TD3 – Requêtes préparées et association de classes SQL JOIN

Ce TD3 est le prolongement du TD2 sur l'enregistrement des données dans une BDD en utilisant la classe `PDO` de PHP. Nous poursuivons par le concept très important de requêtes préparées. Puis nous coderons des associations entre plusieurs tables de la BDD.

Attention : Il est nécessaire d'avoir fini la section 2.1 du TD précédent, qui vous faisait coder votre première requête `SELECT * FROM voiture`, pour attaquer ce TD.

Nous vous invitons toujours à utiliser les différentes commandes `status/log/add/commit` de `git` pour savoir où vous en êtes et enregistrer vos modifications.

I. Les injections SQL

I.1 Exemple d'injection SQL

Imaginez un site Web qui, pour connecter un utilisateur, exécute la requête SQL suivante et accepte la connexion dès que la requête renvoie au moins une réponse.

```
SELECT uid FROM Users WHERE name = '$nom' AND password = '$motDePasse';
```

Un utilisateur malveillant pourrait taper les renseignements suivants :

- Utilisateur : `Dupont';--`
- Mot de passe : n'importe lequel

La requête devient :

```
SELECT uid FROM Users WHERE name = 'Dupont'; -- ' AND password = 'mdp';
```

ce qui est équivalent à

```
SELECT uid FROM Users WHERE name = 'Dupont';
```

L'attaquant peut alors se connecter sous l'utilisateur Dupont avec n'importe quel mot de passe. Cette attaque du site Web s'appelle une **injection SQL**.

Vous aurez un exercice à la fin du TD pour simuler une injection SQL.

Source : https://fr.wikipedia.org/wiki/Injection_SQL

2. Les requêtes préparées

Imaginez que nous ayons codé une fonction `getVoitureParImmat($immatriculation)` comme suit

```php
function getVoitureParImmat($immatriculation) {
    $sql = "SELECT * from voiture WHERE immatriculation='$immatriculation'";
    $pdoStatement = Model::getPdo()->query($sql);
    return $pdoStatement->fetch();
}
```

Cette fonction marche, mais pose un gros problème de sécurité ; elle est vulnérable aux **injections SQL** et un utilisateur pourrait faire comme dans l'exemple précédent pour exécuter le code SQL qu'il souhaite.

Pour empêcher les **injections SQL**, nous allons utiliser une fonctionnalité qui s'appelle les **requêtes préparées** et qui est fournie par PDO. Voici comment les requêtes préparées fonctionnent :

1. On met un *tag* `:nomTag` en lieu de la valeur à remplacer dans la requête SQL
2. On doit "préparer" la requête avec la commande `prepare($requeteSql)`
3. Puis utiliser un tableau pour associer des valeurs aux noms des tags des variables à remplacer :

   ```php
   $values = array("nomTag" => "une valeur"); // Sans deux points devant nomTag
   ```

4. Et exécuter la requête préparée avec `execute($values)`
5. On peut alors récupérer les résultats comme précédemment (e.g. avec `fetch()`)

Voici toutes ces étapes regroupées dans une fonction :

```php
function getVoitureParImmat(string $immatriculation) : Voiture {
    $sql = "SELECT * from voiture WHERE immatriculation=:immatriculationTag";
    // Préparation de la requête
    $pdoStatement = Model::getPdo()->prepare($sql);

    $values = array(
        "immatriculationTag" => $immatriculation,
        //nomdutag => valeur, ...
    );
    // On donne les valeurs et on exécute la requête
    $pdoStatement->execute($values);

    // On récupère les résultats comme précédemment
    // Note: fetch() renvoie false si pas de voiture correspondante
    $voiture = $pdoStatement->fetch();

    return static::construire($voiture);
}
```

Remarque : Il existe une autre solution pour associer une à une les valeurs aux variables d'une requête préparée avec la fonction `bindParam()` de la classe PDO (qui permet de donner le type de la valeur). Cependant, nous vous conseillons d'utiliser systématiquement la syntaxe avec un tableau `execute($values)`.

Exercice 1

1. Copiez/collez dans un nouveau dossier TD3 les fichiers `Conf.php`, `Model.php`, `Voiture.php` et `lireVoiture.php`.
2. Copiez la fonction précédente dans la classe `Voiture` en la déclarant publique et statique.
3. Testez la fonction `getVoitureParImmat` dans `lireVoiture.php`.
4. On souhaite que `getVoitureParImmat` renvoie `null` s'il n'existe pas de voiture d'immatriculation `$immatriculation`. Mettez à jour le code et la déclaration de type. Testez votre code.

Désormais, toutes les requêtes SQL doivent être codées en utilisant des requêtes préparées, sauf éventuellement des requêtes SQL sans variable comme `SELECT * FROM voiture`.

Exercice 2

1. Créez une fonction `public function sauvegarder() : void` dans la classe `Voiture` qui insère la voiture courante (`$this`) dans la BDD. On vous rappelle la syntaxe SQL d'une insertion :

   ```
   INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...)
   ```

   Attention : La requête `INSERT INTO` ne renvoie pas de résultats ; il ne faut donc pas faire de `fetch()` sous peine d'avoir une erreur `SQLSTATE[HY000]: General error`.
2. Testez cette fonction dans `lireVoiture.php` en créant un objet de classe `Voiture` et en l'enregistrant.

Exercice 3

Branchons maintenant notre enregistrement de voiture dans la BDD au formulaire de création de voiture du TD1 :

1. Copiez dans le dossier TD3 les fichiers `creerVoiture.php` et `formulaireVoiture.html` du TD1.
2. Modifier la page `creerVoiture.php` de sorte qu'elle sauvegarde l'objet `Voiture` reçu (en GET ou POST, au choix).
3. Testez l'insertion grâce au formulaire `formulaireVoiture.html`.

   Remarque : Vous aurez sans doute une erreur `Class "Model" not found`. Où inclure `Model.php` : dans `Voiture.php` ou dans `creerVoiture.php` ? Règle simple : chaque fichier doit inclure les classes dont il a besoin. Comme `Voiture.php` a besoin de la classe `Model` (à cause de l'instruction `Model::getPdo()`), c'est au début de `Voiture.php` qu'il faut faire `require_once "Model.php";`.
4. Vérifiez dans PhpMyAdmin que les voitures sont bien sauvegardées.
5. Essayez de rajouter une voiture dont un champ contient un guillemet simple `'`, par exemple une marque `"Roll's Royce"`. Est-ce qu'elle a bien été sauvegardée ? Si ce n'est pas le cas, c'est sûrement que vous n'avez pas utilisé les requêtes préparées.

3. Création des tables de notre site de covoiturage

Reprenons les classes du TD précédent sur le covoiturage afin d'y ajouter la gestion de la persistance.

Exercice 4

Si vous ne l'avez pas déjà fait, créez des tables `utilisateur` et `trajet` comme suit :

1. Dans votre PhpMyAdmin, créez une table `utilisateur` avec les champs suivants :
   - `login` : VARCHAR 32, clé primaire
   - `nom` : VARCHAR 32
   - `prenom` : VARCHAR 32

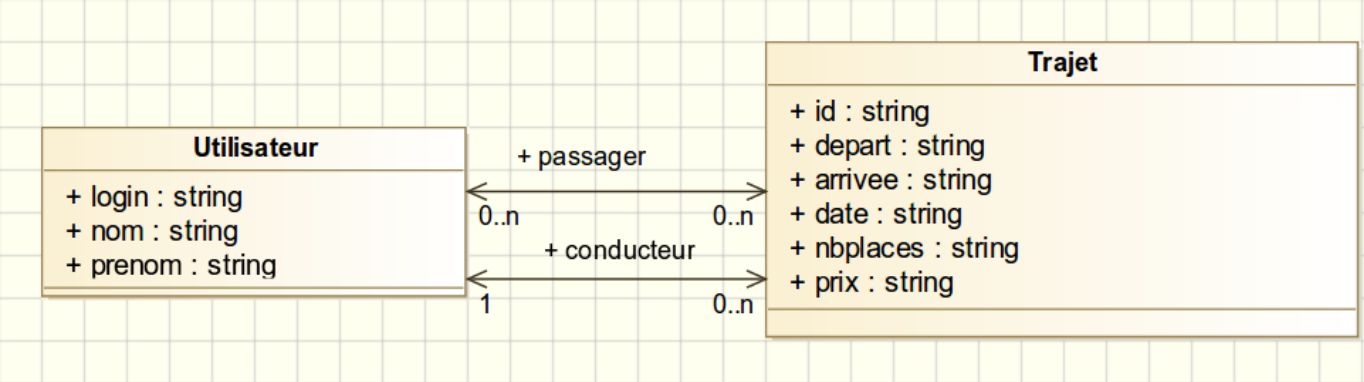
   Important : Pour faciliter la suite du TD, mettez à la création de toutes vos tables `InnoDB` comme moteur de stockage, et `utf8_general_ci` comme interclassement (c'est l'encodage des données, et donc des accents, caractères spéciaux...).
2. Insérez quelques utilisateurs.
3. Créez une table `trajet` avec les champs suivants :
   - `id` : INT, clé primaire, qui s'auto-incrémente (voir en dessous)
   - `depart` : VARCHAR 32
   - `arrivee` : VARCHAR 32
   - `date` : DATE
   - `nbPlaces` : INT
   - `prix` : INT
   - `conducteurLogin` : VARCHAR 32

   Note : On souhaite que le champ primaire `id` s'incrémente à chaque nouvelle insertion dans la table. Pour ce faire, cochez la case `A_I` (auto-increment) pour le champ `id`.

   Important : Avez-vous bien pensé à `InnoDB` et `utf8_general_ci` comme précédemment ?
4. Insérez quelques trajets en prenant soin de ne pas remplir la case `id` (pour que l'auto-incrément marche) et en mettant dans `conducteurLogin` un login d'utilisateur valide (pour éviter des problèmes par la suite).

4. Premier lien entre `utilisateur` et `trajet`

Vous avez couvert dans le cours *R2.01 – Développement orienté objets* les diagrammes de classes. Ce type de diagramme est utile pour penser la base de donnée d'une application Web. Voici le nôtre :



Question : Comment implémenteriez-vous l'association *conducteur* entre utilisateurs et trajets dans la BDD en tenant compte de sa multiplicité ?

Notre solution : Comme il n'y a qu'un conducteur par trajet, nous allons rajouter un champ `conducteurLogin` à la table `trajet`.

On souhaite que le champ `trajet.conducteurLogin` corresponde à tout moment à un login de conducteur `utilisateur.login`. Vous souvenez-vous quelle est la fonctionnalité des bases de données qui permet ceci ?

Réponse : Il faut utiliser des clés étrangères.

Exercice 5

Voici les étapes pour faire ce lien :

1. À l'aide de l'interface de PhpMyAdmin, faites de `trajet.conducteurLogin` un **index**.

   Aide : Dans l'onglet `Structure` de la table `trajet`, cliquez sur l'icône de l'action `index` en face du champ `conducteurLogin`.

   Plus de détails : Dire que le champ `conducteurLogin` est un **index** revient à dire à MySql que l'on veut trouver rapidement les lignes qui ont un `conducteurLogin` donné. Du coup, MySql va construire une structure de donnée pour permettre cette recherche rapide. Une **clé étrangère** est nécessairement un **index**, car on a besoin de ce genre de recherches pour tester rapidement la contrainte de clé étrangère.

2. Rajoutez la contrainte de **clé étrangère** entre `trajet.conducteurLogin` et `utilisateur.login`. Pour ceci, allez dans l'onglet `Structure` de la table `trajet` et cliquez sur `Gestion des relations` pour accéder à la gestion des clés étrangères (ou `Vue relationnelle` pour les PHPMyAdmin plus récents).

   Nous allons utiliser le comportement `ON DELETE CASCADE` pour qu'une association soit supprimé si la clé étrangère est supprimée, et le comportement `ON UPDATE CASCADE` pour qu'une association soit mise à jour si la clé étrangère est mise à jour.

   Attention : Pour supporter les clés étrangères, il faut que le moteur de stockage de toutes vos tables impliqués soit `InnoDB`. Vous pouvez choisir ce paramètre à la création de la table ou le changer après coup dans l'onglet `Opérations`.

5. Association entre utilisateurs et trajets

5.1 Dans la base de donnée

Question : Comment implémenteriez-vous l'association *passager* entre utilisateurs et trajets dans la BDD en tenant compte de ses multiplicités ?

Réponse : Comme la relation *passager* est non bornée (on ne limite pas le nombre d'utilisateurs d'un trajet et inversement), on utilise une table de jointure.

Nous choisissons donc de créer une table `passager` qui contiendra deux champs :

- l'identifiant INT `trajetId` d'un trajet et
- l'identifiant VARCHAR(32) `utilisateurLogin` d'un utilisateur.

Pour inscrire un utilisateur à un trajet, il suffit d'écrire la ligne correspondante dans la table `passager` avec leur `utilisateurLogin` et leur `trajetId`.

Question : Quelle est la clé primaire de la table `passager` ?

Réponse : Le couple (trajetId,utilisateurLogin). Si vous choisissez trajetId seul comme clé primaire, un trajet aura au plus un passager, et si vous choisissez utilisateurLogin, chaque utilisateur ne pourra être passager que sur un unique trajet.

Exercice 6

1. Créer la table `passager` en utilisant l'interface de PhpMyAdmin.

   Important : Avez-vous bien pensé à `InnoDB` et `utf8_general_ci` comme précédemment ?

2. Assurez-vous que vous avez bien le bon couple en tant que clé primaire. Cela se voit dans `Gestion des relations`.

3. Rajoutez la contrainte de **clé étrangère** entre `passager.trajetId` et `trajet.id`, puis entre `passager.utilisateurLogin` et `utilisateur.login`. Utiliser encore les comportements `ON DELETE CASCADE` et `ON UPDATE CASCADE` pour qu'une association soit mise à jour si la clé étrangère est mise à jour.

4. À l'aide de l'interface de PhpMyAdmin, insérer quelques associations pour que la table `passager` ne soit pas vide.

5. Vous allez maintenant vous assurer de la bonne gestion des clés étrangères en testant le comportement `ON DELETE CASCADE`. Pour cela :

   i. créez un trajet correspondant à un certain conducteur,
   ii. puis inscrivez des passagers pour ce trajet
   iii. supprimez ensuite le conducteur en question de la table `utilisateur` et vérifiez que les lignes de la table `passager` précédemment insérées ont bien été supprimées elles aussi.

5.2 Au niveau du PHP

i. Liste des utilisateurs d'un trajet et inversement

Nous allons maintenant pouvoir compléter le code PHP de notre site pour gérer l'association. Commençons par rajouter des fonctions à nos classes `Utilisateur` et `Trajet`.

Note : Si vos fichier `Utilisateur.php` et `Trajet.php` ne sont pas complets, vous pouvez repartir des fichiers suivants : `Utilisateur.php`, `Trajet.php`.

Avant toute chose, vous souvenez-vous comment faire une jointure en SQL ? Si vous n'êtes pas tout à fait au point sur les différents `JOIN` de SQL, vous pouvez vous rafraîchir la mémoire en lisant https://www.w3schools.com/sql/sql_join.asp.

Exercice 7

1. Créer une `public static function getPassagers($id)` dans `Trajet.php` qui prendra en entrée un identifiant de trajet. Cette fonction devra retourner un tableau d'objets de classe `Utilisateur` correspondant aux utilisateurs inscrits au trajet d'identifiant `$id` en faisant la requête adéquate.

   Indices :

   - Utiliser une requête à base d'`INNER JOIN`. Une bonne stratégie pour développer la bonne requête est d'essayer des requêtes dans l'onglet SQL de PhpMyAdmin jusqu'à tenir la bonne.
   - Il faut peut-être mettre à jour la classe `Utilisateur` pour qu'elle ait les mêmes attributs que la table `utilisateur` de la BDD. Il faut aussi mettre à jour le constructeur comme on l'a fait pour `Voiture`.
   - Comme vous demandez à `fetch` de créer des objets de la classe `Utilisateur`, il faut inclure le fichier de classe. De manière générale, la bonne pratique est que chaque fichier PHP inclus les fichiers dont il a besoin. C'est plus sûr que de compter sur les autres fichiers. Et le `once` du `require_once` vous mets à l'abri d'une inclusion multiple du même fichier de déclaration de classe.

2. Testons votre fonction. Créez une page `testGetPassagers.php` qui

   i. charge les classes nécessaires,
   ii. appelle la fonction `getPassagers($id)` avec un identifiant de trajet existant,
   iii. affiche les utilisateurs renvoyés grâce à la fonction `afficher`.

3. Créez un formulaire `formGetPassagers.php` de méthode `GET` avec un champ texte où l'on rentrera l'identifiant d'un trajet. La page de traitement de ce formulaire sera `testGetPassagers.php`. Modifiez `testGetPassagers.php` pour qu'il récupère l'identifiant envoyé par le formulaire.

Avez-vous bien utilisé une requête préparée dans `getPassagers` ?

6. Créez une injection SQL

Si vous êtes en avance sur les TDs, nous vous proposons de créer un exemple d'injection SQL. Mettons en place notre attaque SQL :

1. Pour ne pas supprimer une table importante, créons une table `voiture2` qui ne craint rien :
   - allez dans PHPMyAdmin et cliquez sur votre base de donnée (celle dont le nom est votre login à l'IUT)
   - Dans l'onglet SQL `Importer`, donnez le fichier `voiture2.sql` qui créera une table `voiture2` avec quelques voitures.

2. Nous vous fournissons le fichier PHP que nous allons attaquer : `formGetImmatSQL.php`
   Ce fichier contient un formulaire qui affiche les informations d'une voiture étant donné son immatriculation.
   Testez ce fichier en donnant une immatriculation existante.
   Lisez le code pour être sûr de bien comprendre le fonctionnement de cette page (et demandez au professeur si vous ne comprenez pas tout !).

3. Le point clé de ce fichier est que la fonction `getVoitureParImmat` a été codée sans requête préparée et est vulnérable aux injections SQL.

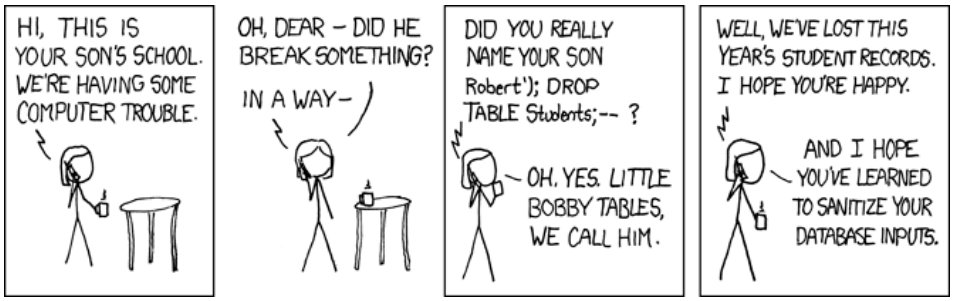
```php
function getVoitureParImmat(string $immat) : ?Voiture {
   $sql = "SELECT * from voiture2 WHERE immatriculation='$immat'";
   echo "<p>J'effectue la requête <pre>\"$sql\"</pre></p>";
   $pdoStatement = Model::getPDO()->query($sql);
   $voitureTableau = $pdoStatement->fetch();

   if ($voitureTableau !== false) {
         return Voiture::construire($voitureTableau);
   }
   return null;
}
```

   Trouvez ce qu'il faut taper dans le formulaire pour que `getVoitureParImmat` vide la table `voiture2` (SQL Truncate).

6.1 Deux cas concrets

Pour éviter les radars, il y a des petits malins.

Ou un petit XKCD



7. Et si le temps le permet…

Si vous êtes bien avancés sur les TDs, voici une liste d'idées pour compléter notre site.

7.1 Liste des trajets d'un utilisateur

De la même manière que dans l'exercice sur `getPassagers()`, utilisons une jointure SQL pour trouver tous les trajets d'un utilisateur.

Exercice 8

1. Créez une `public static function getTrajets($login)` dans `Utilisateur.php` qui prendra en entrée un login d'utilisateur `$login` et retourne les trajets auxquels il est inscrit en tant que passager.
2. Créez une page de test `testGetTrajets.php` et un formulaire `formGetTrajets.php`.

7.2 Désinscrire un utilisateur d'un trajet et inversement

Rajoutons une fonctionnalité : dans une future vue qui listera les trajets d'un utilisateur, nous voudrions avoir un lien 'Désinscrire' qui enlèvera l'utilisateur courant du trajet sélectionné.

Exercice 9

1. Créer une `public static function supprimePassager($trajetId, $utilisateurLogin)` dans `Trajet.php`. Cette fonction devra désinscrire l'utilisateur `utilisateurLogin` du trajet `trajetId`.
2. Créez une page de test `testSupprimePassager.php` et un formulaire `formSupprimePassager.php` de sorte que l'on puisse rentrer un identifiant de trajet et un login d'utilisateur dans le formulaire, et que l'envoi du formulaire redirige sur `testSupprimePassager.php` qui supprimera le passager dans la BDD.

7.3 Gestion des erreurs

Traitons plus systématiquement tous les cas particuliers. Pour l'instant, les méthodes suivantes sont correctement codés :

- `getVoitures()` de `Voiture.php` n'a pas de cas particulier,
- `getVoitureParImmat()` de `Voiture.php` gère une immatriculation inconnue en renvoyant la voiture `null`.

Par contre, vous allez améliorer les méthodes suivantes :

- `sauvegarder()` de `Voiture.php` ne traite pas :

    - le cas d'une voiture existant déjà en base de donnée (`SQLSTATE[23000]: Integrity constraint violation`)
    - le cas d'un problème de données :
        - chaîne de caractères trop longue (`SQLSTATE[22001]: String data, right truncation`)
        - entier trop grand (`SQLSTATE[22003]: Numeric value out of range`)
- `supprimePassager()` de `Trajet.php` ne traite pas le cas d'un passage inexistant.

Exercice 10

1. Pour la méthode `sauvegarder()`, les cas particuliers génèrent une exception de la classe `PDOException`. Modifiez la déclaration de type de la méthode pour qu'elle retourne un booléen pour indiquer si la sauvegarde s'est bien passée. Modifiez la méthode pour intercepter les `PDOException` avec un `try/catch` et retourner `false` en cas de problème.

2. Pour la méthode `supprimePassager()`, utilisez la méthode `rowCount()` de la classe `PDOStatement` pour vérifier que la requête de suppression a bien supprimé une ligne de la BDD.

Documentation :

- SQLSTATE error codes
- MySql error reference

7.4 (Optionnel) Quelques idées complémentaires

Voici une liste d'idées pour compléter notre site :

1. Notre liste des trajets d'un utilisateur est incomplète : il manque les trajets dont il est conducteur (et non passager). La page qui liste les trajets d'un utilisateur pourrait donner les deux listes comme conducteur et comme passager.
2. Similairement, nous avons oublié le conducteur de la liste des passagers d'un trajet. Le rajouter avec un statut à part.
3. Vous pouvez aussi éventuellement mettre en place des `trigger` dans votre SQL pour gérer le nombre de passagers par véhicule ...
4. Nous n'avons pas intégré les voitures aux utilisateurs, ni aux trajets dans notre schéma de bases de données.
   Que pourriez-vous faire ? À quelles relations pensez-vous entre ces tables ?
