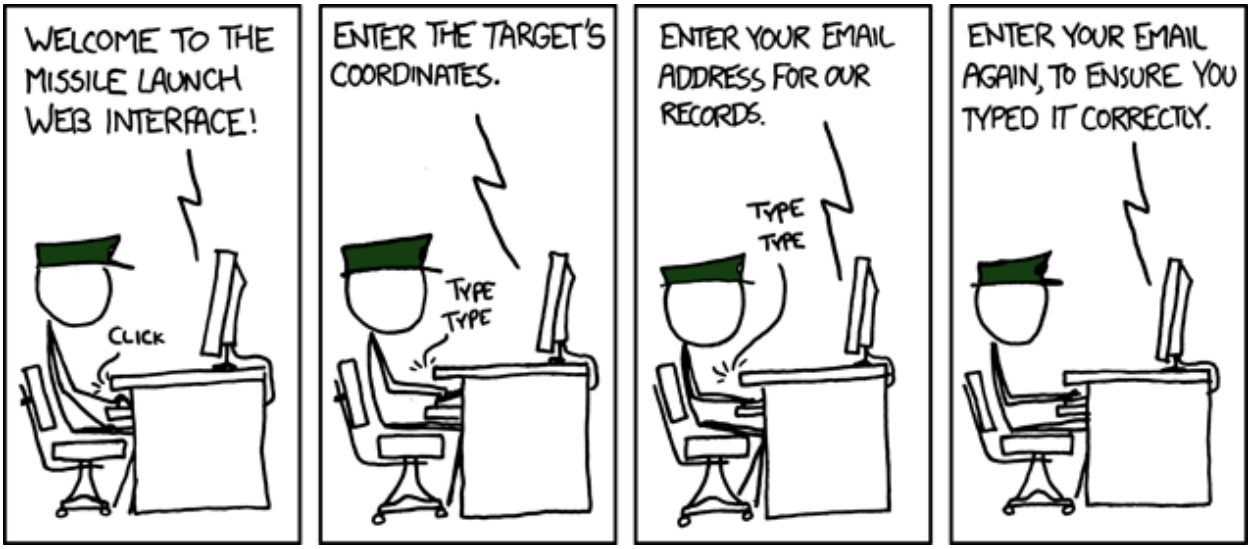
PHP and MySQL

From CSE330 Wiki

MySQL is great, but it's not useful for web applications without a way to integrate it with your application logic. This article discusses how to query MySQL from within PHP.

This article assumes that you are comfortable with the content in the following guides. If you are not, you should read the guides listed below first.

- PHP
- Introduction to MySQL
- MySQL Schema and State



Contents

PHP Libraries for Database Manipulation

PHP comes with many different ways of querying databases. This guide instructs you on how to use **MySQL Improved Prepared Statements**, which is a cleaner, more elegant way to perform queries on your database.

Connecting to a MySQL Database in PHP

To connect to a database using the MySQL Improved interface, use the mysqli command:

```php
<?php
// Content of database.php

$mysqli = new mysqli('localhost', 'username', 'password',
'databasename');

if($mysqli->connect_errno) {
    printf("Connection Failed: %s\n", $mysqli->connect_error);
    exit;
}
?>
```

You need to run this code before you can perform *any* other query on your database. Many PHP web developers put this code in a separate file and **require** it into other files that require database access.

You should create a new MySQL user with minimal privileges just for PHP to use.

Performing Queries that Require Parameters

When you want to INSERT or UPDATE an entry in your database, you probably need to supply some value from PHP for MySQL to interpret. This section discusses *parameter binding*, which is how you can do this sort of thing.

The first type of prepared query we will discuss are INSERT queries.

Suppose you had a form that submitted the following information:

- **first**, the user's first name
- **last**, the user's last name
- **dept**, the user's department

You could insert data received from the form into the employees database using the following PHP code:

```php
<?php
require 'database.php';

$first = $_POST['first'];
$last = $_POST['last'];
$dept = $_POST['dept'];

$stmt = $mysqli->prepare("insert into employees (first_name, last_name,
department) values (?, ?, ?)");
if(!$stmt){
    printf("Query Prep Failed: %s\n", $mysqli->error);
    exit;
}

$stmt->bind_param('sss', $first, $last, $dept);

$stmt->execute();

$stmt->close();

?>
```

The query that will end up being executed by the above code is:

```
INSERT INTO employees (first_name, last_name, department) VALUES
('John', 'Deere', 'EECE')
```

Notice how the MySQLi interface uses object-oriented PHP. Refer to the PHP guide if you need a refresher on object-oriented PHP.

bind_param()

The most tricky line in the above example is the **$stmt->bind_param()** line. Here's how it works.

The first parameter defines the data types of data to be inserted into the query. It is a string with one character per data type. The possible data types are:

- **i** - Integer
- **d** - Decimal
- **s** - String
- **b** - Blob

Note that data types in queries are *not* the same as data types in schemas. For instance, when saving a date into MySQL, you need to use a string representation of the data (except for timestamp, which uses an integer), so you would use the **s** option.

The remaining parameters define values associated with each of the question marks in your query.

In the example code in the previous section, there are three question marks in the query, all of which are strings, and so the first argument to **$stmt->bind_param()** is 'sss'. The remaining three parameters is the information itself to be "injected" into the query.

Performing Queries that Return Data

When you perform a successful SELECT query, MySQL always returns a table with the results of the query. This section discusses *result binding*, which is how you can load the result table into PHP.

Consider the following example:

```
<?php
require 'database.php';

$stmt = $mysqli->prepare("select first_name, last_name from employees
order by last_name");
if(!$stmt){
    printf("Query Prep Failed: %s\n", $mysqli->error);
    exit;
}

$stmt->execute();

$stmt->bind_result($first, $last);

echo "<ul>\n";
while($stmt->fetch()){
    printf("\t<li>%s %s</li>\n",
        htmlspecialchars($first),
        htmlspecialchars($last)
    );
}
echo "</ul>\n";
```

```
$stmt->close();
?>
```

Here, we use **$stmt->bind_result()** instead of **$stmt->bind_param()** because we are interested in the output from, not the input to, the query.

bind_result() and fetch()

Here, our query returns a table with two columns: first_name and last_name. So, we need to pass two parameters into **$stmt->bind_result()**: one to store the first name, and one to store the last name.

After we call **$stmt->bind_result()**, we call **$stmt->fetch()**. **$stmt->fetch()** stores the values from the next row in the result to the variables you specified in **$stmt->bind_result()**. **$stmt->fetch()** will continue binding rows into the variables until there are no rows left in the result set, at which time it will return null, thereby leaving the *while* loop.

Note: Use very unique variable names inside **$stmt->bind_result()**, because whatever variables you use will be overwritten. This includes session variables, which are sometimes linked to the local variable of the same name.

get_result() and fetch_assoc()

If you do not want to bind the results from your query to variables, there is an alternative option that returns the results in arrays.

Consider this code:

```
<?php
require 'database.php';

$stmt = $mysqli->prepare("select first_name, last_name from employees
order by last_name");
if(!$stmt){
    printf("Query Prep Failed: %s\n", $mysqli->error);
    exit;
}

$stmt->execute();

$result = $stmt->get_result();

echo "<ul>\n";
while($row = $result->fetch_assoc()){
    printf("\t<li>%s %s</li>\n",
        htmlspecialchars( $row["first_name"] ),
        htmlspecialchars( $row["last_name"] )
    );
}
echo "</ul>\n";

$stmt->close();
?>
```

Instead of binding the columns of the result set to variables, they are returned in an associative array. **$result->fetch_assoc()** returns a new array for each row in the result set, and when there are no more rows, it returns null, thereby exiting the loop.

Troubleshooting

If you are getting an error `call to undefined method mysqli_stmt::get_result()`, you probably installed the **php(5)-mysql** instead of the **php(5)-mysqlnd** package. To fix this, first uninstall the old package and then install the new one.

In Debian:

```
$ sudo apt-get remove php5-mysql
$ sudo apt-get install php5-mysqlnd
```

In RHEL:

```
$ sudo yum remove php-mysql
$ sudo yum install php-mysqlnd
```

Combined Queries

You can also perform queries that both take input and produce output; you simply call both **$stmt->bind_param()** and **$stmt->bind_result()**. For example, the following selects all employees in the department specified in the GET variable **dept**:

```
<?php
require 'database.php';

$dept = $_GET['dept'];

$stmt = $mysqli->prepare("select first_name, last_name from employees
where department=?");
if(!$stmt){
    printf("Query Prep Failed: %s\n", $mysqli->error);
    exit;
}

$stmt->bind_param('s', $dept);

$stmt->execute();

$stmt->bind_result($first, $last);

echo "<ul>\n";
while($stmt->fetch()){
    printf("\t<li>%s %s</li>\n",
        htmlspecialchars($first),
        htmlspecialchars($last)
    );
}
echo "</ul>\n";

$stmt->close();

?>
```