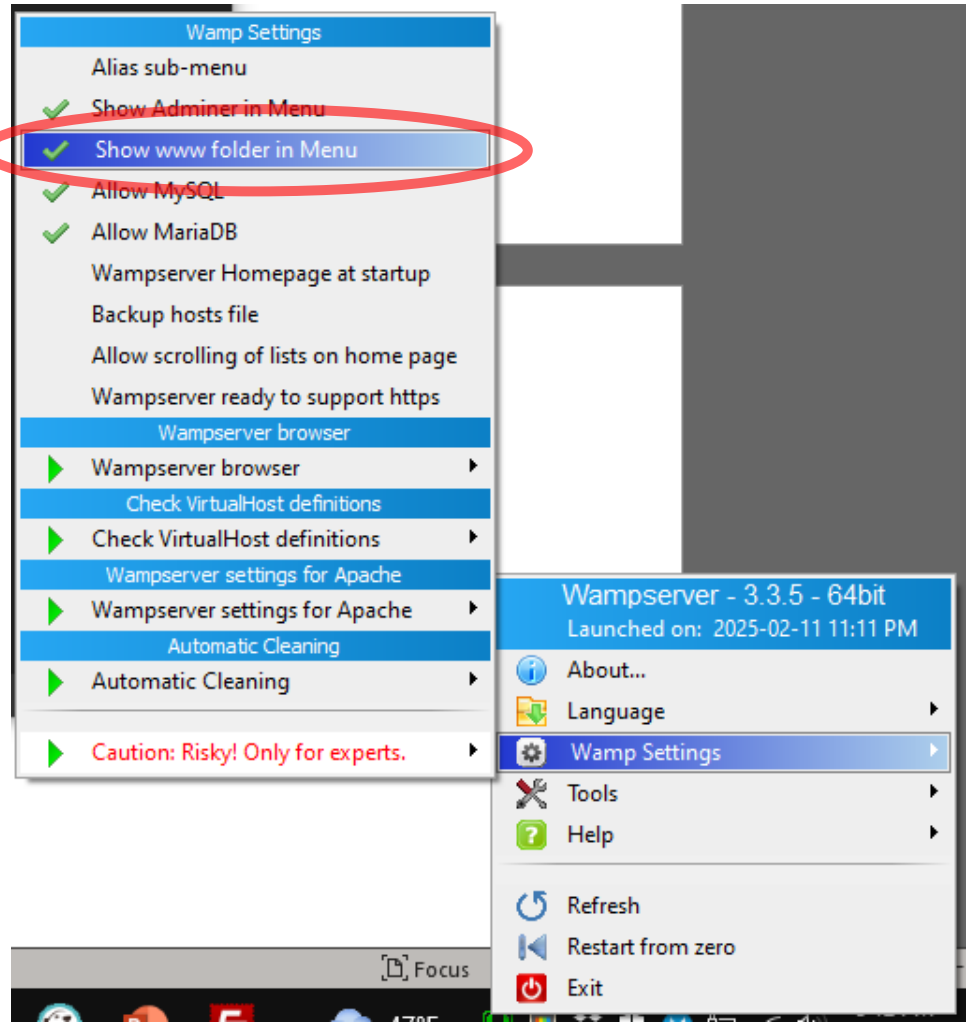
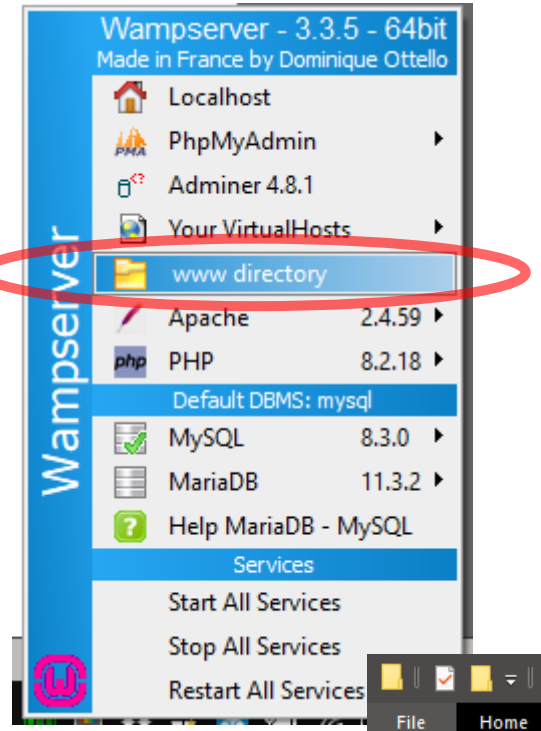
Lesson 7.6: Programmatic DB Communication

CSC430/530 – DATABASE MANAGEMENT SYSTEMS

Setup in WAMP

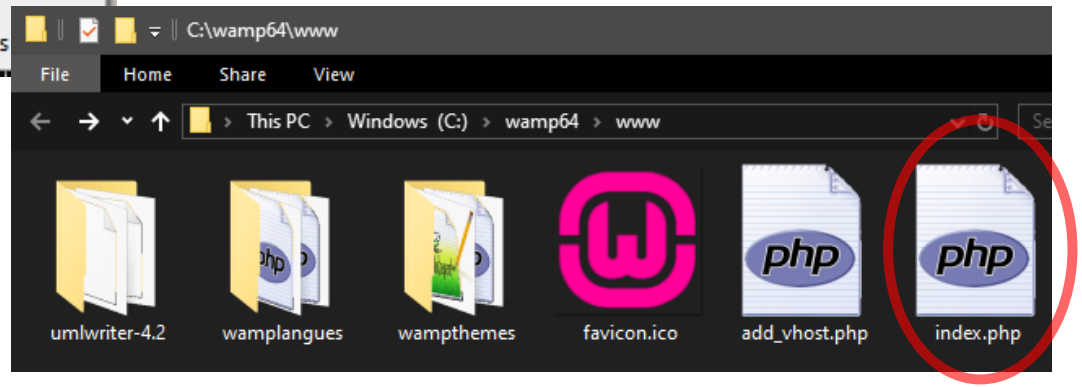


Right click
on WAMP



Left click
on WAMP

Navigate to "localhost"



Create index.php

PHP Hello World

```php
<?php

echo "Hello World <br />";

?>
```

PHP Basics

- Code must be inside `<?php ?>` tags
- "echo" used for printing
- Variables start with `$`
- Syntax similar to C
- Is object oriented
- Has hundreds of built in functions
- Is dynamically typed

```php
<?php
  require('../master.php');
  Master::putHeader(Page::$GRADE_VIEWER);
?>

<form action="index.php" method="post">
<?php
  require_once('students.php');
  require_once('../login.php');
  require_once('../profile.php');
  require_once('stats.php');
  require_once('../encryption.php');
  require_once('../gui.php');

  function viewGrade() {
    $profile = Login::getProfile();
    if ($profile->auth != AuthType::STUDENT) {
      echo "<h2>What are you doing here?<br />You are a Professor.<br />Impersonate a student to see their grades.</h2>";
      return;
    }

    GUI::getCourseSelectorResult($profile);
    GUI::writeCourseSelector($profile);

    $coursefile = $profile->getActiveCourse()->toFilename();
    $file = Master::$currentPage->rootPath . Files::GRADES . "{$coursefile}.data";
    $xml = Encryption::decryptXmlFile($file);
    if (!$xml) {
      die("<h3>No Grades Yet</h3>");
    }

    $node = null;
    Stats::load($xml);

    for($i = 0; $i < count($xml->Student); $i++) {
      $studName = $xml->Student[$i]['name'];
      if ($xml->Student[$i]['email'] == $profile->shortEmail) {
        $node = $xml->Student[$i];
        break;
      }
    }
    if ($node != null) {
      $hash = md5($node->asXML());
      $profile->setActiveCourseGradeHash($hash);
    }
    $student = new Student($studName, $profile->shortEmail);
    for($i = 0; $i < count($node->gi); $i++) {
      $item = $node->gi[$i];
      $gradedItem = new GradedItem($item, $i);
      $student->addGradedItem($gradedItem);
    }
    $student->display();
  }
  viewGrade();
  echo "<div style='clear:both; margin-bottom:16em;' />";
?>
</form>

<?php
  Master::putFooter();
?>
```

Talk to DB

- Use mysqli library

```php
<?php

// connection string settings
$server = 'localhost';
$user = 'username';
$pass = 'password';
$db = 'database_name';

// connect to database
$conn = new mysqli($server, $user, $pass, $db);

// check for errors
if ($conn->connect_error) {
    die('connection error: ' . $conn->connect_error);
}

?>
```

Run Simple Queries

```php
<?php

// construct and run query
$query = 'SELECT * FROM employee';
$result = $conn->query($query);

// error checking
if ($result->num_rows == 0) {
    echo 'No results from query';
}

$sqlError = mysqli_error($conn);
if (strcmp($sqlError, '')) {
    echo "Sql error: $sqlError";
}

// show results
while ($row = $result->fetch_assoc()) {
    echo "{$row['fname']} {$row['lname']}<br />";
}

?>
```

Better Modularity

```php
<?php
function select($attrs, $table) {
    // construct and run query
    $query = "SELECT $attrs FROM $table";
    $result = $conn->query($query);

    // error checking
    if ($result->num_rows == 0) {
        echo 'No results from query';
    }

    $sqlError = mysqli_error($conn);
    if (strcmp($sqlError, '')) {
        echo "Sql error: $sqlError";
    }

    // show results
    return $result;
}

// main
$result = select('*', 'employee');
while ($row = $result->fetch_assoc()) {
    echo "{$row['Fname']} {$row['Lname']}<br />";
}
?>
```

Using Triggers

```
/* SQL */
DELIMITER $$
CREATE TRIGGER bob_to_bobby
BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
    IF (NEW.fname = 'Bob') THEN
        SET NEW.fname = 'Bobby';
    END IF;
END $$
DELIMITER ;
```

```
// PHP
$query = "INSERT INTO employee VALUES ('bob', 'x', 'robert', 475869251, '2000-01-01',
'123 main, ruston, la', 'm', 10, NULL, 1)";
$result = $conn->query($query);
```

Using Views

```sql
/* SQL */
CREATE VIEW dept_5_emps AS
    SELECT *
    FROM employee
    WHERE dno = 5;
```

```php
// PHP
$query = 'SELECT * FROM dept_5_emps';
$result = $conn->query($query);

while ($row = $result->fetch_assoc()) {
    echo "{$row['fname']} {$row['lname']}<br />";
}
```

Using Stored Procedures

```sql
/* SQL */
DELIMITER $$
CREATE PROCEDURE give_raise_to_all_by_name_prefix(IN fname_part VARCHAR(10))
BEGIN
    UPDATE employee
    SET salary = salary * 1.1
    WHERE fname LIKE concat(fname_part, '%');
END $$
DELIMITER ;
```

```php
// PHP
$query = "CALL give_raise_to_all_by_name_prefix('a')";
$result = $conn->query($query);
```

Using Functions

```sql
/* SQL */
DELIMITER $$
CREATE FUNCTION highest_paid_emp_by_dept(dept_number INT)
RETURNS VARCHAR(50)
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE high_paid_emp varchar(50);
    SELECT concat(fname, ' ', lname) INTO high_paid_emp
    FROM employee
    WHERE dno = dept_number AND
          salary >= ALL ( SELECT salary FROM employee WHERE dno = dept_number );
    RETURN high_paid_emp;
END $$
DELIMITER ;
```

```php
// PHP
$query = 'SELECT highest_paid_emp_by_dept(5)';
$result = $conn->query($query);
echo 'Highest paid in dept 5 is ' . $result->fetch_array()[0];
```

Executing Multiple Queries

```php
<?php

$query = 'SELECT * FROM employee; SELECT * FROM dependent;';
$conn->multi_query($query);
echo '<h2>Results: </h2>';
do {
    if ($result = $conn->store_result()) {
        while ($row = $result->fetch_row()) {
            echo implode(',', $row) . '<br />';
        }
    }
    if ($conn->more_results()) {
        echo '<br />';
    }
} while ($conn->next_result());

?>
```

Watch Out! SQL Injection

- SQL injection is the process of getting a WHERE clause to succeed on row(s) that it wasn't meant to succeed on
- This gives an unauthorized user access to information they should NOT have
- It can even cause entire tables or databases to be dropped when they shouldn't have been!
- It is extremely easy to do if the developer hasn't protected against it!

Watch Out! SQL Injection

```
<html> <head> </head> <body>
<?php

if (isset($_POST['fname'])) {
    // connection string settings
    $server = 'localhost';
    $user = 'root';
    $pass = '';
    $db = 'company';

    // connect to database
    $conn = new mysqli($server, $user, $pass, $db);

    // check for errors
    if ($conn->connect_error) {
        die("connection error: " . $conn->connect_error);
    }

    $fname = $_POST['fname'];
    $lname = $_POST['lname'];
    $ssn = $_POST['ssn'];

    // PHP
    $query = "SELECT fname, lname, salary FROM employee WHERE fname = '$fname' AND lname = '$lname' AND ssn = '$ssn'"; // this is problematic!
    $conn->multi_query($query);
    $result = $conn->store_result();
    echo '<h2>Results: </h2>';
    while ($row = $result->fetch_assoc()) {
        echo "<u>Name:</u> <strong>{$row['fname']} {$row['lname']}</strong>    <u>Salary:</u> <strong>{$row['salary']}</strong><br />";
    }
}
else {
    echo "<form action='index.php' method='post'>";
    echo "  <label for='fname'>First Name: </label><input type='text' id='fname' name='fname' /><br />";
    echo "  <label for='lname'>Last Name: </label><input type='text' id='lname' name='lname' /><br />";
    echo "  <label for='lname'>SSN: </label><input type='text' id='ssn' name='ssn' /><br />";
    echo "  <input type='submit' />";
    echo "</form>";
}
?>
</body> </html>
```

Watch Out! SQL Injection

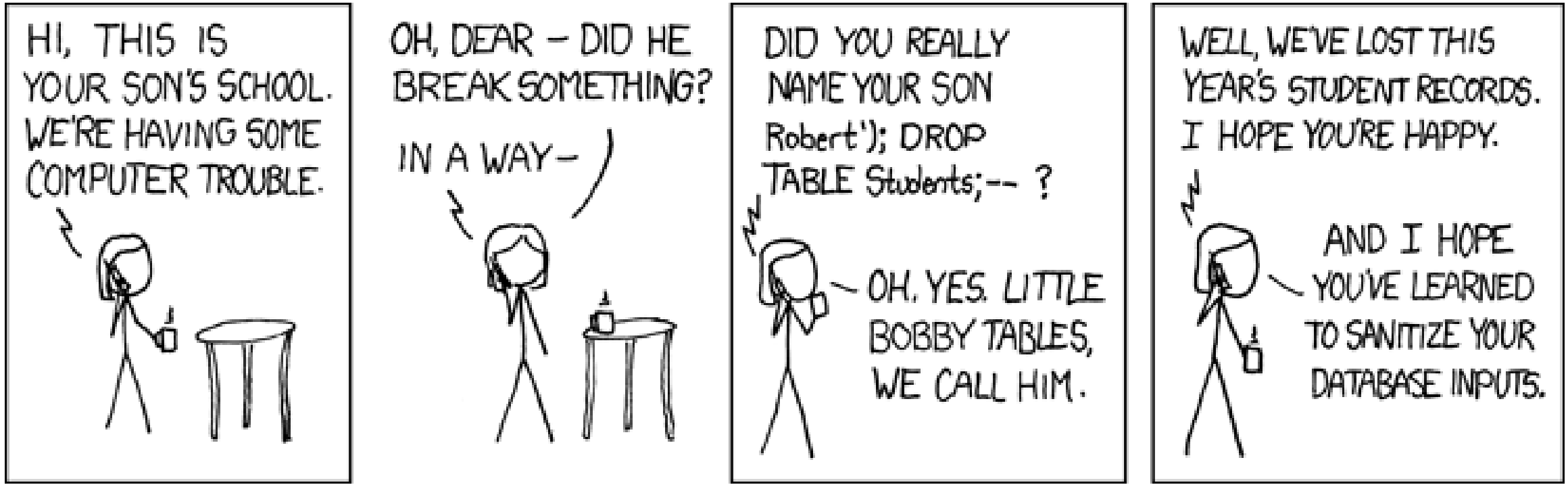
Demo

Even the docs warn against this!
https://www.php.net/manual/en/mysqli.multi-query.php

Ways to Protect Against SQL Injection

- Never use multi_query() function
  - Although this doesn't prevent a single bad query from running
- Sanitize input
  - Look for bad characters (like single/double quotes, two dashes, semi colons) and remove them before adding them to the query
  - PHP has some built in functions to help with this
- Use mysqli prepared statements
  - Allows you to write your query with placeholders for the user information
  - You then give it the user information in a separate function call, along with the data types you expect for each piece of information
  - It double checks the datatypes, and encloses strings in quotes so they can't become part of the query itself
- Do not login as root!
  - Login as a user with only the privileges that are necessary for the app to work
- There are many more techniques
  - This is a pretty good reference: https://websitebeaver.com/prepared-statements-in-php-mysqli-to-prevent-sql-injection

Ways to Protect Against SQL Injection



Good Practice

- Have a separate DB class that handles all database interactions
- Can connect in constructor and save the connection as a field
- Can have methods for all database interactions / queries
- Use require_once to include this script in multiple pages

```php
require_once('db.php');
```

Good Practice

```php
class DB {
  // fields
  private static $server = 'localhost';
  private static $user = 'user';
  private static $pass = 'pass';
  private static $db = 'database';
  private $conn = null;

  // constructor
  function __construct() {
      $this->conn = new mysqli(DB::$server, DB::$user, DB::$pass, DB::$db);
      if ($this->conn->connect_error) {
          die("connection error: " . $this->conn->connect_error);
      }
  }
  function select($attrs, $table) {
      // construct and run query
      $query = "SELECT $attrs FROM $table";
      $result = $this->conn->query($query);

      // error checking
      if ($result->num_rows == 0) {
          echo 'No results from query';
      }
      $sqlError = mysqli_error($conn);
      if (strcmp($sqlError, '')) {
          echo "Sql error: $sqlError";
      }

      // show results
      return $result;
  }
}
```

Other Languages – Ruby (on rails)

- Prerequisites:
  - Ruby on Rails installed
  - MySQL server running locally
- Install:
  - Install the mysql2 gem
  - bundle add mysql2
- Configure file config/database.yml

```
default: &default
adapter: mysql2
encoding: utf8mb4
pool: 5
username: root
password: pass
host: localhost
port: 3306
development:
  <<: *default
database: company
```

- Create an Employee model
  - rails generate model Employee
- Run the query in a controller or Rails console

```
employees = Employee.all
employees.each {
    |emp| puts emp.inspect
}
```

Other Languages – Node.js

- Prerequisites:
  - Node.js installed
  - MySQL server running locally
- Install:
  - Install the mysql2 package
  - npm install mysql2

- Create a connection and execute the query

```
const mysql = require('mysql2/promise');
async function queryEmployees() {
  const connection = await mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'pass',
    database: 'company',
    port: 3306
  });
  const [rows] = await connection.execute(
                                    'SELECT * FROM employee');
  console.log(rows);
  await connection.end();
}
queryEmployees();
```

Other Languages – Blazor

- Prerequisites:
  - NET SDK installed
  - MySQL server running locally
  - Pomelo.EntityFrameworkCore.MySql (NuGet package)
- Configure the database context in Program.cs:

```
string connStr = "server=localhost;port=3306;user=root;password=pass;database=company";

builder.Services.AddDbContext<CompanyContext>(
    options => options.UseMySql(connStr,
        ServerVersion.AutoDetect(connStr)
    )
);
```

Other Languages – Blazor

- Define the Employee model and CompanyContext

```
public class Employee {
    public int Id { get; set; }
    public string Name { get; set; }
}
public class CompanyContext : DbContext {
    public DbSet<Employee> Employees { get; set; }
}
```

- Run the query in a Razor component or service

```
@inject CompanyContext _context
@code {
    private List<Employee> employees = new();
    protected override async Task OnInitializedAsync() {
        employees = await _context.Employees.ToListAsync();
    }
}
```

Other Languages – ASP.Net

- Prerequisites:
  - NET SDK installed
  - MySQL server running locally
  - Pomelo.EntityFrameworkCore.MySql (NuGet package)
- Configure the connection in appsettings.json:

```
{ "ConnectionStrings": {
      "CompanyDB": "server=localhost;port=3306;user=root;password=pass;database=company"
  }
}
```

- Define the Employee model and CompanyContext

```
public class Employee {
    public int Id { get; set; }
    public string Name { get; set; }
}
public class CompanyContext : DbContext {
    public CompanyContext(DbContextOptions<CompanyContext> options) : base(options) { }
    public DbSet<Employee> Employees { get; set; }
}
```

Other Languages – ASP.Net

- Run the query in a controller

```
[ApiController]
[Route("[controller]")]
public class EmployeeController : ControllerBase {
    private readonly CompanyContext _context;

    public EmployeeController(CompanyContext context) {
        _context = context;
    }

    [HttpGet]
    public IEnumerable<Employee> Get() {
        return _context.Employees.ToList();
    }
}
```

Other Languages – Python (with Flask)

- Prerequisites:
  - Python installed
  - MySQL server running locally
- Install packages:
  - pip install Flask mysql-connector-python
- Create a Flask app

```
from flask import Flask
import mysql.connector
app = Flask(__name__)
@app.route('/employees')
def get_employees():
    connection = mysql.connector.connect(host='localhost', user='root', password='pass',
                                         database='company', port=3306)
    cursor = connection.cursor(dictionary=True)
    cursor.execute('SELECT * FROM employee')
    employees = cursor.fetchall()
    cursor.close()
    connection.close()
    return {'employees': employees}
app.run()
```