

# CS 547

## Week 5 Day 1 Site Structure

-

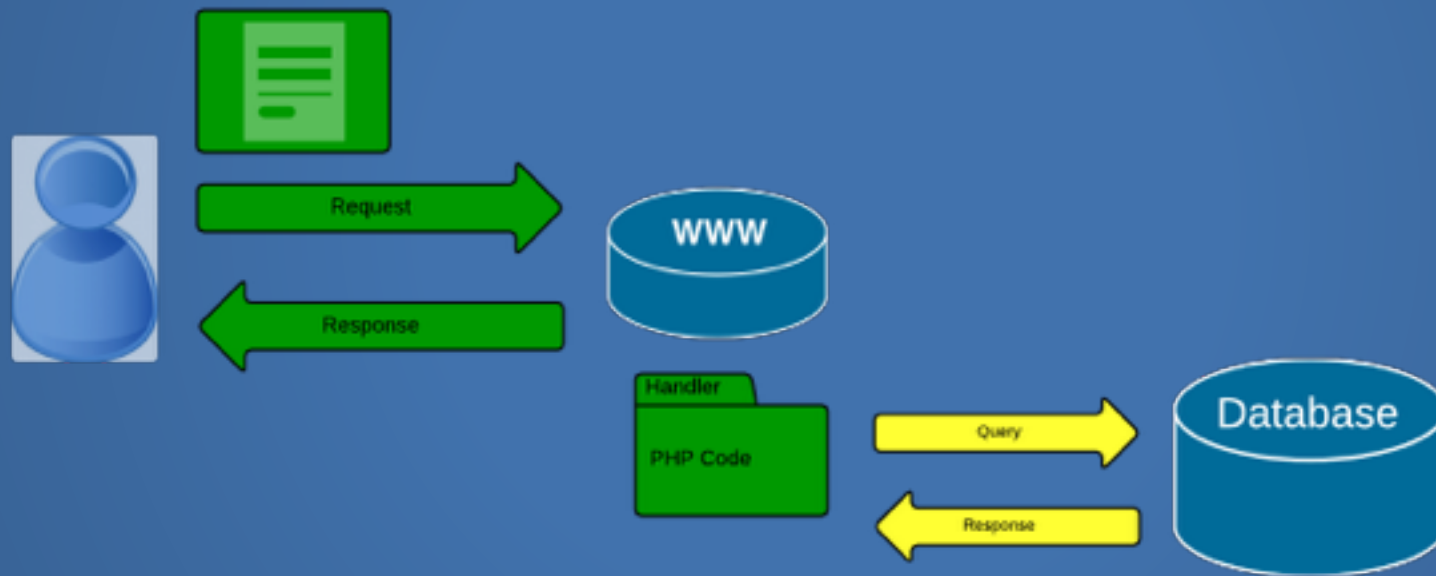
# Agenda

- PHP Session
- Cookies

# Announcement

- No Class on Feb 19, 2015

# Review



# MySQLi Basics

```
mysqli_conn();  
mysqli_select();  
mysqli_insert();  
mysqli_update();  
mysqli_delete();
```

# MySQLi Prepared Statements

Why?

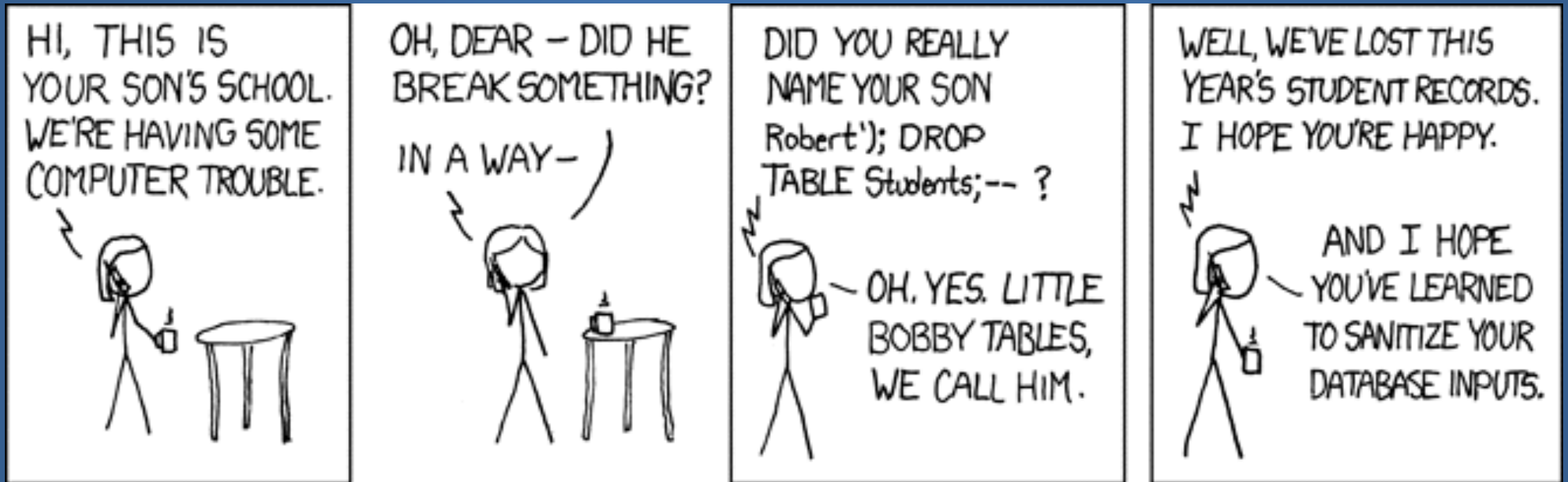
Used to mitigate against SQL injections.

Benefits

Reduce time and minimize bandwidth

- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. ***If the original statement template is not derived from external input, SQL injection cannot occur.***

# Prepared Statements: Little Bobby Tables



# SQL Injection: The problem

```
//
if ($conn->connect_error) {
    trigger_error('Database connection failed: ' . $conn->connect_error, E_USER_ERROR);
}
else {
$sql='SELECT * FROM mytable WHERE (pkid = "" . $_POST["name"] . "") ; ';
    $rs=$conn->query($sql);
    if($rs === false) {
        trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $conn->error, E_USER_ERROR);
    } else {
        $rows_returned = $rs->num_rows;
        echo ("Found " . $rows_returned . " rows.<hr>");
        $rs->data_seek(0);
        while($row = $rs->fetch_assoc()){
            echo ("Name = " . $row['name'] . " age = " . $row['realAge'] . "<br>" )
        }
    }
}
```



# SQL Injection: The problem

- With malformed strings like  
2' or 1; select \* from mytable ;

We get results we didn't expect

# Demo

SQL Injection Example

localhost/xampp/Demos/demo\_sqlsec.html

## SQL Injection Demo

Name:

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.

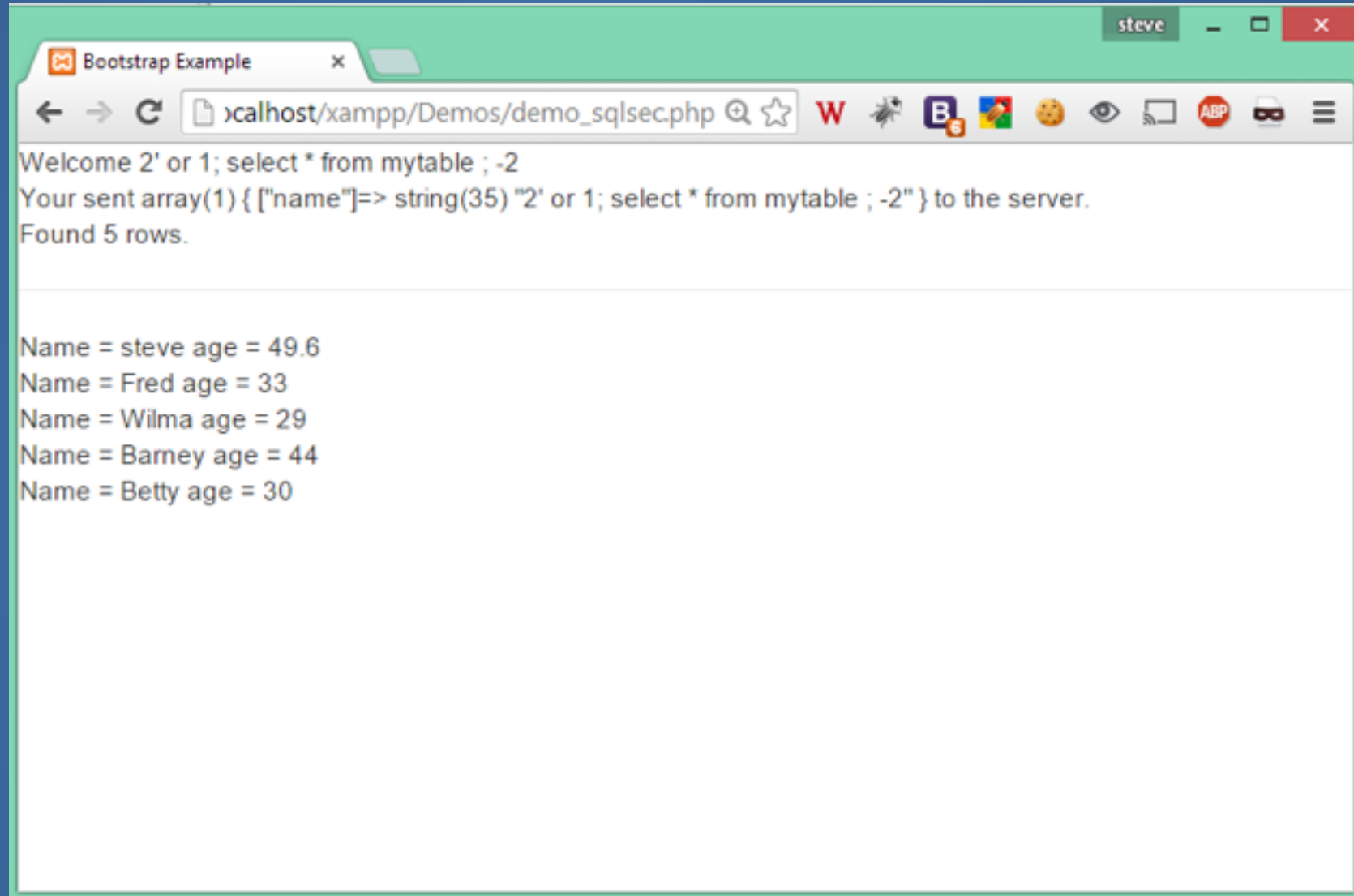
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.

AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

This is a demonstration of how to prevent SLQ injection such as in the above xkcd comic .

# Demo

Not what you  
want.



# SQL Injection Mitigation

The solution to SQL injection types of attacks is complicated and requires vigilance and understanding on the part of the programmer.

**One can not blindly rely on a one size fits all solution.**

You must understand what your code is doing and design your code appropriately.

# SQL Injection Mitigation

Techniques:  
Prepared Statements  
Sanitizing Input

# Prepared Statements and Bound Parameters

## Lifecycle

- Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?").  
Example: **INSERT INTO mytable VALUES(?, ?, ?)**
- Parse: The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result **without executing it**
- Execute: At a later time, the application **binds the values to the parameters**, and the database executes the statement. The application may execute the statement as many times as it wants with different values

# SQL Injection Solution

```
• if ($stmt = $conn->prepare("SELECT pkID, name, age, realAge, phone, email FROM mytable WHERE
                                name=?")) {
•
•     $stmt->bind_param("s", $_POST["name"]); // Bind a variable to the parameter as a string.
•
•     $stmt->execute(); // Execute the statement.
•
•     $stmt->bind_result($pkID, $name, $age, $realAge, $phone, $email);
•
•     $stmt->store_result(); /* Store the result (to get properties) */
•     $num_of_rows = $stmt->num_rows;
•
•     /* Bind the result to variables */
•     $stmt->bind_result($pkID, $name, $age, $realAge, $phone, $email);
•
•     $stmt->fetch(); // Fetch the data.
•
•     // Display the data.
•     printf("ID %s is %s. They are %s, but really they are %s", $name, $pkID, $age, $realAge);
•
• }
```

# Caution about User Input

- Never trust user Input.
- All user input should be sanitized.
- Use [mysqli\\_real\\_escape\\_string\(\);](#)
- or [PDO::quote\(\);](#)



# Sanitizing User input

`mysqli_real_escape_string` -- Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection

Parameters

**link**

Procedural style only: A link identifier returned by `mysqli_connect()` or `mysqli_init()`

**escapestr**

The string to be escaped. Characters encoded are NUL (ASCII 0), `\n`, `\r`, `\`, `'`, `"`, and Control-Z.

Returns: the escaped string;

# What if you want to validate input?

To validate user input you must match the input to a pattern. PHP uses Regular Expressions (called RegEx) to do this. A regex is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations

# PHP Regular Expressions

`preg_match()` — Perform a regular expression match

```
int preg_match ( string $pattern , string $);
```

```
int preg_match ( string $pattern , string $subject [, array &$matches [,  
int $flags = 0 [, int $offset = 0 ]]] );
```

# Regular Expressions

## Validate Name

```
$name = test_input($_POST["name"]);  
if (preg_match("/^[a-zA-Z ]*$/", $name)) {  
    echo ("Found a name");  
}
```

# RegEx Patterns

```
if (preg_match("/^[a-zA-Z]*$/", $name))
```

the yellow text is the important part.

# RegEx Patterns

- Operator list (the pattern to match) is contained in the / /  
Simple characters match
  - /a/ -> a
  - /b/ -> b
  - /cd/ -> cd

# RegEx

- Square brackets denote a class of characters  
/[ab]/ -> a , ab , b, and ba but not A

# RegEx

- A dash (-) inside square brackets denote a range of characters

`/[a-z]/` -> all lower case characters but not A-Z



# Reg Ex

The ^, \$ \* and . have special meanings

^ = Line start

\$ =Line end

\* = 0 or more time

. = any character except a new line

# Reg Ex: Character Classes

## Character classes

.	any character except newline
\w \d \s	word, digit, whitespace
\W \D \S	not word, digit, whitespace
[abc]	any of a, b, or c
[^abc]	not a, b, or c
[a-g]	character between a & g

# Reg Ex Boundaries

<code>^abc\$</code>	start / end of the string
<code>\b</code>	word boundary

# RegEx: Escaped Characters

`\. \* \\\`

escaped special characters

`\t \n \r`

tab, linefeed, carriage return

`\u00A9`

unicode escaped ©

# RegEx: Quantifiers

<code>a* a+ a?</code>	0 or more, 1 or more, 0 or 1
<code>a{5} a{2,}</code>	exactly five, two or more
<code>a{1,3}</code>	between one & three
<code>a+? a{2,}?</code>	match as few as possible
<code>ab cd</code>	match ab or cd

# Regular Expressions

- Take some time to learn by example.
- visit the site <http://www.regexr.com/>

# Regular Expressions

- Rich amount of regular expression functions in the php library. Check out the [PCRE](#) pages on php.net. Of interest may be
  - `preg_match();` and `preg_match_all();`
  - `preg_filter();` and `preg_replace();`
  - `and`
  - `preg_split();`





# PHP Sessions

Sessions are used to preserve data across page visits. Sessions can either be stored with the URL or in a cookie on the client.

Sessions are created with the `session_start()` function and are accessible through the super global `$_SESSION` variable;

# Session Code Demo

# PHP Cookie

- Cookies store information on the client.

# PHP Cookie Demo