CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC
No.132, AECS Layout
I.T. Park Road, Bangalore 560 037
T: +91 80 28524466/77, F : +91 80 28524630
E: info@cmrit.ac.in I www.cmrit.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

WEB TECHNOLOGY & ITS APPLICATIONS

IAT - 3 Solution

1. **With data flow diagrams, explain the role of PHP's $_GET and $_POST arrays. [10 Marks]**

$_GET and $_POST Super global Arrays

The $_GET and $_POST arrays are the most important super global variables in PHP since they allow the programmer to access data sent by the client in a query string.

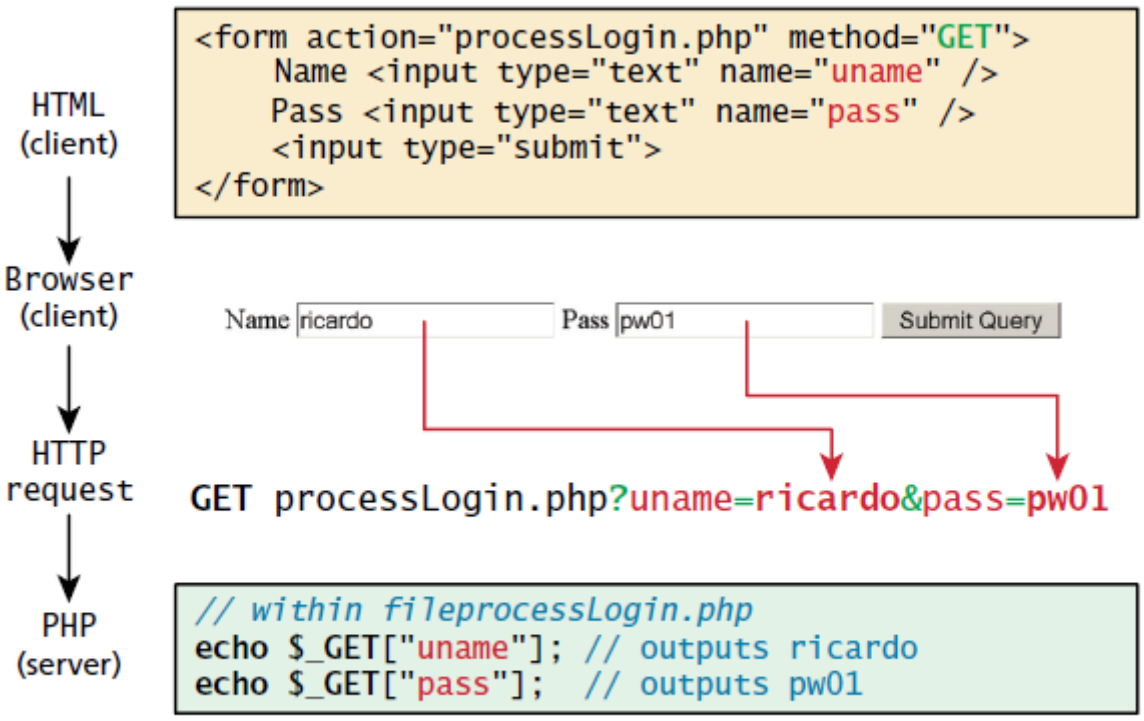

FIGURE 9.5 Illustration of flow from HTML, to request, to PHP's $_GET array

An HTML form (or an HTML link) allows a client to send data to the server. That data is formatted such that each value is associated with a name defined in the form. If the form was submitted using an HTTP GET request, then the resulting URL will contain the data in the query string. PHP will populate the superglobal $_GET array using the contents of this query string in the URL.

If the form was sent using HTTP POST, then the values would not be visible in the URL, but will be sent through HTTP POST request body. From the PHP programmer's perspective,

almost nothing changes from a GET data post except that those values and keys are now stored in the $_POST array.

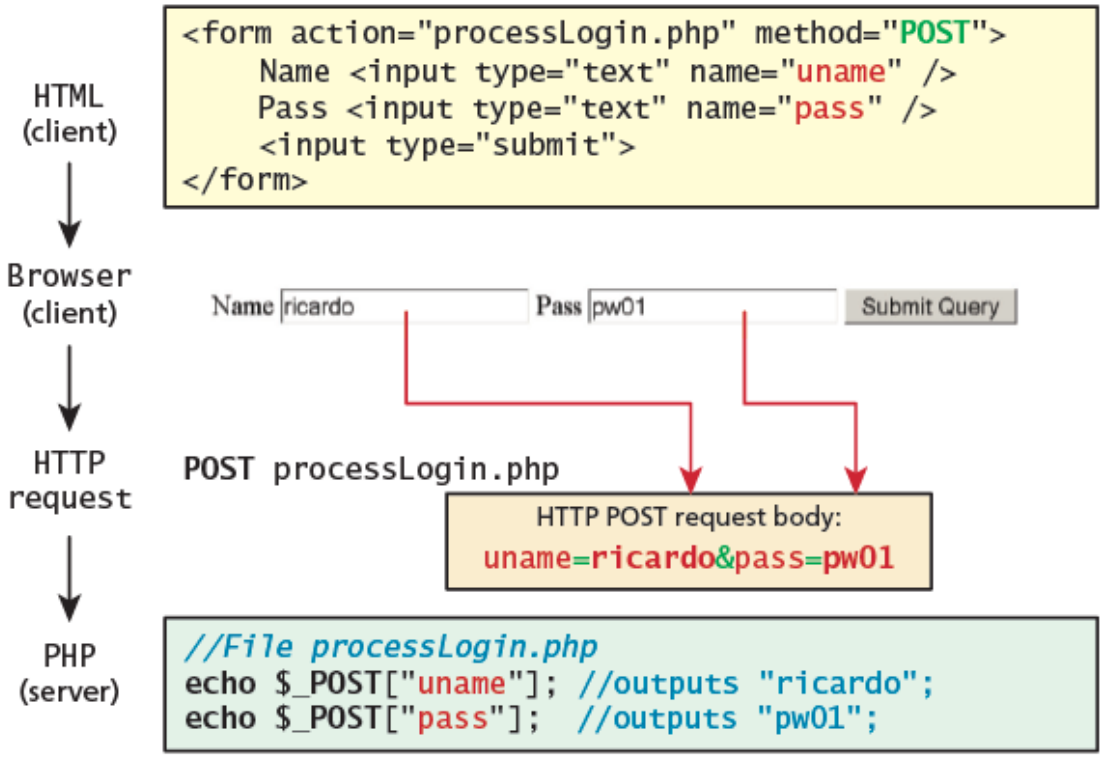

FIGURE 9.6 Data flow from HTML form through HTTP request to PHP's $_POST array

Determining If Any Data Sent

PHP that you will use the same file to handle both the display of a form as well as the form input. For example, a single file is often used to display a login form to the user, and that same file also handles the processing of the submitted form data, as shown in Figure 9.8. In such cases you may want to know whether any form data was submitted at all using either POST or GET.

2. **Explain procedural error handling and object oriented execution handling with suitable code segments. [10 Marks]**

Procedural Error Handling

In the procedural approach to error handling, the programmer needs to explicitly test for error conditions after performing a task that might generate an error. $connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);

$error = mysqli_connect_error();

if ($error != null) {

// handle the error

...

}

Object-Oriented Exception Handling

When a runtime error occurs, PHP *throws* an *exception*. This exception can be *caught* and handled either by the function, class, or page that generated the exception or by the code that called the function or class. If an exception is not caught, then eventually the PHP

environment will handle it by terminating execution with an “Uncaught Exception” message.4

```
// Exception throwing function
function throwException($message = null,$code = null) {
throw new Exception($message,$code);
}
try {
// PHP code here
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME)
or throwException("error");
//...
}
catch (Exception $e) {
echo ' Caught exception: ' . $e->getMessage();
echo ' On Line : ' . $e->getLine();
echo ' Stack Trace: '; print_r($e->getTrace());
} finally {
// PHP code here that will be executed after try or after catch
}
```

The finally block is optional. Any code within it will always be executed *after* the code in the try or in the catch blocks, even if that code contains a return statement. It is typically used if the developer wants certain things done regardless of whether an exception occurred, such as closing a connection or removing temporary files.

```
function processArray($array)
{
// make sure the passed parameter is an array with values
if ( empty($array) ) {
throw new Exception('Array with values expected');
}
// process the array code
...
}
```

One possible strategy for such a scenario is to throw an exception:

```
public function setBirthDate($birthdate){
// set variable only if passed a valid date string
if ( $timestamp = strtotime($birthdate) ) {
$this->birthDate=$timestamp;
}
else {
throw new Exception("Invalid Date in Artist->setBirthDate()");
}
}
try {
// PHP code here
}
catch (Exception $e) {
// do some application-specific exception handling here
...
// now rethrow exception
throw $e;
```

}

4. Explain 3 approaches to restrict file size in file upload with suitable code segments. (10 Marks)

File Size Restrictions

Some scripts limit the file size of each upload. There are three main mechanisms for maintaining uploaded file size restrictions:

HTML in the input form

JavaScript in the input form

PHP coding

HTML in the input form

This mechanism is to add a hidden input field before any other input fields in your HTML form with a name of MAX_FILE_SIZE. This technique allows your php.ini maximum file size to be large, while letting some forms override that largelimit with a smaller one.

```
<form enctype='multipart/form-data' method='post'>
<input type="hidden" name="MAX_FILE_SIZE" value="1000000" />
<input type='file' name='file1' />
<input type='submit' />
</form>
```

JavaScript in the input form

The more complete client-side mechanism to prevent a file from uploading if it is too big is to pre validate the form using JavaScript.

```
<script>
var file = document.getElementById('file1');
var max_size = document.getElementById("max_file_size").value;
if (file.files && file.files.length ==1){
if (file.files[0].size > max_size) {
alert("The file must be less than " + (max_size/1024) + "KB");
e.preventDefault();
}
}
</script>
```

PHP coding

This mechanism is to add a simple check on the server side.

```
$max_file_size = 10000000;
foreach($_FILES as $fileKey => $fileArray) {
if ($fileArray["size"] > $max_file_size) {
echo "Error: " . $fileKey . " is too big";
}
printf("%s is %.2f KB", $fileKey, $fileArray["size"]/1024);
}
```

5. With suitable examples, explain all JQuery selectors. [10 Marks]

Basic Selectors

The four basic selectors were defined back in Chapter 3, and include the universal selector, class selectors, id selectors, and elements selectors. To review:

■ $("*") Universal selector matches all elements (and is slow).
■ $("tag") Element selector matches all elements with the given element name.
■ $(".class") Class selector matches all elements with the given CSS class.
■ $("#id") Id selector matches all elements with a given HTML id attribute.

For example, to select the single element with id="grab" you would write:

```
var singleElement = $("#grab");
```

To get a set of all the <a> elements the selector would be:

```
var allAs = $("a");
```

These selectors are powerful enough that they can replace the use of getElementById() entirely.

The implementation of selectors in jQuery purposefully mirrors the CSS specification, which is especially helpful since CSS is something you have learned and used throughout this book.

In addition to these basic selectors, you can use the other CSS selectors that were covered in Chapter 3: attribute selectors, pseudo-element selectors, and contextual selectors as illustrated in Figure 15.4. The remainder of this section reviews some of these selectors and how they are used with jQuery.

Attribute Selector

An attribute selector provides a way to select elements by either the presence of an element attribute or by the value of an attribute. Chapter 3 mentioned that not all

```
<body>
<nav>
<ul>
<li><a href="#">Canada</a></li>
<li><a href="#">Germany</a></li>
<li><a href="#">United States</a></li>
</ul>
</nav>
<div id="main">
Comments as of <time>November 15, 2012</time>
<div>
```

```
<p>By Ricardo on <time>September 15, 2012</time></p>
<p>Easy on the HDR buddy.</p>
</div>
<hr/>
<div>
<p>By Susan on <time>October 1, 2012</time></p>
<p>I love Central Park.</p>
</div>
<hr/>
</div>
<footer>
<ul>
<li><a href="#">Home</a> | </li>
<li><a href="#">Browse</a> | </li>
</ul>
</footer>
</body>
$("ul a:link")
$("#main>time")
$("#main time")
$("#main div p:first_child")
```

Figure 15.4 Illustration of some jQuery selectors and the HTML being selected

browsers implemented it. jQuery overcomes those browser limitations, providing the ability to select elements by attribute. A list of sample CSS attribute selectors was given in Chapter 3 (Table 3.4), but to jog your memory with an example, consider a selector to grab all <img> elements with an src attribute beginning with /artist/ as:

```
var artistImages = $("img[src^='/artist/']");
```

Recall that you can select by attribute with square brackets ([attribute]), specify a value with an equals sign ([attribute=value]) and search for a particular value in the beginning, end, or anywhere inside a string with ^, $, and * symbols respectively ([attribute^=value], [attribute$=value], [attribute*=value]).

Pseudo-Element Selector

Pseudo-elements are special elements, which are special cases of regular ones. As you may recall from Chapter 3, these pseudo-element selectors allow you to append to any selector using the colon and one of :link, :visited, :focus, :hover, :active, :checked, :first-child, :first-line, and :first-letter.

These selectors can be used in combination with the selectors presented above, or alone. Selecting all links that have been visited, for example, would be specified with:

```
var visitedLinks = $("a:visited");
```

Since this chapter reviews and builds on CSS selectors, you are hopefully remembering some of the selectors you have used earlier and are making associations between those selectors and the ones in jQuery. As you already know from Chapter 6, once you have the ability to select an element, you can do many things to manipulate that element from changing its content or style all the way to removing it.

Contextual Selector

Another powerful CSS selector included in jQuery's selection mechanism is the contextual selectors introduced in Chapter 3. These selectors allowed you to specify

elements with certain relationships to one another in your CSS. These relationships included descendant (space), child (>), adjacent sibling (+), and general sibling (~).
To select all <p> elements inside of elements you would write
var para = $("div p");

6. A) With suitable PHP scripts, explain creating and reading cookies. [5 marks]

Cookies are a client-side approach for persisting state information. They are name=value pairs that are saved within one or more text files that are managed by the browser. These pairs accompany both server requests and responses within the HTTP header. While cookies cannot contain viruses, third-party tracking cookies have been a source of concern for privacy advocates.

Using Cookies
Like any other web development technology, PHP provides mechanisms for writing and reading cookies. Cookies in PHP are created using the setcookie() function and are retrieved using the $_COOKIES superglobal associative array. Below example illustrates the writing of a persistent cookie in PHP.

```
<?php
// add 1 day to the current time for expiry time
$expiryTime = time()+60*60*24;
// create a persistent cookie
$name = "Username";
$value = "Ricardo";
setcookie($name, $value, $expiryTime);
?>
```

The setcookie() function also supports several more parameters, which further customize the new cookie. You can examine the online official PHP documentation for more information. The below example illustrates the reading of cookie values. Notice that when we read a cookie, we must also check to ensure that the cookie exists. In PHP, if the cookie has expired (or never existed in the first place), then the client's browser would not send anything, and so the $_COOKIE array would be blank.

```
<?php
if( !isset($_COOKIE['Username']) ) {
//no valid cookie found
}
else {
echo "The username retrieved from the cookie is:";
echo $_COOKIE['Username'];
}
?>
```

6. B) Explain converting a JSON string to JSON object in JavaScript with suitable code segments. (04 Marks)

PHP comes with a JSON extension and as of version 5.2 of PHP, the JSON extension is bundled and compiled into PHP by default. Converting a JSON string into a PHP object is quite straightforward:

```
<?php
// convert JSON string into PHP object
$text = '{"artist": {"name":"Manet","nationality":"France"}}';
$anObject = json_decode($text);
echo $anObject->artist->nationality;
// convert JSON string into PHP associative array
$anArray = json_decode($text, true);
echo $anArray['artist']['nationality'];
?>
```

7. **With suitable script, explain loading and processing in XML document in Javascript. [10 Marks]**

XML Processing in JavaScript

All modern browsers have a built-in XML parser and their JavaScript implementations support an in-memory XML DOM API, which loads the entire document into memory where it is transformed into a hierarchical tree data structure. You can then use the already familiar DOM functions such as getElementById(), getElementsByTagName(), and createElement() to access and manipulate the data. For instance, Listing 17.5 shows the code necessary for loading an XML document into an XML DOM object, and it displays the id attributes of the <painting> elements as well as the content of each painting's <title> element.

```
<script>
if (window.XMLHttpRequest) {
// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
}
else {
// code for old versions of IE (optional you might just decide to
// ignore these)
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}

// load the external XML file
xmlhttp.open("GET","art.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;
// now extract a node list of all <painting> elements
paintings = xmlDoc.getElementsByTagName("painting");
if (paintings) {
// loop through each painting element
for (var i = 0; i < paintings.length; i++)
{
// display its id attribute
alert("id="+paintings[i].getAttribute("id"));
// find its <title> element
```

```
title = paintings[i].getElementsByTagName("title");
if (title) {
// display the text content of the <title> element
alert("title="+title[0].textContent);
}
}
}
</script>
```

3. Write a php program to create a class Employee with the following specifications:

   (08 Marks) Data members: Name, ID, Payment.

   Member function: Read(getters) and write (setters) Use the above specification to read and print the information of 10 students.

   Data members : Name, Roll number, Average marks

   Member function : Read(getters) and write (setters)

   Use the above specification to read and print the information of 2 students.

```
Class STUDENT{
Public $Name;
Public $Roll-number
Public $Average-marks
}
$student1=new STUDENT()
$student2=new STUDENT()
```

8. Write DTD for the following XML code.

```
<XML version=”1.0” encoding=”ISO-8859-1”?>
```

<art>

<painting id=”290”>

<title>Balcony</title>

<artist>

<name>Manet</name>

<nationality>France</nationality>

</artist>

<year>1868</year>

<medium>Oil on canvas</medium>

</painting>

</art>

Soln:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE art [
<!ELEMENT art (painting*)>
<!ELEMENT painting (title,artist,year,medium)>
<!ATTLIST painting id CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT artist (name,nationality)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT medium (#PCDATA)>
]>
<art>
. . .
</art>
```