5.2 WEB TECHNOLOGIES

L	P
3	4

RATIONALE

A computer engineering diploma student should have good exposure of various web technologies. This course will develop competency amongst the students to design professional database backed dynamic and feature based web sites. The course covers the use of programming with PHP and the concepts of database using MySQL.

COURSE OUTCOMES

After undergoing the subject, the students will be able to:

CO1: Develop different portal using HTML.
CO2: Perform various logical operations in PHP.
CO3: Create database using MySQL.
CO4: Install and configure Joomla.
CO4: Perform database connectivity using PHP.

DETAILED CONTENTS

UNIT I
DEVELOPING PORTALS USING HTML

Introduction to HTML 5 and CSS 3. Basic structure of HTML, designing a web page, inserting links images, horizontal rules, comments. Formatting text, title, headings, colors, fonts, sizes, simple tables and forms. HTML tags, hyperlinks. Adding graphics and images, image maps, image files. Using tables, forms, style sheets and frames. Floating of web site/pages.

UNIT II
PHP

Introduction to PHP: How PHP Works , The php.ini File, Basic PHP Syntax, PHP variables, statements, operators, decision making, loops, arrays, strings, forms, get and post methods, functions.

Introduction to cookies, storage of cookies at client side, Using information of cookies. Creating

single or multiple server side sessions. Timeout in sessions, Event management in PHP. Introduction to content management systems based on PHP.

UNIT III

MySQL

Introduction to MySQL, connecting to MySQL, database, creation, insertion, deletion and retrieval of MySQL data using PHP.

UNIT IV

JOOMLA BASICS AND ADMIN

Installing Wamp Server -Installing Joomla on Web Server, Joomla global configuration -Article manager -Archive manager-FrontPage manager -Section manager - Category manager- Media Manager-Menu manager -Component manager -Content Manager-Extensions manager-Module manager-Plugin manager-Template manager-How to install a new module-How to install a new template-How to install a new plugin-How to install a new component-Understanding the concept of Joomla positions -Changing the layout structure by changing the module position.

UNIT V

JOOMLA FRONTEND

Understanding Basic Joomla Template-Customizing Joomla template-Building Custom Joomla Template-Understanding Templatedetails.xml File-Creating Templatedetails.xml File using tmpl_builder Linking CSS-Linking JavaScript-Understanding Include-Displaying Content in Xhtml-Creating Template installation Package-Creating Custom Forms-Changing the Form Appearance using CSS.

PRACTICAL EXERCISES

1. Design PHP based web pages using correct PHP, CSS, and XHTML syntax, structure.
2. Create Web forms and pages that properly use HTTP GET and POST protocol as appropriate.
3 Design SQL language within MySQL and PHP to access and manipulate databases.
4. Install and configure both PHP and MySQL.
5 Create PHP code that utilizes the commonly used API library functions built in to PHP.
6. Design and create a complete web site that demonstrates good PHP/MySQL client/server design.

7. To store a cookie using PHP on client side.
8. To save the user session on server side.
9. Design website using Joomla.

RECOMMENDED BOOKS

1. Sams Teach Yourself PHP, MySQL, and Apache All in One" by Julie C.
2. Meloni, Publisher: SAMS ,ISBN 0-672-32976-X.
3. Web enabled development application by Ivan Byross: Commercial; TMH.
4. HTML, CSS, JavaScript, Perl, Python and PHP by Schafer Textbooks; Wiley India.
5. E-books/e-tools/relevant software to be used as recommended by AICTE/HSBTE/NITTTR.

RECOMMENDED WEBSITES

1. http://swayam.gov.in

INSTRUCTIONAL STRATEGY

This is hands on practice based subject and topics taught in the class should be practiced in the Lab regularly for development of required skills among the students. This subject contains five units of equal weightage.

STUDY NOTES | EXAM PREPARATION

Web Technologies

Diploma in Computer Engineering — NSQF Level 5

Scheme: L–3, P–4 (per week)

Units Covered: 5 Units, Equal Weightage

Focus: HTML5/CSS3, PHP, MySQL, Joomla CMS

Table of Contents

How to use these elaborated notes for exam preparation:

- Each unit carries roughly equal weightage (~20% of theory marks) — do not skip any unit.
- **Key Points** boxes (green) are quick-revision facts likely to appear as 2-mark questions.
- **Likely Exam Questions** boxes (red) list question patterns commonly asked from each unit — practice these first.
- Code snippets are elaborated for conceptual clarity but simplified for quick recall — practice running them in the lab for the practical exam.
- New in this edition: extended explanations, additional worked examples, comparison tables, and a deeper look at the 'why' behind each concept, not just the 'what'.
- Unit summaries (dark colour boxes at the end of each unit) are ideal for last-minute revision the night before the exam.

UNIT I

Developing Portals using HTML

Approx. weightage: 1/5 of theory marks | Foundation for Practicals 1-2

1.1 Introduction to HTML5 and CSS3

HTML (HyperText Markup Language) is the standard markup language used to create the structure and content of web pages, while CSS (Cascading Style Sheets) is a separate styling language used to control how that structure is presented -- its layout, colours, fonts, spacing and visual effects. It helps to think of HTML as the skeleton of a web page and CSS as the skin, clothing, and styling applied on top of that skeleton. A browser reads the HTML file first, builds an internal tree-like representation of the page called the DOM (Document Object Model), and then applies whatever CSS rules are linked to or embedded in the page before finally rendering the visible result on screen.

HTML5 is the fifth and current major revision of the HTML standard. Compared to HTML4/XHTML, it introduced a large number of **semantic tags** -- elements whose names describe the meaning of the content they wrap rather than just its appearance. Examples include **<header>** (top banner/navigation area), **<footer>** (bottom section with copyright/contact info), **<article>** (a self-contained piece of content such as a blog post), **<section>** (a thematic grouping of content) and **<nav>** (a block of navigation links). Before HTML5, developers had to fake these regions using generic div tags with class names like class='header', which carried no real meaning for browsers, search engines, or screen readers. HTML5 also added native support for embedding audio and video using the audio and video tags, removing the earlier dependence on third-party plugins such as Flash.

CSS3, the current generation of CSS, is not a single monolithic specification but a collection of independent modules, which is why browsers could adopt different CSS3 features at different speeds. It added rounded corners (border-radius), drop shadows (box-shadow, text-shadow), colour gradients (linear-gradient, radial-gradient), smooth transitions between property values (transition), keyframe-based animations (@keyframes / animation), and two powerful modern layout systems -- Flexbox (flex-container / flex-item model for one-dimensional layouts) and CSS Grid (for two-dimensional row-and-column layouts). Together these features drastically reduced the need for images or JavaScript to achieve visual effects that once required extra assets.

■ Key Point (very frequently asked, 2 marks)

- HTML defines the structure and content of a page; CSS defines its presentation and visual styling.
- Separating structure (HTML) from presentation (CSS) -- especially using external stylesheets -- improves maintainability, because one CSS file can restyle hundreds of HTML pages at once, and content authors do not need to touch design code.
- HTML5 semantic tags improve accessibility (screen readers understand the page better) and SEO (search engines can identify the important regions of a page).

1.2 Basic Structure of an HTML Page

Every valid HTML5 document follows the same basic skeleton. Understanding what each part does -- rather than just memorising the tags -- makes it much easier to answer 'explain the structure of an HTML document' style questions.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Heading</h1>
    <p>Paragraph text.</p>
  </body>
</html>
```

Tag	Purpose / Explanation
<!DOCTYPE html>	Tells the browser this document follows the HTML5 standard, so it should use standards (not legacy 'quirks') mode when rendering the page. It is not an HTML tag itself, just a declaration, and it must be the very first line of the file.
<html>	The root element that wraps every other element on the page. All content, both head and body, lives inside this single container.
<head>	Holds meta information about the page that is not directly displayed to the user -- the page title, character encoding, links to CSS files, meta tags for SEO, and script references.
<body>	Contains everything that is actually visible in the browser window -- headings, paragraphs, images, tables, forms, and all other visual content.

A common exam trap is confusing the head section with the header tag -- they are completely different. head is a mandatory structural container for metadata and is never displayed; header is a semantic HTML5 tag that lives inside body and is rendered visibly as the top banner of the page (often containing a logo and navigation menu).

1.3 Designing a Web Page - Links, Images, Rules, Comments

Hyperlinks are what make the 'web' a web -- they connect individual pages into a network of documents. The anchor tag `<a href="page2.html">Next</a>` creates a clickable link; the href attribute specifies the destination, which can be another local page, an external website (https://...), an email address (mailto:), or even a location within the same page using an id reference (#section2).

Images are embedded using the self-closing img tag: `<img src="logo.png" alt="Logo">`. The src attribute points to the image file, while the alt attribute provides a text description that is displayed if the image fails to load and is read aloud by screen readers for visually impaired users -- this dual role (accessibility + fallback text + SEO benefit) is one of the most commonly tested points in this section.

The **horizontal rule** tag hr draws a thin dividing line across the page, traditionally used to visually separate sections of content. **Comments**, written as `<!-- comment text -->`, are completely ignored by the browser and never rendered; they exist purely so developers can leave notes, explanations, or temporarily disable a block of code without deleting it.

1.4 Formatting Text - Title, Headings, Colors, Fonts, Sizes

The title tag, placed inside head, does not appear inside the page body at all -- instead it is shown on the browser tab and is used as the default heading when a page is bookmarked or found through a search engine. Headings range from h1 (the largest and most important, ideally used only once per page as the main title) down to h6 (the smallest), decreasing in both size and semantic importance --

this hierarchy also matters for SEO, since search engines give more weight to text inside higher-level headings.

Basic text formatting can be applied using b or strong for bold text, i or em for italics, and u for underline. It is worth noting the subtle distinction examiners sometimes probe: b and i are purely presentational (just visual styling), whereas strong and em carry semantic meaning (importance / emphasis) in addition to their default bold/italic appearance.

Actual visual styling -- colours, fonts and sizes -- is best handled through CSS properties rather than old-style HTML attributes:

```
h1 { color: navy; font-family: Arial; font-size: 28px; font-weight: bold; }
```

Here, `color` sets the text colour, `font-family` chooses the typeface (with fallback fonts usually listed after a comma in real projects), `font-size` controls how large the text renders (commonly in px, em, or rem units), and `font-weight` controls boldness on a numeric or keyword scale.

1.5 Tables and Forms

Simple Table

HTML tables are built from three core tags nested inside table: tr defines a table row, th defines a header cell (bold and centered by default), and td defines an ordinary data cell. The border attribute (or, in modern practice, a CSS border rule) draws the grid lines.

```
<table border="1">
  <tr><th>Name</th><th>Marks</th></tr>
  <tr><td>Aman</td><td>85</td></tr>
  <tr><td>Riya</td><td>92</td></tr>
</table>
```

Two attributes are frequently tested because they let a single cell occupy more than one row/column of space: **colspan** merges a cell horizontally across multiple columns (e.g. a title row spanning the whole table width), and **rowspan** merges a cell vertically across multiple rows (e.g. one label shared by several data rows below it). Examiners often ask students to design a timetable or mark-sheet using these two attributes together.

Simple Form

Forms are the primary mechanism through which a web page collects input from a user and sends it to a server for processing (very often, that processing is done by a PHP script, which is exactly why forms bridge Unit I and Unit II).

```
<form action="submit.php" method="post">
  Name: <input type="text" name="uname"><br>
  Password: <input type="password" name="pwd"><br>
  <input type="radio" name="gender" value="M">Male
  <input type="radio" name="gender" value="F">Female<br>
  <input type="checkbox" name="subscribe">Subscribe<br>
  <input type="submit" value="Send">
</form>
```

Key form elements include input (whose 'type' attribute changes its behaviour completely -- text, password, radio, checkbox, submit, email, date, etc.), textarea for multi-line free text, and the paired select/option tags for dropdown menus. The **action** attribute specifies which server-side script will process the submitted data, and the **method** attribute (GET or POST) determines exactly how that data travels to the server -- a distinction explored fully in Unit II.

1.6 Style Sheets and Frames

Type of CSS	How it is written	When it is best used
Inline CSS	style="color:red;" written directly inside a tag	Quick one-off tweaks to a single element; not reusable and considered poor practice for large sites
Internal CSS	A style block placed inside head	Styling rules specific to one page only
External CSS	link rel="stylesheet" href="style.css"	Best practice -- one file can style an entire multi-page website, keeping design centralised and easy to update

Frames historically allowed a browser window to be split into multiple independently-scrolling HTML documents using the frameset and frame tags, but this approach is now obsolete and is not supported in HTML5. The modern equivalent is the iframe (inline frame) tag -- `<iframe src="page.html"></iframe>` -- which embeds one complete HTML document inside a rectangular region of another page, commonly used today to embed maps, videos, or advertisements from another source.

1.7 Image Maps and Floating of Web Pages

An **image map** allows a single image to behave like several different hyperlinks depending on which part of the image the user clicks -- useful for things like a clickable geographic map or a diagram with labelled hotspots. It is defined using a map element containing one or more area tags, each specifying a shape (rect, circle, or poly), a set of pixel coordinates, and a destination href.

```
<img src="campus.png" usemap="#campusmap" alt="Campus Map">
<map name="campusmap">
  <area shape="rect" coords="10,10,150,90" href="library.html" alt="Library">
  <area shape="circle" coords="200,60,40" href="canteen.html" alt="Canteen">
</map>
```

Floating is a CSS positioning technique -- `float: left;` or `float: right;` -- that removes an element from the normal top-to-bottom document flow and pushes it to one side, allowing surrounding text or other boxes to wrap around it. This is how early multi-column magazine-style layouts (an image floated left with text wrapping around its right side) were built before Flexbox and Grid became standard, and it remains foundational knowledge for understanding how modern responsive layouts evolved.

✍ Likely Exam Questions - Unit I

1. Differentiate between HTML and CSS. (2 marks)
2. Explain any five new features of HTML5. (4 marks)
3. Write HTML code to design a registration form with text, radio button, and submit button. (4/6 marks)
4. Explain the three ways of applying CSS to an HTML page with examples. (4 marks)
5. What is an image map? Explain with syntax. (2/4 marks)
6. Design a web page using tables to display a timetable, using colspan and rowspan. (6 marks)
7. Differentiate between <b> and , and between <i> and . (2 marks)
8. Explain the difference between iframe and the older frameset approach. (2/4 marks)

Unit I in one glance: HTML5/CSS3 basics -> page structure (DOCTYPE, html, head, body) -> links/images/comments -> text formatting (headings, bold/italic/underline) -> tables (colspan, rowspan) and forms (GET/POST) -> three CSS types (inline, internal, external) -> frames/iframe -> image maps and floating layouts.

UNIT II

PHP

Approx. weightage: 1/5 of theory marks | Core scripting unit - highest practical relevance

2.1 Introduction to PHP - How PHP Works

PHP (originally 'Personal Home Page Tools', now recursively 'PHP: Hypertext Preprocessor') is a server-side scripting language, meaning its code is executed on the web server rather than inside the user's browser. This is the single most important conceptual distinction to understand for this unit: JavaScript runs on the client (browser) and can be viewed by anyone using 'View Source', but PHP runs before the page is even sent out, so the visitor's browser only ever receives plain HTML -- the original PHP source code, database credentials, and business logic remain completely hidden from the client.

The typical request-response cycle proceeds as follows: the user's browser sends an HTTP request for a .php file; the web server (such as Apache within a WAMP/XAMPP stack) recognises the .php extension and hands the file over to the PHP engine/interpreter instead of serving it directly; the PHP engine executes any code between <?php ... ?> tags line by line, optionally connecting to a database such as MySQL to fetch or store data along the way; finally, the engine produces a plain HTML output stream, which the web server sends back to the browser exactly as if it were a static HTML file.

■ Diagram to remember (very common diagram-based question)

• Browser sends a request → Web Server receives it → PHP Engine executes the .php file (accessing a database if the script needs one) → Engine produces pure HTML output → Web Server sends that HTML back → Browser renders the final page for the user.

• The browser never sees the PHP code itself -- only the final HTML result of running that code.

2.2 The php.ini File

php.ini is the central configuration file read by the PHP engine every time it starts up. It governs a very wide range of behaviour: whether and how errors/warnings are displayed (display_errors), the maximum size of files that can be uploaded through a form (upload_max_filesize, post_max_size), how long session data should be kept alive (session.gc_maxlifetime), the default timezone, memory limits per script (memory_limit), and which optional extensions (such as the MySQLi or GD image library) are enabled. Because these settings are only read once when the PHP engine (or, in many setups, the entire web server) starts, any change made to php.ini requires a restart of the web server (e.g. restarting Apache in WAMP) before it takes effect -- simply refreshing the browser is not enough.

2.3 Basic PHP Syntax and Variables

```
<?php
  $name = "World";          // variable, starts with $
  $age  = 21;               // integer
  $gpa  = 8.75;             // float
  echo "Hello, $name!";     // output statement
  echo "Age: " . $age;      // concatenation using the dot operator
?>
```

- PHP code is always enclosed inside <?php ... ?> tags, which can be embedded anywhere within an otherwise ordinary HTML file -- this mixing of PHP and HTML in the same file is one of PHP's defining characteristics.
- Every variable name must start with a dollar sign ($) and PHP variable names are case-sensitive, so $Name and $name are treated as two entirely different variables.
- PHP is a loosely (dynamically) typed language -- there is no need to declare a variable's data type in advance; the same variable can hold a string at one point and a number later on.
- Every executable statement must end with a semicolon (;); forgetting this is one of the most common beginner syntax errors.
- String concatenation uses the dot (.) operator, not the plus (+) sign used by many other languages.

2.4 Operators, Decision Making, Loops, Arrays, Strings

Category	Examples / Notes
Arithmetic	+ - * / % (modulus/remainder) ** (exponent, e.g. 2**3 = 8)
Comparison	== (equal value) != (not equal) === (equal value AND type) <, >, <=, >=
Logical	&& (AND) || (OR) ! (NOT)
Decision making	if, if-else, elseif ladder, switch-case (useful for many discrete options)
Loops	for (fixed number of repetitions), while (condition checked first), do-while (runs at least once), foreach (iterates directly over array elements)
Arrays	Indexed arrays (numeric keys 0,1,2..), Associative arrays (custom string keys), Multidimensional arrays (an array of arrays, e.g. a table of student records)

```
$marks = array("Amit"=>85, "Riya"=>92, "Sara"=>78);
foreach ($marks as $name => $score) {
    echo "$name scored $score<br>";
}
// Simple decision + loop example
for ($i = 1; $i <= 5; $i++) {
    if ($i % 2 == 0) {
        echo "$i is even<br>";
    } else {
        echo "$i is odd<br>";
    }
}
```

■ Key Point (very frequently asked distinction)

• == compares only the value of two operands after converting types if necessary (loose comparison), so "5" == 5 evaluates to true.

• === compares both the value AND the data type without any conversion (strict comparison), so "5" === 5 evaluates to false because one is a string and the other is an integer.

• This distinction becomes especially important when validating form data received as strings from $_POST or $_GET.

PHP also provides a rich set of built-in **string functions** that are commonly asked to be identified or applied in short-answer questions, such as strlen() (returns the length of a string), str_replace()

(replaces occurrences of a substring), strtoupper()/strtolower() (case conversion), substr() (extracts part of a string), and trim() (removes whitespace from both ends).

2.5 Forms - GET and POST Methods

When an HTML form is submitted, the browser packages the field values and sends them to the server using whichever HTTP method the form's 'method' attribute specifies. PHP exposes the received values through two special superglobal arrays: $_GET and $_POST.

Aspect	GET	POST
How data travels	Appended visibly to the URL as a query string (e.g. ?uname=Amit)	Sent inside the body of the HTTP request, hidden from the address bar
Data length	Limited (URL length restrictions apply)	Much larger amounts of data are allowed
Bookmarking / caching	Can be bookmarked and is often cached by browsers	Cannot be bookmarked meaningfully; generally not cached
Security for sensitive data	Less suitable -- data is visible in the URL and browser history	More suitable -- values are not shown in the URL, though still not encrypted by itself
Accessed in PHP via	$_GET['field']	$_POST['field']

```
<?php
  $uname = $_POST['uname'];
  echo "Welcome, " . $uname;
?>
```

A well-written answer should also mention that POST is generally preferred for anything that changes data on the server (login credentials, uploaded files, form submissions that insert database records), while GET is more appropriate for simple, repeatable, shareable requests such as a search query, since a GET URL can be bookmarked or shared and re-run safely.

2.6 Functions

```
function add($a, $b) {
    return $a + $b;
}
echo add(5, 10);   // 15

function greet($name = "Guest") {   // default parameter value
    return "Hello, $name!";
}
echo greet();          // Hello, Guest!
echo greet("Priya");   // Hello, Priya!
```

User-defined functions promote code reuse and organisation -- instead of repeating the same block of logic multiple times throughout a script, it is written once inside a function and simply called wherever needed. PHP also ships with a very large library of built-in functions covering strings (strlen(), str_replace()), arrays (count(), array_sum(), sort(), array_merge()), mathematics (round(), rand(), abs()), and dates (date(), time()), which examiners frequently ask students to identify the purpose of, or use directly in a short program.

2.7 Cookies

HTTP is inherently **stateless** -- by default, a web server treats every incoming request as completely independent, with no memory of any previous request from the same visitor. Cookies exist to solve this problem: a cookie is a small piece of data (typically under 4KB) that a server asks the browser to store, and the browser then automatically re-sends that same data with every subsequent request to the same site, allowing the server to 'remember' information such as a logged-in username, site preferences, or the contents of a shopping cart across multiple page loads.

```
<?php
  setcookie("username", "Amit", time() + 3600); // valid for 1 hour from now
?>
<?php
  echo $_COOKIE['username']; // read the cookie on a later page/request
?>
```

■ Key Point

• Cookies are stored entirely on the client side (inside the user's own browser), not on the server.
• setcookie() must be called before any HTML output (even a single blank line) is sent to the browser, because cookies are transmitted as part of the HTTP response header, and headers must be sent before the body content.
• Cookies can be set to expire after a fixed time, or to persist even after the browser is closed, depending on the expiry time passed to setcookie().

2.8 Sessions

A **session** is the server-side counterpart to a cookie. Instead of storing the actual data in the user's browser, the server stores the data itself (in a temporary file or database on the server), and gives the browser only a small, randomly generated **session ID** — usually delivered to the browser as a cookie behind the scenes. On every subsequent request, the browser sends that session ID back, and the server uses it to look up the corresponding stored session data. Because the actual sensitive data never leaves the server, sessions are considered more secure than cookies for things like login state or shopping-cart totals.

```
<?php
  session_start();                  // must be called before any output, on every page
 that uses sessions
  $_SESSION['user'] = "Amit";       // create/store a session variable
?>
<?php
  session_start();
  echo $_SESSION['user'];           // read it back on a different page
?>
<?php
  session_start();
  session_destroy();                // end the session and clear all session data
?>
```

A **single session** simply refers to one continuous browsing episode for one specific user, tracked by one unique session ID. **Multiple (simultaneous) sessions** occur naturally whenever more than one user is logged into the same website at the same time -- the server maintains a separate session ID and a separate block of stored data for each individual user, so one user's session data never gets mixed up with another's. **Session timeout** refers to automatically ending/invalidating a session after a period of inactivity; this can be configured globally through the session.gc_maxlifetime setting in

php.ini, or implemented manually inside a script by storing a 'last activity' timestamp in the session and checking, on each page load, whether too much time has elapsed since that timestamp.

Aspect	Cookie	Session
Storage location	Client side (browser)	Server side
Relative security	Less secure -- can be viewed/edited by the user	More secure -- actual data never leaves the server
Persistence	Can persist even after the browser is closed, if an expiry time is set	Usually ends when the browser is closed, unless the session ID cookie itself is made persistent
Typical storage size	Small, roughly 4KB per cookie	Can hold much larger and more complex data

2.9 Event Management in PHP

Unlike JavaScript, PHP is not natively event-driven in the browser sense -- it has no direct concept of a 'click' or 'mouseover' happening in real time, because PHP only runs on the server, once, in response to a request that has already been sent. In this course, 'event management in PHP' therefore refers to detecting and responding to user-triggered actions indirectly, through the HTTP request they generate -- for example, submitting a form is the 'event', and PHP detects that this event has happened using a check such as `if (isset($_POST['submit'])) { ... }`, which tests whether a field named 'submit' (typically the submit button itself) was actually present in the incoming POST data.

2.10 Introduction to CMS based on PHP

A **Content Management System (CMS)** is application software that allows non-technical users to create, edit, organise, and publish website content (text, images, pages) through a friendly point-and-click administrative interface, without needing to write or understand any HTML, CSS, or PHP code themselves. Behind the scenes, a CMS is itself typically built using PHP and a database such as MySQL, and it dynamically generates the final HTML pages that visitors see by pulling stored content out of the database at request time. The three most widely used PHP-based CMS platforms are Joomla, WordPress, and Drupal. This concept is the direct bridge into Units IV and V of this course, which study Joomla specifically in depth.

Likely Exam Questions - Unit II

1. Explain how PHP works with a diagram (browser to server to PHP engine to browser). (4 marks)
2. Differentiate between GET and POST methods. (4 marks)
3. Differentiate between cookies and sessions. (4 marks)
4. Write a PHP program to accept two numbers and display their sum using a function. (4 marks)
5. Write a PHP script to create and read a cookie. (4/6 marks)
6. What is session timeout? How is it handled in PHP? (2/4 marks)
7. Explain decision-making and looping statements in PHP with examples. (6 marks)
8. Differentiate between == and === operators in PHP with an example. (2 marks)
9. What is the purpose of the php.ini file? Give any four settings it controls. (4 marks)

Unit II in one glance: PHP execution flow (browser to server to PHP engine to HTML) -> php.ini configuration -> syntax/variables (loosely typed, $ prefix) -> operators/loops/arrays -> GET vs POST forms -> user-defined and built-in functions -> cookies (client-side) vs sessions (server-side) -> session timeout -> PHP-based CMS introduction (Joomla, WordPress, Drupal).

U N I T III

MySQL

Approx. weightage: 1/5 of theory marks | Pairs directly with PHP (Unit II) for CRUD practicals

3.1 Introduction to MySQL

MySQL is an open-source **Relational Database Management System (RDBMS)** -- software that stores structured data inside tables made up of rows and columns, and manages that data using SQL (Structured Query Language). 'Relational' means that separate tables can be linked to one another through shared key values (for example, an 'orders' table can reference a 'customer_id' that exists in a separate 'customers' table), which avoids repeating the same information many times over and keeps the data organised and consistent. MySQL is the database most commonly paired with PHP in real projects and in this syllabus -- it is the 'M' in the classic LAMP (Linux, Apache, MySQL, PHP) or WAMP (Windows, Apache, MySQL, PHP) software stack used to build and host dynamic, database-driven websites.

3.2 Connecting to MySQL from PHP

```
<?php
  $conn = mysqli_connect("localhost", "root", "", "college_db");
  if (!$conn) {
      die("Connection failed: " . mysqli_connect_error());
  }
  echo "Connected successfully";
?>
```

The built-in function mysqli_connect(host, username, password, database) takes four parameters in order: the address of the database server (usually 'localhost' during development), the MySQL username, the corresponding password (often blank by default on a fresh WAMP installation), and the name of the specific database to use. The function returns a connection resource/object on success, or false on failure. Always checking the return value with an if (!$conn) style test -- and calling die() with a descriptive error message if it failed -- is considered good practice and is very frequently the exact code examiners ask students to write when a question says 'write PHP code to connect to a MySQL database'.

3.3 Database and Table Creation

```
CREATE DATABASE college_db;
USE college_db;

CREATE TABLE student (
    id     INT PRIMARY KEY AUTO_INCREMENT,
    name   VARCHAR(50),
    marks  INT
);
```

- CREATE DATABASE makes a brand-new, empty database container on the server.
- USE selects which database the following SQL statements should operate on.
- CREATE TABLE defines a new table's structure -- its column names and their data types.

- PRIMARY KEY marks a column (commonly an id) as the unique identifier for every row in that table -- no two rows may share the same primary key value, and it cannot be left empty (NULL).
- AUTO_INCREMENT tells MySQL to automatically generate the next whole number for that column every time a new row is inserted, so the developer never has to manually track or supply an id value.
- VARCHAR(50) stores variable-length text up to 50 characters, while INT stores whole numbers -- choosing an appropriately sized data type for each column is itself sometimes a small exam point.

3.4 Insertion, Deletion, and Retrieval of Data

Operation	SQL Syntax
Insert	INSERT INTO student (name, marks) VALUES ('Amit', 85);
Retrieve (all rows)	SELECT * FROM student;
Retrieve (with a condition)	SELECT * FROM student WHERE marks > 80;
Update	UPDATE student SET marks = 90 WHERE id = 1;
Delete	DELETE FROM student WHERE id = 1;

The WHERE clause is what makes UPDATE and DELETE statements safe and precise -- it restricts the operation to only the rows matching a given condition. Omitting the WHERE clause on an UPDATE or DELETE statement is a classic and dangerous mistake, since it would apply the change or deletion to every single row in the table; examiners sometimes specifically ask students to explain why the WHERE clause is important for exactly this reason.

3.5 Performing These Operations Using PHP

```
<?php
  $conn = mysqli_connect("localhost", "root", "", "college_db");
  $sql  = "INSERT INTO student (name, marks) VALUES ('Riya', 92)";
  if (mysqli_query($conn, $sql)) {
      echo "Record inserted successfully";
  } else {
      echo "Error: " . mysqli_error($conn);
  }
?>

<?php
  $result = mysqli_query($conn, "SELECT * FROM student");
  while ($row = mysqli_fetch_assoc($result)) {
      echo $row['name'] . " - " . $row['marks'] . "<br>";
  }
?>
```

mysqli_query($conn, $sql) sends any SQL statement -- INSERT, SELECT, UPDATE, or DELETE -- to the database through the given connection and returns either true/false (for INSERT/UPDATE/DELETE) or a special 'result set' object (for SELECT). mysqli_fetch_assoc($result) is then used inside a while loop to pull each row of that result set out, one at a time, as an associative array whose keys match the table's column names -- allowing values to be accessed conveniently as $row['name'] rather than by numeric position.

■ Key Point

• mysqli_query() executes SQL statements sent from PHP to MySQL.
• mysqli_fetch_assoc() retrieves each row of a SELECT result as an associative array, keyed by column name -- this pair of functions forms the backbone of nearly every database-driven practical exercise in this course.
• A closely related function, mysqli_fetch_array(), returns each row as both an associative array (by column name) and a numeric array (by column position) combined -- a common point of differentiation asked in exams.

3.6 Closing the Connection

```
mysqli_close($conn);
```

Once all required database operations for a script have been completed, it is good practice to release the connection resource explicitly using mysqli_close($conn). While PHP will usually close any open connections automatically once a script finishes executing, explicitly closing the connection frees up server resources sooner, which matters more on busy production servers handling many simultaneous requests.

Putting all of the above together, a typical exam-style program is expected to follow this sequence: connect to the database, check the connection succeeded, build and run the required SQL statement through mysqli_query(), process or display any returned rows using mysqli_fetch_assoc(), and finally close the connection with mysqli_close().

✍ Likely Exam Questions - Unit III

1. Write PHP code to connect to a MySQL database. (2/4 marks)
2. Write SQL and PHP code to insert a record into a table and display a success message. (6 marks)
3. Explain the steps to retrieve and display all records from a MySQL table using PHP. (6 marks)
4. Differentiate between mysqli_fetch_assoc() and mysqli_fetch_array(). (2 marks)
5. Write a PHP program to update and delete a student record using MySQL. (6 marks)
6. Explain the significance of the PRIMARY KEY and AUTO_INCREMENT attributes while creating a table. (2/4 marks)
7. Why is the WHERE clause important in UPDATE and DELETE statements? (2 marks)

Unit III in one glance: MySQL basics (RDBMS, LAMP/WAMP) -> connect via mysqli_connect() -> create database/table (PRIMARY KEY, AUTO_INCREMENT) -> INSERT/SELECT/UPDATE/DELETE in plain SQL -> the same CRUD operations executed through PHP using mysqli_query() and mysqli_fetch_assoc() -> close the connection with mysqli_close().

UNIT IV

Joomla Basics and Admin

Approx. weightage: 1/5 of theory marks | Focus on terminology and manager functions

4.1 Installing WAMP Server

WAMP is an acronym for **Windows, Apache, MySQL, PHP** -- a bundled software package that installs all four components together on a Windows machine in one go, turning an ordinary desktop or laptop computer into a fully working local web server. This local server environment is necessary before installing and testing PHP/MySQL-based content management systems such as Joomla, because Joomla is not a standalone program -- it is a large collection of PHP scripts that must be executed by Apache and that store all of their content inside a MySQL database. After installing WAMP, its control panel is used to start/stop the Apache and MySQL services and to confirm both are running correctly before proceeding to install Joomla itself.

4.2 Installing Joomla on the Web Server

1. Download the latest Joomla installation package (a compressed .zip archive) from the official Joomla website.
2. Extract the contents of that archive into WAMP's designated web root folder, commonly named 'www', usually inside a project-specific subfolder.
3. Create a new, empty MySQL database (and, if required, a dedicated database user) for Joomla to use, typically through phpMyAdmin.
4. Open the new site's address in a browser to launch Joomla's own web-based installer, which asks for the site name, an administrator account (username/password/email), and the database connection details (host, database name, username, password) created in the previous step.
5. Once the installer finishes successfully, delete (or rename) the installation folder as instructed on the final screen -- leaving it in place is a security risk, since it could allow the site to be reinstalled or reconfigured by anyone who finds the URL.

This five-step outline -- download, extract, database, web installer, remove installation folder -- is exactly the sequence examiners expect when a question asks students to 'explain the steps to install Joomla on a WAMP server'.

4.3 Joomla Global Configuration

Once Joomla is installed, the **Global Configuration** screen inside the admin back-end is the central place where site-wide settings are controlled. This includes the overall site name (shown in the browser tab and used in page titles), default metadata and SEO-related options (such as whether to use search-engine-friendly URLs), the default content editor used when writing articles, session length (how long an administrator or registered user stays logged in before being logged out automatically), and the database/server connection details originally entered during installation. Because these settings apply across the entire site rather than to any single page or article, Global Configuration is usually one of the first screens a Joomla administrator becomes familiar with.

4.4 Joomla Managers - Definitions Table

Joomla organises almost all of its administrative functionality into a set of distinct 'Managers', each responsible for one type of content or extension. Learning what each manager is responsible for is the single most heavily tested topic in this unit, since most Unit IV questions are direct definition or comparison questions built around this table.

Manager	Function
Article Manager	Create, edit, publish, and organise articles -- the main content units of a Joomla site (similar to individual blog posts or pages).
Archive Manager	Stores older articles that are no longer current but are kept for future reference rather than being permanently deleted.
FrontPage Manager	Controls which specific articles are displayed on the site's home page, and in what order.
Section Manager	Groups multiple categories together under one broad topic (used mainly in older versions of Joomla; largely superseded in newer versions by nested categories).
Category Manager	Organises individual articles into categories within a section, giving the site's content a clear, browsable hierarchy.
Media Manager	Uploads and manages image and other media files that are used across articles and the site as a whole.
Menu Manager	Creates and manages the navigation menus that visitors use to move between different pages/sections of the site.
Component Manager	Manages larger, self-contained functional add-ons, such as contact forms, banner advertising systems, or an online store.
Content Manager	Provides overall/umbrella management of a site's content-related elements.
Extensions Manager	Installs and uninstalls all add-ons -- components, modules, plugins, and templates -- from a single central screen.
Module Manager	Manages modules -- small, self-contained content blocks placed into specific template positions, such as a login box or a 'latest news' panel.
Plugin Manager	Manages plugins -- small pieces of code that run automatically in response to specific system events, such as a search plugin that runs whenever a visitor performs a search.
Template Manager	Manages the templates that control the overall visual design and page layout of the site.

■ Key Point (frequently tested hierarchy)

• Extensions is the umbrella/parent category -- Components, Modules, Plugins, and Templates are all, technically, different types of 'extensions', and all four are ultimately installed and managed through the Extensions Manager, even though each also has its own dedicated manager screen for day-to-day use.
• A useful memory aid: Components = big features, Modules = small display boxes, Plugins = event-triggered background code, Templates = overall design -- together these four make up 'Extensions'.

4.5 Installing New Extensions

Regardless of whether a new addition is a module, template, plugin, or component, Joomla installs all of them through exactly the same general workflow: open the Extensions Manager, choose 'Install', upload the extension's compressed .zip package (or point to a URL/folder containing it), and Joomla

automatically extracts the files, registers the extension in its database, and makes it available in the corresponding specialised manager (e.g. a newly installed module will now also appear inside the Module Manager, ready to be enabled and positioned).

4.6 Joomla Positions and Layout

A **position** is a named, predefined placeholder area built into a Joomla template's design -- common examples include 'left', 'right', 'top', 'footer', and 'breadcrumbs'. Modules are assigned to one specific position each, and changing which position a module is assigned to (through the Module Manager) immediately changes where that module physically appears on the rendered page -- for example, moving a 'Login' module from the 'left' position to the 'right' position moves the login box from the left side of the page to the right side. This system is what allows a site administrator to substantially rearrange a page's layout without writing or editing any code at all.

Likely Exam Questions - Unit IV

1. What is Joomla? List its key features as a CMS. (4 marks)
2. Explain the steps to install Joomla on a WAMP server. (6 marks)
3. Differentiate between Article Manager and Category Manager. (2 marks)
4. Explain any four Joomla managers with their functions. (4/6 marks)
5. What are Joomla positions? How are they used to change page layout? (4 marks)
6. Explain the difference between Components, Modules, and Plugins. (4 marks)
7. What settings can be configured through Joomla's Global Configuration screen? (4 marks)

Unit IV in one glance: WAMP setup -> Joomla installation (download, extract, database, web installer, remove install folder) -> Global Configuration -> 13 managers (Article, Archive, FrontPage, Section, Category, Media, Menu, Component, Content, Extensions, Module, Plugin, Template) -> installing new extensions via Extensions Manager -> positions and layout control.

UNIT V

Joomla Frontend

Approx. weightage: 1/5 of theory marks | Focus on template structure and files

5.1 Understanding the Basic Joomla Template

A Joomla **template** is the collection of files that controls exactly how a site looks -- its overall HTML/CSS structure, colour scheme, typography, and the physical arrangement of module positions on the page -- entirely independent of the actual content (articles, images, menus) stored in the site's database. This separation of design from content is the same core principle behind separating HTML from CSS in Unit I, just applied at the scale of an entire CMS: because a template only controls presentation, an administrator can switch a Joomla site to a completely different template, instantly changing its whole visual appearance, without losing, editing, or duplicating a single article.

5.2 Customizing and Building a Custom Joomla Template

There are two distinct levels of template work an administrator or developer might undertake. **Customising an existing template** is the lighter-weight approach -- typically limited to editing its CSS files to change colours, fonts, and spacing, or creating targeted 'template overrides' for specific layout files without touching the template's core logic. **Building a custom template from scratch** is a much larger undertaking that requires setting up a properly structured template folder containing all of the core files Joomla expects to find -- most importantly index.php (the main layout/logic file) and templateDetails.xml (the manifest file), along with any CSS, JavaScript, and image assets the design requires.

5.3 templateDetails.xml File

templateDetails.xml is a mandatory XML manifest file that every Joomla template must include. It acts as an 'identity card' for the template, telling Joomla the template's name, author, and version number, exactly which files belong to the template package, and which module positions the template makes available for use. Without a valid, correctly formatted templateDetails.xml file, Joomla will not recognise the folder as an installable template at all, and the installation process (through the Extensions Manager) will fail.

```
<extension type="template" client="site">
    <name>MyTemplate</name>
    <author>Student</author>
    <version>1.0.0</version>
    <files>
        <filename>index.php</filename>
        <filename>css/template.css</filename>
    </files>
    <positions>
        <position>left</position>
        <position>right</position>
    </positions>
</extension>
```

Breaking this down: the <extension> root tag declares that this manifest describes a 'template'-type extension meant for the public-facing 'site' (as opposed to the admin back-end). The <files> block lists

every file that must be copied as part of the template package, while the <positions> block declares the named placeholder areas (matching what was discussed in Unit IV, section 4.6) that this particular template design actually supports -- a template can only offer module positions that it explicitly lists here. In practice, writing this file by hand for a large template can be tedious and error-prone, so the tmpl_builder tool can be used to automatically generate a basic, valid templateDetails.xml file, which developers then refine further by hand.

5.4 Linking CSS and JavaScript

Inside a template's index.php file, CSS and JavaScript files are not linked using plain, hard-coded <link>/<script> tags as they might be in a simple static HTML page. Instead, Joomla provides its own **document API**, which lets the template register its stylesheets and scripts through PHP code, allowing Joomla to manage file paths correctly regardless of the site's folder structure or base URL, and to avoid loading duplicate files if multiple modules request the same script.

```
<?php
  $doc = JFactory::getDocument();
  $doc->addStyleSheet('templates/mytemplate/css/template.css');
  $doc->addScript('templates/mytemplate/js/script.js');
?>
```

JFactory::getDocument() retrieves an object representing the current page's output document; calling addStyleSheet() and addScript() on that object queues up the given CSS and JavaScript files so that Joomla automatically inserts the correct <link> and <script> tags into the final rendered page's <head> section.

5.5 Understanding 'Include' and Displaying Content in XHTML

PHP's built-in include() (or the closely related require()) statement is used extensively inside Joomla templates to pull a separate, reusable chunk of code or markup directly into index.php at the point where include() is called -- for example, a template might keep its header markup or a particular module-position block in its own separate file and simply include() it, rather than repeating the same code across multiple layout files. Regardless of how many separate files are stitched together this way, Joomla's final rendered output is expected to be valid, well-formed **XHTML-compliant markup** (properly nested, properly closed tags, lower-case tag names), which ensures the resulting page displays consistently and predictably across different browsers and devices.

5.6 Creating a Template Installation Package

To share or install a custom-built template on any Joomla site, every file that belongs to it -- index.php, templateDetails.xml, all CSS/JS files, and any image or font assets -- must first be gathered together and compressed into a single .zip archive, mirroring the same folder structure the template needs internally. That single .zip file is then installed exactly like any other extension: through Extensions Manager → Install in the Joomla admin panel, where Joomla extracts the archive, reads its templateDetails.xml manifest, and registers the new template so it becomes selectable inside the Template Manager.

■ Key Point (build order to remember for 'explain the steps' questions)

• Plan the intended layout → create the template's folder structure → write index.php → write templateDetails.xml → add the CSS/JS/image assets → compress everything into a single .zip file → install that .zip through the Extensions Manager.

5.7 Creating Custom Forms and Changing Form Appearance using CSS

Custom forms embedded within a Joomla site -- such as a contact form or a feedback form -- are built using the same standard HTML form elements covered in Unit I (<form>, <input>, <textarea>, <select>, and so on). Their visual appearance is then adjusted purely through CSS classes and rules -- changing input border styles, background colours, button colours, internal spacing (padding/margin), and font choices -- without ever needing to alter the form's underlying HTML structure or its functional behaviour. This again reinforces the structure-versus-presentation separation that runs throughout this entire syllabus, from plain HTML pages in Unit I all the way through to full Joomla templates here in Unit V.

✍ Likely Exam Questions - Unit V

1. What is a Joomla template? Explain its role in separating design from content. (4 marks)
2. Explain the purpose and structure of the templateDetails.xml file. (4/6 marks)
3. Explain the steps to create and install a custom Joomla template package. (6 marks)
4. How is CSS linked to a Joomla template? Give the relevant code. (4 marks)
5. What is the tmpl_builder tool used for? (2 marks)
6. Explain how a custom form's appearance can be changed using CSS in Joomla. (4 marks)
7. What is the significance of using include() inside a Joomla template's index.php file? (2/4 marks)

Unit V in one glance: Template basics (design/content separation) -> customising vs building custom templates -> templateDetails.xml manifest (plus the tmpl_builder helper tool) -> linking CSS/JS through JFactory::getDocument() -> using include() and producing valid XHTML output -> packaging the template as a .zip -> installing it via Extensions Manager -> custom forms styled with CSS.

End of Elaborated Study Notes — Web Technologies (Units I–V)

Revise the Key Point and Likely Exam Question boxes from every unit one final time before the exam. Good luck!