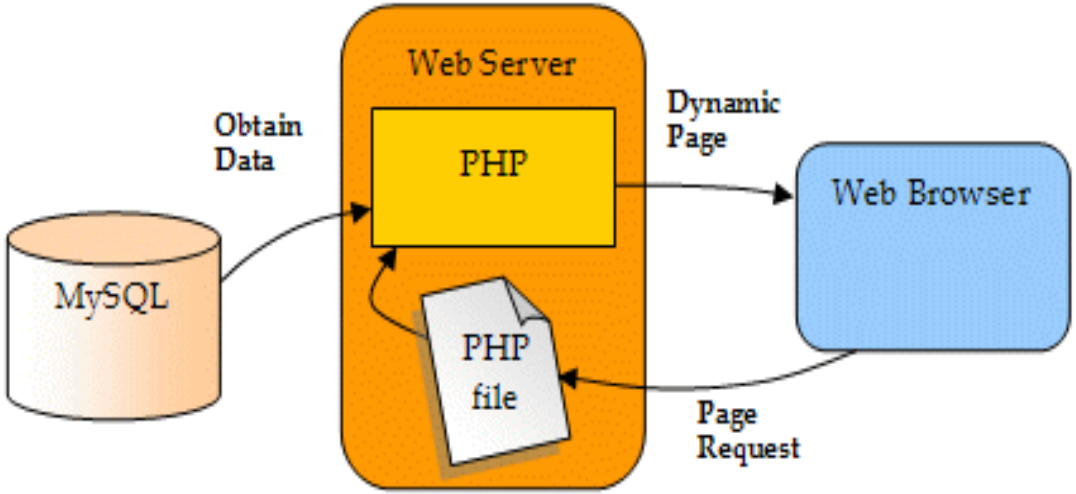
INTRODUCTION

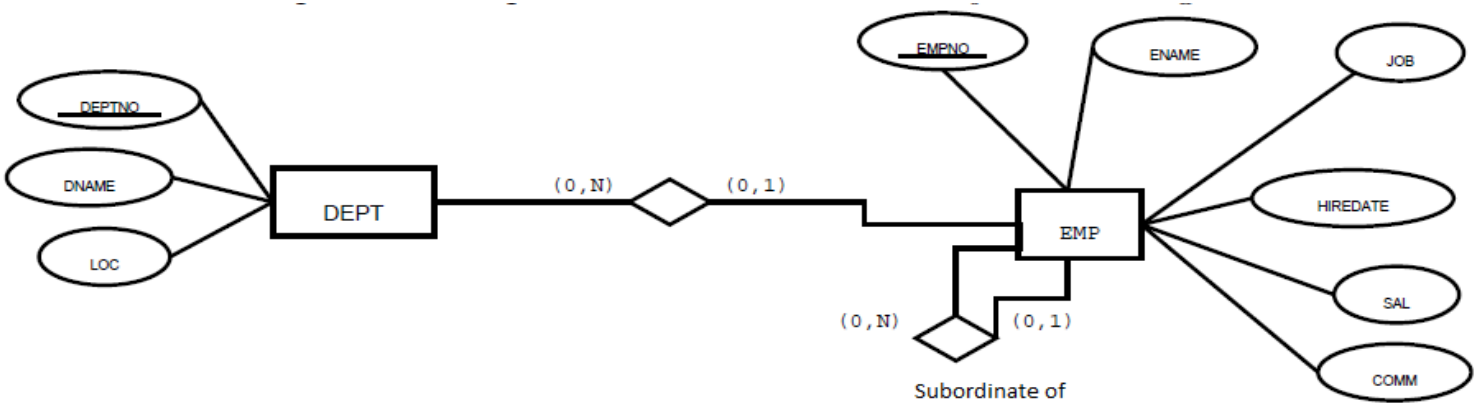
The PHP scripting language processes the page request and fetches the data from the MySQL database.
It then dynamically returns the formatted HTML page that the browser expects.



PREPARING THE DATABASES: TABLES, USERS AND PERMISSIONS

For security reasons it is better not to use the root user. You should create a new user account with only the specific privileges it needs to work on the database that your website depends upon. To do it. Go to phpmyadmin:localhost/phpmyadmin and follow the steps in create_tables.pdf

Entity Relationship Model in the image: database "ies", two tables: dept and emp



CONNECTING METHODS

There are three methods of connecting to a MySQL Server from PHP:

- **MySQL library:** oldest method. It has been removed from PHP entirely since PHP 7.0.Therefore, we will not study it but it is very similar to the procedural style mysqli
- **MySQLi library:** only mysql
- **PDO library:** many databases

There are a few differences between PDO and MySQLi, but the main one is that you can use the PDO library to connect to almost any database server—such as an Oracle server, or a Microsoft SQL Server. For this reason, most recent PHP projects use the PDO library.

However we will explain the two ways: MySQLi and PDO

MySQLi LIBRARY CONNECTION

Supports both: procedural and object-oriented language.

Object-oriented is better than procedural:

- Some Mysqli Functions are in the aliases and deprecated section of https://www.php.net/manual/en/ref.mysqli.php

- The main reason is that PHP is moving steadily in the direction of OO programming.
It's worth noting that the PDO library, which is considered the ideal for most DB code in PHP is OOP-only. It doesn't have a procedural interface.

There's also the point about the ability to create an extension class for your DB -- something like this:

```
class myDB extends mysqli
{
    .... your own stuff here to extend and improve the base mysqli class
}
```

So, we recommend you use the MySQLi OOP Interface

MySQLi PROCEDURAL LANGUAGE

However, if you want to know the procedural mysqli library, you establish the connection using the mysqli_connect() function:

```
$conn = mysqli_connect(host, user, password, database, port, socket);
```

You don't need to include the port or socket values

```
<?php
$servername = "localhost";
$username = "dwes";
$password = "2DAWdwes";
$dbname = "ies";

// Create the connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Close the connection:
mysqli_close($conn);
```

More information in: https://www.php.net/manual/en/mysqli.summary.php
Study these functions. Then we will see them with oo:

```
$result = mysqli_query($conn,$sql);
$num= mysqli_num_rows($result)
$fila = mysqli_fetch_array($result);
```

MySQLi OBJECT-ORIENTED LANGUAGE

In this case we work with mysqli class and their methods

```
// Create the connection
$db = new mysqli($servername, $username, $password, $dbname);

/* Close the connection... it's not necessary, but you have that option
available if you need it!*/
$db->close();
```

Good practice (in both: procedural and object-oriented language): create a single file for store the data of login and password and include them wherever they are needed

login.php

```
<?php
$servername = "localhost";
$username = "dwes";
$password = "2DAWdwes";
$dbname = "ies";
?>
```

MySQLi OBJECT-ORIENTED LANGUAGE

```php
<?php
require_once 'login.php';

// Create the connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check it
if ($conn->connect_error) {
    die("Fatal Error: " . $conn->connect_error);
}
echo "Connected to db";
?>
```

die: Output a message and terminate the current script

MySQLi OO: SELECT

Once connected, you use the query() method of the mysqli connection instance to submit SQL statements:

```
$query = 'SELECT * FROM emp';
$result = $conn->query($query);
```

Methods with Object 'result' from a query:

fetch_array() method: Fetch the next row of a result set as an associative, a numeric array, or both based on the second parameter (MYSQLI_ASSOC, MYSQLI_NUM, or MYSQLI_BOTH).

fetch_assoc() method: Fetch the next row of a result set as an associative array, using the data field names as the array keys.

fetch_row() method: Fetch the next row of a result set as an enumerated array, using numeric indexes for each data field (starting at 0, and using the data field order specified in the table or SELECT statement data field).

close(): It is not necessary to free memory because php frees it at the end of the script. But in cases of high traffic it can be useful

MySQLi OO: SELECT EXAMPLE

Execute this script, with dbname ies

```php
<?php
require_once 'login.php';
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
    die("Fatal Error: " . $conn->connect_error);
}

$query = 'SELECT * FROM dept';
$result = $conn->query($query);
if (!$result) die("Fatal Error");

$rows = $result->num_rows; // number of rows
for ($j = 0 ; $j < $rows ; ++$j)
{
$row = $result->fetch_assoc();
echo 'Departments: '. htmlspecialchars($row['dname']) . '<br>';
}
$result->close();
$conn->close();
?>
```

SECURITY CONCERNS: SQL INJECTION ATTACK

You can submit any type of SQL statements using query() method, but it's not recommended when there are user input parameters

```
$conn = new mysqli($servername, $username, $password, $dbname);
$sql="DELETE FROM dept WHERE dname= ".$_POST['dname'];
$result = $conn->query($sql);
if (!$result) die ("Database access failed");
```

There is a serious problem with this code. It is one of the most common security holes that hackers find. This type of attack is called an SQL injection attack.

Imagine if the user type this into the input field of the form:

```
"; DELETE FROM dept; -
```

The safer way of submitting data in a statement is to use a prepared statement, which defines a template of the query you want to execute on the MySQL server, and then sends the data separate from the template.

MySQLi OO PREPARED STATEMENT

With a prepared statement, you create the query string as normal, but instead of including data values, you use a question mark as a placeholder for each value, like this:

```
$sql = "INSERT INTO emp VALUES (?, ?, ?, ?, ?, ?)";
```

Then you use the prepare() method to submit it:

```
$stmt = $conn->prepare($sql);
```

And bind_param to pass the values.

```
$stmt->bind_param("isssi", $empid, $lname, $fname, $start, $birth, $salary);
```

The first parameter defines the data type of each of the data Values:

b: A blob data type value
i: An integer data type value
d: A double data type value
s: A string data type value

And finally, you must execute the prepared statement:

```
$stmt->execute();
```

MySQLi OO PREPARED STATEMENT

You must always use prepared statements for any SQL query that would contain a PHP variable (insert, update and delete), even with select

```
$sql = "SELECT * FROM emp WHERE empno=?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("i", $id);
$stmt->execute();
$result = $stmt->get_result();
while ($row = $result->fetch_assoc())
{    echo $row['ename'];  }
```

Good practice: Separate the database interaction from the HTML output.

Instead of while loop: It could be just a single line with fetch_all: returns an array from the query result:

```
$data = $result->fetch_all(MYSQLI_ASSOC)
```

We should create and include another php script with the output

```
foreach($data as $row)
   echo $row['name'];
```

MySQLi MULTIPLE QUERIES: MULTI_QUERY

Executes one or multiple queries which are concatenated by a semicolon.

```
$sql="INSERT INTO dept VALUES(50,'RRHH1','VALENCIA');";
$sql .="INSERT INTO dept VALUES(60,'RRHH2','VALENCIA');";
$sql .="INSERT INTO dept VALUES(70,'RRHH3','MADRID')";

if ($conn->multi_query($sql)) { echo "Right"; }
 else { echo  "INSERT Failed"; }
```

Remember SQL injection and if the query contains any variable input then use parameterized prepared statements

Alternatively, the data must be properly formatted and all strings must be escaped using the real_escape_string() method.

HANDLE DATES

CREATE A DATE

```
$date = new DateTime(); //  If you don't give a date, it uses the current date/time
$date = new DateTime('2023-01-01');
$date = new DateTime('5th March 2023');
$date = new DateTime($_POST['date']); // from a FORM
$date = DateTime::createFromFormat('j-M-Y', '15-Feb-2023');
```

PRINT A DATE

format METHOD: Returns date formatted according to given format.

```
echo $date->format('d/m/Y H:i:s'); // 08/07/2023 19:12:34
echo $date->format('Y-m-d H:i:s'); // 2023-07-08 19:12:34
```

INSERT INTO THE DB

Dates and times in MySQL are always stored using the format YYYY-MM-DD

So you have to use: `format('Y-m-d H:i:s')`

IF YOU CREATE THE DATE IN SPANISH FORMAT: YOU HAVE TO CONVERT TO ENGLISH FORMAT

```
$date = DateTime::createFromFormat('d-m-Y', $birthday)->format('Y-m-d');

$query = "INSERT INTO users(id, name, birthday) VALUES('$id','$name','$date')";

SELECT DATE_FORMAT(birthday, '%W %M %D %Y') from user;  Saturday August 12th 2023
```

HANDLE PASSWORDS

In the form: input the password

```
<input type="password" name="txtpassword">
```

Php page:Creates a password hash and store in the db

```
$pass = $_POST["txtpassword"];
$strongpass = password_hash($pass, PASSWORD_DEFAULT);
$sql = "INSERT INTO login(usu, pass) values ('$name','$strongpass')";
```

Verify the password

```
 $sql = "SELECT pass FROM login WHERE usu='John'";
 $result = $conn->query($sql);
 if (!$result) die("Fatal Error");
 $row = $result->fetch_assoc();
 if (password_verify($pass, $row['pass'])) echo 'Correct Password';
```