Unitat Formativa 6. POO. Introducció a la persistència en BD

1. Disseny de programes que accedeixen a BD

Característiques i mètodes d'accés als SGBDR's (Sistemes Gestors de Bases de Dades Relacionals)

Una base de dades es una col·lecció de fitxers amb una estructura definida i amb unes relacions entre els mateixos. Per accedir a la informació guardada a les bases de dades es desenvolupen **extensions** i **API's** (Interfícies per a la Programació d'Aplicacions). Es necessari un cert coneixement del llenguatge de consulta SQL i de la estructura d'una base de dades per poder formular les peticions.

PHP descriu les seves extensions segons la tecnologia d'accés que volem fer servir o amb quina base de dades hem de treballar: https://www.php.net/manual/es/refs.database.php

Un exemple clàssic de treball es la combinació del Llenguatge PHP amb la base de dades MySQL i es pot comprovar com aquesta extensió (**MySQLi**) està molt més desenvolupada que la resta:

https://www.php.net/manual/es/book.mysqli.php

Aquest enllaç és un índex de enllaços a explicacions i exemples de totes les possibilitats, ens ajuda a aprofundir tant com sigui necessari amb el tractament de dades procedents de MySQL. També amb l'anterior enllaç coneixerem les possibilitats de treballar amb altres gestors de fitxers de dades.

1.1. Formes de establir la connexió amb la base de dades

Les extensions que hem vist a l'apartat anterior ens ajuden a treballar de forma simple, però tenim diverses possibilitats. No aprofundirem als aspectes del disseny de les bases de dades ni del llenguatge SQL, però si amb les tecnologies que podem fer servir per accedir a les bases de dades i explotar la seva informació. La documentació ens avisa que disposem d'una interfície "dual":

https://www.php.net/manual/es/mysqli.quickstart.dual-interface.php

- Una interfície està basada en una *extensió procedimental* **mysqli** (basada en funcions) per a projectes on es treballa amb POO i que substitueix a la obsoleta extensió **mysql** (amb funcions que ja no es recomana utilitzar per insegures): https://www.php.net/manual/es/ref.mysqli.php

- I l'altra interfície es bassa en la *Programació Orientada a Objectes*, amb la classe **mysqli**: https://www.php.net/manual/es/mysqli.summary.php

Si estudiem una mica l'enllaç veurem que podem servir qualsevol de les dues possibilitats amb uns canvis de sintaxi mínims, però aquí volem mostrar les possibilitats que ofereix la **POO**.

Obviem per ara la opció **PDO,** molt utilitzada en grans projectes:
https://code.tutsplus.com/es/tutorials/why-you-should-be-using-phps-pdo-for-database-access--net-12059

Al següent enllaç es mostren les característiques internes de la classe **mysqli**, els seus atributs i els seus serveis o funcions: https://www.php.net/manual/es/class.mysqli.php

(Al final de la pàgina de l'enllaç un usuari ens ha deixat un exemple de com podríem enllaçar les classes que cobreixen els conceptes del nostre negoci o activitat, i les classes que cobreixen la part de la seva salvaguarda i recuperació cap a una base de dades, genèricament es coneix com **persistència**).

1.2. Operacions inicials sobre bases de dades

Tenim un ampli ventall d'operacions per a l'apertura/tancament de connexions, i de lectura/escriptura sobre les dades: https://www.php.net/manual/es/book.mysqli.php

Les més comuns:

- Establir la connexió, ja sigui amb l'estil procedimental u orientat a objectes: https://www.php.net/manual/es/mysqli.construct.php
- Executar una consulta: https://www.php.net/manual/es/mysqli.query.php
- Tractament dels resultats, on disposem d'una col·lecció de funcions específica: https://www.php.net/manual/es/mysqli-result.fetch-assoc.php

1.3. Modularització de les operacions sobre les bases de dades

Els mètodes de la classe mysqli ja ofereixen molts serveis, però ens pot interessar un encapsulament de nivell superior, per aconseguir una major simplicitat com es mostra als exemples, on veiem com podem treballar directament amb la classe mysqli o bé dissenyar una nova classe que agrupi operacions o serveis habituals a un sol mètode.

Per exemple. Dissenyem una classe que contingui els camps o atributs que necessitem tractar als usuaris del nostre portal, i ens queda una classe semblant a la següent:

```
declare(strict_types=1);

class User {
   protected int $id;
   protected string $name;
   protected int $level;
   protected int $points;
   protected string $password;
```

```
    public function __construct(int $id, string $name, string $password, int $level = 1, int $points = 0) {
        $this->id = $id;
        $this->name = $name;
        $this->password = $password;
        $this->level = $level;
        $this->points = $points;
    }

    public function getId(): int {
        return $this->id;
    }

    public function getName(): string {
        return $this->name;
    }

    public function getLevel(): int {
        return $this->level;
    }

    public function getPoints(): int {
        return $this->points;
    }

    public function getPassword(): string {
        return $this->password;
    }

    public function setId(int $id): void {
        $this->id = $id;
    }

    public function setName(string $name): void {
        $this->name = $name;
    }

    public function setLevel(int $level): void {
        $this->level = $level;
    }

    public function setPoints(int $points): void {
        $this->points = $points;
    }

    public function setPassword(string $password): void {
        $this->password = $password;
    }
}
```

Al treballar amb una base de dades és útil fer servir algun tipus "d'adaptador" que ens faciliti la gestió de la connexió i consulta de les dades que emmagatzema. Hem vist classes i funcions que ens ho permeten, però encara així no volem que es coneguin alguns detalls interns com usuari i contrasenya, el nom i port exacte de la base de dades...

Ni els usuaris han de conèixer com es comprova que la connexió s'ha establert amb èxit i com obtenir l'error en cas contrari, com alliberar els recursos després de la desconnexió, gestionar com es presentaran les dades pel seu posterior tractament.

Un exemple de adaptador per un treball genèric amb MySQL podria ser el següent:

```
declare(strict_types=1);

class MysqlAdapter {
   protected mysqli $connection;
   protected bool $connected = false;

   public function __construct () {
      try {
         $this->connection = new mysqli("127.0.0.1", "userdaw1", "M3phpdaw@", "softlearning", 3306);
         $this->connected = true;
      } catch (mysqli_sql_exception $ex) {
         throw new ServiceException("DB Connection Failure: " . $ex->getMessage());
      }
   }

   public function __destruct() {
      $this->closeConnection();
   }

   public function isConnected(): bool {
      return $this->connected;
   }

   public function connect (string $host, int $port, string $user, string $password, string $db): void{
      if ($this->connected == true) {
         $this->closeConnection();
      }
      try {
         $this->connection = new mysqli ($host, $user, $password, $db, $port);
         $this->connected = true;
      } catch (mysqli_sql_exception $ex) {
         throw new ServiceException("DB Connection Failure: " . $ex->getMessage());
      }
   }

   public function closeConnection() {
      if ($this->connected == true) {
         $this->connection->close();
         $this->connected = false;
      }
   }
```

```
    public function readQuery(string $query): array {
        $dataset[] = null;
        $result = $this->connection->query($query);
        if ($result != false) {
            while ($row = $result->fetch_assoc()) {
                $dataset[] = $row;
            }
        }
        return $dataset;
    }

    public function writeQuery(string $query): bool {
        $result = $this->connection->query($query);
        return $result;
    }

}
```

Com podem comprovar aquesta classe incorpora un altre mètode que es cridarà per defecte: **__destruct**, en aquest cas es cridarà quan es destrueixi l'objecte igualant-lo a null.

Encapsulament de serveis en classes.

Analitzant els serveis genèrics encapsulats per aquesta classe observem que tenim creada una base per obrir, tancar i reobrir connexions, realitzar consultes de lectura i escriptura, e inclús transformar les consultes de lectura a arrays associatius...

A partir d'aquest component podem crear d'altres més específics per que treballin amb les dades dels nostres objectes de negoci.

Prenem com exemple els objectes de la classe **User**. Crearem una classe més específica que heretarà de **MysqlAdapter** i que oferirà els serveis específics per realitzar les operacions **CRUD** (**Create-Read-Update-Delete**) amb els objectes **User**.

Aquesta classe que anomenarem **MysqlUserAdapter** ens permetrà realitzar aquestes operacions fàcilment a partir dels seus mètodes, encapsulant al seu interior tots els detalls, el codi SQL natiu de MySQL, el tractament d'Excepcions, etc. Tota la part específica queda encapsulada dins i de cara al seu ús només cal cridar als mètodes i gestionar les possibles excepcions de tipus **ServiceException** que puguin generar-se.

A continuació es mostra un exemple de com encapsular aquests serveis dins d'una classe específica (al projecte **DemoDB** es mostra com es validen els seus mètodes a partir d'uns tests funcionals):

```
class MysqlUserAdapter extends MysqlAdapter {

   public function getUser ( int $id ) : User {
      $data = $this->readQuery ("SELECT id, name, password, level, points FROM users WHERE id=" .
                                    $id . ";");
      if ( count ( $data ) > 0 ) {
         return new User ( (int) $data[0]["id"], $data[0]["name"], $data[0]["password"],
               (int) $data[0]["level"], (int) $data[0]["points"]  );
      }else {
         throw new ServiceException("Not User found with id = " . $id);
      }
   }

   public function deleteUser ( int $id ) : bool {
      try {
         return $this->writeQuery("DELETE FROM users WHERE id = " . $id . ";");
      } catch (mysqli_sql_exception $ex) {
         throw new ServiceException ("Error al esborrar al usuari " . $id . "-->" .  $ex->getMessage() );
      }
   }

   public function addUser ( User $u ) : bool {
      try {
         return $this->writeQuery ( "INSERT INTO users (id,name,password,level,points)" .
                  " VALUES (" . $u->getId() . ",\"" . $u->getName() . "\",\"" .
                  $u->getPassword() . "\"," . $u->getLevel() . "," . $u->getPoints() . ");" );
      } catch (mysqli_sql_exception $ex) {
         throw new ServiceException ( "Error al insertar usuari -->" .  $ex->getMessage() );
      }
   }

   public function updateUser ( User $u ) : bool {
      try {
         return $this->writeQuery ( "UPDATE users SET name = \"" . $u->getName() . "\"," .
                  "password = \"" . $u->getPassword() . "\"," .  "level = " . $u->getLevel() . "," .
                  "points = " . $u->getPoints() .   " WHERE id = " . $u->getId() . ";");
      } catch (mysqli_sql_exception $ex) {
         throw new ServiceException("Error al actualitzar usuari -->" .   $ex->getMessage());
      }
   }
}
```

Aquesta classe ens ofereix una interfície pública molt senzilla de utilitzar, amb uns mètodes que gestionen internament les operacions amb una BD. Es una alternativa al ús dels **ORM** (Object-Relational Mapping) com **Doctrine** o **Eloquent** utilitzats ens projectes amb **Frameworks** com **Symphony** o **Laravel**.