

1. INTRODUCCIÓN.

En la UT1 (selección de arquitecturas y herramientas de programación) se explicaron las arquitecturas web; utilizar una arquitectura web basada en capas, consistía en dividir el software (el código) de una aplicación, en partes, atendiendo a un criterio claro; ese criterio es establecer la división del código en base a la función de las componentes software de cada capa:

- **Presentación**
- **Lógica de negocio**
- **Datos o persistencia.**

Con lo visto hasta ahora en la tecnología de desarrollo elegida para nuestras aplicaciones web (PHP) y el repaso realizado a MySQL, disponemos de las herramientas para poder diseñar aplicaciones web estructuradas en tres capas.

Con este documento se pretende completar y actualizar los contenidos de la UT6 del libro de texto, desarrollaremos en profundidad la capa de datos o persistencia utilizando las herramientas que PHP nos proporciona para ello. En la actividadUT6 guiada, vimos cómo utilizar una de las herramientas de PHP, en concreto las funciones disponibles en la **extensión php_mysqli** y con ellas, desde PHP, acceder a los datos de una BD gestionada por MySQL. Sin embargo, y tal y como se adelantó en dicha actividad, no es la única herramienta disponible para ello ni tampoco los datos deben pertenecer únicamente a una BD MySQL.

Desde PHP es posible acceder a las BBDD gestionadas por muchos gestores diferentes (SQLite, SQL Server, PostgreSQL,...) utilizando la **extensión específica para cada gestor**. A partir de la versión 5 de PHP, surge una nueva librería, **PDO (PHP Object Data)** que permite utilizar una sintaxis común para trabajar con cualquier BD desde PHP. (decir, que las extensiones nativas de cada gestor de BD ofrecen funciones específicas de un gestor que no se incluyen en PDO).

Decir también, que aunque lo más frecuente es acceder a los datos de una BD, también se puede acceder a los datos almacenados en ficheros o incluso, que desde la BD podamos acceder a ficheros.

2. HERRAMIENTAS PARA EL MANEJO DE LOS DATOS ALMACENADOS EN UNA BD.

2.1.– La librería mysqli (<https://www.php.net/manual/es/book.mysqli.php>)

Con esta librería es posible:

- Disponer de un conjunto de funciones para acceder y trabajar con una BD MySQL (*interfaz procedimental*) o bien disponer de métodos y atributos, propios de las clases, y utilizar así la POO.
- Gestionar transacciones.
- Gestionar consultas preparadas.
- Gestionar posibles errores (*control de errores tradicional*)

Algunas de las funciones disponibles en la API o librería *mysqli* y manejadas en la actividad guiada son:

- **mysqli_connect():** establece la conexión con el servidor MySQL.(se obtiene el manejador de la conexión)
- **mysqli_select_db():** se pone en uso la BD especificada, luego la conexión se establece con esa BD.
- **mysqli_query():** los parámetros que necesita son la conexión y la operación a realizar especificada en una sentencia SQL. Está función dependiendo el tipo de operación realizada, puede que interese almacenar el resultado en alguna variable (resultset).
- **mysqli_fetch_array():** en muchas ocasiones, tal es el caso de una consulta, el resul set no será una única tupla y interesará recorrerlas. Las tuplas o filas devuelvas, serán tratadas como un array que después se recorrerá o bien como un array indexado o bien como un array asociativo.
- **mysqli_fetch_row():** el *resultset* será tratado como un array indexado.
- **mysqli_fetch_assoc():** el *resultset* será tratado como un array asociativo.
- **mysqli_close():** necesita como parámetro la conexión para cerrarla.

Esta API, también cuenta con la **clase mysqli** (<https://www.php.net/manual/es/class.mysqli.php>) para que utilizando, atributos y métodos, también podamos acceder a los datos de una BD MySQL (se dice por ello que la extensión o librería *mysqli* ofrece un interfaz de programación dual). Su manejo es sencillo.

Resumen de las funciones de la extensión MySQLi

Resumen de los métodos de `mysqli`

Funciones manejadas en la actividad guiada

Clase <code>mysqli</code>	Interfaz de procedimiento	Alias (no usar)	Descripción
<i>Propiedades</i>			
<code>\$mysqli->affected_rows</code>	<code>mysqli_affected_rows()</code>	N/A	Obtiene el número de filas afectadas en una operación previa de MySQL
<code>\$mysqli->client_info</code>	<code>mysqli_get_client_info()</code>	N/A	Devuelve la versión del cliente de MySQL como una cadena
<code>\$mysqli->client_version</code>	<code>mysqli_get_client_version()</code>	N/A	Devuelve la versión del cliente de MySQL como un entero
<code>\$mysqli->connect_errno</code>	<code>mysqli_connect_errno()</code>	N/A	Devuelve el código de error de la última llamada de conexión
<code>\$mysqli->connect_error</code>	<code>mysqli_connect_error()</code>	N/A	Devuelve una cadena descriptiva del último error de conexión
<code>\$mysqli->errno</code>	<code>mysqli_errno()</code>	N/A	Devuelve el código de error para la función invocada más reciente
<code>\$mysqli->error</code>	<code>mysqli_error()</code>	N/A	Devuelve una cadena descriptiva del último error

Extracto obtenido de <https://www.php.net/manual/es/mysqli.summary.php>

Nota: Al igual que en otras ocasiones, desde el fichero `php.ini`, existen directivas configurables que afectan al funcionamiento de la extensión `mysqli`.

Utilizando el ejemplo de la práctica guiada, la codificación sería la siguiente:

```
<?php
$db_host="localhost";
$db_usuario="root";
$db_clave="";
$db_nombre="ciclo";

//para establecer una conexión hay que instanciar la clase mysqli
$conexion= new mysqli($db_host,$db_usuario,$db_clave,$db_nombre);

//para comprobar si se ha establecido la conexión
//se puede utilizar el atributo connect_errno de la clase
if ($conexion->connect_errno)
{
    echo "Fallo en la conexión#243n <br>";
    exit();
}
else
    echo "conexión#243n establecida <br>";

//para realizar una consulta, se crea la consulta y se invoca al método query de la clase
$consulta="select * from alumno";
$resultados=$conexion->query($consulta);

//el resultado obtenido en un objeto y que se almacena en $resultados;
//para su recorrido, se utiliza el método fetch_array por ejemplo
while ($fila=$resultados->fetch_array())
    echo $fila[0]. " ". $fila[1]. " ". $fila[2]. " ". $fila[3]. "<br>";

//para cerrar la conexión y liberar recursos, se utiliza el método close
$conexion->close();
?>
```

conexión establecida
1 Ana 18
2 Sergio 18
3 Jorge 21
4 belen 22
5 maria 24
6 Felipe 23

Salida obtenida

ACTIVIDAD 1: Consultar el enlace (<https://www.php.net/manual/es/class.mysql.php>)

ACTIVIDAD 2: ¿Qué añadirías a la codificación anterior para indicarle al servidor MySQL que utilice UTF-8 como sistema de codificación de los datos transmitidos?

ACTIVIDAD 3: Puedes realizar las actividades incluidas en la práctica guiada pero utilizando la clase mysqli.

2.1.1.- Consultas.

La **función mysqli_query** o el **método query**.

- Si se ejecutan consultas que no devuelven resultados (ej. **INSERT**, **DELETE** o **UPDATE**) la función o el método, devuelve true o false en función de si la consultada se ha realizado con éxito o no.
- Si se ejecutan consultas que sí devuelven resultados (ej. **SELECT**) la función o el método, devuelve un objeto de la clase *mysqli_result* (<https://www.php.net/manual/es/class.mysql-result.php>, clase dual), siendo necesaria la utilización de alguno de los métodos de esa clase para su tratamiento, como el utilizado en los ejemplos, `fetch_array()`.

Nota: query, puede recibir un parámetro opcional: `int $resultmode`, siendo sus posibles valores `MYSQLI_STORE_RESULT` o `MYSQLI_USE_RESULT`; por defecto el valor es `MYSQLI_STORE_RESULT` ya que con él el resultado de la consulta se almacena de forma local en el cliente desde el cual se ejecuta la consulta para después ser procesado.

La **función mysqli_free** o el **método free**.

Es interesante liberar la memoria ocupada por un resultado una vez que deja de ser utilizado.

ACTIVIDAD 4: Modifica los datos de los alumnos de la BD ciclo de tal forma que los alumnos con 18 años ahora tengan 19. Visualiza cuántos alumnos has modificado (función `mysqli_affected_rows()` o método).

ACTIVIDAD 5: Visualiza el nombre de los alumnos que tengan 19 años.

2.1.2.- Consultas preparadas o parametrizadas.

En muchas ocasiones será necesario realizar una misma consulta varias veces con diferentes datos (por ejemplo insertar nuevas tuplas) o realizar una consulta donde se seleccionen tuplas de una tabla donde la selección dependa de ciertos parámetros (entradas desde un formulario, resultados de otras ejecuciones,...). Para este tipo de consultas, PHP dispone de nuevo de varias opciones con las que crear consultas preparadas o parametrizadas.

La base del funcionamiento de este tipo de consultas es la **clase mysqli_stmt** (<https://www.php.net/manual/es/class.mysql-stmt.php>) que aportará los correspondientes métodos para su tratamiento, siendo necesario que, tras establecer la conexión con la BD se cree un objeto de esta clase invocando a la función `mysqli_stmt_init()` o al método `stmt_init()` de la **clase mysqli**.

Ejemplo de consulta preparada:

```
//consulta preparada o parametrizada
$edad=18;
$preparada=$conexion->stmt_init(); //crear un objeto de la clase mysqli_stmt para la conexión establecida
$preparada->prepare('SELECT nombre FROM alumno WHERE edad=?'); //preparar la consulta parametrizada (uso de ?)utilizando el metodo prepare
$preparada->bind_param('i',$edad); //establecer los tipos de los parámetros de la consulta (en este caso 'i' de tipo int que es edad
$preparada->execute(); //ejecutar la consulta
$preparada->bind_result($nom); //obtener los resultados de la consulta y como son varios y será preciso un tratamiento, utilizar bind_result
echo "Estos son los alumnos con ".$edad. " años: <br>";
while ($preparada->fetch()) //fetch, permitirá recorrer el resultado
    echo $nom."<br>";
$preparada->close(); //liberar recursos consumidos con el objeto creado en esta consulta
//fin consulta preparada
```

conexión establecida
1 Ana 18
2 Sergio 18
3 Jorge 21
4 belen 22
5 maria 24
6 Felipe 23
Estos son los alumnos con 18 años:
Ana
Sergio

ACTIVIDAD 6: Diseña una consulta para mostrar el código del alumno (id_al) de aquellos cuyo nombre sea Ana y tengan 18 años.

2.1.3.- Transacciones.

Una **transacción** es una **secuencia de operaciones** realizadas como una sola unidad lógica de trabajo, caracterizada por cuatro propiedades que son:

- **Atomicidad:** es una unidad de trabajo en su totalidad, se lleven a cabo o no las modificaciones correspondientes en la BD.
- **Coherencia:** debe mantener la integridad de la BD.
- **Aislamiento:** las transacciones deben aislarse de otras transacciones concurrentes. (*seriabilidad*)
- **Durabilidad:** Concluida una transacción, las modificaciones en la BD son permanentes.

Por ello, SQL proporciona mecanismos que aseguren la integridad física de cada transacción, proporcionando:

- Servicio de bloqueo para garantizar el aislamiento de las transacciones.
- Servicio de registro para garantizar la durabilidad de las transacciones.
- **Servicio de administración de transacciones** para garantizar la atomicidad y coherencia de una transacción. Es precisamente en este servicio en el que **nos vamos a centrar en este apartado** para el cual el SGBD MySQL ofrece dos sentencias: **COMMIT y ROLLBACK**.
- **COMMIT**, es una sentencia que permite confirmar las modificaciones asociadas a la ejecución de sentencias sobre una BD en disco; es decir, si se ejecuta un INSERT, el COMMIT sería la sentencia que confirmara los cambios en la BD almacenada en un soporte como un disco.
- **ROLLBACK**, es una sentencia que permite deshacer o menor no hacer efectivos en disco los cambios en una BD tras la ejecución de sentencias tras el último COMMIT; es decir, si se ejecuta un INSERT, el ROLLBACK evita que la información se inserte físicamente en la BD, es decir en el disco.

Teniendo en cuenta la función de ambas sentencias nunca hemos ejecutado COMMIT tras realizar una sentencia SQL porque por defecto el **COMMIT es automático** (ello se debe a que existe una variable, AUTOCOMMIT cuyo valor es 1; **si le asignamos valor 0 (SET AUTOCOMMIT=0)** entonces será necesario ejecutar COMMIT para que los cambios en la BD sean efectivos). Si COMMIT es automático, ROLLBACK no se puede ejecutar.

Un ejemplo:

```
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO socios
-> VALUES (1015,'XXXXX','981918991','2005-01-01','KAJDAJKJ',11111);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM SOCIOS;
+-----+-----+-----+-----+-----+-----+
| socio_no | apellidos | telefono | fecha_alta | direccion | codigo_postal |
+-----+-----+-----+-----+-----+-----+
| 1001 | PEREZ CERRO | 918451256 | 2005-01-01 | C. MAYOR 31 | 22222 |
| 1003 | ROMA LEIUA | 918451256 | 2005-01-01 | C. PANADEROS 9 | 28431 |
| 1004 | GOMEZ DURUELO | 918654329 | 2005-01-01 | C. REAL 15 | 22222 |
| 1005 | PEÑA MAYOR | 918515256 | 2005-01-01 | C. LARGA 31 | 28431 |
| 1006 | JHJHJKH | 98989898 | 2000-01-01 | GKGHGHJ | 1 |
| 1008 | LOPEZ PEREZ | 916543267 | 2005-01-01 | C. REAL 5 | 22222 |
| 1015 | XXXXX | 981918991 | 2005-01-01 | KAJDAJKJ | 11111 |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT * FROM SOCIOS;
+-----+-----+-----+-----+-----+-----+
| socio_no | apellidos | telefono | fecha_alta | direccion | codigo_postal |
+-----+-----+-----+-----+-----+-----+
| 1001 | PEREZ CERRO | 918451256 | 2005-01-01 | C. MAYOR 31 | 22222 |
| 1003 | ROMA LEIUA | 918451256 | 2005-01-01 | C. PANADEROS 9 | 28431 |
| 1004 | GOMEZ DURUELO | 918654329 | 2005-01-01 | C. REAL 15 | 22222 |
| 1005 | PEÑA MAYOR | 918515256 | 2005-01-01 | C. LARGA 31 | 28431 |
| 1006 | JHJHJKH | 98989898 | 2000-01-01 | GKGHGHJ | 1 |
| 1008 | LOPEZ PEREZ | 916543267 | 2005-01-01 | C. REAL 5 | 22222 |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Y si queremos que la inserción se realice de forma permanente en la base de datos:

```
mysql> INSERT INTO socios
-> VALUES (1015,'XXXXX','981918991','2005-01-01','KAJDAJKJ',11111);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM socios;
+-----+-----+-----+-----+-----+-----+
| socio_no | apellidos | telefono | fecha_alta | direccion | codigo_postal |
+-----+-----+-----+-----+-----+-----+
| 1001 | PEREZ CERRO | 918451256 | 2005-01-01 | C. MAYOR 31 | 22222 |
| 1003 | ROMA LEIUA | 918451256 | 2005-01-01 | C. PANADEROS 9 | 28431 |
| 1004 | GOMEZ DURUELO | 918654329 | 2005-01-01 | C. REAL 15 | 22222 |
| 1005 | PEÑA MAYOR | 918515256 | 2005-01-01 | C. LARGA 31 | 28431 |
| 1006 | JHJHJKH | 98989898 | 2000-01-01 | GKGHGHJ | 1 |
| 1008 | LOPEZ PEREZ | 916543267 | 2005-01-01 | C. REAL 5 | 22222 |
| 1015 | XXXXX | 981918991 | 2005-01-01 | KAJDAJKJ | 11111 |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.13 sec)

mysql> SELECT * FROM socios;
+-----+-----+-----+-----+-----+-----+
| socio_no | apellidos | telefono | fecha_alta | direccion | codigo_postal |
+-----+-----+-----+-----+-----+-----+
| 1001 | PEREZ CERRO | 918451256 | 2005-01-01 | C. MAYOR 31 | 22222 |
| 1003 | ROMA LEIUA | 918451256 | 2005-01-01 | C. PANADEROS 9 | 28431 |
| 1004 | GOMEZ DURUELO | 918654329 | 2005-01-01 | C. REAL 15 | 22222 |
| 1005 | PEÑA MAYOR | 918515256 | 2005-01-01 | C. LARGA 31 | 28431 |
| 1006 | JHJHJKH | 98989898 | 2000-01-01 | GKGHGHJ | 1 |
| 1008 | LOPEZ PEREZ | 916543267 | 2005-01-01 | C. REAL 5 | 22222 |
| 1015 | XXXXX | 981918991 | 2005-01-01 | KAJDAJKJ | 11111 |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

Con este breve repaso, decir también que, la gestión de transacciones en MySQL dependerá del motor de almacenamiento elegido para la gestión de las tablas; MyISAM, que es el motor de almacenamiento por defecto, no soporta transacciones; sin embargo InnoDB, sí y por ello será necesario tu uso para este apartado.

Para la gestión de las transacciones, de nuevo php a través de la librería mysqli, ofrece una serie de funciones o métodos para su gestión (de nuevo hay dos formas, procedimental u orientada a objetos).

Por ejemplo:

```
//gestión transacciones

$conexion->autocommit(false); //similar a asignar a la variable AUTOCOMMIT=0
$conexion->query('DELETE FROM ALUMNO WHERE edad=21');
//$conexion->rollback();
$conexion->commit();

//fin gestión transacciones
```

conexión establecida
1 Ana 18
2 Sergio 18
4 belen 22
5 maria 24
6 Felipe 23
Estos son los alumnos con 18 años:
Ana
Sergio

Salida obtenida (con respecto al paso anterior ya no está Jorge)

ACTIVIDAD 7: Plantea una pequeña transacción a ejecutar en tu BD para comprobar el funcionamiento de los métodos o funciones asociadas a ellas.

2.1.4.- Tratamiento de errores.

Para el tratamiento de errores, es posible manejar o bien métodos o funciones, o bien atributos.

ACTIVIDAD 8: Consulta la documentación de la clase mysqli de PHP para el tratamiento de los errores y plantea una forma diferente de controlar el error en una conexión.

2.2.- La librería PDO.

Esta librería (<https://www.php.net/manual/es/class.pdo>) cuenta con las herramientas necesarias para acceder a los datos de una BD cualquiera y no sólo gestionada por MySQL; es decir, ofrece un interfaz común para el manejo de cualquier BD, tanto relacionales (ej: MySQL, SQL Server, Oracle,...) como no relacionales (ej: MongoDB).

Además utilizando PDO, el control de errores es implementado a través del manejo de excepciones. La clase PDO dispone de un atributo (ATTR_ERRMODE) que es utilizado para ello. Se debe invocar al método `setAttribute` de esta clase para la gestión de los errores utilizando excepciones.

Siguiendo con nuestro ejemplo, utilizando la PDO, quedaría así:

```
<?php
$db_usuario="root"; $db_clave="";
try
{
    /*para establecer una conexión hay que instanciar la clase PDO
    el constructor requiere de tres parámetros:
        el primero es el driver para establecer la conexión indicando:
            la ubicación de la BD(host=127.0.0.1:3308, ese es mi puerto)
            y el nombre de la DB (dbname)
        el segundo el usuario y el tercero el password */
    // el establecimiento de conexión también se puede poner como se indica en el comentario anterior
    $conexion= new PDO('mysql:host=127.0.0.1; port=3308; dbname=ciclo',$db_usuario,$db_clave);
    //probar el código con y sin la siguiente sentencia y un error (tabla alumna) o un puerto erroneo
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Conexión establecida con éxito utilizando PDO <br><br>";

    /*para realizar una consulta, se crea la consulta y se invoca al método query que
    Ejecuta una sentencia SQL, devolviendo un conjunto de registros como un objeto PDOStatement*/
    $consulta="select * from alumno";
    $resultados=$conexion->query($consulta);

    //el método query devuelve un objeto de tipo PDOStatement si no ha habido un error
    // para su tratamiento se necesitan los métodos de esa clase de objetos
    //OPCIÓN A) DE TRATAMIENTO DEL RESULTADO (sin utilizar métodos específicos de la clase PDOStatement
    echo "Sin métodos específicos de un objeto PDOStatement <br>";
    foreach ($resultados as $fila)
    {
        echo $fila['id_al']." ".$fila['nombre']." ".$fila['edad']." ".$fila['codigo']."<br>";
    }
    $resultados->closeCursor(); /*si es un método específico de la clase PDOStatement
    que permite volver a utilizar $resultados para repetir o realizar
    otra consulta*/

    // OPCIÓN B) DE TRATAMIENTO DEL RESULTADO (utilizando métodos específicos de la clase PDOStatement
    $resultados=$conexion->prepare($consulta); //prepare es un método de la clase PDO; sirve para preparar consultas.
    $resultados->execute(); //execute es un método de la clase PDOStatement, ejecuta la consulta
    echo "Con métodos específicos de un objeto PDOStatement <br>";
    while ($fila=$resultados->fetch(PDO::FETCH_ASSOC)) /*fetch es un método de la clase PDOStatement*/
    {
        echo $fila['id_al']." ".$fila['nombre']." ".$fila['edad']." ".$fila['codigo']."<br>";
    }
    $resultados->closeCursor();
}
catch (PDOException $e)
{
    die("Error con la base de datos: ".$e->getMessage());
}
finally
{
    $conexion=null; //liberar los recursos de la conexión
}
?>
```

Conexión establecida con éxito utilizando PDO

Sin métodos específicos de un objeto PDOStatement

1	Ana	18
2	Sergio	18
3	Jorge	21
4	belén	22
5	maria	24
6	Felipe	23

Con métodos específicos de un objeto PDOStatement

1	Ana	18
2	Sergio	18
3	Jorge	21
4	belén	22
5	maria	24
6	Felipe	23

Salida obtenida

ACTIVIDAD 4: Consultar el enlace <https://www.php.net/manual/es/class.pdo>

ACTIVIDAD 5: ¿Qué añadirías a la codificación anterior para indicarle al servidor MySQL que utilice UTF-8 como sistema de codificación de los datos transmitidos?

ACTIVIDAD 6: Puedes realizar las actividades incluidas en la práctica guiada pero utilizando la clase PDO.

2.2.1.- Consultas.

El método *query* de la clase PDO, de nuevo es el utilizado para este fin; este método devuelve false si hay algún error o un **objeto de la clase PDOStatement si es correcta la ejecución**. Luego, el resultado obtenido será tratada utilizando los métodos de esta nueva clase. (<https://www.php.net/manual/es/class.pdostatement.php>) como es el método *fetch()* del ejemplo anterior.

2.2.2.- Consultas preparadas o parametrizadas.

En concepto de consulta preparada ya le conocemos y en el ejemplo se aprecia su uso. Decir también que en este tipo de consultas además es frecuente el **uso de marcadores**. Cuando se ejecuta la consulta, el método *execute* recibirá los parámetros necesarios para ejecutar la consulta previamente preparada; puede recibirlos por posición o por nombre. En este segundo caso, hablamos de marcadores.

Un ejemplo:

El resultado de la consulta preparada sin marcadores donde el nombre es belen y el número es 4 es: 22

El resultado de la consulta preparada con marcadores donde la edad es 22 y el número es 4 es: belen

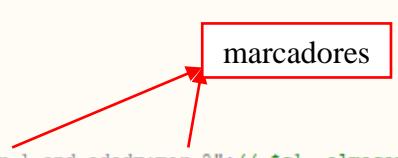
Salida obtenida

```
// consulta preparada sin marcadores (por posición)

$n=4;
$nombre="belen";
$c1="select edad from alumno where id_al=? and nombre=?"; // $c1, almacena la consulta nº 1
$r1=$conexion->prepare($c1); // $r1, almacena el resultado de la consulta nº 1
$r1->execute(array($n,$nombre));
echo "El resultado de la consulta preparada sin marcadores donde el nombre es ".$nombre." y el número es ".$n." es: ";
while ($fila=$r1->fetch(PDO::FETCH_ASSOC))
    echo $fila['edad']."<br><br>";
$r1->closeCursor();

// fin de consulta preparada sin marcadores

$n=4;
$edad=22;
$c2="select nombre from alumno where id_al=:mar_1 and edad=:mar_2"; // $c1, almacena la consulta nº 2
$r2=$conexion->prepare($c2); // $r2, almacena el resultado de la consulta nº 2
$r2->execute(array(':mar_1'=>$n, ':mar_2'=>$edad));
echo "El resultado de la consulta preparada con marcadores donde la edad es ".$edad." y el número es ".$n." es: ";
while ($fila=$r2->fetch(PDO::FETCH_ASSOC))
    echo $fila['nombre']."<br>";
$r2->closeCursor();
//consulta preparada con marcadores (por nombre)
```



ACTIVIDAD 7: Plantea alguna consulta parametrizada en tus BBDD con y sin marcadores.

2.2.3.- Transacciones.

De nuevo esta clase dispone de los métodos necesarios para su realización.

Ejemplo:

```
//transacciones con PDO
$conexion->beginTransaction(); //inicio de la transacción.
$insertar="insert into alumno (id_al,nombre,edad,codigo) values (10,'Monica',20,NULL)";
$r3=$conexion->query($insertar);
if (!$r3)
{
    echo "Error. ".$r3->errorinfo();
    $conexion->rollback();
    echo "<br>La transferencia no ha sido realizada.<br>";
}
else
    $conexion->commit();

$r3->closeCursor();

//fin de la transacción
```

1 Ana 18
2 Sergio 18
4 belen 22
5 maria 24
6 Felipe 23
10 Monica 20

Salida obtenida

ACTIVIDAD 8: Plantea alguna transacción en tus BBDD.

2.2.4.- Control de errores.

Como hemos visto anteriormente, la clase PDO tiene su propio controlador de excepciones siendo éste el mecanismo para el control de errores.