, which are just the POST htmlentities versions of name, email and password. You don't have to put htmlentities on user_id because you know it's an integer, right, it's in your database. That didn't come from the user, you've made it. You'll notice that sometimes, I don't put htmlentities around the success messages, because I am making those messages in my code. If there's user data in it, you've got to do htmlentities, but most of the messages I'm doing are just hard coded constants. So you don't have to put htmlentities on the user_id, because you know that's an integer, and it's always going to be an integer. Right, so I've got these variables, and then I fall into the view. This whole thing is the view.

15:56
Fall into the view and then I'm just going to put, it's pretty much the same as the add, except I put value=. And then I use <?= which is the short version of echo, basically. And that's going to print I was like, where is htmlentities? I just had a moment there, I'm like, did I forget htmlentities? I don't know, I did it right here Chuck. It is okay, it will be okay. Don't freak out man, chill.

16:20
Okay, so then this is Glenn with htmlentities. This is Glenn's email with htmlentities, right there. And this is the password, Glenn's password with htmlentities. He has a very bad choice of passwords. And then we have a hidden variable, which is right there that you can't see, which is the user_id value. And then of course, a submit button and an href to get out of here. So then we press this button.

16:44
And we press this button and it comes into here, because now these three things, name, email, and password, and user_id are set. So now we're going to write some SQL that you haven't seen, at least for some time now, it's an UPDATE statement. So UPDATEs look different than INSERTs. UPDATE users SET column equals placeholder, column equals placeholder, column equals placeholder. WHERE column equals placeholder. So we're saying, update the name, email, and password to new values from the POST data, where we've hit the right user_id. So then, we're going to prepare it and then we're going to execute it taking the name, email, password, and user_id. This one's coming from the hidden one.

17:25
The hidden field that's right here, is coming in and that's the user_id. We assume that works and we say, we're happy, success message, redirect back to index.php, and return out of edit.php, because the next the browser's going to do is do a GET request index.php. And so, this is pretty much how it works, right? We click on Glenn's Edit button, it does a SELECT to retrieve the old data. We htmlentity it, put it in there. We send this update, we click Update, it's a POST.

17:59
So these three things come in as POSTs, including the hidden user_id is hidden there. So that comes in.

18:08
He's ready hidden comes in there. We do the UPDATE that I just mentioned, we do the prepare, because we don't want SQL injection. So we do prepare execute. Don't get lazy and just concatenate this stuff, that would be bad, bad, bad. Don't do that, we know SQLI. None, we don't want any SQL injection. We want to do happy face, happy face. Use prepared statements, happy face. Placeholders, pdo, prepared statements, happy face. And so, then we just do the, we put the stuff in the session, and then we redirect to index.php, this is a GET request.

18:45
And then it's going to pull it out of session. And then it's going to wipe it out of session, so its a flash message. And away you go.

18:51
This is so awesome, because if you're understanding this, think of all the things that you actually had to know.

19:01
SQL, HTTP, GETs, POSTs, hidden fields. There's like model view controller, there's so many things. This, [LAUGH] and you should really understand this CRUD stuff, right? You should understand this lecture, if you're like, I'm going to sleep to this lecture and I'll pick it up later. No, it turns out, what I'm going to do is I'm going to assume you know every line of this lecture and going forward. I'm going to be like CRUD, boom, you're suppose to know it. Yeah, that's a this that piece of CRUD. Yeah, flash message. You're suppose to know this stuff from now on, because I'm going to change it. This is the simple one that you should understand. And when I start throwing things like AJAX and jQuery, and who knows what at you. I'm like, hey, this is just like the CRUD and you're like, what was CRUD? So watch this lecture, I know it's a longer than some of them, but watch this lecture a couple times. We have done a whole bunch of stuff in getting this database stuff figured out. And this CRUD is really, the moment you fully understand CRUD is the moment that you can start riffing on this pattern and building real applications.

Downloads

Lecture Video mp4

Subtitles (English) WebVTT

Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?