

# Persistانس des objets

Technologies web



# Créer une base de données

Réaliser des solutions CRUD en PHP



# phpMyAdmin

- **phpMyAdmin** est une **interface web** inclus dans le pack **MAMP** permettant de gérer des **bases de données**
- Elle propose différentes options pour créer et gérer des **bases de données**, des **tables**, des **champs**, **ajouter des données**, en **supprimer**, etc.
- En d'autres termes, elle permet de **gérer des bases de données** sans nécessairement connaître le **langage SQL**
- Sur la page de démarrage de **MAMP**, il suffit de se rendre dans l'onglet « **TOOLS** » puis « **PHPMYADMIN** » pour accéder à l'interface

# Quelques définitions

- **SQL** (pour **Structured Query Language** ou **langage de requête structurée**) est un langage informatique servant à **lire, ajouter, modifier et supprimer** des données stockées dans des **bases de données relationnelles**
- Une **table SQL** représente une **entité** d'une **BDD** utilisée pour stocker des informations ordonnées dans des **colonnes** - C'est à sa création que l'on définit ses **colonnes** et le **type de données** (**entier, chaîne de caractères, date**, etc.)
- Une **colonne** apparaît dans une **table** et correspond à une **catégorie d'information**
- Une **ligne** apparaît dans une **table** et correspond à un **enregistrement** (ou **entrée**)
- Un **champ** représente **l'intersection entre une ligne et une colonne d'une table**

# Créer une base de données sur phpMyAdmin

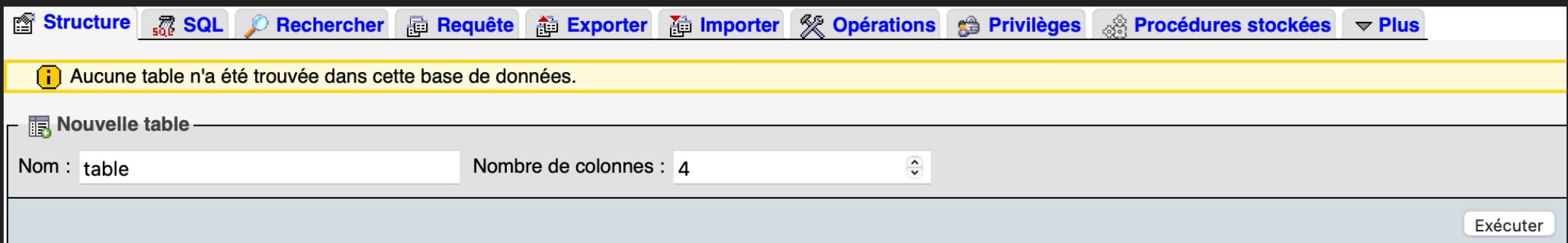
- Dans **phpMyAdmin**, il suffit de se rendre sur l'onglet « **Base de données** » et de saisir le nom de la nouvelle **base de données** avant de cliquer sur le bouton « **Créer** » (pas besoin de modifier jeu de caractère sélectionné)



- Notez qu'il ne faut jamais placer de caractères spéciaux dans les noms de vos **bases de données, tables, champs**, etc.

# Créer une table sur phpMyAdmin

- Pour créer une **table**, il suffit de se placer dans la **base de données** souhaitée puis de saisir le **nom de la table** et de sélectionner éventuellement le **nombre de colonnes** à la création  
- Celui-ci pourra être modifié par la suite



- Cliquez sur le bouton **Exécuter** pour lancer la **requête SQL** de **création de la table** de manière implicite

# Créer les colonnes d'une table sur phpMyAdmin

- On va maintenant définir les 4 **colonnes** de notre **table**
- Imaginons que nous souhaitons représenter une **personne** - Une **personne** aura donc au minimum un **nom**, un **prénom** et un **âge**
- On y ajoutera également un **identifiant unique** pour pouvoir différencier deux **personnes** ayant les mêmes **nom**, **âge** et **prénom**, par exemple
- Prenez l'habitude de systématiquement définir un **id** pour chacune de vos **tables** afin d'accéder facilement à un **enregistrement** donné
- Chaque **colonne** doit avoir un **type** associé - Ici, les colonnes **id** et **age** sont des **entiers** (**INT**) tandis que **firstname** et **lastname** sont des **chaines de caractères** (**VARCHAR**)

# Créer les colonnes d'une table sur phpMyAdmin

The screenshot shows the 'Structure' tab of phpMyAdmin for a table named 'table'. The table structure is defined as follows:

Nom	Type	Taille/Valeurs*	Valeur par défaut	Interclassement	Attributs	Null	Index	A_I
id_table	INT		Aucun(e)			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>
lastname	VARCHAR	20	Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>
firstname	VARCHAR	20	Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>
age	INT		Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>

Below the table structure, there are sections for 'Commentaires de table', 'Interclassement', 'Moteur de stockage', and 'Définition de PARTITION'. The 'Moteur de stockage' is set to InnoDB.

- Remarquez que nous avons sélectionné l'index **PRIMARY** sur la colonne **id** pour préciser qu'il s'agit de la **clé primaire** de notre **table**
- De son côté, le champ **A\_I** pour **Auto Incremental** permet d'augmenter automatiquement la valeur de l'**id** de **1** à chaque nouvel **enregistrement**

# Quelques définitions

- Une **clé primaire** permet d'identifier de manière unique un **enregistrement** dans une **table** - Elle peut être composée d'une ou plusieurs **colonnes**
- Une **clé étrangère** permet d'identifier une **colonne** d'une **table** faisant référence à une **colonne** d'une autre **table** - Elle permet donc de créer un lien entre 2 **tables**

# Modifier et supprimer et des lignes et des colonnes

Screenshot of the MySQL Workbench interface showing the 'Structure' tab for a table named 'table'.

**Structure de table**

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	<b>id_table</b>	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier  Supprimer  Plus
2	<b>firstname</b>	varchar(20)	utf8_general_ci		Non	Aucun(e)			Modifier  Supprimer  Plus
3	<b>lastname</b>	varchar(20)	utf8_general_ci		Non	Aucun(e)			Modifier  Supprimer  Plus
4	<b>age</b>	int(11)			Non	Aucun(e)			Modifier  Supprimer  Plus

Tout cocher Avec la sélection : [Parcourir](#) Modifier Supprimer Primaire Unique Index Texte entier

[Imprimer](#) [Suggérer des optimisations de structure](#) [Déplacer des colonnes](#) [Normaliser](#)

Ajouter 1  après age

**Index**

Action	Nom de l'index	Type	Unique	Compressé	Colonne	Cardinalité	Interclassement	Null	Commentaire
Éditer  Supprimer	PRIMARY	BTREE	Oui	Non	id_table	0	A	Non	

Créer un index sur 1  Exécuter

# Ajouter des enregistrements à une table

- Toujours dans la **table** de votre choix, rendez-vous dans l'onglet « **Insérer** » pour lui ajouter de nouveaux **enregistrements** (ou **entrées**)
- Ici, il suffit de saisir les **valeurs** souhaitées pour chacune des **colonnes** de votre **table** - Il est possible de générer plusieurs **entrées** en une **exécution** - Notez qu'il est inutile de donner une valeur à une **colonne auto incrémentale** comme l'**id**
- Cliquez enfin sur le bouton « **Exécuter** » pour lancer la **requête SQL** de manière implicite
- Notez qu'un bouton « **Exécuter** » apparaît pour chaque nouvel **enregistrement** et qu'un bouton du même nom est placé en bas de page pour permettre l'**ajout de plusieurs entrées en une exécution**

Colonne	Type	Fonction	Null	Valeur
id_table	int(11)	<input type="text"/>		<input type="text"/>
firstname	varchar(20)	<input type="text"/>		Chris
lastname	varchar(20)	<input type="text"/>		Chevalier
age	int(11)	<input type="text"/>		28

Ignorer

Colonne	Type	Fonction	Null	Valeur
id_table	int(11)	<input type="text"/>		<input type="text"/>
firstname	varchar(20)	<input type="text"/>		<input type="text"/>
lastname	varchar(20)	<input type="text"/>		<input type="text"/>
age	int(11)	<input type="text"/>		<input type="text"/>

Insérer comme une nouvelle ligne



et ensuite

Retourner à la page précédente



Continuer l'insertion avec  lignes

# La requête SQL **INSERT INTO**

Parcourir Structure SQL Rechercher Insérer Exporter Importer Privilèges Opérations Déclencheurs

✓ 1 ligne insérée.  
Identifiant de la ligne insérée : 1

```
INSERT INTO `ma-table` (`id_table`, `firstname`, `lastname`, `age`) VALUES (NULL, 'Chris', 'Chevalier', '28');
```

[Éditer en ligne] [ Éditer ] [ Créer le code source PHP ]

Exécuter une ou des requêtes SQL sur la table « todo-list.ma-table »: ⓘ

```
1 INSERT INTO `ma-table` (`id_table`, `firstname`, `lastname`, `age`) VALUES (NULL, 'Chris', 'Chevalier', '28');
```

Colonnes  
id\_table  
firstname  
lastname  
age

SELECT \* SELECT INSERT UPDATE DELETE Effacer Format Récupérer la requête auto-sauvegardée

Lier les paramètres ⓘ

[ Délimiteur ; ]  Afficher à nouveau la requête après exécution  Conserver la boîte de requêtes  ROLLBACK à la fin  Activer la vérification des clés étrangères

Exécuter

# Consulter les entrées d'une table

- En se rendant dans l'onglet « **Parcourir** », vous accédez aux **enregistrements** de la **table** dans laquelle vous vous situez

The screenshot shows the MySQL Workbench interface with the 'Parcourir' (Browse) tab selected. At the top, there's a toolbar with various icons: Parcourir, Structure, SQL, Rechercher, Insérer, Exporter, Importer, Privilèges, Opérations, and Déclencheurs. Below the toolbar, a green status bar displays: 'Affichage des lignes 0 - 0 (total de 1, traitement en 0.0004 seconde(s).)' and the SQL query: 'SELECT \* FROM `ma-table`'. To the right of the status bar are links for Profilage, Éditer en ligne, Éditer, Expliquer SQL, Créer le code source PHP, and Actualiser. The main area shows a table with one row of data:

	id_table	firstname	lastname	age
<input type="checkbox"/>	1	Chris	Chevalier	28

Below the table are buttons for Éditer, Copier, Supprimer, and Exporter. There are also buttons for Tout afficher, Nombre de lignes (set to 25), and Filtrer les lignes. At the bottom, there are additional buttons for Imprimer, Copier dans le presse-papiers, Exporter, Afficher le graphique, and Crée une vue.

- Notez que la **requête SQL** exécutée pour obtenir ce résultat s'affiche à l'écran

# Les entités

Persistante des objets

# De tableau à objet (1/2)

- En programmation procédurale, les données stockées en base sont contenues dans des **tableaux associatifs**

```
3 // '$db' est un objet issu de la classe PHP, PDO
4 $req = $db->query("SELECT nom, pv, pm, attaque FROM `personnage`");
5 $persos = $req->fetchAll(); // On récupère un tableau de tableaux de personnages
6
7 foreach ($persos as $perso) { // On parcours la tableau de personnages
8     echo "{$perso['nom']} a {$perso['pv']} PV, {$perso['pm']} PM et une attaque de {$perso['attaque']}.";
9 }
```

- Avec la **POO**, nous souhaitons travailler non plus avec des **tableaux** mais avec des **objets** - Il nous faut donc transformer le **tableau** reçu en **objet**

# La classe Personnage

- Pour travailler avec des **objets Personnage**, il nous faut, bien sûr, une **classe Personnage**

```
3  class Personnage {  
4      private $id;  
5      private $nom;  
6      private $pv;  
7      private $pm;  
8      private $attaque;  
9  }
```

- Nous avons configuré ces **attributs privés** afin de respecter le **principe d'encapsulation**, il nous faut donc des **getters** et **setters** pour accéder et mettre à jour leur valeur

```
3 class Personnage
4 {
5     private $id;
6     private $nom;
7     private $pv;
8     private $pm;
9     private $attaque;
10
11    // Getters
12    public function getId() {
13        return $this->id;
14    }
15
16    public function getNom() {
17        return $this->nom;
18    }
19
20    public function getPv() {
21        return $this->pv;
22    }
23
24    public function getPm() {
25        return $this->pm;
26    }
27
28    public function getAttaque() {
29        return $this->attaque;
30    }
```

```
32 // Setters
33 public function setId($id) {
34     $id = (int) $id; // Conversion en entier
35     if ($id > 0) { // L'id doit être supérieur à 0
36         $this->id = $id;
37     }
38 }
39
40 public function setNom($nom) {
41     if (is_string($nom)) { // Le nom doit être une chaîne de caractères
42         $this->_nom = $nom;
43     }
44 }
45
46 public function setPv($pv) {
47     $pv = (int) $pv; // Conversion en entier
48     if ($pv >= 0 && $pv <= 100) { // Les pv doivent être compris entre 1 et 100
49         $this->pv = $pv;
50     }
51 }
52
53 public function setPm($pm) {
54     $pm = (int) $pm; // Conversion en entier
55     if ($pm >= 0 && $pm <= 100) { // Les pm doivent être compris entre 1 et 100
56         $this->pm = $pm;
57     }
58 }
59
60 public function setAttaque($attaque) {
61     $attaque = (int) $attaque; // Conversion en entier
62     if ($attaque >= 1 && $attaque <= 100) { // L'attaque doit être compris entre 1 et 100
63         $this->attaque = $attaque;
64     }
65 }
```

# De tableau à objet (2/2)

- Reprenons le code permettant la récupération de données sous forme de **tableaux PHP** puis modifions-le pour pouvoir manipuler des **objets**

```
3 // '$db' est un objet issu de la classe PHP, PDO
4 $req = $db->query("SELECT nom, pv, pm, attaque FROM `personnage`");
5 $donnees = $req->fetchAll(); // On récupère un tableau de tableaux de personnages
6
7 foreach ($donnees as $donnee) { // On parcours la tableau de personnages
8     $perso = new Personnage($donnee); // On associe les valeurs du personnage à un objet
9     echo "{$perso->getNom()} a {$perso->getPv()} PV, {$perso->getPm()} PM et une attaque de {$perso->getAttaque()}.";
```

- Il est plus intuitif de travailler sur des **objets** plutôt que sur des **tableaux**, notamment sur des applications importantes comprenant de nombreux **objets**

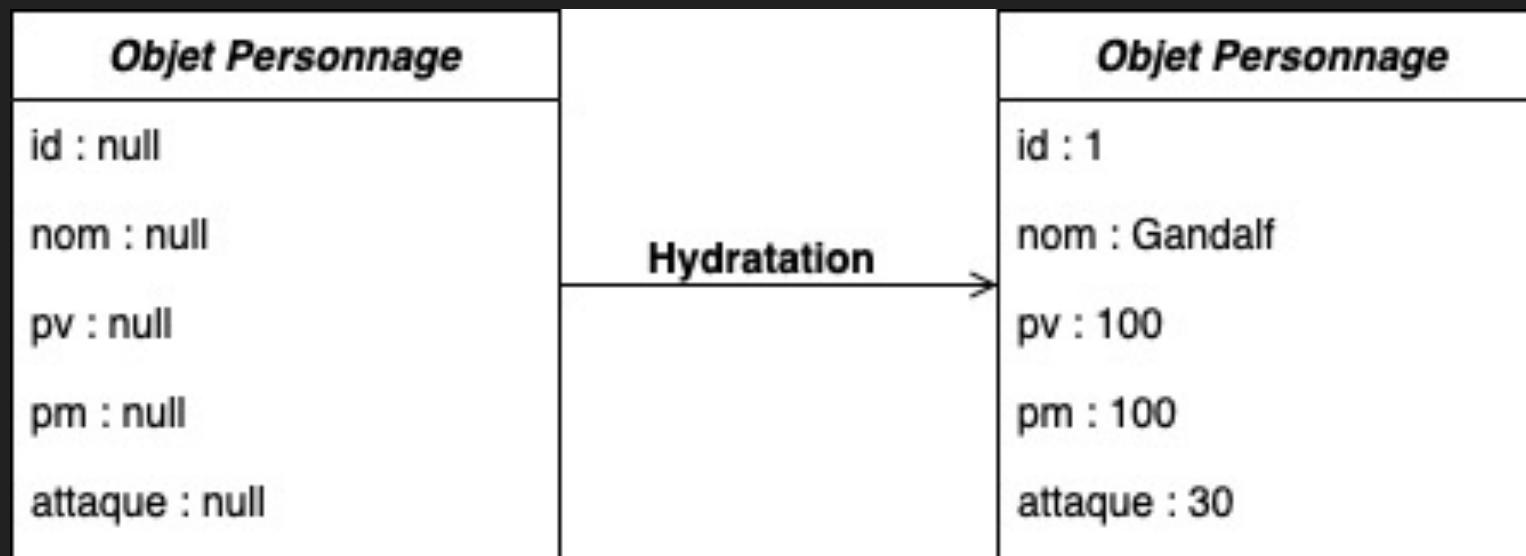
# L'hydratation

Persistante des objets

# L'hydratation en quelques mots...

- En **POO**, l'**hydratation** est une notion essentielle lorsque l'on travaille avec des **objets** stockés en base de données
- L'**hydratation** correspond au fait d' « **hydrater un objet** » - Cela veut dire qu'on lui apporte ce dont il a besoin pour fonctionner correctement
- Il s'agit donc de lui fournir les données correspondant à ses attributs afin qu'il leur assigne les valeurs souhaitées
- Imaginons un **objet** de type **Personnage** avec des **attributs** vides - L'**hydrater** consiste à le rendre fonctionnel en lui fournissant des **attributs** valides

# L'hydratation en image...



# L'hydratation en pratique...

- **Hydrater** un **objet** revient donc à assigner des valeurs à ses **attributs** - Il nous faut donc ajouter une **méthode** à notre **classe Personnage** permettant l'**hydratation** de ses **objets**
- Pour ce faire et ainsi respecter le **principe d'encapsulation**, il nous faut utiliser les **setters** que nous avons mis en place précédemment

```

10 public function hydrate(array $donnees) { // Reçoit un tableau de données
11     if (isset($donnees['id'])) {
12         $this->setId($donnees['id']);
13     }
14
15     if (isset($donnees['nom'])) {
16         $this->setNom($donnees['nom']);
17     }
18
19     if (isset($donnees['pv'])) {
20         $this->setPv($donnees['pv']);
21     }
22
23     if (isset($donnees['pm'])) {
24         $this->setPm($donnees['pm']);
25     }
26
27     if (isset($donnees['attaque'])) {
28         $this->setAttaque($donnees['attaque']);
29     }
30 }

```

```

3     $perso = new Personnage();
4
5     // Le tableau de données correspondant
6     $donnees = [
7         'id' => 1,
8         'nom' => 'Gandalf',
9         'pv' => 100,
10        'pm' => 100,
11        'attaque' => 30
12    ];
13
14     $perso->hydrate($donnees);

```

# Explications

- La **méthode** `hydrate()` vérifie, pour chacune des valeurs du **tableau associatif**, reçu en **paramètre**, si elle existe avant de l'assigner à l'**attribut** correspondant à l'aide des **méthodes setters**
- Sur la capture d'écran de droite, j'**instancie** la classe `Personnage` avant de l'**hydrater** avec le **tableau associatif** de données passé en **argument** de l'appel à la **méthode** `hydrate()`
- Cette **méthode** est fonctionnelle mais pas idéale - En effet, il nous faut la modifier pour chaque nouveaux **attributs** que l'on souhaitera intégrer à la **classe**
- Pensons plus générique...

# L'hydratation en pratique...

```
10  public function hydrate(array $donnees) {
11      foreach ($donnees as $cle => $valeur) {
12          $methode = 'set' . ucfirst($cle); // On récupère le nom du setter correspondant à l'attribut parcouru
13          if (method_exists($this, $methode)) { // Si le setter existe
14              $this->$methode($valeur); // On l'appelle
15          }
16      }
17  }
```

# Explications

- Pour rendre cette **méthode** plus générique, on boucle d'abord sur le **tableau associatif** de données fournit en **paramètre** en stockant chacun des couples **clé/valeur** dans des variables correspondant (`$cle` et `$valeur`)
- Au sein de cette boucle, on stocke dans une **variable** `$methode` la concaténation de la chaîne de caractère `"set"` avec la valeur contenue dans la **variable** `$clé` en prenant soin de passer la première lettre en majuscule grâce à la **fonction PHP** `ucfirst()`
- Cette concaténation donne les **chaînes de caractères** `setId` puis, `setNom`, `setPv`, etc.
- On s'assure que la **chaîne de caractères** ainsi stockée dans la variable `$methode` est bien accessible dans l'**objet courant** (`$this`) avant de l'appeler avec la valeur passée en **argument**

# Complément sur l'hydratation

- Cette **méthode** se retrouve dans de nombreux projets - Il est important de la retenir pour toujours **hydrater** correctement l'ensemble des **attributs** d'un **objet** de manière générique
- Une bonne pratique consiste à appeler cette **méthode** depuis le **constructeur** afin que l'**objet** soit correctement **hydraté** dès sa **création**

# Le manager

Persistante des objets

# Une classe, un rôle

- En **POO**, on évite au maximum d'écrire du code en dehors de nos classes afin de conserver une organisation optimale
- On pourrait alors être tenté d'écrire nos **requêtes SQL** directement parmi les **méthodes** de nos **classes**
- Or, en **POO**, il est essentiel de retenir qu'une **classe** correspond à un et un seul **rôle**
- En l'occurrence, un **objet** issu d'une **classe** comme **Personnage** a pour rôle de représenter une **ligne** d'une **table** en BDD
- La **classe Personnage** n'est donc pas censée **gérer** mais bien **représenter** des données - Pour les gérer, nous avons besoin d'un **manager**

# Un manager en quelques mots...

- Un **manager** est un **objet** chargé de gérer des **entités** issues d'une BDD
- Implémentons notre propre **manager** pour interagir avec nos personnages stockés en base
- Pour ce faire, nous avons besoin d'une nouvelle **classe** `PersonnagesManager`
- Nous souhaitons mettre en place un **CRUD (Create, Read, Update, Delete)** de personnages

# La classe manager

- Pour fonctionner correctement, un **manager** a nécessairement besoin d'une connexion à une BDD
- Pour ce faire, on stockera un **objet PDO** représentant cette connexion dans un **attribut** de la **classe**
- Prévoyons également un **setter** à appeler à la **construction** du **manager**
- Concernant ses **méthodes**, on souhaite pouvoir enregistrer une nouvelle **entité**, la modifier, la supprimer et la lire
- Voyons la structure de notre **classe PersonnagesManager**

```
3 # ./PersonnagesManager.php
4 class PersonnagesManager {
5     private $db; // Instance de PDO
6
7     // Méthodes
8     public function __construct() { // Connexion à la BDD MySQL "game"
9         $dbName = 'game';
10        $port = 8889; // Peut être 3306 (à vérifier sur MAMP ou WAMP)
11        $username = 'root';
12        $password = 'root'; // Chaine de caractère vide ("") sous WAMP
13        try {
14            $this->setDb(new PDO("mysql:host=localhost;dbname=$dbName;port=$port", $username, $password));
15        } catch(PDOException $exception) { // Si la connexion à la BDD échoue
16            echo $exception->getMessage(); // Un message d'erreur s'affiche à l'écran
17        }
18    }
19
20    public function setDb($db) {
21        $this->db = $db;
22    }
23
24
25    public function add(Personnage $perso) { // Ajoute le personnage $perso
26        // L'implémentation est détaillé plus bas dans ce cours
27    }
28
29    public function get(int $id) { // Retourne le personnage correspondant à '$id'
30        // L'implémentation est détaillé plus bas dans ce cours
31    }
32
33    public function getAll() { // Retourne tous les personnages
34        // L'implémentation est détaillé plus bas dans ce cours
35    }
36
37    public function update(Personnage $perso) { // Met à jour le personnage $perso
38        // L'implémentation est détaillé plus bas dans ce cours
39    }
40
41    public function delete(int $id) { // Supprime le personnage correspondant à '$id'
42        // L'implémentation est détaillé plus bas dans ce cours
43    }
44 }
```

# CRUD

Persistante des objets

# CRUD

- **CRUD** est l'abréviation des opérations **CREATE, READ, UPDATE et DELETE**
- Il s'agit des 4 **opérations de base** des interactions entre un site web et sa **base de données**
- Ces **opérations** servent bien souvent sur des sites **back office**
- Un **back office** est un site ou partie d'un site accessible uniquement par ses **administrateurs** et permettant de **gérer son contenu**, ses **fonctionnalités**, etc. - Il s'agit donc d'une **interface d'administration**
- Implémentons maintenant les différentes **méthodes** à l'aide de **requêtes SQL**

# CREATE

```
14     public function add(Personnage $perso) {
15         $req = $this->db->prepare("INSERT INTO `personnage` (nom, pv, pm, attaque) VALUES (:nom, :pv, :pm, attaque)");
16
17         $req->bindValue(":nom", $perso->getNom(), PDO::PARAM_STR);
18         $req->bindValue(":pv", $perso->getPv(), PDO::PARAM_INT);
19         $req->bindValue(":pm", $perso->getPm(), PDO::PARAM_INT);
20         $req->bindValue(":attaque", $perso->getAttaque(), PDO::PARAM_INT);
21
22         $req->execute();
23     }
```

# READ

```
25     public function get(int $id) {
26         $req = $this->db->prepare("SELECT * FROM `personnage` WHERE id = :id");
27         $req->bindValue(":id", $id, PDO::PARAM_INT);
28         $req->execute();
29         $donnees = $req->fetch();
30         $personnage = new Personnage($donnees);
31         return $personnage;
32     }
33
34     public function getAll() {
35         $personnages = [];
36         $req = $this->db->prepare("SELECT * FROM `personnage` ORDER BY name");
37         $req->execute();
38         $donnees = $req->fetchAll();
39         foreach ($donnees as $donnee) {
40             $personnage = new Personnage($donnee);
41             $personnages[] = $personnage;
42         }
43         return $personnages;
44     }
45 }
```

# UPDATE

```
42     public function update(Personnage $perso) {
43         $req = $this->db->prepare("UPDATE `personnage` SET nom = :nom, pv = :pv, pm = :pm, attaque = :attaque WHERE id = :id");
44
45         $req->bindValue(":id", $perso->getId(), PDO::PARAM_INT);
46         $req->bindValue(":nom", $perso->getNom(), PDO::PARAM_STR);
47         $req->bindValue(":pv", $perso->getPv(), PDO::PARAM_INT);
48         $req->bindValue(":pm", $perso->getPm(), PDO::PARAM_INT);
49         $req->bindValue(":attaque", $perso->getAttaque(), PDO::PARAM_INT);
50
51         $req->execute();
52     }
```

# DELETE

```
54     public function delete(int $id) {  
55         $req = $this->db->prepare("DELETE FROM `personnage` WHERE id = :id");  
56         $req->bindValue(":id", $id, PDO::PARAM_INT);  
57         $req->execute();  
58     }
```

# CRUD

- Testons ces implémentations dans le programme principal de notre projet - Imaginons un fichier `index.php` dans lequel je souhaiterai afficher le nom de chacun de mes personnages

```
1  <?php
2
3  // Fonction d'auto-chargement de classes
4  function loadClass($class)
5  {
6      require "$class.php";
7  }
8  spl_autoload_register("loadClass");
9
10 // Crée une instance de la classe "PersonnagesManager"
11 $personnagesManager = new PersonnagesManager();
12 // Stocke tous les objets "Personnage" renvoyés par la méthode "getAll()"
13 $personnages = $personnagesManager->getAll();
14
15 // Parcours le tableau d'objets "Personnage" pour afficher chacun de leur nom
16 foreach ($personnages as $personnage) { ?>
17     <p><?= $personnage->getNom() ?></p>
18 <?php } // Il ne faut pas oublier de fermer l'accolade du "foreach" !
19 ?>
```

# Projet - Site d'actualités (Modèle-Vue-Contrôleur)

Persistiance des objets

# Projet - La base de données

- Créez une base de données `blog` contenant une unique **table** `article` (Cf. diapos 5 à 8)
- La **table** `article` est composée de 5 **colonnes** (Cf. diapo 8)
  - `id` (`INT`) représentant l'**identifiant unique** d'un article - Il s'agit de la **clé primaire** - Sa valeur est **auto incrémentale**
  - `title` (`VARCHAR(60)`) représentant le **titre** d'un article
  - `content` (`TEXT`) représentant le **contenu** d'un article
  - `created_at` (`DATETIME`) représentant le **date** et **l'heure** de publication d'un article
  - `author` (`VARCHAR(50)`) représentant l'auteur d'un article

# Projet - La classe Article (Modèle)

- Créez une **classe Article** représentant un **article** (Cf. diapos 18 et 19)
- Un **article** est composé d'un **id**, d'un **titre**, d'un **contenu**, d'une **date et heure de publication** et d'un **auteur**
- Prévoyez un **getter** pour chaque **attribut**
- Prévoyez un **setter** pour chaque **attribut** - Un **id** est un entier supérieur à 0, le **titre**, le **contenu** et l'auteur sont des **chaînes de caractères**
- Ajoutez à votre **classe** une **méthode d'hydratation générique** (Cf. diapo 27) - Elle doit être appelée à la **construction d'objets** de type **Article**

# Projet - La classe **ArticlesManager** (Contrôleur)

- Créez une **classe ArticlesManager** chargée de gérer les articles stockés en BDD (Cf. diapo 34)
- Son seul **attribut \$db** représente une **instance de PDO**, son **constructeur** permet de lui assigner une valeur lui permettant de se connecter à votre BDD
- Prévoyez des **méthodes** pour gérer un **CRUD** des articles stockés en BDD (Cf. diapos 37 à 40)

# Projet - Les vues (Vue)

- Ajoutez 4 fichiers à votre projet :
  - `index.php` pour **afficher** tous les articles stockés en BDD
  - `create.php` pour **ajouter** un nouvel article en BDD
  - `update.php` pour **assigner** de nouvelles valeurs à un article déjà existant en BDD
  - `read.php` pour **lire** un article existant en BDD
  - `delete.php` pour **supprimer** un article existant en BDD
- Avant de d'implémenter quoi que ce soit, je vous conseille de dessiner des maquettes basiques sur papier pour imaginer la structure et l'aspect visuel de votre future site web
- Vous pouvez utiliser **Bootstrap** pour faciliter l'intégration visuelle de votre projet

# Projet - Proposition d'affichage final (1/3)

index.php

Coding News Tous les articles Rédiger un article

**Lorem ipsum**

Dolor sit amet

**Lorem ipsum dolor**

Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed do  
eiusmod tempor incididunt ut labore ...



hexagone

Ecole Supérieure d'Informatique

**Un programme  
pluridisciplinaire en  
informatique.**

Les 2 premières années forment le  
Cycle Préparatoire en formation initiale.  
Les 3 années suivantes f...



# Projet - Proposition d'affichage final (2/3)

create.php

Coding News Tous les articles Rédiger un article

## Rédiger un nouvel article

Titre

Le titre de l'article

Contenu

Le contenu de l'article

URL

L'URL de l'image

**Publier**

update.php

Coding News Tous les articles Rédiger un article

## Rédiger un nouvel article

Titre

Lorem ipsum dolor

Contenu

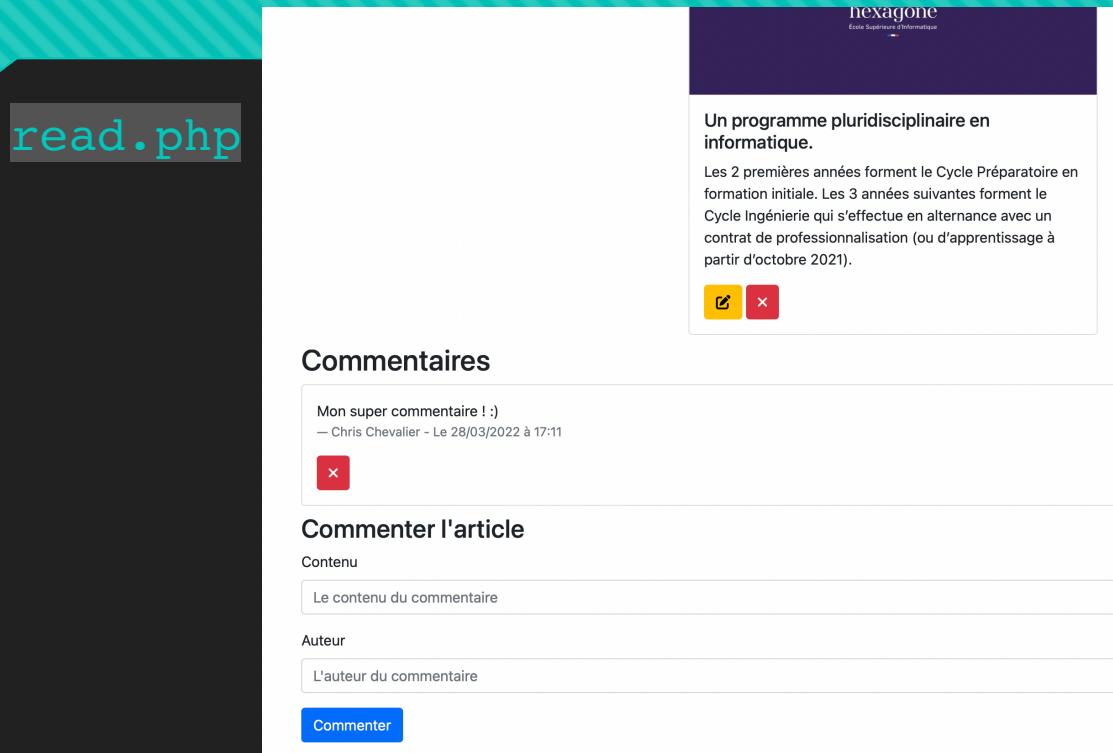
Lorem ipsum dolor sit amet, consectetur adipisicing  
ullamco laboris nisi ut aliquip ex ea commodo con

URL

<https://www.taptouche.com/fr/blogue/lorem-ipsum>

**Publier**

# Projet - Proposition d'affichage final (3/3)



- Mon **site d'actualités** est disponible à cette adresse : <https://serene-reef-11955.herokuapp.com> - Vous êtes libre de vous inspirer de son rendu visuel

# Projet - Pour aller plus loin... (Cf. Critères d'évaluation)

- Afin de développer un **site d'actualités** un peu plus complet, mettez en place un **CRUD des commentaires** d'article
- Un **commentaire** est donc caractérisé par son **id**, son **contenu**, son **auteur**, et l'**id de l'article** auquel il est associé (**clé étrangère**)
- Je vous suggère également de travailler le **style** visuel de votre application ainsi que son aspect **responsif**
- Il vous est également demandé d'implémenter au moins une fonctionnalité en **JavaScript** permettant d'afficher ou masquer les **commentaires** en cliquant sur un bouton
- Enfin, vous pouvez intégrer une **fonctionnalité bonus** (non demandée ici) à votre projet - Soyez inventif !
- Les **critères d'évaluation** sont détaillés dans le PDF du même nom communiqué par votre formateur

# Des questions ?