IES NTRA. SRA. DE LOS REMEDIOS DE UBRIQUE

Tema 05 – php y MySQL

2º DAW – Desarrollo Web Entorno Servidor

Profesor Juan Carlos Moreno

CURSO 2025/2026

Tabla de Contenido

6 Php y MySQL

6.1 Introducción

En este capítulo vamos a ver cómo gestionar la comunicación de estas aplicaciones web con una base de datos, la cual lavamos a utilizar para almacenar toda la información. Esto conlleva aprender la sintaxis necesaria para conectarnos con la base de datos y las sentencias SQL de inserción, actualización, borrado y selección.

También veremos cómo utilizar la información recuperada de la base de datos, como realizar transacciones y cómo acceder a la base de datos de manera serializable para que la base de datos no se quede en ningún momento inconsistente. Esto puede ocurrir debido a que la base de datos con frecuencia deberá soportar el acceso de varias personas a la vez.

Por último, explicaremos cómo obtener la información de otro tipo de fuentes que no sean la propia base de datos.

6.2 Extensiones PHP para interacturar con Base de Datos

A la hora de interactuar con una base do datos lo primero quo tememos que hacer es establecer la conexión desde nuestra aplicación web para que posteriormente podamos ejecutar las sentencias SQL (Structured Query Language) que necesitemos de entre todo el abanico de sentencias que nos ofrece este lenguaje.

PHP ofrece diferentes formas de conectarse a bases de datos MySQL y ejecutar comandos SQL dependiendo de la versión que se use. A continuación, se detallan las funciones y clases principales disponibles en distintas versiones:

6.2.1 Extensión mysql

(Deprecada desde PHP 5.5.0, eliminada en PHP 7.0.0)

- Proporciona funciones simples para conectarse y trabajar con bases de datos MySQL.
- Funciones principales:
  - mysql_connect(): Abre una conexión a un servidor MySQL.
  - mysql_select_db(): Selecciona una base de datos MySQL.
  - mysql_query(): Ejecuta una consulta SQL.
  - mysql_fetch_assoc(): Obtiene una fila como un array asociativo.
  - mysql_close(): Cierra la conexión con el servidor.

Nota: Esta extensión ya no debe usarse debido a problemas de seguridad y rendimiento.

6.2.2 Extensión mysqli

(MySQL Improved, PHP 5.x y versiones posteriores)

Esta extensión mejora las capacidades de la anterior, con soporte para Programación Orientada a Objetos (POO) y procedimientos.

Modo Orientado a Procedimientos

- Funciones principales:
  - mysqli_connect(): Establece una conexión a un servidor MySQL.
  - mysqli_select_db(): Selecciona la base de datos.
  - mysqli_query(): Ejecuta una consulta SQL.
  - mysqli_fetch_assoc(): Obtiene una fila como array asociativo.
  - mysqli_close(): Cierra la conexión.

Ejemplo

php Copiar código

```php
<?php
// Datos de conexión
$conexion = mysqli_connect("localhost", "root", "", "mi_base");

// Verificar conexión
if (!$conexion) {
    die("Error de conexión: " . mysqli_connect_error());
}

// Ejecutar consulta
$resultado = mysqli_query($conexion, "SELECT * FROM tabla");

while ($fila = mysqli_fetch_assoc($resultado)) {
    echo $fila['id'] . " - " . $fila['nombre'] . "<br>";
}

// Cerrar conexión
mysqli_close($conexion);
?>
```

Modo Orientado a Objetos

- Clases principales:
  - mysqli: Clase principal para la conexión.
- Métodos comunes:
  - $mysqli->connect(): Establece una conexión.
  - $mysqli->query(): Ejecuta una consulta SQL.
  - $mysqli->close(): Cierra la conexión.
  - mysqli_stmt: Para sentencias preparadas.
  - mysqli_result: Para manejar resultados de consultas.

Ejemplo

```php
$mysqli = new mysqli("localhost", "usuario", "contraseña", "base_datos");

if ($mysqli->connect_error) {
    die("Error de conexión: " . $mysqli->connect_error);
}

$resultado = $mysqli->query("SELECT * FROM tabla");
while ($fila = $resultado->fetch_assoc()) {
    echo $fila['columna'];
}

$mysqli->close();
```

6.2.3 Extensión PDO

(PHP Data Objects, PHP 5.1.0 y posteriores)

Es una interfaz unificada para trabajar con múltiples bases de datos, incluyendo MySQL.

Clases principales:

- PDO: Clase principal para la conexión.
- PDOStatement: Clase para manejar resultados.

Métodos principales:

- new PDO(): Establece la conexión.
- query(): Ejecuta una consulta SQL.
- prepare(): Prepara una consulta para ejecutar con parámetros.
- execute(): Ejecuta una consulta preparada.
- fetch(), fetchAll(): Recupera datos del resultado.

```php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=base_datos", "usuario", "contraseña");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $stmt = $pdo->query("SELECT * FROM tabla");
    while ($fila = $stmt->fetch(PDO::FETCH_ASSOC)) {
        echo $fila['columna'];
    }
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
```

6.2.4 Comparación de Métodos

Características	mysql	mysqli	PDO
Soporte POO	No	Sí	Sí
Consultas preparadas	No	Sí	Sí
Múltiples bbdd	No (MySQL)	No (MySQL)	Sí
Deprecada/obsoleto	SI	No	No

Recomendación

- Usa **PDO** si necesitas trabajar con múltiples tipos de bases de datos o quieres escribir código más flexible y seguro.
- Usa **mysqli** si necesitas funcionalidades específicas de MySQL y no planeas cambiar de base de datos.
- **Evita mysql**, ya que está completamente eliminado en versiones modernas de PHP.

6.3 Extensión mysqli

La extensión mysqli (MySQL Improved) es una herramienta de PHP para interactuar con bases de datos MySQL. Sustituye a la antigua extensión mysql (ya obsoleta) y ofrece soporte para programación orientada a objetos, sentencias preparadas, y mayor seguridad y rendimiento.

Como hemos visto en el apartado anterior mysqli puede utilizarse de dos formas principales:

- Modo Procedural: Más parecido al antiguo mysql, con funciones.
- Modo Orientado a Objetos (POO): Usando clases y métodos.

6.3.1 Conexión con la Base de Datos

Para conectarnos con la base de datos tenemos que facilitar:

- $servidor. El servidor donde se encuentra la base de datos. El servidor se puede especificar facilitando el nombre del servidor, seguido del puerto (nombre_ host :puerto) o mediante una ruta para un servidor local (: ruta)
- $usuario. El usuario
- $contraseña. La contraseña de acceso que tiene $user a la base de datos.
- $base_datos. Base de datos a la que nos queremos conectar

Modo Procedural

```php
<?php
// Datos de conexión
$servidor = "localhost";    // Dirección del servidor (puede ser '127.0.0.1' o un dominio)
$usuario = "root";          // Usuario de la base de datos
$contrasena = "";           // Contraseña del usuario
$base_datos = "mi_base";    // Nombre de la base de datos

// Establecer conexión
$conexion = mysqli_connect($servidor, $usuario, $contrasena, $base_datos);

// Verificar conexión
if (!$conexion) {
    die("Error al conectar con la base de datos: " . mysqli_connect_error());
}
echo "Conexión exitosa a la base de datos.";

// Consulta a la base de datos
$sql = "SELECT * FROM tabla_ejemplo";
$resultado = mysqli_query($conexion, $sql);

if ($resultado) {
    // Mostrar resultados
    while ($fila = mysqli_fetch_assoc($resultado)) {
        echo "ID: " . $fila['id'] . " - Nombre: " . $fila['nombre'] . "<br>";
    }
} else {
    echo "Error en la consulta: " . mysqli_error($conexion);
}

// Cerrar la conexión
mysqli_close($conexion);
?>
```

Modo POO

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");

if ($conexion->connect_error) {
    die("Error de conexión: " . $conexion->connect_error);
}
echo "Conexión exitosa";
$conexion->close();
?>
```

En este apartado vemos dos propiedades muy importantes que nos pueden ser muy útiles en nuestro desarrollo web:

- **connect_errno:** Nos devuelve un número de error al conectarnos a la base de datos. También se podría haber usado la función mysqli_connect_errno().
- **connect_error:** Nos devuelve la descripción del error. También se podría usar la función mysqli_connect_error().

6.3.2 Ejecución de sentencias SQL

Una vez que hemos logrado conectar la base de datos a nuestra aplicación, ya estamos preparados para ejecutar cualquier tipo de sentencia SQL. Más tarde aprenderemos a utilizar los resultados obtenidos tras la ejecución de estas sentencias.

Dentro de las sentencias SQL que podremos ejecutar existen tres tipos:

- Sentencias de definición de datos (DDL)
- Sentencias de manipulación de datos (DML)
- Sentencias de control (DCL)

En PHP existe el método **query()** o la función equivalente **mysqli_query()** que ejecuta las sentencias SQL sobre la base de datos conectada y devuelve un valor booleano o un objeto según use la función o el método.

Devolverá un **valor booleano** en el caso de ejecutar sentencias SQL que no devuelvan información al usuario, como, por ejemplo: **la inserción**, el **borrado de datos**, **actualización**, etc. En estos casos el valor booleano informa si la ejecución de la instrucción SQL se ha realizado de forma satisfactoria (true) o por el contrario se ha producido algún error (falsa).

Por otro lado, este método cuando se ejecuta una sentencia SQL de tipo SELECT devuelve un objeto de la clase **mysqli_result** que contiene el resultado de ejecutar dicha sentencia.

Ejemplo

php Copiar código

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");

$sql = "SELECT * FROM usuarios";
$resultado = $conexion->query($sql);

while ($fila = $resultado->fetch_assoc()) {
    echo $fila['id'] . " - " . $fila['nombre'] . "<br>";
}

$conexion->close();
?>
```

Debemos tener claro que **$resultado** es un objeto de la clase **mysqli_result**.

6.3.3 Clase mysqli_result

Un objeto mysqli_result se genera automáticamente cuando se ejecuta una consulta SELECT con mysqli_query() o con un método como $mysqli->query().

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");

$resultado = $conexion->query("SELECT * FROM usuarios");
if ($resultado instanceof mysqli_result) {
    echo "Consulta exitosa. Total de filas: " . $resultado->num_rows;
}
$conexion->close();
?>
```

Propiedades

- $resultado->num_rows. Devuelve el número de filas
- $resultado->fetch_assoc(). Devuelve del conjunto de resultados como array asociativo
- $resultado->fetch_row(). Devuelve una fila como array indexado numérico.
- $resultado->fetch_obj(). Devuelve una fila como objeto de la clase estándar.
- $resultado->data_seek(). Establece el puntero en una fila específica del conjunto de resultado

Método num_rows()

Devuelve el número de filas en el conjunto de resultados.

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");
$resultado = $conexion->query("SELECT * FROM usuarios");

echo "Número de filas: " . $resultado->num_rows;

$resultado->free(); // Liberar el objeto mysqli_result
$conexion->close();
?>
```

Método fetch_assoc()

Devuelve una fila del conjunto de resultados como un array asociativo.

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");
$resultado = $conexion->query("SELECT * FROM usuarios");

while ($fila = $resultado->fetch_assoc()) {
    echo "ID: " . $fila['id'] . " - Nombre: " . $fila['nombre'] . "<br>";
}

$resultado->free(); // Liberar memoria
$conexion->close();
?>
```

Método fetch_row()

Devuelve una fila como un array indexado numéricamente.

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");
$resultado = $conexion->query("SELECT * FROM usuarios");

while ($fila = $resultado->fetch_row()) {
    echo "ID: " . $fila[0] . " - Nombre: " . $fila[1] . "<br>";
}

$resultado->free();
$conexion->close();
?>
```

Método fetch_object()

Devuelve una fila como un objeto de la clase estándar.

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");
$resultado = $conexion->query("SELECT * FROM usuarios");

while ($fila = $resultado->fetch_object()) {
    echo "ID: " . $fila->id . " - Nombre: " . $fila->nombre . "<br>";
}

$resultado->free();
$conexion->close();
?>
```

Método data_seek()

Establece el puntero en una fila específica del conjunto de resultados.

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");
$resultado = $conexion->query("SELECT * FROM usuarios");

// Mover el puntero a la segunda fila
$resultado->data_seek(1);

$fila = $resultado->fetch_assoc();
echo "Fila actual: ID: " . $fila['id'] . " - Nombre: " . $fila['nombre'];

$resultado->free();
$conexion->close();
?>
```

6.3.4 Cierre de conexiones

Es importante liberar los recursos asociados con un objeto mysqli_result una vez que se ha terminado de usar. Esto se hace con el método free(). Por otro lado Una vez que ya hemos terminado de trabajar con la base de datos es preciso cerrar la conexión con el método close()

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");
$resultado = $conexion->query("SELECT * FROM usuarios");

// Procesar resultados...
$resultado->free(); // Liberar memoria

$conexion->close();
?>
```

6.3.5 Ejemplo completo

```php
<?php
$conexion = new mysqli("localhost", "root", "", "mi_base");

if ($conexion->connect_error) {
    die("Error de conexión: " . $conexion->connect_error);
}

$sql = "SELECT id, nombre FROM usuarios";
$resultado = $conexion->query($sql);

if ($resultado instanceof mysqli_result) {
    echo "Número de filas: " . $resultado->num_rows . "<br>";

    // Mostrar todas las filas
    while ($fila = $resultado->fetch_assoc()) {
        echo "ID: " . $fila['id'] . " - Nombre: " . $fila['nombre'] . "<br>";
    }

    // Liberar memoria del resultado
    $resultado->free();
} else {
    echo "Error en la consulta: " . $conexion->error;
}

$conexion->close();
?>
```

6.3.6 Sentencias Preparadas

Una declaración preparada es una función que se utiliza para ejecutar las mismas (o similares) sentencias SQL repetidamente con una alta eficiencia.

Las declaraciones preparadas básicamente funcionan como sigue:

1. **Prepare**. Una plantilla de instrucción SQL se crea y se envía a la base de datos. Algunos valores no se especifican llamado parámetros (con la etiqueta "?"). Ejemplo: INSERT INTO Clientes VALUES (?,?,?)
2. **Análisis y optimización**. La base de datos analiza, compila, y lleva a cabo la optimización de consultas en la plantilla de instrucción SQL, y almacena el resultado sin ejecutarlo.
3. **Ejecutar**: En un momento posterior, la aplicación se une a los valores a los parámetros, y la base de datos ejecuta la instrucción. La aplicación puede ejecutar la instrucción tantas veces como se quiera con diferentes valores.

En comparación con la ejecución de las sentencias SQL directamente, las declaraciones preparadas tienen las siguientes ventajas:

- **Reduce tiempo análisis**. Declaraciones preparadas reducen el tiempo de análisis ya que la preparación de la consulta se realiza sólo una vez (aunque la sentencia se ejecuta varias veces)
- **Minimiza ancho banda**. Se minimiza el ancho de banda al servidor ya que se necesite enviar sólo los parámetros cada vez, y no toda la consulta.
- **Mayor seguridad**. Declaraciones preparadas son muy útiles contra inyecciones SQL, porque los valores de los parámetros, que se transmiten más tarde usando un protocolo diferente, no necesitan ser escapados. Si la plantilla de declaración original no se deriva de la entrada externa, no puede ocurrir la inyección de SQL.

Vamos a ver un ejemplo de Sentencias Preparadas en PHP con MySQLi.

```php
<?php
// Configuración de la conexión
$server = "localhost";
$user = "usuario";
$password = "password";
$dbname = "ejemplo";

// 1. Conexión a la base de datos
$db = new mysqli($server, $user, $password, $dbname);

// Verificar conexión
if ($db->connect_error) {
    die("La conexión ha fallado, error número " . $db->connect_errno . ": " . $db->connect
}

// 2. Preparar la sentencia
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto) VALUES (?, ?, ?)");
if (!$stmt) {
    die("Error al preparar la consulta: " . $db->error);
}

// Vincular parámetros ('ssi' = string, string, integer)
$stmt->bind_param('ssi', $nombre, $ciudad, $contacto);

// 3. Establecer parámetros y ejecutar varias inserciones
$nombre = "Donald Trump";
$ciudad = "Madrid";
$contacto = 4124124;
$stmt->execute(); // Primera ejecución

$nombre = "Hillary Clinton";
$ciudad = "Barcelona";
$contacto = 4665767;
$stmt->execute(); // Segunda ejecución

// 4. Mensaje de éxito
echo "Se han creado las entradas exitosamente";

// 5. Cerrar la sentencia y la conexión
$stmt->close();
$db->close();
?>
```

Incluimos un interrogante donde queremos sustituir un integer, un string, un double o un blob en la función prepare(). Después usamos la función bind_param():

```
$stmt->bind_param('ssi', $nombre, $ciudad, $contacto);
```

La función enlaza los parámetros con la consulta SQL y le dice a la base de datos que parámetros son. El argumento "ssi" especifica el tipo de dato que se espera que sea el parámetro. Pueden ser de cuatro tipos:

- Letra i - número entero
- Letra d - doble
- Letra s - string
- Letra b - BLOB

Se debe especificar uno por cada parámetro.

Método Bind_result().

Este método se usa para vincular las columnas del resultado a variables específicas, lo que permite acceder a los valores de las filas devueltas por una consulta.

Veamos el siguiente ejemplo:

```php
// 1. Establecer conexión
$db = new mysqli($server, $user, $password, $dbname);

// Verificar conexión
if ($db->connect_error) {
    die("Error de conexión: " . $db->connect_errno . " - " . $db->connect_error);
}

// 2. Preparar la consulta
$stmt = $db->prepare("SELECT id, nombre, ciudad FROM Clientes WHERE ciudad = ?");
if (!$stmt) {
    die("Error al preparar la consulta: " . $db->error);
}

// 3. Vincular parámetros para la consulta (en este caso, ciudad)
$ciudad_buscar = "Madrid";
$stmt->bind_param("s", $ciudad_buscar);

// 4. Ejecutar la consulta
$stmt->execute();

// 5. Vincular los resultados a variables
$stmt->bind_result($id, $nombre, $ciudad);

// 6. Recuperar y procesar los resultados
echo "Resultados para la ciudad: $ciudad_buscar<br>";
while ($stmt->fetch()) {
    echo "ID: $id - Nombre: $nombre - Ciudad: $ciudad<br>";
}

// 7. Cerrar la sentencia y la conexión
$stmt->close();
$db->close();
?>
```

Este método es útil cuando quieres procesar los resultados directamente en variables específicas, evitando la necesidad de usar métodos como fetch_assoc() o fetch_row().

6.3.7 Manejo de Excepciones

El manejo de excepciones en PHP permite capturar y gestionar errores de forma controlada, mejorando la robustez y la claridad del código. Este mecanismo utiliza bloques **try-catch** y funciona con cualquier clase que extienda de Exception.

6.3.7.1 Esctructura básica try-catch

La estructura try-catch permite identificar errores en un bloque de código y gestionarlos en otro.

Sintáxis

```php
try {
    // Código que puede generar una excepción
} catch (Exception $e) {
    // Código para manejar la excepción
    echo "Error: " . $e->getMessage();
} finally {
    // Código que se ejecuta siempre (opcional)
    echo "Bloque finally ejecutado.";
}
```

Explicación

- try: Contiene el código que podría generar una excepción.
- catch: Captura la excepción lanzada. $e es la instancia de la excepción capturada, y su método principal es:
  - $e->getMessage(): Obtiene el mensaje del error.
- finally (opcional): Se ejecuta siempre, independientemente de si ocurre una excepción o no. Útil para liberar recursos.

6.3.7.2 Excepción con la clase Clase

La clase base Exception se usa para manejar errores genéricos o personalizados. Puedes lanzar una excepción con throw.

Ejemplo de una excepción genérica

```php
<?php
try {
    $numero = -1;
    if ($numero < 0) {
        throw new Exception("El número no puede ser negativo.");
    }
    echo "Número válido: $numero";
} catch (Exception $e) {
    echo "Error: " . $e->getMessage(); // Captura el mensaje de la excepción
} finally {
    echo "<br>Fin del programa.";
}
?>
```

Propiedades y métodos de la clase Exception:

- $e->getMessage(): Obtiene el mensaje de error.
- $e->getCode(): Código del error.
- $e->getFile(): Archivo donde ocurrió el error.
- $e->getLine(): Línea donde ocurrió el error.

Ejemplo avanzado

```php
<?php
try {
    throw new Exception("Error genérico", 100);
} catch (Exception $e) {
    echo "Mensaje: " . $e->getMessage() . "<br>";
    echo "Código: " . $e->getCode() . "<br>";
    echo "Archivo: " . $e->getFile() . "<br>";
    echo "Línea: " . $e->getLine() . "<br>";
}
?>
```

6.3.7.3 *Excepciones con clase mysqli_sql_exception*

Cuando trabajas con bases de datos usando mysqli, los errores pueden ser manejados a través de la clase **mysqli_sql_exception**.

Habilitar el Modo Excepciones

Desde PHP 8.1, mysqli lanza excepciones automáticamente. Si usas una versión anterior, debes habilitarlo manualmente:

```php
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
```

Ejemplo manejo excepciones en mysqli

```php
<?php
try {
    // Habilitar el modo de excepciones en mysqli
    mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);

    // Conexión a la base de datos
    $conexion = new mysqli("localhost", "root", "password", "mi_base");

    // Intentar ejecutar una consulta inválida
    $resultado = $conexion->query("SELECT * FROM tabla_inexistente");

} catch (mysqli_sql_exception $e) {
    // Manejar errores específicos de MySQL
    echo "Error MySQL: " . $e->getMessage() . "<br>";
    echo "Código: " . $e->getCode() . "<br>";
} finally {
    echo "Fin de la operación.";
}
?>
```

Comparación entre Exception y mysqli_sql_esception

Características	exception	Mysqli_sql_exception
Uso principal	Manejo errores genéricos	Sí
Clase base	Exception	Mysqli_sql_exeption
Métodos importantes	getMessage(), getCode()	getMessage(),getCode()
Contexto de uso	Validaciones generales, lógica	Conexión y consultas MySQL

6.3.8 Transacciones mysqli

Una transacción es un conjunto de operaciones SQL que se ejecutan como una unidad indivisible. Todas las operaciones deben completarse correctamente; de lo contrario, se deshacen (rollback).

Propiedades de una transacción (ACID):

- Atomicidad: Todas las operaciones se completan o ninguna lo hace.
- Consistencia: El sistema pasa de un estado válido a otro.

- Aislamiento: Los cambios realizados en una transacción no son visibles para otras hasta que se confirmen.
- Durabilidad: Una vez confirmada, los cambios persisten incluso ante fallos del sistema.

Para trabajar con transacciones con php y mysql es importante tener en cuenta:

- Que tu base de datos tenga soporte para tablas innodb.
- Las transacciones sólo son soportadas para las tablas tipo innodb, así que necesitamos crear las tablas o modificarlas con este tipo. ¿Cómo me aseguro de que mis tablas sean innodb? Ejecutando la siguiente query en tu base de datos: ALTER TABLE tutabla ENGINE = INNODB

Métodos mysqli para el uso de las transacciones:

- begin_transaction(): Inicia una transacción.
- commit(): Confirma (guarda) los cambios.
- rollback(): Revierte (deshace) los cambios.

Ejemplo práctico con *mysqli_sql_exception*

php Copiar código

```php
<?php
// Habilitar excepciones en MySQLi
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);

try {
    // Conexión a la base de datos
    $mysqli = new mysqli("localhost", "usuario", "contraseña", "mi_base_datos");

    // Configurar el conjunto de caracteres
    $mysqli->set_charset("utf8");

    // Iniciar la transacción
    $mysqli->begin_transaction();

    // Primera operación: Restar dinero de una cuenta
    $mysqli->query("UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1");

    // Segunda operación: Añadir dinero a otra cuenta
    $mysqli->query("UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2");

    // Confirmar las operaciones
    $mysqli->commit();
    echo "Transacción completada con éxito.";
} catch (mysqli_sql_exception $e) {
    // Revertir los cambios en caso de error
    if (isset($mysqli)) {
        $mysqli->rollback();
    }
    echo "Error en la transacción: " . $e->getMessage();
} finally {
    // Cerrar la conexión
    if (isset($mysqli)) {
        $mysqli->close();
    }
}
?>
```

6.4 PDO – PHP Data Objetcts

PDO (PHP Data Objects) es una extensión de PHP que proporciona una interfaz unificada para acceder a bases de datos, independientemente del sistema gestor (MySQL, PostgreSQL, SQLite, etc.). Permite realizar consultas de forma segura, utilizando sentencias preparadas para prevenir ataques de inyección SQL.

La clase PDO se fundamenta en 3 clases:

- PDO
- PDOStatement
- PDOException

La clase **PDO** se encarga de mantener la conexión a la base de datos y otro tipo de conexiones específicas como transacciones, además de crear instancias de la clase PDOStatement. La clase **PDOStatement**, es la que maneja las sentencias SQL y devuelve los resultados y la clase **PDOException** se utiliza para manejar los errores.

6.4.1 Conexión BBDD con PDO

El constructor de PDO requiere:

- DSN (Data Source Name): Define el tipo de base de datos, host, nombre de la base de datos, y otras configuraciones.
- Usuario y contraseña: Credenciales de acceso.
- Opciones (opcional): Mediante un array se especifican diferentes opciones de de la clase.

Diferentes sistemas gestores de bases de datos tienen distintos métodos para conectarse. La mayoría se conectan de forma parecida a como se conecta a MySQL

Ejemplo

```php
<?php
try {
    $dsn = 'mysql:host=localhost;dbname=mi_base_datos;charset=utf8mb4';
    $usuario = 'usuario';
    $contraseña = 'contraseña';

    // Crear instancia de PDO
    $pdo = new PDO($dsn, $usuario, $contraseña);

    // Configurar el modo de errores para excepciones
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    echo "Conexión exitosa.";
} catch (PDOException $e) {
    echo "Error en la conexión: " . $e->getMessage();
}
?>
```

Normalmente se suelen crear un objeto de la clase PDO enviando un array de opciones de conexión, aunque como se dijo en el párrafo anterior es opcional. Las opciones más comunes son:

- PDO::ATTR_ERRMODE: Define cómo se manejan los errores. Recomendado: PDO::ERRMODE_EXCEPTION.
- PDO::ATTR_DEFAULT_FETCH_MODE: Establece el modo de obtención de datos por defecto. Común: PDO::FETCH_ASSOC, PDO::FETCH_OBJ
- PDO::ATTR_EMULATE_PREPARES: Habilita/deshabilita las sentencias preparadas emuladas. Es mejor deshabilitarlo (false) para mayor seguridad.

Veamos el siguiente ejemplo

php Copiar código

```php
<?php
try {
    // Datos de conexión
    $dsn = "mysql:host=localhost;dbname=mi_base_datos;charset=utf8mb4";
    $usuario = "usuario";
    $contraseña = "contraseña";

    // Opciones de conexión
    $opciones = [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, // Manejo de errores
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC, // Modo de fetch por defecto
        PDO::ATTR_EMULATE_PREPARES => false, // Usar sentencias preparadas nativas
    ];

    // Crear instancia de PDO
    $pdo = new PDO($dsn, $usuario, $contraseña, $opciones);

    echo "Conexión exitosa.";
} catch (PDOException $e) {
    echo "Error en la conexión: " . $e->getMessage();
}
?>
```

Otro ejemplo podría ser

```php
<?php
try {
    // Datos de conexión
    $dsn = "mysql:host=localhost;dbname=mi_base_datos;charset=utf8mb4";
    $usuario = "usuario";
    $contraseña = "contraseña";

    // Opciones de configuración para PDO
    $opciones = [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,            // Manejo de errores con exc
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,       // Devolver resultados como
        PDO::ATTR_EMULATE_PREPARES => false,                   // Deshabilitar emulación de
        PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES 'utf8mb4'", // Comando inicial para estab
        PDO::ATTR_PERSISTENT => true                           // Habilitar conexiones persis
    ];

    // Crear la conexión PDO
    $pdo = new PDO($dsn, $usuario, $contraseña, $opciones);

    echo "Conexión establecida correctamente.";
} catch (PDOException $e) {
    echo "Error al conectar con la base de datos: " . $e->getMessage();
}
?>
```

Explicación del array de opciones:

- PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION. Configura el modo de errores para que PDO lance excepciones cuando ocurra un problema. Esto permite manejar errores con bloques try-catch.
- PDO::ATTR_PERSISTENT => false. Desactiva las conexiones persistentes. Cada conexión es cerrada al finalizar el script, útil para liberar recursos en entornos con múltiples usuarios.
- PDO::ATTR_EMULATE_PREPARES => false. Desactiva las sentencias preparadas emuladas para utilizar las nativas del sistema gestor, mejorando la seguridad y confiabilidad.
- PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES " . CHARSET . " COLLATE " . COLLECTION. Configura la codificación (utf8mb4) y la colación (utf8mb4_unicode_ci) al iniciar la conexión. Esto asegura que los datos se almacenen y manipulen correctamente.

Para cerrar una conexión:

```
$pdo = null;
```

6.4.2 Excepciones y opciones con PDO

PDO maneja los errores en forma de excepciones, por lo que la conexión siempre ha de ir encerrada en un bloque try/catch. Se puede (y se debe) especificar el modo de error estableciendo el atributo error mode:

- $dbh->setAttribute(PDO::ATTRR_ERRMODE, PDO::ERRMODE_SILENT);
- $dbh->setAttribute(PDO::ATTRR_ERRMODE, PDO::ERRMODE_WARNING);
- $dbh->setAttribute(PDO::ATTRR_ERRMODE, PDO::ERRMODE_EXCEPTION);

No importa el modo de error, si existe un fallo en la conexión siempre producirá una excepción, por eso siempre se conecta con try/catch.

PDO::ERRMODE_SILENT. Los errores no se mostrarán. Se tendrían que emplear PDO::errorCode() y PDO::errorInfo() o su versión en PDOStatement::errorCode() y PDOStatement::errorInfo().

PDO::ERRMODE_WARNING. Muestra las advertencias mediante un mensaje E_WARNING. Modo empleado para depurar o hacer pruebas para ver errores sin interrumpir el flujo de la aplicación.

PDO::ERRMODE_EXCEPTION. Además de establecer el código de error, PDO lanzará una excepción PDOException y establecerá sus propiedades para luego poder reflejar el error y su información. Este modo se emplea en la mayoría de las situaciones, ya que permite manejar los errores y a la vez esconder datos que podrían ayudar a alguien a atacar tu aplicación.

El modo de error se puede aplicar con el método PDO::setAttribute o mediante un array de opciones al instanciar PDO.

6.4.3 Clase PDOStatement.

La clase PDOStatement permite la ejecución de comandos SQL. Esta clase proporciona métodos y propiedades que permiten ejecutar consultas SQL, gestionar parámetros y recuperar resultados de manera eficiente y segura.

Los métodos de la clase PDO que devuelven un objeto de la clase PDOStatement son:

- prepare()
- query()
- execute()

El método **prepare()** se utiliza para crear una sentencia preparada. Devuelve un objeto PDOStatement que puede ser ejecutado posteriormente con los parámetros correspondientes.

El método **query()** ejecuta una consulta SQL directamente (sin parámetros) y devuelve un objeto PDOStatement con el resultado de la consulta.

Aunque el método **execute()** no pertenece directamente a PDO, se usa después de prepare() para ejecutar una sentencia preparada. El resultado se devuelve a través del objeto PDOStatement previamente creado por prepare()

6.4.3.1 *Sentencias Preparadas en PDO*

Para crear una sentencia preparada se utiliza el método prepare() de la clase PDO. Esto genera un objeto PDOStatement que puede ser ejecutado varias veces con diferentes valores de parámetros.

php Copiar código

```php
<?php
try {
    $dsn = "mysql:host=localhost;dbname=mi_base_datos;charset=utf8mb4";
    $pdo = new PDO($dsn, "usuario", "contraseña", [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    ]);

    // Crear una sentencia preparada
    $sql = "SELECT * FROM usuarios WHERE id = :id";
    $stmt = $pdo->prepare($sql);

    // Vincular el parámetro y ejecutar
    $stmt->execute([':id' => 1]);

    // Obtener resultados
    $usuario = $stmt->fetch(PDO::FETCH_ASSOC);
    print_r($usuario);
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
?>
```

PDO facilita el uso de sentencias preparadas en PHP, que mejoran el rendimiento y la seguridad de la aplicación.

Cuando se obtienen, insertan o actualizan datos, el esquema es:

```
PREPARE -> [BIND] -> EXECUTE.
```

6.4.3.1.1 Parámetros de la sentencia

Se pueden indicar los parámetros en la sentencia con un interrogante "?" o mediante nombres específicos, variables especiales o placeholders. También se pueden pasar los valores mediante un array con el uso del método execute().

Uso de interrogantes (?)

Se definen los parámetros de la misma forma que con la clase mysqli.

```php
<?php
try {
    // Configuración de conexión
    $dsn = "mysql:host=localhost;dbname=mi_base_datos;charset=utf8mb4";
    $usuario = "usuario";
    $contraseña = "contraseña";

    $pdo = new PDO($dsn, $usuario, $contraseña, [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    ]);

    // Sentencia preparada con parámetros posicionales
    $sql = "INSERT INTO usuarios (nombre, email, edad) VALUES (?, ?, ?)";
    $stmt = $pdo->prepare($sql);

    // Valores para los parámetros
    $valores = ["Juan", "juan@example.com", 30];

    // Ejecutar la sentencia preparada
    $stmt->execute($valores);

    echo "Usuario insertado con éxito.";
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
?>
```

Variables específicas o placeholders

Normalmente para especificar los parámetros de la sentencia sql se usan nombres específicos o variables especiales en vez de interrogantes.

Ventajas del uso de placeholders:

1. Legibilidad: Las consultas son más claras porque cada parámetro tiene un nombre que describe su propósito.
    - Por ejemplo, :edad es más comprensible que un simple ?.
2. Orden independiente: Los valores no dependen del orden en que aparecen en la consulta, a diferencia de los parámetros posicionales (?).
3. Compatibilidad con estructuras complejas: Si la consulta tiene muchos parámetros, los placeholders nombrados facilitan su gestión.

Veamos el siguiente ejemplo

```php
// Configuración de conexión
$dsn = "mysql:host=localhost;dbname=mi_base_datos;charset=utf8mb4";
$usuario = "usuario";
$contraseña = "contraseña";

$pdo = new PDO($dsn, $usuario, $contraseña, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
]);

// Sentencia preparada con placeholders nombrados
$sql = "SELECT * FROM usuarios WHERE edad > :edad AND email LIKE :email";
$stmt = $pdo->prepare($sql);

// Valores a vincular
$edad = 25;
$email = '%@example.com%';

// Vincular parámetros nombrados
$stmt->bindParam(':edad', $edad, PDO::PARAM_INT);     // Vincula :edad
$stmt->bindParam(':email', $email, PDO::PARAM_STR);   // Vincula :email

// Ejecutar la sentencia preparada
$stmt->execute();

// Obtener resultados
$usuarios = $stmt->fetchAll(PDO::FETCH_ASSOC);

foreach ($usuarios as $usuario) {
    echo "Nombre: " . $usuario['nombre'] . ", Email: " . $usuario['email'] . "<br>";
}
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
```

6.4.3.1.2 Vinculación de variables

Para vincular variables con los parámetros de la sentencia preparada, podemos usar dos métodos:

- bindParam()
- bindValue()

bindParam()

Vincula una variable por referencia. Útil si el valor cambia antes de ejecutar la sentencia.

bindValue()

Vincula el valor directamente. Una vez vinculado si esa variable cambia de valor no afecta al excecute()

```php
$stmt = $pdo->prepare("UPDATE usuarios SET nombre = :nombre WHERE id = :id");

// Vincular con bindParam
$id = 1;
$nombre = "Carlos";
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
$stmt->bindParam(':nombre', $nombre, PDO::PARAM_STR);
$stmt->execute();

// Vincular con bindValue
$stmt->bindValue(':id', 2, PDO::PARAM_INT);
$stmt->bindValue(':nombre', "María", PDO::PARAM_STR);
$stmt->execute();
```

Método bindParam()

```php
bool PDOStatement::bindParam(
    string|int $parameter,
    mixed &$variable,
    int $data_type = PDO::PARAM_STR,
    int $length = null,
    mixed $driver_options = null
)
```

Parámetros principales:

- **$parameter**: El nombre del placeholder en la consulta (por ejemplo, :nombre) o el índice del parámetro posicional (1 para el primer ?, 2 para el segundo, etc.).
- **$variable**: La variable PHP que será vinculada al parámetro. Se vincula por referencia, lo que significa que cualquier cambio en la variable antes de ejecutar la consulta afecta al valor utilizado.
- **$data_type**: Especifica el tipo de datos del parámetro. Puede ser uno de los siguientes valores:
    - PDO::PARAM_INT: Para valores enteros.
    - PDO::PARAM_STR: Para cadenas de texto (por defecto).
    - PDO::PARAM_BOOL: Para valores booleanos (true/false).
    - PDO::PARAM_NULL: Para valores nulos.
    - Otros valores específicos del controlador, como PDO::PARAM_LOB (grandes objetos binarios).

- **$length (opcional)**: Longitud máxima del dato vinculado. Útil para columnas de longitud fija.
- **$driver_options (opcional)**: Opciones específicas del controlador de base de datos.

El método bindParam() es extremadamente útil para gestionar sentencias preparadas que necesitan valores dinámicos. Definir el tipo de dato en el tercer parámetro no solo mejora la claridad, sino que también garantiza la seguridad y consistencia al interactuar con la base de datos.

php Copiar código

```php
<?php
try {
    // Configuración de la conexión PDO
    $dsn = "mysql:host=localhost;dbname=mi_base_datos;charset=utf8mb4";
    $usuario = "usuario";
    $contraseña = "contraseña";

    $pdo = new PDO($dsn, $usuario, $contraseña, [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    ]);

    // Sentencia preparada con placeholders nombrados
    $sql = "INSERT INTO productos (nombre, precio, en_stock, descripcion)
            VALUES (:nombre, :precio, :en_stock, :descripcion)";
    $stmt = $pdo->prepare($sql);

    // Variables de diferentes tipos
    $nombre = "Laptop";                  // Tipo STRING
    $precio = 799.99;                    // Tipo FLOAT (manejado como STRING por PDO)
    $enStock = true;                     // Tipo BOOLEAN
    $descripcion = "Laptop de gama alta"; // Tipo STRING

    // Vinculación de variables con tipo y longitud
    $stmt->bindParam(':nombre', $nombre, PDO::PARAM_STR, 50);        // Máximo 50 caracteres
    $stmt->bindParam(':precio', $precio, PDO::PARAM_STR, 10);        // Máximo 10 caracteres
    $stmt->bindParam(':en_stock', $enStock, PDO::PARAM_BOOL);        // Longitud no aplica
    $stmt->bindParam(':descripcion', $descripcion, PDO::PARAM_STR, 255); // Máximo 255 carac

    // Ejecutar la sentencia preparada
    $stmt->execute();

    echo "Producto insertado correctamente.";
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
?>
```

6.4.4 Consultar Datos con PDO

La consulta de datos se realiza mediante PDOStatement::fetch, que obtiene la siguiente fila de un conjunto de resultados. Antes de llamar a fetch (o durante) hay que especificar como se quieren devolver los datos:

- PDO::FETCH_ASSOC: devuelve un array asociativo donde los índices se corresponden conel nombrede las columnas
- PDO::FETCH_NUM: devuelve un array indexado numéricamente.
- PDO::FETCH_BOTH: valor por defecto. Devuelve un array asociativo y numérico.
- PDO::FETCH_OBJ: devuelve cada fila como un objeto de clase anónimo.
- PDO::FETCH_CLASS: mapea cada fila a una instancia de una clase específica definida previamente. Esta es la opción que usaremos para UPDATE e INSERT.
- PDO::FETCH_COLUMN: devuelve un único valor de una instancia por fila.
- PDO::FETCH_KEY_PAIR: devuelve un array donde la primera columna es la clave y la segunda el valor.

Ejemplos prácticos con los tipos de fetch()

Ejemplo **Conexión**

php Copiar código

```php
<?php
$dsn = 'mysql:host=localhost;dbname=ejemplo';
$username = 'root';
$password = '';
$options = [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION];

try {
    $pdo = new PDO($dsn, $username, $password, $options);
} catch (PDOException $e) {
    die("Error en la conexión: " . $e->getMessage());
}
?>
```

Uso de PDO::FETCH_ASSOC

php Copiar código

```php
<?php
$sql = "SELECT id, nombre, email FROM usuarios WHERE id > :idMin";
$stmt = $pdo->prepare($sql);

$idMin = 10;
$stmt->bindParam(':idMin', $idMin, PDO::PARAM_INT);

$stmt->setFetchMode(PDO::FETCH_ASSOC);
$stmt->execute();

while ($fila = $stmt->fetch()) {
    echo "ID: {$fila['id']}, Nombre: {$fila['nombre']}, Email: {$fila['email']}<br>";
}
?>
```

Uso de PDO::FETCH_NUM

php Copiar código

```php
<?php
$sql = "SELECT id, nombre, email FROM usuarios WHERE id > :idMin";
$stmt = $pdo->prepare($sql);

$idMin = 3;
$stmt->bindParam(':idMin', $idMin, PDO::PARAM_INT);

$stmt->setFetchMode(PDO::FETCH_OBJ);
$stmt->execute();

while ($fila = $stmt->fetch()) {
    echo "ID: {$fila->id}, Nombre: {$fila->nombre}, Email: {$fila->email}<br>";
}
?>
```

Uso de PDO::FETCH_CLASS

php Copiar código

```php
<?php
class Usuario {
    public $id;
    public $nombre;
    public $email;
}

$sql = "SELECT id, nombre, email FROM usuarios WHERE id > :idMin";
$stmt = $pdo->prepare($sql);

$idMin = 1;
$stmt->bindParam(':idMin', $idMin, PDO::PARAM_INT);

$stmt->setFetchMode(PDO::FETCH_CLASS, "Usuario");
$stmt->execute();

while ($usuario = $stmt->fetch()) {
    echo "ID: {$usuario->id}, Nombre: {$usuario->nombre}, Email: {$usuario->email}<br>";
}
?>
```

Uso de PDO::FETCH_COLUMN

```php
<?php
$sql = "SELECT nombre FROM usuarios WHERE id > :idMin";
$stmt = $pdo->prepare($sql);

$idMin = 2;
$stmt->bindParam(':idMin', $idMin, PDO::PARAM_INT);

$stmt->setFetchMode(PDO::FETCH_COLUMN);
$stmt->execute();

while ($nombre = $stmt->fetch()) {
    echo "Nombre: $nombre<br>";
}
?>
```

Uso PDO::FETCH_KEY_PAIR

En este caso, dado que PDO::FETCH_KEY_PAIR se utiliza normalmente con fetchAll(). Se suele usar para obtener los arrays a partir de los cuales generar las listas desplegables y listas de checbox dinámicas en las vistas.

```php
<?php
$sql = "SELECT id, nombre FROM usuarios WHERE id > :idMin";
$stmt = $pdo->prepare($sql);

$idMin = 0;
$stmt->bindParam(':idMin', $idMin, PDO::PARAM_INT);

$stmt->execute();
$stmt->setFetchMode(PDO::FETCH_KEY_PAIR);

$resultado = $stmt->fetchAll();

foreach ($resultado as $id => $nombre) {
    echo "ID: $id, Nombre: $nombre<br>";
}
?>
```

Uso de fetch() y fetchAll()

Elegir entre fetch() y fetchAll() depende del tamaño del conjunto de resultados, la estructura del código y el contexto de uso.

Usar **fetchAll():**

1. **El conjunto de datos es pequeño o manejable en memoria:**
    - Si sabes que el número de filas devueltas por la consulta será reducido y no supondrá un problema de rendimiento al cargarlas todas en memoria.
    - Por ejemplo, cuando estás mostrando una lista corta de elementos (como categorías o menús).
2. **Necesitas acceder a todas las filas a la vez:**
    - Si el procesamiento de los datos requiere que todas las filas estén disponibles al mismo tiempo, como en casos de agregaciones o transformaciones.
3. **Se requiere reusar los datos:**
    - Si necesitas iterar sobre los datos múltiples veces, fetchAll() es más eficiente porque los datos ya están en memoria, evitando múltiples llamadas a fetch().
4. **Facilidad de uso con operaciones en lote:**
    - Si planeas exportar datos (a CSV, JSON, etc.) o enviarlos a una API, tenerlos todos en un solo array facilita el proceso.

Usar **fetch()**

1. **El conjunto de datos es grande o desconocido:**
    - Para evitar problemas de memoria al manejar grandes cantidades de datos. fetch() recupera una fila a la vez, reduciendo la carga de memoria.
2. **El procesamiento es fila por fila:**
    - Cuando cada fila se procesa individualmente y no necesitas acceder a las demás. Por ejemplo, enviar correos electrónicos uno por uno o realizar actualizaciones por fila.
3. **Eficiencia en consultas prolongadas:**
    - En scripts de larga duración o con muchas operaciones, fetch() es más eficiente porque no se almacenan todas las filas en memoria simultáneamente.
4. **Manejo de flujos de datos en tiempo real:**
    - Si estás procesando datos mientras llegan, como en sistemas de transmisión en tiempo real o cargas diferidas, fetch() es una opción ideal.

6.4.5 Otras métodos de utilidad

1. exec()

Definición:

- Ejecuta una sentencia SQL directamente en la base de datos y devuelve el número de filas afectadas.
- Se utiliza para sentencias que no devuelven resultados, como INSERT, UPDATE, DELETE, o CREATE TABLE.
- No se utiliza para consultas que devuelven datos, como SELECT.

Ejemplo

php Copiar código

```php
<?php
// Conexión a la base de datos
$dsn = 'mysql:host=localhost;dbname=mi_base_datos;charset=utf8';
$usuario = 'root';
$contraseña = '';
$pdo = new PDO($dsn, $usuario, $contraseña);

// Ejecutar una sentencia SQL
$sql = "UPDATE usuarios SET estado = 'activo' WHERE inactivo_desde IS NULL";
$filaAfectadas = $pdo->exec($sql);

echo "Filas afectadas: $filaAfectadas";
?>
```

2. lastInsertId()

Definición:

- Devuelve el ID autogenerado por la última consulta INSERT en una tabla con una columna AUTO_INCREMENT.
- Es útil para obtener el ID de un registro recién insertado.

Ejemplo

```php
<?php
// Conexión a la base de datos
$dsn = 'mysql:host=localhost;dbname=mi_base_datos;charset=utf8';
$usuario = 'root';
$contraseña = '';
$pdo = new PDO($dsn, $usuario, $contraseña);

// Insertar un nuevo usuario
$sql = "INSERT INTO usuarios (nombre, email) VALUES ('Juan Pérez', 'juan@example.com')";
$pdo->exec($sql);

// Obtener el último ID insertado
$ultimoId = $pdo->lastInsertId();

echo "Último ID insertado: $ultimoId";
?>
```

3. rowCount()

Definición:

- Devuelve el número de filas afectadas por la última operación SQL ejecutada a través de un objeto PDOStatement.
- Es útil para saber cuántas filas fueron afectadas por una consulta UPDATE, DELETE o SELECT.

Ejemplo:

```php
<?php
// Conexión a la base de datos
$dsn = 'mysql:host=localhost;dbname=mi_base_datos;charset=utf8';
$usuario = 'root';
$contraseña = '';
$pdo = new PDO($dsn, $usuario, $contraseña);

// Preparar y ejecutar una consulta
$sql = "DELETE FROM usuarios WHERE estado = 'inactivo'";
$stmt = $pdo->prepare($sql);
$stmt->execute();

// Obtener el número de filas afectadas
$filaAfectadas = $stmt->rowCount();

echo "Filas eliminadas: $filaAfectadas";
?>
```

6.4.6 Transacciones con PDO

Las transacciones permiten agrupar varias operaciones SQL en una sola unidad lógica. Si alguna operación falla, se puede revertir todo el conjunto, garantizando la integridad de los datos.

Conceptos Clave

1. **Transacción:**
    - Es un conjunto de operaciones SQL que se ejecutan como una unidad indivisible.
    - Garantiza las propiedades ACID (*Atomicidad, Consistencia, Aislamiento y Durabilidad*).
2. **Inicio de Transacción:**
    - Se inicia una transacción con el método beginTransaction().
3. **Confirmación (*Commit*):**
    - Si todas las operaciones tienen éxito, los cambios se confirman con commit().
4. **Reversión (*Rollback*):**
    - Si ocurre un error, los cambios realizados desde el inicio de la transacción se revierten con rollBack().

Métodos Principales

- beginTransaction():
    - Inicia una nueva transacción.
    - El modo automático de confirmación de consultas se desactiva hasta que se ejecute un commit() o un rollBack().
- commit():
    - Confirma los cambios realizados dentro de la transacción.
- rollBack():
    - Revierte todos los cambios realizados desde el inicio de la transacción.
- inTransaction():
    - Devuelve true si hay una transacción activa, o false en caso contrario.
- Manejo de errores con excepciones:
    - Es recomendable usar bloques try-catch para capturar errores y asegurar el rollBack() en caso de fallo.

Ejemplo Práctico

```php
<?php
try {
    $pdo = new PDO('mysql:host=localhost;dbname=mi_base_datos;charset=utf8', 'usuario', 'con
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Iniciar la transacción
    $pdo->beginTransaction();

    // Sentencia preparada para insertar usuarios
    $stmt = $pdo->prepare("INSERT INTO usuarios (nombre, email) VALUES (:nombre, :email)");
    $stmt->execute([':nombre' => 'Ana López', ':email' => 'ana@example.com']);

    // Sentencia preparada para actualizar una cuenta
    $stmt = $pdo->prepare("UPDATE cuentas SET saldo = saldo + :monto WHERE id_usuario = :id"
    $stmt->execute([':monto' => 200, ':id' => 2]);

    // Confirmar la transacción
    $pdo->commit();
    echo "Transacción completada con éxito.";
} catch (Exception $e) {
    // Revertir la transacción en caso de error
    $pdo->rollBack();
    echo "Error en la transacción: " . $e->getMessage();
}
?>
```