# TP 7 : Mise en place d'une Base de Données

---

## Objectifs du TP:

- Créer une BD au format sqlite.
- Utiliser cette BD avec PDO dans un modèle.
- Récupérer les lignes de la BD dans des objets.
- Obtenir un site Web dynamique fonctionnel avec une BD

---

## 1. Accéder à une base de donnée depuis PHP

On désire afficher des contacts stockés dans une base de donnée. Un contact est définit par :

- name : prénom de la personne.
- phone : son téléphone.
- city : sa ville de résidence.

La base de données contient une seule table définie de la manière suivante :

```
CREATE TABLE contact (
  id INTEGER PRIMARY KEY,
  name STRING,
  phone INTEGER,
  city STRING
);
```

### 1.1 Construction de la BD avec SQlite

Nous allons utiliser d'abord SQlite pour construire une base de donnée locale donc sans serveur. Les données brutes sont dans le fichier `data/contact.txt`. Chaque enregistrement de la base de données est sur une ligne, séparée par le caractère `|`. Le but de la partie suivante est de créer et charger une base de données.

**A faire**

- Se placer dans le répertoire `data`.
- Créer le fichier `create.sql` qui doit contenir la définition en SQL de la table `contact` (cf. ci-dessus).
- Ouvrir la base de données dans le fichier `contact.db` avec la commande (dans le shell) :

  ```
  sqlite3 contact.db
  ```

- Pour charger le schéma de la BD, lancer la commande (dans sqlite):

  ```
  sqlite> .read create.sql
  ```

- Vérifier que le schéma de la base est correct avec la commande (dans sqlite):

  ```
  sqlite> .schema
  ```

- Charger les données dans la base avec la commande (dans sqlite) :

  ```
  sqlite> .import contact.txt contact
  ```

- Tester le chargement avec une requête SQL (dans sqlite):

  ```
  sqlite> SELECT * FROM contact WHERE city='Dallas';
  ```

### 1.2 Utilisation de la BD avec PDO

La couche logicielle `PDO` permet d'accéder à une base de données. Il faut créer une instance de la classe `PDO` pour lui passer le `dataSourceName` : il s'agit d'une chaine qui indique le type de base de données ainsi que sa localisation.

```
$db = new PDO($dataSourceName);
```

Dans cet exercice le type de BD est `sqlite`. Il faut donc simplement indiquer ensuite le chemin (path) local vers le fichier qui contient la base de données. La constante prédéfinie `__DIR__` indique le chemin absolu du fichier php qui le contient. L'utilisation de cette constante permet de contruire facilement un chemin absolu.

```
$dataSourceName = 'sqlite:'.__DIR__.'/data/contact.db';
```

Une fois la BD ouverte (i.e. création de l'instance PDO), on peut créer une requête avec la méthode `prepare`. La requête est du SQL dans une chaîne de caractère avec des parties variables nommées par `:XXX`. Ces parties variables seront connu au moment de l'exécution de la requête.

IMPORTANT: une requête SQL est un code exécutable d'un langage de programmation. Manipuler dans une chaîne un code exécutable représente un potentiel trou de sécurité. Il ne faut donc pas être construire dynamiquement une requête (concaténation, interpolation de variables, ...) mais au contraire une requête doit être une constante chaîne avec des simples quotes.

```
const QUERY = 'SELECT * FROM contact where city=:city';
$requete = $db->prepare(QUERY);
```

La préparation de la requête consiste en sa compilation pour être envoyée au serveur SQL. Pendant le développement, il est important de vérifier si la compilation de la requête (prepare) a bien fonctionné. On doit alors capturer l'exception (ou la valeur de retour qui vaut FALSE), et afficher un message d'erreur en indiquant clairement la requête pour faciliter la mise au point.

```
try {
  $requete = $db->prepare(QUERY);
} catch (PDOException $e) {
  die('PDO query Error on "'.QUERY.'" '.$e->getMessage());
}
```

Si la préparation de la requête fonctionne, on peut alors utiliser cette requête en fixant des valeurs aux parties variables dans un tableau avec la syntaxe `':NOM' => valeur`. La valeur peut etre le résultat d'une expression PHP.

```
$requete->execute([':city' => $city]);
```

La récupération du résultat de l'exécution de la requête se fait avec la méthode fetchall.

```
$data = $requete->fetchall();
```

Les données sont alors disponibles dans un tableau (array) de tableaux : il s'agit de la répresentation d'une table SQL. Chaque élément du tableau `$data` est un tableau représentant une ligne. On récupère chaque donnée de cette ligne avec un index qui correspond **à un nom d'attributs SQL**. Par exemple, avec la requête précédente sur la table `contact` chaque ligne `$i` de `$data` contient des valeurs que l'on peut récupérer dans des variables :

```
<?php
...
// Récupère la ligne $i du résultat de la requête
$line = $data[$i];

// Accès à toutes les colonnes de la ligne $i
$id    = $line['id'];
$name  = $line['name'];
$phone = $line['phone'];
$city  = $line['city'];
```

On peut ainsi traiter toutes les lignes avec `foreach` pour les placer dans une table :

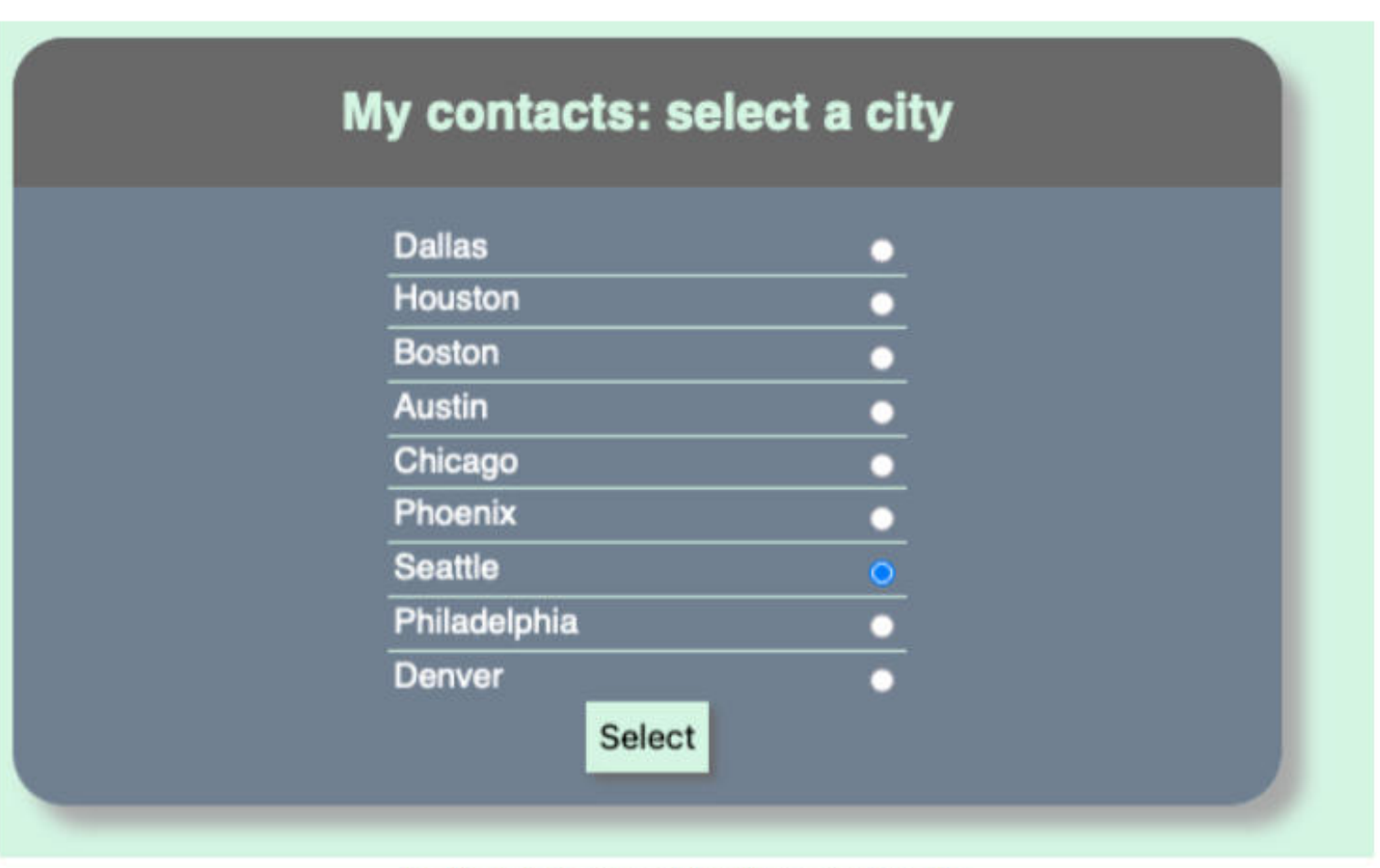
```
<?php foreach ($data as $contact): ?>
  <tr>
    <td>
      <?= $contact['name'] ?>
    </td>
  </tr>
<?php endforeach; ?>
```

**A faire :**

- Télécharger le fichier data/contact.zip .
- Compléter le fichier `contact.php` pour faire afficher la liste des contacts qui habitent la ville de Dallas. Voous devez interroger la base de données en `SQlite` avec la bibliothèque `PDO`.

### 1.3 Selectionner une ville

La page `contact.php` affiche les contacts d'une ville. Nous proposons d'ajouter la page `selectCity.php` qui permet de choisir une ville parmi celle qui existent dans la base de données, puis affiche les contacts de cette ville.



*Formulaire de sélection de la ville*

**A faire :**

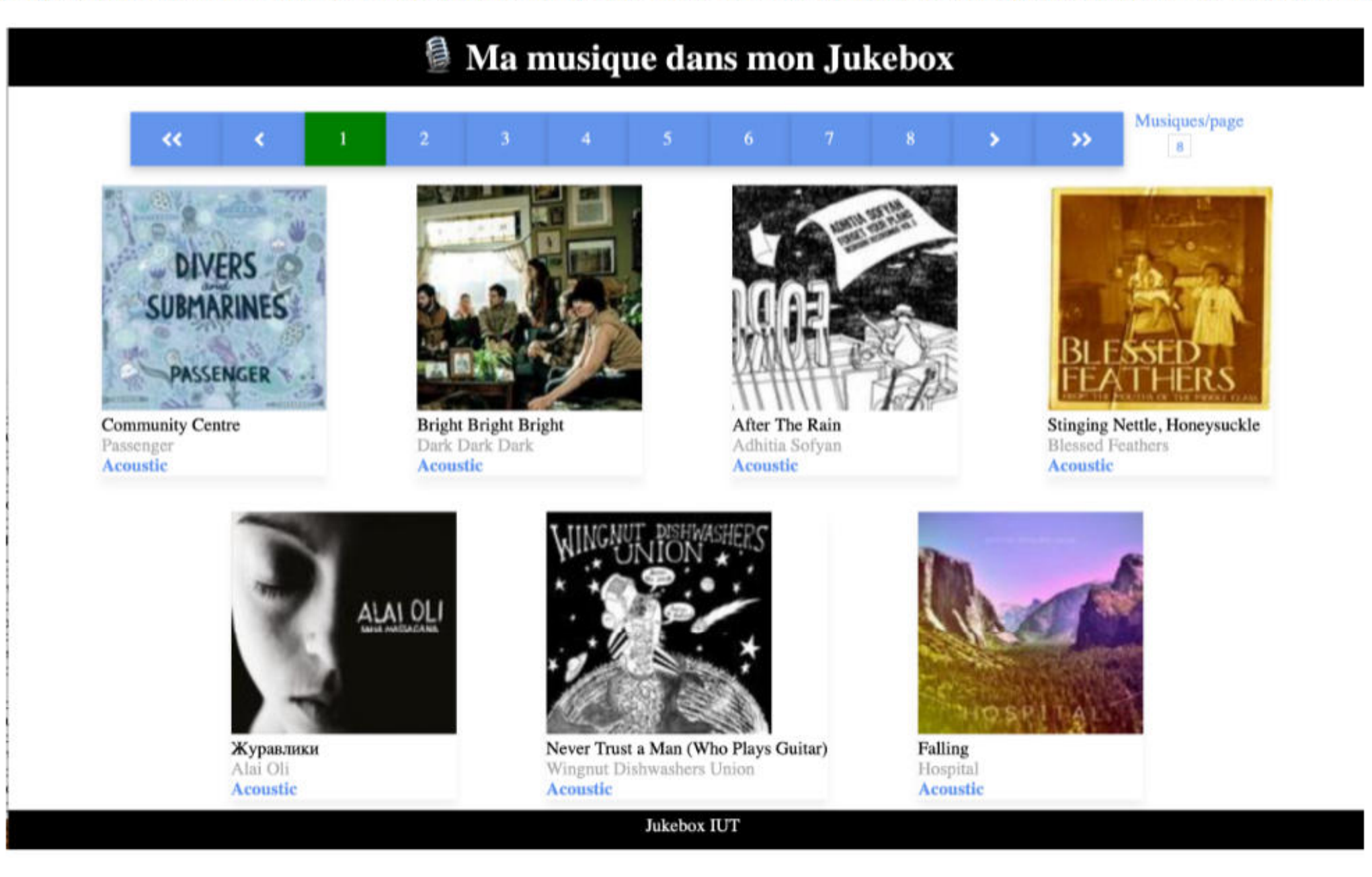
- Créer le fichier `selectCity.php` qui propose un formulaire avec la liste des villes (uniques) de la table `contact`. Ce formulaire a des entrées de type `radio` à l'intérieur d'un `label` et un bouton d'envoi du choix à `contact.php`.

## 2. Site Web : Jukebox avec une base de données

Dans un TP précédent, vous avez construit un site dynamique qui manipule une grande quantité de données sous la forme de fichiers. Dans cette partie, nous allons remplacer certains fichiers par une base de données.

### 2.1 Analyse

L'objectif de cet exercice est de réaliser un site donnant accès à des musiques en naviguant dans les pages affichant une image, un titre, un auteur et une catégorie. On sélectionne le titre à écouter en cliquant sur son image. Un menu (en bleu) affiche un certain nombre de pages et 4 boutons. Ces boutons permettent d'aller à la première ou à la dernière page, ou de se déplacer de 8 pages à la fois. Cliquer sur un numéro de page donne un accès direct à cette page. Une petite case à droite permet de changer le nombre de musiques affichées par page.



### 2.2 Conception

Pour réaliser ce site, nous proposons de créer dans un modèle la notion de `Music` qui représente une musique à jouer avec les attributs suivants :

- id : l'identifiant (unique) de la musique.
- author : le nom de l'auteur.
- title : le titre du morceau de musique.
- cover : le nom du fichier image associé à la musique.
- mp3 : le nom du fichier mp3 associé à la musique.
- category : le nom de la catégorie de musique associée à ce morceau.

La classe `Music` sera dans le fichier `music.class.php` dans le sous-répertoire `model`.

Pour réaliser le contrôle et l'affichage de cette application nous proposons deux fichiers :

- `jukebox.php` : le point d'entrée de l'application. Il affiche la liste des musiques, le passage de page en page, et la modification du nombres d'éléments par page.
- `playId.php` : est chargé de jouer une musique en indiquant son `id` de la base de données. On peut revenir à la page principale, mais en conservant la page sur laquelle on était, ainsi que le nombre de musiques par page.

**À faire**

- Télécharger une version incomplète de cette application à partir du lien suivant : data/dataJukeboxDB.zip
- Installer ce code sur votre serveur WEB, le faire fonctionner et examiner les fichiers.

## 3. Mise en place de la base de données

Comme la base de données est de petite taille, et pour éviter une gestion d'un véritable serveur de base de données, nous vous proposons d'utiliser le système de base de données `sqlite`. Il permet de construire une base de données complète dans un seul fichier. Cette base de données est accessible par la couche `PHP Data Object` vue en cours (PDO).

La première étape consiste à créer le schéma des tables de la base de données dans un fichier `create.sql`, de créer les tables avec ce fichier, puis de charger les données dans la BD. Le fichier des données est `data/musicDB.txt`. Il contient la liste des musiques disponibles. Chaque musique est décrite sur une ligne. Les champs de cette ligne sont séparés par le caractère `|`.

Exemple :

```
1|Passenger|Community Centre|1.jpg|1.mp3|Acoustic
```

La signification dans l'ordre (de gauche à droite) est la suivante :

- Numéro unique d'identifiant de la musique : "1".
- Nom de l'auteur (artiste ou groupe) : "Passenger"
- Titre du morceau : "Community Centre"
- Nom du fichier image associée (format jpeg ou png) : "1.jpg"
- Nom du fichier audio au format mp3 : "1.mp3"
- Catégorie musicale : "Acoustic"

La base de données doit être composée d'une seule table avec les champs suivants :

- id : l'identifiant (unique) de la musique.
- author : le nom de l'auteur.
- title : le titre du morceau de musique.
- cover : le nom du fichier image associé à la musique.
- mp3 : le nom du fichier mp3 associé à la musique.
- category : le nom de la catégorie de musique associée à ce morceau.

**Travail à réaliser :**

1. Aller dans le sous-répertoire `data` qui va contenir le fichier de la base de données.
2. Donner les droits (chmod) de ce répertoire aux autres (other) en écriture (w) et en accès (x).
3. Créer un fichier `create.sql` contenant la définition d'une table SQL définie de la manière suivante :

   ```
   CREATE TABLE music (
     id INTEGER PRIMARY KEY,
     author STRING,
     title STRING,
     cover STRING,
     mp3 STRING,
     category STRING
   );
   ```

4. Démarrez le gestionnaire shell `sqlite3` de base de données pour créer une base dans le fichier `music.db` (db pour "database") :

   ```
   sqlite3 music.db
   ```

5. Examinez la listes des commandes de sqlite sur le site https://www.sqlite.org/cli.html et par la commande :

   ```
   sqlite> .help
   ```

6. Créer la table `music` en chargeant sa définition depuis le fichier `create.sql` :

   ```
   sqlite> .read create.sql
   ```

7. Chargez le fichier des données dans la table en indiquant le type du séparateur :

   ```
   sqlite> .separator |
   sqlite> .import musicDB.txt music
   ```

8. Vérifiez que votre base de données fonctionne à l'aide d'une requête SQL comme :

   ```
   sqlite> SELECT author FROM music;
   ```

9. Sortir du shell de sqlite par la commande :

   ```
   sqlite .quit
   ```

   Pour examiner à nouveau le contenu de la base de données stockée dans le fichier `music.db` reprendre au point No 5.

### 3.1 Aide technique

Dans cet exercice nous vous proposons d'utiliser une classe `DAO` simplifiée pour encapsuler les accès PDO à la Base de Données, principalement pour détecter et afficher les erreurs. Cette classe n'est pas indispensable pour ce TP, néanmoins elle permet de simplifier un peu l'usage de la BD. Elle possède deux méthodes :

- `function __construct()` : le constructeur permet d'ouvrir la Base de Données avec PDO, et de gérer les erreurs.
- `public function prepare(string $query) : PDOStatement` : prépare une requête, s'il y a un problème affiche le message d'erreur et la requête et termine le programme.

L'accès à la base de données doit se faire avec cette classe DAO et la sur-couche PHP Data Object (PDO). L'objet DAO se connecte automatiquement à la base `music.db`. Le code de cette connexion est dans le constructeur de `DAO`. Dans ce TP on associe des données d'une Base de Données relationnelle à un objet dans le langage de programmation : on nomme cela Object-Relational Mapping (ORM).

Important : il faut placer les droits en **écriture** pour tous (chmod) sur le fichier de la base de données, **et également sur le répertoire qui contient ce fichier**. En effet, SQlite va potentiellement modifier le fichier et créer des fichiers temporaires dans le répertoire où se trouve le fichier BD.

C'est à la méthode `read` de la classe `Music` de se connecter à la BD avec un objet DAO, de créer une requête et d'interroger la BD puis créer un nouvel objet Music avec les données extraites de la BD. Après avoir préparé, puis exécuté la requête SQL, vous devez récupérer les données avec `fetchAll` de PDO. **Attention** : même si une requête SQL ne produit qu'un seul élément, le résultat d'une requête de type `SELECT` se **trouve toujours dans un tableau (array)** à plusieurs dimension.

### 3.2 Developpement de la classe Music

La classe `Music` dans le fichier `model/music.class.php` représente une musique issue de la base de données. Elle est composé des attributs privés identiques à ceux de la BD. Les getters permennet d'accéder à ces attributs.

La méthode de classe :

```
    public static function read(int $id): Music
```

retourne un **nouvel objet** de la classe `Music` ayant l'identifiant `$id`. Pour cela utilisez un objet DAO et la requête SQL :

```
SELECT * FROM music WHERE id=:id
```

Une fois les données lues, créez un nouvel objet Music et retourner sa référence.

La méthode de classe :

```
    public static function maxId() : int
```

retourne la valeur du plus grand identifiant connu. Cette valeur est une constante pour simplifier le code.

L'URL des images et des fichiers MP3 n'est pas stockée dans la BD, mais c'est une constante de la classe Music.

```
  // Chemin URL à ajouter pour avoir l'URL du MP3 et du COVER
  private const URL = 'http://www-info.iut2.upmf-grenoble.fr/intranet/enseignements/ProgWeb/data/musiques/';
```

Ce sont les getter `getCover()` et `getMp3()` qui doivent construire une URL complète avec le chemin correct vers l'image et la musique.

**À faire :**

- Compléter le constructeur de la classe `Music`.
- Ajouter les getters avec une méthode par attribut `xxx` de nom `getXxx()`.
- Complétez la méthode de classe `read`.
- Tester cette classe en activant en ligne le fichier `music.test.php` dans le répertoire `test`. Vous devez obtenir le résultat suivant :

  ```
  php music.test.php
  OK : la classe 'Music' a passé les tests
  ```

### 3.3 La page playId.php

Cette page fonctionne en indiquant dans l'URL, la musique à jouer par exemple :

```
http://localhost/Jukebox_BD/playId.php?id=1&page=1&pageSize=8
```

Les valeurs supplémentaires `page` et `pageSize` sont présentes pour conserver l'état de la page principale, et ainsi lorsqu'on active le bouton 'back' on se retrouve dans l'état précédent.

**À faire**

- Faire fonctionner cette page qui joue la musique indiquée dans l'`id`. Dans cette première version, vous pouvez ignorer les deux autres valeurs de l'URL.

### 3.4 La page principale : jukebox.php

Cette page affiche les 8 première musiques de la base de données. La version initiale de cette page est statique. Il faut remplacer le code HTML statique par du code produit par PHP.

**À faire :**

- Compléter le fichier `jukebox.php` pour qu'il affiche les 8 premières musiques, de manière identique à la page statique.

### 3.5 Gestion des liens sur la page

Dans cette partie, vous apprenez à mettre en place une interaction avec une mémoire de l'interaction : l'utilisateur doit pouvoir interagir avec la page principale et faire "défiler" les musiques du jukebox, de manière à avoir accès à toutes les musiques disponibles. Cette interaction possède une "mémoire" ou un "état" qui doit être conservé de page en page. Cette mémoire est simplement le numéro de la page affichée.

Nous utilisons simplement un paramètre supplémentaire dans l'URL qui indique le numéro de la page : `page`.

**À faire**

- Modifier le menu qui est actuellement statique, par une version dynamique produite par le PHP. En particulier, il faut remplacer la liste des boutons :

  ```
  <a href="jukebox.php?page=2&pageSize=8">2 </a>
  ```

  par une boucle en PHP.
- Compléter le formulaire qui donne le nombre de musiques à afficher pour qu'il change le nombre de musiques. Attention : vous devez passer la page courante comme une entrée cachée du formulaire (hidden input) pour qu'elle soit ajouté à l'URL et ainsi conserve l'informations.