TP4 – Application du pattern MVC – livres, adhérents et autres…

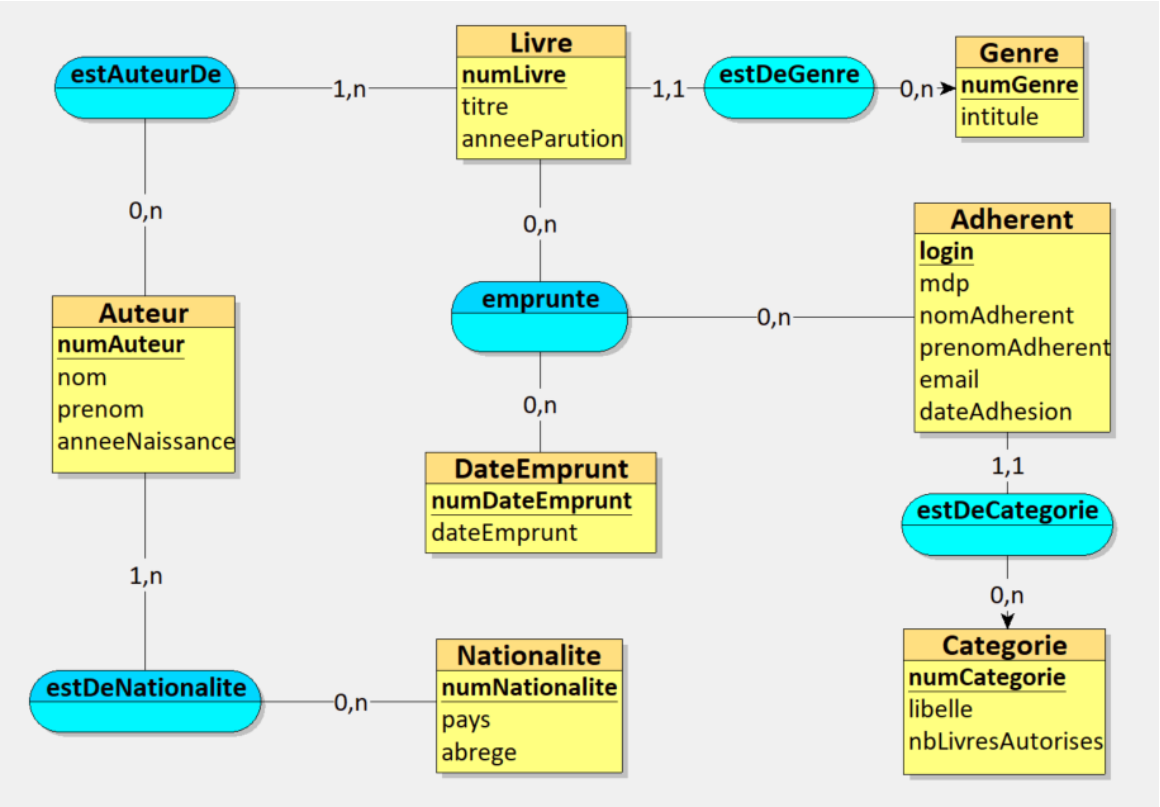
Nous allons dans la suite des TP mettre en place une application où les adhérents peuvent emprunter des livres. Les auteurs nous ont permis de découvrir le pattern MVC. Nous allons réinvestir cela en intégrant les livres et les adhérents.

Tout servira, puisqu'à terme un adhérent pourra, avant sa réservation, faire une recherche par auteur par exemple.

Pour le moment, dans ce TP4, nous allons appliquer ce qui a été fait pour les auteurs à toutes les autres entités.

Voici un Modèle Conceptuel de Données (MCD) de ce que l'on va prendre comme base de travail. Il est à étudier et à comprendre précisément pour pouvoir continuer la construction de notre site.

Vous accorderez notamment de l'attention à **l'association ternaire** « emprunte ». Un adhérent peut emprunter plusieurs fois le même livre à des dates différentes. Il faut donc inclure la date d'emprunt en clé primaire, d'où cette entité DateEmprunt.



Modèle Conceptuel de Données

Le Schéma Relationnel correspondant donne les tables :

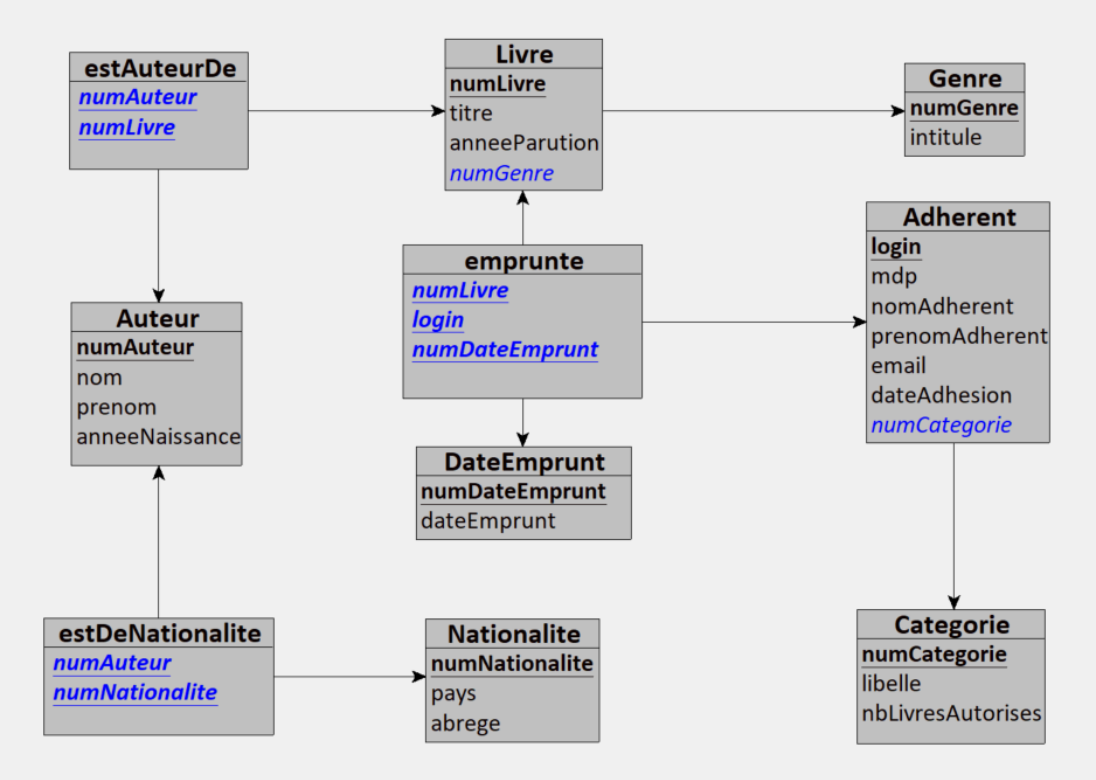


Schéma Relationnel

Auteur = (**numAuteur**, nom, prenom, anneeNaissance);

Genre = (**numGenre**, intitule);

Categorie = (**numCategorie**, libelle, nbLivresAutorises);

Nationalite = (**numNationalite**, pays, abrege);

DateEmprunt = (**numDateEmprunt**, dateEmprunt);

Livre = (**numLivre**, titre, anneeParution, *#numGenre*);

Adherent = (**login**, mdp, nomAdherent, prenomAdherent, email, dateAdhesion, *#numCategorie*);

estAuteurDe = (***#numAuteur, #numLivre***);

estDeNationalite = (***#numAuteur, #numNationalite***);

emprunte = (***#numLivre, #login, #numDateEmprunt***);

Les entités à définir sont : Auteur, Genre, Categorie, Nationalite, DateEmprunt, Livre et Adherent.

Les 3 autres tables représentent les associations complexes.

Un fichier bibliotheque.sql vous est fourni sur Moodle pour mettre en place la base sur phpMyadmin.

Vous disposez d'une architecture de base ex1.zip, basée sur la solution du TP3.

Exercice 0 – La base de données

1. Créez à partir du fichier `bibliotheque.sql` vos tables de données par l'interface phpMyadmin. Vérifiez que les insertions se sont bien déroulées.

2. Créez dans votre architecture le dossier `TP4/ex1` qui reprend le code de `ex1.zip`

Exercice 1 – Les fichiers Modèle

Sur le modèle du fichier auteur.php qui définit la classe Auteur (disponible sur Moodle), définissez dans le répertoire modele les fichiers genre.php, categorie.php, nationalite.php, dateEmprunt.php, livre.php, et adherent.php.

Ces fichiers auront leurs getter, setter, constructeur (adapté à la base de données, voir cours), ainsi qu'une méthode afficher() basique qui donne les éléments essentiels de l'instance fabriquée.

Ce travail est rébarbatif mais nécessaire pour comprendre l'importance de la généricité du travail (nous verrons plus tard ce que cela signifie et implique).

Exercice 2 – Le Contrôleur Adhérent et les vues correspondantes

Dupliquez `TP4/ex1` en `TP4/ex2`.

1. Créez le fichier contrôleur correspondant aux adhérents, sur le modèle du fichier controleurAuteur.php.

2. Créez les vues lesAdherents.php et unAdherent.php sur le modèle des vues lesAuteurs.php et unAuteur.php. Si vous trouvez que ces fichiers se ressemblent vraiment, et que vous vous répétez dans votre travail, c'est normal, nous allons remédier bientôt à ça !

3. Enrichissez le menu pour y ajouter un lien qui permettra d'afficher tous les adhérents. Vous devez remarquer que maintenant, il y a deux contrôleurs. Donc vous prendrez aussi le temps de modifier les items du menu pour y faire figurer le contrôleur.

```
<nav>
  <a href="routeur.php?controleur=ControleurAuteur&action=lireAuteurs">tous les auteurs</a>
  <a href="routeur.php?controleur=ControleurAdherent&action=lireAdherents">tous les adhérents</a>
</nav>
```

4. Modifiez le routeur pour qu'il puisse gérer le contrôleur passé dans l'url. Vérifiez que tout fonctionne !

Exercice 3 – Le Contrôleur Livre et les vues correspondantes

Dupliquez `TP4/ex2` en `TP4/ex3`.

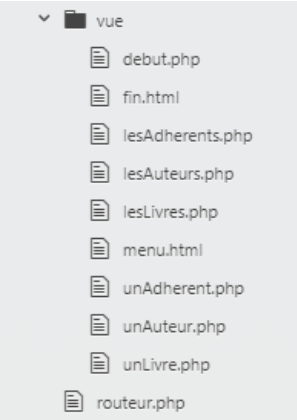
1. Créez le fichier contrôleur correspondant aux livres, sur le modèle des deux autres.
2. Créez les vues lesLivres.php et unLivre.php sur le modèle des vues précédentes.
3. Enrichissez le menu pour y ajouter un lien qui permettra d'afficher tous les livres.
4. Modifiez le routeur pour qu'l puisse gérer le contrôleur des livres. Vérifiez que tout fonctionne !

Exercice 4 – début de généricité avec les vues

Dupliquez `TP4/ex3` en `TP4/ex4`.

Si vous êtes observateur vous avez remarqué que vous avez fait trois fois le même travail ☹

Cela mérite réflexion, et c'est le moment de factoriser le code en introduisant de la généricité. Pour le moment, on va s'occuper des vues. Votre architecture doit ressembler à la capture ci-contre



1. Normalement, vos fichiers lesAdherents.php, lesLivres.php et lesAuteurs.php sont identiques ou quasi identiques. Créez un fichier lesObjets.php qui reprend le code, et supprimez les trois autres fichiers.
2. Partout où c'est nécessaire, modifiez les appels de vue pour passer à ce nouveau fichier. Vérifiez que tout fonctionne.
3. De même, les fichiers unLivre.php, unAdherent.php et unAuteur.php sont quasi identiques. En vous arrangeant pour donner le même nom à la variable à afficher, arrangez-vous pour que ces fichiers puissent se résumer à un seul fichier unObjet.php, supprimez les fichiers inutiles et changez partout où c'est nécessaire le code pour que tout fonctionne. Vérifiez.

Nous allons, dans les exercices qui suivent, nous lancer dans une factorisation du code modèle.

Si vous observez attentivement le code des fichiers modèle `auteur.php`, `livre.php` et adherent`.php`, il est clair que ces trois fichiers sont très similaires.

Comme cela se passe dans ce cas, nous allons factoriser le code commun en créant une classe mère, que nous appellerons `Objet`, dans un fichier `objet.php`. Les classes `Livre`, `Adherent` et `Auteur` étendront la classe `Objet`.

Exercice 5 – généricité du modèle – les getter et les setter

Dupliquez votre structure `TP4/ex4` en `TP4/ex5`.

1. Créez un fichier `TP4/ex5/modele/objet.php`, qui définit une classe `Objet`.

2. Donnez à cette classe le getter suivant :

```php
// getter générique
public function get($attribut) {
  return $this->$attribut;
}
```

   Ce getter utilisera une chaîne de caractères `$attribut` passée en paramètre et retournera l'attribut correspondant.

3. Sur le même modèle, complétez le code d'un setter générique dont la définition sera :

```php
// setter générique
public function set($attribut,$valeur)
```

4. Faites en sorte que les classes `Auteur`, `Adherent` et `Livre` héritent de `Objet`, puis supprimer les anciens getter et setter de ces classes.

5. Passez tous les attributs en visibilité `protected`, ce sera nécessaire dans cette nouvelle hiérarchie avec héritage.

6. Faites bien attention, dans les contrôleurs, que les appels aux anciens getter soient actualisés ! Par exemple, `getLogin()` doit maintenant devenir `get('login')`.

7. N'oubliez pas d'insérer la classe `Objet`, par exemple dans le routeur. Vérifiez que tout fonctionne !

Exercice 6 – Création d'un constructeur générique

Dupliquez tout en `TP4/ex6`.

Entre les trois constructeurs spécifiques, la structure est la même. Seul le nombre de paramètres change. Nous allons résoudre ce problème en passant au constructeur non pas un certain nombre de paramètres variable, mais un unique tableau de paramètres (et donc, en fait, un seul paramètre : le tableau).

```php
// constructeur générique
public function __construct($donnees = NULL) {
  if (!is_null($donnees)) {
    foreach($donnees as $attribut => $valeur) {
      $this->set($attribut,$valeur);
    }
  }
}
```

De toute façon, nous n'utilisons le constructeur que dans le cadre d'import des données de la base. Faites néanmoins ce changement en codant ce constructeur générique dans la classe `Objet`, puis supprimez les constructeurs devenus inutiles dans les trois classes `Livre`, `Auteur` et `Adherent`, et vérifiez que tout fonctionne.

Exercice 7 – Une méthode générique de sélection de tous les objets

Dupliquez tout en `TP4/ex7`.

Dans cet exercice, nous allons regrouper `getAllAuteurs`, `getAllAdherents` et `getAllLivres`, qui sont similaires, en une méthode générique, `getAll`, et qui sera codée dans la classe `Objet` pour remplacer les trois précédentes.

On pourrait passer en paramètre de cette méthode le nom de la table concernée, par exemple `getAll("Auteur")`.

On va voir une solution plus esthétique, que permet PHP, et qui évite d'utiliser ce genre de paramètre.

1. Définissez dans le fichier `auteur.php` un attribut de **classe** selon le schéma suivant, pour la classe `Auteur` :

```
class Auteur extends Objet {

  // attributs de classe
  protected static $objet = "Auteur";
```

Faites de même dans les fichiers `livre.php` et `adherent.php`.

2. Recopiez le code de la méthode `getAllAuteurs` de la classe `Auteur` du fichier `auteur.php` dans une méthode `public static getAllObjets()` de la classe `Objet` du fichier `objet.php`.

Le problème est que l'appel à cette méthode `getAllObjets` doit s'adapter à la classe qui l'utilise :

- Pour la classe `Auteur`, la requête sera `SELECT * FROM Auteur;`
- Pour la classe `Livre`, la requête sera `SELECT * FROM Livre;`
- Pour la classe `Adherent`, la requête sera `SELECT * FROM Adherent;`

Il faut donc récupérer le nom de la table, et c'est précisément le rôle des attributs static définis dans la question précédente. A partir de la classe mère, on peut y accéder de la façon suivante :

```
public static function getAllObjets() {
  $table = static::$objet;
```

Ainsi, en fonction de la classe fille qui appelle la méthode, la variable `$table` va adapter sa valeur.

Utilisez cette technique pour compléter la méthode `getAllObjets` de la classe `Objet`, puis supprimez les méthodes `getAllLivres`, `getAllAuteurs` et `getAllAdherents` des classes filles.

3. Transférez vos fichiers. S'il y a une erreur, c'est normal, vous avez oublié de corriger les contrôleurs… Corrigez ce qu'il faut, et vérifiez que tout fonctionne !

Exercice 8 – Une méthode générique de sélection d'un objet

Dupliquez votre code dans `TP4/ex8`. Nous allons cette fois créer une méthode générique dans la classe Objet pour remplacer les méthodes de sélection :

- d'un `Adherent` par son `login`,
- d'un `Auteur` par son `numAuteur`,
- d'un `Livre` par son `numLivre`.

Ici, va se poser le problème de la clé primaire utilisée. On pourrait la passer en paramètre, mais on va utiliser la technique de l'exercice précédent : les attributs `static` des classes filles, récupérables au niveau de la classe mère.

1. Définissez un attribut `protected static $cle` dans les trois classes `Auteur`, `Adherent` et `Livre`, et donnez à ces trois attributs la valeur adéquate.

2. Recopiez le code de la méthode `getAuteurByNum($numAuteur)` de la classe `Auteur` du fichier `auteur.php` dans une méthode `public static function getObjetById($id)` de la classe `Objet` du fichier `objet.php`.

   En vous inspirant de l'exercice précédent, mettez tout en place pour qu'on puisse supprimer les trois anciennes méthodes et ne plus utiliser que la nouvelle.
   Supprimez les anciennes méthodes et vérifiez, après transfert, que tout fonctionne ! Si vous avez des erreurs, c'est probablement que vous n'avez pensé à modifier le nom de la méthode de sélection d'un objet dans les contrôleurs…

   BILAN : Regardez à quoi ressemblent maintenant vos fichiers modèle :

   - Des attributs de classe spécifiques ;
   - Des attributs d'instance spécifiques ;
   - Une méthode afficher spécifique ;

   Et c'est tout. Vous avez mis en place de la généricité, et c'est bien.

Exercice 9 – Pour voir si vous avez bien compris…

Dupliquez votre code dans `TP4/ex9`.

Maintenant, vous allez appliquer ce qui a été fait pour la généricité des vues et du modèle aux quatre autres classes : `Genre`, `Nationalite`, `Categorie` et `DateEmprunt`. N'oubliez pas de vérifier, après transfert, que TOUT fonctionne.