PHP MVC

Merci de vous installer par binôme de niveau différent

skema BUSINESS SCHOOL

EFMD EQUIS ACCREDITED

Objectif

- **Structure** de code **professionnel**

Objectif

➢ **Structure** de code **professionnel**

➢ **Modulaire**: découpé en de nombreux fichiers où chaque fichier à un et un seul rôle
➢ **Découplé**: chaque fichier fonctionne indépendamment
➢ **Documenté**: commentaires placés sur les classes et les méthodes, fichier readme. possibilité de générer automatiquement ce fichier avec phpdoc. expliquer le pourquoi, pas le comment
➢ **Simple**: utiliser seulement le niveau de complexité / abstraction necessaire
➢ **Anglais**: langue des devs

Avantages

➢ **Réutilisable:** un travail bien effectué peut souvent resservir dans d'autres projets
➢ **Partageable:** les fichiers étant indépendant, on peut travailler à plusieurs dessus
➢ **Évolutif:** apte au changement, il est facile d'ajouter de nouvelles fonctionnalités en gardant un code bien organisé

Concrètement

➢ **On va déplacer** des morceaux de code dans de nouveaux fichiers
➢ **On va renommer** certaines fonctions (en anglais)

Les étapes

➢ **Faire un code qui marche** (c'est ce qu'on a fait jusqu'ici)
➢ **Faire un code modulaire** (c'est ce qu'on va faire aujourd'hui)
➢ **Utiliser un framework:** symfony, zend, votre propre framework (à vous d'aller plus loin en fonction de vos intérêts et des besoins du marché du travail)

Les étapes

➢ **Attention à ne pas le faire à l'envers.** Souvent on utilise un framework parce que c'est la mode ou parce qu'on nous l'impose. Or, il faut avoir compris pourquoi on a besoin de ce framework pour comprendre tout son sens. Ainsi, vous ne serez pas perdu dans l'architecture souvent complexe de ces frameworks.

Exercices

- Importez la base de donnée mise à disposition
- Affichez ces 2 articles dans une page
  - de la façon la plus simple possible

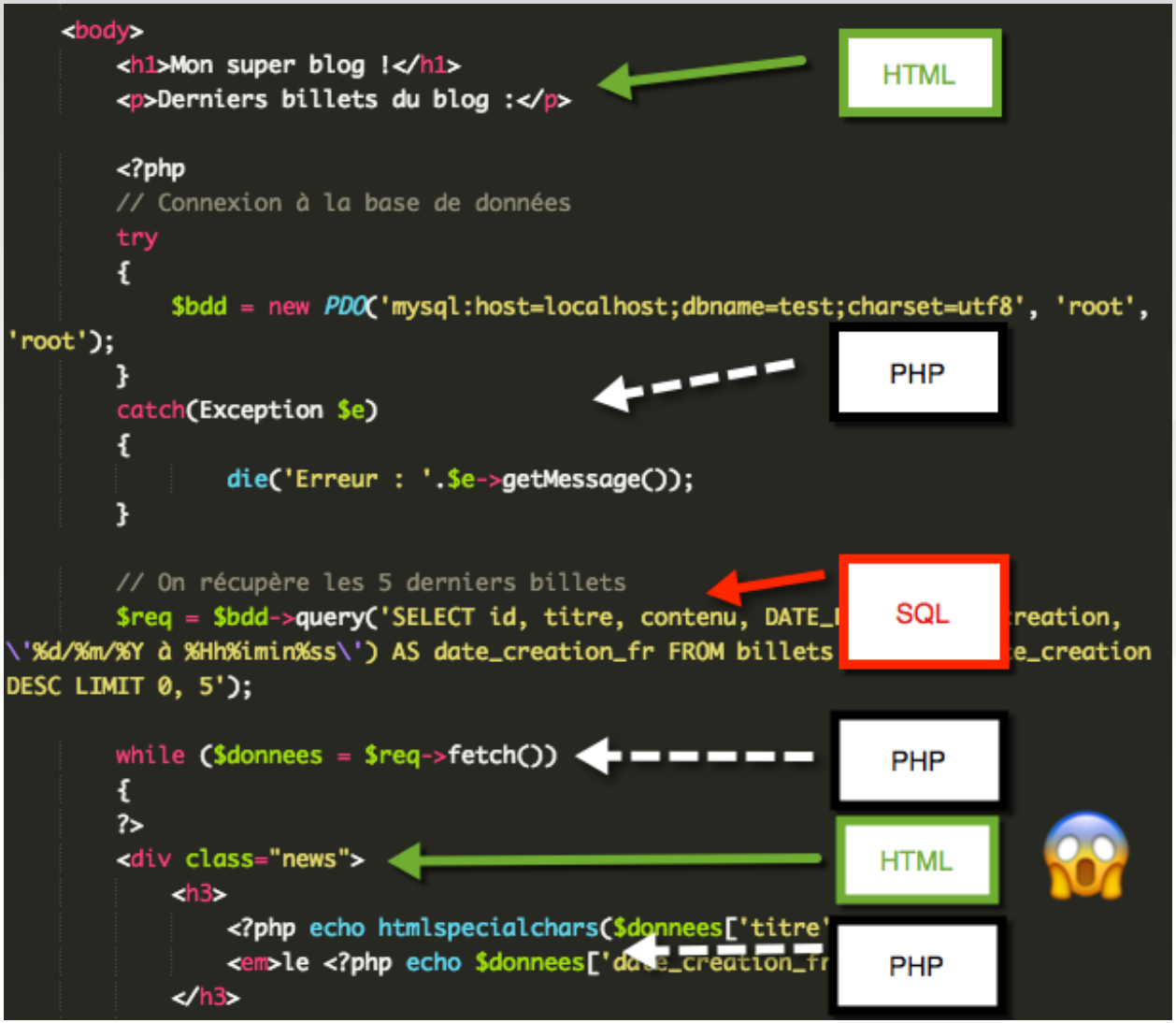
Analyse

➢ Certe le code fonctionne. Cependant, lorsque vous allez vouloir ajouter des fonctionnalités, le fichier va grossir et faire plusieurs centaines de ligne.

➢ Ca continueras à fonctionner.
➢ Ca ne sera pas forcément plus lent.

➢ Cependant le code ne sera pas intelligible.
- Lorsque vous allez le reprendre quelques semaines plus tard, il vous semblera trop compliqué et vous allez être découragé.
- Encore pire, imaginez une personne extérieur qui tente de comprendre votre code: il va vous détester

Analyse



A améliorer

➢ 1: langue du code: utiliser l'anglais pour le code PHP. Le code affiché à l'utilisateur reste lui en français.
➢ 2: commentaires: faire des commentaires orientés « objectif »
➢ 3: modularité du code: tout est dans un seul fichier. Séparer le HTML, CSS et PHP

Exo v2

➢ Séparez dans le php le code qui permet de récupérer les données de celle qui permet de les afficher, dans le même fichier
➢ Sortez le CSS dans un fichier externe

Exo v3

- Séparez en 2 fichiers:
  - index.php (il y a toujours un index.php, c'est le fichier de base de votre site lu en premier) : il s'occupera de récupérer les données et d'appeler l'affichage.
  - affichageAccueil.php : il affichera les données dans la page.

- Vous venez de faire ce qu'on appelle de la refactorisation

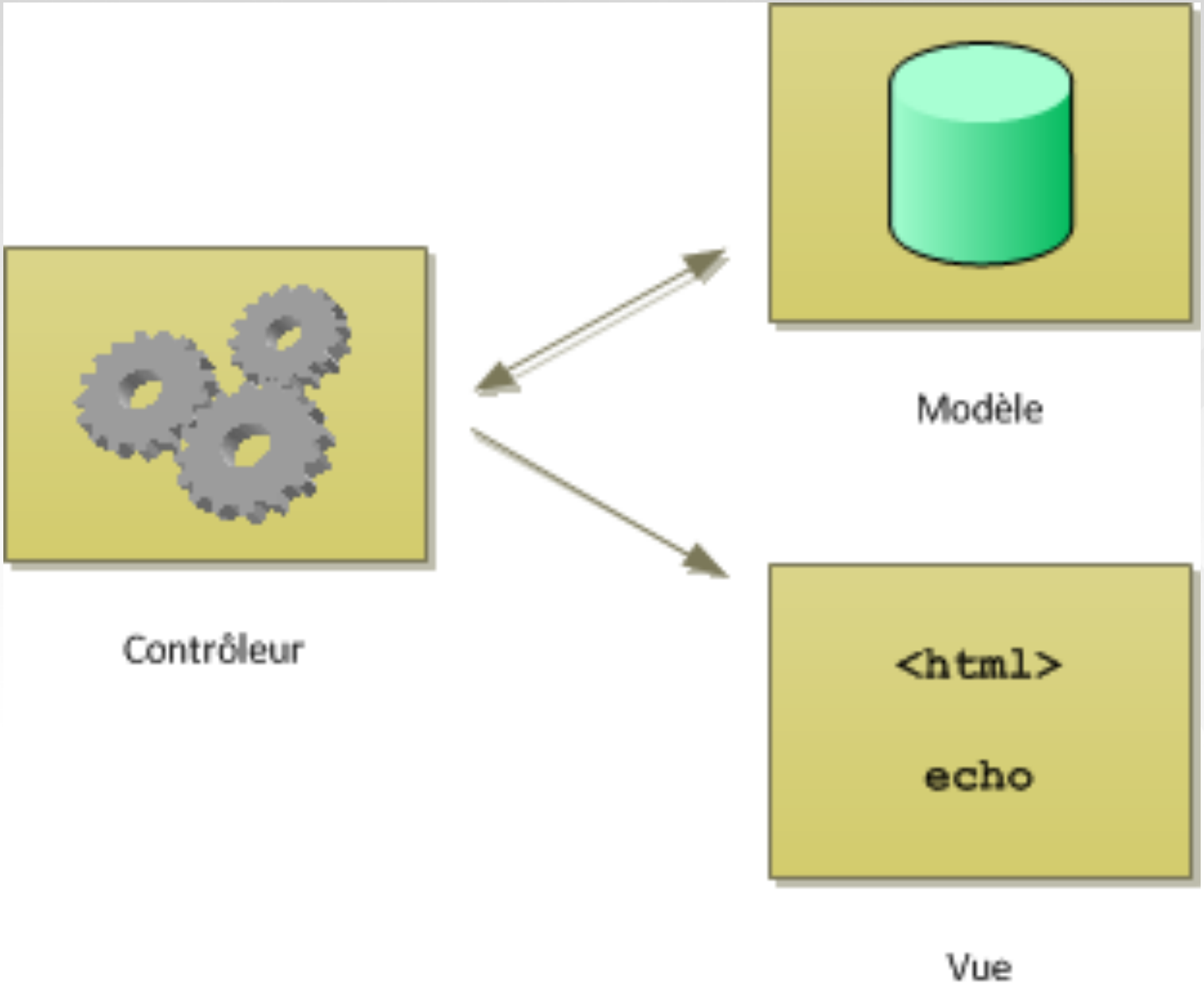
Exo v4

➢ Séparez complètement l'accès aux données (tout le traitement SQL) dans un fichier spécifique:

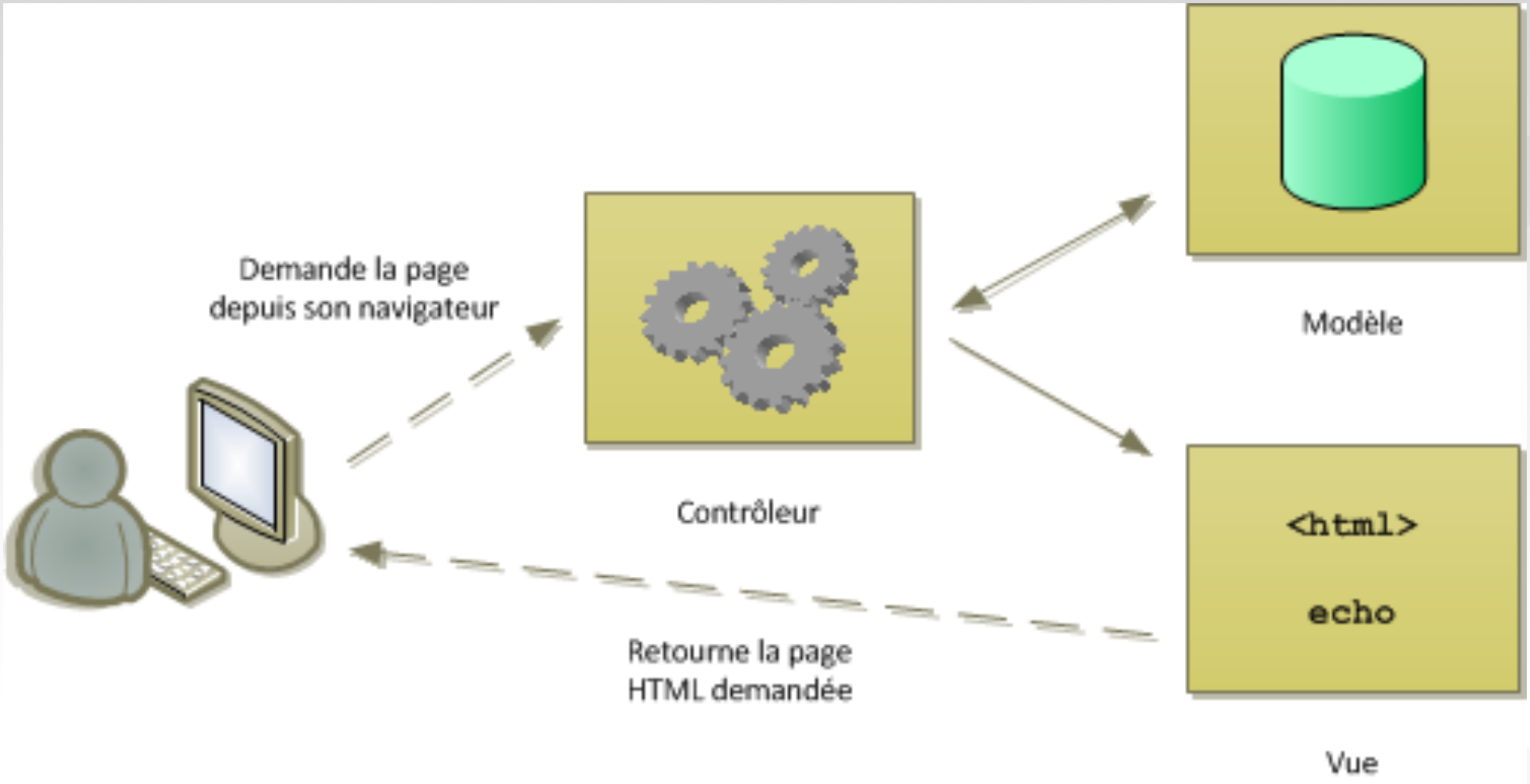
- modele.php : se connecte à la base de données et récupère les billets via une fonction getBillets
- affichageAccueil.php : affiche la page. Ce fichier ne va pas changer du tout.
- index.php : fait le lien entre le modèle et l'affichage

➢ **Nous venons de faire du MVC (Modele, Vue, Contrôleur)**

Architecture MVC



Architecture MVC



Description de la correction

➢ On charge le fichier du modèle. Il ne se passe rien pour l'instant, parce qu'il ne contient qu'une fonction.
➢ On appelle la fonction, ce qui exécute le code à l'intérieur de modele.php . On y récupère la liste des billets dans la variable $req.
➢ On charge le fichier de la vue (l'affichage), qui va présenter les informations dans une page HTML.

MVC, un design pattern

➢ MVC est un design pattern (bonne pratique de développement)

➢ Le pattern MVC permet de **bien organiser** son code source. Il va vous aider à **savoir quels fichiers créer**, mais surtout à **définir leur rôle**. Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

L’architecture MVC: MV

➢ **Modèle** : cette partie gère les **données** de votre site. Son rôle est d'aller récupérer les informations « brutes » dans la **base de données**, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autres les **requêtes SQL**.

➢ **Vue** : cette partie se concentre sur **l'affichage**. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher par exemple une liste de messages.

L’architecture MVC: C

➢ **Contrôleur** : cette partie gère la **logique** du code qui prend des décisions. C'est en quelque sorte **l'intermédiaire** entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, **prendre des décisions** et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d’accès).

L’architecture MVC: C

➢ **Contrôleur** :

Le contrôleur est le **chef d’orchestre**.
Exemple 1: il reçoit une requête index.php: il va alors se charger de renvoyer tout les articles
Exemple 2: il reçoit une requête index.php?page2: il va alors se charger de renvoyer seulement les articles 10 à 20.

Si il y a des autorisations d’accès (exemple: la page admin ne doit être accessible qu’à une personne logée avec un statut « admin »), c’est également le contrôleur qui s’en charge.

Exo v5

➢ Voici ce que nous allons améliorer :

➢ **L'anglais**: les fonctions, la BDD et le code en général (sauf bien sur au texte français que nous ajoutons à l'écran) doivent être en anglais
➢ **La balise de fermeture PHP**: Les fichiers contenant uniquement du php n'ont plus de « ?> » à la fin (norme PSR-2)
➢ **L'utilisation de short open tags** <?php echo htmlspecialchars($data['title']); ?> deviendra <?= htmlspecialchars($data['title']) ?>
➢ L'indentation

Correction:

➢ L'affichage (aussi appelé "vue" ou "view" en anglais) contient uniquement du code en anglais ($donnees est devenu $data par exemple).
Le code est un peu mieux indenté : on décale toujours d'un cran vers la droite le code à l'intérieur d'une balise. Le décalage se fait toujours en ajoutant 4 espaces à chaque fois qu'on veut décaler.

Problème

- Les articles ne s’affichent pas. Pourquoi?

Solution

- Regardez la base de donnée

Retour sur les étapes

➢ Pour le moment, on ne voit pas un grand intérêt à avoir ajouter autant de complexité au projet. C'est lorsque le projet va grandir que l'on va s'en rendre compte.

➢ C'est le but de la suite où nous allons ajouter un système de commentaires et voir ainsi que notre code est plus clair que si nous étions resté sur la création d'un simple « code qui marche » sur une page.

Création des commentaires

➢ On voudrait un système qui prenne en paramètre un identifiant d'article et qui affiche cet article et en dessous ses commentaires

➢ Que devons nous mettre à jour?
➢ Comment s'y prendre?

Création des commentaires

➢ On voudrait un système qui prenne en paramètre un identifiant d'article (ex: index.php?id=2) et qui affiche cet article et en dessous ses commentaires

➢ Que devons nous mettre à jour?
  ➢ Il faut mettre à jour aussi bien le Modèle, la Vue que le Contrôleur
➢ Comment s'y prendre?
  ➢ On commence généralement par le Modèle et d'abord la base de donnée sur laquelle elle va travailler

➢ Note pour plus tard: c'est pour cela qu'on démarre généralement de l'UML / MCD

Commentaires Exo 1

- Création du nécessaire en base de donnée pour accueillir les commentaires
  - 1. Que créer ?
  - 2. Quoi y mettre ?
  - 3. Ou le faire?

- LETS'GO !

Création des commentaires

➢ On continue à créer le modèle en mettant cette fois à jour la page model.php

- ➢ 1. **Elle doit contenir la récupération d'un post et la récupération de ses commentaires**
- ➢ 2. Elle ne doit pas répéter de code

Commentaires Exo 2

- Créez les fonctions qui:
  - récupère un post en fonction d'un id passé en GET.
    - 1. Où va d'abord se placer l'id dans le code?
  - qui récupère un commentaire en fonction d'un id passé en GET

Création des commentaires

- On continue à créer le modèle en mettant cette fois à jour la page model.php
  - 1. Elle doit contenir la récupération d'un post et la récupération de ses commentaires
  - 2. **Elle ne doit pas répéter de code**

Commentaires Exo 3

- Vous avez effectué dans chacune des fonctions une connexion à la base de donnée. Votre code est donc dédupliqué.
  - 1. Que devez vous faire?
  - 2. Comment?

LET'S GO!

Création des commentaires

➢ Quelle précautions prendre avant de récupérer un articles et ses commentaires d'un certain id?
➢ Ou le faire?

Commentaires Exo 5

- Votre contrôleur actuel charge tout les posts
- Faites un contrôleur qui charge un post et ses commentaires associés et qui avant cela vérifie qu'un id existe

Finalisation: v6

➢ Insérez l’ensemble de vos pages nouvellement créé dans les exo pour faire un site qui fonctionne

Le templating

➢ Nous avons des blocs répétitif de page dans les view, or on veut éviter les répétitions.
➢ Pour éviter cela, on avait jusqu'ici appris à séparer les blocs répétitif des pages (le header et le footer) dans des fichiers à part: header.php et footer.php
➢ Problème: 1. que faire si on veut que le title, situé dans le header soit différent à chaque page?

➢ La meilleure solution est de retourner le problème avec le principe du **templating**

Le templating

- Le template reprend toute la structure de la page et la remplace de « trou » qui sont des variables php: ici: title et content
- Dans la page indexView, vous devez définir ces variables php: title et content
  - title est simple
  - content à plus de contenu, on va donc utiliser ob_start(); et ob_get_clean(); pour réunir tout un bloc. Tout ce qui sera entre ob_start(); et le $var =ob_get_clean() ira dans la $var

Exo 7

- Faire un système de templating

Exo 7 correction

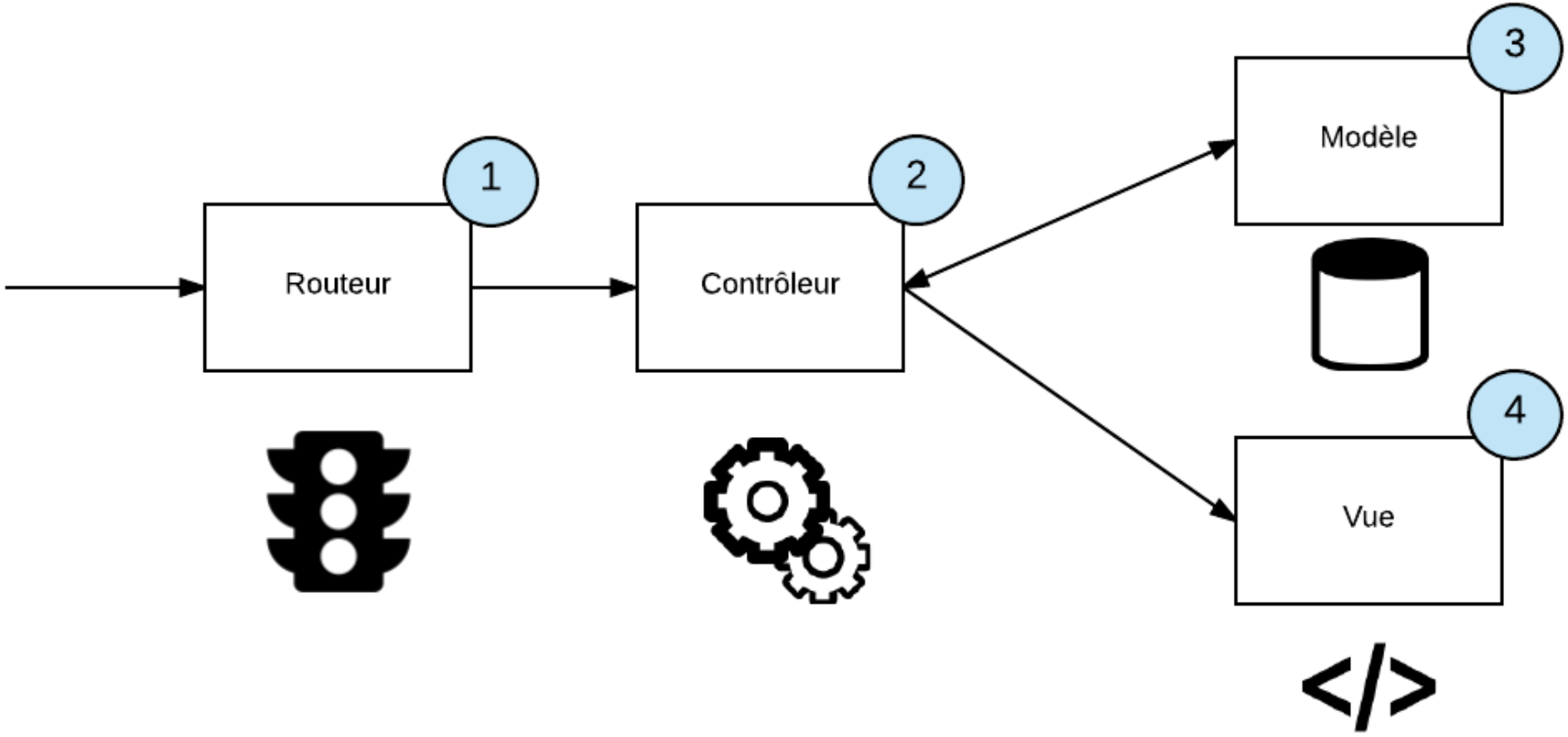
```php
<?php ob_start(); ?> Début
<p>Derniers billets du blog :</p>


<?php
while ($data = $posts->fetch())

?>
    <div class="news">
        <h3>
            <?= htmlspecialchars($data['title']) ?>
            <em>le <?= $data['creation_date_fr'] ?></em>
        </h3>

        <p>
            <?= nl2br(htmlspecialchars($data['content'])) ?>
            <br />
            <em><a href="post.php?id=<?= $data['id'] ?>">Commentaires</a></em>
        </p>
    </div>
<?php

posts->closeCursor();

<?php $content = ob_get_clean(); ?> Fin
```

Tout le code généré ici va dans $content

Le routing

➢ Notre fichier contrôleur (index.php) manque d'organisation: il deviendrait vite incompréhensible en présence de plusieurs pages
➢ Il faut un système qui appelle le bon contrôleur en fonction de la page (un contrôleur frontal)
➢ C'est le principe du **routeur**

Le routing



Le routing: exercice v8

➢ Faites une page controller.php qui mette sous la forme de fonction les 2 pages que le système peut actuellement appelé

➢ Faites une page index.php qui en fonction d'un variable « action » va faire les tests et appeler le bon contrôleur, par défaut elle appellera le contrôleur qui affiche tout les posts

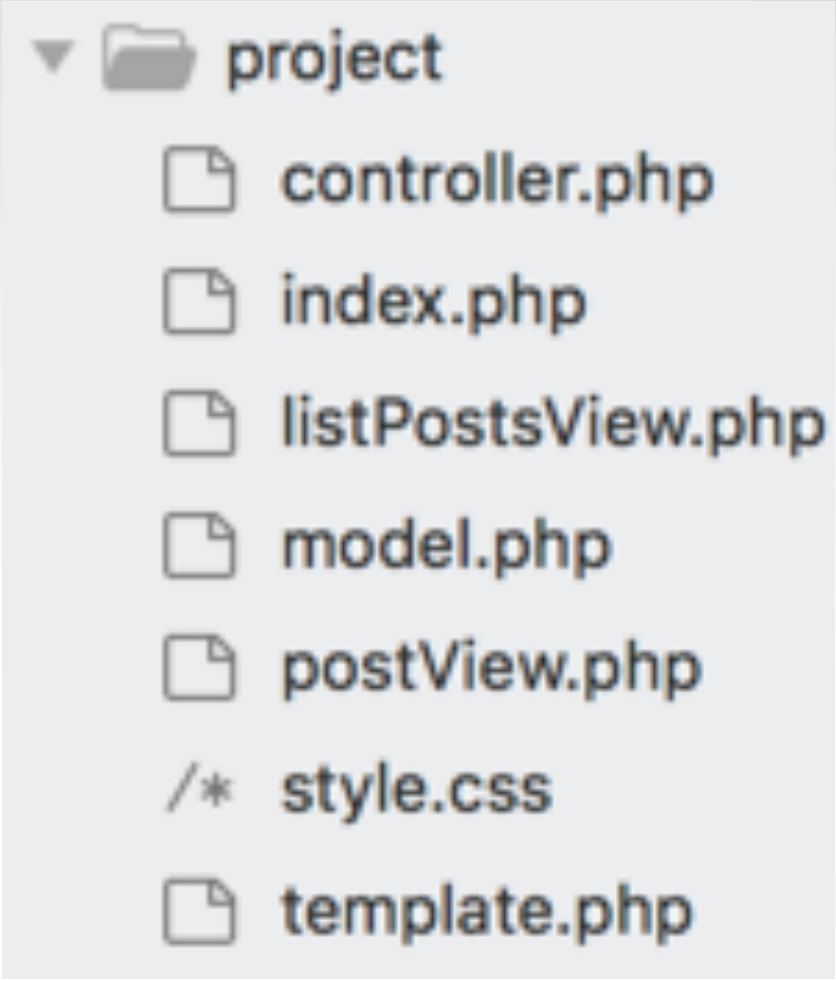
➢ Tester votre système en ne mettant pas d'action puis un mettant un action post et un id 2

Le routing: exercice v8 correction

➢ Dans le routeur (index.php) on charge le fichier controller.php pour mettre les fonctions en mémoire
➢ On teste la variable action passée en GET pour savoir quel contrôleur appeler.

Organiser ses fichiers en MVC

➢ Actuellement les fichiers sont un peu fouillis:



Organiser ses fichiers en MVC

➢ On va organiser nos fichiers et dossiers pour qu'il prenne une architecture MVC:
controller/ : le dossier qui contient nos contrôleurs.
view/ : nos vues.
model/ : notre modèle.

➢ Il ne restera en fait que le fichier `index.php` (notre routeur) à la racine... parce qu'il faut bien appeler un fichier à la base !

Organiser ses fichiers: exo v9

➢ Organisez vos fichiers de la façon suivante
controller/ : le dossier qui contient nos contrôleurs.
view/ : nos vues.
model/ : notre modèle.

➢ 1. Que ne doit on pas oublier?

Organiser ses fichiers: exo v10

➢ Le MVC est une architecture de projet
➢ Pour l'instant il n'y a qu'un fichier par dossier, ex: controller/controller.php. Cependant, il y aura plus de fichiers quand le site va grandir: avec par exemple un espace d'administration.
➢ On va donc devoir discerner ce qui est du front office (ce qu'on affiche à l'écran) et ce qui est du back office (l'administration)

➢ Exo: Renommer les fichiers dans les dossier pour leur donner leur rôle (frontoffice ou back office)

Ajouter des commentaires

- Dans quel ordre ajouter les fonctionnalités?
  - Écrire le modèle
  - Écrire le contrôleur pour récupérer les informations du modèle et les envoyer à la vue
  - Écrire la vue
  - Mettre à jour le routeur pour envoyer vers le bon contrôleur

Ajouter des comment.: exo v11

➢ 1: Modèle: commencez par écrire la fonction postComment d'insertion d'un commentaire
➢ 2: Contrôler: récupère les informations dont on a besoin et les renvoie au modèle
➢ 3: Vue: mettre à jour la vue pour qu'un formulaire récupère les données du commentaire et les envoie

Gestion des erreurs

➢ Pour le moment, nos erreurs sont envoyés dès qu'elles apparaissent, ce qui rend le code moins organisé

➢ PHP prévoit une fonction pour organiser ces erreurs: le try… catch… (1)

➢ Pour pouvoir organiser les erreurs et les envoyer à un moment précis, on utilise le throw new Exception (2)

Gestion des erreurs: exo v12

- Utilisez les try catch throw sur le routeur pour récupérer l'erreur en cas de:
  - Indiquez qu'aucun identifiant de billets n'a été envoyé
  - Tous les champs ne sont pas remplis
  - Aucun identifiant de billet envoyé
  - Afficher les éventuelles erreurs à la fin du code

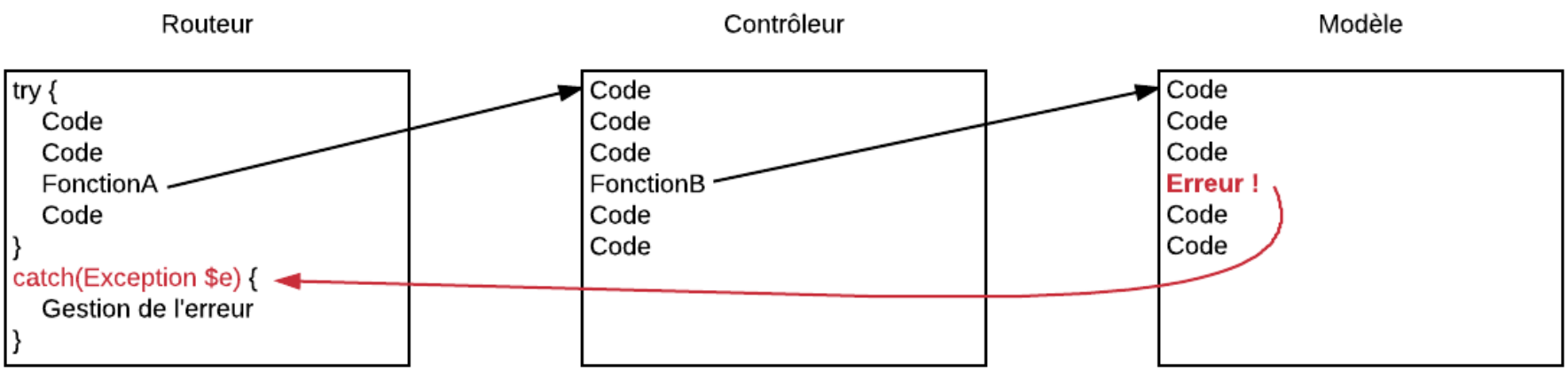
Gestion des erreurs

➢ Utile aussi pour remplacer les die par des erreurs bien prises en chargent

➢ Exo v13: remplacer le die du addComment du contrôleur par la nouvelle gestion des erreurs

Gestion des erreurs

➢ L'erreur va être remontée jusqu'au catch



Gestion des erreurs: exo v14

➢ Ducoup on peut supprimer le try catch de la connexion à la db

LET'S GO

La POO

- Programmation Orientée Objet
  - Vous verrez parfois phpOO
  - Un langage comme Java est full objet, donc nul besoin de le préciser

- Exemple de la boite noire

La POO: création de class

➢ Une classe commence par le mot clé class
➢ On précise ensuite les variables, appelées attributs dans le cadre de l'OO
➢ On précise après les fonctions, appelées méthodes dans le cadre de l'OO

La POO: création de class: exo v15

➢ Créez une class Maison qui à pour attributs: porte et température et comme méthodes ouvrirPorte, fermerPorte et modifierTemperature

➢ Par convention, les noms des class commencent toujours par une majuscule

La POO: instantiation de class

➢ Dans un nouveau fichier, instancier une class: `$maisonDeMathieu = new Maison();` ici $maisonDeMathieu est un objet

➢ Appliquer une méthode sur un objet: $maisonDeMathieu->ouvrirPorte();

La POO: instantiation de class

- Exo v16
  - (1) Instanciez la maison de Julie
  - Ouvrez la porte, augmentez la température à 21 et fermez la porte

La POO: retour sur le blog

- L'objectif est d'encapsuler le modèle dans des class:
  - PostManager : un gestionnaire de post de blog
  - CommentManager: un gestionnaire de commentaire
  - on aura donc 2 fichiers
- Pourquoi les appeler « Manager »: car elles vont aider à « manager » les post et les commentaires

La POO: exo v17

- Repartez de v14 et mettez le modèle sous la forme de 2 fichiers:
  - classPostManager.php
  - classCommentManager.php
- Faites en des class contenant les méthodes qui correspondent à leur fonction actuelles

La POO: exo v17 correction

- 2 remarques:
  - La méthode dbConnect est privée car elle ne doit être utilisé qu'à l'intérieur de la class
  - On utilise this->dbConnect() pour faire référence à une fonction membre de la classe.

La POO: maj du controller

➢ Il faut maintenant mettre à jour le contrôleur pour que celui ci appelle des méthodes et non des fonctions.

LET'S GO

La POO: maj du controller

➢ Correction:
➢ On fait un require_once pour s'assurer que la classe n'est appelée qu'une fois.
➢ On instancie les class et on appelle ensuite la bonne méthode

La POO: l'héritage

- Une des forces de l'OO
- On va l'utiliser pour éviter de dédupliquer du code.

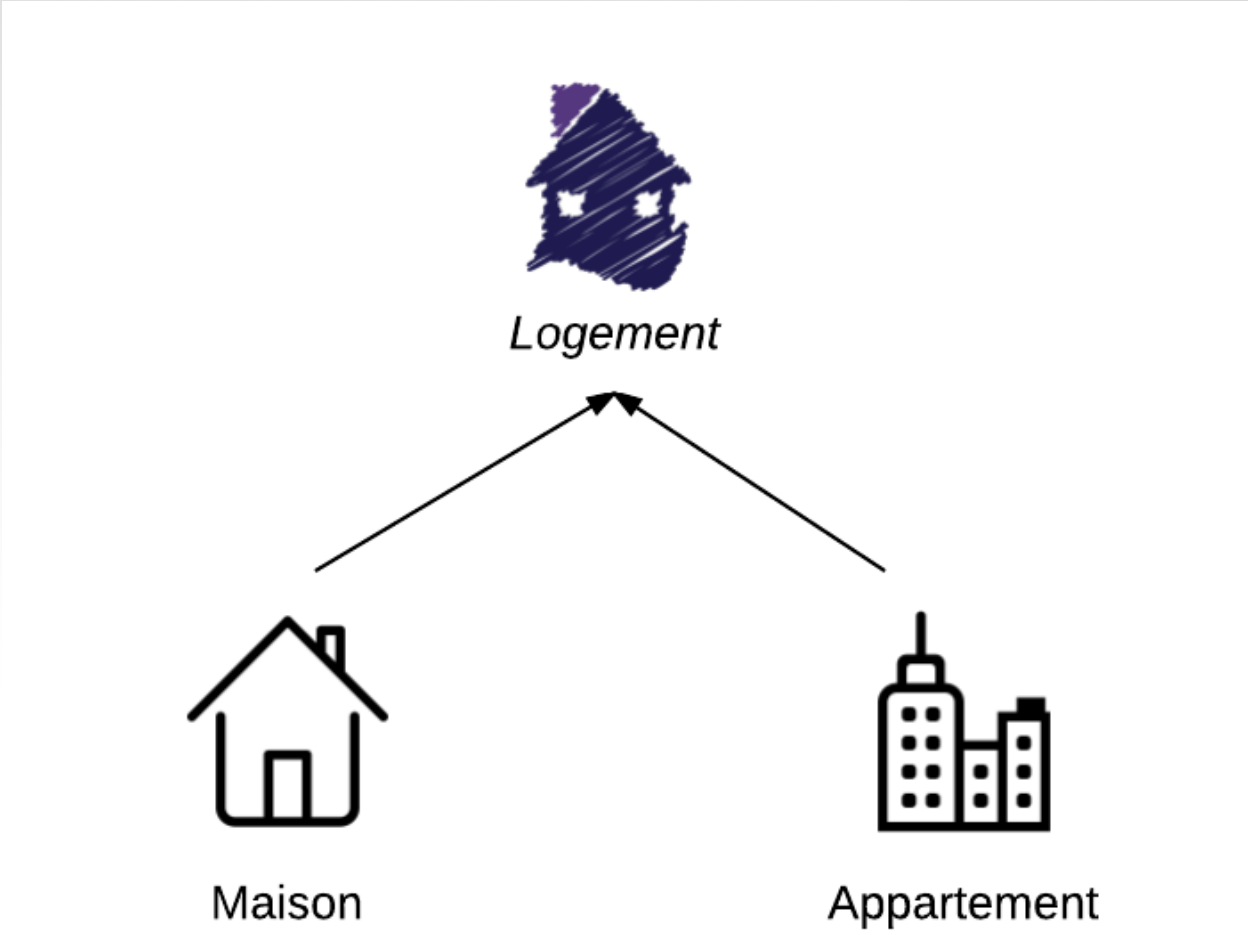
Question: quel code avons nous dupliqué? (1)

La POO: principe de l'héritage

➢ Imaginons que nous travaillons à coder des animaux: chats et chiens.
➢ Ces chats et ses chiens ont des méthodes spécifiques: miauler pour le chat, aboyer pour le chien. Cependant elles ont des fonctions communes: manger
➢ Pour la fonction manger, on pourrait donc créer une class animal qui va contenir la méthode commune aux 2 animaux: la méthode manger

La POO: principe de l'héritage

➢ Exemple avec les logements:



La POO: principe de l'héritage

- Dans cet exemple: où placeriez vous:
  - ouvrirPorte()
  - fermerPorte()
  - prendreAscenseur()
  - allerDansLeJardin()

La POO: principe de l'héritage

- Le mot clé pour l'héritage: extends

- Concrètement on va écrire:
  - class Maison extends Logement
  - class Appartement extends Logement

La POO: principe de l'héritage: Exo

➢ Utilisez l'héritage pour faire une class Manager qui va recevoir la méthode commune aux 2 classes (1)
➢ Implémentez l'héritage aux 2 classes concernés (2)

La POO: principe de l'héritage

N'oubliez pas de supprimer la connexion à la db dans les class post et comment
La class manager possède sa méthode ni en private ni en public mais en protected, ce qui la rend accessible uniquement à ses classes « filles ».

Exercice final

➢ Créer un système de livre d'or sur votre CV en ligne qui reprenne tout les concepts vu ici:

- ➢ Création d'une BDD adaptée
- ➢ Création du modèle en OO
- ➢ Création des contrôleurs
- ➢ Création des vues adaptés

Le SQL

➢ Nous avons déjà vu les méthodes Insert (Create) et Select (Read). Or le SQL à encore 2 autres options importante. Indice: CRUD (1)

➢ Exo: Faites les fonctions qui permettent de supprimer et et mettre à jour un commentaire. Indice: Vous avez ces fonctions dans les exercices finaux du CV de base.

Le SQL

➢ Retour sur les bases
➢ Travail dans phpMyAdmin, administrateur de bdd (base de donnée)
➢ Importez jeux-video.sql. Que doit-ton faire avant? (1)
➢ Essayer la requête SELECT * FROM jeux_video (2)
➢ Essayer la requête SELECT nom FROM jeux_video
➢ Essayer la requête SELECT nom, console FROM jeux_video

➢ Exo: Afficher le nom, le commentaires et le prix des jeux videos

Le SQL: exo v19

➢ Retour au PHP:

➢ Utilisez les fonctions PHP de MySQL pour afficher un paragraphe pour chacun des jeux. Ce paragraphe doit contenir toutes les informations du jeu sous forme de phrase.

Le SQL: exo v19 explication

➢ Différence entre le $reponse et $donnee
➢ Le <?php $reponse->closeCursor(); ?> ferme la connexion à la base (pour ne pas l'encombrer).

Le SQL: les fonctions avancées

- Imaginons que je souhaite obtenir uniquement la liste des jeux disponibles de la console « Nintendo 64 » et les trier par prix croissants.
- Il existe des fonctions pour cela:
  - WHERE
  - ORDER BY
  - LIMIT

Le SQL: les fonctions avancées

- Le WHERE
  - Permet de trier les données. Exemple: SELECT * FROM jeux_video WHERE possesseur='Patrick'
  - Exo: dans phpMyAdmin, n'afficher que les jeux sur la nintendo 64 (1)
- Le AND et le OR
  - Permet d'indiquer plusieurs conditions inclusive (AND) ou exclusive(OR)
  - Exemple: SELECT * FROM jeux_video WHERE possesseur='Patrick' AND prix < 20
  - Exo: Afficher les jeux à 4 joueurs ou plus sur PC (2)
  - Exo: Afficher les jeux dont le prix est inférieur à 10 euros ou que le nombre de joueurs est > 6 (3)

Le SQL: les fonctions avancées

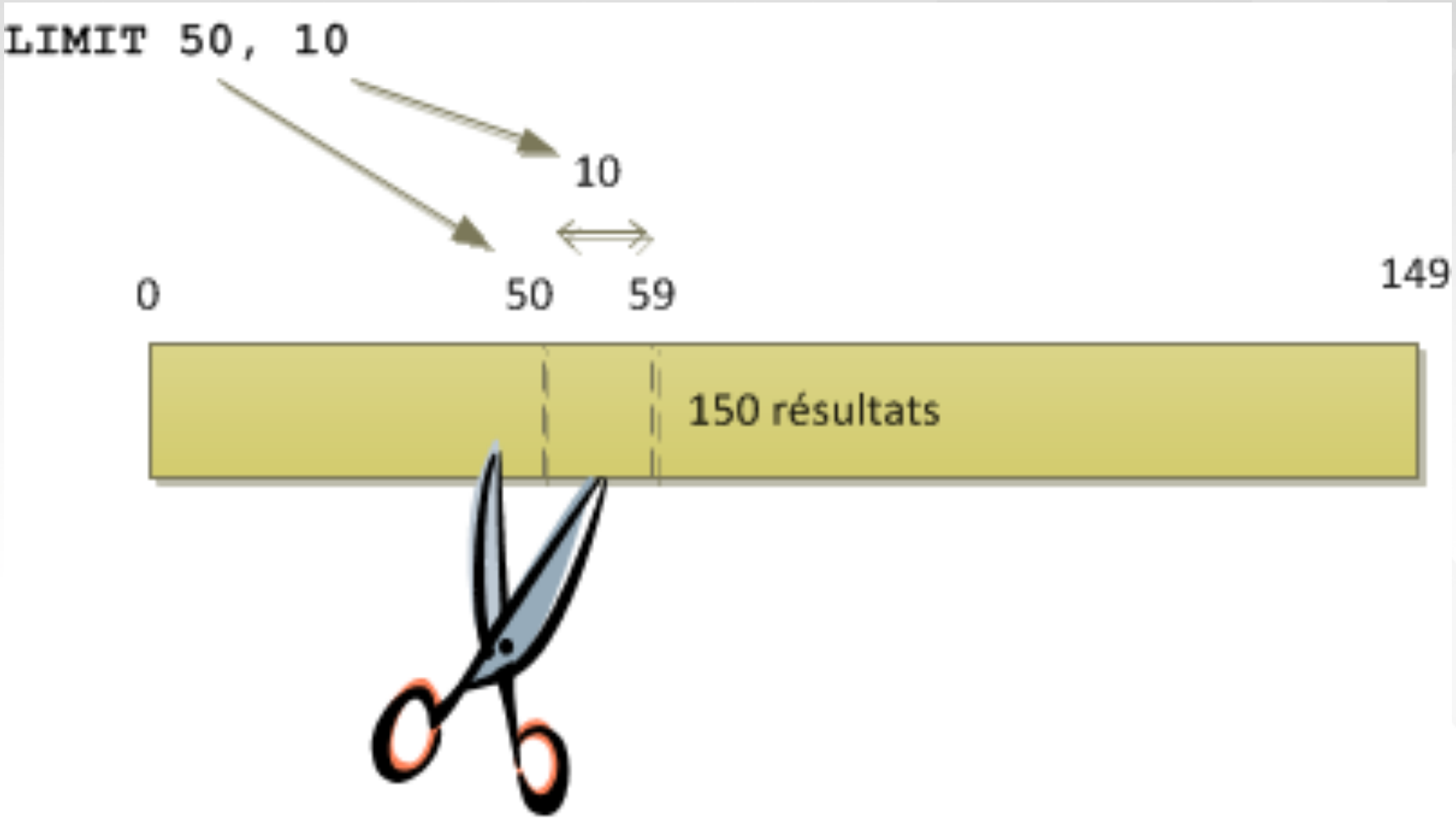
- Le ORDER BY
  - Permet d'ordonner les résultats
  - Exemple: Trier par prix: SELECT * FROM jeux_video ORDER BY prix
  - Exo: Trier par nombre de joueurs (1)
  - Il nous serait plus utile de trier dans l'ordre inverse.
- Pour trier par ordre inverse on utile DESC
  - Exemple: SELECT * FROM jeux_video ORDER BY prix DESC
  - Exo: faites le sur la dernière requête (2)

Le SQL: les fonctions avancées

- Le LIMIT
  - Permet de ne sélectionner qu'une partie des résultats
  - Exemple: SELECT * FROM jeux_video LIMIT 0, 20
  - Attention la syntaxe est un peu particulère:
    - LIMIT 0, 20 : affiche les vingt premières entrées ;
    - LIMIT 5, 10 : affiche de la sixième à la quinzième entrée ;
    - LIMIT 10, 2 : affiche la onzième et la douzième entrée.

Le SQL: les fonctions avancées

➢Le LIMIT



Le SQL: les fonctions avancées

- Le LIMIT
  - Exo: Afficher le nom, possesseur, console et prix des jeux videos ou la console est Xbox ou PS2 ordonnée par le prix décroissant et seulement les 10 premières entrées

Le SQL: retour au PHP

- L'utilisation des variables dans les requêtes
  - On pourrait faire comme cela:

```
<?php
$reponse = $bdd->query('SELECT nom FROM
jeux_video WHERE possesseur=\'Patrick\');
?>
```

  - Ou avec le possesseur passé en GET:

```
<?php
$reponse = $bdd->query('SELECT nom FROM
jeux_video WHERE possesseur=\'' .
$_GET['possesseur'] . '\');
?>
```

  - Mais cela ouvrirait la voie aux **injections SQL**

Le SQL: retour au PHP

➢ La bonne méthode est d'utiliser le prépare:

```
<?php
$req = $bdd->prepare('SELECT nom FROM jeux_video WHERE possesseur = ?');
$req->execute(array($_GET['possesseur']));
?>
```

➢ S'il y a plusieurs marqueurs, il faut indiquer les paramètres dans le bon ordre :

```
<?php
$req = $bdd->prepare('SELECT nom FROM jeux_video WHERE possesseur = ? AND prix <= ?');
$req->execute(array($_GET['possesseur'], $_GET[‘prix_max']));
?>
```

Le SQL: retour au PHP: Exo v20

➢ Construisez une page en php qui prenne en get le possesseur et le prix max et le prenne en compte dans la requête pour l'afficher

➢ (1) Afficher ensuite les jeux de Michel à moins de 10 euros

Le SQL: retour au PHP

➢ Mieux, on peut utiliser les marqueurs nominatifs:

```
WHERE possesseur = :possesseur AND prix <= :prixmax’
```

avec

```
execute(array('possesseur' => $_GET['possesseur'], 'prixmax' => $_GET[‘prix_max']));
```

➢ Avantage: les variables sont plus facilement identifiable. De plus, l’ordre dans lesquel on envoie les GET n’ont plus d’importance puisqu’ils sont marqué dans le array.

Le SQL: les erreurs

➢ Lorsqu'une erreur indique un problème au niveau du fetch, cela provient souvent de la requête SQL qui est mauvaise

➢ N'hésitez donc pas à faire la requête au préalable dans PhpMyAdmin

Le SQL: les autres fonctions

- INSERT
- UPDATE
- DELETE

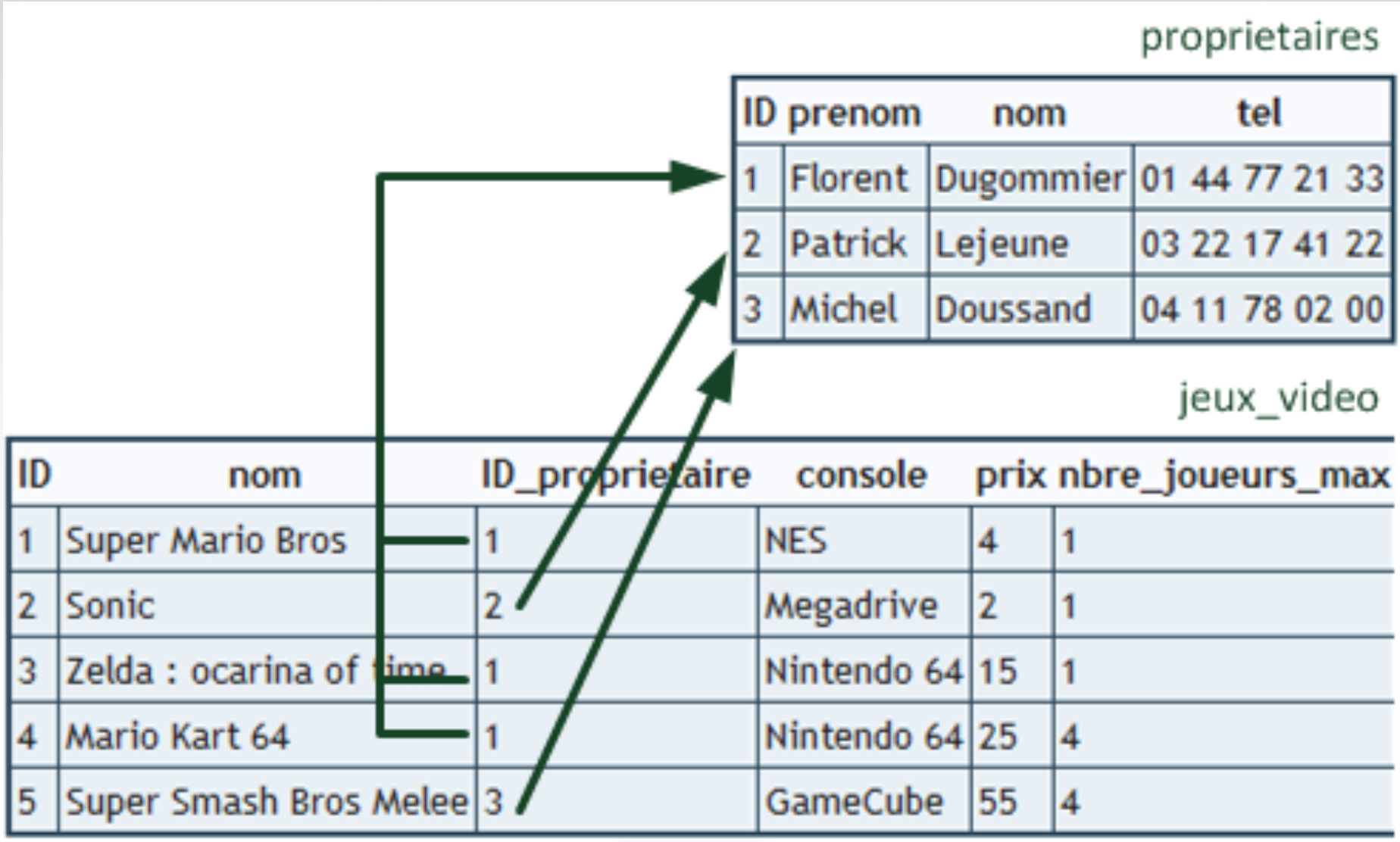
- Les fonctions scalaires: UPPER, LOWER, LENGTH, ROUND etc…
- Les fonctions d'agrégat: AVG, SUM, MAX, MIN, COUNT

- Les alias (le AS)
- Le GROUP BY
- Le HAVING
- Les formats: INT, VARCHAR, TEXT, TIME, DATETIME

Le SQL: BDD relation. et jointure

➢ Permet de mieux organiser les données
➢ Exemple: dans la table JV, plutôt que de répéter plusieurs fois les mêmes noms de possesseurs, on va créer une nouvelle table possesseur qu'on va lier à la table JV par l'intermédiaire d'un id « id-possesseur »
➢ C'est ce qu'on appelle un id secondaire, faisant de notre base une BDD relationnelle
➢ Pour pouvoir faire des requêtes, il va dès lors falloir faire des jointures

Le SQL: BDD relationnelles et joint



proprietaires

ID	prenom	nom	tel
1	Florent	Dugommier	01 44 77 21 33
2	Patrick	Lejeune	03 22 17 41 22
3	Michel	Doussand	04 11 78 02 00

jeux_video

ID	nom	ID_proprietaire	console	prix	nbre_joueurs_max
1	Super Mario Bros	1	NES	4	1
2	Sonic	2	Megadrive	2	1
3	Zelda : ocarina of time	1	Nintendo 64	15	1
4	Mario Kart 64	1	Nintendo 64	25	4
5	Super Smash Bros Melee	3	GameCube	55	4

Le SQL: BDD relationnelles et joint

➢ Exemple: pour afficher devant les consoles leur propriétaire on fait:

1. SELECT jeux_video.console, proprietaires.prenom
   FROM proprietaires, jeux_video
   WHERE jeux_video.ID_proprietaire = proprietaires.ID

Le SQL: jointures: exo

➢ On crée 2 bases relationnelles (sql fournit)
➢ Faites la jointure pour afficher devant le nom des jeux video le nom des propriétaires

L'UML

- Unified Modeling Language:
  - Modéliser les classes
  - Modéliser leurs interactions

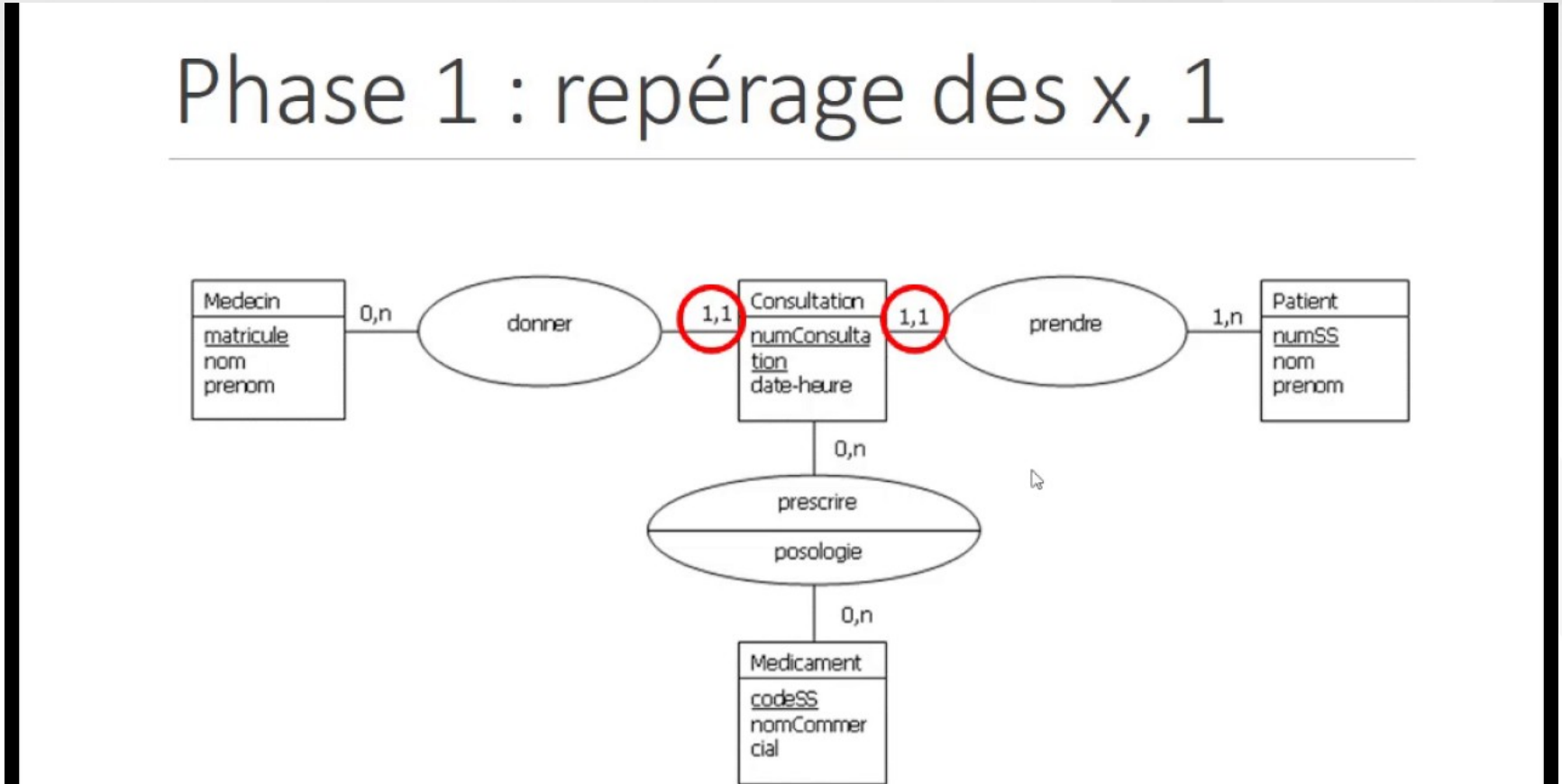
L’UML

➢ C’est ce qui permet de modéliser sa base de donnée. L’objectif est d’arriver à un MCD: Modèle Conceptuel de Donnée.

➢ Cela permet d’éviter d’oublier des morceaux de son application en modélisant la liste exhaustive des fonctionnalités attendues.

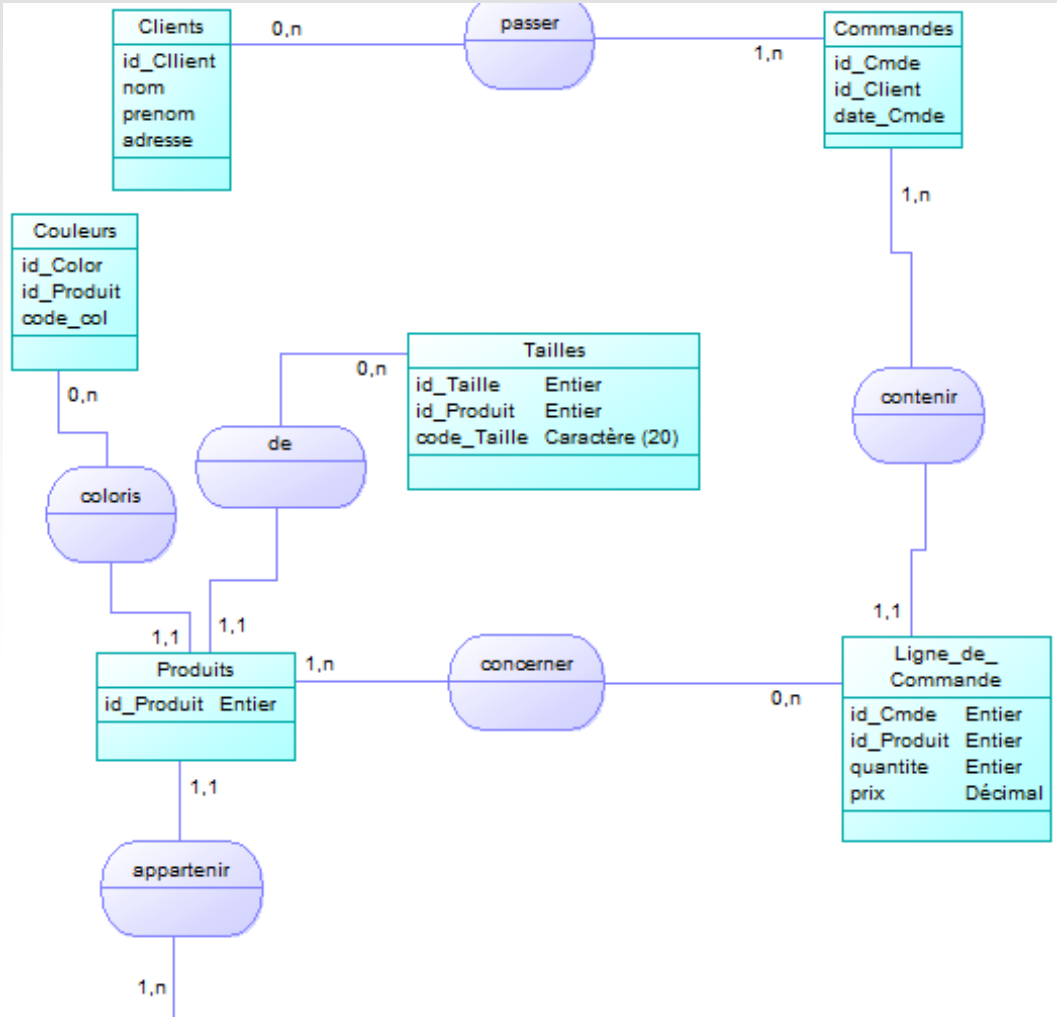
Exemple d'UML

➢ Ici, nous allons travailler que sur les diagrammes de classe, ceux utile pour le développement
➢ Bien réparer les cardinalités: 0, 1, n



Exemple d'UML pour le commerce

➢ Avec des id secondaire



L’UML

➢ L’UML permet de mieux visualiser l'application, mieux visualiser les liens entre les données DONC **mieux penser l'app**

L’UML: langage ou méthodologie

➢ C’est une méthodologie qui ne servira pas que pour PHP mais aussi tout autre langage orienté objet: Java, C#, Objective C etc.

➢ Lorsque l'on débute en OO et que l'on trouve le concept abstrait, le passage à l'UML peut être un bon moyen de se représenter plus facilement le projet, changer son regard sur le projet

➢ Les diagrammes vont représenter les classes

Les namespaces

- Il ne doit pas y avoir 2 fois la même classe dans votre projet
- Pour éviter un plantage du système, il faut alors utiliser les namespaces
  - Fonctionne comme des dossiers

- Exo v21: en partant de l'ex 18, utilisez les namespaces en entête des class
  - Utilisez namespace repertoire\du\namespace; dans les managers
  - N'oubliez pas d'indiquer le chemin dans l'instanciation des class (dans le contrôler)
  - Attention, le PDO faisant parti du namespace global, on utilisera new \PDO

Aller plus loin: le f# Symfony

➢ Installez Composer, préalable à Synfony:
`php -r « eval(‘?>'.file_get_contents('http://getcomposer.org/installer'));"`

➢ Installez un projet symfony: composer create-project symfony/skeleton mooc-symfony4

➢ Lancez votre application: http://localhost:8888/symfony/mooc-symfony4/public/index.php