MySQL via PHP

CIS1152 Adv Web Dev

Steve Ruegsegger

Modified by Peter Chapin with permission

Outline

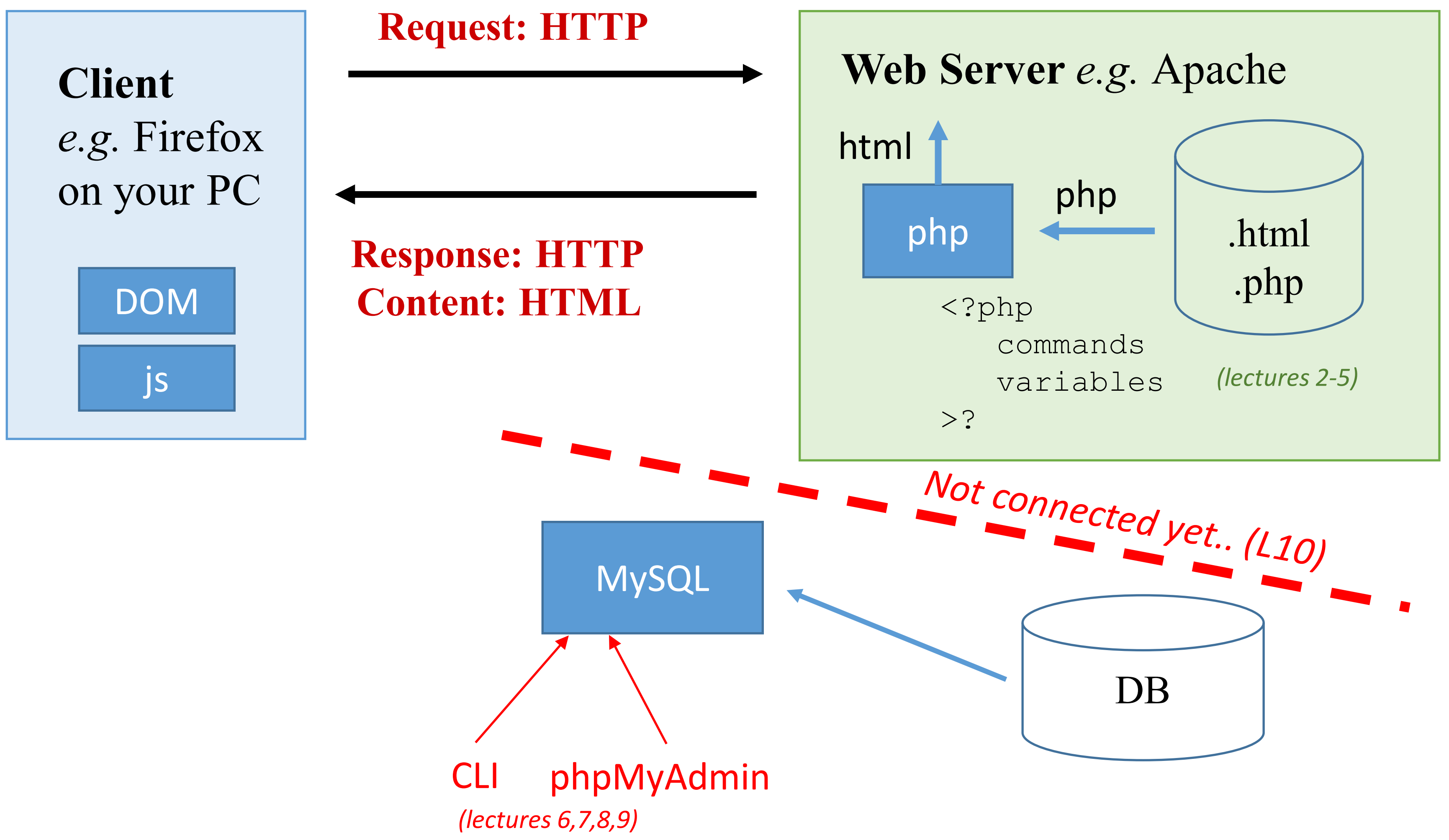
1. Back to the Big Picture
2. 4 ways to communicate SQL via PHP
3. mysqli part 1
   - Simple template
4. The *problem* with "simple template" in part 1
5. mysqli part 2
   - A more complicated, but safer template
6. mysqli part3
   - Reading a larger recordset from SELECT
7. Example code walkthru!

Big Picture

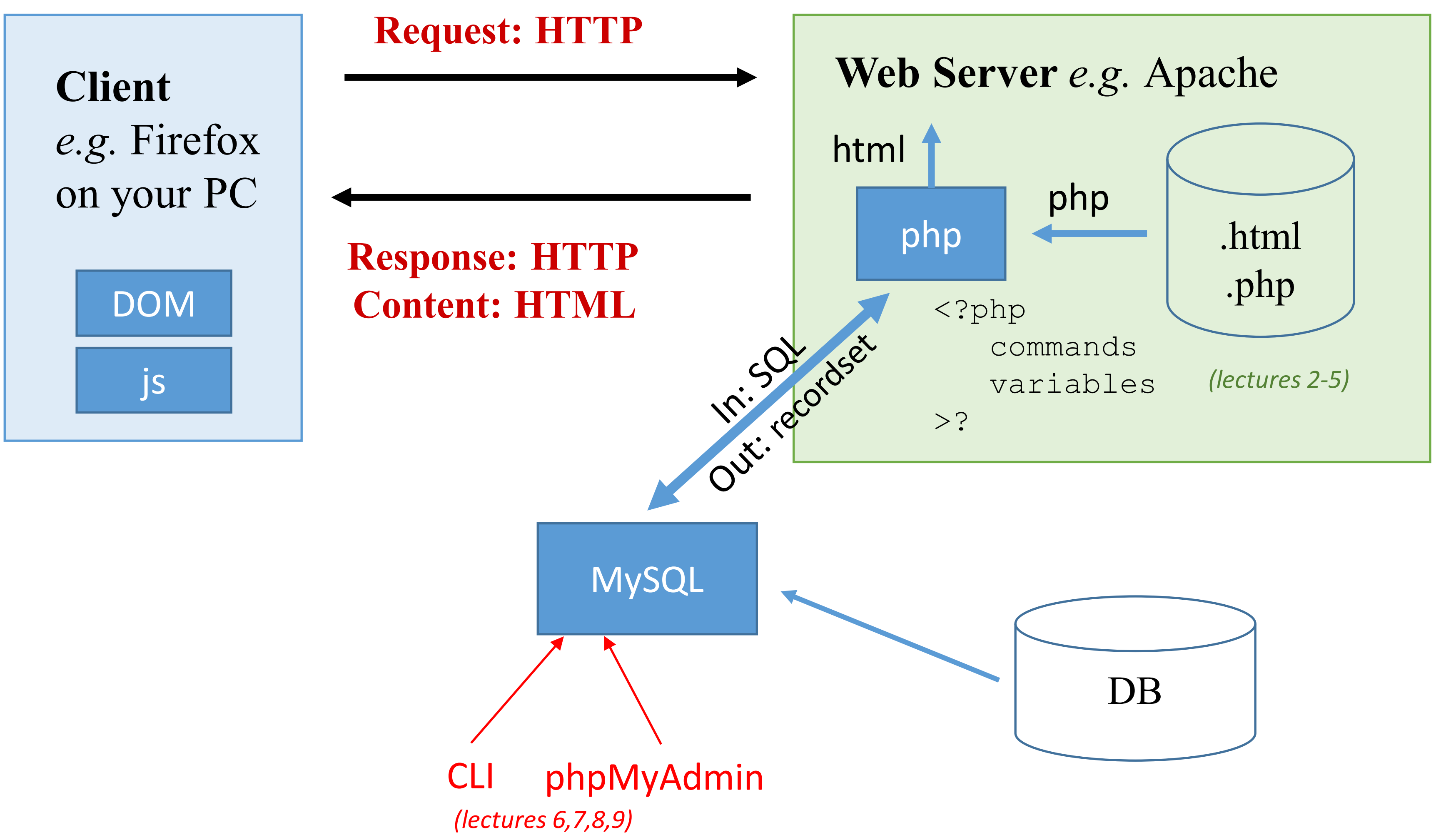
How did we get here?

Where are we going?

Three Tier Web Architecture



Three Tier Web Architecture



What's the goal here?

Simple:

- Within PHP,
- create SQL text to send to a DB,
- loop through the sql results one row at a time,
- accessing each sql col as a php variable,
- to be used to create HTML for a great end-user experience

4 ways to communicate SQL to PHP

Actually, there are 3

How do we send SQL to PHP???

- PHP has *always* been very good at sending SQL and receiving recordsets.
- Particularly to MySQL.
- As PHP has *evolved*, there have become multiple ways to do this:
  1. **mysql_*** routines. These are *deprecated*. Don't use these any more.
  2. **mysqli_*** routines. (The "i" means "*improved*")
  3. **mysqli** Object Oriented (oo) methods and objects of #2
  4. **PDO** – Portable Data Objects
- *Comments*
  - The first one is very old. *Don't use it*.
  - The mysqli is *pretty* new. PDO is *very* new.
  - You can use the procedural (#2) or OO (#3) versions of mysqli, they are really the *same* code
  - PDO is only OO. It's designed to work with *any* DBMS, not just MySQL
  - The books *both* present mysqli
  - Since we are using MySQL from XAMPP, let's just stick with mysqli (#2 or 3)

Some examples...

Instantiate a class

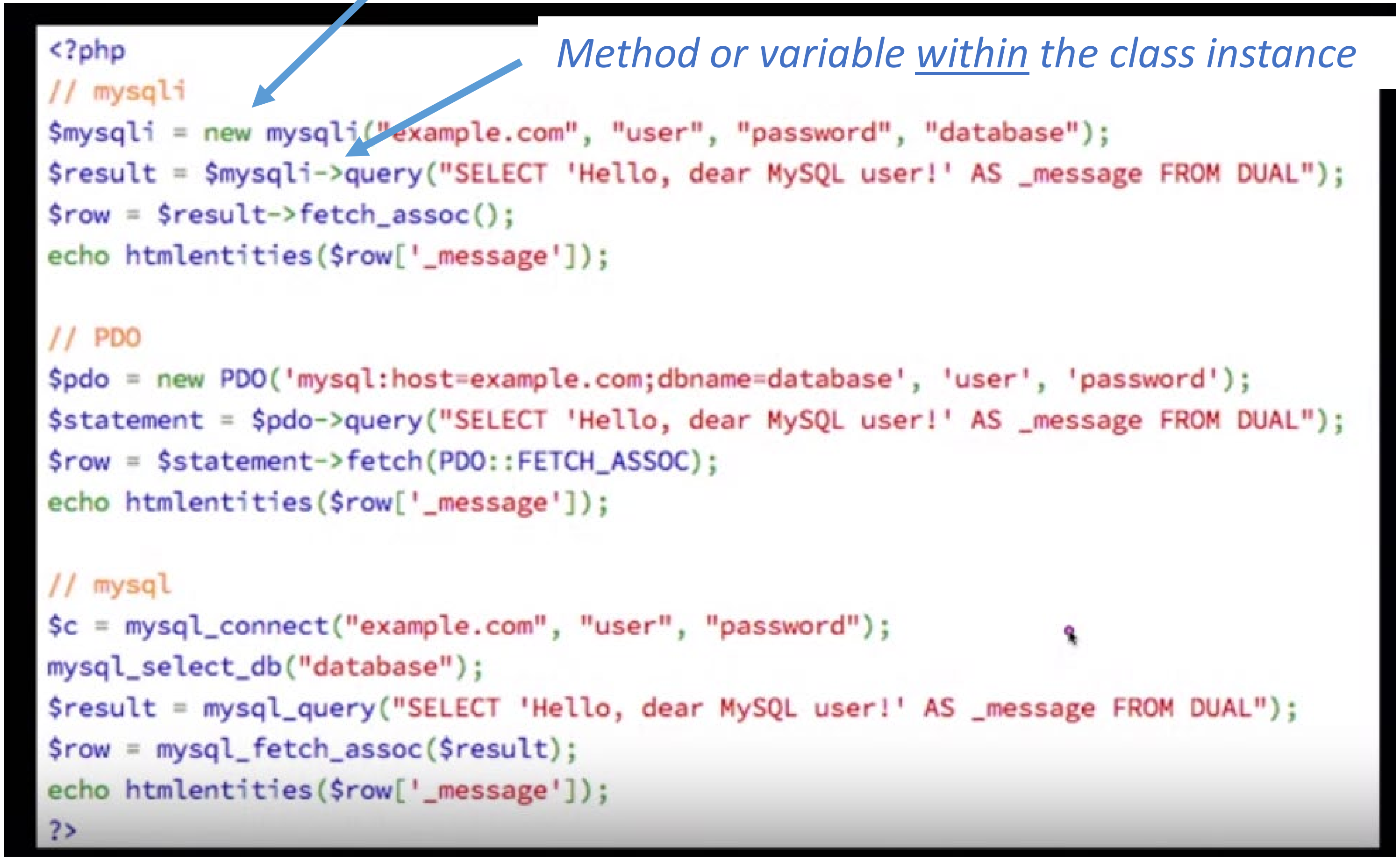
Method or variable within the class instance

method

#3

#4

#2

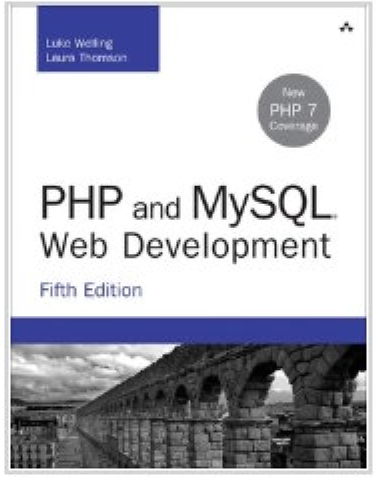


```
<?php
// mysqli
$mysqli = new mysqli("example.com", "user", "password", "database");
$result = $mysqli->query("SELECT 'Hello, dear MySQL user!' AS _message FROM DUAL");
$row = $result->fetch_assoc();
echo htmlentities($row['_message']);

// PDO
$pdo = new PDO('mysql:host=example.com;dbname=database', 'user', 'password');
$statement = $pdo->query("SELECT 'Hello, dear MySQL user!' AS _message FROM DUAL");
$row = $statement->fetch(PDO::FETCH_ASSOC);
echo htmlentities($row['_message']);

// mysql
$c = mysql_connect("example.com", "user", "password");
mysql_select_db("database");
$result = mysql_query("SELECT 'Hello, dear MySQL user!' AS _message FROM DUAL");
$row = mysql_fetch_assoc($result);
echo htmlentities($row['_message']);
?>
```

Course text books

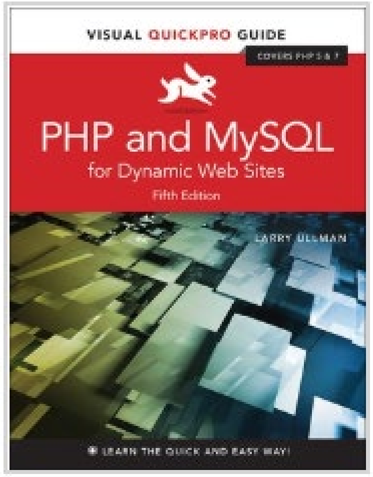
Welling & Thomson (purple): ch 11

- Shows both mysqli OO *and* procedural methods.
- They have a pretty complicated methodology
- They present PDO in one little section



Ullman (red): ch 9

- Only presents the mysqli *procedural* method
- He presents a more simple methodology



mysqli – part 1

simple PHP/mysql communication template

`$result = mysqli->query()`

i.e. from the Ullman (red) book

A simple template for SQL via PHP

1. Connect to database
2. Create your query string in PHP for SQL
3. Send query to MySQL DBMS
    a) Capture results in variable
    b) Boolean-test results
4. Get the SQL results for use in PHP

(Note: no #5 above. We don't have 'clean up' – it seems most programmers let that happen automagically.)

Here are some blogs with similar simple, beginners template examples:

- http://bit.ly/2u84AoE
- http://bit.ly/3ar7yp8
- http://bit.ly/2u9L8rC
- http://bit.ly/30xAKWO - PHP mysqli official documentation. Lists all the functions

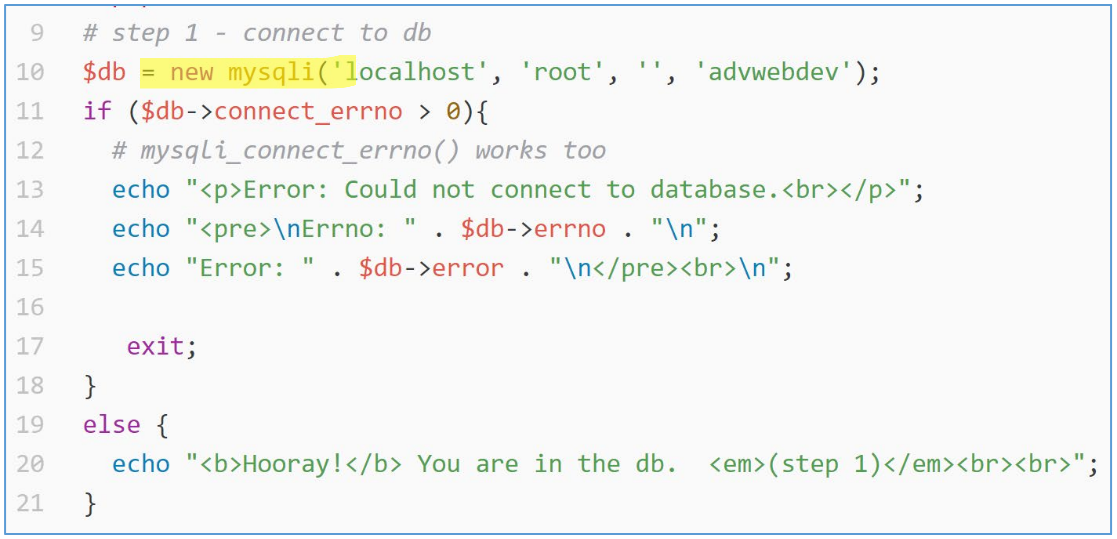
1. Connect (mysqli)

Recall that there are 2 protocols to use mysqli in PHP

A. In the OO method, you *instantiate* a new variable of class mysqli

B. In the procedural method, you *call* the mysqli_connect() function

OO



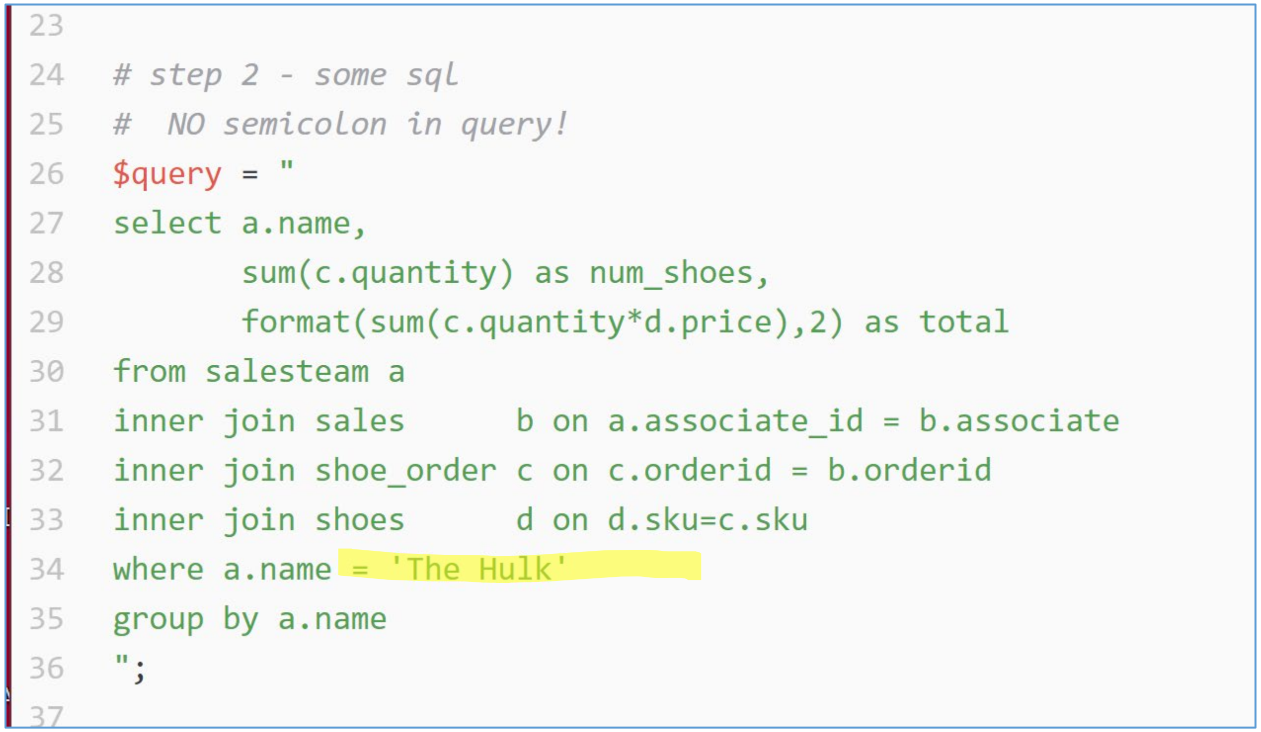
```php
# step 1 - connect to db
$db = new mysqli('localhost', 'root', '', 'advwebdev');
if ($db->connect_errno > 0){
  # mysqli_connect_errno() works too
  echo "<p>Error: Could not connect to database.<br></p>";
  echo "<pre>\nErrno: " . $db->errno . "\n";
  echo "Error: " . $db->error . "\n</pre><br>\n";

   exit;
}
else {
  echo "<b>Hooray!</b> You are in the db.  <em>(step 1)</em><br><br>";
}
```

procedural

```
// Make the connection:
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME)  or
          die(‘Could not connect to MySQL: ‘ . mysqli_connect_error() );
```

2. Create your query string

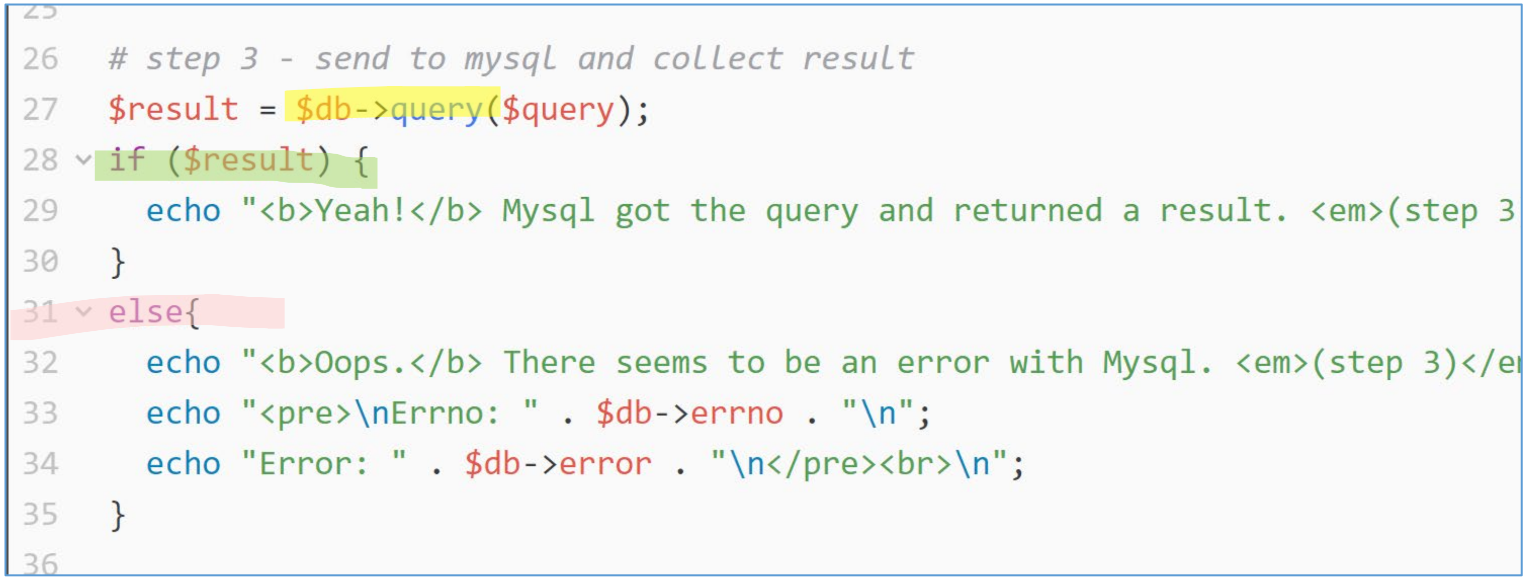
- This is simply the *exact* same SQL syntax which you would type into SQL.
- (except no semi at the end)
- Note: This is all hard-coded values. We'll deal with variables later.



```
# step 2 - some sql
#  NO semicolon in query!
$query = "
select a.name,
       sum(c.quantity) as num_shoes,
       format(sum(c.quantity*d.price),2) as total
from salesteam a
inner join sales      b on a.associate_id = b.associate
inner join shoe_order c on c.orderid = b.orderid
inner join shoes      d on d.sku=c.sku
where a.name = 'The Hulk'
group by a.name
";
```

3. Send query to mysql dbms

- Use the query() OO method or mysqli_query() procedure.
- Capture the returned $result
- Boolean test $result to see if error or not.
- If error:
  - OO: use errno (INT) and error (STRING) to get the mysql message
  - procedural: use mysql_connect_error() function

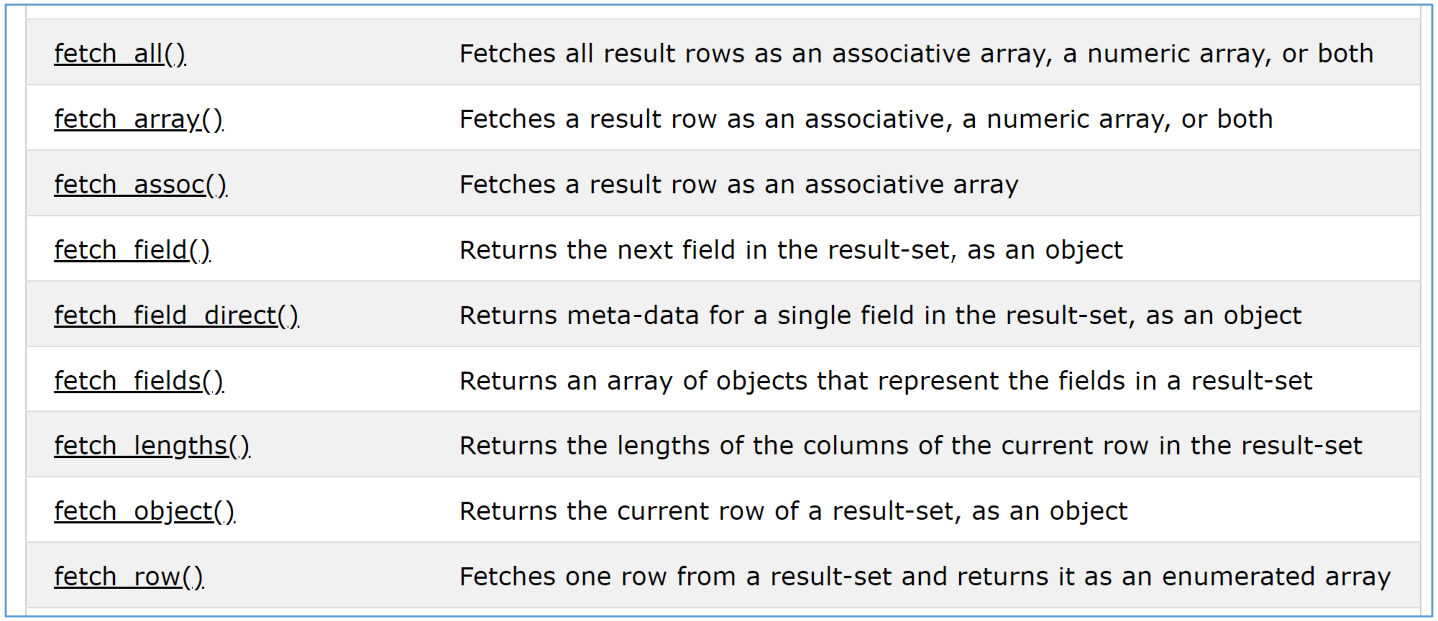


3a
3b

```
# step 3 - send to mysql and collect result
$result = $db->query($query);
if ($result) {
  echo "<b>Yeah!</b> Mysql got the query and returned a result. <em>(step 3
}
else{
  echo "<b>Oops.</b> There seems to be an error with Mysql. <em>(step 3)</e
  echo "<pre>\nErrno: " . $db->errno . "\n";
  echo "Error: " . $db->error . "\n</pre><br>\n";
}
```

4. Get the SQL results for use in PHP

- The recordset from MySQL are in the $result PHP var.
- We have to retrieve them. There are *many ways* to do this...
- Use <u>any</u> of the "fetch_*" methods; in both OO and procedural protocols
- These are documented in the PHP documentation: http://bit.ly/30xAKWO

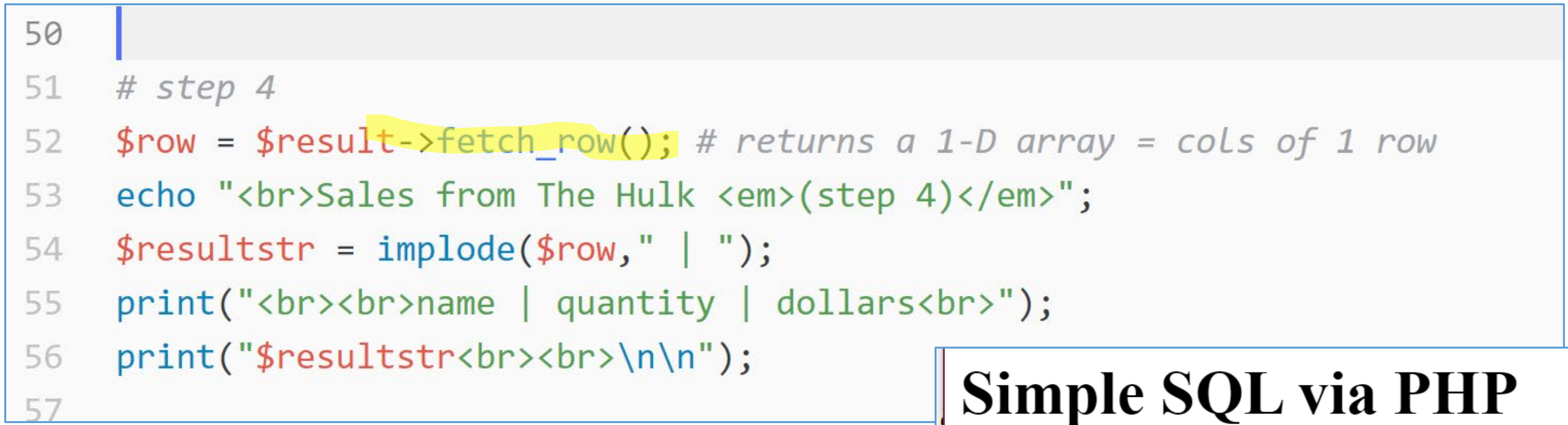


fetch_all()	Fetches all result rows as an associative array, a numeric array, or both
fetch_array()	Fetches a result row as an associative, a numeric array, or both
fetch_assoc()	Fetches a result row as an associative array
fetch_field()	Returns the next field in the result-set, as an object
fetch_field_direct()	Returns meta-data for a single field in the result-set, as an object
fetch_fields()	Returns an array of objects that represent the fields in a result-set
fetch_lengths()	Returns the lengths of the columns of the current row in the result-set
fetch_object()	Returns the current row of a result-set, as an object
fetch_row()	Fetches one row from a result-set and returns it as an enumerated array

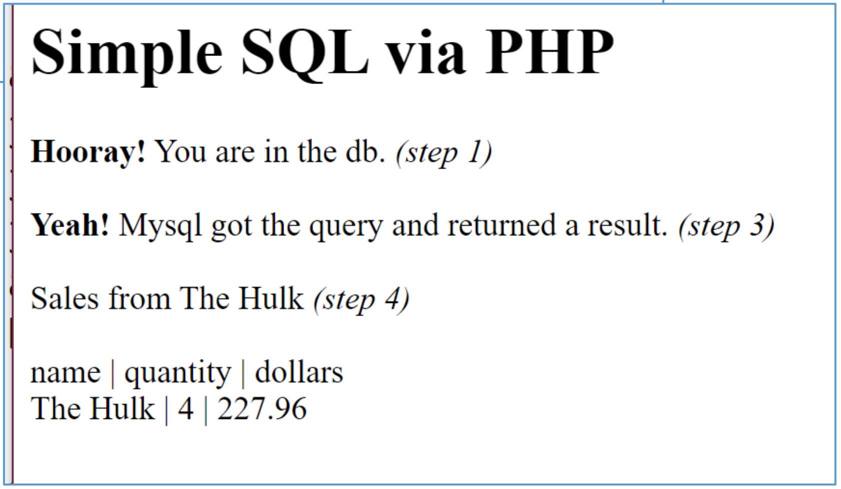
*Ack! **So many**.   Q: Which one? A: You decide... you are the programmer!*

4. Get the SQL results for use in PHP

- Let's pick fetch_row() – which returns an array for each row
- For this result, we "know" we only get 1 row back...



```php

# step 4
$row = $result->fetch_row(); # returns a 1-D array = cols of 1 row
echo "<br>Sales from The Hulk <em>(step 4)</em>";
$resultstr = implode($row," | ");
print("<br><br>name | quantity | dollars<br>");
print("$resultstr<br><br>\n\n");
```



Simple SQL via PHP

Hooray! You are in the db. *(step 1)*

Yeah! Mysql got the query and returned a result. *(step 3)*

Sales from The Hulk *(step 4)*

name | quantity | dollars
The Hulk | 4 | 227.96

http://localhost/lectures/l11_simple_sql_via.php

The problem with this simple template

a.k.a. SQL Injection

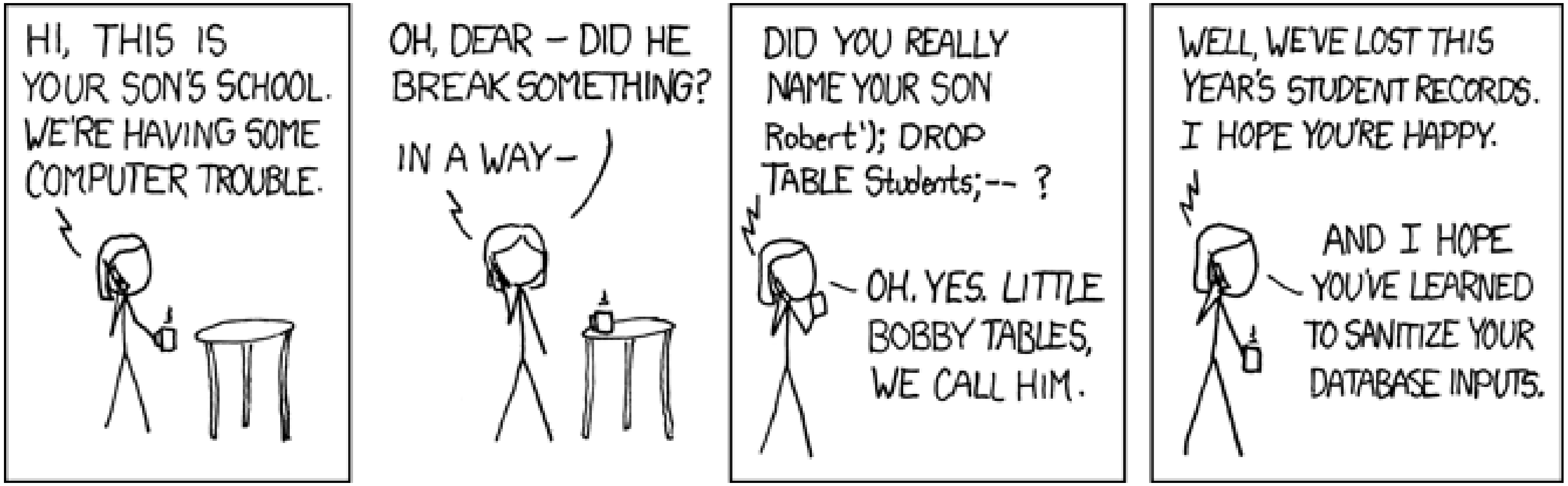
Dynamic SQL

- We *can't* have static, non-changing, hardcoded SQL queries. Yuck!
- We want dynamic SQL – with variables selected by the user!
- So, let's do that.... it's easy in PHP, right?
  - l10_simple_sql_with_input_via.php

SQL injection

- Remember HTML injection? *Solution*: `htmlentities()`
- SQL has the same problem.

This is good stuff…


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY –
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

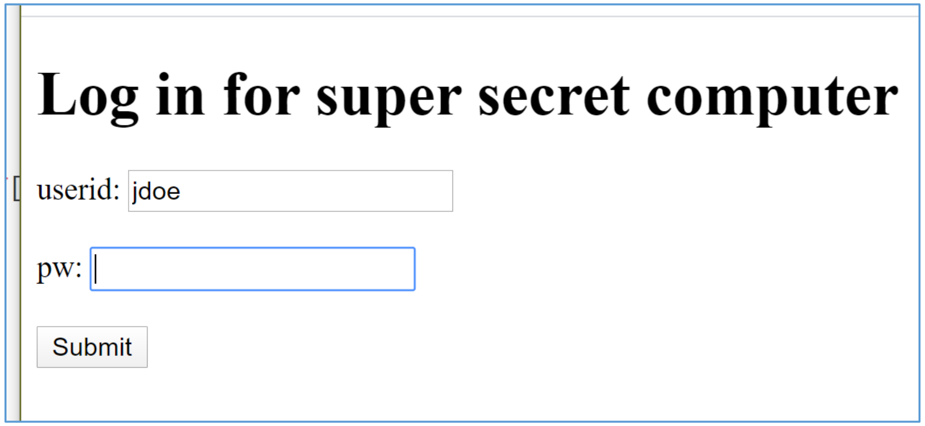
#327

Example

- The bad guy can figure out your quotes.
- He can then add quotes to a field to totally change your SQL.
    - l10b_users_setup.sql
    - l10b_users_setup.php

- **Fix**: we need to *escape* special characters, just like htmlentities() did for us.
- In mysqli, this is called using ***prepared*** statements!

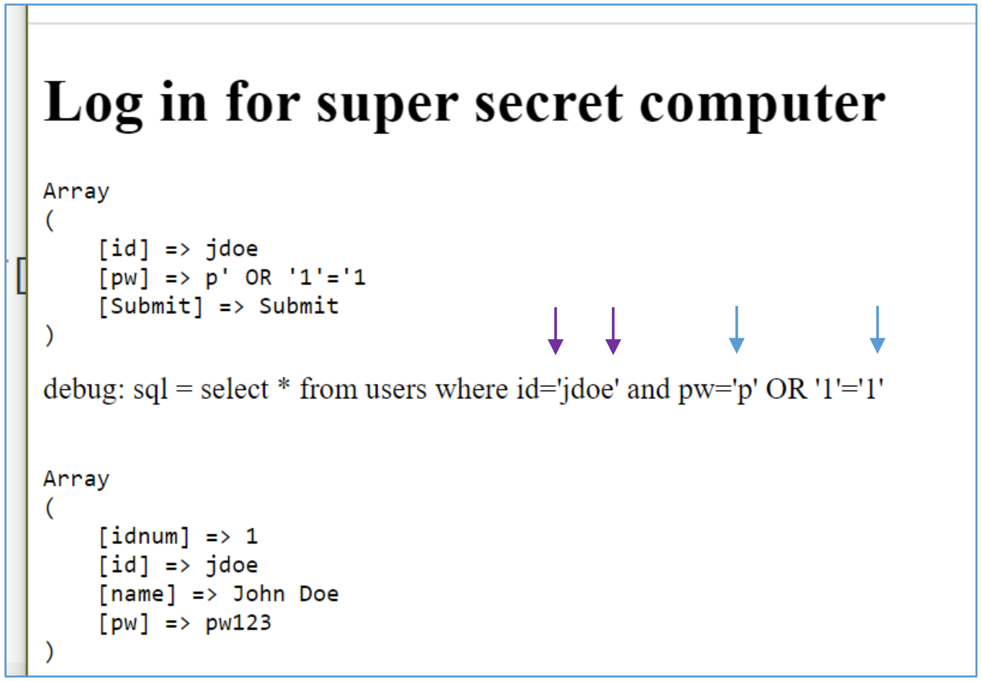
http://localhost/lectures/l11b_sql_injection.php



Log in for super secret computer

userid: jdoe

pw:



Log in for super secret computer

```
Array
(
    [id] => jdoe
    [pw] => p' OR '1'='1
    [Submit] => Submit
)
```

debug: sql = select * from users where id='jdoe' and pw='p' OR '1'='1'

```
Array
(
    [idnum] => 1
    [id] => jdoe
    [name] => John Doe
    [pw] => pw123
)
```

mysqli – part 2

A more complicated, safer mysqli template

The *prepared* statements!

Note: can be in OO or procedural mysqli_ methods

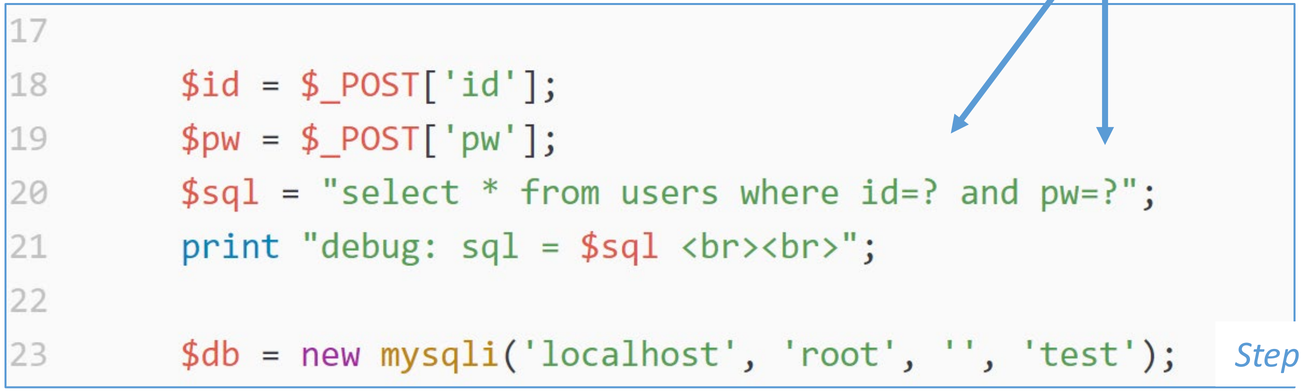
A more complex, yet safer template

1. *Connect* to MySQL DBMS (same as part 1 in this lecture)
2. *Create* a string template of the (*dynamic*) SQL query you want to send – it probably has *variables* from a user form
3. ★ *Prepare* the SQL string into a statement → escape all chars (quotes, etc.)
4. *Bind_param* any input PHP variables into to the statement
5. *Execute* the SQL command on the mysql db
6. *Retrieve* (i.e. collect, store, fetch) the results from mysql
7. *Free* up the result set (clean-up memory)
8. *Close* the DBMS connection

2. Create SQL query string with "markers"

- Most often, the SQL will have user input variables from a form
- As we did earlier, we *could* just put those user input variables right into the SQL (*no, no, no*)
    - This is HIGHLY ADVISED AGAINST. There is a potential for security issues.
- *Rather*, mark the variable locations with "?" to be replaced later
- *(Note: no quotes for ?. And still no semi.)*

Put "?" where user variables will *eventually* go



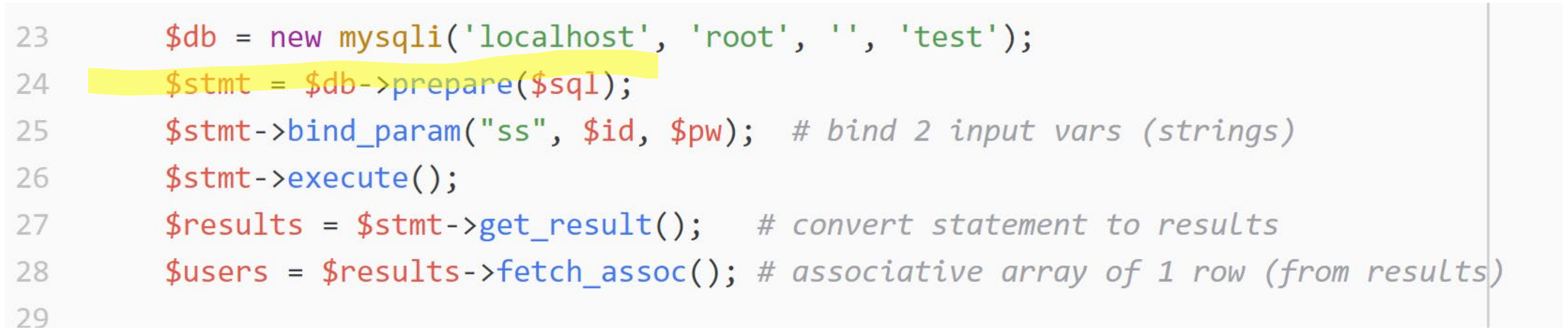
```
        $id = $_POST['id'];
        $pw = $_POST['pw'];
        $sql = "select * from users where id=? and pw=?";
        print "debug: sql = $sql <br><br>";

        $db = new mysqli('localhost', 'root', '', 'test');
```

Step 1 connection is the same

3. Prepare

- What does *prepare* do?
- It actually talks to the DB optimizer about the db PLAN
- Line 24 below is the OO format method for prepare().
- The procedural method equivalent function is: mysqli_stmt_prepare()
- You should check for an error here:

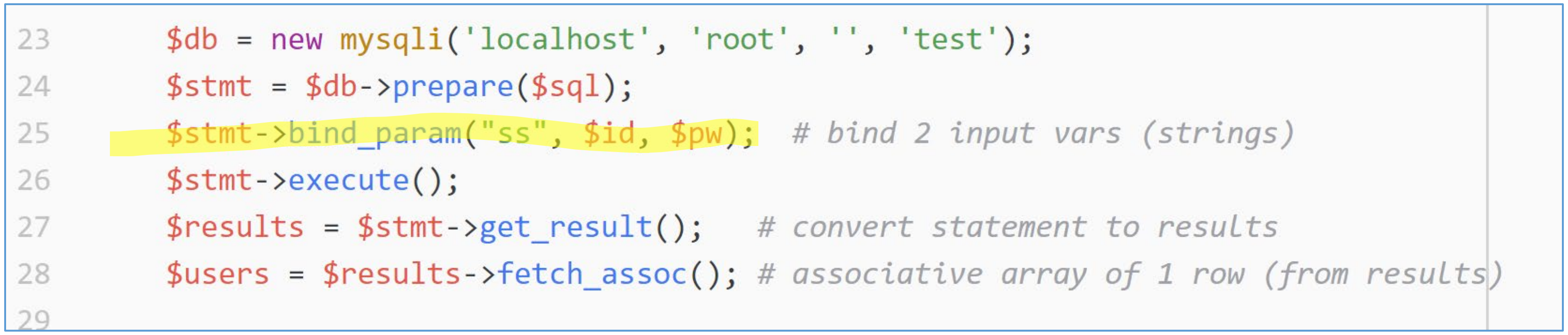
```
if (! $stmt) { error & exit }
```



```
23      $db = new mysqli('localhost', 'root', '', 'test');
24      $stmt = $db->prepare($sql);
25      $stmt->bind_param("ss", $id, $pw);  # bind 2 input vars (strings)
26      $stmt->execute();
27      $results = $stmt->get_result();   # convert statement to results
28      $users = $results->fetch_assoc(); # associative array of 1 row (from results)
29
```

4. bind_param – the input bind

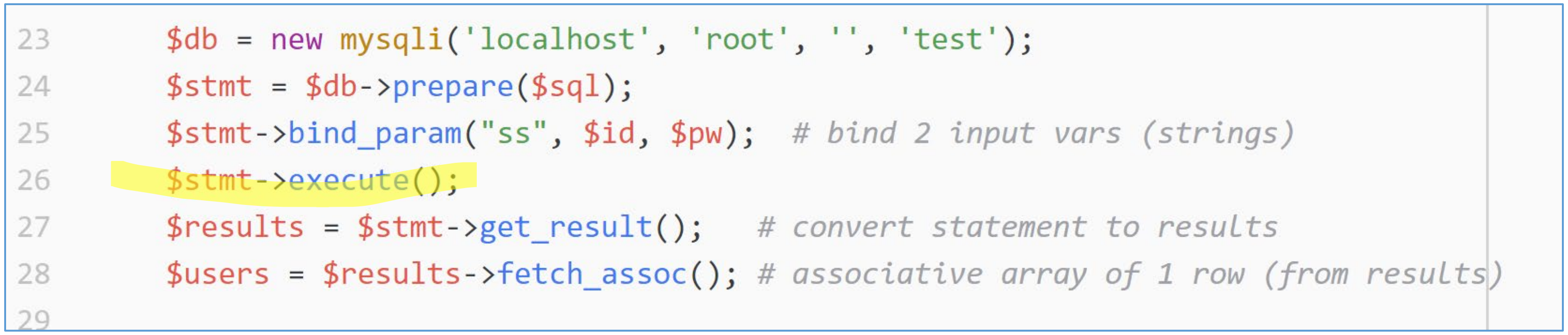
- This is the method of *replacing* the "?" in the statement with the user-given methods from the HTML form.
- The *bind* method properly escapes, quotes, checks and filters for security issues.
- The procedural function is: mysqli_stmt_bind_param()
- The first parameter is a string which has 1 character per input variable. That 1 character describes the variable *type*.
    - s = string
    - d = decimal (float)
    - i = int



```
        $db = new mysqli('localhost', 'root', '', 'test');
        $stmt = $db->prepare($sql);
        $stmt->bind_param("ss", $id, $pw);  # bind 2 input vars (strings)
        $stmt->execute();
        $results = $stmt->get_result();    # convert statement to results
        $users = $results->fetch_assoc(); # associative array of 1 row (from results)
```

5. Execute

- This sends the prepared statement to the mysql dbms.
- The procedural version is: mysqli_stmt_execute()
- *Very Important* Note:
  - You don't capture the $result here yet!
  - The execute() command leaves the results on the sever!
  - i.e. it does *not* automatically pull down the results to the client
  - You need to do that in the next step
  - This is **unlike** the $result = $db->query() command from earlier



```php
    $db = new mysqli('localhost', 'root', '', 'test');
    $stmt = $db->prepare($sql);
    $stmt->bind_param("ss", $id, $pw);  # bind 2 input vars (strings)
    $stmt->execute();
    $results = $stmt->get_result();    # convert statement to results
    $users = $results->fetch_assoc(); # associative array of 1 row (from results)
```

6a. Retrieve (store) results

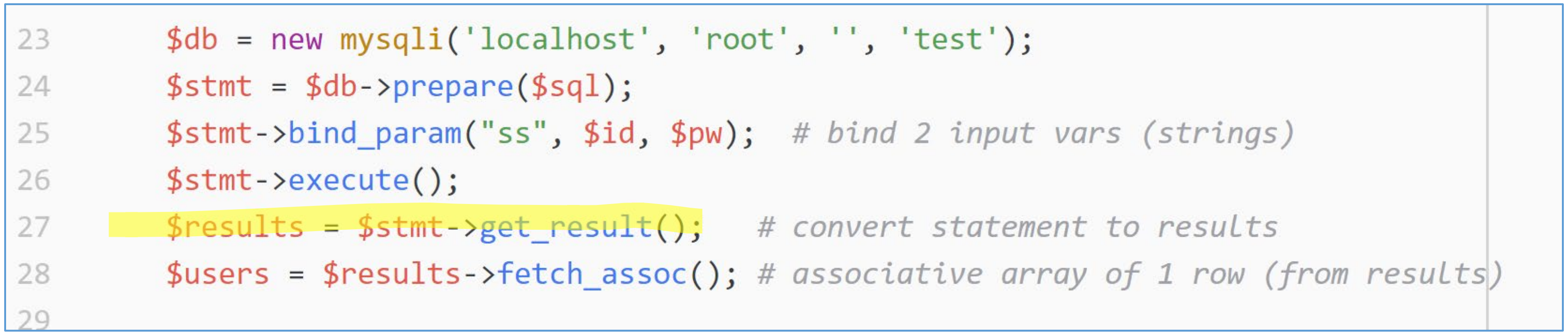
- The results are still on the mysql server
- After a mysqli execute(), there are a couple of ways to get the results to the client from the server

1. put the result *into* the $stmt object

```
$stmt->store_result()
```

2. return result to *new* object

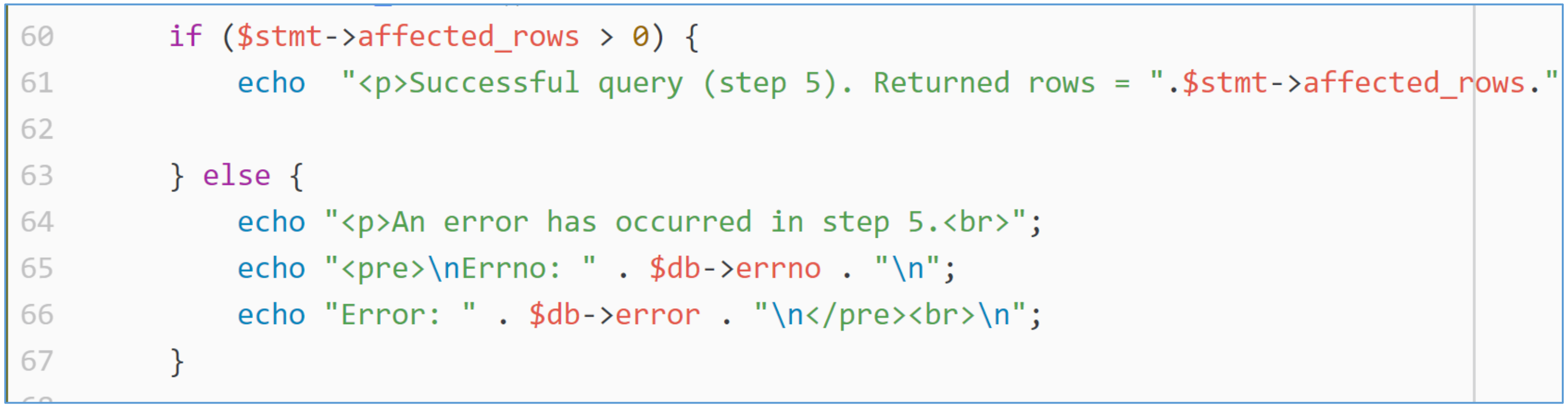
```
$results = $stmt->get_result()
```



```
$db = new mysqli('localhost', 'root', '', 'test');
$stmt = $db->prepare($sql);
$stmt->bind_param("ss", $id, $pw);  # bind 2 input vars (strings)
$stmt->execute();
$results = $stmt->get_result();    # convert statement to results
$users = $results->fetch_assoc(); # associative array of 1 row (from results)
```

6b. Check if results or error

- Once you $stmt->store_results(), they are now down on the client and we can check results.
- $stmt-> affected_rows is one way.   If == 0 nothing was returned.

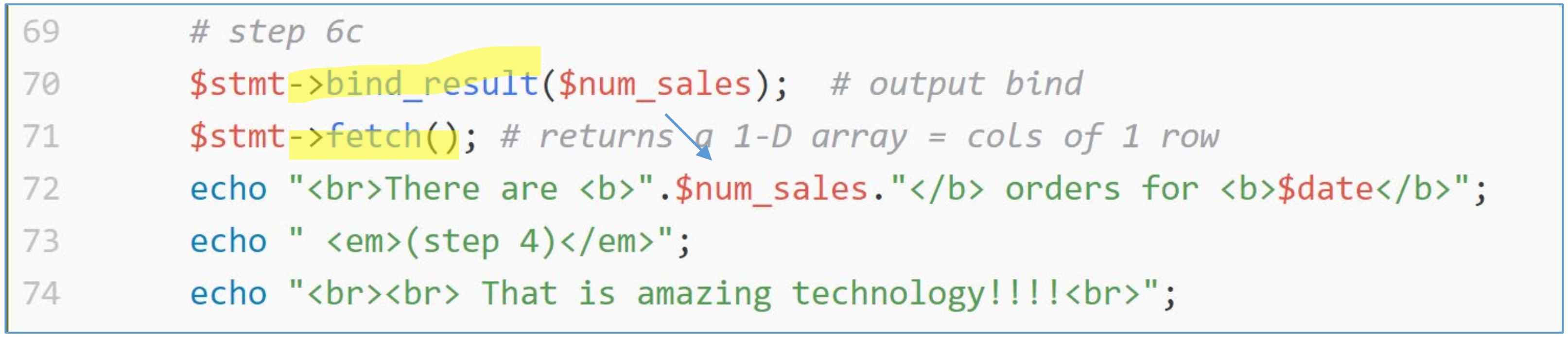


```
    if ($stmt->affected_rows > 0) {
        echo  "<p>Successful query (step 5). Returned rows = ".$stmt->affected_rows."

    } else {
        echo "<p>An error has occurred in step 5.<br>";
        echo "<pre>\nErrno: " . $db->errno . "\n";
        echo "Error: " . $db->error . "\n</pre><br>\n";
    }
```

6c. Bind the results

If you "stored" results in the $stmt object,

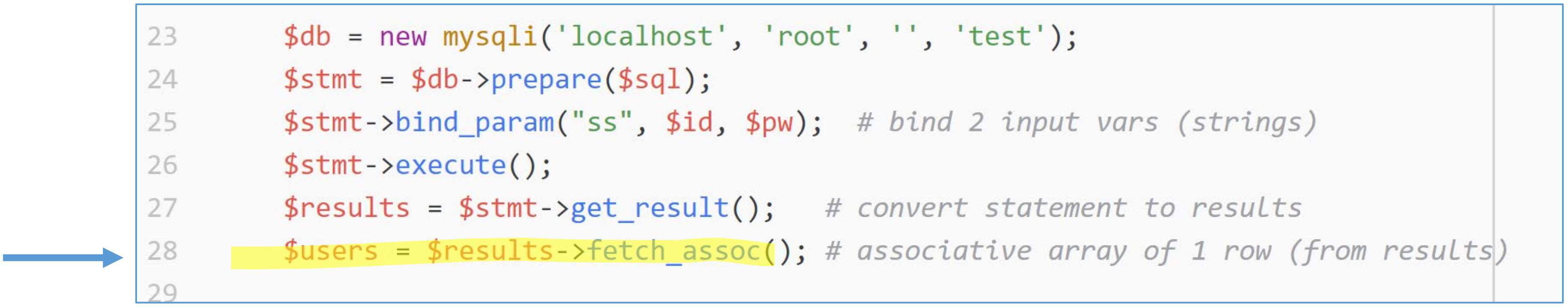
- Just like we did a bind_params() for the *input* vars, let's try a bind_result() for the *output* vars.
- This method "links" SQL output columns to particular variables in PHP.
- On line 70, the PHP var $num_sales is now linked to the first column returned from the db at the next fetch().



```
69        # step 6c
70        $stmt->bind_result($num_sales);   # output bind
71        $stmt->fetch(); # returns a 1-D array = cols of 1 row
72        echo "<br>There are <b>".$num_sales."</b> orders for <b>$date</b>";
73        echo " <em>(step 4)</em>";
74        echo "<br><br> That is amazing technology!!!!<br>";
```

6d. Fetch the results

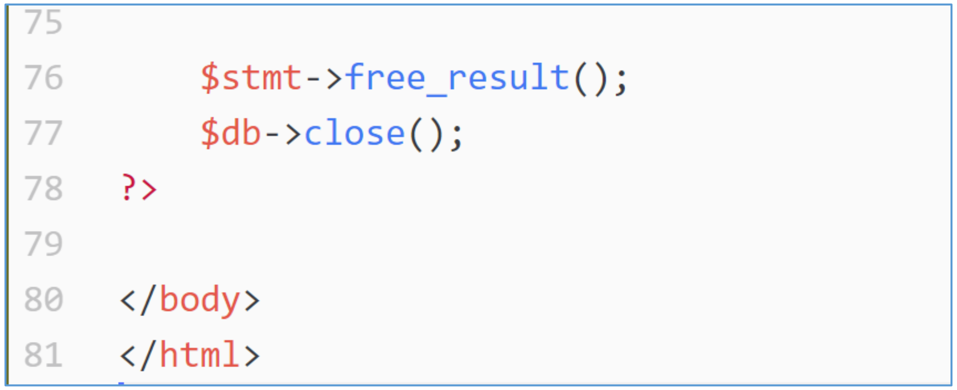
- Again, all the fetch_*() methods and functions are available to you.



```
    $db = new mysqli('localhost', 'root', '', 'test');
    $stmt = $db->prepare($sql);
    $stmt->bind_param("ss", $id, $pw);  # bind 2 input vars (strings)
    $stmt->execute();
    $results = $stmt->get_result();   # convert statement to results
    $users = $results->fetch_assoc(); # associative array of 1 row (from results)
```

7. free, 8. close

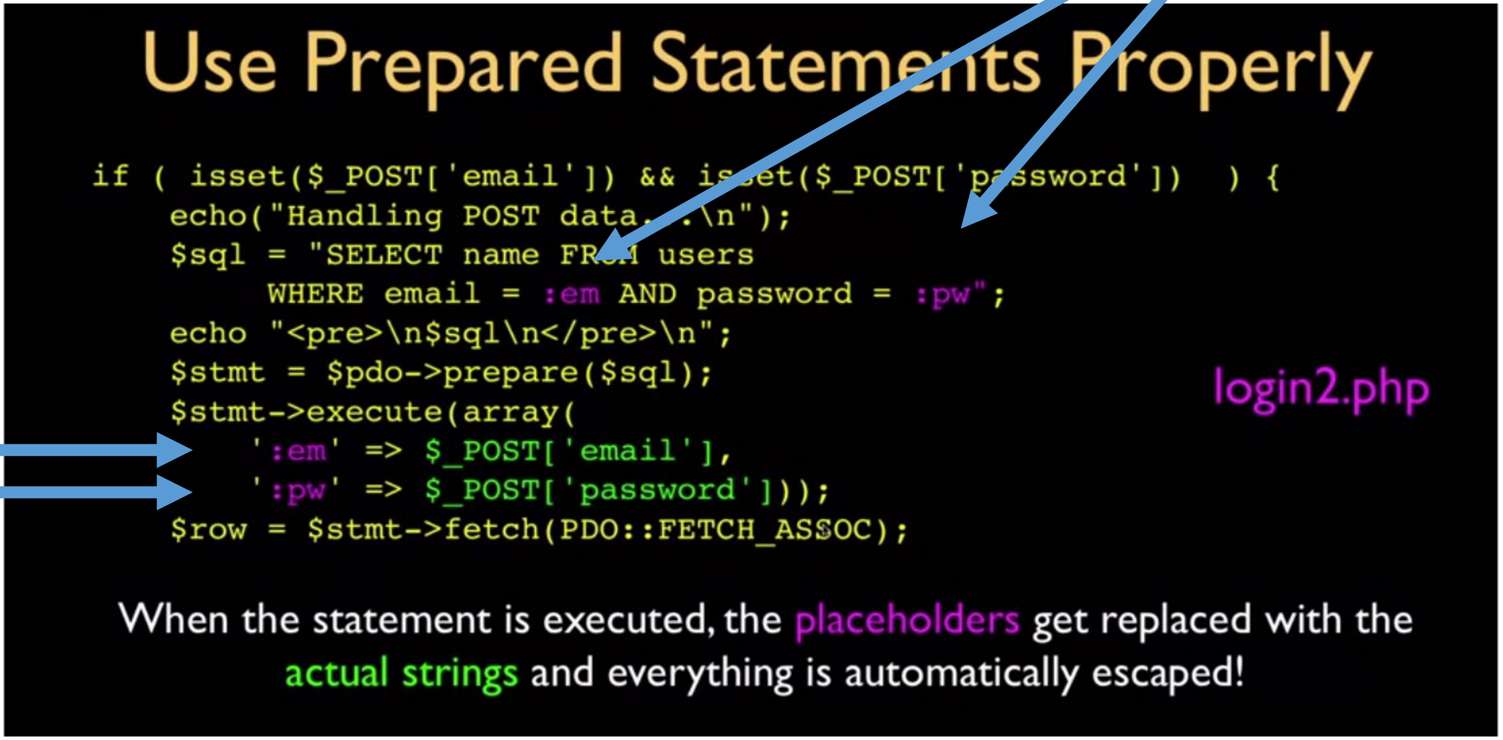
- We are not required to put these in our code.
- When PHP exits, it will do them automatically if we do not.
- This would be important for large recordset results and scripts that might take a longer time.



```

        $stmt->free_result();
        $db->close();
    ?>

    </body>
    </html>
```

PDO prepared statements

- PDO is a little different.
- Rather than ? as the *mark* character, it uses :var.



Use Prepared Statements Properly

```
if ( isset($_POST['email']) && isset($_POST['password'])  ) {
    echo("Handling POST data...\n");
    $sql = "SELECT name FROM users
         WHERE email = :em AND password = :pw";
    echo "<pre>\n$sql\n</pre>\n";
    $stmt = $pdo->prepare($sql);
    $stmt->execute(array(
        ':em' => $_POST['email'],
        ':pw' => $_POST['password']));
    $row = $stmt->fetch(PDO::FETCH_ASSOC);
```

login2.php

When the statement is executed, the placeholders get replaced with the actual strings and everything is automatically escaped!

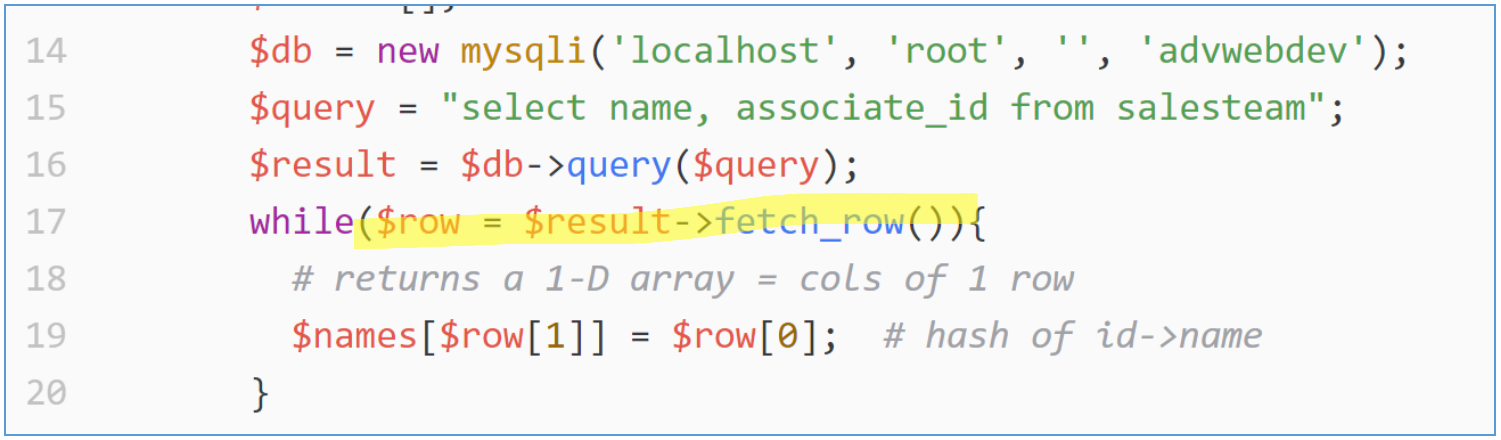
mysqli – part 3

Accessing rows and cols from a SELECT statement

Examples of fetch_*() methods

while($row=fetch_*())

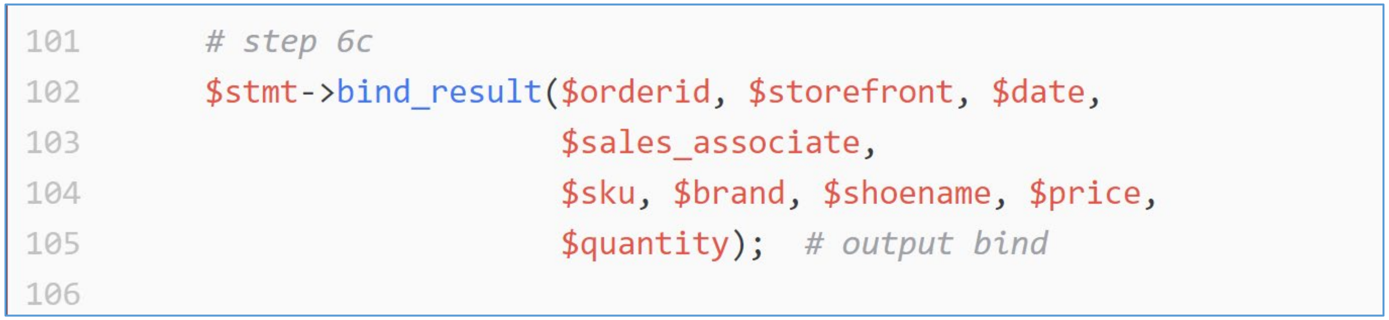
- When multiple rows, put the fetch in a while loop
- As long as there is a row from the database to fetch, the while will be True
- And a $row array will be defined.
- Each element is a col from 1 row. $row[0] is first col, $row[1] is 2nd
- You can use any of the fetch*() methods this way.
- e.g. fetch_assoc() builds associative array (hash, dictionary) with column names as keys.



```php
        $db = new mysqli('localhost', 'root', '', 'advwebdev');
        $query = "select name, associate_id from salesteam";
        $result = $db->query($query);
        while($row = $result->fetch_row()){
          # returns a 1-D array = cols of 1 row
          $names[$row[1]] = $row[0];  # hash of id->name
        }
```

Get fancy with bind_result

- You can also tell myslqi the PHP variables you'd like to assign (i.e. link) to each sql column
- This is called bind_result
  - OO: $stmt->bind_result()
  - procedural: mysqli_stmt_bind_result()
- The 9 PHP vars below will correspond to the 9 SQL columns requested in the SQL $query in the $stmt->execute()

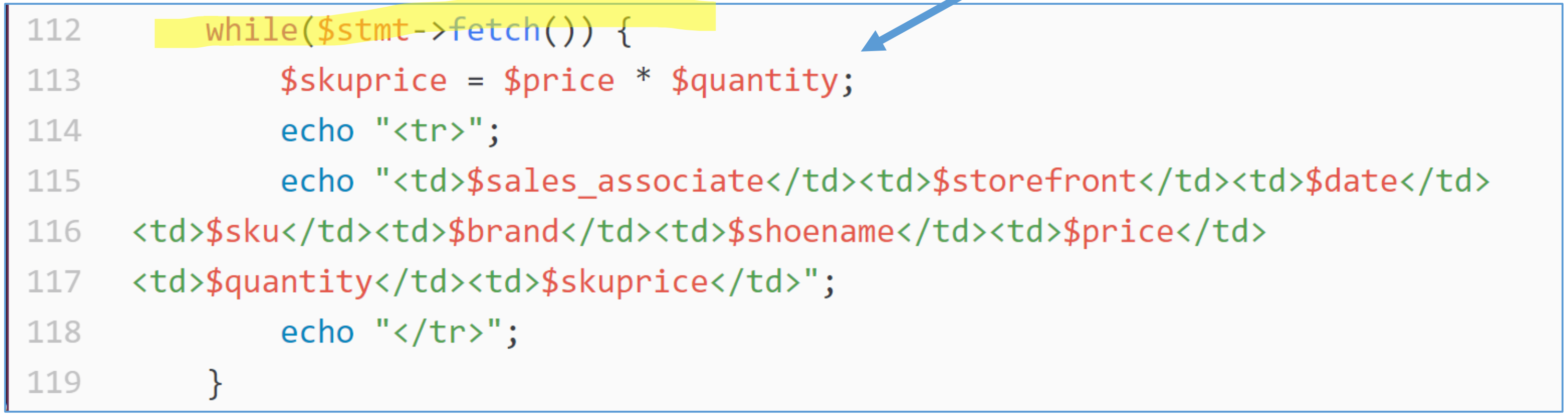


```
        # step 6c
        $stmt->bind_result($orderid, $storefront, $date,
                           $sales_associate,
                           $sku, $brand, $shoename, $price,
                           $quantity);   # output bind

```

Get fancy with bind_result

- The bind_result() is *outside* the while loop. (line 102)
- Think of it as a **1-time** assignment.
- Now, you don't need a $row variable array. Right? Just use while(fetch())
- Because now, in the while(fetch()) loop, the "binded" variables are updated each loop!

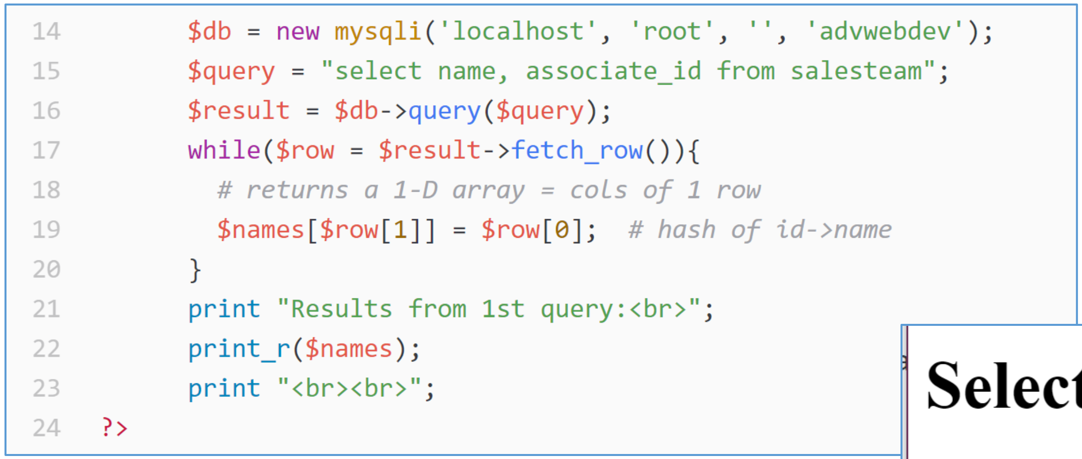
These PHP vars are defined in bind_results()
They changes for every loop, i.e. row from SQL



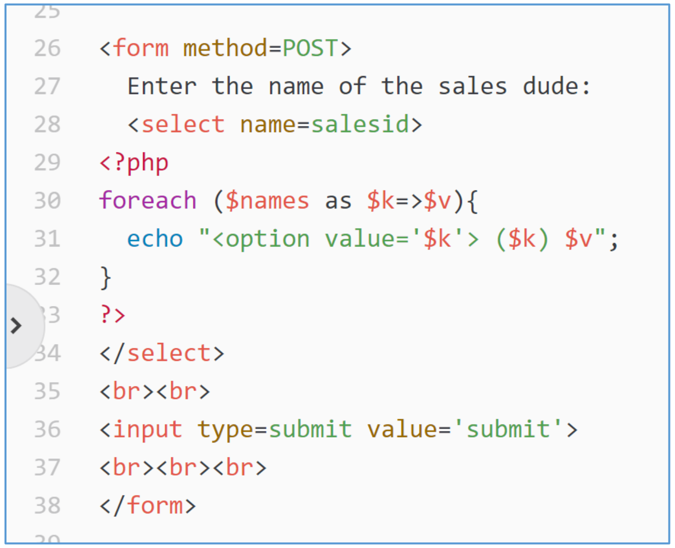
```
    while($stmt->fetch()) {
        $skuprice = $price * $quantity;
        echo "<tr>";
        echo "<td>$sales_associate</td><td>$storefront</td><td>$date</td>
<td>$sku</td><td>$brand</td><td>$shoename</td><td>$price</td>
<td>$quantity</td><td>$skuprice</td>";
        echo "</tr>";
    }
```

Example code walkthru

SQL to build a HTML form pulldown select

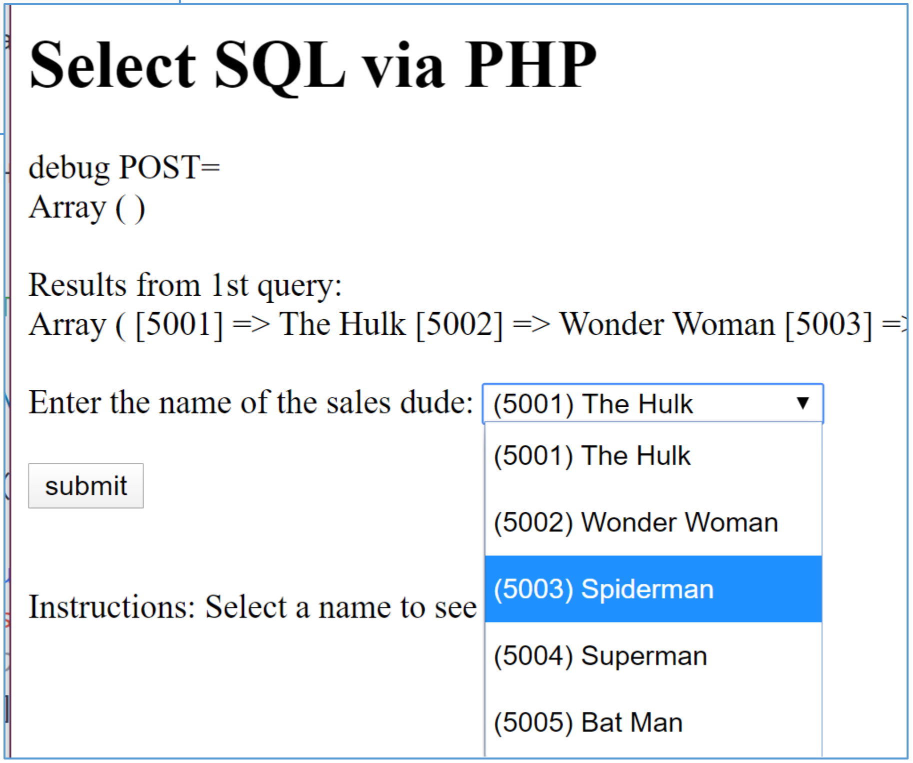


```php
        $db = new mysqli('localhost', 'root', '', 'advwebdev');
        $query = "select name, associate_id from salesteam";
        $result = $db->query($query);
        while($row = $result->fetch_row()){
          # returns a 1-D array = cols of 1 row
          $names[$row[1]] = $row[0];  # hash of id->name
        }
        print "Results from 1st query:<br>";
        print_r($names);
        print "<br><br>";
?>
```



```php
<form method=POST>
  Enter the name of the sales dude:
  <select name=salesid>
<?php
foreach ($names as $k=>$v){
  echo "<option value='$k'> ($k) $v";
}
?>
</select>
<br><br>
<input type=submit value='submit'>
<br><br><br>
</form>
```

http://localhost/lectures/l11_select_queries_via.php



Select SQL via PHP

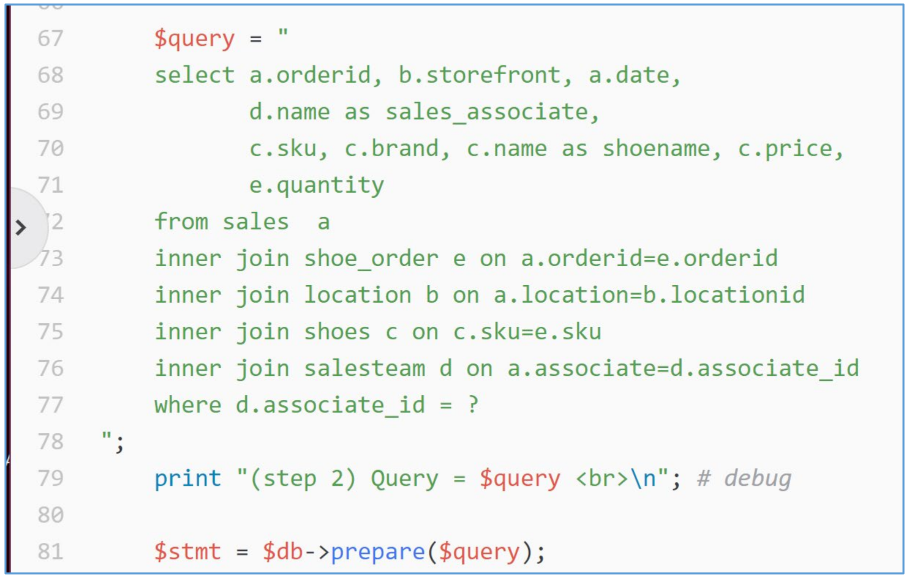
debug POST=
Array ( )

Results from 1st query:
Array ([5001] => The Hulk [5002] => Wonder Woman [5003] =>

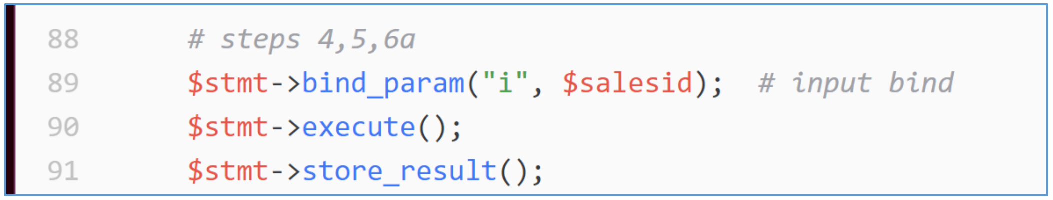
Enter the name of the sales dude: (5001) The Hulk ▼

Instructions: Select a name to see

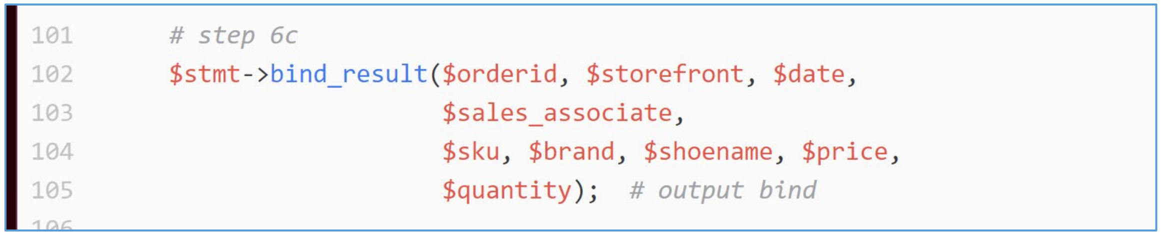
From the pulldown, build a query



```
    $query = "
    select a.orderid, b.storefront, a.date,
           d.name as sales_associate,
           c.sku, c.brand, c.name as shoename, c.price,
           e.quantity
    from sales  a
    inner join shoe_order e on a.orderid=e.orderid
    inner join location b on a.location=b.locationid
    inner join shoes c on c.sku=e.sku
    inner join salesteam d on a.associate=d.associate_id
    where d.associate_id = ?
";
    print "(step 2) Query = $query <br>\n"; # debug

    $stmt = $db->prepare($query);
```



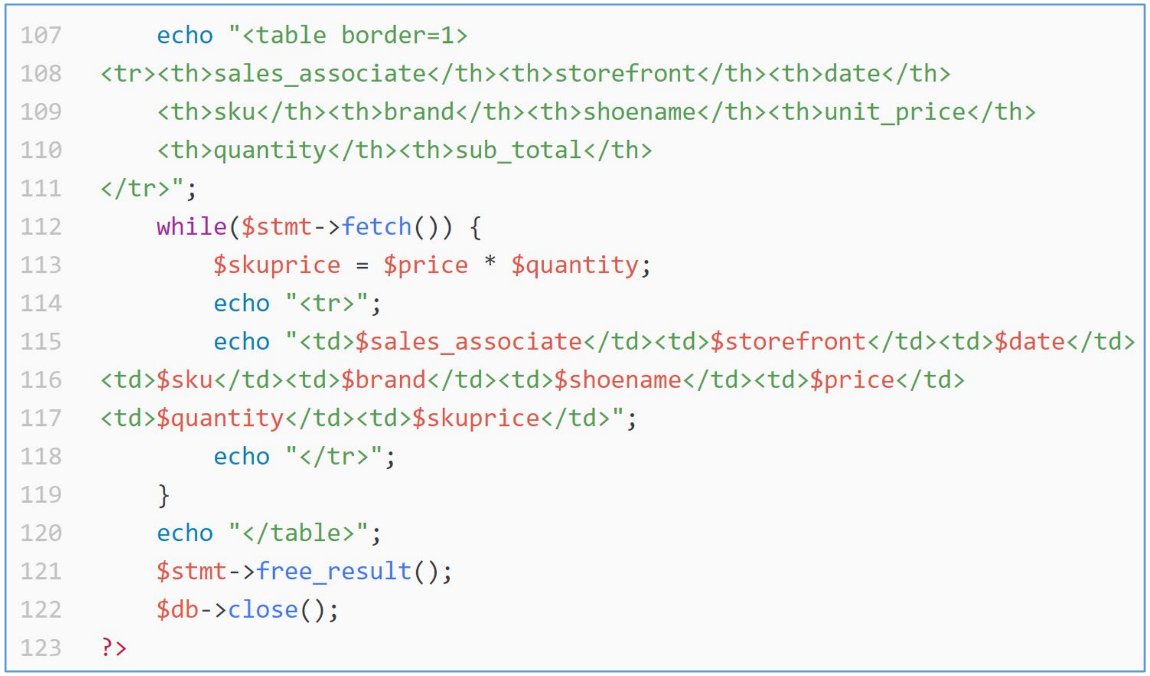
```
    # steps 4,5,6a
    $stmt->bind_param("i", $salesid);   # input bind
    $stmt->execute();
    $stmt->store_result();
```



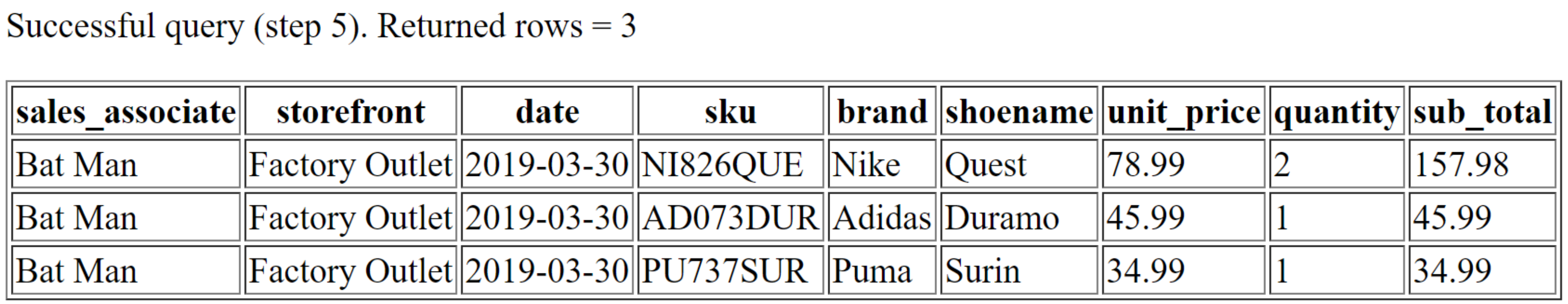
```
    # step 6c
    $stmt->bind_result($orderid, $storefront, $date,
                       $sales_associate,
                       $sku, $brand, $shoename, $price,
                       $quantity);  # output bind
```

From query results, built html table

Each row of mysql results = <tr> row of table </tr>



```
    echo "<table border=1>
<tr><th>sales_associate</th><th>storefront</th><th>date</th>
    <th>sku</th><th>brand</th><th>shoename</th><th>unit_price</th>
    <th>quantity</th><th>sub_total</th>
</tr>";
    while($stmt->fetch()) {
        $skuprice = $price * $quantity;
        echo "<tr>";
        echo "<td>$sales_associate</td><td>$storefront</td><td>$date</td>
<td>$sku</td><td>$brand</td><td>$shoename</td><td>$price</td>
<td>$quantity</td><td>$skuprice</td>";
        echo "</tr>";
    }
    echo "</table>";
    $stmt->free_result();
    $db->close();
?>
```



Successful query (step 5). Returned rows = 3

sales_associate	storefront	date	sku	brand	shoename	unit_price	quantity	sub_total
Bat Man	Factory Outlet	2019-03-30	NI826QUE	Nike	Quest	78.99	2	157.98
Bat Man	Factory Outlet	2019-03-30	AD073DUR	Adidas	Duramo	45.99	1	45.99
Bat Man	Factory Outlet	2019-03-30	PU737SUR	Puma	Surin	34.99	1	34.99