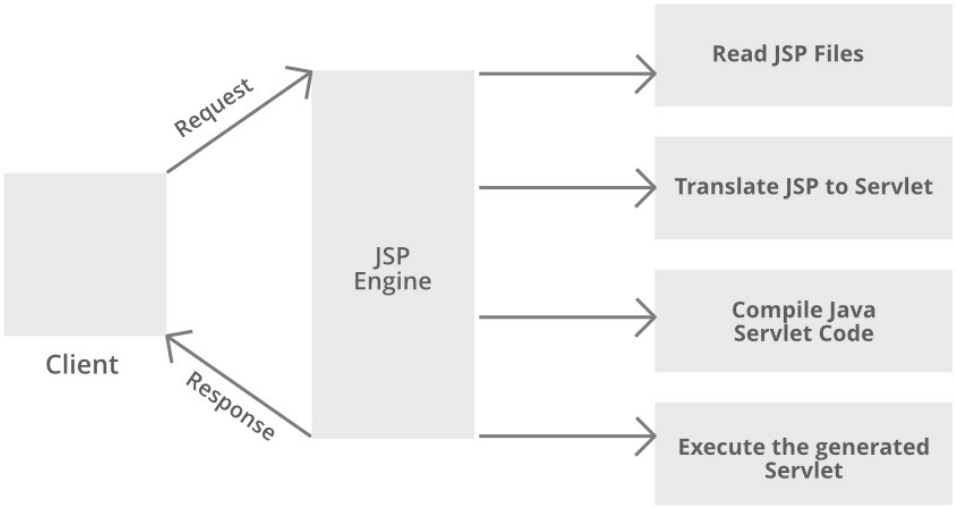
MODULE – IV

Introduction to JSP: The Anatomy of a JSP Page, JSP Processing:



JSP (JavaServer Pages) is a server-side technology that allows developers to embed Java code within HTML pages to create dynamic web applications. A JSP page is processed by a JSP engine within a web container (like Apache Tomcat), which converts it into a servlet that is then executed to generate the final HTML response sent to the client.

JSP is an advanced version of Servlets. It provides enhanced capabilities for building scalable and platform-independent web pages.

How is JSP More Advantageous than Servlets?

JSP simplifies web development by combining the strengths of Java with the flexibility of HTML. Some advantages of JSP over Servlets are listed below:

- JSP code is easier to manage than Servlets as it separates UI and business logic.
- JSP minimizes the amount of code required for web applications.
- Generate content dynamically in response to user interactions.
- It provides access to the complete range of Java APIs for robust application development.
- JSP is suitable for applications with growing user bases.

Key Features of JSP

- It is platform-independent; we can write once, run anywhere.
- It simplifies database interactions for dynamic content.

- It contains predefined objects like request, response, session and application, reducing development time.
- It has built-in mechanisms for exception and error management.
- It supports custom tags and tag libraries.

JSP Architecture

JSP follows a three-layer architecture:

- **Client Layer**: The browser sends a request to the server.
- **Web Server Layer**: The server processes the request using a JSP engine.
- **Database/Backend Layer**: Interacts with the database and returns the response to the client.

Differences Between JSP and Servlets

Features	JSP	Servlet
Code Length	Jsp required less code	Servlets required more code
Ease of Use	Jsp is simple to use	Servlet is more complex to use
Dynamic Content	Easily embedded in HTML	Requires HTML generation in code
Page Maintenance	Easier to maintain	More challenging

Steps to Create a JSP Application

JSP (JavaServer Pages) is a technology used to create dynamic web applications by embedding Java code directly into HTML pages. Follow these simple steps to create your first JSP application.

Prerequisite:

- *Install Java JDK (8 or above)*
- *Install Eclipse IDE (or any IDE with web support)*
- *Download and configure Apache Tomcat server in Eclipse*

The anatomy of a JSP page:-

A JSP page is a text-based document that combines static markup (like HTML) with dynamic JSP elements.

Static template text

This is the standard, static part of the web page, such as HTML tags, plain text, and CSS. The JSP container passes this content directly to the client's browser without modification.

Dynamic JSP elements

These are special tags that the JSP engine processes to generate dynamic content. The main types of elements include:

- Directives <%@ ... %>: Provide global instructions to the JSP container for the entire page.
    - page: Sets page-dependent attributes like the scripting language, error page, and imported Java classes (<%@ page import="java.util.Date" %>).
    - include: Inserts the content of another file during the translation phase (<%@ include file="header.jsp" %>).
    - taglib: Declares a tag library for custom actions (<%@ taglib uri="..." prefix="..." %>).
- Declarations <%! ... %>: Define class-level variables and methods that are added to the servlet class and are available throughout the JSP page.
- Scriptlets <% ... %>: Embed any valid Java code within the JSP page, typically for business logic or flow control.
    - Example: <% for (int i=0; i<5; i++) { out.println("Hello
"); } %>
- Expressions <%= ... %>: Evaluate a Java expression and insert the result directly into the HTML output. It is a shortcut for out.print().
    - Example: Current time is: <%= new java.util.Date() %>
- Actions <jsp: ... />: Perform predefined actions using an XML-like syntax. They control the flow between pages and interact with JavaBeans.
    - Example: <jsp:include page="footer.jsp" /> includes another page dynamically at request time.

JSP processing

When a client requests a .jsp page, the web container's JSP engine processes it through a series of phases.

1. Translation: The JSP engine translates the JSP page into a Java servlet source file. It converts static HTML into println() statements and JSP elements into Java code within the servlet.
2. Compilation: The generated servlet source file (.java) is compiled into a servlet class file (.class) that the Java Virtual Machine (JVM) can execute.
3. Class Loading: The servlet class is loaded into memory by the class loader.
4. Instantiation: An instance of the servlet class is created. The container manages a single instance of the servlet to handle multiple requests.
5. Initialization: The jspInit() method is called once to initialize the servlet instance. This is typically used for one-time setup tasks like creating a database connection pool.
6. Request Processing (Execution): For each client request, the _jspService() method is invoked. This method handles the request and generates the dynamic content to be sent back to the client.
7. Destruction: When the servlet is removed from service (e.g., when the server shuts down), the jspDestroy() method is called once for cleanup tasks, such as releasing resources.

In JavaServer Pages (JSP), special tags and built-in objects are used to embed dynamic content and handle page logic.

Directives

Directives provide global information to the JSP container during the translation phase, influencing how the JSP page is compiled into a servlet.

- Syntax: <%@ directive attribute="value" %>
- page directive: Sets page-specific attributes.
    - <%@ page import="java.util.Date" %>: Imports Java classes.
    - <%@ page errorPage="error.jsp" %>: Specifies an error page for unhandled exceptions.
- include directive: Statically includes the content of another file during translation. It's best for reusable, static content like headers and footers.
    - <%@ include file="header.html" %>

- taglib directive: Declares a custom tag library for use in the JSP page.
    - <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

Declarations

Declarations are used to define class-level variables and methods that will be available to all parts of the JSP page.

- Syntax: <%! declaration %>
- The code within a declaration is placed outside the _jspService() method in the generated servlet, making it a member of the class.
- Best for: Defining helper methods or instance variables that need to be shared across requests.
- Example:

jsp

```
<%!
  private int hitCount = 0;
  public String getPageTitle() {
    return "My Dynamic Page";
  }
%>
```

Expressions

Expressions are used to embed a Java expression's value directly into the HTML output.

- Syntax: <%= expression %>
- The expression is automatically converted into a string and inserted into the page's output stream. No semicolon is needed.
- Best for: Displaying dynamic data inline, such as the value of a variable or a method's return value.
- Example:

jsp

```
<p>The page title is: <%= getPageTitle() %></p>
<p>The current time is: <%= new java.util.Date() %></p>
```

Scriptlets (code snippets)

Scriptlets allow any valid Java code, including control-flow structures, to be embedded into the JSP page.

- Syntax: <% code %>
- The code within a scriptlet is placed inside the _jspService() method of the generated servlet.
- Best for: Implementing control-flow logic, such as loops and conditional statements.
- Example:

jsp

```
<% for (int i = 1; i <= 5; i++) { %>
  <p>This is paragraph #<%= i %></p>
<% } %>
```

Implicit objects:

Implicit objects are built-in Java objects that the JSP container automatically makes available on every JSP page. They can be used directly within scriptlets and expressions without explicit declaration.

Implicit Object	Type	Description
request	HttpServletRequest	Represents the client's HTTP request. Used to get parameters, headers, and cookies.
response	HttpServletResponse	Represents the HTTP response sent back to the client. Used to set headers, cookies, or redirect.
session	HttpSession	Represents the user's session across multiple requests. Used to store user-specific data.

application	ServletContext	Represents the web application context. Used to store application-wide data.
out	JspWriter	Used to write content to the client's output stream.
config	ServletConfig	Provides configuration information for the servlet.
pageContext	PageContext	Provides access to all namespaces (scopes) of the JSP page.
page	java.lang.Object	A synonym for this, referring to the current servlet instance.
exception	java.lang.Throwable	Represents an uncaught exception in an error page (isErrorPage="true").

Example using implicit objects

jsp

```
<p>User-Agent: <%= request.getHeader("User-Agent") %></p>
<%
  String username = (String) session.getAttribute("username");
  if (username != null) {
    out.println("<p>Welcome back, " + username + "!</p>");
  }
%>
```

BEANS:

Using Java Beans in JSP pages is a way to separate business logic from presentation, promoting a clean, reusable, and maintainable design. A JavaBean is a standard Java class that follows specific design conventions, which makes it easy to manipulate within JSP using special action tags.

```java
package javaskool;
import java.io.*;
import java.util.*;
import java.sql.*;

public class Customer implements Serializable{

        private String custID;
        private String custName;
        private int qty;
        private float price;
        private float total;
        private int storeCust;

        public String getCustID() {
            return custID;
        }

        public void setCustID(String custID) {
            this.custID = custID;
        }

        public String getCustName() {
            return custName;
        }

        public void setCustName(String custName) {
            this.custName = custName;
        }

        public int getQty() {
            return qty;
        }
```


Getter or Accessor

Setter or Mutator

Key JavaBean conventions

A Java class qualifies as a JavaBean if it adheres to these rules:

- It must have a public no-argument constructor, allowing the JSP container to instantiate it easily.
- Its properties (instance variables) should be private.
- It must provide public getter (getXxx) and setter (setXxx) methods to access and modify its private properties. For example, for a name property, there should be getName() and setName() methods.
- It should be serializable, enabling its state to be saved and restored, especially in distributed applications.

JSP action tags for using beans

JSP provides three main action tags to interact with JavaBeans.

1. <jsp:useBean>

This tag is used to find or create an instance of a JavaBean and assign it an ID. If a bean with the specified ID and scope already exists, it is used; otherwise, a new instance is created.

Syntax:

xml

<jsp:useBean id="beanName" class="package.ClassName" scope="..."/>

Use code with caution.

- id: A variable name used to reference the bean object in the JSP.
- class: The fully qualified name of the JavaBean class.
- scope: Defines the bean's lifecycle and visibility. The possible values are:
    - page (default): The bean is available only on the current page.
    - request: The bean is available for the duration of the current HTTP request.
    - session: The bean is available throughout the user's session.
    - application: The bean is shared across the entire web application and all users.

2. <jsp:setProperty>

This tag sets the values of a bean's properties by calling its setter methods.

Syntax:

xml

<jsp:setProperty name="beanName" property="propertyName" value="value"/>

Use code with caution.

or, to automatically match request parameters to bean properties:

xml

<jsp:setProperty name="beanName" property="*"/>

Use code with caution.

- name: The ID of the bean set with <jsp:useBean>.
- property: The name of the property to set.

- value: The value to assign to the property.
- param: Sets a property value from a request parameter.

3. <jsp:getProperty>

This tag retrieves the value of a bean's property by calling its getter method and inserts the result into the output.

Syntax:

xml

<jsp:getProperty name="beanName" property="propertyName"/>

Use code with caution.

- name: The ID of the bean.
- property: The name of the property to retrieve.

Example: Using a user bean

This example demonstrates a typical use case for beans in a web application, where a bean holds data submitted from a form.

Step 1: Create the JavaBean class (User.java)
This class encapsulates user data with private fields and public getter and setter methods.

java

```
package com.example;


import java.io.Serializable;


public class User implements Serializable {
   private String firstName;
   private String lastName;


   // No-argument constructor
   public User() {}
```

```
    // Getters and Setters
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

Step 2: Create the JSP page (process.jsp)
This page processes the form data, sets the bean's properties, and displays the results.

jsp

```
<%@ page import="com.example.User" %>

<jsp:useBean id="userBean" class="com.example.User" scope="request"/>

<%-- Automatically maps form parameters to bean properties --%>
<jsp:setProperty name="userBean" property="*"/>
```

```
<!DOCTYPE html>
<html>
<head>
  <title>JSP Bean Example</title>
</head>
<body>
  <h1>User Information</h1>
  <p>First Name: <jsp:getProperty name="userBean" property="firstName"/></p>
  <p>Last Name: <jsp:getProperty name="userBean" property="lastName"/></p>
</body>
</html>
```

Use code with caution.

Step 3: Create the HTML form (index.html)
This form sends data to the process.jsp page.

html

```
<!DOCTYPE html>
<html>
<head>
  <title>User Form</title>
</head>
<body>
  <form action="process.jsp" method="post">
    <label>First Name:</label>
    <input type="text" name="firstName"><br><br>
    <label>Last Name:</label>
    <input type="text" name="lastName"><br><br>
    <input type="submit" value="Submit">
  </form>
```

</body>

</html>

Stateless Web Applications:

- **Definition:** Each request from a client to the server is treated as an independent and self-contained transaction. The server does not store any information about previous requests from the same client.
- **Characteristics:**
    - **No server-side session data:** The server does not maintain user session information or "state" between requests.
    - **Each request contains all necessary information:** Any data needed to process a request must be included within that request itself (e.g., in headers, parameters, or the request body).
    - **Scalability:** Highly scalable as any server can handle any request, and there's no need to manage persistent connections or shared state across servers.
    - **Resilience:** More resilient to server failures, as there's no state to lose or restore.
- **Examples:** RESTful APIs, static content servers (like CDNs), many microservices.

Stateful Web Applications:

- **Definition:**

The server maintains information about the client's session or "state" across multiple requests. Subsequent requests from the same client can leverage this stored information for context and continuity.

- **Characteristics:**
    - **Server-side session management:** The server stores and manages user session data (e.g., user login status, shopping cart contents, user preferences).
    - **Dependency on previous interactions:** Requests often depend on the context established by prior interactions within the same session.
    - **Less scalable (potentially):** Can be more challenging to scale horizontally, as session data needs to be managed and potentially shared across multiple servers.

- **Less resilient (potentially):** Server failures can lead to loss of session data, requiring mechanisms for session persistence or recovery.

- **Examples:**

Traditional web applications with server-side sessions (e.g., using Java Servlets, ASP.NET Web Forms), online shopping carts (where items are stored on the server), and applications requiring continuous user interaction.

Session Tracking:

A web session is the conversation between a user and a server over a period of time. Session tracking is the process of maintaining a user's state across multiple requests within that session. This is essential because the HTTP protocol is stateless, meaning each request from a client is independent and not inherently connected to previous ones. The Java Servlet API provides the Http Session interface to facilitate session tracking, with each JSP page having implicit access to the session object.

In JSP, both Cookies and the Http Session object are used for session tracking to maintain state across multiple user requests.

How JSP session tracking works with HTTP Session :

1. Session creation: When a user's browser makes its first request, the servlet container creates a unique Http Session object for that user. A unique session ID is generated and sent back to the client, typically stored as a cookie in the browser. On subsequent requests, the browser sends this ID back, allowing the server to identify the user's session.
2. State management: The Http Session object acts as a server-side storage for user-specific data, such as a username after login or a shopping cart for an e-commerce site. This state information is securely stored on the server and is not visible to the client, unlike cookies.
3. Session access: In JSP pages, the session object is one of the implicit objects, meaning it's readily available to use without manual instantiation. You can use its methods within scriptlets or leverage the Expression Language (EL) for easier access.

Key HttpSession methods

The session object provides several methods for managing session data:

- session.setAttribute(String name, Object value): Binds an object to the session with a specific name.

- session.getAttribute(String name): Retrieves the object bound to the session with the specified name. The return type is Object, so you must cast it to the correct type.
- session.removeAttribute(String name): Removes the object bound with the specified name from the session.
- session.invalidate(): Destroys the entire session and unbinds all objects. This is typically used for logging out a user.
- session.isNew(): Returns true if the client doesn't know about the session yet or has chosen not to join it.
- session.getId(): Returns the unique session ID.

Example:

```
<%
session.setAttribute("userEmail", "john.doe@example.com");
%>
```

To retrieve :

```
<%
String userEmail = (String) session.getAttribute("userEmail");
if (userEmail != null) {
  out.println("Your email: " + userEmail);
} else {
  out.println("Email not found in session.");
}
%>
```

Using Cookies for Session Tracking:

Cookies are small pieces of data sent from a website and stored in the user's web browser while the user is browsing that website. They can be used to store a session ID or other user-specific information directly on the client-side.

Creating and Adding a Cookie:

```
<%
```

```
Cookie userCookie = new Cookie("username", "JohnDoe");
userCookie.setMaxAge(60 * 60 * 24); // Set cookie to expire in 24 hours
response.addCookie(userCookie);
%>
```

To retrieve:

```
<%
String username = "Guest";
Cookie[] cookies = request.getCookies();
if (cookies != null) {
  for (Cookie cookie : cookies) {
    if (cookie.getName().equals("username")) {
      username = cookie.getValue();
      break;
    }
  }
}
out.println("Welcome, " + username);
%>
```

connecting to database in JSP

To connect to a database in a JSP page, you typically

use JDBC (Java Database Connectivity), which provides a standard API for Java applications to interact with relational databases. While you can embed JDBC code directly into a JSP, using separate Java classes (like JavaBeans) is a more maintainable and secure approach.

Prerequisites

1. Install a database server, such as MySQL, and create the database and table you need.

2. Download the JDBC driver for your database (e.g., mysql-connector-java.jar) and place the .jar file in your web application's WEB-INF/lib directory.
3. Ensure your web server (e.g., Apache Tomcat) has access to the driver by restarting it if necessary.

Method 1: Connecting directly with a scriptlet (for demonstration)

This method involves embedding all the JDBC code directly into your JSP page using a scriptlet. While simple for learning, this is not recommended for production applications due to security risks and code maintainability issues.

jsp

```
<%@ page import="java.sql.*" %>


<%!
  // Database credentials and driver setup
  private final String dbUrl = "jdbc:mysql://localhost:3306/your_database_name";
  private final String dbUser = "your_username";
  private final String dbPassword = "your_password";
  private final String driverClass = "com.mysql.cj.jdbc.Driver";

  // Static initialization block to load the driver once
  static {
    try {
      Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
%>

<html>
```

```
<head>
  <title>Database Connection Example</title>
</head>
<body>
  <%
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
      // Step 1: Establish the connection
      conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);

      // Step 2: Create a statement and execute a query
      stmt = conn.createStatement();
      String sql = "SELECT id, name, email FROM users";
      rs = stmt.executeQuery(sql);
  %>
  <h2>User List</h2>
  <table border="1">
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Email</th>
    </tr>
    <%
      // Step 3: Process the result set
      while (rs.next()) {
```

```
        %>
        <tr>
            <td><%= rs.getInt("id") %></td>
            <td><%= rs.getString("name") %></td>
            <td><%= rs.getString("email") %></td>
        </tr>
        <%
            }
        %>
    </table>
    <%
        } catch (SQLException se) {
            // Log and handle SQL errors
            out.println("<h3>Error connecting to database:</h3><pre>" + se.getMessage() +
"</pre>");
            se.printStackTrace();
        } finally {
            // Step 4: Close resources in a finally block
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close();
            } catch (SQLException se) {
                se.printStackTrace();
            }
        }
    %>
</body>
```

</html>

MODULE -V

Database Design using MySQL: An Overview of SQL:

Database design with MySQL involves using Structured Query Language (SQL) to create an efficient and organized structure for storing and managing data. A robust database design is essential for ensuring data integrity, minimizing redundancy, and optimizing performance.

Phases of database design

The process of designing a database is systematic and typically broken down into the following stages:

1. Requirements gathering: Understand the purpose of the database and the information that needs to be stored, such as details for products or user orders.
2. Conceptual design: Create a high-level, visual model of the database, often using an Entity-Relationship Diagram (ERD). This helps identify the key entities (tables), their attributes (columns), and the relationships between them.
3. Logical design: Translate the conceptual model into a logical structure, defining specific tables, columns, and keys. This is when normalization rules are applied to minimize data redundancy.
4. Physical design: Implement the logical design on a specific Database Management System (DBMS), such as MySQL. This involves selecting appropriate data types, defining indexes, and considering storage and performance.

Core concepts in database design

Effective database design utilizes several key concepts. Data is organized into tables with rows and columns. Keys, such as primary keys which uniquely identify rows, and foreign keys which link tables, are fundamental. Relationships between tables, like one-to-many and many-to-many, help manage connected data and reduce redundancy; many-to-many relationships require an intermediate table. Normalization is a technique using "normal forms" to minimize redundancy and enhance data integrity. Constraints, such as NOT NULL, UNIQUE, and CHECK, are rules applied to columns to maintain data quality. Columns are assigned data types like INT, VARCHAR, and DATE to ensure they store appropriate data.

Overview of SQL

SQL is the standard language for interacting with relational databases like MySQL. SQL commands are categorized by function:

Data Definition Language (DDL)

DDL commands manage database structures. They include CREATE DATABASE, CREATE TABLE, ALTER TABLE to modify structures, and DROP TABLE to delete tables.

Data Manipulation Language (DML)

DML commands are used to work with data within tables. Key commands are INSERT INTO to add data, UPDATE to modify records, and DELETE FROM to remove records.

Data Query Language (DQL)

DQL commands are for retrieving data. The main command is SELECT to retrieve data. Other important commands include WHERE to filter records, JOIN to combine data from multiple tables, ORDER BY to sort results, and GROUP BY to summarize data.

XAMPP and MySQL Setup:

XAMPP:

To set up XAMPP, first, download the installer from apachefriends.org, then run it as an administrator, following the on-screen prompts to select components and installation location. After the installation completes, launch the XAMPP Control Panel, click "Start" for the Apache and MySQL modules, and then open your web browser to http://localhost/ to view the XAMPP dashboard and confirm the installation.

1. Download XAMPP

- Go to the official Apache Friends website and download the installer for your operating system (e.g., Windows).

2. Run the Installer

- Locate the downloaded .exe file and run it.
- You may see a warning about User Account Control (UAC); click "OK" or "Yes" to proceed.
- If you have antivirus software, it's recommended to temporarily deactivate it during installation to prevent interference.

3. Follow the Setup Wizard

- In the setup window, click "Next".
- Choose the default components, which include Apache, MariaDB (for MySQL), PHP, and Perl, or select the ones you need.

- Accept the default installation folder (e.g., C:\xampp), as installing in C:\Program Files can cause issues with User Account Control.
- Uncheck the "Learn more about Bitnami" option if you don't want to install additional features.
- Complete the installation by clicking "Next" and then "Finish".

4. Start the XAMPP Control Panel

- After installation, open the XAMPP Control Panel. You can find it in the Start menu.
- Click the "Start" button for the Apache and MySQL modules.
- You may need to allow Apache access to your network via the Windows firewall.

5. Access the Localhost Dashboard

- Open your web browser.
- Type http://localhost/ into the address bar.
- If you see the XAMPP dashboard, the installation was successful. From here, you can access modules like PHPMyAdmin for managing databases.

MY SQL:

Setting up MySQL typically involves downloading the appropriate installer for your operating system and then following the installation and configuration steps.

1. Download the MySQL Installer:

- Navigate to the official MySQL website.
- Locate the "Downloads" section and choose "MySQL Community Downloads."
- Select the "MySQL Installer for Windows" (or the appropriate version for macOS/Linux).
- Choose the larger "Community" MSI file for a comprehensive installation.

2. Run the Installer and Choose Setup Type:

- Execute the downloaded installer file.
- When prompted, select a setup type. "Developer Default" or "Full" are common choices that include the MySQL server and development tools like MySQL Workbench.
- Proceed through any requirement checks, allowing the installer to resolve dependencies if necessary.

3. Installation and Configuration:

- Click "Execute" to begin the installation of the selected components.
- Once installed, the installer will guide you through the configuration process.
- **Server Configuration Type:** Choose a configuration type (e.g., Development Machine).
- **Authentication Method:** Select a strong password encryption method.
- **Root Account Password:** Set a secure password for the 'root' user. This is crucial for administrative access.
- **Windows Service (Windows only):** Choose whether to run MySQL as a Windows service for automatic startup.
- **Apply Configuration:** Execute the configuration steps to finalize the setup.

4. Verification:

- After installation and configuration, you can verify the setup.
- **MySQL Command Line Client:** Open the MySQL Command Line Client (or terminal/command prompt) and log in with the 'root' user and the password you set.
- **MySQL Workbench:** If installed, launch MySQL Workbench and establish a connection to your local MySQL server using the 'root' user and password.

This process establishes a functional MySQL server on your system, ready for database creation and management.

After your MySQL environment is set up, you can write your SQL program. Below is the example to display " Hello World" using SQL.

1. Create a database named test_db

```
CREATE DATABASE test_db;
```

2. Use the test_db database

```
USE test_db;
```

3. Create a table named greetings

```
CREATE TABLE greetings (
   id INT AUTO_INCREMENT PRIMARY KEY,
   message VARCHAR(255)
);
```

3. Insert the message 'Hello, World!' into the greetings table

```
INSERT INTO greetings (message)
VALUES ('Hello, World!');
```

4. Retrieve the message from the greetings table

SELECT message FROM greetings;

Aliases & CONCAT()

Aliases are temporary names given to columns or tables to make queries more readable and concise. They exist only for the duration of the query. The CONCAT() function is used to join or concatenate multiple strings together into a single string. You can use an alias to give a new, meaningful name to a column created with CONCAT().

Syntax

sql

-- Column alias with CONCAT()

SELECT CONCAT(column1, ' ', column2) AS alias_name

FROM table_name;

-- Table alias

SELECT alias_name.column1, alias_name2.column2

FROM table_name AS alias_name, table_name2 AS alias_name2;

Example

To display a full name by concatenating first_name and last_name columns from an employees table:

sql

SELECT CONCAT(first_name, ' ', last_name) AS full_name

FROM employees;

UPDATE, DELETE, & ALTER

These are key Data Manipulation Language (DML) and Data Definition Language (DDL) commands used to modify data and table structures in a database.

- UPDATE: Modifies existing records in a table.
    - Syntax: UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
    - Example: UPDATE employees SET email = 'john.doe@example.com' WHERE employee_id = 1;
- DELETE: Removes rows from a table.

    - Syntax: DELETE FROM table_name WHERE condition;
    - Example: DELETE FROM employees WHERE employee_id = 5;
- ALTER TABLE: Modifies the structure of an existing table.
    - Syntax for adding a column: ALTER TABLE table_name ADD COLUMN new_column_name datatype;
    - Syntax for modifying a column: ALTER TABLE table_name MODIFY COLUMN column_name new_datatype;
    - Syntax for dropping a column: ALTER TABLE table_name DROP COLUMN column_name;
    - Example: ALTER TABLE employees ADD COLUMN start_date DATE;

Foreign keys

A foreign key is a field in one table that references the primary key in another table. This establishes a parent-child relationship between tables and maintains referential integrity, preventing actions that would break the link between them. An example is an orders table referencing the customer_id from a customers table. You can find the SQL code for this example in the referenced documents.

Table joins

Table joins combine rows from two or more tables based on a related column. Common types of joins include:

- INNER JOIN: Returns only rows with matching values in both tables.
- LEFT JOIN: Returns all rows from the left table and matched rows from the right.
- RIGHT JOIN: Returns all rows from the right table and matched rows from the left.
- FULL OUTER JOIN: Returns all rows when there is a match in either table. While MySQL doesn't have a direct FULL JOIN, you can achieve a similar result using a UNION of LEFT JOIN and RIGHT JOIN.
- SELF JOIN: Joins a table to itself using aliases.

PHP Programming:

PHP programming fundamentals

PHP is a widely used server-side scripting language for web development. PHP code can be embedded directly into HTML files and is executed on a web server, producing dynamic content.

Basic syntax and variables

- Tags: PHP code is enclosed within <?php and ?> tags.

- Case sensitivity: Variable names are case-sensitive. $name and $Name are different variables.
- Statements: Each statement in PHP must end with a semicolon (;).
- Variables: Variable names must start with a dollar sign ($), followed by a letter or underscore, and are assigned values using the equals sign (=).
- Comments: Single-line comments start with //, while multi-line comments are enclosed between /* and */.

Control structures

- if...else...elseif: Executes different blocks of code based on a condition.
- switch: Compares a single expression against different possible values.
- for loop: Repeats a block of code a fixed number of times.
- while loop: Executes a block of code as long as a condition is true.
- foreach loop: Iterates over elements in an array.

Functions

- User-defined functions: Declared with the function keyword, they encapsulate reusable blocks of code.
- Parameters: Functions can accept inputs, either by value or by reference.
- Built-in functions: PHP includes a vast library of built-in functions for tasks like string manipulation, file handling, and date/time operations.

PHP data types & dates

PHP is a loosely typed language, meaning you do not have to specify the data type of a variable. PHP automatically determines the type based on the value assigned.

Data types

PHP supports several data types, including String (character sequences in quotes), Integer (whole numbers), Float (numbers with decimals), and Boolean (true or false). It also includes Array (stores multiple values), Object (instances of classes), NULL (single NULL value), and Resource (reference to external resources like files or database connections).

Dates and time

PHP offers functions for handling dates and times:

- date(format, timestamp): Formats a local date/time. Common format characters are available in the source documents.
- time(): Returns the current Unix timestamp.

- strtotime(string): Converts a human-readable date string to a timestamp.
- date_default_timezone_set(timezone): Sets the default timezone.

Cookies:

Cookies are small data pieces a web server stores on a user's browser to identify users and save information like preferences.