Web - TP n^o 6
G. Subrenat, R. Zrour

Licence Informatique L3 - 2020-2021
UFR SFA - Université de Poitiers

Survol de PHP

Table des matières

1 Résumé

Dans ce TP nous verrons comment utiliser une base de données via la bibliothèque *PDO*.

Pour chaque exercice ou question d'exercice, faites une page web à part pour être sûr que vos divers codes n'interfèrent pas.

Le site http://php.net contient, entre autre, la description des fonctions de PHP ; vous aurez besoin de vous y référer tout au long du TP.

2 Manipulation de phpMyAdmin

Point de cours : http

phpMyAdmin est une interface web pour manipuler une base de données sur un serveur distant (ou non d'ailleurs).
L'interface est assez complète :

- création de tables
- manipulation des lignes (CRUD)
- exportation/importation
- administration
- ...

Il existe d'autres interfaces :

- *adminer* semble appréciée
- le mode console si les requêtes SQL ne vous rebutent pas (il faut pouvoir se connecter en *ssh* sur le serveur)

Les IDE (comme PhpStorm) proposent également un module pour se connecter à une base de données distante.
Il peut y avoir un problème technique cependant avec les IDE : en règle générale une IDE tourne sur le poste du développeur qui est différent du serveur de bases de données. Or il est fréquent qu'un serveur de bases de données interdise les connexions extérieures. Il faut alors voir avec l'administrateur du serveur.

Seule la question a) est au programme de ce TP ; mais vous êtes encouragés à faire l'intégralité de l'exercice par ailleurs.
Note : avec les nouvelles versions de *PhpMyAdmin*, il sera peut-être nécessaire d'adapter les indications des énoncés.

2.1 Manipulation de la base de données

Une base de données est pré-créée sur *tpweb*. Les identifiants d'accès (nom de la base, login et mot de passe) sont indiqués dans le fichier *.my.cnf* se situant dans votre répertoire d'accueil sur *tpweb*.
Ne modifiez en aucun cas ce fichier ou le contenu du répertoire *.my.cnf.d*, ni votre mot de passe.
Je répète : **NE MODIFIEZ RIEN**. Comme ce n'est peut-être pas clair : **NE MODIFIEZ PAS**, **NE TOUCHEZ PAS** [1] au fichier *.my.cnf* ou au contenu du répertoire *.my.cnf.d*.

Vous pouvez récupérer les tables de la base sur Updago et l'importer via :
`http://phpmyadmin.projet.sfasp2mi.univ-poitiers.fr`
dans l'onglet *Importer*.

Donc la suite de l'exercice n'est pas au programme de cette séance.

1. ou alors avec les yeux, et encore.

2.2 Construction des tables

Vous choisirez systématiquement le moteur de stockage *InnoDB* pour gérer les clés étrangères. Pour l'interclassement (à préciser à la création d'une table), vous choisirez systématiquement *utf8_general_ci*. *PhpMyAdmin* est accessible à `http:///phpmyadmin.projet.sfasp2mi.univ-poitiers.fr`

Créez les tables énumérées ci-dessous.

Table *tp_villes* :

- *id* : clé primaire (int, not null, auto-incrémentée, primary key)
- *nom* : nom de la ville (varchar 80, not null)
- *code* : code postal (varchar 6, not null)

Table *tp_auteurs* :

- *id* : clé primaire (int, not null, auto-incrémentée, primary key)
- *nom* : nom de la personne (varchar 80, not null)
- ville_id : lieu de résidence (int, null)

Table *tp_articles* :

- *id* : clé primaire (int, not null, auto-incrémentée, primary key)
- *id_auteur* : identifiant de l'auteur (int, not null)
- *titre* : titre de l'article (varchar 80, not null)
- *contenu* : le texte de l'article (text, not null)
- *date* : date de parution (date, null)

2.3 Gestion des clés étrangères

Pour la table *auteurs*, il faut relier la clé étrangère *ville_id* à l'*id* de la table *villes*[2].

Pour cela il faut aller sur l'onglet *Structures* de la table *auteurs*.

Dans le tableau récapitulatif des champs de la table, cliquez sur "Index" pour le champ *ville_id*. Puis choisissez l'entrée "Vue relationnelle" sous le tableau :

- il n'est pas utile de remplir les données pour la "Relation interne",
- une ville ne peut être supprimée que si elle n'est référencée par aucun auteur (directive *no action*),
- si l'*id* d'une ville est modifié, les clés étrangères de la table *auteurs* doivent être mises automatiquement à jour (directive *cascade)*,
- profitez-en pour mettre la "Colonne descriptive" à "nom".

Faites de même avec la table *articles*, avec "titre" pour la "Colonne descriptive".

2.4 Alimentation

Mettez quelques enregistrements par table.

2.5 Test des clés étrangères

Essayez de créer un auteur qui référence une ville qui n'existe pas.
Essayez de supprimer une ville utilisée et vérifiez que SQL bloque automatiquement la destruction.
Modifiez l'*id* d'un auteur ayant écrit un article et vérifiez la propagation dans la table *articles*.

2. Le préfixe *tp_* sera désormais implicite lorsqu'une table sera nommée.

3 PHP et MySQL

Dans cet exercice, vous utiliserez les fonctions PHP relatives à PDO pour manipuler la base de données. Vous êtes en mode auto-apprentissage ; le site http://php.net propose une documentation agrémentée d'exemples.

3.1 Affichage des tables

Faites 3 pages distinctes (i.e. des fichiers PHP différents) permettant d'afficher le contenu de chaque table.

Faites en sorte que le code commun aux trois pages ne soit écrit qu'une seule fois : généralement un fichier dédié propose une fonction de connexion, voire une fonction pour encapsuler les requêtes.

3.2 Affichage dynamique

Faites une page qui affiche tous les habitants d'une ville dont le nom est saisi par l'utilisateur dans une liste déroulante. Le résultat doit s'afficher en dessous du formulaire qui a permis de choisir la ville.

3.3 Modification d'un enregistrement

L'utilisateur choisit un article via une liste des titres et toutes les données de l'enregistrement s'affichent en dessous pour que l'utilisateur puisse les modifier. Faites la page correspondante.

3.4 Ajout d'un auteur

Faites une page qui permet d'ajouter un auteur. La ville d'habitation doit être choisie dans une liste déroulante, avec possibilité de dire que l'auteur n'habite aucune ville.

3.5 Suppression d'un auteur

Faites une page qui permet de détruire un auteur.

Si l'auteur a écrit un article, la suppression ne doit pas être faite et un message d'avertissement apparaît.

Faites la vérification de deux manières :

— d'abord manuellement (en regardant dans les tables),

— puis en faisant une suppression sans vérification et laisser SQL gérer les contraintes d'intégrité (et donc refuser la suppression).

Point de cours : où vérifier la validité d'une requête

La question est la suivante : est-ce le SGBD ou le code PHP qui doit s'assurer de la validité d'une requête ?

Il est absolument nécessaire que le SGBD s'assure de la validité d'une requête via les clés étrangères et les contraintes d'intégrité :

- on est assuré que la base de données est toujours cohérente
- il est possible qu'un autre site web manipule la base de données et lui pourrait ne pas faire les vérifications.

Inconvénient de vérifier la requête dans le code PHP :

- un *auteur* peut être lié à un grand nombre de tables et la vérification peut être pénible
- si on modifie la structure de la base de données, il faut mettre le code à jour

Mais :

- la paresse n'est pas un argument recevable

Avantage de vérifier la requête dans le code PHP :
- philosophique : la démarche “je lance le travail et si plante c’est que je n’avais pas le droit” me choque intellectuellement
- les messages d’erreurs sont plus précis

Ceci dit, normalement un site correct ne propose à l’internaute que les auteurs pouvant être supprimés. Dans le cas contraire, l’internaute pourrait choisir une suppression et un message lui dirait qu’il n’a pas le droit : ce n’est pas cohérent.
De fait le travail de vérification est fait au moment où on construit la liste des auteurs pouvant être supprimés.

3.6 Authentification

Point de cours

Le principe d’authentification est relativement simple.

Lorsqu’un internaute s’authentifie avec un mot de passe correct, on positionne un booléen en variable de session. Généralement on en profite pour stocker d’autres informations (login, droits, ...).
Voici le pseudo-code à la réception du formulaire de connexion :

```php
if (/* login/passwd ok */)
{
    $_SESSION['auth'] = true;
    $_SESSION['user'] = $_POST['login'];
    $_SESSION['rights'] = /* cf. BD */;
}
```

Lorsque l’internaute se déconnecte, il suffit d’invalider la variable de session :

```php
$_SESSION['auth'] = false;
// ou
unset($_SESSION['auth']);
```

En début de chaque script PHP, on vérifie cette variable :

```php
if ((! isset($_SESSION['auth'])) || ($_SESSION['auth'] !== true))
    exit(1);
```

Il s’agit de permettre l’accès aux pages précédentes uniquement si l’internaute s’est authentifié.

Construisez une page de connexion saisissant le login et mot de passe de l’internaute.
Si la saisie est correcte :
- l’internaute reste sur la même page
- un menu se rajoute permettant d’aller aux différentes pages précédemment créées
- le formulaire d’identification n’apparaît plus

Si elle est incorrecte, un message d’erreur l’annonce.
Note : le login et le mot de passe seront testés en dur dans le code.

Si un internaute essaie d’aller sur une des pages de manipulation de la BD sans être authentifié (en tapant directement l’URL par exemple), il doit être redirigé sur la page de connexion.

Enfin, ajoutez sur chaque page un lien permettant de se déconnecter. Ce lien n’existera, sur la page de connexion, que si l’internaute est réellement authentifié.
Le lien de déconnexion mène sur une page PHP dédiée qui s’occupe de la déconnexion (dans notre cas mettre un booléean à *false*) ; cette page fait ensuite une redirection vers la page de connexion.