Subject : Web Technologies
Class : III -I CSE
Department : Computer Science and Engineering

Note : All the questions are designed for 12M each.

Unit – I

1. **Discuss in detail about Internet and World Wide Web.** L1
   [CO1] [12]

ANS

The Internet is a global communication system that links together thousands of individual networks. It allows exchange of information between two or more computers on a network.

Thus the internet helps in transfer of messages through mail, chat, video & audioconference, etc.

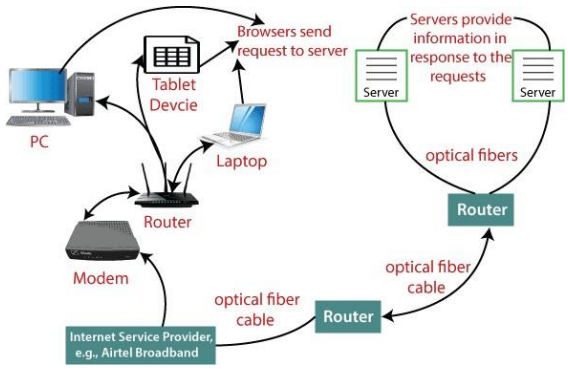
It has become mandatory for day-to-day activities: bills payment, online shopping andsurfing, tutoring, working, communicating with peers, etc.

Internet was evolved in 1969, under the project called ARPANET (Advanced Research Projects Agency Network) to connect computers at different universities and U.S. defence

It uses the standard internet protocol suite (TCP/IP) to connect billions of computer usersworldwide.

It is set up by using cables such as optical fibers and other wireless and networkingtechnologies.

The Internet is the fastest means of sending or exchanging information and data betweencomputers across the world.



Advantages of the Internet:

- Instnat Messaging: You can send messages or communicate to anyone usinginternet, such as email, voice chat, video conferencing, etc.
- Get directions: Using GPS technology, you can get directions to almost every placein a city, country, etc. You can find restaurants, malls, or any other service near your location.
- Online Shopping: It allows you to shop online such as you can be clothes, shoes,book movie tickets, railway tickets, flight tickets, and more.
- Pay Bills: You can pay your bills online, such as electricity bills, gas bills, collegefees, etc.
- Online Banking: It allows you to use internet banking in which you can check yourbalance, receive or transfer money, get a statement, request cheque-book, etc.
- Online Selling: You can sell your products or services online. It helps you reachmore customers and thus increases your sales and profit.

1.1 World Wide Web(WWW)

The World Wide Web(WWW) is an information sharing model that allows accessinginformation over the medium of the Internet.

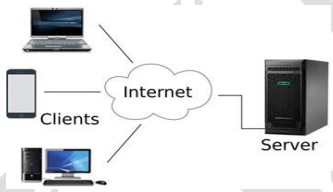
It is the collection of electronic documents that are linked together. These electronic documents are known as `Web Pages'. A collection of related WebPages is known as a `WebSite.

A Website resides on Server computers that are located around the world. Information onthe WWW is always accessible, from anywhere in the world.

World Wide Web, which is also known as a Web, is a collection of websites or web pagesstored in web servers and connected to local computers through the internet. These

websites contain text pages, digital images, audios, videos, etc.

The basic architecture is characterized by a Web Browser that displays informationcontent and Web Server that transfer's information to the client.



Working of WWW:

The World Wide Web is based on several different technologies:

- Web browsers,
- Hypertext Markup Language (HTML),
- Uniform Resource Locator (URL)
- Hypertext Transfer Protocol (HTTP).

☐ A Web browser is used to access web pages. Web browsers can be defined as programs which display text, data, pictures, animation and video on the Internet.

☐ HTML: Hyper Text Markup Language for creating and editing document

content.

☐ URL: Uniform Resource Locator for locating resources on the Internet.

☐ HTTP: HyperText Transfer Protocol to transfer the data.

- The Web was invented in 1991 by Tim Berners-Lee, while consulting at CERN(European Organization for Nuclear Research) in Switzerland.
- The Web is a distributed information system.
- The Web contains multimedia.
- Information on the Web is connected by

hyperlinks.

Features of WWW:

- ➢ HyperText Information System.
- ➢ Cross-Platform.
- ➢ Distributed.
- ➢ Open Standards and Open Source.
- ➢ Uses Web Browsers to provide a single interface for many services.
- ➢ Dynamic, Interactive and Evolving.

2. Give the structure of HTTP request and response message and explain it in detail. L3 [CO1] [12]

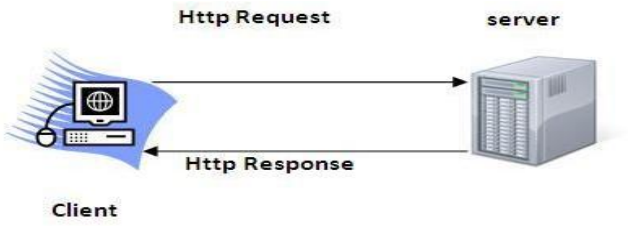
ANS

HTTP (Hyper Text Transfer Protocol)

- The Hypertext Transfer Protocol (HTTP) is application-level protocol forcollaborative, distributed, hypermedia information systems.
- It is the data communication protocol used to establish communication between client and server.
- It is a protocol used to access the data on the World Wide Web (www).
- The HTTP protocol can be used to transfer the data in the form of plain

text,hypertext, audio, video, and so on.

- HTTP is used to carry the data in the form of MIME-like format.(Multipurpose Internet Mail Extension)
- It is the data communication protocol used to establish communication between client and server.



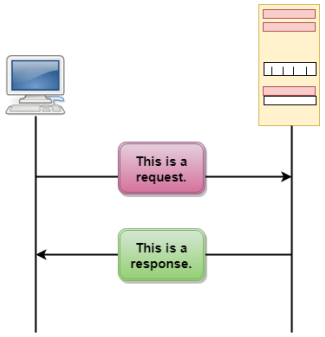
The Basic Characteristics of HTTP

- It is the protocol that allows web servers and browsers to exchange data over theweb.
- It is a request response protocol.
- It uses the reliable TCP connections by default on TCP port 80.
- It is stateless means each request is considered as the new request.

Basic Features of HTTP

- HTTP is media independent: It specifies that any type of media content can be sentby HTTP as long as both the server and the client can handle the data content.
- HTTP is connectionless: The HTTP client, i.e., a browser initiates an HTTP requestand after a request is made, the client waits for the response. The server processes the request and sends a response back after which client disconnect the connection. So client and server knows about each other during current request and response only.

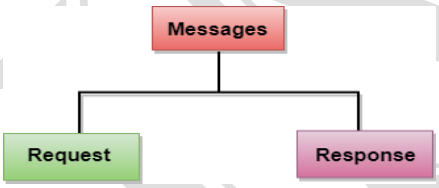
- HTTP is stateless: The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless natureof protocol, neither the client nor the server can retain the



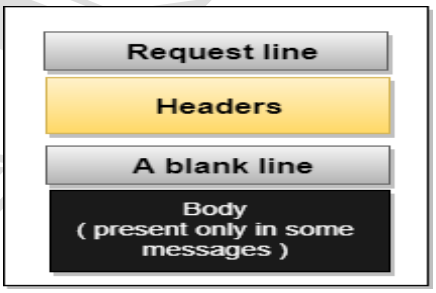
information about different request across the web pages.

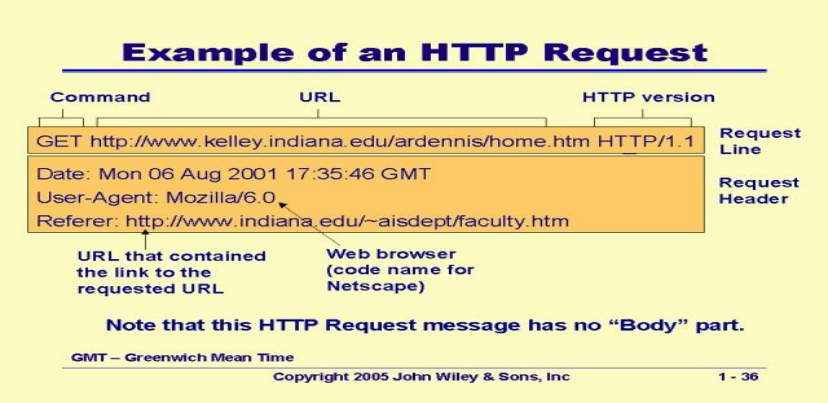
Messages

HTTP messages are of two types: request and response.



Request Message: The request message is sent by the client that consists of a requestline, headers, and body.



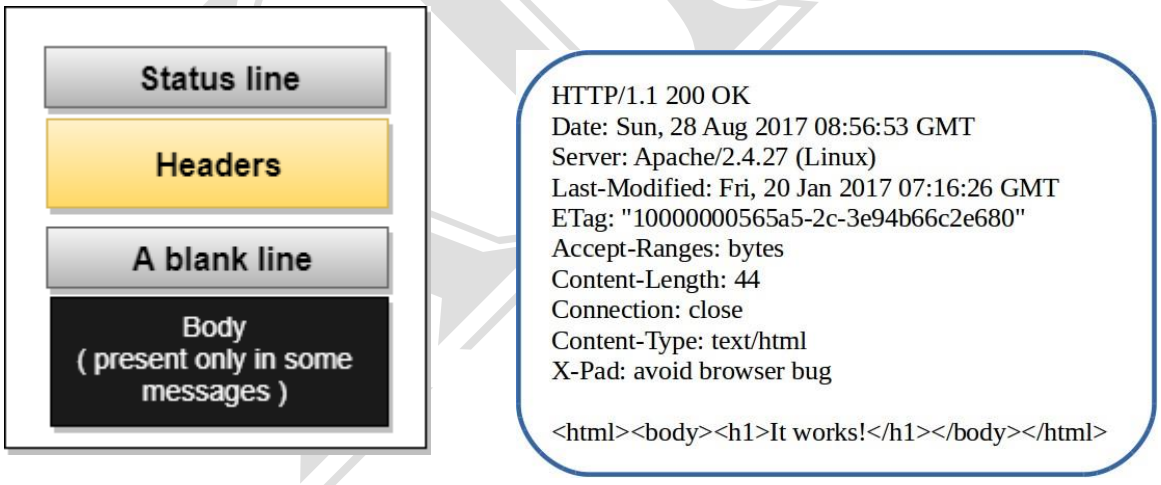


Response Message: The response message is sent by the server to the client thatconsists of a status line, headers, and sometimes a body.

3. **Explain in detail the working of the SMTP Protocol.** L2 [CO1] [12]

Ans

SMTP

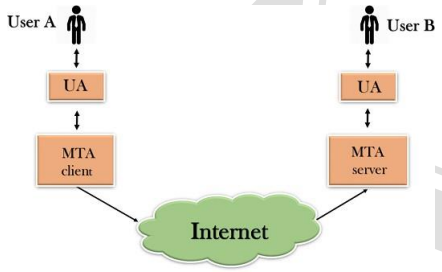


- SMTP stands for Simple Mail Transfer Protocol.
- SMTP is a set of communication guidelines that allowsoftware to transmit an electronic mail over the internet is called Simple Mail Transfer Protocol.
- SMTP (Simple Mail Transfer Protocol) is a TCP/IPprotocol used in sending and receiving e-mail.

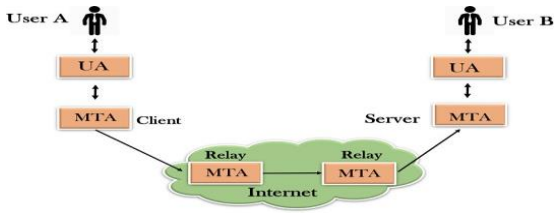
- It is a program used for sending messages to othercomputer users based on e-mail addresses.
- It can send a single message to one or morerecipients.
- Sending message can include text, voice, video orgraphics.
- The main purpose of SMTP is used to set upcommunication rules between servers.

Components of SMTP

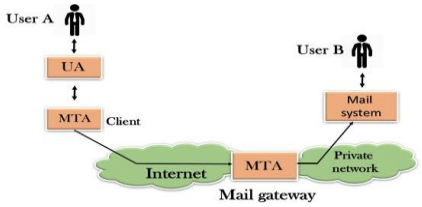
Two components such as user agent (UA) and mailtransfer agent (MTA).
The user agent (UA) prepares the message, creates theenvelope and then puts the message in the envelope.
The mail transfer agent (MTA) transfers this mail acrossthe internet.



SMTP allows a more complex system by adding a relaying system. Instead of just having one MTA at sending side and oneat the receiving side, more MTAs can be added, acting either as a client or server to relay the email.



The relaying system without TCP/IP protocol can also be usedto send the emails to users, and this is achieved by the use of the mail gateway. The mail gateway is a relay MTA that can beused to receive an email.



4. Discuss in detail about the working of the FTP Protocol. L2 [CO1] [12]

Ans

FTP(File Transfer Protocol)

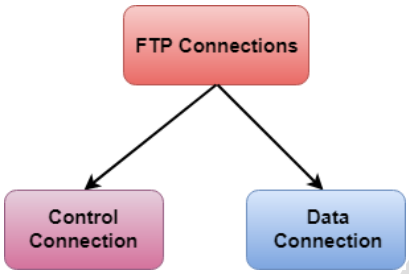
- FTP stands for File transfer protocol.
- File Transfer Protocol (FTP) is an application layer protocol that is used to transfer the files between thelocal devices (PC, smartphone, etc.) to a server. It transfers both text and binary files over the Internet.
- FTP is a standard internet protocol provided by TCP/IP used for transmitting the files from one hostto another.
- The first feature of FTP was developed by AbhayBhushan in 1971.
- It is mainly used for transferring the web page files onthe internet.
- It is also used for downloading the files to thecomputer from other servers.

Objectives of FTP

- It provides the sharing of files.
- It is used to connect remote computers.

- It transfers the data more reliably and efficiently.

  FTP opens two connections between the computers



One connection is used for data connection, and another connection is used for the control connection.

Control Connection

- A Control Connection is established on Port number 21. It is the primary connection and is used to send commands back and forth between the client and the server.
- It is used for sending the control information like user identification, password, and remote directory, etc., once the control connection is established.

Data Connection

Data Connection is initiated on Port number 20.

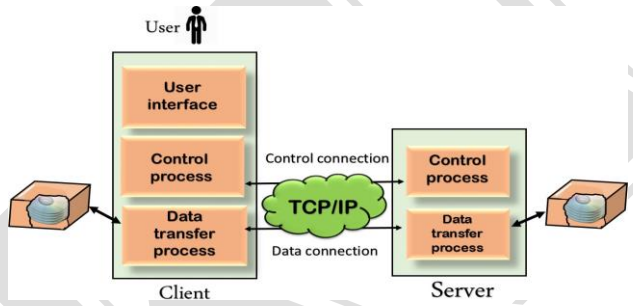
Using the established Control Connection, the client and server will create a separate Data Connection to transfer the requested data.

The Data Connection stays open until the transfer is complete Data Connections are closed by either the client or the server,depending on which party is sending the information.

When a client is retrieving data from a server, the server willclose the connection once all the data has been transferred.

When the client is transferring data to the server, the client will terminate the connection after all the data has been transferred.

Mechanism of FTP



- The FTP client has three components: The user interface, control process, and data transfer process.

  The server has two components: the server controlprocess and the server data transfer process.

FTP transfers files in three different modes –

- Stream mode – Here, the FTP handles the data as a string ofbytes without separating boundaries.

- Block mode – In the block mode, the FTP decomposes theentire data into different blocks of data.
- Compressed mode – In this mode, the FTP uses theLempel-Ziv algorithm to compress the data.

5. State and explain any four HTML5 elements in detail. L1 [CO1] [12]

Ans

HTML5 introduced several new elements that help structure web content more semantically and provide better accessibility. Here are four HTML5 elements explained in detail:

1. `<header>`:
- The `<header>` element represents a container for introductory content or a set of navigational links at the top of a webpage or a section within a webpage.
- It typically includes elements like `<h1>`, `<h2>`, or `<hgroup>` for headings, `<nav>` for navigation menus, and other elements that provide context to the content.
- Example:

```html
<header>
  <h1>My Website</h1>
  <nav>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/about">About</a></li>
      <li><a href="/contact">Contact</a></li>
    </ul>
  </nav>
</header>
```

2. `<section>`:- The `<section>` element defines a section of content within a webpage. It is used to group related content together, making it easier to understand the structure of the page.
- It is often used in conjunction with the `<h1>`, `<h2>`, `<h3>`, etc., elements to provide hierarchical structuring of content.
- Example:

```html
<section>
```

```
    <h2>Introduction</h2>
    <p>Welcome to our website. We provide...</p>
  </section>
  <section>
    <h2>Services</h2>
    <p>We offer a range of services...</p>
  </section>
```

3. `<article>`:
   - The `<article>` element represents a self-contained piece of content that can be independently distributed, reused, or syndicated. It's often used for blog posts, news articles, forum posts, etc.
   - Each `<article>` should have a unique and meaningful title, typically placed within an `<h1>` or an appropriate heading element.
   - Example:
     ```html
     <article>
       <h1>How to Bake the Perfect Cake</h1>
       <p>In this article, we will explore...</p>
     </article>
     ```

4. `<footer>`:
   - The `<footer>` element defines a container for the footer of a section or a webpage. It usually contains information about the author, copyright notices, contact information, or links to related documents.
   - It's a good practice to include a copyright notice and possibly links to a privacy policy and terms of service within the `<footer>` element.
   - Example:
     ```html
     <footer>
       <p>&copy; 2023 My Website. All rights reserved.</p>
       <p>Contact us at <a href="mailto:contact@mywebsite.com">contact@mywebsite.com</a></p>
     </footer>
     ```

These HTML5 elements help improve the structure and semantics of web documents, making them more accessible to both humans and search engines while providing better organization and clarity for developers.

6. Write XSLT code to display employee details in a Table form which is stored is XML. L3 [CO1] [12]

Ans

HTML5 introduced several new elements that help structure web content more semantically and provide better accessibility. Here are four HTML5 elements explained in detail:

1. `<header>`:
   - The `<header>` element represents a container for introductory content or a set of navigational links at the top of a webpage or a section within a webpage.
   - It typically includes elements like `<h1>`, `<h2>`, or `<hgroup>` for headings, `<nav>` for navigation menus, and other elements that provide context to the content.
   - Example:

```html
<header>
  <h1>My Website</h1>
  <nav>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/about">About</a></li>
      <li><a href="/contact">Contact</a></li>
    </ul>
  </nav>
</header>
```

2. `<section>`:
   - The `<section>` element defines a section of content within a webpage. It is used to group related content together, making it easier to understand the structure of the page.
   - It is often used in conjunction with the `<h1>`, `<h2>`, `<h3>`, etc., elements to provide hierarchical structuring of content.
   - Example:

```html
<section>
  <h2>Introduction</h2>
  <p>Welcome to our website. We provide...</p>
</section>
<section>
  <h2>Services</h2>
  <p>We offer a range of services...</p>
```

```
</section>
```

3. `<article>`:
   - The `<article>` element represents a self-contained piece of content that can be independently distributed, reused, or syndicated. It's often used for blog posts, news articles, forum posts, etc.
   - Each `<article>` should have a unique and meaningful title, typically placed within an `<h1>` or an appropriate heading element.
   - Example:
```
```html
<article>
  <h1>How to Bake the Perfect Cake</h1>
  <p>In this article, we will explore...</p>
</article>
```
```

4. `<footer>`:
   - The `<footer>` element defines a container for the footer of a section or a webpage. It usually contains information about the author, copyright notices, contact information, or links to related documents.
   - It's a good practice to include a copyright notice and possibly links to a privacy policy and terms of service within the `<footer>` element.
   - Example:
```
```html
<footer>
  <p>© 2023 My Website. All rights reserved.</p>
  <p>Contact us at <a
href="mailto:contact@mywebsite.com">contact@mywebsite.com</a></p>
</footer>
```
```

These HTML5 elements help improve the structure and semantics of web documents, making them more accessible to both humans and search engines while providing better organization and clarity for developers.

7. **Describe in detail about the different types of DTD with an example. L1 [CO1] [12]**

Ans

Document Type Definitions (DTDs) are a way to define the structure and rules for an XML document. They specify the elements, attributes, and the structure that a valid XML

document must follow. DTDs come in two main types: internal and external. Let's explore each type in detail and provide examples for better understanding:

1. **Internal DTD**:

An internal DTD is defined directly within the XML document itself. It is placed between the `<!DOCTYPE>` declaration and the root element of the XML document. Here's a breakdown of the components of an internal DTD with an example:

```xml
<!DOCTYPE rootElement [
  <!-- Internal DTD declarations go here -->
]>
<rootElement>
  <!-- XML content goes here -->
</rootElement>
```

- `<!DOCTYPE rootElement [...]>`: This declares the document type and links it to the root element of the XML document.

- `[...]`: Inside the square brackets, you define the DTD declarations, including element definitions, attribute definitions, and entity definitions.

- Example of an internal DTD for a simple address book:

```xml
<!DOCTYPE addressbook [
  <!ELEMENT addressbook (contact+)>
  <!ELEMENT contact (name, email, phone?)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT email (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
  <!ATTLIST contact id ID #REQUIRED>
]>
<addressbook>
  <contact id="1">
    <name>John Doe</name>
    <email>john@example.com</email>
    <phone>123-456-7890</phone>
```

```
  </contact>
  <contact id="2">
    <name>Jane Smith</name>
    <email>jane@example.com</email>
  </contact>
</addressbook>
```

In this example, the internal DTD defines rules for an address book XML document, specifying that it should contain a root element called `addressbook`, which consists of one or more `contact` elements. Each `contact` element must have a `name` and `email` element and can optionally have a `phone` element. The `id` attribute is also defined for the `contact` element.

2. **External DTD**:

An external DTD is defined in a separate file and referenced in the XML document using a system identifier or a public identifier. This approach allows for the reuse of DTDs across multiple XML documents. Here's how it works:

- **External DTD File (addressbook.dtd)**:

  ```xml
  <!-- addressbook.dtd -->
  <!ELEMENT addressbook (contact+)>
  <!ELEMENT contact (name, email, phone?)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT email (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
  <!ATTLIST contact id ID #REQUIRED>
  ```

- **XML Document with External DTD Reference**:

  ```xml
  <!DOCTYPE addressbook SYSTEM "addressbook.dtd">
  <addressbook>
    <contact id="1">
      <name>John Doe</name>
      <email>john@example.com</email>
```

```
    <phone>123-456-7890</phone>
  </contact>
  <contact id="2">
    <name>Jane Smith</name>
    <email>jane@example.com</email>
  </contact>
</addressbook>
```

In this example, the DTD declarations are placed in a separate file named `addressbook.dtd`, and the XML document references this external DTD using the `SYSTEM` identifier. This separation of DTD and XML content promotes modularity and reusability.

When working with larger XML documents or sharing DTDs across multiple documents, external DTDs are often preferred for maintainability and consistency.

In summary, DTDs provide a way to define the structure and rules of XML documents. Internal DTDs are defined within the XML document itself, while external DTDs are stored in separate files for reuse. Both types help ensure that XML documents adhere to a specific structure and content model.

8. Discover XML document to store voter ID, voter name, address and date of birth details and validate the document with the help of DTD. L3 [CO1] [12]

Ans

To create an XML document for storing voter ID, voter name, address, and date of birth details and validate it with a Document Type Definition (DTD), you can follow these steps:

Step 1: Create the XML Document

Here's an example of an XML document that stores voter details:

```xml
<?xml version="1.0"?>
<!DOCTYPE voterList SYSTEM "voter.dtd">
<voterList>
  <voter>
    <voterID>V123456</voterID>
    <name>John Doe</name>
    <address>123 Main Street, Cityville</address>
```

```
    <dob>1990-05-15</dob>
  </voter>
  <voter>
    <voterID>V789012</voterID>
    <name>Jane Smith</name>
    <address>456 Elm Avenue, Townsville</address>
    <dob>1985-11-30</dob>
  </voter>
</voterList>
```

In this XML document:

- `voterList` is the root element containing a list of `voter` elements.
- Each `voter` element includes `voterID`, `name`, `address`, and `dob` (date of birth) elements to store the respective information.

Step 2: Create the DTD File

Now, you need to create a Document Type Definition (DTD) file named `voter.dtd` that defines the structure and rules for the XML document.

```xml
<!ELEMENT voterList (voter+)>
<!ELEMENT voter (voterID, name, address, dob)>
<!ELEMENT voterID (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT dob (#PCDATA)>
```

In this DTD:

- `voterList` is defined as the root element, and it should contain one or more `voter` elements.
- Each `voter` element must contain `voterID`, `name`, `address`, and `dob` elements.
- `voterID`, `name`, `address`, and `dob` elements are defined as containing parsed character data (#PCDATA).

Step 3: Validate the XML Document

You can validate the XML document against the DTD using various XML parsers or validators. Here's an example of how to do it in Python using the `lxml` library:

```python
from lxml import etree

# Load the XML document and DTD
xml_file = "voter.xml"
dtd_file = "voter.dtd"

# Parse the XML document and DTD
xml_parser = etree.XMLParser(dtd_validation=True)
xml_tree = etree.parse(xml_file, xml_parser)
dtd = etree.DTD(open(dtd_file))

# Validate the XML document against the DTD
if dtd.validate(xml_tree):
    print("XML document is valid.")
else:
    print("XML document is not valid.")
    print(dtd.error_log)
```

Make sure to adjust the file paths (e.g., `xml_file` and `dtd_file`) in the Python code to match the locations of your XML document and DTD file.

When you run the code, it will parse the XML document, load the DTD, and validate the XML document against the defined rules in the DTD. If the XML document adheres to the DTD's structure and rules, it will be considered valid. Otherwise, validation errors will be displayed.

9. Summarize in detail the XML schema, built in and user defined data types.L3 [CO1] [12]

Ans

XML Schema (XSD), built-in data types, and user-defined data types are essential concepts in XML that help define the structure, data types, and constraints for XML documents. Let's delve into each of these concepts in detail:

1. XML Schema (XSD):

XML Schema, often referred to as XSD, is a standard for describing the structure and content of XML documents. It serves as a blueprint for defining the elements, attributes, data types, and constraints that an XML document must adhere to. XML Schema offers a way to validate XML documents, ensuring they conform to the specified rules.

Key features and components of XML Schema:

- **Elements and Attributes**: XML Schema defines the elements and attributes that can appear in an XML document. Elements can have data types, and attributes can have values defined by XML Schema.

- **Data Types**: XML Schema provides a range of built-in data types (e.g., string, integer, date) and allows for the creation of custom (user-defined) data types.

- **Constraints**: XML Schema allows you to specify constraints such as minimum and maximum occurrences of elements, default values, and patterns for data validation.

- **Complex Types**: XML Schema supports the definition of complex types, including sequences (order of child elements), choices (alternative elements), and groups (logical groupings of elements).

- **Namespaces**: XML Schema can be associated with XML namespaces, allowing for the validation of documents with multiple namespaces.

2. Built-In Data Types:

XML Schema defines a set of built-in data types that are commonly used for elements and attributes in XML documents. These data types provide a standardized way to represent and validate data. Some of the common built-in data types include:

- **xs:string**: Represents text data, including plain text and character strings.

- **xs:integer**: Represents whole numbers, including positive and negative integers.

- **xs:decimal**: Represents decimal numbers, including real numbers with fixed or floating-point precision.

- **xs:date**: Represents dates in the format YYYY-MM-DD.

- **xs:time**: Represents times in the format HH:MM:SS.

- **xs:boolean**: Represents boolean values (true or false).

- **xs:double** and **xs:float**: Represent floating-point numbers with single and double precision, respectively.

These built-in data types can be used to specify the data type of elements and attributes in XML Schema definitions.

3. User-Defined Data Types:

XML Schema allows you to define custom (user-defined) data types when the built-in data types do not meet your specific requirements. User-defined data types can be created using the `<simpleType>` and `<complexType>` elements within an XML Schema definition. Here are some examples of user-defined data types:

- **Enumerations**: You can define an enumeration by specifying a list of allowed values for an element or attribute. For example, you can create an enumeration of country codes.

- **Patterns**: You can use regular expressions to define patterns that values must match. For instance, you can define a pattern for validating email addresses.

- **Restrictions**: You can restrict a built-in data type to add additional constraints. For example, you can define a custom type that restricts the `xs:integer` type to only allow positive values.

User-defined data types provide flexibility for defining specialized data validation rules and constraints tailored to your specific XML document's needs.

In summary, XML Schema (XSD) is a powerful tool for defining the structure and content of XML documents. It includes built-in data types for common use cases and allows you to create custom data types when necessary. This schema-based approach helps ensure data consistency, interoperability, and validation in XML documents.

10. Explain about DOM based XML Parsing L3 [CO1] [12]

Ans

DOM (Document Object Model) based XML parsing is a method used to parse and manipulate XML documents in a programming environment by representing the XML

document as a tree-like structure. This approach provides a comprehensive and versatile way to access and modify XML data within a document.

Here's an explanation of how DOM-based XML parsing works and its key components:

1. **Parsing XML into a DOM Tree**:

   - **Parsing**: In DOM-based parsing, an XML parser is used to read an XML document and convert it into a structured tree-like representation in memory. This tree is called the Document Object Model (DOM) tree.

   - **DOM Tree**: The DOM tree consists of various nodes that represent different parts of the XML document, such as elements, attributes, text nodes, comments, and more. These nodes are organized hierarchically, reflecting the structure of the XML document.

2. **Accessing and Modifying Data**:

   Once the XML document is parsed into a DOM tree, you can perform various operations on the data:

   - **Traversal**: You can traverse the DOM tree by moving between nodes, such as navigating from parent nodes to child nodes or sibling nodes. This allows you to access specific elements or attributes within the document.

   - **Accessing Data**: You can read data from the XML document by accessing the values stored in the nodes. For example, you can extract element text content or attribute values.

   - **Modifying Data**: You can make changes to the XML document by adding, modifying, or deleting nodes and their content. This enables you to update the document programmatically.

3. **Example in JavaScript**:

   Here's an example of DOM-based XML parsing in JavaScript using the browser's built-in DOM manipulation capabilities:

   ```javascript
   // Load an XML document
   const parser = new DOMParser();
   ```

```
const xmlString = '<bookstore><book><title>Harry Potter</title></book></bookstore>';
const xmlDoc = parser.parseFromString(xmlString, 'text/xml');

// Access and modify data
const bookTitles = xmlDoc.getElementsByTagName('title');
console.log(bookTitles[0].textContent); // Output: "Harry Potter"

// Modify data
const newTitle = xmlDoc.createElement('title');
const textNode = xmlDoc.createTextNode('The Hobbit');
newTitle.appendChild(textNode);
const firstBook = xmlDoc.getElementsByTagName('book')[0];
firstBook.replaceChild(newTitle, bookTitles[0]);

// Serialize the modified DOM back to XML
const serializedXML = new XMLSerializer().serializeToString(xmlDoc);
console.log(serializedXML);
```

In this JavaScript example, we first parse an XML string into a DOM tree using `DOMParser`. Then, we access and modify the data in the document, replacing the title of the first book with a new title. Finally, we serialize the modified DOM back into an XML string.

DOM-based XML parsing is widely used in programming languages like JavaScript, Java, Python, and others. It provides a flexible way to work with XML data, making it suitable for tasks such as data extraction, transformation, and document generation in web applications, data processing pipelines, and more.

Unit – 2

11. Summarize in different types of arrays in PHP with suitable examples. L2 [CO1] [12]

Ans

In PHP, arrays are versatile data structures used to store and manipulate collections of values. PHP offers several types of arrays to suit different needs. Here's a summary of the various types of arrays in PHP with suitable examples:

1. **Indexed Arrays**:

- Indexed arrays, also known as numeric arrays, use numerical indices (starting from 0) to access elements.
- They are the most common type of array in PHP.

```php
// Indexed array
$colors = array("Red", "Green", "Blue");
echo $colors[0]; // Output: "Red"
```

2. **Associative Arrays**:

- Associative arrays use named keys (strings) to access elements. Each key is associated with a specific value.

```php
// Associative array
$person = array("first_name" => "John", "last_name" => "Doe", "age" => 30);
echo $person["first_name"]; // Output: "John"
```

3. **Multidimensional Arrays**:

- Multidimensional arrays contain arrays as elements. They are used to represent more complex data structures, like matrices or tables.

```php
// Multidimensional array (matrix)
$matrix = array(
    array(1, 2, 3),
    array(4, 5, 6),
    array(7, 8, 9)
);
echo $matrix[1][2]; // Output: 6
```

4. **Sequential Arrays**:

- Sequential arrays are indexed arrays where keys are automatically assigned starting from 0, and keys are incremented by 1 for each element.

```php
// Sequential array
$fruits = array("Apple", "Banana", "Orange");
echo $fruits[1]; // Output: "Banana"
```

5. **Sparse Arrays**:

   - Sparse arrays are indexed arrays with non-contiguous keys (e.g., missing keys). They are allowed in PHP and can be useful for optimizing memory usage.

```php
// Sparse array
$sparseArray = array(0 => "A", 2 => "B", 4 => "C");
echo $sparseArray[2]; // Output: "B"
```
```

These are the main types of arrays in PHP, each serving different purposes. You can choose the appropriate array type based on your specific data storage and manipulation requirements in PHP applications.

**12. Design a simple HTML form with validation in PHP. L3 [CO2] [12]**

Ans

```
<!DOCTYPE html>
<html>
<head>
    <title>Login Form</title>
</head>
<body>
    <?php
    $validUsername = "DUMPA ANANTH";
    $validPassword = "KING";
    $loginMessage = "";

    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $username = $_POST["username"];
        $password = $_POST["password"];
```
```

```
        if ($username == $validUsername && $password == $validPassword) {
            $loginMessage = "Login successful!";
        } else {
            $loginMessage = "Invalid username or password.";
        }
    }
    ?>

    <h2>Login Form</h2>
    <form method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>

        <input type="submit" value="Login">
    </form>

    <p><?php echo $loginMessage; ?></p>
</body>
</html>
```

13. Explain variables and operators with example in PHP. L1 [CO2] [12]

Ans

What is Operators in PHP

Operators are symbols that tell the PHP processor to perform certain actions. For example, the addition (+) symbol is an operator that tells PHP to add two variables or values, while the greater-than(>) symbol is an operator that tells PHP to compare two values.

The following lists describe the different operators used in PHP.

PHP Arithmetic Operators

Operator	Description	Example	Result

+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y.
*	Multiplication	$x * $y	Product of $x and $y.
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $

The arithmetic operators are used to perform common arithmetical operations, such as addition, subtraction, multiplication etc. Here's acomplete list of PHP's arithmetic operators:

The following example will show you these arithmetic operators inaction:

Example

Run this code »

```
<?php
$x = 10;
$y = 4;
echo($x + $y); // 0utputs: 14 echo($x - $y);
// 0utputs: 6 echo($x * $y); // 0utputs: 40
echo($x / $y); // 0utputs: 2.5echo($x % $y);
// 0utputs: 2
?>
```

PHP Assignment Operators

The assignment operators are used to assign values to variables.

The

Operator	Description	Example	Is The S
=	Assign	$x = $y	$x =
+=	Add and assign	$x += $y	$x =
-=	Subtract and assign	$x -= $y	$x =
*=	Multiply and assign	$x *= $y	$x =
/=	Divide and assign quotient	$x /= $y	$x =
%=	Divide and assign modulus	$x %= $y	$x =

following example will show you these assignment operators inaction:

Example

Run this code »

<?php

```
$x =    10;
echo    $x; //          Outputs:    10

 $x =   20;
$x += 30;
 echo   $x; //          Outputs:    50

 $x =   50;
$x -= 20;
 echo   $x; //          Outputs:    30

 $x =   5;
$x *= 25;
 echo   $x; //          Outputs:    125

 $x =   50;
$x /= 10;
 echo   $x; //          Outputs:    5

 $x =   100;
$x %= 15;
echo $x; // Outputs: 10

?>
```

PHP Comparison Operators

The comparison operators are used to compare two values in aBoolean fashion.

Operator	Name	Example	Result
==	Equal	$x == $y	True if $x is equal to $y
===	Identical	$x === $y	True if $x is equal to $y, and they are
!=	Not equal	$x != $y	True if $x is not equal to $y

<>	Not equal	$x <> $y	True if $x is not equal to $y
!==	Not identical	$x !== $y	True if $x is not equal to $y, or they atype
<	Less than	$x < $y	True if $x is less than $y
>	Greater than	$x > $y	True if $x is greater than $y
>=	Greater than or equal to	$x >= $y	True if $x is greater than or equal to $
<=	Less than or equal to	$x <= $y	True if $x is less than or equal to $y

The following example will show you these comparison operators inaction:

Example

Run this code »

```
<?php
$x = 25;
$y = 35;
$z = "25";
var_dump($x == $z); // Outputs: boolean true var_dump($x === $z); // Outputs: boolean falsevar_dump($x != $y); // Outputs: boolean true var_dump($x !== $z); // Outputs: boolean true
var_dump($x < $y);                // Outputs: boolean true
var_dump($x > $y);                // Outputs: boolean false
var_dump($x <= $y); // Outputs: boolean true var_dump($x >= $y); // Outputs: boolean false
?>
```

PHP Incrementing and DecrementingOperators

The increment/decrement operators are used to increment/decrement a variable's value.

Operator	Name	Effect
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

The following example will show you these increment and decrement operators in action:

Example

Run this code »

```
<?php
$x = 10;
echo ++$x;     // Outputs: 11
echo $x;       // Outputs: 11

$x = 10;
echo $x++;     // Outputs: 10
echo $x;       // Outputs: 11

$x = 10;
echo --$x;     // Outputs: 9
echo $x;       // Outputs: 9

$x = 10;
echo $x--;     // Outputs: 10
echo $x;       // Outputs: 9
?>
```

PHP Logical Operators

The logical operators are typically used to combine conditional statements.

Operator	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

The following example will show you these logical operators inaction:

Example

Run this code »

```
<?php
$year = 2014;
// Leap years are divisible by 400 or by 4 but not100
if(($year % 400 == 0) || (($year % 100 != 0) &&
($year % 4 == 0))){
        echo "$year is a leap year.";
} else{
        echo "$year is not a leap year.";
}
```

```
?>
```

PHP String Operators

There are two operators which are specifically designed for strings.

Operator	Description	Example	Result
.	Concatenation	$str1 . $str2	Concatenation of $
.=	Concatenation assignment	$str1 .= $str2	Appends the $str2

The following example will show you these string operators inaction:

Example

Run this code »

```
<?php
$x = "Hello";
$y = " World!";
echo $x . $y; // Outputs: Hello World!

$x .= $y;
echo $x; // Outputs: Hello World!
?>
```

PHP Array Operators

The array operators are used to compare arrays:

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	True if $x and $y have the same key/value pairs
===	Identity	$x === $y	True if $x and $y have the same key/value pairs in thesame types
!=	Inequality	$x != $y	True if $x is not equal to $y
<>	Inequality	$x <> $y	True if $x is not equal to $y
!==	Non-identity	$x !== $y	True if $x is not identical to $y

The following example will show you these array operators in action:

Example

Run this code »

```
<?php
$x = array("a" => "Red", "b" => "Green", "c" =>"Blue");
$y = array("u" => "Yellow", "v" => "Orange", "w" =>"Pink");
$z = $x + $y; // Union of $x and $y
var_dump($z);
var_dump($x == $y);              // Outputs: boolean false
var_dump($x === $y); // Outputs: boolean false
var_dump($x != $y);              // Outputs: boolean true
var_dump($x <> $y);              // Outputs: boolean true
var_dump($x !== $y);   // Outputs: boolean true
?>
```

14. Demonstrate various control statements in PHP.
L1 [CO2] [12]

Ans

Control statements in PHP are used to control the flow of execution in a script based on certain conditions or loops. They include conditional statements (if, else, switch), looping statements (for, while, foreach), and branching statements (break, continue, return). Let's demonstrate various control statements in PHP with examples:

Conditional Statements:

1. **if-else Statement**:

```php
$age = 25;

if ($age >= 18) {
   echo "You are an adult.";
} else {
   echo "You are not yet an adult.";
}
```

2. **Switch Statement**:

```php
$day = "Monday";

switch ($day) {
   case "Monday":
      echo "It's the start of the week.";
      break;
   case "Friday":
      echo "It's almost the weekend!";
      break;
   default:
      echo "It's a regular day.";
}
```

Looping Statements:

3. **for Loop**:

```php
for ($i = 1; $i <= 5; $i++) {
    echo "Iteration $i<br>";
}
```

4. **while Loop**:

```php
$num = 1;

while ($num <= 5) {
    echo "Value: $num<br>";
    $num++;
}
```

5. **foreach Loop** (for iterating over arrays):

```php
$colors = array("Red", "Green", "Blue");

foreach ($colors as $color) {
    echo "Color: $color<br>";
}
```

Branching Statements:

6. **break Statement** (used in loops to exit prematurely):

```php
for ($i = 1; $i <= 10; $i++) {
    if ($i == 5) {
```

```
        break;
    }
    echo "Iteration $i<br>";
}
```

7. **continue Statement** (used in loops to skip the current iteration):

```php
for ($i = 1; $i <= 5; $i++) {
    if ($i == 3) {
        continue;
    }
    echo "Iteration $i<br>";
}
```

8. **return Statement** (used in functions to return a value and exit the function):

```php
function add($a, $b) {
    return $a + $b;
}

$result = add(3, 4);
echo "Result: $result";
```

These control statements provide the ability to conditionally execute code, iterate over data structures, and manage program flow in PHP applications. Depending on the specific requirements of your code, you can use these control statements to achieve different logic and functionality.

15. Explain the predefined and user defined functions in PHP with example. L2 [CO2] [12]

Ans

In PHP, functions are blocks of reusable code that can be called to perform specific tasks or computations. They help modularize code, improve readability,

and promote code reuse. Functions in PHP can be categorized into predefined (built-in) functions and user-defined functions.

Predefined (Built-in) Functions in PHP:

Predefined functions are functions that are provided by PHP itself. They cover a wide range of tasks, from mathematical calculations to string manipulations and file handling. Here are some examples of predefined functions in PHP:

1. **`strlen()`** - String Length:

This function returns the length (number of characters) of a string.

```php
$str = "Hello, World!";
$length = strlen($str);
echo "Length of the string: $length"; // Output: Length of the string: 13
```

2. **`array_push()`** - Add Element to Array:

This function adds one or more elements to the end of an array.

```php
$fruits = array("Apple", "Banana");
array_push($fruits, "Orange", "Mango");
print_r($fruits);
// Output: Array ( [0] => Apple [1] => Banana [2] => Orange [3] => Mango )
```

3. **`file_get_contents()`** - Read File Contents:

This function reads the entire contents of a file into a string.

```php
$fileContent = file_get_contents("example.txt");
echo $fileContent;
```

User-Defined Functions in PHP:

User-defined functions are functions created by the programmer to perform specific tasks based on their requirements. Here's an example of a user-defined function:

```php
// User-defined function to calculate the square of a number
function square($num) {
    return $num * $num;
}

$result = square(5);
echo "Square of 5 is: $result"; // Output: Square of 5 is: 25
```

In this example:

- We define a function named `square` that takes one parameter (`$num`).
- Inside the function, we calculate the square of the input number.
- We call the `square` function with the argument `5` and store the result in the `$result` variable.

User-defined functions are beneficial for encapsulating logic, making code more readable, and promoting code reusability. They allow you to create custom functions tailored to your specific application's needs.

Here's another example of a user-defined function that calculates the factorial of a number using recursion:

```php
// User-defined function to calculate the factorial of a number
function factorial($n) {
    if ($n <= 1) {
        return 1;
    } else {
        return $n * factorial($n - 1);
    }
}
```

```
$result = factorial(5);
echo "Factorial of 5 is: $result"; // Output: Factorial of 5 is: 120
```

In this example, the `factorial` function uses recursion to calculate the factorial of a number, demonstrating the flexibility and power of user-defined functions in PHP.

16. Define Cookies. How to create and retrieve cookies in PHP with example. L2 [CO2] [12]

Ans

Cookies are small pieces of data that a web server sends to a user's web browser and are stored on the user's device. Cookies are often used to store information about a user's interaction with a website. They can be accessed and modified both on the client-side (in JavaScript) and on the server-side (in PHP, for example).

In PHP, you can create and retrieve cookies using the `setcookie()` function to set a cookie and `$_COOKIE` superglobal array to retrieve the cookie's value.

Here's how to create and retrieve cookies in PHP with examples:

Creating Cookies:

To create a cookie in PHP, you use the `setcookie()` function. It takes several parameters, but the most commonly used ones are the name, value, and optional expiration time.

```php
// Syntax: setcookie(name, value, expire, path, domain, secure, httponly);

// Example: Create a cookie with name "user" and value "John" that expires in 1 hour
setcookie("user", "John", time() + 3600);
```

In this example:

- `"user"` is the name of the cookie.

- `"John"` is the value associated with the cookie.
- `time() + 3600` sets the expiration time to 1 hour from the current time.

Retrieving Cookies:

To retrieve cookies in PHP, you can use the `$_COOKIE` superglobal array. It contains all the cookies sent by the client.

```php
if (isset($_COOKIE["user"])) {
    $username = $_COOKIE["user"];
    echo "Welcome back, $username!";
} else {
    echo "No cookie found!";
}
```

In this example:

- We check if a cookie named `"user"` exists using `isset($_COOKIE["user"])`.
- If the cookie exists, we retrieve its value using `$_COOKIE["user"]`.

Example: Setting and Retrieving Cookies:

Let's create a PHP script that sets a cookie when a user visits a page and retrieves it on subsequent visits:

set_cookie.php:

```php
<?php
// Set a cookie that expires in 1 hour
setcookie("user", "John", time() + 3600);
echo "Cookie set!";
?>
```

retrieve_cookie.php:

```php
<?php
if (isset($_COOKIE["user"])) {
    $username = $_COOKIE["user"];
    echo "Welcome back, $username!";
} else {
    echo "No cookie found!";
}
?>
```

Here's how it works:

1. Visit `set_cookie.php` to set the cookie.
2. Visit `retrieve_cookie.php` to retrieve the cookie and see a message like "Welcome back, John!" if the cookie is set.

Remember that cookies are stored on the client-side and can be modified or deleted by the user. They are often used for storing user preferences or session identifiers, but sensitive information should not be stored in cookies for security reasons.

17. **Explain the steps in the PHP code for querying a database with suitable examples.** L3 [CO2] [12]

Ans

Querying a database in PHP involves several steps, including establishing a database connection, creating and executing SQL queries, fetching and processing the results, and handling errors. Here are the key steps in PHP code for querying a database, along with suitable examples using MySQL as the database:

Step 1: Establish a Database Connection

You need to establish a connection to the database using credentials such as the host, username, password, and database name. PHP provides functions like `mysqli_connect` or PDO for this purpose.

```php
$host = "localhost";
```

```
$username = "your_username";
$password = "your_password";
$database = "your_database";

$conn = mysqli_connect($host, $username, $password, $database);

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
```

Step 2: Create and Execute SQL Queries

You can create SQL queries as strings and execute them using functions like `mysqli_query` or PDO's `prepare` and `execute` methods.

```php
$query = "SELECT * FROM users";
$result = mysqli_query($conn, $query);

if (!$result) {
    die("Query failed: " . mysqli_error($conn));
}
```

Step 3: Fetch and Process Results

After executing a query, you can fetch and process the results using functions like `mysqli_fetch_assoc` or PDO's `fetch`.

```php
while ($row = mysqli_fetch_assoc($result)) {
    echo "User ID: " . $row["user_id"] . "<br>";
    echo "Username: " . $row["username"] . "<br>";
    // Process other columns as needed
}
```

Step 4: Close the Database Connection

It's essential to close the database connection when you're done to free up resources.

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

Step 5: Handle Errors and Exceptions

Proper error handling is crucial. You should use try-catch blocks for PDO or check for errors using `mysqli_error` for mysqli.

Here's a complete example querying a user table and handling potential errors using mysqli:

```php
$host = "localhost";
$username = "your_username";
$password = "your_password";
$database = "your_database";

$conn = mysqli_connect($host, $username, $password, $database);

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$query = "SELECT * FROM users";
$result = mysqli_query($conn, $query);

if (!$result) {
    die("Query failed: " . mysqli_error($conn));
}

while ($row = mysqli_fetch_assoc($result)) {
    echo "User ID: " . $row["user_id"] . "<br>";
    echo "Username: " . $row["username"] . "<br>";
    // Process other columns as needed
```

```
}

mysqli_close($conn);
```

Please note that this is a basic example. In real-world scenarios, you may need to use prepared statements to prevent SQL injection, handle more complex queries, and implement more robust error handling.

18. Define Session. Explain by writing a program. L2 [CO2] [12]

Ans

A **session** in web development is a server-side mechanism for maintaining stateful information about a user's interactions with a website across multiple HTTP requests. Sessions are crucial for preserving data between page loads and providing a personalized user experience. Sessions are typically used for tasks like user authentication, shopping cart management, and remembering user preferences.

In PHP, sessions are implemented using a unique session identifier (usually stored in a cookie) that associates a user with a session data storage on the server. Here's a simple PHP program that demonstrates the concept of sessions by creating a basic login/logout system:

```php
<?php
// Start a new session or resume the existing session
session_start();

// Check if the user is logged in
if (isset($_SESSION['user_id'])) {
    // User is logged in; display a welcome message and a logout link
    echo "Welcome, User {$_SESSION['user_id']}! <a
href='logout.php'>Logout</a>";
} else {
    // User is not logged in
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        // Process login form submission
        $username = $_POST['username'];
        $password = $_POST['password'];
```

```
        // Perform basic authentication (replace with a proper authentication
mechanism)
        if ($username === 'user' && $password === 'password') {
            // Authentication successful; store user information in the session
            $_SESSION['user_id'] = $username;
            echo "Login successful! Welcome, $username! <a
href='logout.php'>Logout</a>";
        } else {
            // Authentication failed; display an error message
            echo "Invalid credentials. <a href='login.html'>Try again</a>";
        }
    } else {
        // Display the login form
        echo "<form method='post' action=''>
                <label for='username'>Username:</label>
                <input type='text' id='username' name='username' required><br>
                <label for='password'>Password:</label>
                <input type='password' id='password' name='password'
required><br>
                <input type='submit' value='Login'>
              </form>";
    }
}
?>
```

In this program:

1. We start a PHP session using `session_start()`. If a session already exists, it's resumed; otherwise, a new session is started.

2. We check if the user is logged in by looking for a `user_id` key in the `$_SESSION` superglobal.

3. If the user is not logged in and a POST request is received (indicating that the login form was submitted), we validate the provided username and password (for simplicity, using hardcoded values).

4. If the authentication is successful, we store the username in the session (`$_SESSION['user_id']`).

5. If the authentication fails or the user is not logged in, we display the login form.

This example illustrates the basic concept of using sessions for user authentication. In a real-world application, you would implement a more secure and robust authentication system, including password hashing, database storage, and session management best practices.

19. List and explain scope of variable and super global variable in PHP with example. L1 [CO2] [12]

Ans

In PHP, variables have different scopes, which determine where in your code the variable can be accessed and modified. Additionally, PHP has a set of superglobal variables that are accessible from any part of the script. Let's explore the scope of variables and superglobal variables in PHP with examples:

1. Local Variables:

Local variables are declared within a function or block of code and can only be accessed within that function or block. They are not accessible outside of their defined scope.

Example:

```php
function calculateSum($a, $b) {
    $result = $a + $b; // $result is a local variable
    return $result;
}

echo calculateSum(5, 3); // Outputs: 8
// echo $result; // This will cause an error since $result is not accessible here
```

In this example, `$result` is a local variable within the `calculateSum` function and can only be used within that function.

2. Global Variables:

Global variables are declared outside of any function and can be accessed from any part of the script. To use a global variable within a function, you need to use the `global` keyword to specify that you're referring to the global variable.

Example:

```php
$globalVar = 10; // $globalVar is a global variable

function incrementGlobalVar() {
    global $globalVar;
    $globalVar++;
}

incrementGlobalVar();
echo $globalVar; // Outputs: 11
```

In this example, `$globalVar` is a global variable, and we use the `global` keyword within the `incrementGlobalVar` function to modify its value.

3. Static Variables:

Static variables are declared within a function and retain their values between function calls. They are accessible only within the function where they are defined.

Example:

```php
function countCalls() {
    static $count = 0; // $count is a static variable
    $count++;
    echo "Function called $count times.<br>";
}
```

```
countCalls(); // Outputs: Function called 1 times.
countCalls(); // Outputs: Function called 2 times.
countCalls(); // Outputs: Function called 3 times.
```

In this example, `$count` is a static variable within the `countCalls` function, and its value persists across multiple calls to the function.

Superglobal Variables:

Superglobal variables are built-in global variables that are accessible from anywhere in your PHP script. They are typically used to collect data from forms, handle sessions, and more. Some common superglobal variables include:

- `$_GET`: Contains data sent to the script via URL parameters (HTTP GET method).
- `$_POST`: Contains data sent to the script via HTTP POST method (usually from forms).
- `$_REQUEST`: Combines data from both `$_GET` and `$_POST`.
- `$_SESSION`: Stores session variables that can be used to persist data across multiple requests.
- `$_COOKIE`: Contains data sent by the client's browser in the form of cookies.
- `$_SERVER`: Contains information about the server and the current script's environment.

Example using `$_GET`:

```php
// URL: http://example.com/script.php?name=John&age=30

$name = $_GET['name']; // Access data from the URL
$age = $_GET['age'];

echo "Name: $name, Age: $age"; // Outputs: Name: John, Age: 30
```

In this example, `$_GET` is a superglobal variable that holds data from the URL parameters.

These are the main scopes of variables and some commonly used superglobal variables in PHP. Understanding variable scope and the use of superglobals is essential for effective PHP programming.

20. Explain the process of handling a form in PHP with example. L3 [CO2] [12]

Ans

Handling a form in PHP involves capturing user input from an HTML form, processing that data on the server-side, and responding to the user's input accordingly. Below, I'll outline the steps to handle a form in PHP with an example:

Step 1: Create an HTML Form

First, you need to create an HTML form that allows users to input data. For example, let's create a simple form that captures a user's name and email:

```html
<!DOCTYPE html>
<html>
<head>
  <title>Simple Form</title>
</head>
<body>
  <h1>Simple Form</h1>
  <form action="process_form.php" method="post">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>
    <br><br>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>
    <br><br>

    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

In this example, we have created an HTML form with two fields: "Name" and "Email." The form's `action` attribute specifies where the form data will be sent when the user submits it. We've set it to `process_form.php`.

Step 2: Create a PHP Script to Process the Form Data

Next, you need to create a PHP script (`process_form.php` in this case) that will handle the form data when it's submitted.

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Retrieve form data
    $name = $_POST["name"];
    $email = $_POST["email"];

    // Process the data (in this example, we'll just display it)
    echo "Name: $name<br>";
    echo "Email: $email<br>";

    // You can perform additional processing or database operations here
} else {
    // If the request method is not POST, handle it accordingly
    echo "Invalid request method.";
}
?>
```

In this PHP script:

- We check if the request method is `POST` to ensure that the form has been submitted.
- We retrieve the form data using the `$_POST` superglobal.
- We process the data (in this example, we simply display it back to the user).

Step 3: Display the Form or Results

After processing the form data, you can choose to display a response to the user. In the example above, we displayed the data back to the user. However, in a

real-world scenario, you might save the data to a database or send an email confirmation.

Step 4: Add Validation and Security

In a production environment, it's essential to add validation and security measures to your form handling code. Validate user input to ensure it matches expected formats and sanitize data to prevent security vulnerabilities like SQL injection and cross-site scripting (XSS).

Handling forms in PHP is a fundamental part of web development. As your forms become more complex, you may need to implement additional features like file uploads, user authentication, and more extensive data processing.

Unit – 3

1. **Define Node JS. Explain in detail about Process Model. L1 [CO3] [12]**

Ans

Node.js is an open-source, server-side runtime environment built on Chrome's V8 JavaScript engine. It allows you to execute JavaScript code on the server, which is traditionally used for client-side scripting in web browsers. Node.js is designed to be non-blocking and event-driven, making it highly efficient for building scalable and real-time applications.

One of the key concepts in Node.js is its **Process Model**, which is fundamental to understanding how Node.js handles asynchronous operations. Let's explore the Node.js Process Model in detail:

Node.js Process Model:

1. **Single-Threaded Event Loop**:

Node.js operates on a single-threaded event loop. Unlike traditional multi-threaded server models, where each incoming connection spawns a new thread, Node.js uses a single thread to handle all incoming requests. This single-threaded event loop is at the core of Node.js's non-blocking, asynchronous nature.

2. **Non-Blocking I/O**:

Node.js relies on non-blocking, asynchronous I/O operations to efficiently manage concurrent connections and perform tasks without waiting for one operation to complete before starting another. This means that Node.js can handle a large number of connections concurrently without the need for a separate thread for each.

3. **Event-Driven Architecture**:

Node.js follows an event-driven programming model. It uses event emitters and listeners to respond to events such as incoming requests, file system operations, and network activities. When an event occurs, Node.js triggers the associated callback function, allowing developers to write code that responds to events in a non-blocking manner.

4. **Event Loop**:

The event loop is the central component of Node.js's architecture. It continuously checks the event queue for pending events and executes the associated callback functions. This enables Node.js to efficiently handle I/O operations, timers, and other asynchronous tasks while keeping the main thread free for handling new requests.

5. **Callbacks**:

Callback functions are a common pattern in Node.js. When an asynchronous operation is initiated, a callback function is provided to handle the result or response when the operation is completed. Callbacks are executed once the operation finishes, allowing other tasks to proceed in the meantime.

6. **Concurrency and Scalability**:

Node.js's event-driven, non-blocking model makes it highly suitable for building scalable and real-time applications, such as web servers, chat applications, and streaming services. It can efficiently handle a large number of concurrent connections with relatively low resource usage.

Example of Asynchronous I/O in Node.js:

Here's a simple example to illustrate the non-blocking, asynchronous nature of Node.js:

```javascript
const fs = require('fs');

// Asynchronously read a file
fs.readFile('example.txt', 'utf8', (error, data) => {
  if (error) {
    console.error('Error:', error);
    return;
  }
  console.log('File Contents:', data);
});

console.log('Reading file...');
```

In this example, the `fs.readFile` function initiates an asynchronous file read operation. While the file is being read, the event loop continues executing the `console.log('Reading file...')` statement without waiting for the file read to complete. Once the file read operation finishes, the callback function is invoked to handle the data or any errors.

Node.js's Process Model allows developers to build highly performant and scalable applications by efficiently managing I/O-bound tasks and enabling real-time communication through events and callbacks.

2. Describe Node JS Built in Modules with an example. L2 [CO3] [12]

Ans

Node.js provides a set of built-in modules that extend its core functionality and make it easy to perform various tasks without the need for external libraries. These built-in modules cover a wide range of functionalities, from file system operations to network communication. Below are some commonly used Node.js built-in modules with examples:

1. `fs` (File System Module):

The `fs` module allows you to work with the file system, including reading and writing files. Here's an example of reading a file synchronously and asynchronously:

```javascript
const fs = require('fs');

// Synchronous file read
const dataSync = fs.readFileSync('example.txt', 'utf8');
console.log('Synchronous Read:', dataSync);

// Asynchronous file read
fs.readFile('example.txt', 'utf8', (error, data) => {
    if (error) {
        console.error('Error:', error);
        return;
    }
    console.log('Asynchronous Read:', data);
});
```

2. `http` (HTTP Module):

The `http` module allows you to create HTTP servers and make HTTP requests. Here's an example of creating an HTTP server:

```javascript
const http = require('http');

const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, World!\n');
});

const port = 3000;
server.listen(port, () => {
    console.log(`Server running at http://localhost:${port}`);
});
```

3. `path` (Path Module):

The `path` module provides utilities for working with file and directory paths. It helps ensure cross-platform compatibility. Example:

```javascript
const path = require('path');

const filePath = '/user/documents/file.txt';
const fileName = path.basename(filePath);
console.log('File Name:', fileName); // Output: File Name: file.txt

const absolutePath = path.join(__dirname, 'files', 'example.txt');
console.log('Absolute Path:', absolutePath);
```

4. `os` (Operating System Module):

The `os` module provides information about the operating system. Example:

```javascript
const os = require('os');

console.log('Hostname:', os.hostname());
console.log('Platform:', os.platform());
console.log('Total Memory (bytes):', os.totalmem());
```

5. `events` (Events Module):

The `events` module allows you to create and handle custom events in your Node.js applications. Example:

```javascript
const EventEmitter = require('events');

class MyEmitter extends EventEmitter {}
```

```
const myEmitter = new MyEmitter();

myEmitter.on('customEvent', () => {
  console.log('Custom event was triggered.');
});

myEmitter.emit('customEvent'); // Output: Custom event was triggered.
```

These are just a few examples of the many built-in modules that Node.js provides. Each module has its own set of functions and methods for performing specific tasks. By leveraging these built-in modules, you can efficiently build a wide variety of applications in Node.js without relying on external libraries for common operations.

3. Explain events in Node JS with suitable example. L2 [CO3] [12]

Ans

In Node.js, events are a core part of its asynchronous, event-driven architecture. Events allow you to create, emit, and handle custom events, enabling you to build applications that respond to various occurrences or interactions. The core module for handling events in Node.js is the `events` module, which provides an `EventEmitter` class for managing events and event listeners.

Here's an explanation of events in Node.js with a suitable example:

Step 1: Import the `events` Module

To work with events, you need to require the `events` module and create an instance of the `EventEmitter` class.

```javascript
const EventEmitter = require('events');
const myEmitter = new EventEmitter();
```

Step 2: Emitting Events

You can use the `emit()` method to trigger (emit) an event. An event can have a name (a string identifier) and can optionally carry data to pass to event listeners.

```javascript
myEmitter.emit('customEvent', 'Event data goes here');
```

In this example, we emit a custom event named `'customEvent'` with some data.

Step 3: Registering Event Listeners

To respond to an event, you can register event listeners using the `on()` or `addListener()` methods. Event listeners are functions that get executed when the specified event occurs.

```javascript
myEmitter.on('customEvent', (data) => {
  console.log('Event occurred with data:', data);
});
```

In this example, we register an event listener for the `'customEvent'` event that logs the event data when the event is emitted.

Step 4: Handling Events

Once you've registered event listeners, they will be executed when the associated event is emitted.

```javascript
// Emit the event, triggering the listener
myEmitter.emit('customEvent', 'Event data goes here');
```

When we emit the `'customEvent'` event, the registered event listener is executed, and it logs the event data.

Complete Example:

Here's a complete example that puts all these steps together:

```javascript
const EventEmitter = require('events');
const myEmitter = new EventEmitter();

// Register an event listener
myEmitter.on('customEvent', (data) => {
  console.log('Event occurred with data:', data);
});

// Emit the event, triggering the listener
myEmitter.emit('customEvent', 'Event data goes here');
```

When you run this code, it will output:

```
Event occurred with data: Event data goes here
```

This demonstrates the basic concept of events in Node.js. You can emit custom events to notify your application of specific occurrences and register event listeners to handle those events. This pattern is especially useful for building event-driven and asynchronous applications, such as web servers, real-time applications, and more.

4. Define NPM. List some features and advantages of Node JS. L1 [CO3] [12]

Ans

NPM (Node Package Manager) is a package manager and dependency management tool for Node.js. It is the default package manager that comes bundled with Node.js and is used to install, manage, and share packages (libraries and modules) written in JavaScript. NPM makes it easy for developers to include third-party libraries and utilities in their Node.js projects, making the development process more efficient and collaborative.

Features of NPM:

1. **Package Installation**: NPM simplifies the process of installing packages, including their dependencies, by providing a single command, `npm install`, to fetch and install packages from the official NPM registry or other sources.

2. **Dependency Management**: NPM automatically manages and tracks dependencies for your project. It generates a `package.json` file that lists all the packages used in your project and their versions, making it easy to reproduce the environment on different machines.

3. **Version Control**: NPM allows you to specify version constraints for packages in your `package.json` file. This helps ensure that your project uses compatible package versions and can be easily maintained over time.

4. **Global and Local Packages**: You can install packages globally to make them available system-wide or locally within a specific project directory, depending on your needs.

5. **Script Execution**: NPM provides a convenient way to run scripts defined in your `package.json` file, allowing you to automate tasks like testing, building, and deployment.

6. **Publishing Packages**: Developers can publish their own packages to the NPM registry, making it easy for others to discover, use, and contribute to their code.

Advantages of Node.js:

1. **Asynchronous and Non-Blocking**: Node.js is designed for asynchronous programming, which makes it highly efficient for handling I/O-bound tasks. It can handle many concurrent connections without the need for multithreading.

2. **Fast Execution**: Node.js uses Google's V8 JavaScript engine, known for its speed and performance. This allows Node.js applications to execute quickly and efficiently.

3. **Event-Driven Architecture**: Node.js follows an event-driven programming model, making it suitable for building real-time applications and handling events efficiently.

4. **Large Ecosystem**: NPM hosts a vast ecosystem of open-source packages, libraries, and modules that can be easily integrated into Node.js applications. This saves development time and effort.

5. **Cross-Platform**: Node.js is available on various platforms, including Windows, macOS, and Linux, making it a versatile choice for developing applications that can run on different operating systems.

6. **Community Support**: Node.js has a large and active community of developers, which means you can find extensive resources, documentation, and support online.

7. **Scalability**: Node.js applications can be easily scaled both horizontally and vertically, making it suitable for handling high levels of traffic and load balancing.

8. **Full-Stack Development**: Node.js can be used for both server-side and client-side development, making it a versatile choice for full-stack development.

9. **Real-Time Web Applications**: Node.js is well-suited for building real-time web applications, such as chat applications and online gaming platforms, where real-time communication is essential.

Overall, Node.js and NPM form a powerful combination for JavaScript developers, allowing them to build fast, efficient, and scalable applications while leveraging a rich ecosystem of open-source packages and libraries.

5. Explain Node.js File System Module with suitable example. L2 [CO3] [12]

Ans

The Node.js File System (fs) module provides a set of functions for working with the file system, allowing you to perform various operations such as reading, writing, updating, and deleting files. This module is especially useful for performing I/O operations on files and directories. Here, I'll explain the Node.js File System module with a suitable example.

Example: Reading a File Using the fs Module

In this example, we will use the `fs` module to read the contents of a text file synchronously and asynchronously.

1. Synchronous File Read:

```javascript
const fs = require('fs');

try {
    // Synchronously read a file
    const data = fs.readFileSync('example.txt', 'utf8');
    console.log('Synchronous Read:');
    console.log(data);
} catch (error) {
    console.error('Error:', error);
}
```

In the synchronous approach:

- We use the `fs.readFileSync()` function to read the contents of the file named 'example.txt' using the 'utf8' encoding.
- The `readFileSync` function blocks the execution of your code until the file is fully read or an error occurs.

2. Asynchronous File Read:

```javascript
const fs = require('fs');

// Asynchronously read a file
fs.readFile('example.txt', 'utf8', (error, data) => {
    if (error) {
        console.error('Error:', error);
    } else {
        console.log('Asynchronous Read:');
        console.log(data);
    }
});
```

```

In the asynchronous approach:

- We use the `fs.readFile()` function to read the file 'example.txt' asynchronously.
- The callback function is executed once the file is read or when an error occurs.
- The asynchronous approach does not block the execution of your code, allowing other tasks to continue while the file is being read.

**Creating an Example File (example.txt):**

Before running the code, make sure you have a file named 'example.txt' in the same directory as your Node.js script. You can create this file and add some text content to it.

**Running the Code:**

After creating the 'example.txt' file, you can run either the synchronous or asynchronous code example to read the file's contents. The code will display the content of 'example.txt' in the console.

Remember that the asynchronous approach is generally preferred in Node.js, especially in scenarios where you want to avoid blocking the event loop, allowing your application to remain responsive to other tasks. However, the synchronous approach may be suitable for certain use cases, such as script initialization.

**6. Describe Node JS HTTP Modules with an example. L2 [CO3] [12]**

**Ans**

In Node.js, the HTTP module allows you to create HTTP servers and make HTTP requests. It's a core module that simplifies building web servers and handling HTTP communication. Here, I'll describe the Node.js HTTP module with an example of creating a simple HTTP server.

**Example: Creating a Simple HTTP Server**

In this example, we'll use the Node.js HTTP module to create a basic HTTP server that listens on a specified port and responds to incoming HTTP requests.
```

```javascript
const http = require('http');

// Create an HTTP server
const server = http.createServer((req, res) => {
  // Set the response header
  res.writeHead(200, { 'Content-Type': 'text/plain' });

  // Write the response body
  res.end('Hello, Node.js HTTP Server!\n');
});

const port = 3000;

// Listen on the specified port
server.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

Here's a breakdown of the key components in the example:

1. We import the `http` module, which provides functionality for creating an HTTP server and handling HTTP requests and responses.

2. We create an HTTP server using the `http.createServer()` method. This method takes a callback function that gets executed for each incoming HTTP request. The callback function receives two arguments: `req` (the request object) and `res` (the response object).

3. Inside the callback function, we set the response header using `res.writeHead()`. In this example, we set the status code to `200` (OK) and specify the content type as plain text.

4. We write the response body using `res.end()`. In this case, we send the "Hello, Node.js HTTP Server!\n" message as the response.

5. We define the `port` on which the server will listen for incoming requests (e.g., port 3000).

6. Finally, we use the `server.listen()` method to start the server and specify the port to listen on. When the server starts listening, it logs a message to the console.

Running the HTTP Server:

To run this code, save it to a file (e.g., `http_server.js`) and execute it using Node.js:

```bash
node http_server.js
```

Once the server is running, you can access it in your web browser or using tools like `curl`. Open a browser and visit `http://localhost:3000/`, and you should see the "Hello, Node.js HTTP Server!" message in the browser.

This example demonstrates the basics of creating an HTTP server in Node.js. You can extend this server to handle various types of HTTP requests, route requests to different handlers, and build more complex web applications and APIs using Node.js's HTTP module.