PHP – MYSQL

http://php.net/manual/fr/langref.php
http://www.w3schools.com/php/
https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql

SOMMAIRE

PHP – MYSQL

1. Bibliothèques utilisées

Utilisation de la base de données en PHP

- 3 jeux de fonctions (API) permettent de se connecter à la BD et de l'utiliser :
  - mysql,
  - mysqli
  - PDO
    - ➔ http://php.net/manual/fr/mysqlinfo.api.choosing.php
- Le jeu mysql est le plus ancien : mieux vaut l'éviter.
- Mysqli permet d'utiliser MySQL et est plus facile à mettre en œuvre. Il vaut mieux l'éviter
- PDO permet d'utiliser n'importe quel SGBD et est plus complexe à mettre en œuvre.
  - On utilisera préférentiellement PDO_MYSQL.

2. Création d'une BD en SQL

Pour nos codes, d'exemple, on suppose une BD existant dont le code de création est le suivant :

```
DROP database if exists DB_movies_anthropocene;

CREATE database DB_movies_anthropocene;
USE DB_movies_anthropocene;

CREATE table movies(
   id integer auto_increment,
   titre varchar(30) not null,
   annee integer not null,
   realisateur varchar(20) not null,
   primary key(id)
);

INSERT INTO movies VALUES
   (null, "Une vérité qui dérange", 2006, "Davis Guggenheim"),
   (null, "Don't look up", 2021, "Adam McKay"),
   (null, "Beasts of the southern wild", 2012, "Benh Zeitlin"),
   (null, "Melencholia", 2011, "Lars von Trier"),
   (null, "Sauve qui peut (la vie)", 1980, "Jean-Luc Godard"),
   (null, "Nope", 2022, "Jordan Peele"),
   (null, "L'amour à mort", 1984, "Alain Resnais");
```

PHP – MYSQL – V1 : PDO

00. Présentation du document

Objectifs généraux de ce document

- Utilisation de la bibliothèque PDO
- Usage de la programmation objet en PHP
- Select et DML en PDO
- Organisation du code non MVC

Exemples

1. Connexion à la BD : new PDO (exemple-1 – connexion)
2. Afficher le contenu d'une table -1 : query(), fetch(), print_r (exemple-2-select)
3. Accès aux champs, order by, like, limit (exemple-3-Select Where)
4. Requêtes variables : where realisateur = ? (exemple-4-Select Variable)
5. Ajouter, modifier, supprimer des données dans une table (exemple-5-insert-update-delete)
6. Exemple-6 : Site Artiste - Etape 1 : uniquement les œuvres

01. Bibliothèques utilisées

Utilisation de la base de données en PHP

- 3 jeux de fonctions (API) permettent de se connecter à la BD et de l'utiliser :
  - mysql,
  - mysqli
  - PDO
    - ➔ http://php.net/manual/fr/mysqlinfo.api.choosing.php
- Le jeu mysql est le plus ancien : mieux vaut l'éviter.
- On peut utiliser mysqli ou PDO (PHP Data Object), et particulièrement PDO_MYSQL.

PDO ou mysqli ? PDO !!!

- https://websitebeaver.com/php-pdo-vs-mysqli
- https://openclassrooms.com/forum/sujet/debat-pdo-vs-mysqli
- Dans les framework modernes, c'est PDO qui est utilisé.
- L'intérêt du PDO est que c'est une interface d'abstraction permettant l'utilisation de n'importe quelle BD. De plus elle est « orienté objet ».
  - ➔ On utilisera plutôt PDO_MYSQL.
  - ➔ http://php.net/manual/en/ref.pdo-mysql.php

BD utilisée

On va créer la BD suivante (fichier BD_Utilisateur.sql de l'exercice 1).

```
drop database if exists BD_Utilisateurs;
create database BD_Utilisateurs;

use BD_Utilisateurs;

CREATE TABLE Utilisateurs (
  id int(11) AUTO_INCREMENT,
  prenomNom varchar(20) NOT NULL,
  adMail varchar(20) NOT NULL,
  motDePasse varchar(20) NOT NULL,
  annee int(4) NOT NULL
  primary key(id)
) ENGINE=InnoDB;

Insert into Utilisateurs values (NULL, 'Sia PEII',
'ji@gmail.com','jipei', 1995);
Insert into Utilisateurs values (NULL, 'Yawei CAI',
'jawei@yahoo.com','yaweicai',1996);
Insert into Utilisateurs values (NULL, 'Zikeng PENG',
'zikeng@china.com','zikeng',1994);
Insert into Utilisateurs values (NULL, 'Jiawen LI',
'jiawen@orange.fr','jiawen',1995);
Insert into Utilisateurs values (NULL, 'Xiaoyu LIU',
'xiowyu@gmail.fr','liu',1996);
Insert into Utilisateurs values (NULL, 'Olivier
TRAN','tran@gmailcom','olivier',1997);
```

1. Connexion à la BD et print_r du contenu d'une table

Connexion à la BD : new PDO (exemple-1 – connexion)

Code

```php
<?php
function connexionBD($dbname){
    // paramètres de la base de donnée
    $sgbdname='mysql';
    $host='localhost';
    $charset='utf8';
    // dsn : data source name
    $dsn =
        $sgbdname .
        ':host='.$host .
        ';dbname='.$dbname.
        ';charset='.$charset;

    // utilisateur connecté à la base de donnée
    $username = 'root';
    $password = '';

    // pour avoir des erreurs SQL plus claires
    $erreur = array(
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    );

    try {
        // connexion à la BD : new PDO
        $bdd = new PDO($dsn, $username, $password, $erreur);
        echo'<p>Connexion réussie</p>';
        return $bdd;
    } catch (PDOException $e) {
        echo 'Connexion échouée : ' . $e->getMessage();
        return NULL;
      // die ('Connexion échouée : ' . $e->getMessage() );
      // pas de die avec PDO
    }
}
?>
```

new PDO

On fait un new PDO avec 4 paramètres

- PDO est une classe.
  ➔ http://php.net/manual/fr/pdo.construct.php
- On crée un nouvel objet de la classe qu'on appelle $bdd.
- Le new PDO à 4 paramètres :
  ➔ $**dsn** (data source name) : contient des infos sur le SGBD (mysql), le serveur (host, ici : localhost), le nom de la BD, le jeu de caractères utilisé (UTF8 pour que ce soit le plus générique).
  ➔ $**username** : nom de l'utilisateur qui se connecte à la BD.
  ➔ $**password** : password de l'utilisateur qui se connecte à la BD.
  ➔ $**erreur** : pour gérer les messages d'erreur.

On utilise des variables pour rendre le code lisible et facile à paramétrer :

- $host : la machine du serveur de SGBD,
- $sgbdname : le type de SGBD,
- $username : le nom de l'utilisateur qui se connecte sur la BD,
- $password : le mot de passe de cet utilisateur,
- $dbname : le nom de la BD à laquelle on accède sur le SGBD.

$erreur : gestion des erreurs

```
$erreur = array(
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
);
```

- En ajoutant le paramètre $erreur tel qu'il est là dans le new PDO, on aura des messages d'erreurs du SGBD, par exemple si le SELECT est mal écrit.
- $erreur est un tableau associatif. On lui fournit un couple key => value.
  - ➔ key : PDO ::ATTR_ERRMODE
  - ➔ value : PDO::ERRMODE_EXCEPTION
- Syntaxe de POO-PHP :
  - ➔ PDO ::ATTR_ERRMODE et PDO::ERRMODE_EXCEPTION sont des constantes de classe définies dans la classe PDO.
  - ➔ https://www.w3schools.com/php/php_oop_constants.asp
  - ➔ Classe::CONSTANTE_DE_CLASSE
  - ➔ PDO ::ATTR_ERRMODE : http://php.net/manual/fr/pdo.constants.php
  - ➔ PDO::ERRMODE_EXCEPTION : http://php.net/manual/fr/pdo.error-handling.php

Gestion des erreurs avec try catch

- La structure « try » « catch » est une structure de programmation objet pour gérer les erreurs : les exceptions.
  - ➔ On essaye un code : try
  - ➔ Si ça « plante », on attrape une exception et on la gère
  - ➔ https://www.w3schools.com/php/php_exception.asp
- Le die permet d'arrêter proprement l'exécution de la page en cas d'erreur.
  - ➔ Il ne s'utilise pas avec PDO.

Afficher le contenu d'une table -1 : query(), fetch(), print_r (exemple-2-select)

Exemple

```php
<?php
    // 1 : on écrit la requête
    $reqSQL='SELECT * FROM utilisateurs';

    // 2 : on récupère le résultat
    $reqPHP=$bdd->query($reqSQL);
    echo '<pre>'; print_r($reqPHP); echo '</pre>';

    // 3 : on affiche le résultat ligne par ligne
    while($ligne=$reqPHP->fetch()){
        echo '<pre>'; print_r($ligne); echo '</pre>';
        echo '<p> nombre d'éléments de $ligne : ' .sizeof($ligne). '</p>';
    }

    // 4 : on libère les tables de la requête
    $reqPHP->closeCursor(); // pour finir le traitement
?>
```

Explications

7. On écrit la requête SQL dans $reqSQL : c'est une simple chaine de caractères.
8. On utilise la méthode (fonction) query de $bdd (->) en passant la $reqSQL en paramètre.
   - ➔ Le résultat est dans $reqPHP : c'est un objet complexe sur lequel on peut appliquer des méthodes (des fonctions) qui donnera accès au tableau de données du select.
   - ➔ On affiche la $reqPHP pour voir son contenu : seule la requête s'affiche (la requête n'est pas exécutée à ce stade).
9. $reqPHP est « fétché » : on récupère les lignes de la réponse à la requête une par une. Quand il n'y en a plus ça retourne false.
   - ➔ Chaque $ligne retournée par le fetch est un tableau associatif : on peut faire un print_r de $ligne.
   - ➔ On constate alors qu'on a deux accès possibles à chaque donnée : par le nom du champs (id par exemple) ou par un numéro (0 pour id) : il y a donc 2 fois plus d'éléments que prévu dans chaque ligne
10. Quand on a fini de travailler, on fait un closeCursor, pour libérer les tables de la requête

Résultats

```
Connexion réussie

PDOStatement Object ( [queryString] => SELECT * FROM utilisateurs
)
DEBUT

Array
(
    [id] => 1
    [0] => 1
    [prenomNom] => Sia PEI
    [1] => Sia PEI
    [adMail] => ji@gmail.com
    [2] => ji@gmail.com
    [motDePasse] => jipei
    [3] => jipei
    [annee] => 1995
    [4] => 1995
)
nombre d'éléments de $ligne : 10

Array
(
    [id] => 2
    [0] => 2
    Etc.
```

Version MVC avec fetchAll()

Controleur

Le controleur include le fichier de connexion puis se connecte à la BD.
Il include aussi le « modèle_utilisateur » : les fonctions pour utiliser les utilisateurs.

Ensuite, il fait le travail du contrôleur : travail principal : charger les utilisateurs.

Pour finir, il include la vue qui affichera les utilisateurs.

```php
<?php
// Modèle
include("connexion.php");
include("modele_utilisateur.php");
$bdd = connexionBD('BD_Utilisateurs');

// Controleur
$utilisateurs = select_utilisateurs($bdd);

// Vue
include("vue_ex2_select.php");
?>
```

Modèle : avec fetchAll()

La fonction fetchAll permet de récupérer un tableau de tous les utilisateurs : c'est un tableau numéroté de tableaux associatifs.

```php
<?php
function select_utilisateurs($bdd)
{
  // 0 : on écrit la requête SQL
  $reqSQL = 'SELECT * FROM utilisateurs';

  // 1 : on fabrique la requête PHP
  $reqPHP = $bdd->query($reqSQL);
  print_r($reqPHP);

  // 2 : on récupère les résultats
  $lignes = $reqPHP->fetchAll();

  // 3 : on libère les tables de la requête
  $reqPHP->closeCursor(); // pour finir le traitement

  // 4 : on return le résultat
  return $lignes;
```

```
}
```

Vue

```
<!DOCTYPE html>
<html lang="fr">

<head>
  <meta charset="utf-8">
  <title>Test de BD</title>
</head>

<body>
  <h1> DEBUT </h1>
  <?php
  foreach ($utilisateurs as $utilisateur) {
    echo '<pre>';
    print_r($utilisateur);
    echo '</pre>';
    echo '<p> nombre d'éléments de $utilisateur : ' . sizeof($utilisateur) . '</p>';
  }
  ?>
  <h1> FIN </h1>
</body>

</html>
```

Exercices

Exercice-1 : chargez les exemples 1 et 2 et testez-les

- Il faut parfois charger les BD pour pouvoir tester les exemples.

Exercice-2 :

- Dans le code MVC, afficher tout le tableau $utilisateurs avec un print_r ou un var_dump. Mettez-le dans une balise <pre> pour que ce soit plus lisible. Faites ça dans le controleur.
- Dans le code MVC, affichez les utilisateurs proprement dans une table HTML. Pour accéder aux attributs on écrit : $ligne[‘id’].

2. Technique de programmation – PDO et PDOStatement - $bdd, $reqSQL, $reqPHP

- Pour manipuler la BD, on utilise principalement 2 classes correspondant à 2 objets :

  ➔ classe **PDO** -> objet **$bdd** (ou objet $pdo)

  ➔ classe **PDOStatemenet** -> objet **$reqPHP** (ou objet $pdoStatement)

Terminologie : $bdd - $reqSQL - $reqPHP - $ligne

- **$bdd :** un objet de la classe PDO sera appelé $bdd (ou $pdo)
  - C'est l'objet qui permet l'accès concret à la base de données, pour un utilisateur et une base de donnée.
- **$reqSQL :** le texte de la requête sera mis dans un $reqSQL.
  - C'est une simple chaine de caractères. Il ne doit pas être confondu avec le résultat de la requête : $requete.
- **$reqPHP :** un objet de la classe PSOStatement sera appelé $reqPHP (ou $pdoStatement : statement peut vouloir dire « requête ».
  - $reqPHP est un objet complexe qui contient à la fois le $reqSQL et permet l'accès au résultat de la requête une fois celle-ci exécutée.
- **$ligne :** le résultat d'un fetch() est une ligne : $ligne =$requete->fetch().
  - C'est un tuple de la table résultant de la requête SQL.
- **$lignes :** le résultat d'un fetchAll() c'est une liste de lignes : $lignes =$requete->fetchAll().
  - Ce sont tous les tuple de la table résultant de la requête SQL.

$bdd (PDO) – query - prepare

- http://php.net/manual/fr/class.pdo.php
- La classe PDO ne contient que des méthodes (elle ne contient pas d'attributs). Notons Particulièrement :
  - ➔ **query($reqSQL)** : renvoie un **$reqPHP** auquel est associé le $reqSQL passé en paramètre et donne accès au résultat de la requête (le résultat du Select) prêt à être fetché (prêt à être parcouru).
  - ➔ **prepare($reqSQL)** : renvoie un **$reqPHP** auquel est associé le $reqSQL passé en paramètre. La requête n'a pas été exécutée. Elle peut contenir des variables.
- PDO contient aussi des méthodes propres à une BD comme la gestion des transactions : commit, rollback, etc., et d'autres choses.

$requete (PDOStatement) - execute - fetch - closeCurseur

- http://php.net/manual/fr/class.pdostatement.php
- La classe PDOStatement contient un attribut : la valeur du $resSQL fourni en paramètre quand il a été créé. Il contient aussi des méthodes. Notons particulièrement :
  - ➔ **execute()** : permet d'exécuter une une requête avec des variables. Il faut fournir en paramètre un tableau de valeurs pour les variables.
  - ➔ **fetch()** : permet de récupérer les lignes du résultat de la requête, une par une.
  - ➔ **fetchAll()** : permet de récupérer les lignes du résultats de la requête, toutes dans un tableau.
  - ➔ **closeCurseur()** : permet de refaire un execute.

Synthèse

$bdd : PDO	$reqPHP : PDOStatement
-> query($reqSQL) : PDOStatement	-> fetch() : ligne
-> prepare($reqSQL) : PDOStatement	-> exectute() : bool
	-> closeCursor() : bool
	-> fetchAll() : toutes les lignes

Déboguer : or die bdd->errorInfo() : INUTILE en PDO

Présentation

- On a déjà vu les 2 techniques pour déboguer :
  - ➔ le paramètre $erreur dans le new PDO()
  - ➔ le « or die() » : à éviter avec les PDO.

Créer le $bdd avec la gestion des erreurs

- Pour afficher les détails d'une erreur, on crée un $bdd avec la gestion des erreurs :

```
$erreur = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
$bdd = new PDO($dsn, $username, $password, $erreur);
```

- C'est suffisant pour obtenir des messages d'erreur propres.

Aternative : exécuter la requête (query ou execute) « or die »

- Si on n'utilise pas un $erreur dans le new PDO, on peut ajouter un « or die » au query ou au execute pour afficher les détails d'une erreur.

```
$requete=$bdd->query($reqSQL) or die(print-r($bdd-
>errorInfo())) ;
```

ou

```
$requete->execute(array( … )) or die(print-r($bdd-
>errorInfo())) ;
```

  - ➔ Le « or die » est inutile avec une connexion PDO en ERRMODE.
  - ➔ http://php.net/manual/fr/pdo.errorinfo.php

Les limites du « or die » : confidentialité et user friendly

- Le « or die » peut être pratique en phase de développement.
- En production, il met au jour des informations qui peuvent être confidentielles et n'est pas user-friendly.
  - Il y a donc un problème de SECURITE !!!
- http://www.alsacreations.com/tuto/lire/676-gestion-erreurs-mysql-php-or-die.html

Conclusion

- Le « or die » c'est du PHP « à la papa » : PHP 1.0 ! On oublie !

3. Premiers Select

Accès aux champs, order by, like, limit (exemple-3-Select Where)

Accéder aux champs

- $ligne est un tableau associatif. On peut donc accéder à chacune de ses clés et on peut écrire (code PHP séparé du HTML) :

```
<p><strong> prenom  nom </strong></p>
<?php while($ligne=$requete->fetch()){ ?>
    <p>
        <?php echo $ligne[prenom]. ' ' . $ligne[nom] ; ?>
    </p>
<?php } ?>
```

Where et Order by

```
$reqSQL='
   Select realistateur, titre, annee from films
   where realisateur = \'King Vidor\'
   order by annee
';
// on precise 3 champs : realisateur, titre et annee
// attention au \'
// on trie par annee
```

like

```
$reqSQL='
   SELECT * FROM films
   WHERE realisateur like \'%manki%\'
   order by realisateur, annee
';
// like % manki % : n'importe quoi autour de manki
// order by realisateur, annee : plusieurs realisateurs possibles
// dans le resultat : j'ordonne le résultat
```

limit

```
$reqSQL='
          Select realistateur, titre, annee from films
          where annee = 1960
          order by annee
          limit 0, 10
') ;
// on prend les 10 premiers (de 1 à 10)
// limit 10, 10 pour les 10 suivants
// limite 20, 10 pour les 10 suivants, etc.
```

Requêtes variables : where realisateur = ? (exemple-4-Select Variable)

Objectif

- Mettre une variable dans une requête
  ➔ par exemple, une information saisie par l'utilisateur dans un formulaire).

Solution basique à éviter : risque XSS

- On pourrait mettre un $_GET dans la $reqSQL :

```
$reqSQL='Select… where auteur=\'' .$_GET['realisateur']. '\'') ;
```

  ➔ A éviter !!! si le $_GET contient 'toto \' or \'a\'=\'a, le select renverra toute la table !

Solution avec requête préparée : where ? , prepare et execute

- On met un « ? » pour la valeur variable de la requête.
- Ensuite on utilise une methode prepare() pour préparer la $reqPHP
- Ensuite on utilise une methode execute() sur le $reqPHP en lui passant en paramètres la ou les valeurs pour prendre la place du ou des « ? »

```
$reqSQL='Select… where realisateur = ?;

$requete=$bdd->prepare($reqSQL)

$requete->execute(array(
    $_GET['realisateur']
));
```

➔ On sépare les arguments par des « , » dans le array du execute

Remplacer les ? par des :alias

- On peut remplacer les « ? » par des alias de la forme « :nomAlias ».
- Du coup, dans le exécute on passe des couples clé-valeur : la clé correspond à l'alias, dans le « : ».
- Cette solution est la plus lisible et celle qu'on va privilégier.

```
$reqSQL='Select… where realisateur= :realisateur;

$requete=$bdd->prepare($reqSQL)

$requete->execute(array(
    'realisateur'=> $_GET['realisateur']
));
```

Exercices : c'est du MVC : comprenez bien l'architecture

Exercice-3 : chargez l'exemples 3 et testez-le

- Il faut parfois charger la BD pour pouvoir tester les exemples.
- Dans le 3, il y a plusieurs contrôleurs : testez-les tous et comprenez comment ça marche.
- **C'est du MVC : comprenez bien l'architecture !**

Exercice-4 : chargez l'exemples 4 et testez-le

- Il y plusieurs contrôleurs. :il faut tous les tester. Les contrôleurs appellent un autre contrôleur : vous devez bien comprendre l'architecture et le code.
- L'exemple 4 teste des problèmes de sécurité : vérifiez que ça marche bien en faisant ce qui est dit à l'affichage des contrôleurs.
- **C'est du MVC : comprenez bien l'architecture !**

4. DML

Ajouter, modifier, supprimer des données dans une table (exemple-5-insert-update-delete)

Select via phpMyAdmin – Afficher

- On peut utiliser l'interface graphique.
- On peut aussi entrer une commande SQL.

DML via phpMyAdmin – SQL

- On peut entrer les commandes SQL : INSERT, UPDATE et DELETE.
- Le système propose un pré-remplissage des commandes.

DML en PHP : INSERT

```
$reqSQL='
    INSERT INTO films
    (titre, realisateur, annee) VALUES
    (:titre, :realisateur, :annee)
';

//exemple('Dead Man','Jim JarmushKing Vidor','1995');

$requete=$bdd->prepare($reqSQL)

$resultat=$req-> execute(array(
    'titre'=>$_GET['titre'],
    'realisateur'=>$_GET['realisateur'],
    'annee'=>$_GET['annee']
)); // or die(print-r($bdd->errorInfo())) ;

/* le or die est inutile avec la connexion en ERRMODE */
/* $resultat pour traiter les erreurs proprement, sans ERRMODE */
```

DML en PHP : DELETE

```
$reqSQL='
    DELETE FROM films
    WHERE titre = :titre AND realisateur = :realisateur'
;

$requete=$bdd->prepare($reqSQL);
$resultat=$requete->execute(array(
    'titre'=>$_GET['titre'],
    'realisateur'=>$_GET['realisateur']
));

/* pour tester le résultat : 0 si pas de DELETE */
if($requete->rowCount() ){ // rowCount compte le nombre de delete
    echo '<br/>DELETE effectué ' .$requete->rowCount(). ' fois';
}
else {
    echo '<br/> Le DELETE a échoué';
}
```

- rowCount permet de savoir combien de delete on été effectués.
- http://www.astuces-webmaster.ch/page/mysql-pdo

ATTENTION au DELETE !!

- Attention au delete : quand une donnée est supprimée, on ne peut pas la récupérer si on est en mode validation (autocommit) ce qui est le plus fréquent !

- Il faut donc faire des vérifications, par exemple :

```
if (
   !isset($_GET['realisateur']) or
   !isset($_GET['titre']) or
   $_GET['realisateur']=='' or
   $_GET['titre']==''
){
   echo '<br/> Vous n\'avez pas saisi tous les paramètres';
}
```

DML en PHP : UPDATE

```
$reqSQL='
          UPDATE films
          SET duree=:duree
          WHERE titre = :titre AND realisateur = :realisateur
';
```

ATTENTION à l'UPDATE!!

- Attention à l'UPDATE : quand une donnée est modifiée, on ne peut pas la récupérer si on est en mode validation (autocommit) ce qui est le plus fréquent !

Bons usages

- A la place de :

```
'titre'=>$_GET['titre']
```

- on aura

```
'titre'=>$titre
```

  ➔ Les variables $titre, $realisateur, etc. seront récupérées via un $_POST ou un $_GET.

Exercices

Exercice-5 : chargez l'exemples 5 et testez-le

- Il faut parfois charger la BD pour pouvoir tester les exemples.
  - ➔ Pour charger une BD, on peut faire un copier-coller directement dans un client mysql ou dans la fenêtre SQL de phpMyAdmin
  - ➔ Ou alors, dans phpMyAdmin, on peut aussi faire une importation de la création des tables et des tuples dans une BD existant déjà
    - ➔ A noter qu'il faut peut-être laisser 2 lignes vides au début du fichier.
  - ➔ Ou alors, on peut aussi démarrer un client mysql en lui fournissant un fichier :

```
C:>mysql -uroot -p <maBD.sql
```

    - ➔ A noter qu'on peut avoir des difficultés avec les accents
- Dans les exemples, quand on fait du DML, vérifier le résultat dans MySQL : soit dans phpMyAdmin, soit dans un client MySQL.
- C'est du MVC : comprenez bien l'architecture !

5. CRUD

CRUD = Create – Read – Update - Delete

Principes

- Une application WEB tourne autour de 4 actions de bases dans la BD :
  - **C** : Create : c'est à la fois le CREATE de la BD et des tables et l'INSERT dans les tables.
  - **R** : Read : c'est un SELECT : on va chercher des données dans la BD.
  - **U** : Create : c'est un UPDATE : on modifie des données dans la BD.
  - **D :** Delete : c'est un DELETE : on supprime des données dans la BD.

Concevoir une application WEB c'est :

- Concevoir la BD : faire un modèle type MCD, MLD ou MPD.
- Pour chaque table, se demander : quel CRUD sur la table : quel INSERT, SELECT, UPDATE, DELETE

  ➔ C'est ce qu'on fera avec l'application Hackathon.

Remarque CRUD sur ce cours :

- Dans ce cours, il y a 3 parties :
  - La connexion à la BD : du coup, il faut que la BD existe et il y a le CREATE de la BD et des tables : une partie du C du CRUD
  - Le R du CRUD : les techniques pour faire des SELECT
  - Les CUD du CRUD : les techniques pour faire du DML : INSERT, UPDATE et DELETE

PHP – MYSQL – V2 : MYSQLI (A EVITER)

0 : présentation de l'usage de mysqli

Pour utiliser la bibliothèque mysqli, il n'y a rien à faire : elle est accessible « nativement » en PHP.

Pour pouvoir utiliser une BD avec mysqli, on suit 4 étapes :

1. Gestion des erreurs
2. Connexion à la BD
3. Envoi de la requête
4. Fermeture de la connexion

Documentation mysqli :

- **php.net :** https://www.php.net/manual/fr/book.mysqli.php
- **w3schools :** https://www.w3schools.com/php/php_ref_mysqli.asp

1 : la gestion des erreurs avec mysqli

Le code ci-dessous permet d'avoir des messages d'erreur explicites.

```
    // gérer les messages d'erreurs
    mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
```

2 : la connexion à la BD avec mysqli

Le code ci-dessous permet de se connecter à la BD.

On précise la machine sur laquelle se trouve le serveur de BD (pour nous : localhost), l'utilisateur qui se connecte (root), le mot de passe (rien sur PC, root sur mac), la database à laquelle on va accéder.

On instancie un nouvel objet qu'on appelle en général $mysql avec ces 4 paramètres.

```
   // connexion à la BD
    $localhost = "localhost";
    $user = "root";
    $password = "";
    $database = "DB_movies_anthropocene";
    $mysqli = new mysqli($localhost, $user, $password, $database);
```

3-1 : Envoi d'une requête avec mysqli – version de base

La méthode « query » permet d'envoyer une requête SQL.
Le résultat sera exploitable avec une boucle foreach.

```
    $reqSQL = "
       SELECT *
       FROM movies
       ORDER BY annee DESC
    ";

    $films = $mysqli->query($reqSQL);

    foreach ($films as $film) {
       echo $film["id"] . " - ";
       echo $film["titre"] . " - ";
       echo $film["titre"] . " - ";
       echo $film["titre"] . "<br>";
    }
```

3-2 : Envoi d'une requête avec mysqli – version « préparé »

Pour passer des variables en paramètres à une requête, on utilise la syntaxe des requêtes « préparées » : ça sécurise le code en évitant les injections SQL.

Exemple avec un fonction delete et un $id en paramètre : on ne retourne rien.

```
function deleteFilm($id){
    $sql = "
       DELETE FROM  movies
       WHERE id = ?
    ";

    $stmt = $mysqli->prepare($sql); // stmt : statement
    $stmt->bind_param("i", $id);    // "i" pour un integer
                                    // "is" pour integer et string
                                    // "iii" pour 3 integer
    $stmt->execute();
}
```

4 : Fermeture de la connexion

Pour alléger le serveur, on peut de déconnecter :

```
$mysqli->close();
```