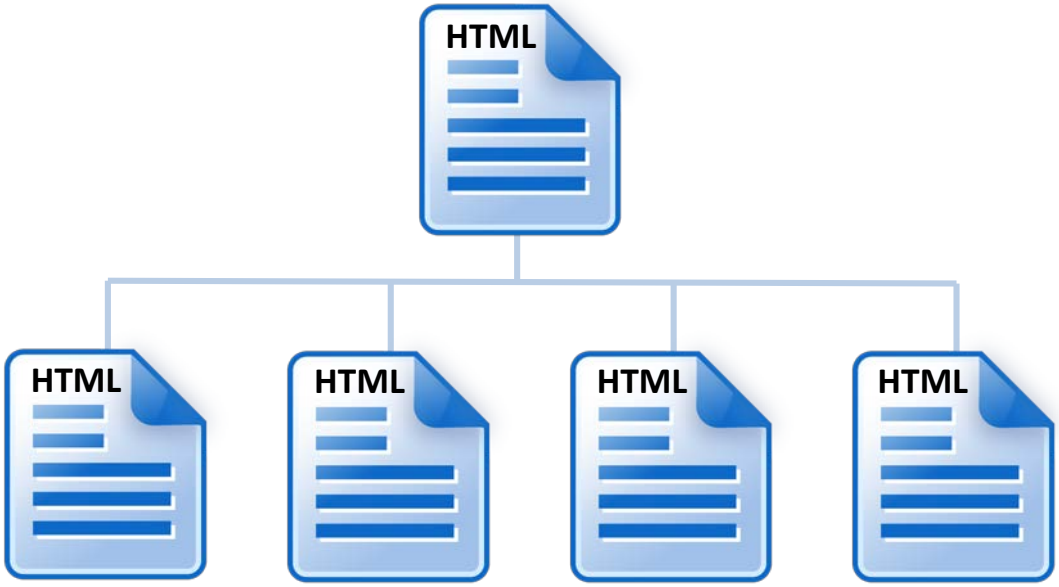
PHP with URL Parameters

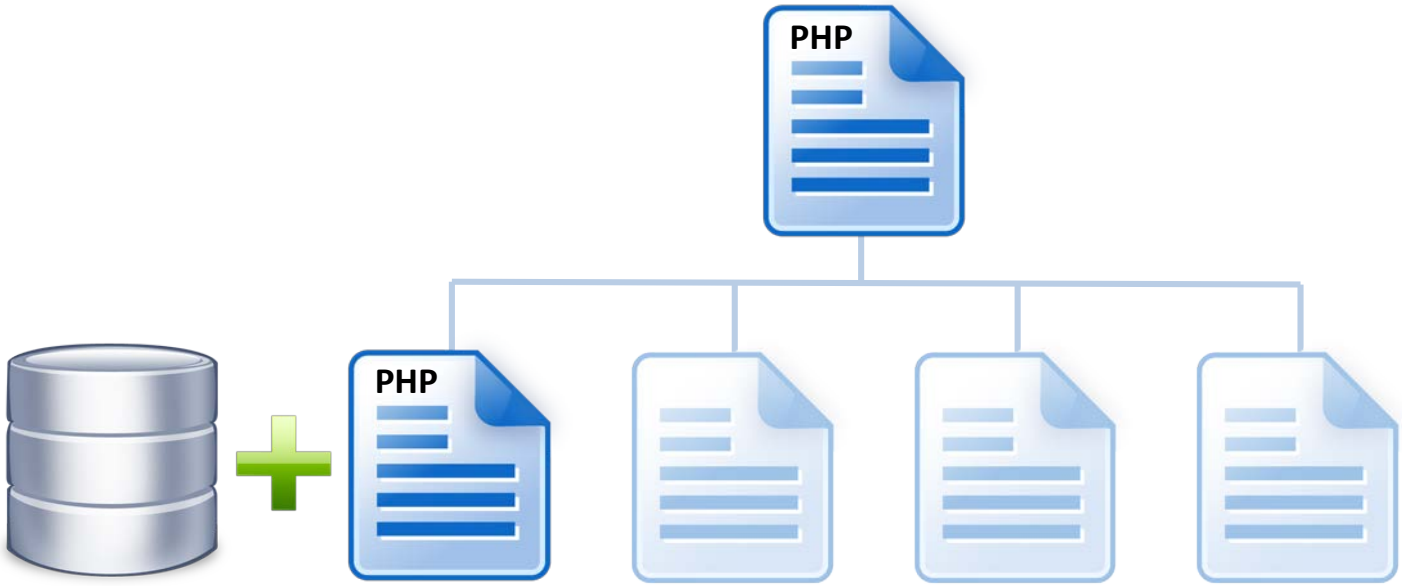
Website Planning

Traditional, static website



Every page of a website is rendered using a unique HTML file.

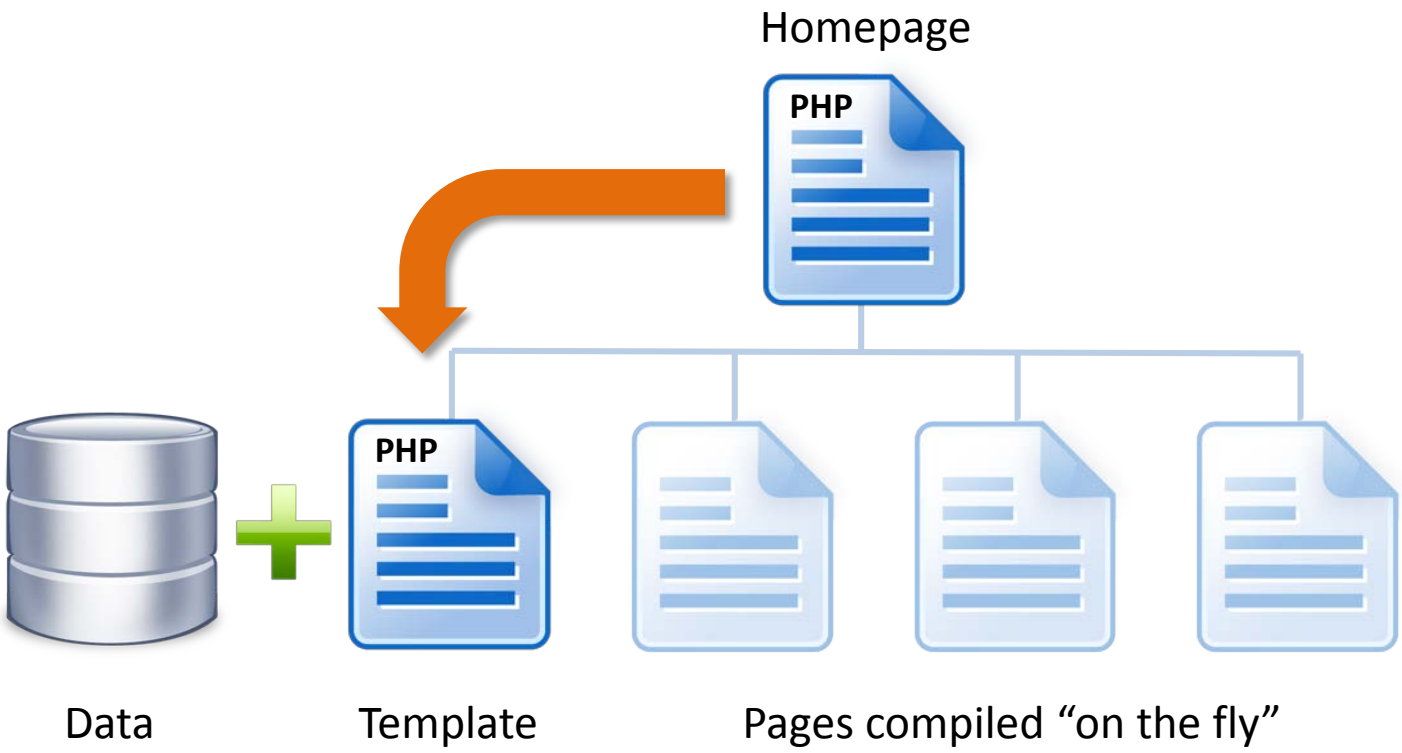
Template-driven, dynamic website



Data Template Pages compiled “on the fly”

Template-driven sites use one or more template files to generate multiple pages by pulling data from a database.

How do links work?



How can the homepage file tell the template file which bit of data it should use in order to compile the required page?

Passing data between scripts

Data can be passed from one PHP page to another using a *URL parameter*. This is a bit of data (a number or a text string) which is added to the address of the template file. In the example above, the value “8” is being sent and is identified by the name “id”, just like assigning a value to a variable. The question mark character “?” is used as a separator between the address and the parameter.

How does a script access the data?

All data passed to a file using a URL parameter is stored in a special super global array called $_GET. In the example above, that data can be assigned to a variable like this:

```
$article_id = $_GET['id'];
```

Multiple values

article.php?id=2&title=good

Homepage

Template

URL parameters can be used to send multiple values at once. The ampersand character “&” is used as a separator between name and value pairs. In the example above, id has a value of “2” and title has the value “good”. $_GET can contain multiple values because it is an array and not just a simple variable.

Validate that data!

Homepage

Template

Any data passed to a script via a URL parameter should be considered potentially dangerous because it can easily be tampered with. In the above example, the “8” could easily be changed to something else, including a fragment of PHP! It must therefore be *validated* before it can be used safely.

Check the data type

An easy way to validate a URL parameter is to check the data type. In this example, the data should be an integer and we can check for that using the ctype_digit function:

```
ctype_digit($_GET['id'])
```

PHP contains a number of functions for data validation…

Testing for an integer

```
if (isset($_GET['id']) && filter_var($_GET['id'], FILTER_VALIDATE_INT)) {
    $article=$_GET['id'];
}else{
    header('HTTP/1.0 404 Not Found');
    exit("<h1>Not Found</h1>\n<p>The submitted data is not valid.</p>");
}
```

The above if/else statement checks the incoming data and reacts depending on whether the data looks OK or not. It uses the isset function to check whether $_GET['id'] contains a value and the filter_var function is used to check that the value is an integer. If the test is passed, the data is assigned to the variable $article. If the test is not passed, a 404 error is generated, the script is terminated and a message is printed. The filter_var function was introduced in PHP 5.2 but the ctype_digit function can be used in the same way for earlier versions of PHP.

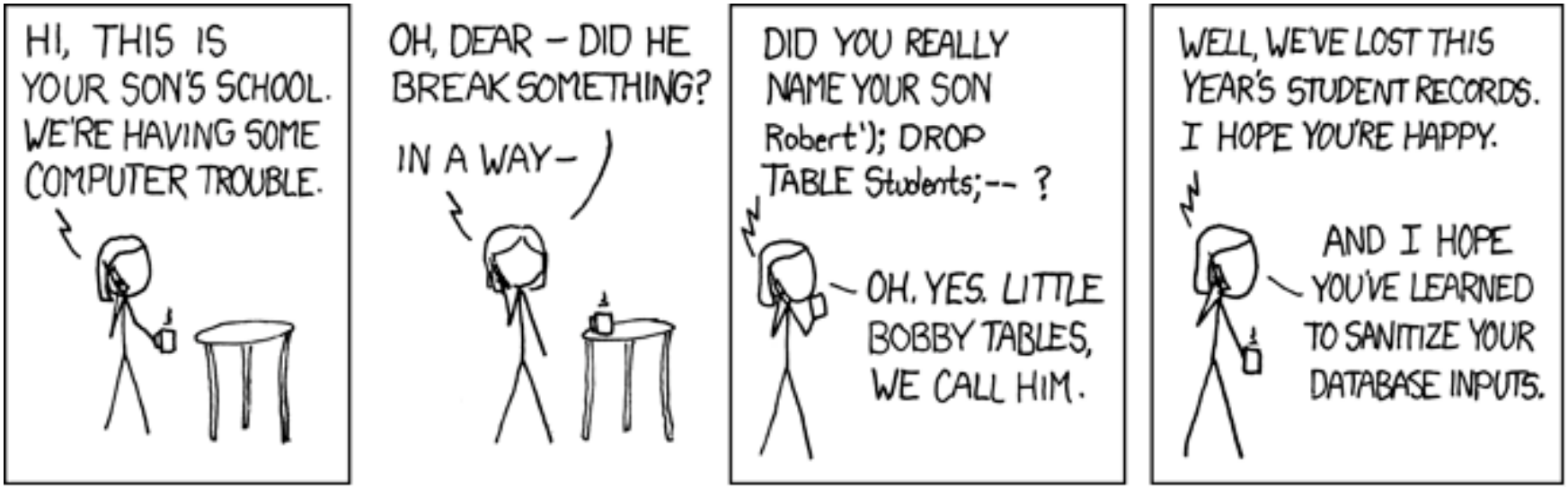
Testing for other data types

Testing for an integer is very easy but testing for other data types is more difficult. Say we wanted to pass a text string. We can check that it is a string but that still doesn't tell us whether the string is good or bad, so we have to be extra careful with string data and all strings should be *sanitized* to remove or escape suspicious characters such a quotes.

Sanitizing string data

```
if (isset($_GET['id']) && filter_var($_GET['id'], FILTER_SANITIZE_STRING)) {
    $article=$_GET['id'];
}else{
    header('HTTP/1.0 404 Not Found');
    exit("<h1>Not Found</h1>\n<p>The submitted data is not valid.</p>");
}
```

One of the first things a hacker will do to test for vulnerability is modify the URL parameter to include a quote character. This could be used to prematurely terminate the query string and insert some malicious code into your script. The FILTER_SANITIZE_STRING option will encode all quotes so that the script interprets them as part of the string and not as a string termination character. There are many methods for sanitizing strings depending on the expected output but you must at least deal with quotes in order to combat *SQL injection*.



Just using:
filter_var($var, FILTER_SANITIZE_STRING)
would have avoided this problem because the string:
Robert'); DROP TABLE students;--
would have been sanitized as the harmless:
Robert\'); DROP TABLE students;--

Cartoon by xkcd

Warning

Never use the GET method to send sensitive data between scripts because any data sent will be clearly visible in the URL.

Always validate and sanitise any data received via the GET method; it may have been tampered with and should be considered potentially harmful.

slideshow.php?status=end