COURS INTRODUCTIF

PHP et bases de données

Comprendre le chemin des données, utiliser PDO et éviter les failles essentielles

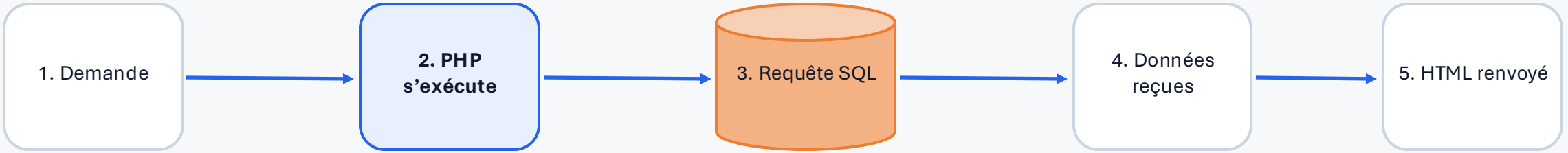


Objectif : être capable de lire et écrire un code simple, propre et sécurisé.

Requête HTTP → script → SQL → HTML

1. Qu’est-ce qu’une page Web dynamique ?

Le serveur fabrique la page au moment où elle est demandée.



Idée clé Le navigateur ne reçoit jamais le code PHP. Il reçoit uniquement le HTML produit par le serveur.

2. PDO : le pont entre PHP et la base

PDO fournit une interface commune pour PostgreSQL, MySQL, MariaDB et d'autres SGBD.

Pourquoi PDO ?

Une API unique
Des erreurs gérables par exceptions
Des requêtes préparées
Un code plus facile à maintenir

Vocabulaire

SGBD : logiciel qui gère la base
PDO : objet de connexion côté PHP
PDOStatement : requête préparée
DSN : informations de connexion

Réflexe débutant

1 connexion par script
Préparer avant d'exécuter
Séparer données et HTML
Ne jamais concaténer une saisie utilisateur dans SQL

PDO ne remplace pas SQL : il permet à PHP d'envoyer du SQL proprement.

3. Se connecter à PostgreSQL

La connexion est créée lorsque l'objet PDO est construit.

```php
<?php
$dsn = 'pgsql:host=localhost;dbname=bdCours';
$login = 'ali';
$mdp = 'mdp';

$bd = new PDO($dsn, $login, $mdp);
$bd->setAttribute(
    PDO::ATTR_ERRMODE,
    PDO::ERRMODE_EXCEPTION
);
```

DSN

pgsql = pilote
localhost = serveur
bdCours = nom de la base

Gestion des erreurs

Le mode exception arrête le traitement et signale clairement une erreur.

À retenir

La connexion est créée une seule fois, puis réutilisée pour toutes les requêtes.

4. Le cycle d’une requête PDO

Toujours le même enchaînement : préparer, fournir les valeurs, exécuter, récupérer.




```php
$req = $bd->prepare('SELECT nom FROM Personnes');
$req->execute();
$ligne = $req->fetch(PDO::FETCH_ASSOC);
```

5. Lire les résultats : fetch()

Chaque appel récupère une ligne, puis avance vers la suivante.

```php
$req = $bd->prepare(
    'SELECT Nom, Prenom FROM Personnes'
);
$req->execute();

while ($ligne = $req->fetch(PDO::FETCH_ASSOC)) {
    echo '<p>' . $ligne['Nom'] . '</p>';
}
```

PDO::FETCH_ASSOC

Résultat d’une ligne

['Nom' => 'Grappe',
 'Prenom' => 'Roland']

Fin des données

fetch() renvoie false : la boucle s’arrête.

6. fetch() ou fetchAll() ?

Le choix dépend surtout du volume de données et de l'usage prévu.

fetch()

Lit une ligne à la fois
Idéal pour parcourir un grand résultat
Consomme moins de mémoire
S'utilise souvent dans une boucle while

fetchAll()

Charge toutes les lignes dans un tableau
Pratique pour un petit résultat
Facile à transmettre à une vue
Attention aux gros volumes

Règle simple : peu de lignes → fetchAll() ; beaucoup de lignes → fetch() ou LIMIT.

7. Les marqueurs de place

Une valeur utilisateur doit rester une valeur, jamais devenir du code SQL.

```php
$req = $bd->prepare(
  'SELECT nom, couleur
   FROM fruit
   WHERE calories < :calories
   AND couleur = :couleur'
);

$req->bindValue(':calories', 150, PDO::PARAM_INT);
$req->bindValue(':couleur', 'rouge', PDO::PARAM_STR);
$req->execute();
```

Lisibilité

:calories indique clairement la donnée attendue.

Réutilisation

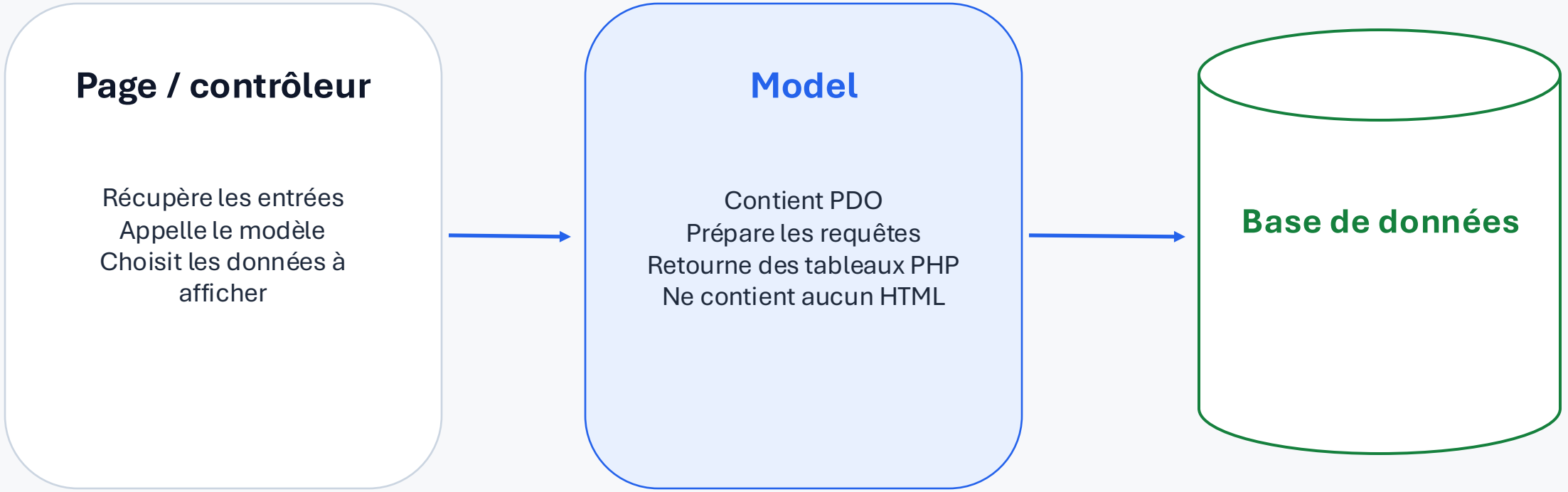
La même requête peut être exécutée avec d'autres valeurs.

Sécurité

La valeur n'est pas interprétée comme une partie de la requête SQL.

8. Organiser le code avec une classe Model

Le code d’accès aux données doit être séparé du code HTML.



Singleton = une seule instance du Model, donc une seule connexion

9. Injection SQL : le danger de la concaténation

Une saisie malveillante peut modifier la requête attendue.

À éviter

```
1 $sql = 'SELECT * FROM Commandes WHERE IdC = ' .
$_GET['id'];
2 $req = $bd->prepare($sql);
```

Pourquoi c’est dangereux ?

La saisie est collée au SQL. Elle peut donc changer la structure de la requête.

À faire

```
1 $req = $bd->prepare(
2   'SELECT * FROM Commandes WHERE IdC = :id'
3 );
4 $req->bindValue(':id', $_GET['id'],
PDO::PARAM_INT);
5 $req->execute();
```

Pourquoi c’est sûr ?

Le SGBD traite :id comme une donnée, pas comme du SQL.

10. XSS : protéger l’affichage HTML

Une donnée stockée en base peut contenir du HTML ou du JavaScript indésirable.

Donnée présente en base

```
1 <script>confirm("attaque")</script>
```



Le navigateur interprète la balise : le script peut s’exécuter.

Protection

```
1 echo htmlspecialchars(
2     $ligne['Prenom'],
3     ENT_QUOTES,
4     'UTF-8'
5 );
```

Les caractères spéciaux sont affichés comme du texte, pas exécutés.

11. Les mots de passe : hacher, puis vérifier

On ne stocke jamais un mot de passe en clair.

Inscription

```
$hash = password_hash(
  $motDePasse,
  PASSWORD_DEFAULT
);
```

Base de données

On stocke uniquement le hash. Deux mots de passe identiques peuvent produire des hash différents.

Connexion

```
password_verify(
  $motDePasse,
  $hashStocke
);
```

Important : un hash n'est pas un mot de passe "chiffré à déchiffrer". On vérifie une correspondance avec password_verify().

Synthèse : les 6 réflexes à retenir

Une checklist simple avant de rendre votre code.

1. **Une seule connexion PDO**
2. **prepare() avant execute()**
3. **Marqueurs pour toute saisie**
4. **FETCH_ASSOC pour des clés lisibles**
5. **htmlspecialchars() à l'affichage**
6. **password_hash() / password_verify()**

Le bon ordre mental : entrée → SQL sécurisé → données → HTML sécurisé.

Mini-exercice de vérification

Repérez les problèmes avant de proposer une correction.

```php
$req = $bd->prepare(
  "SELECT * FROM Personnes WHERE Nom = '" . $_GET['nom'] . "'"
);
$req->execute();
$personnes = $req->fetchAll();

foreach ($personnes as $p) {
    echo '<p>' . $p['Nom'] . ' ' . $p['Prenom'] . '</p>';
}
```

Question 1

Où se trouve le risque d'injection SQL ?

Question 2

Où se trouve le risque XSS ?