Database access from PHP (MySQL I)

DWES UD4.1

1.- Database access from PHP

The vast majority of web applications are dynamic.

The term *dynamic* refers to the fact that the pages change depending on several factors:

- Time at which it is accessed
- Be registered/logged in
- Data that is consulted
- ...

The power of dynamic web applications lies largely in the access to a database that changes, so that, depending on the factors described above, different content will be displayed on the web.

1.- Database access from PHP

PHP supports more than 15 DBMS (**D**ata**b**ase **M**anagement **S**ystems).

Historically, access to the database was done through specific **native extensions** for each DBMS.

This meant that if you needed to work with a PostgreSQL database, this extension had to be installed on the server.

Each extension had its functions and objects, so there was no compatibility between extensions.

1.- Database access from PHP

Starting with PHP 5, an extension was introduced that allows access to different DBMSs in the same way.

This extension is **PDO** (**P**HP **D**ata **O**bjects), and thanks to it the same syntax can be used even if the DBMS of the web application is changed.

Nowadays the choice of a native extension over PDO will depend on the needs of the application:

- Native extensions offer more power.
- Native extensions offer, in some cases, more speed.
- PDO offers a common set of functions.
- With PDO, you can change the DBMS without changing the application.

1.- Database access from PHP

In this module you will learn how to use MySQL/MariaDB:

- It's open source (GNU GPL License).
- It's the M from AMP, XAMPP, WAMPP, LAMPP MAMPP.
- It is the most used DBMS with PHP.

The use of both, the native **MySQLi** extension and **PDO** will be covered. With both you can perform actions such as:

- Establish connections.
- Execute SQL statements.
- Get the records returned by an SQL statement.
- Employ transactions.
- Run stored procedures.
- Manage errors that occur during the connection or during its establishment.

2.- MySQL

MySQL is a relational database management system (DBMS).

It is an open source program that is offered under the GNU GPL license, although it also offers a commercial license.

When Sun Microsystems purchased MySQL, the MariaDB fork was created to ensure its continuity as open source.

MySQL is used in multiple web applications, linked in most cases to the PHP language and the Apache web server

Documentation for the latest version as of today is available here:
https://dev.mysql.com/doc/refman/8.4/en/

2.- MySQL

MySQL has several **storage engines**, each with its own characteristics: some are faster, others provide greater security or better search capabilities.

When creating a table you can choose the most suitable storage engine.

The default engine if nothing is indicated is InnoDB, it provides referential integrity and transactions.

Other motors:

- MyISAM: Very fast, does not offer referential integrity or transactions.
- Memory: Creates tables whose content is stored in memory.
- ...

2.- MySQL

Character set vs Collation

A **character set** is a set of symbols and encodings. A **collation** is a set of rules for comparing characters in a character set. To understand it better, review the documentation.

The recommended character set is *utf8mb4*
The recommended collation is *utf8mb4_0900_ai_ci*

```
mysql> SHOW COLLATION WHERE Charset = 'utf8mb4';
+----------------------------+---------+-----+---------+----------+---------+---------------+
| Collation                  | Charset | Id  | Default | Compiled | Sortlen | Pad_attribute |
+----------------------------+---------+-----+---------+----------+---------+---------------+
| utf8mb4_0900_ai_ci         | utf8mb4 | 255 | Yes     | Yes      |       0 | NO PAD        |
| utf8mb4_0900_as_ci         | utf8mb4 | 305 |         | Yes      |       0 | NO PAD        |
| utf8mb4_0900_as_cs         | utf8mb4 | 278 |         | Yes      |       0 | NO PAD        |
| utf8mb4_0900_bin           | utf8mb4 | 309 |         | Yes      |       1 | NO PAD        |
| utf8mb4_bin                | utf8mb4 |  46 |         | Yes      |       1 | PAD SPACE     |
| utf8mb4_croatian_ci        | utf8mb4 | 245 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_cs_0900_ai_ci      | utf8mb4 | 266 |         | Yes      |       0 | NO PAD        |
| utf8mb4_cs_0900_as_cs      | utf8mb4 | 289 |         | Yes      |       0 | NO PAD        |
| utf8mb4_czech_ci           | utf8mb4 | 234 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_danish_ci          | utf8mb4 | 235 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_da_0900_ai_ci      | utf8mb4 | 267 |         | Yes      |       0 | NO PAD        |
| utf8mb4_da_0900_as_cs      | utf8mb4 | 290 |         | Yes      |       0 | NO PAD        |
| utf8mb4_de_pb_0900_ai_ci   | utf8mb4 | 256 |         | Yes      |       0 | NO PAD        |
| utf8mb4_de_pb_0900_as_cs   | utf8mb4 | 279 |         | Yes      |       0 | NO PAD        |
| utf8mb4_eo_0900_ai_ci      | utf8mb4 | 273 |         | Yes      |       0 | NO PAD        |
| utf8mb4_eo_0900_as_cs      | utf8mb4 | 296 |         | Yes      |       0 | NO PAD        |
| utf8mb4_esperanto_ci       | utf8mb4 | 241 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_estonian_ci        | utf8mb4 | 230 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_es_0900_ai_ci      | utf8mb4 | 263 |         | Yes      |       0 | NO PAD        |
| utf8mb4_es_0900_as_cs      | utf8mb4 | 286 |         | Yes      |       0 | NO PAD        |
| utf8mb4_es_trad_0900_ai_ci | utf8mb4 | 270 |         | Yes      |       0 | NO PAD        |
| utf8mb4_es_trad_0900_as_cs | utf8mb4 | 293 |         | Yes      |       0 | NO PAD        |
| utf8mb4_et_0900_ai_ci      | utf8mb4 | 262 |         | Yes      |       0 | NO PAD        |
| utf8mb4_et_0900_as_cs      | utf8mb4 | 285 |         | Yes      |       0 | NO PAD        |
| utf8mb4_general_ci         | utf8mb4 |  45 |         | Yes      |       1 | PAD SPACE     |
| utf8mb4_german2_ci         | utf8mb4 | 244 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_hr_0900_ai_ci      | utf8mb4 | 275 |         | Yes      |       0 | NO PAD        |
| utf8mb4_hr_0900_as_cs      | utf8mb4 | 298 |         | Yes      |       0 | NO PAD        |
| utf8mb4_hungarian_ci       | utf8mb4 | 242 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_hu_0900_ai_ci      | utf8mb4 | 274 |         | Yes      |       0 | NO PAD        |
```

2.- MySQL

phpMyAdmin

It is a web administration tool for the MySQL DBMS.

It is included in XAMPP and similar.

http://localhost/phpmyadmin

Allows to:

- create databases, tables and relationships
- execute SQL statements
- manage users and their permissions
- ...

3.- MySQLi

Extension developed for PHP versions starting from 4.1.3.

It is included as of PHP 5.

Offers dual programming interface:

Use of functions:

```
$conexion = mysqli_connect('localhost', 'usuario', 'contraseña', 'base_de_datos');
echo mysqli_get_server_info($conexion);
```

Use of objects:

```
$conexion = new mysqli('localhost', 'usuario', 'contraseña', 'base_de_datos');
print conexion->server_info;
```

3.- MySQLi

Even though it is a dual programming interface, the **$connection** variable in both cases is an **object**.

Using the object notation of the **mysqli** class produces shorter and more readable code.

Some improvements introduced by the **mysqli** extension compared to the old mysql are:

- Object-oriented interface.
- Transaction support.
- Support for prepared queries.
- Better debugging options.

3.- MySQLi

In the **php.ini** file there is a specific section to configure **mysqli**.

Some of the options:

- mysqli.allow_persistent → Allows you to create persistent connections.
- mysqli.default_port → Default TCP port number.
- mysqli.reconnect → Indicates whether to automatically reconnect if the connection is lost.
- mysqli.default_host. → Default host (server).
- mysqli.default_user → Default username.
- mysqli.default_pw → Default password.

3.- MySQLi

Establishing connections

The first step in communicating from a **PHP script** to a MySQL server is to establish a connection.

All communications will be made from that connection.

Usually the same host has the web server and the database server installed, so the connection is internal (localhost or 127.0.0.1).

It could be the case that the servers were on different hosts so the IP of the database server would be needed.

3.- MySQLi

Establishing connections

Establishing a connection means creating an instance of the **mysqli** class.

The class constructor can receive 6 parameters although only the first 4 are usually used.

- The **hostname** or IP address of the MySQL server to connect to.
- A **username** with permissions to establish the connection.
- The user's **password**.
- The name of the **database** you want to connect to.
- The **port** number on which the MySQL server is running.
- The **socket** or named pipe to use.

Exercise

Dump the **tienda** database into the DBMS using phpMyAdmin

- First dump the structure:**crear_db_tienda.sql**
  The creation script also creates the user: **dwes** with password: **dwes**

- Then dump the data: **datos_tienda.sql**

4.- MySQLi

Establishing connections

Ways to connect to the **tienda** database

```
// using function calls
$dwes = mysqli_connect('localhost', 'dwes', 'dwes', 'tienda');


// using the class constructor
$dwes = new mysqli('localhost', 'dwes', 'dwes', 'tienda');

// Alternative method with the connect method
$dwes = new mysqli();
$dwes->connect('localhost', 'dwes', 'dwes', 'tienda');
```

4.- MySQLi

Establishing connections

It is important to check that the connection has been established before continuing. To do this you can use the mysqli properties:

- connect_errno → error number or null if no error occurs.
- connect_error → error message or null if no error occurs.

You can also use the functions:

- mysqli_connect_errno()
- mysqli_connect_error()

*From now on only the use of mysqli properties and methods will be shown. To see the use of functions, consult the official documentation.

4.- MySQLi

Establishing connections

Connection error handling example

```
@$dwes = new mysqli('localhost', 'dwes', 'dwes', 'tienda');
if ($dwes->connect_errno != null) {
	echo 'Error conectando a la base de datos: ';
	echo '$dwes->connect_error';
	exit();
}
```

With the **@** character, execution errors in PHP will not be displayed on the screen, thus avoiding showing information to the user.

4.- MySQLi

Database change

If you need to change the database on which to perform operations, you can use the following method that will return **true** or **false**.

The user with whom you established the first connection must have permissions on the new one.

```
$dwes->select_db('otra_base_de_datos');
```

4.- MySQLi

Queries

In SQL, queries can be grouped into two types:

- Control → they do not return data
  UPDATE, INSERT or DELETE

- Query → they return data
  SELECT

4.- MySQLi

Query execution

To execute queries with the MySQLi extension, use the **query** method:

```
$resultado = $dwes->query('DELETE FROM stock WHERE unidades=0;');
```

For UPDATE, INSERT or DELETE queries, **true** or **false** returns.

For SELECT queries, an object **mysqli_result** returns.

4.- MySQLi

Queries that DO NOT RETURN DATA → UPDATE, INSERT or DELETE

After executing a query of this type, the number of records affected by the query can be observed using the **affected_rows** property of the connection.

```
$resultado = $dwes->query('DELETE FROM stock WHERE unidades=0');

if ($resultado) {
                echo 'Se han borrado '. $dwes->affected_rows .' registros.';
    }
```

4.- MySQLi

Queries that DO RETURN DATA → SELECT

Just as in queries that do not return data, the query method is used to execute a **query**.

```
$resultado = $dwes->query(''SELECT producto, unidades FROM stock;');
```

In this case, if the query produces an error, it will return **FALSE**.
If the query is executed correctly it will return a **mysqli_result** object.

4.- MySQLi

Queries that DO RETURN DATA→ SELECT

mysqli_result

This object has different properties and methods that allow access to the data returned by the SELECT query.

```
$resultado->num_rows
$resultado->fetch_all()
$resultado->fetch_row()
$resultado->fetch_array()
$resultado->fetch_assoc()
...
```

4.- MySQLi

Queries that DO RETURN DATA → SELECT

The **query method** has two different behaviors, which reside in adding an optional parameter to the method (system constants):

- MYSQLI_STORE_RESULT (default option, similar to putting nothing):
  The data is recovered all together and stored locally.

```
$resultado = $dwes->query("SELECT producto, unidades FROM stock;');
```

- MYSQLI_USE_RESULT:
  data is retrieved from the server as needed.

```
$resultado = $dwes->query("SELECT producto, unidades FROM stock;', MYSQL_USE_RESULT);
```

4.- MySQLi

Queries that DO RETURN DATA → SELECT

Data obtained in a SELECT query will be stored in memory while it is in use.

Apache understands that the data is going to be used during the execution of the entire php script.

If several SELECT queries are made during a php script, memory can be wasted if they are not freed.

```
$resultado->free();
```

4.- MySQLi

Obtaining and using SELECT results.

Successfully executed SELECT query will return a **mysqli_result** object.

A SELECT query may not return values even if it executes successfully.

To check the number of records returned by a SELECT query, use the **num_rows** property:

```
if($resultado->num_rows == 0)
        echo 'La consulta no ha devuelto resultados.';
```

The following methods allow you to work with the results, if any.

4.- MySQLi

Obtaining and using SELECT results.

fetch_all: Fetch all result rows as an associative array, a numeric array, or both.

```
$stock = $resultado->fetch_all(MYSQLI_NUM);      //numeric array

$stock = $resultado->fetch_all(MYSQLI_ASSOC);   //associative array


$stock = $resultado->fetch_all(MYSQLI_BOTH);     //both
            // equals to: $resultado->fetch_all();
```

4.- MySQLi

Obtaining and using SELECT results.

fetch_array: obtains the first row returned and stores it in an array, by default it will contain both numeric and associative keys.

```
$stock = $resultado->fetch_array(MYSQLI_NUM);      //numeric array

$stock = $resultado->fetch_array(MYSQLI_ASSOC);   //associative array


$stock = $resultado->fetch_array(MYSQLI_BOTH);     //both
            // equals to: $resultado->fetch_array();
```

4.- MySQLi

Obtaining and using SELECT results.

fetch_array

```
$consulta = ''SELECT producto, unidades FROM stock WHERE unidades<2';
$resultado = $dwes->query($consulta);
$stock = $resultado->fetch_array();          // The first record is obtained
$producto = $stock['producto'];              // also $stock[0];
$unidades = $stock['unidades'];              // also $stock[1];
echo 'Producto '. $producto .'('. $unidades .' unidades)<br>';
```

4.- MySQLi

Obtaining and using SELECT results.
In the following examples, both instructions are equivalent

fetch_row: // Fetch the next row of a result set as an enumerated array
$stock = $resultado->fetch_row();
$stock = $resultado->fetch_array(MYSQLI_NUM);

fetch_assoc: // Fetch the next row of a result set as an associative array
$stock = $resultado->fetch_assoc()
$stock = $resultado->fetch_array(MYSQLI_ASSOC);

fetch_object: //Fetch the next row of a result set as an object
$stock = $resultado->fetch_object();

4.- MySQLi

Obtaining and using SELECT results.

```
$consulta = 'SELECT producto, unidades FROM stock WHERE unidades<2';
$resultado = $dwes->query($consulta);

$stock = $resultado->fetch_object();  //returns the object with its properties
while ($stock != null) {
        echo 'Producto '. $stock->producto .'('. $stock->unidades .' unidades)<br>';
        $stock = $resultado->fetch_object();
}
```

Exercise – Virtual Shop

Create a virtualhost called store (store.local)

The folder on that virtual host will be htdocs/store.

The **tienda** database will be used.

The main file (index.php or main.php, as you wish) should show a list of products, each product will be a link to a stock.php file.

stock.php must receive the product id and will show the stock of said product in each of the stores.

4.- MySQLi

Prepared queries

Prepared queries allow you to speed up the process when you have to perform the same query several times.

For example, if several values have to be inserted into a table.

The prepared queries are stored on the database server and executed when necessary.

Its use is not always recommended since it can overload the server.

4.- MySQLi

Prepared queries

Prepared queries can be static or dynamic. Dynamics admit parameters.

With dynamic prepared queries you tell the server which part is 'executable code', and which part is data.

Thanks to this functionality, prepared queries help prevent SQL injection attacks.

4.- MySQLi

Prepared queries

```
$dwes = new mysqli('localhost', 'dwes', 'dwes', 'tienda');

$consulta = $dwes->stmt_init(); //inicializa la sentencia y devuelve un objeto mysqli_stmt
$consulta->prepare('INSERT INTO familia (cod, nombre) VALUES ("TABLET", "iPad");');
$consulta->execute();
$consulta->close();

$dwes->close();
```

The previous example does not have much functionality since it will not be common for this type of query to be carried out.

4.- MySQLi

Prepared queries - bind_param

```
$dwes = new mysqli('localhost', 'dwes', 'dwes', 'tienda');

$consulta = $dwes->stmt_init();
$consulta->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?);');
$cod_producto = "TABLET";
$nombre_producto = "iPad";
$consulta->bind_param('ss', $cod_producto, $nombre_producto);
$consulta->execute();
$consulta->close();

$dwes->close();
```

4.- MySQLi

Prepared queries - bind_param

Characters depending on data type:

i → integer
d → float
s → string
b → content in binary format (BLOB)

With prepared queries you can only use variables:
$consulta->bind_param('ss', $cod_producto, $nombre_producto);

If literal values are used, it doesn't work:
$consulta->bind_param('ss', 'TABLET', 'iPAD');

4.- MySQLi

Prepared queries - bind_param

```
$dwes = new mysqli('localhost', 'dwes', 'dwes', 'tienda');

$consulta = $dwes->stmt_init();
$consulta->prepare('SELECT producto, unidades FROM stock WHERE unidades<2');
$consulta->execute();
$consulta->bind_result($producto, $unidades);
while($consulta->fetch()) {
		echo "Producto '. $producto .'('. $unidades .' unidades.<br>';
}
$consulta->close();

$dwes->close();
```

4.- MySQLi

Transactions

Transactions are a set of queries that have to be carried out in blocks.

If any of the queries in the set are not executed correctly, it will return to the initial state, undoing those queries in the set that were executed correctly.

Either all or none.

In order to carry out transactions, the table storage engine must support this behavior → InnoDB

4.- MySQLi

Transactions

With InnoDB the default behavior is that each individual query is included within its own transaction.

To change the default behavior use:

```
$dwes->autocommit(false);
```

With the previous instruction, automatic transactions are deactivated so that all queries made afterward will be part of a transaction.

4.- MySQLi

Transactions

Transactions must be completed manually with a php instruction.

- **commit** → All queries in the transaction have been executed successfully and the changes are saved to the database.

- **rollback** → Some transaction query has failed and the changes made must be undone.

4.- MySQLi

Transactions

There is no automatic method to know if any of the executed queries have failed.

It is the programmer's job to check one-by-one that they have been executed correctly and then commit or rollback.

4.- MySQLi

```
$todo_bien = true;
$dwes->autocommit(false);                    // start of transaction
$sql = 'UPDATE stock SET unidades=1 WHERE producto="3DSNG" AND tienda=1;';
$todo_bien = $dwes->query($sql);
if ($todo_bien) {
    $sql = 'INSERT INTO stock (producto, tienda, unidades) VALUES ("3DSNG", 3, 1);';
    $todo_bien = $dwes->query($sql);
}
if ($todo_bien) {
    $dwes->commit();
}
else {
    $dwes->rollback();
}
$dwes->autocommit(true);                     // end of transaction
```

4.- MySQLi

Use of try-catch

PHP does not automatically throw exceptions.

PHP has the **try-catch** block that allows you to **throw** and **catch** exceptions.

Using a try-catch block you can create a transaction with more readable code.

```
$dwes->autocommit(false);
try {
        $sql = 'UPDATE stock SET unidades=1 WHERE producto="3DSNG" AND tienda=1;';
        $todo_bien = $dwes->query($sql);
        if(!$todo_bien)
                throw new Exception('Error update', 1);

        $sql = 'INSERT INTO stock (producto, tienda, unidades) VALUES ("3DSNG", 3, 1);';
        $todo_bien = $dwes->query($sql);
        if(!$todo_bien)
                throw new Exception('Error insert', 1);

        $dwes->commit();
}
catch (Exception $e) {
        $dwes->rollback();
        print_r($e);
}
```

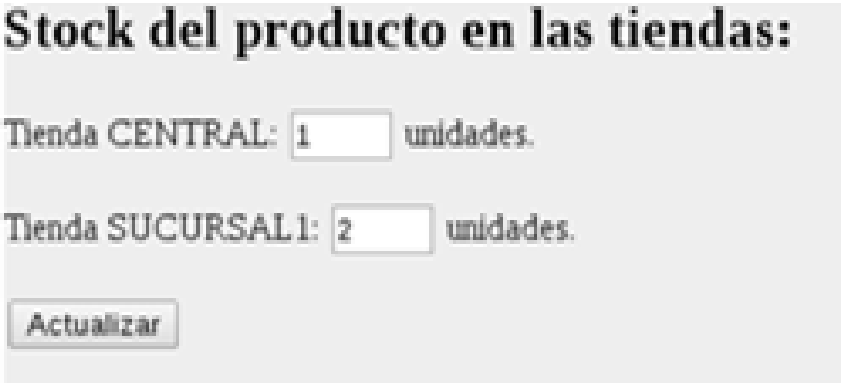
```
$dwes->autocommit(false);
try {
		$sql = 'UPDATE stock SET unidades=1 WHERE producto="3DSNG" AND tienda=1;';
		if(!$dwes->query($sql))
			throw new Exception('Error update', 1);

		$sql = 'INSERT INTO stock (producto, tienda, unidades) VALUES ("3DSNG", 3, 1);';
		if(!$dwes->query($sql))
			throw new Exception('Error insert', 1);

		$dwes->commit();
}
catch (Exception $e) {
		$dwes->rollback();
		print_r($e);
}
```

Exercise – Virtual Shop

In the virtual store, the stock.php file must be modified so that it allows modifying the stock quantity of the product in each store:



The data will be sent to the stock.php file itself:

- The query prepared for updating the units will be created.
- The query will be executed as many times as there are stores.
- A transaction must be used to update the units.