

Uso de técnicas de acceso a datos con PHP

Sitio: [Aula Virtual do IES de Teis](#)

Curso: Desarrollo web en entorno servidor 2023-24 (DAW-DUAL-A)

Libro: Uso de técnicas de acceso a datos con PHP

Impreso por: Joaquín Lafuente Espino

Data: Luns, 11 de Decembro de 2023, 09:33

Táboa de contidos

1. PDO

- 1.1. Creación de BD con phpMyAdmin
- 1.2. Conexión a BD con PDO
- 1.3. Consultas
- 1.4. Consultas preparadas
- 1.5. Resultado de consulta en array
- 1.6. Inserciones de datos
- 1.7. Actualizaciones de datos
- 1.8. Borrado de datos
- 1.9. Transacciones

2. mysqli

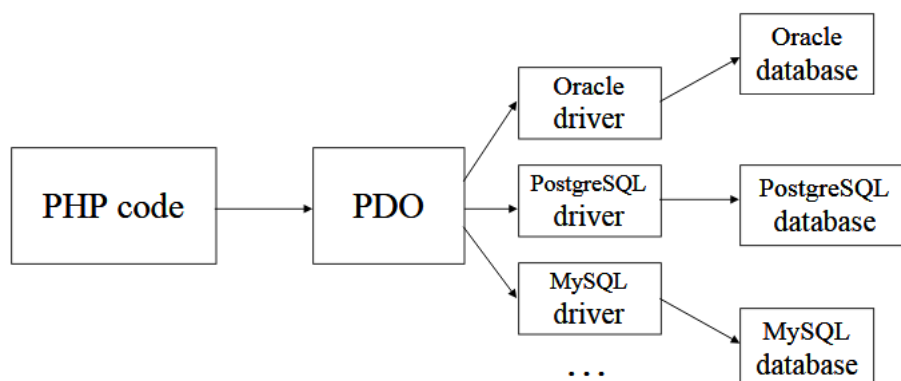
- 2.1. Crear una conexión
- 2.2. Ejecutar sentencias no preparadas
- 2.3. Sentencias preparadas
- 2.4. Transacciones
- 2.5. Manejo de errores

3. Bases de datos NoSQL: MongoDB

- 3.1. MongoDB: Instalación
- 3.2. Instalación servidor MongoDB y aplicación gráfica MongoDB Compass
- 3.3. Instalación de Composer
- 3.4. Instalación de la librería MongoDB para PHP
- 3.5. Creación de colecciones de ejemplo
- 3.6. Operaciones CRUD: Create
- 3.7. Operaciones de consulta
- 3.8. Operaciones update
- 3.9. Operaciones delete

1. PDO

- La extensión **Objetos de Datos de PHP** (PDO por sus siglas en inglés) define una **interfaz** para poder acceder a bases de datos en PHP.
- PDO proporciona una capa de abstracción de *acceso a datos* desde *PHP 5.1*
- Cada controlador de bases de datos que implemente la **interfaz PDO** puede exponer características específicas de la base de datos



Fuente: URL: <https://slideplayer.com/slide/6402864/>

Para su instalación, es necesario hacerse con los drivers o controladores del tipo específico de Base de datos a utilizar, descargarlos y situarlos en el directorio indicado en php.ini en la directiva `extension_dir`

; Directory in which the loadable extensions (modules) reside.

; <https://php.net/extension-dir>

;extension_dir = "./"

; On windows:

extension_dir = "C:\xampp\php\ext"

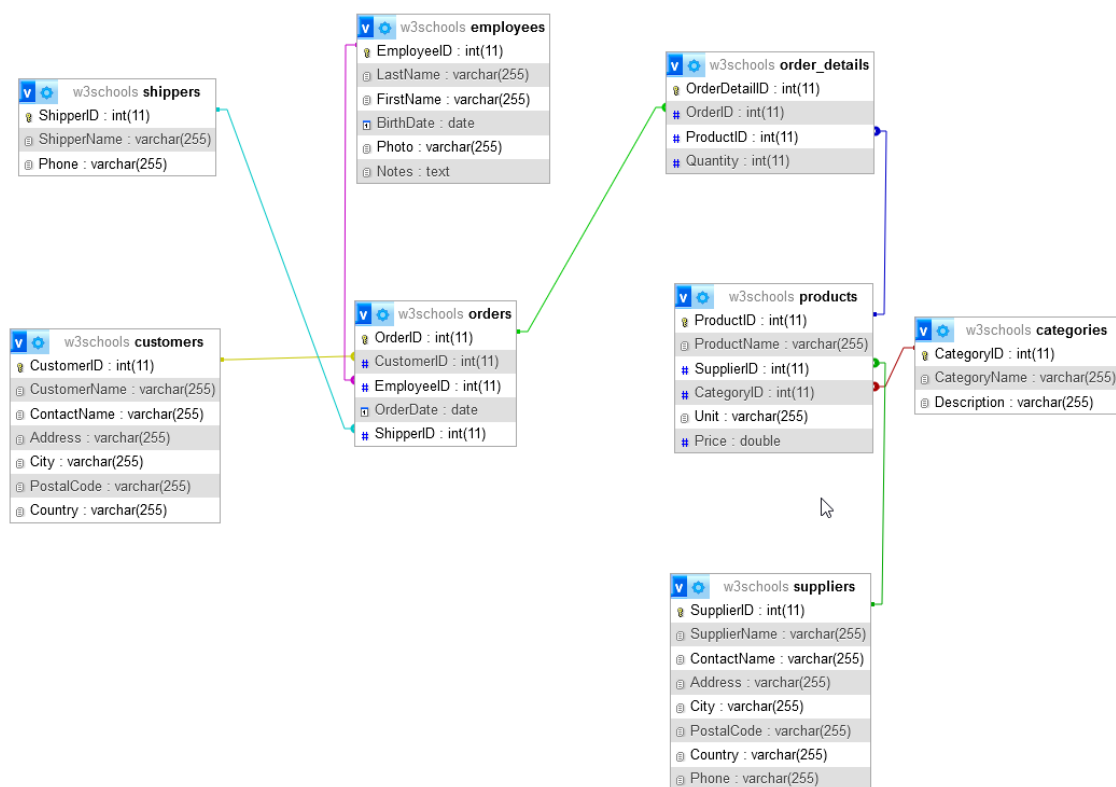
Y descomentar o añadir la extensión para ese tipo específico de BD en php.ini

```
933 ;extension=pdo_firebird
934 extension=pdo_mysql
935 ;extension=pdo_oci
936 ;extension=pdo_odbc
937 ;extension=pdo_pgsql
938 extension=pdo_sqlite
939 ;extension=pgsql
940 ;extension=shmop
941
```

Por defecto, en la instalación con XAMPP, vienen activadas las extensiones para MySQL y SQLite.

1.1. Creación de BD con phpMyAdmin

Vamos a crear una base de datos w3schools con las siguientes tablas:



Ejecuta el script que encontrarás en [MySQL scripts: w3schools Cartafol](http://localhost/phpmyadmin). Para ejecutarlos copia o arrastra el documento en la pestaña SQL de <http://localhost/phpmyadmin> y, a continuación, presiona el botón continuar:

Servidor: 127.0.0.1 » Base de datos: bookdb

Estructura SQL Buscar Generar una consulta Expor

Ejecutar la(s) consulta(s) SQL en la base de datos bookdb:

1

Limpiar Formato Obtener consulta almacenada automáticamente

☐ Enlazar parámetros

Guardar esta consulta en favoritos:

[Delimitador ;] ☒ Mostrar esta consulta otra vez ☐ Mantener la caja de texto con la consulta ☐ Deshacer («rollback») al finalizar ☒ Habilite la revisión de las claves foráneas

Continuar

1.2. Conexión a BD con PDO

Para conectarnos a una BD, tendremos que crear un objeto de [la clase PDO](#):

- Debemos capturar la excepción para evitar la finalización de ejecución del script
- Para cerrar la conexión, es necesario destruir el objeto asegurándose de que todas las referencias a él existentes sean eliminadas; esto se puede hacer asignando `null` a la variable que contiene el objeto. Si no se realiza explícitamente, PHP cerrará automáticamente la conexión cuando el script finalice. Si existen otras variables que hacen referencia a la misma instancia de PDO, también deben asignarse a `null`
- En general, en las aplicaciones web, se recomienda el uso de conexiones persistentes: **Las conexiones persistentes no se cierran al final del script**, sino que son almacenadas en caché y son reutilizadas cuando otro script solicite una conexión que use las mismas credenciales. La creación de una nueva conexión se considera costosa en tiempo y recursos. Para utilizar conexiones persistentes, se deberá establecer `PDO::ATTR_PERSISTENT` en las opciones del array del controlador pasado al constructor de PDO. Si este atributo se establece con `PDO::setAttribute()` después de la instanciación del objeto, el controlador **no utilizará conexiones persistentes**.

Contamos con ejemplos de creación de objetos PDO en la URL: <https://www.php.net/manual/es/pdo.connections.php>

Nosotros vamos a optar por utilizar los datos de conexión a la BD en un archivo de configuración, como se indica en el primer comentario de la documentación de [la clase PDO](#).

Podemos ver un ejemplo en GitHub: <https://github.com/dudwcs/Ejemplo4.1-SearchBook>

1.3. Consultas

A partir de la conexión que representa el objeto de la clase PDO podemos invocar el método

public [query](#)(string \$query, ?int \$fetchMode = null): [PDOStatement](#)|false

que devolverá un objeto [PDOStatement](#) o false si se ha producido un error.

Por ejemplo:

```
$pdostmt = $mbd->query('SELECT * from books');
while (($row = $pdostmt->fetch(PDO::FETCH_BOTH)) !== false) {
    echo "<pre>";
    print_r($row);
    echo "</pre>";
}
```

El primer argumento del método [fetch](#), indica cómo se recuperarán los datos según las constantes [PDO::FETCH_*](#). El valor por defecto es `PDO::FETCH_BOTH`, por lo que podría omitirse. Algunos valores de los valores que hemos visto son:

- `PDO::FETCH_ASSOC`

```
Array
(
    [book_id] => 1
    [title] => PHP Programming
    [isbn] =>
    [published_date] => 2023-01-01
    [publisher_id] => 1
)
```

Sintaxis de acceso al dato de una columna en concreto: `$row["book_id"]`

- `PDO::FETCH_NUM`

```
Array
(
    [0] => 1
    [1] => PHP Programming
    [2] =>
    [3] => 2023-01-01
    [4] => 1
)
```

Sintaxis de acceso al dato de una columna en concreto: `$row[0]`

- `PDO::FETCH_BOTH`

```
Array
(
    [book_id] => 1
    [0] => 1
    [title] => PHP Programming
    [1] => PHP Programming
    [isbn] =>
    [2] =>
    [published_date] => 2023-01-01
    [3] => 2023-01-01
    [publisher_id] => 1
    [4] => 1
)
```

Sintaxis de acceso al dato de una columna en concreto: `$row["book_id"]` o `$row[0]`

- `PDO::FETCH_OBJ`

Se creará un objeto de una clase estándar (stdClass) con tantos atributos como columnas se recuperen. En PHP, al contrario que en Java, se accede a los métodos o atributos no estáticos con flecha `->` en lugar de `.`

```
stdClass Object
(
    [book_id] => 1
    [title] => PHP Programming
    [isbn] =>
    [published_date] => 2023-01-01
    [publisher_id] => 1
)
```

Sintaxis de acceso al dato de una columna en concreto: `$row->book_id`

1.4. Consultas preparadas

Se usan para consultas que se pueden ejecutar varias veces:

- optimiza el rendimiento: se almacena en caché del plan de consulta
- ayuda a prevenir inyecciones SQL eliminando la necesidad de entrecomillar manualmente los parámetros
- permite añadir parámetros genéricos que serán reemplazados por valores literales o por los valores de variables en concreto

Para llevar a cabo consultas preparadas, se llama a [PDO::prepare\(\)](#), en lugar de a [PDO::query](#) y, a continuación, se ejecuta llamando a [PDOStatement::execute\(\)](#).

```
$pdostmt = $mbd->prepare('SELECT * from books where title like :nombre ');
$filtro = "P%";
$pdostmt->bindParam("nombre", $filtro);
$pdostmt->execute();

while (($row = $pdostmt->fetch(PDO::FETCH_OBJ)) !== false) {
    echo "<pre>";
    print_r($row);
    echo "</pre>";
}
```

Tipos de parámetros

Los parámetros pueden ser:

- **nominales:** Utilizan los dos puntos : seguidos de un identificador, como en el caso anterior.

```
$pdostmt = $mbd->prepare('SELECT * from books where title like :nombre ');
$filtro = "P%";
$pdostmt->bindParam("nombre", $filtro);
$pdostmt->execute();

while (($row = $pdostmt->fetch(PDO::FETCH_OBJ)) !== false) {
    echo "<pre>";
    print_r($row);
    echo "</pre>";
}
```

- **posicionales:** Utilizan interrogante ? y se sustituyen utilizando el índice numérico que ocupan

```
$pdostmt = $mbd->prepare('SELECT * from books where title like ? or isbn like ? ');
$filtro_title = "P%";
$filtro_isbn = "%38%";
$pdostmt->bindParam(1, $filtro_title);
$pdostmt->bindParam(2, $filtro_isbn);
$pdostmt->execute();
while (($row = $pdostmt->fetch(PDO::FETCH_OBJ)) !== false) {
    echo "<pre>";
    print_r($row);
    echo "</pre>";
}
```

Tipos de sustitución de parámetros

Sustitución por variable: Se vincula una variable de PHP a un parámetro de sustitución con nombre o de signo de interrogación:


```
$pdostmt = $mbd->prepare('SELECT * from books where title like :nombre ');
$filtro = "P%";
$pdostmt->bindParam("nombre", $filtro);
$pdostmt->execute();
while (($row = $pdostmt->fetch(PDO::FETCH_OBJ)) !== false) {
    echo "<pre>";
    print_r($row);
    echo "</pre>";
}
```

Se vinculan por referencia, es decir, si se modifica \$filtro después de llamar a bindParam, se ejecutará la consulta con el nuevo valor de \$filtro.

Sustitución por valor literal: Vincula a un valor (no a una referencia) el parámetro de sustitución con nombre o de signo de interrogación:
(Se puede usar un valor o una variable)

```
$pdostmt = $mbd->prepare('SELECT * from books where title like :nombre ');
$pdostmt->bindValue("nombre", "P%");
$pdostmt->execute();
while (($row = $pdostmt->fetch(PDO::FETCH_OBJ))
    !== false) {
    echo "<pre>";
    print_r($row);
    echo "</pre>";
}
```

Sustitución por array en el método execute(). Sirve tanto para parámetros nominales como posicionales:

```
$pdostmt = $mbd->prepare('SELECT * from books where title like :nombre or isbn like :isbn ');
$pdostmt->execute( array("nombre"=> "P%", "isbn"=> "%38%" ));
while (($row = $pdostmt->fetch(PDO::FETCH_OBJ)) !== false) {
    echo "<pre>";
    print_r($row);
    echo "</pre>";
}

$pdostmt = $mbd->prepare('SELECT * from books where title like ? or isbn like ? ');
$pdostmt->execute( array("P%", "%38%" ));
while (($row = $pdostmt->fetch(PDO::FETCH_OBJ)) !== false) {
    echo "<pre>";
    print_r($row);
    echo "</pre>";
}
```

Depuración Consultas SQL

Para depuración, podemos usar el método de **PDOStatement::debugDumpParams** que vuelca la información contenida en una sentencia preparada directamente en la salida, veremos el SQL de la consulta y el número de parámetros, nombre, tipo, etc.

```
$pdostmt->execute();
$pdostmt->debugDumpParams();
```

1.5. Resultado de consulta en array

[fetchAll](#) permite obtener el resultado de una consulta en un único array

```
public PDOStatement::fetchAll(int $fetch_style = ?, mixed $fetch_argument = ?, array $ctor_args = array()): array
```

`fetch_style`: coincide con el modo del primer argumento de `fetch`: `PDO::FETCH_ASSOC`, `PDO::FETCH_BOTH`, `PDO::FETCH_NUM`, etc.

Si se usa `PDO::FETCH_COLUMN` permite obtener todos los valores de una única columna de un conjunto de resultados. Se puede indicar con un índice numérico **basado en cero** para la columna que se desea obtener.

```
echo "FetchAll de libros columna 0";  
$pdostmt = $mbd->query('SELECT * FROM books');  
$array = $pdostmt->fetchAll(PDO::FETCH_COLUMN, 0);  
  
echo "<pre>";  
print_r($array);  
echo "</pre>";
```

```
Array  
(  
    [0] => 1  
    [1] => 2  
    [2] => 3  
)
```

1.6. Inserciones de datos

Para crear nuevos registros en una tabla, usaremos una sentencia SQL con INSERT INTO. Puede tratarse de una consulta preparada o no. Dado que las inserciones normalmente simplemente cambian los argumentos a insertar, se recomienda una consulta preparada.

Para recuperar el identificador generado en las columnas con autoincrement en MariaDB, usaremos el método [lastInsertId\(\)](#).

Si el driver de la BD soporta este tipo de columnas, devolverá una cadena de texto. En caso contrario, devolverá un error.

```
function createPublisher(string $name):string|false {
    global $conn;

    $pdostmt = $conn->prepare("INSERT INTO publishers(name) VALUES ( :name) ");
    $pdostmt->bindValue("name", $name);

    $pdostmt->execute();

    //Recuperamos el id de la última inserción
    $publisher_Id = $conn->lastInsertId();
    //Devolvemeos id o false si hubo error
    return ($publisher_Id !== false) ? $publisher_Id : null;
}
```

Si se usa en una transacción, debe llamarse a [lastInsertId\(\)](#) antes de hacer commit.

1.7. Actualizaciones de datos

En el caso de las actualizaciones, se usará una sentencia SQL UPDATE.

De nuevo, se recomiendan las sentencias preparadas.

```
//Devuelve true si se ejecutó sin error, false en caso contrario
public function update($pub_id, $new_name):bool{

    global $conn;

    $conn->prepare("UPDATE publihsers"
        . " SET name= :name "
        . "WHERE publisher_id= :pub_id");
    $pdostmt->bindValue("name", $new_name);
    $pdostmt->bindValue("pub_id", $pub_id);

    $pdostmt->execute();
    return ($pdostmt->rowCount()===1);
}
```

1.8. Borrado de datos

En el caso de las actualizaciones, se usará una sentencia SQL DELETE.

De nuevo, se recomiendan las sentencias preparadas.

Podemos ayudarnos del método [rowCount\(\)](#) para averiguar si realmente se ha eliminado el registro por id.

```
//devuelve true si se ha borrado correctamente, false en caso contrario
public function delete($pub_id): bool {
    global $conn;

    $conn->prepare("DELETE FROM publishers WHERE publisher_id
= :pub_id");
    $pdostmt->bindValue("pub_id", $pub_id);
    $pdostmt->execute();
    return ($pdostmt->rowCount() === 1);
}
```

1.9. Transacciones

Cuando necesitamos hacer cambios en la base de datos en varias tablas como parte de una única operación lógica, necesitamos ejecutar una transacción.

El típico ejemplo de transacción es la transferencia de una cuenta bancaria a otra, donde se modificarán al menos dos registros (cuenta origen y cuenta destino) restando un montante en la primera y sumándolo en la segunda.

En la base de datos **bookdb** esto debería hacerse para la operación de eliminar un libro con autores, pues se modificarán al menos la tabla **book** y **book_authors**.

La [clase PDO](#) tiene los siguientes métodos relacionados con transacciones:

- public **beginTransaction()**: bool: Desactiva el modo 'autocommit' que indica que cada consulta/modificación a la BD se realice en una transacción. Mientras el modo 'autocommit' esté desactivado, no se consignarán los cambios realizados en la base de datos a través de una instancia de PDO hasta que se finalice la transacción con una llamada a [PDO::commit\(\)](#). Una llamada a [PDO::rollBack\(\)](#) revertirá todos los cambios de la base de datos y devolverá la conexión al modo 'autocommit'. **Lanza una [PDOException](#) si ya hay una transacción iniciada o el controlador no admite transacciones.**
- public **commit()**: bool: Confirma los cambios en la BD. Lanza una [PDOException](#) si no hay ninguna transacción activa.
- public **rollBack()**: bool: Revierte la BD al estado anterior a comenzar la transacción. Lanza una [PDOException](#) si no hay ninguna transacción activa.
- public **inTransaction()**: bool: Comprueba si una transacción está actualmente activa dentro del controlador. Este método únicamente funciona para controladores de bases de datos que admitan transacciones.

Tenemos un ejemplo en <https://github.com/dudwcs/Tarea04.1-enunciado.git>:

```

 9  function borrar_libro(int $cod): bool
10  {
11      $exito = false;
12
13
14      try {
15          $conProyecto = getConnection();
16          //Tenemos que modificar 2 tablas
17          //Comenzamos la tx
18          $conProyecto->beginTransaction();
19
20          $delete = "delete from books where book_id= ?";
21          $delete_book_authors = "delete from book_authors where book_id = ?";
22
23          $stmt = $conProyecto->prepare($delete);
24          $stmt_del_book_authors = $conProyecto->prepare($delete_book_authors);
25
26          //Ejecutamos y reemplazamos los parámetros con un array
27          $exito = $stmt_del_book_authors->execute([$cod]);
28          $exito = $exito && $stmt->execute([$cod]);
29          if ($exito)
30              $conProyecto->commit();
31          else
32              $conProyecto->rollBack();
33      } catch (PDOException $ex) {
34          $conProyecto->rollBack();
35          $exito = false;
36          echo "Ocurrió un error al borrar el libro con book_id $cod, mensaje: " . $ex->getMessage();
37      }
38
39      //Devolvemos el resultado de la operación
40      return $exito;
41  }
42

```

2. mysqli

Hasta ahora hemos visto cómo conectarnos a una BD de MySQL con PDO, donde el programador realiza las llamadas a la API de PDO, PDO utiliza el driver PDO MySQL para llevar a cabo la comunicación con el servidor MySQL. PDO puede trabajar con drivers de diferentes fabricantes.

Existen además 2 extensiones de PHP específicas para conectarse a una base de datos MySQL y que proporcionan su propia API:

- Extensión **mysql** de PHP: Proporciona una interfaz procedimental, y está pensada para usar sólo con versiones de MySQL anteriores a la 4.1.3.
- Extensión **mysqli** de PHP: (mysql improved). Proporciona una interfaz dual (orientada a objetos y procedimental). Es la recomendada para versiones MySQL 4.1 y posterior.

mysqli está habilitada por defecto en la instalación de XAMPP. Se puede ver en el php.ini:

```
927 extension=curl
928 ;extension=ffi
929 ;extension=ftp
930 extension=fileinfo
931 ;extension=gd
932 extension=gettext
933 ;extension=gmp
934 ;descomentado intl 2023-09-30 fechas
935 extension=intl
936 ;extension=imap
937 extension=mbstring
938 extension=exif ; Must be after mbstring as it depends on it
939 extension=mysqli
940 ;extension=oci8_12c ; Use with Oracle Database 12c Instant Client
941 ;extension=oci8_19 ; Use with Oracle Database 19 Instant Client
942 ;extension=odbc
943 ;extension=openssl
944 ;extension=pdo_firebird
945 extension=pdo_mysql
946 ;extension=pdo_oci
947 ;extension=pdo_odbc
948 ;extension=pdo_pgsql
949 extension=pdo_sqlite
950 ;extension=pgsql
951 ;extension=shmop
952
```

Más información: <https://www.php.net/manual/es/mysqli.overview.php> donde tenéis una tabla comparativa de las funcionalidades de los 3 modos de acceso a MySQL

2.1. Crear una conexión

Para ver cómo crear una conexión, visitemos la URL: <https://www.php.net/manual/es/mysqli.quickstart.connections.php>

Nosotros **usaremos la versión orientada a objetos de `mysqli`**

El nombre del host `localhost` tiene un significado especial en Unix. No es posible abrir una conexión TCP/IP usando como nombre de host `localhost`, se debe usar `127.0.0.1` en su lugar.

Para cerrar una conexión: public `close()`: bool

Para ver una comparativa de interfaz POO y la interfaz procedimental: [PHP: Resumen de las funciones de la extensión MySQLi - Manual](#)

Desde la versión 8.1.0, la configuración por defecto para gestionar los posibles errores surgidos durante la conexión `MYSQLI_REPORT_ERROR` | `MYSQLI_REPORT_STRICT` Para versiones anteriores se usaba `MYSQLI_REPORT_OFF`.

Por lo tanto, desde la versión 8.1.0, si hay algún error se lanzará una excepción `mysqli_sql_exception`

Los distintos valores para la gestión de errores se puede consultar en <https://www.php.net/manual/en/mysqli-driver.report-mode.php>

Un ejemplo de cómo conseguir una conexión con `mysqli` con los datos directamente en el código (mala práctica) se puede ver en la siguiente imagen:

```
16     $con = null;
17     $host = "localhost";
18     $db = "bookdb";
19     $user = "user-bookdb";
20     $pass = "abc123.";
21
22     try {
23
24         $con = new mysqli($host, $user, $pass,$db);
25
26
27     } catch (mysqli_sql_exception $ex) {
28
29         echo "Error en la conexión: mensaje: " . $ex->getMessage();
30     }
31     catch(Exception $ex){
32         echo "Ha ocurrido una excepción: " . $ex->getMessage();
33     }
```


2.2. Ejecutar sentencias no preparadas

<https://www.php.net/manual/es/mysqli.quickstart.statements.php>

Las sentencias no preparadas se pueden ejecutar con las funciones:

- **`mysqli::query(string $query, int $resultmode = MYSQLI_STORE_RESULT): mixed`**: Es la más común y combina la ejecución de la sentencia con la obtención del resultado en un *buffer*. Devolverá un objeto [mysqli_result](#) (para consultas que devuelvan conjunto de resultados), false si hay un error y true para consultas que no devuelven nada y son exitosas. Con la opción por defecto, `MYSQLI_STORE_RESULT`, los resultados se recuperan todos juntos del servidor de la base de datos y se almacenan de forma local. Si cambiamos esta opción por el valor `MYSQLI_USE_RESULT`, los datos se van recuperando del servidor según se vayan necesitando, fila a fila.
 - Se recomienda usar `MYSQLI_STORE_RESULT` si no se esperan muchos registros en el resultado.
 - No se recomienda usar si hay mucho procesamiento en el lado cliente `MYSQLI_USE_RESULT` pues impedirá que otros hilos actualicen cualquier tabla desde la cuales se están obteniendo los datos.
- **`mysqli_real_query()`**: Llamar a [mysqli_query\(\)](#) es idéntico que llamar a [mysqli_real_query\(\)](#) seguido de [mysqli_store_result\(\)](#). [mysqli_real_query\(\)](#) ejecuta una sola consulta contra la base de datos cuyo resultado puede ser recuperado o almacenado mediante las funciones [mysqli_store_result\(\)](#) o [mysqli_use_result\(\)](#).
- **`mysqli_multi_query()`**: Ejecuta una o múltiples consultas concatenadas por puntos y comas.

Obtención de resultados

[mysqli_result](#) permite obtener los resultados a través de los siguientes métodos:

- public [fetch_all](#)(int \$mode = `MYSQLI_NUM`): array: Permite obtener **todas las filas** en un array asociativo, numérico, o en ambos
- public [fetch_array](#)(int \$mode = `MYSQLI_BOTH`): array|null|false: Permite obtener **una fila** de resultados como un array asociativo, numérico, o ambos
- public [fetch_assoc](#)(): array|null|false: Permite obtener **una fila** de resultado como un array asociativo
- public [fetch_object](#)(string \$class = "stdClass", array \$constructor_args = []): object|null|false: Devuelve la fila actual de un conjunto de resultados como un objeto
- public [fetch_row](#)(): array|null|false: Obtiene **una fila** de resultados como un array enumerado

```

23  function obtener_libros_no_preparada(): array
24  {
25
26      $libros_array = [];
27      $resultado = null;
28      try {
29          //$conProyecto es de tipo mysqli
30          $conProyecto = getConnection();
31          $consulta = "select book_id, title from books order by title";
32          $resultado = $conProyecto->query($consulta);
33
34          while ($row = $resultado->fetch_assoc()) {
35              array_push($libros_array, $row);
36          }
37          //otra opción en lugar de registro a registro
38          //$libros_array = $resultado->fetch_all(MYSQLI_ASSOC);
39
40      } catch (Exception $e) {
41      }
42
43      <div class="alert alert-danger" role="alert">
44          <?php echo "Ha ocurrido una excepción: " . $e->getMessage(); ?>
45      </div>
46
47      <?php
48      } finally {
49          if ($resultado != null) {
50              $resultado->close();
51          }
52      }
53  }
54  }
55  }
56  }
57  }
58  }
59  }
60  }
61  }
62  }
63  }
64  }
65  }
66  }
67  }
68  }
69  }
70  }
71  }
72  }
73  }
74  }
75  }
76  }
77  }
78  }
79  }
80  }
81  }
82  }
83  }
84  }
85  }
86  }
87  }
88  }
89  }
90  }
91  }
92  }
93  }
94  }
95  }
96  }
97  }
98  }
99  }
100 }
```

Para liberar los recursos de un `mysqli_result` se puede usar cualquiera de los siguientes métodos:

- public [free](#)(): void
- public [close](#)(): void

- public [free_result\(\)](#): void

Por defecto, las sentencias no preparadas devolverán todos los resultados como cadenas de texto. Este comportamiento predeterminado se puede cambiar usando una opción de conexión. Si se utiliza la opción de conexión no existirán diferencias. Ejemplo 5 en <https://www.php.net/manual/es/mysqli.quickstart.statements.php>

2.3. Sentencias preparadas

La ejecución de sentencias preparadas consiste en dos etapas:

- **preparación:** se envía una plantilla de sentencia al servidor de bases de datos. El servidor realiza una comprobación de sintaxis e inicializa los recursos internos del servidor para su uso posterior: En la clase `mysqli`, hay que llamar al método `prepare(string $query): mysqli_stmt`
- **ejecución:** En la clase `mysqli_stmt` hay que llamar al método `execute(): bool`

Se recomienda el cierre explícito del objeto `mysqli_stmt` con el método `close(): bool`

Una vez preparada y ejecutada la consulta, finalmente se obtendrán los resultados con los mismos métodos que hemos visto en el apartado anterior (`fetch_*`)

Sustitución de parámetros posicionales

`mysqli` soporta el uso de parámetros de **sustitución posicionales** anónimos con `?`, pero **`mysqli` no soporta parámetros nominales**

En la clase `mysqli_stmt` se puede realizar la sustitución con el método

`bind_param(string $types, mixed &$var1, mixed &$... = ?): bool`

donde `$types` es una cadena que contiene **uno o más caracteres** (un carácter por cada parámetro) que especifican los tipos para el correspondiente enlazado de variables:

Especificación del tipo de caracteres

Carácter	Descripción
i	la variable correspondiente es de tipo entero
d	la variable correspondiente es de tipo double
s	la variable correspondiente es de tipo string
b	la variable correspondiente es un blob y se envía en paquetes

Habrà tantas variables como símbolos `?` haya en la sentencia preparada.

```

28 try {
29     $con = new mysqli();
30     $stmt = $con->prepare("select title as resultado from books where UPPER
        (title) like ?
        union
        select TRIM(Concat(coalesce(first_name, ''), coalesce(middle_name, ''
        ), coalesce(last_name, '')))
        as resultado from authors where first_name like ?");
31
32     $filtro = "%" . strtoupper($terminos_busqueda) . "%";
33     $stmt->bind_param("ss", $filtro, $filtro);
34     $stmt->execute();
35
36     $resultado = $stmt->get_result();
37
38     $counter = 0;
39     /*fetch_assoc devuelve: array asoc, null si no hay más filas o false si
40     falla algo */
41     while (($row = $resultado->fetch_assoc())) {
42         $counter++;
43         if ($counter == 1) {
44             echo "<ol>";
45         }
46         echo "<li>" . $row["resultado"] . "</li>";
47     }
48     if ($counter > 0) {
49         echo "</ol>";
50     }
51     if ($counter == 0) {
52         echo "<p>No se han encontrado resultados</p>";
53     }
54     catch (Exception $e) {
55         echo "<p>Ha ocurrido una excepción: " . $e->getMessage() . "</p>";
56     }
57
58     $con = null;
59     if ($stmt != null)
60         $stmt->close();
61     if ($resultado != null)
62         $resultado->close();
63 }

```

Cierre de recursos

Reemplazo de parámetros por medio de un array

A partir de la versión PHP 8.0.1 se ha permitido reemplazar los parámetros con un array en el método

`public mysqli_stmt::execute(?array $params = null): bool`

```

/* Prepare an insert statement */
$stmt = $mysqli->prepare('INSERT INTO myCity (Name, CountryCode, District) VALUES (?, ?, ?)');

/* Execute the statement */
$stmt->execute(['Stuttgart', 'DEU', 'Baden-Wuerttemberg']);

```

Inserciones preparadas

[mixed \\$mysqli->insert_id](#): Devuelve el id autogenerado que se utilizó en la última consulta

```

public function create($name) {
    //... se crea la conexión
    $sentencia = $conn->prepare("INSERT INTO publishers(name) VALUES ( ? )");

    $sentencia->bind_param("s", $name);

    $sentencia->execute();

    //Recuperamos el id de la última inserción
    $publisher_Id = $conn->insert_id;

    $sentencia->close();
    return $publisher_Id;
}

```

Eliminación de registros

Las consultas preparadas también pueden servir para ejecutar sentencias SQL DELETE. Podemos ayudarnos de la propiedad [affected_rows](#)

```

public function delete($id): bool {
    //creamos un objeto mysqli $conn
    $sentencia = $conn->prepare( "DELETE FROM publishers WHERE publisher_id = ?");
    $sentencia->bind_param("i", $id);
    $sentencia->execute();

    $exito= ($sentencia->affected_rows ===1);

    $sentencia->close();

    return $exito;
}

```

2.4. Transacciones

Al igual que PDO, mysqli cuenta con al menos los siguientes 3 métodos relacionados con transacciones:

- [mysqli::autocommit\(\)](#) — Activa o desactiva el modo 'auto-commit' en consultas para la conexión a la base de datos. El modo [auto-commit](#) significa que toda consulta que se ejecute tiene su propia transacción implícita, si la base de datos la admite. Se suele usar en versiones anteriores a PHP 5.5 porque no existía begin_transaction.
- [mysqli::begin_transaction](#) — Inicia una transacción (disponible a partir de PHP 5.5)
- [commit\(int \\$flags = ?, string \\$name = ?\)](#): bool
- [mysqli::rollback](#) — Revierte la transacción actual

Un ejemplo de uso de transacción para la eliminación de un libro de bookdb se puede ver en la imagen:

```

13 }
14 try {
15     $conProyecto = getConnection();
16     //Tenemos que modificar 2 tablas
17
18     //Comenzamos la tx
19     $conProyecto->begin_transaction();
20
21     $delete = "delete from books where book_id= ?";
22     $delete_book_authors = "delete from book_authors where book_id = ?";
23
24     $stmt = $conProyecto->prepare($delete);
25     $stmt_del_book_authors = $conProyecto->prepare($delete_book_authors);
26
27     //Ejecutamos y reemplazamos los parámetros con un array
28     $exito = $stmt_del_book_authors->execute([$cod]);
29     $exito = $exito && $stmt->execute([$cod]);
30
31     if ($exito) {
32         $conProyecto->commit();
33     } else {
34         $conProyecto->rollBack();
35     }
36     //Throwable es una clase madre común a error y a Exception. Capturará un fatal error y cualquier excepción
37 } catch (Throwable $ex) {
38     $conProyecto->rollBack();
39     $exito = false;
40     echo "Ocurrió un error al borrar el libro con book_id $cod, mensaje: " . $ex->getMessage();
41 }

```

2.5. Manejo de errores

Existen 2 atributos para las clases [mysqli](#) y [mysqli_stmt](#).

- int [\\$mysqli_stmt->errno](#): Devuelve el código de error de la llamada de la sentencia más reciente
- string [\\$mysqli_stmt->error](#): Devuelve una descripción en forma de string del último error de una sentencia

Sin embargo, desde la versión de PHP 8.1.0, se usan por defecto las opciones `MYSQLI_REPORT_ERROR` | `MYSQLI_REPORT_STRICT` que lanzarán una [mysqli_sql_exception](#) para los errores generados por las funciones de mysqli.

<https://www.php.net/manual/en/mysqli-driver.report-mode.php>

Nosotros usaremos try-catch para controlar cualquier excepción incluida [mysqli_sql_exception](#).

3. Bases de datos NoSQL: MongoDB

- **MongoDB** (del inglés humongous, "enorme") es un sistema de base de datos NoSQL orientado a documentos de código abierto: Las versiones lanzadas antes del 16 de octubre de 2018 se publican bajo licencia AGPL. Todas las versiones posteriores al 16 de octubre de 2018, incluidos los parches lanzados para versiones anteriores, se publican bajo [Licencia pública del lado del servidor \(SSPL\) v1](#).
- En lugar de guardar los datos en tablas, **guarda estructuras de datos BSON** (una especificación similar a **JSON**, pero con un formato binario)
- Usa un esquema dinámico, no todos los documentos están obligados a tener la misma estructura de datos, pero todos deberían tener un identificador `_id`

```
_id: ObjectId('5c8eccc1caa187d17ca6ed16')
city: "ALPINE"
zip: "35014"
loc: Object
  y: 33.331165
  x: 86.208934
pop: 3062
state: "AL"
```

```
_id: ObjectId('5c8eccc1caa187d17ca6ed17')
city: "BESSEMER"
zip: "35020"
loc: Object
  y: 33.409002
  x: 86.947547
pop: 40549
state: "AL"
```

- Comparando con una base de datos relacional, se puede decir que las **colecciones son como tablas** y **los documentos son registros o filas de la tabla**.

SQL	MongoDB
Tabla	Conjunto
Fila	Documento
Columna	Campo
Clave principal	ObjectId
Índice	Índice
Ver	Ver
Tabla u objeto anidado	Documento incrustado

Fuente: <https://aws.amazon.com/es/nosql/> (también podéis encontrar una comparativa SQL con NoSQL)

Características

Permite uso de lenguaje de consultas, indexación, replicación y balanceo de carga, agregación (similares al "GROUP BY" de SQL), etc.

Mas información: <https://www.mongodb.com/es/what-is-mongodb>

3.1. MongoDB: Instalación

Instalación de la extensión PHP para MongoDB

Seguimos las instrucciones de la documentación: <https://www.php.net/manual/en/mongodb.installation.windows.php>

Visitamos el repositorio de la última versión 1.17:

<https://github.com/mongodb/mongo-php-driver/releases/tag/1.17.0>

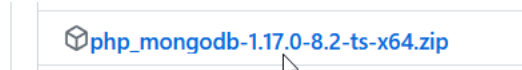
Identificamos la versión de PHP que tenemos instalada: 8.2.4 (Para averiguar vuestra versión de PHP podéis usar `phpinfo()`; en un script de PHP) =>

1- buscamos [php_mongodb-1.17.0-8.2-ts-x64.zip](#)

2- identificamos la versión **ts** (thread-safe)

3- identificamos la arquitectura de la máquina x64 o x86

En mi caso es este fichero:



4- Descargamos el fichero .zip

5- Descomprimos

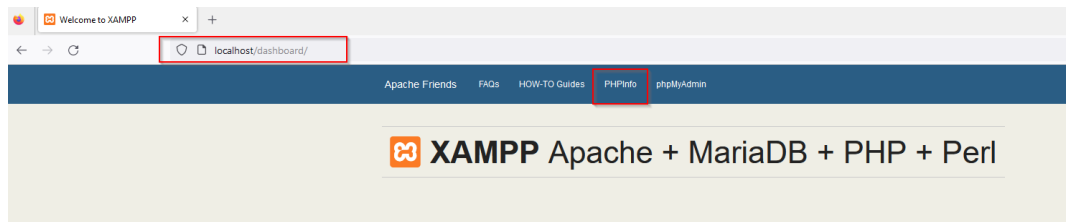
6- Copiamos el fichero **php_mongodb.dll** en C:\xampp\php\ext

7- Editamos el fichero C:\xampp\php\php.ini añadiendo la línea

```
extension=php_mongodb.dll
;extension=xml
;extension=zip
;zend_extension=opcache
extension=php_mongodb.dll
;extension=...
```

8- Arranca Apache desde el panel de control

Una vez realizados los 8 pasos, usamos un navegador y en la URL introducimos localhost y seleccionamos la pestaña PHPINFO:



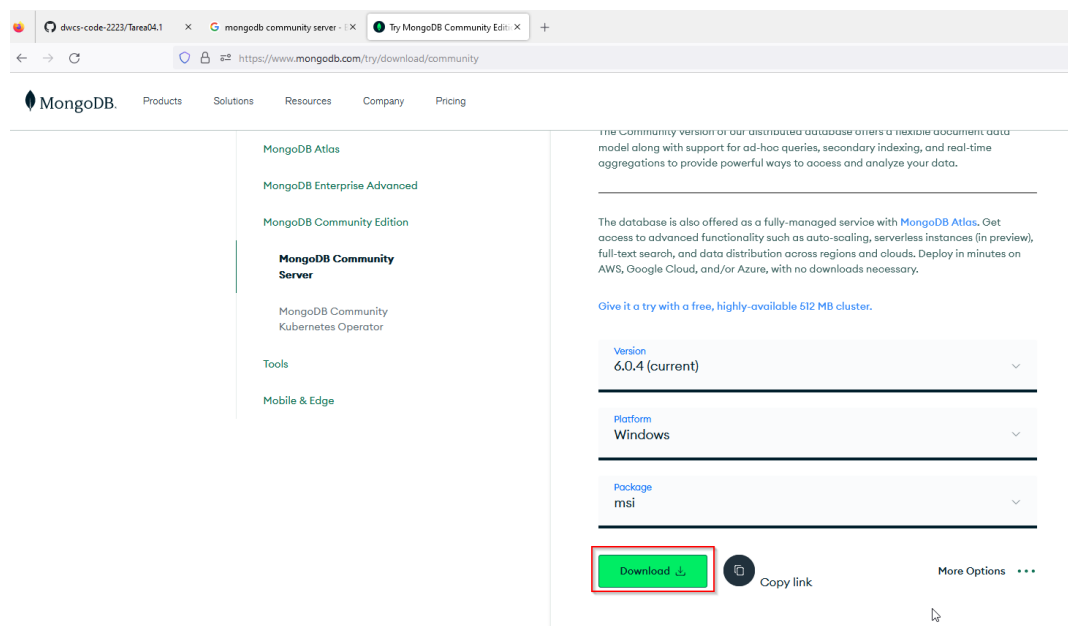
Buscamos con CTRL+F la palabra *mongo*. Deberíamos ver información del driver para la conexión con MongoDB:



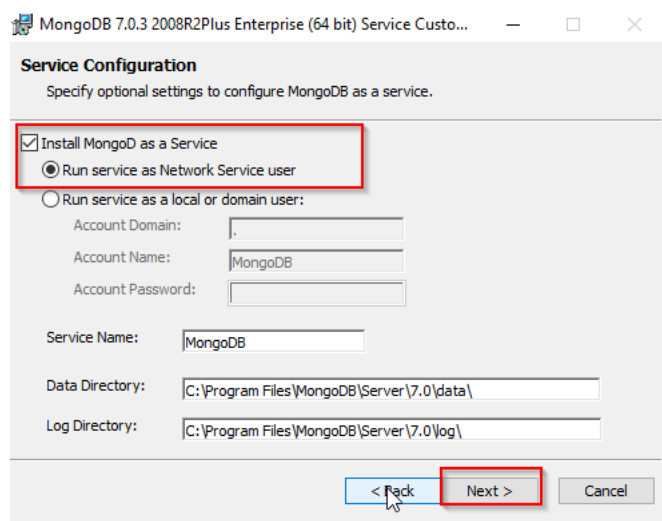
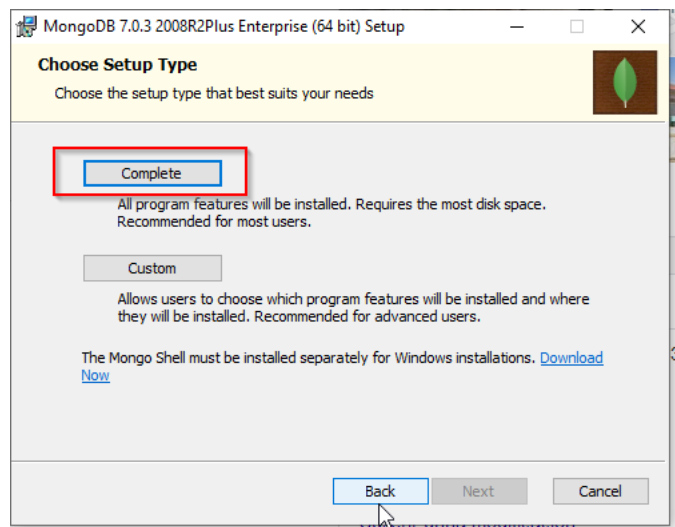
MongoDB support	enabled
MongoDB extension version	1.13.0
MongoDB extension stability	stable
libbson bundled version	1.21.1
libmongoc bundled version	1.21.1
libmongoc SSL	enabled
libmongoc SSL library	OpenSSL
libmongoc crypto	enabled
libmongoc crypto library	libcrypto
libmongoc crypto system profile	disabled
libmongoc SASL	enabled
libmongoc ICU	disabled
libmongoc compression	disabled
libmongocrypt bundled version	1.3.2
libmongocrypt crypto	enabled
libmongocrypt crypto library	libcrypto

3.2. Instalación servidor MongoDB y aplicación gráfica MongoDB Compass

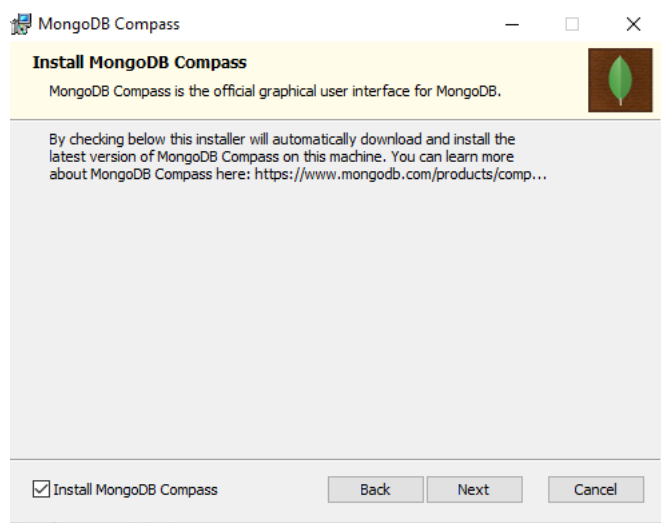
Visita la URL: <https://www.mongodb.com/try/download/enterprise> y descarga el instalador (tendrás que indicar algunos datos):



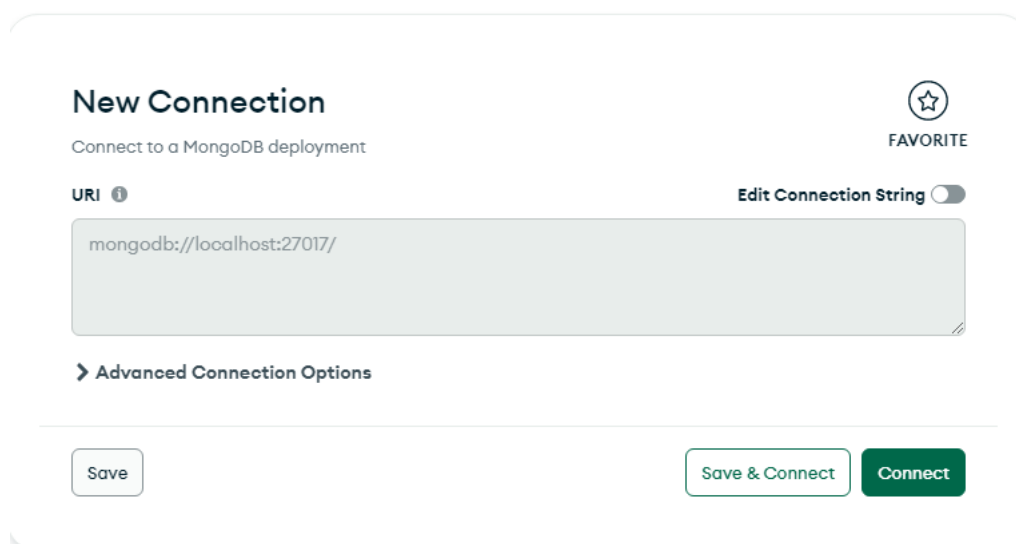
Realiza una instalación **COMPLETA** por defecto como **servicio** Network Service.



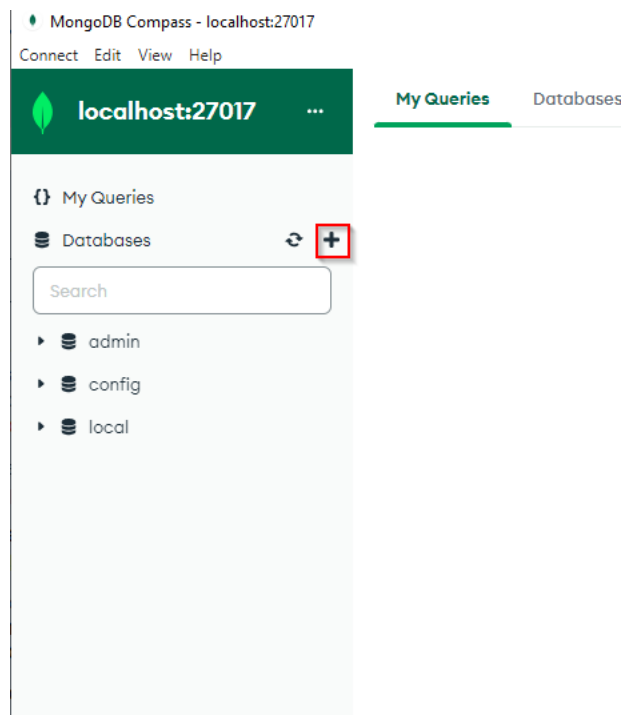
Incluye la aplicación gráfica MongoDB Compass para interactuar con el servidor:



Conectémonos al servidor local en el puerto por defecto 27017:



Crea una nueva BD **test** y una colección **users**:



Create Database

Database Name

test

Collection Name

users

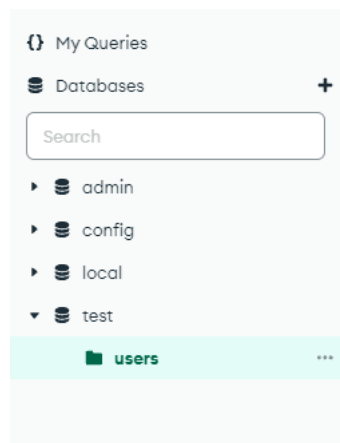
> Advanced Collection Options

(e.g. Time-Series, Capped, Clustered collections)

Cancel

Create Database

En el resultado deberíamos ver la nueva base de datos **test** y la colección **users**:

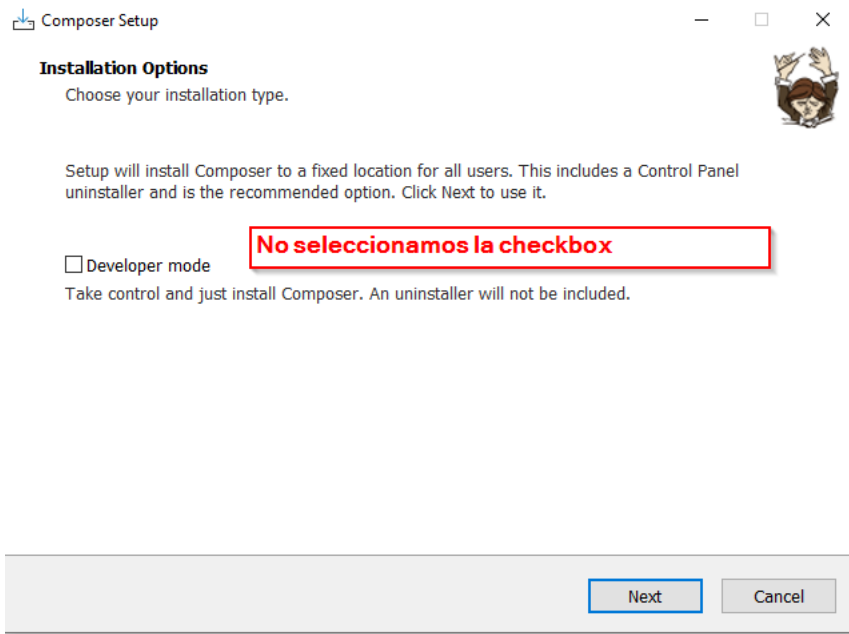


3.3. Instalación de Composer

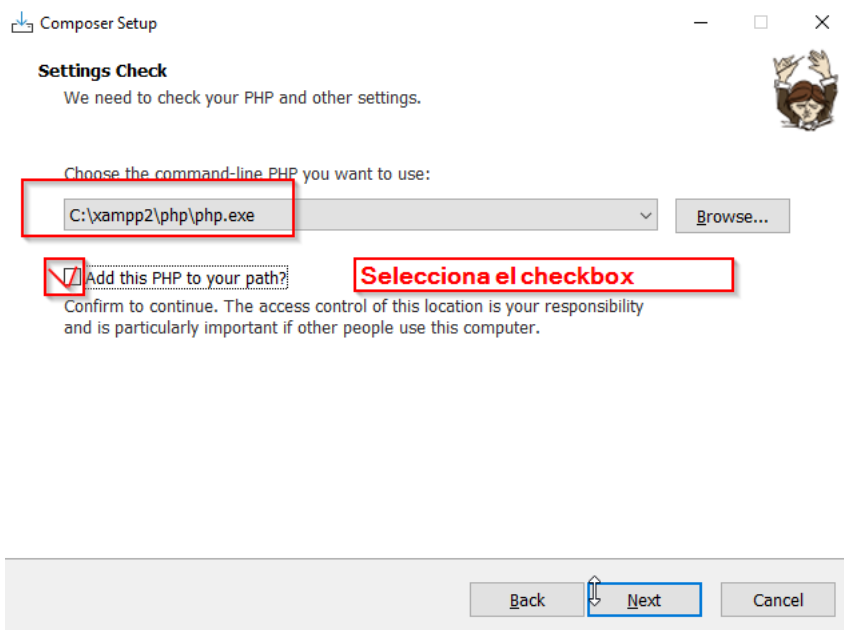
El [método recomendado](#) para instalar la librería de PHP para MongoDB es a través de [Composer](#), un gestor de dependencias.

1. Descargar [Composer-Setup.exe](#).

2. Instalamos para todos los usuarios **sin** **chequear developer mode**



3. Seleccionamos la ruta a php.exe dentro de xampp. **¡Atención!** En la imagen se ve **xampp2**, pero si habéis hecho una instalación por defecto, en el aula, deberíais tener un directorio **xampp** en lugar de xampp2. Seleccionamos la checkbox para añadir el ejecutable php al PATH (Si ya está en el path por la variable de entorno que editamos cuando instalamos PHP Server, es posible que no os aparezca el checkbox)



Al pasar al siguiente paso, es posible que obtengáis este error:

installing composer Program Output: Xdebug: [Step Debug] Time-out connecting to debugging client, waited: 200 ms. Tried: localhost:9003 (through xdebug.client_host/xdebug.client_port).

Si no obtenéis el error, pasad directamente al paso 4. Si lo obtenéis, podéis probar a resolverlo comentando solo durante la instalación de composer la línea que figura en el php.ini coo 2029 (podría variar el nº de línea en vuestro php.ini):

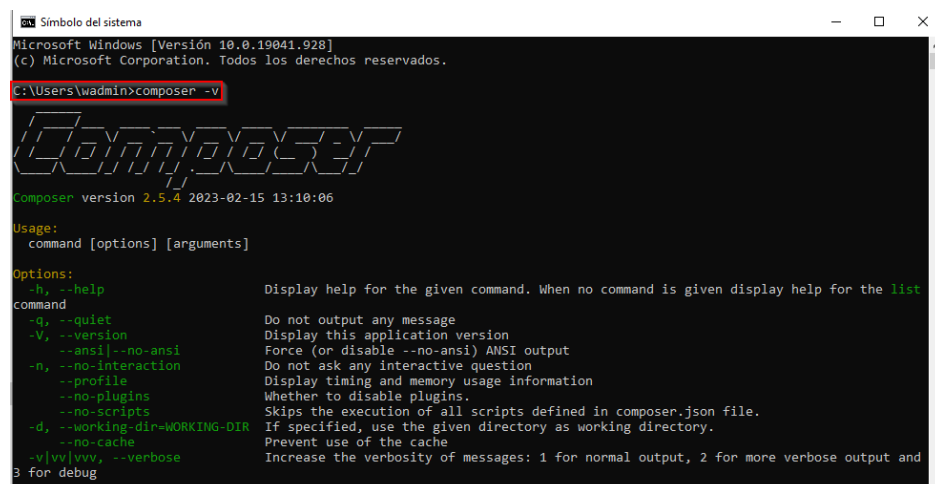
```

2026 [Xdebug]
2027 zend_extension = xdebug
2028 xdebug.mode = debug
2029 ;xdebug.start_with_request = yes
2030

```

Al finalizar la instalación, descomentad de nuevo la línea 2029.

4. Cuando termine el instalador, abrimos una ventana de comandos y tecleamos `composer -v` para comprobar que se detecta la versión instalada de composer.



```

Microsoft Windows [Versión 10.0.19041.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\wadmin>composer -v

Composer version 2.5.4 2023-02-15 13:10:06

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display help for the given command. When no command is given display help for the list
  -q, --quiet               Do not output any message
  -V, --version              Display this application version
  --ansi|--no-ansi          Force (or disable --no-ansi) ANSI output
  -n, --no-interaction       Do not ask any interactive question
  --profile                 Display timing and memory usage information
  --no-plugins               Whether to disable plugins.
  --no-scripts              Skips the execution of all scripts defined in composer.json file.
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  --no-cache                Prevent use of the cache
  -v|vv|vvv, --verbose      Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and
                             3 for debug

```

3.4. Instalación de la librería MongoDB para PHP

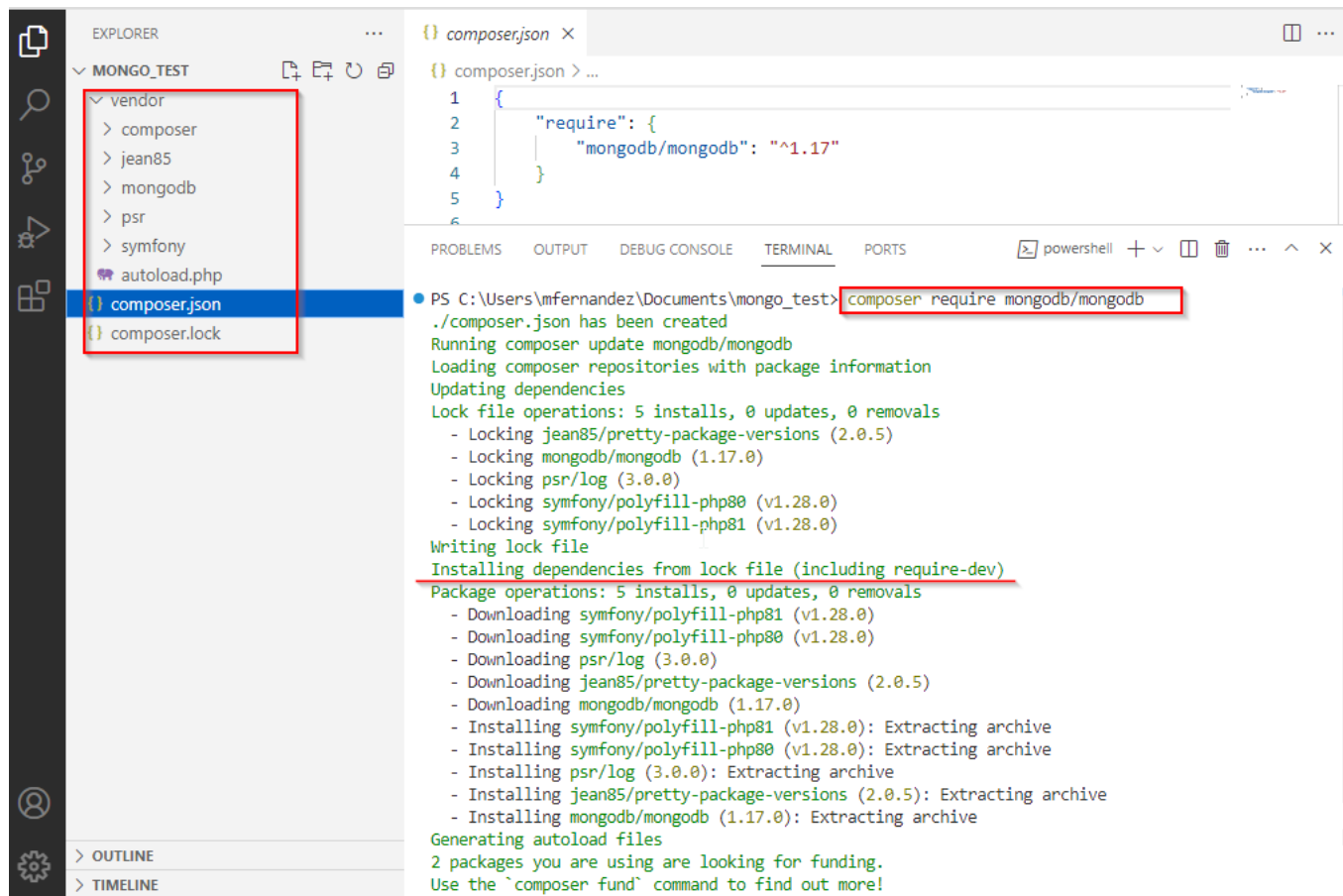
Vamos a crear una nueva carpeta llamado **Mongo_TEST** y la abrimos con Visual Studio Code.

Una vez instalado Composer, desde el terminal integrado en VS Code, según [la documentación de MongoDB](#), debemos ejecutar:

composer require mongodb/mongodb

Con ello, habremos instalado la librería MongoDB. Solo será necesario hacerlo la primera vez.

La instalación ha creado varios archivos y carpetas en nuestro proyecto:



- **composer.json**: Un fichero en formato JSON que indica las dependencias del proyecto. Se sitúa normalmente en la raíz del proyecto.

```
{
  "require": {
    "mongodb/mongodb": "^1.17"
  }
}
```

- **composer.lock**: Composer apunta en el archivo **composer.lock** la versión exacta que se ha instalado de cada librería. De esta forma, el proyecto se fija a unas determinadas versiones. Se recomienda que forme parte del sistema de control de versiones (VCS)
- la carpeta **vendor**: la ubicación por convención para todo el código de terceros en un proyecto.

Composer facilita un sistema de **autoload** para cada librería que facilite ese tipo de información. * Para ello crea un fichero **vendor/autoload.php** que se integra con la estructura de dependencias que gestiona Composer, facilitando así el desarrollo.

Para usar las clases de MongoDB, simplemente tendremos que incluir en nuestros archivos .php:

```
require_once __DIR__ . '/vendor/autoload.php';
```

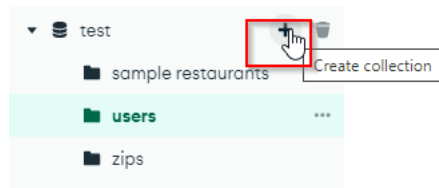
*PHP cuenta también con un sistema de [autoload o de carga automática de clases](#) que facilitan la dependencia de otros ficheros php sin tener que escribir una larga lista de include o require. Lo veremos con más calma cuando veamos Programación Orientada a Objetos con PHP. Por el momento, nos basta con saber que para usar la librería de mongodb, tendremos que usar

```
require_once __DIR__ . '/vendor/autoload.php';
```


3.5. Creación de colecciones de ejemplo

Vamos a crear un par de colecciones de ejemplo en la BD test con la interfaz gráfica de MongoDB Compass:

1- Sobre la BD test, hacemos clic sobre el icono +:

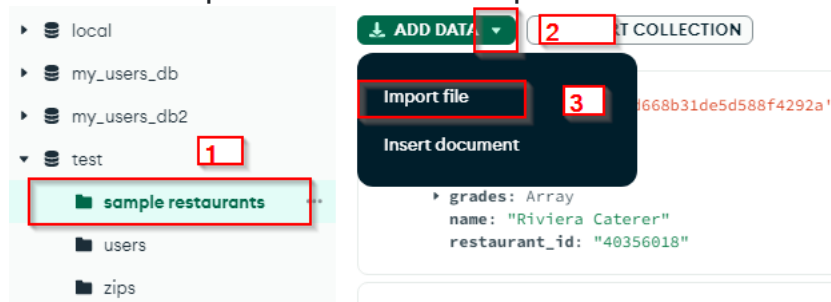


Creamos las colecciones:

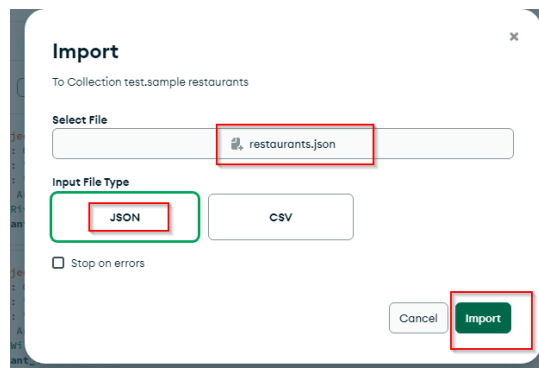
- a) sample restaurants
- b) zips

2- Descargamos los ficheros de la carpeta: [MongoDB: Colecciones de ejemplo](#)

3- Seleccionamos **sample restaurants** > **Add Data** > **Import file**



4- Seleccionamos **restaurants.json** > formato **JSON** e importamos



5- Repetimos los pasos 3 y 4 para la colección **zips** y archivo **zips.json**

3.6. Operaciones CRUD: Create

Vamos a seguir la documentación del [manual de la librería de PHP para MongoDB](#) (la URL está disponible también en el apartado de **Recursos**) en el apartado de **CRUD Operations**. Siguiendo ese mismo manual, tenéis disponible un proyecto basado en los ejemplos en el repositorio

<https://github.com/dudwcs/Ejemplos-MongoDB.git>

Una vez clonado, desde el terminal, habrá que forzar la creación de la carpeta vendor con:

composer install

```
require_once __DIR__ . '/vendor/autoload.php';

PS C:\Users\maria\Documents\Curso Drive\mariaferroteis\Curso 23-24\Dual\
UD4\MongoDB\MongodB-Test> composer install
Xdebug: [Step Debug] Time-out connecting to debugging client, waited: 20
0 ms. Tried: localhost:9003 (through xdebug.client_host/xdebug.client_po
rt).
Installing dependencies from lock file (including require-dev)
Verifying lock file contents can be installed on current platform.
Package operations: 5 installs, 0 updates, 0 removals
- Failed to download symfony/polyfill-php81 from dist: The zip extensi
on and unzip/7z commands are both missing, skipping.
The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
Now trying to download from source
- Failed to download psr/log from dist: The zip extension and unzip/7z commands are both missi
ng, skipping.
The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
Now trying to download from source
- Syncing psr/log (3.0.0) into cache
- Failed to download jean85/pretty-package-versions from dist: The zip extension and unzip/7z
commands are both missing, skipping.
The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
Now trying to download from source
- Syncing jean85/pretty-package-versions (2.0.5) into cache
- Failed to download mongodb/mongodb from dist: The zip extension and unzip/7z commands are bo
th missing, skipping.
The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
Now trying to download from source
- Syncing mongodb/mongodb (1.17.0) into cache
- Installing symfony/polyfill-php81 (v1.28.0): Cloning 7581cd600f from cache
- Installing symfony/polyfill-php80 (v1.28.0): Cloning 6caa57379c from cache
- Installing psr/log (3.0.0): Cloning fe5ea303b0 from cache
- Installing jean85/pretty-package-versions (2.0.5): Cloning ae547e455a from cache
- Installing mongodb/mongodb (1.17.0): Cloning 9d9c917cf7 from cache
Generating autoload files
2 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
```

La primera vez que ejecutamos **composer install** en un proyecto, Composer lee el fichero **composer.json**, resuelve las dependencias que hay en él e instala los paquetes en el directorio **vendor**. Las próximas veces que se ejecute **composer install** en dicho proyecto, Composer leerá ese fichero e instalará aquellos paquetes que aparezcan en el fichero y no se encuentren en el directorio **vendor**.

El comando **composer install** debe ser ejecutado desde el directorio raíz de la aplicación.

Obtención de una conexión a través de un cliente:

Usamos el constructor de [MongoDB\Client](#)

function __construct(?string \$uri = null, array \$uriOptions = [], array \$driverOptions = [])

Si no se le envía ningún parámetro es equivalente a pasarle por parámetro:

```
"mongodb://127.0.0.1:27017"
$client = new MongoDB\Client("mongodb://127.0.0.1:27017");
```

es equivalente a:

```
$client = new MongoDB\Client();
```

Operaciones create

MongoDB\Collection::insertOne()

- Si no se aporta el campo **_id**, se creará uno automáticamente que podéis recuperar con **\$insertOneResult->getInsertedId()**. Será de un tipo de datos **MongoDB\BSON\ObjectId**
- Si se aporta el campo **_id**, MongoDB comprobará si ya existe un documento con ese **_id**. Si existe lanzará una excepción **MongoDB\Driver\Exception\BulkWriteException: E11000 duplicate key error collection:**
- Se puede conocer el nº de documentos insertados con **insertOneResult->getInsertedCount()**

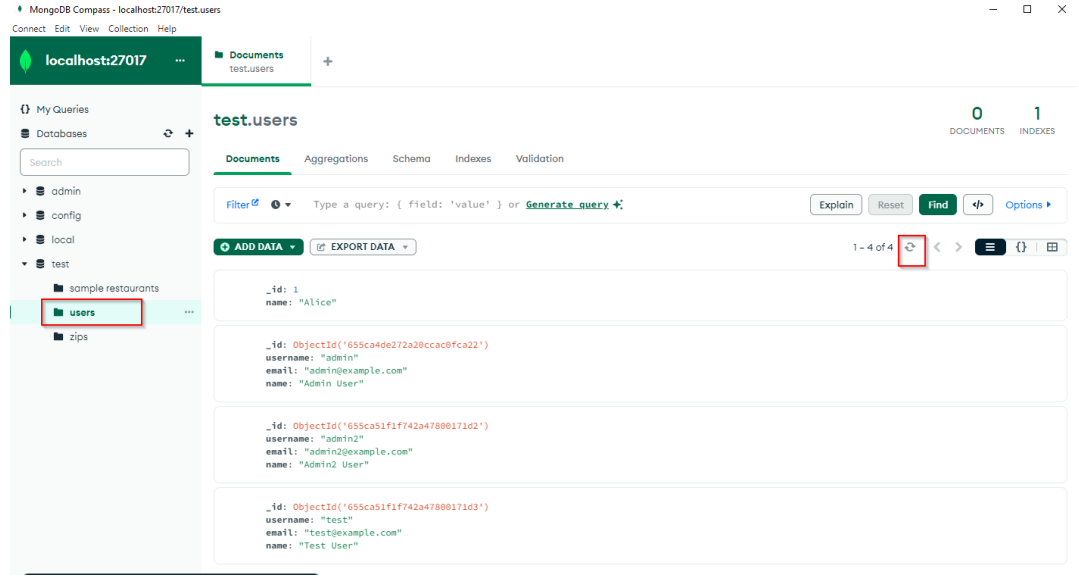
MongoDB\Collection::insertMany()

Se comporta de forma similar a **insertOne**, pero permite insertar más de un documento a la vez

Se pueden ver ejemplos en los ficheros del repositorio:

- insertOne.php
- insertOne_id.php
- insertMany.php

Se puede comprobar el resultado de la ejecución de los 3 en la colección users:

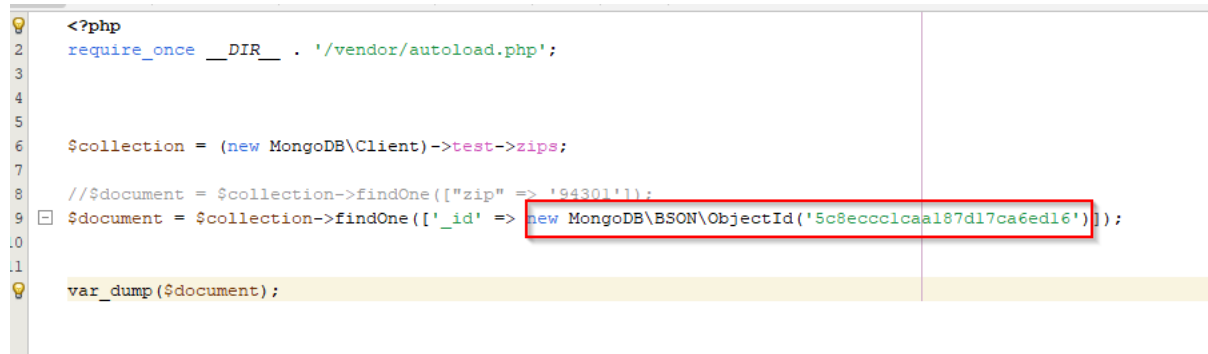


3.7. Operaciones de consulta

MongoDB\Collection::findOne()

Devuelve el **primer documento que encaje con el filtro** indicado o **null** en otro caso.

- ¡**Atención!** En la documentación el filtro por id no va a funcionar usando un simple string. Debemos usar [MongoDB\BSON\ObjectId](#):



```
<?php
2 require_once __DIR__ . '/vendor/autoload.php';
3
4
5
6 $collection = (new MongoDB\Client)->test->zips;
7
8 // $document = $collection->findOne(["zip" => '94301']);
9 $document = $collection->findOne(['_id' => new MongoDB\BSON\ObjectId('5c8ecc1ca187d17ca6ed16')]);
10
11
12 var_dump($document);
```

- La documentación nos advierte que **no es lo mismo usar 94301 que '94301'** porque los tipos de datos utilizados en MongoDB distinguen entre enteros y cadenas de texto (entre otros)

MongoDB\Collection::find()

- Devuelve un [MongoDB\Driver\Cursor](#) con los documentos que encajan con los criterios establecidos. El cursor permite tratar como un array asociativo los documentos.
- Proyecciones sobre consultas:**

Por defecto, se devuelven todos los campos de un documento.

Es posible seleccionar solo algunos campos con una proyección (como en un select de SQL) con el método **find** con la opción **projection**, indicando en un array asociativo los campos que deseamos como clave y como valor 1.

El campo **_id** se devuelve siempre salvo que se excluya explícitamente con valor 0

Ordenación y paginación

Se permite obtener una serie de campos (o todo el documento) ordenados por uno o más criterios, escogiendo cuantos elementos obtener y empezando en un índice concreto.

- sort:** Se indican los campos por los que ordenar el resultado: Con un 1 se ordena ascendentemente y con -1 descendentemente.
- skip:** El número de elementos que se saltarán. Por ejemplo si hay 20 códigos postales, si queremos mostrarlos en páginas de 5 elementos en cada página, para mostrar la 2ª página, tendremos que saltar los 5 primeros
- limit:** El máximo número de documentos que formará parte del resultado.

Se pueden ver ejemplos en los ficheros del repositorio:

- find.php
- findProjection.php
- findProjection_sin_id.php

3.8. Operaciones update

MongoDB\Collection::updateOne()

- Actualiza **el primero de los documentos** que encaje con un filtro.
- Requiere 2 parámetros obligatorios:
 - El filtro de selección
 - El operador \$set con los campos a actualizar y sus valores
- **\$updateResult->getMatchedCount()** devuelve el nº de documentos que encajaron con el filtro
- **\$updateResult->getModifiedCount()** devuelve el nº de documentos actualizados

Si una operación de actualización/reemplazo no produce ningún cambio en el documento (por ejemplo, establece el valor de un campo en su valor actual), el valor devuelto por **getMatchedCount()** puede ser mayor que el valor devuelto por **getModifiedCount()**.

MongoDB\Collection::updateMany()

- Similar a **updateOne**, pero puede modificar más de un documento.

La opción **upsert**

Tanto **updateOne** como **updateMany** permiten usar la opción boolean **upsert**. Esta opción permite insertar si no existen documentos que encajen con el filtro o actualizar los documentos que sí encajan con el filtro.

MongoDB\Collection::replaceOne()

- Similar a **UpdateOne**, pero en lugar de modificar algunos campos modifica el documento entero **respetando el _id**

Se pueden ver ejemplos en los ficheros del repositorio:

- `updateOne.php`
- `updateOne_v2.php`
- `updateMany.php`
- `upsert.php`
- `replaceOne.php`

3.9. Operaciones delete

[MongoDB\Collection::deleteOne\(\).](#)

Permite eliminar un único documento (**el primero que encaje**) con un filtro

[MongoDB\Collection::deleteMany\(\).](#)

Permite eliminar todos los documentos que encajen con un filtro.

En ambos casos se puede obtener el nº de elementos eliminados con `$deleteResult->getDeletedCount()`

Se pueden ver ejemplos en los ficheros del repositorio:

- `deleteOne.php`
- `deleteMany.php`