

# M3104

PHP

Développement Web côté serveur

# Notions abordées

- Web, HTML, environnement de développement
- PHP, bases du langage
- Accès aux BD MySQL
- Formulaires
- Sessions, Cookies & Cie
- Expressions Régulières

## Côté client

- Javascript
- CSS
- AJAX

# Bases de données

- Utiliser un gestionnaire de base de données
- Stocker et accéder aux données
- Considérations autour de la sécurité

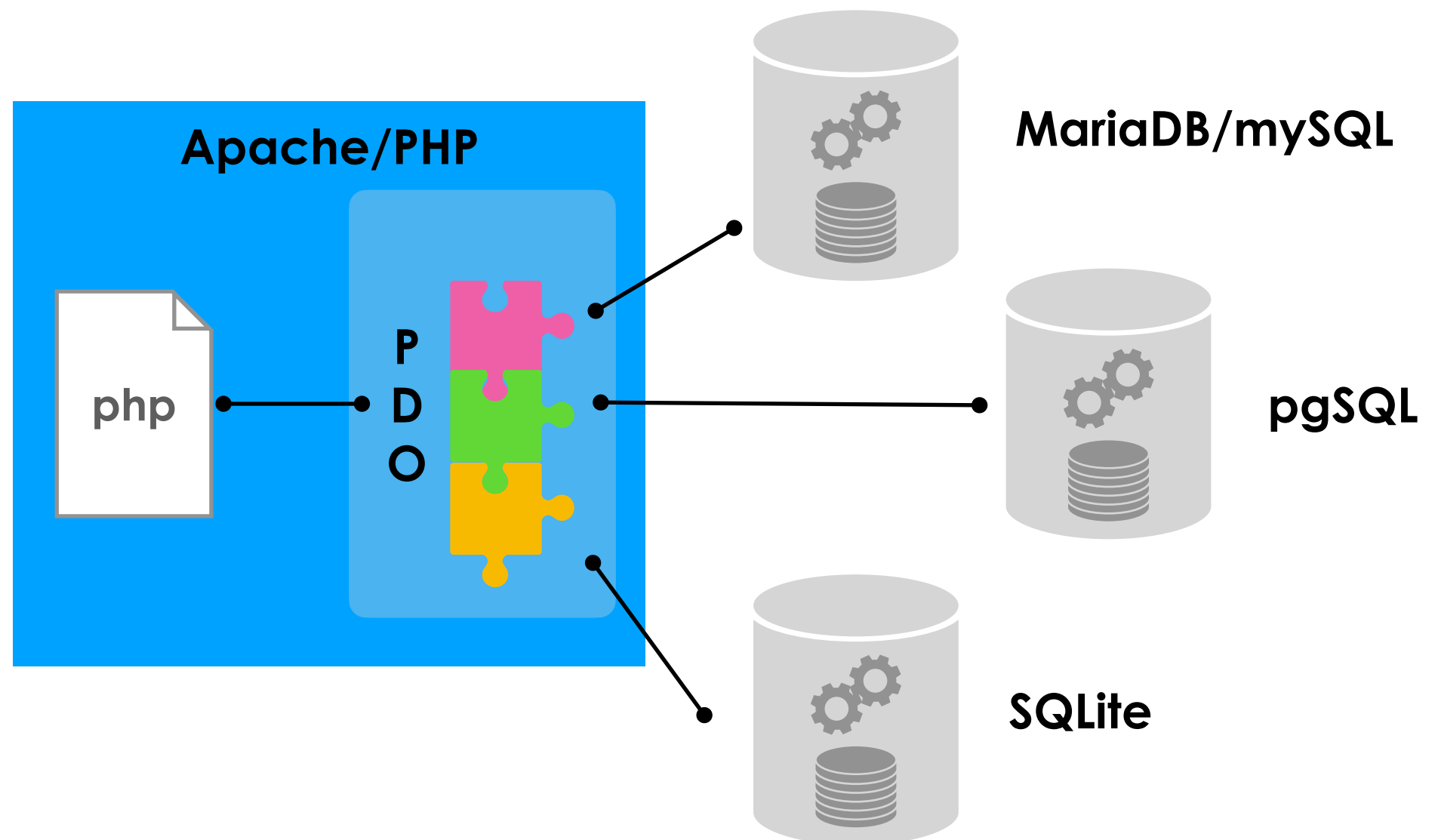
# PDO

## PHP Data Object

- PDO est une **couche d'abstraction** de la base de données
- Elle permet d'accéder d'une manière unique à plusieurs types de SGBD supportés.
- Une application PHP basée sur PDO est donc relativement indépendante de l'architecture SGBD utilisée (MySQL, PostgreSQL, Oracle, SQLite...).

# PDO

## Principe



# PDO

## Initialisation

- La connexion et la sélection de la base de données s'effectue en créant **une instance de l'objet PDO**.
- Les paramètres passés au constructeur décrivent :
  - le type de SGBD utilisé (mysql, oracle, etc), l'hôte, le port et la base à ouvrir
  - les identifiant et mot de passe

# PDO initialisation

```
<?php
$db = new PDO(
    'mysql:host=localhost;
    port=3306;dbname=MaBase',
    'user01',
    'pass01'
);
?>
```

# PDO

## requêtes

- A partir de l'instance PDO, plusieurs méthodes permettent d'envoyer des requêtes vers le serveur
- Trois types de requêtes peuvent être lancées :
  - **requête** sans résultat attendu
  - **requêtes** avec résultat attendu
  - **requêtes préparées** avec ou sans résultat



# PDO

## exécution sans retour

- il s'agit d'envoyer une requête au serveur, sans attendre de résultat, typiquement :
  - insertion dans la base
  - suppression
  - commande de configuration

# PDO

## exécution sans retour

```
<?php
```

```
//demande au serveur de communiquer en UTF8
```

```
$db ->exec( "SET NAMES UTF8 " );
```

```
$db ->exec( "DELETE FROM dbtest  
WHERE u=5 limit 1" );
```



```
?>
```

- la notation  $\rightarrow$  correspond à l'opérateur d'accès aux attributs et méthodes d'une instance en PHP (notation . en java)

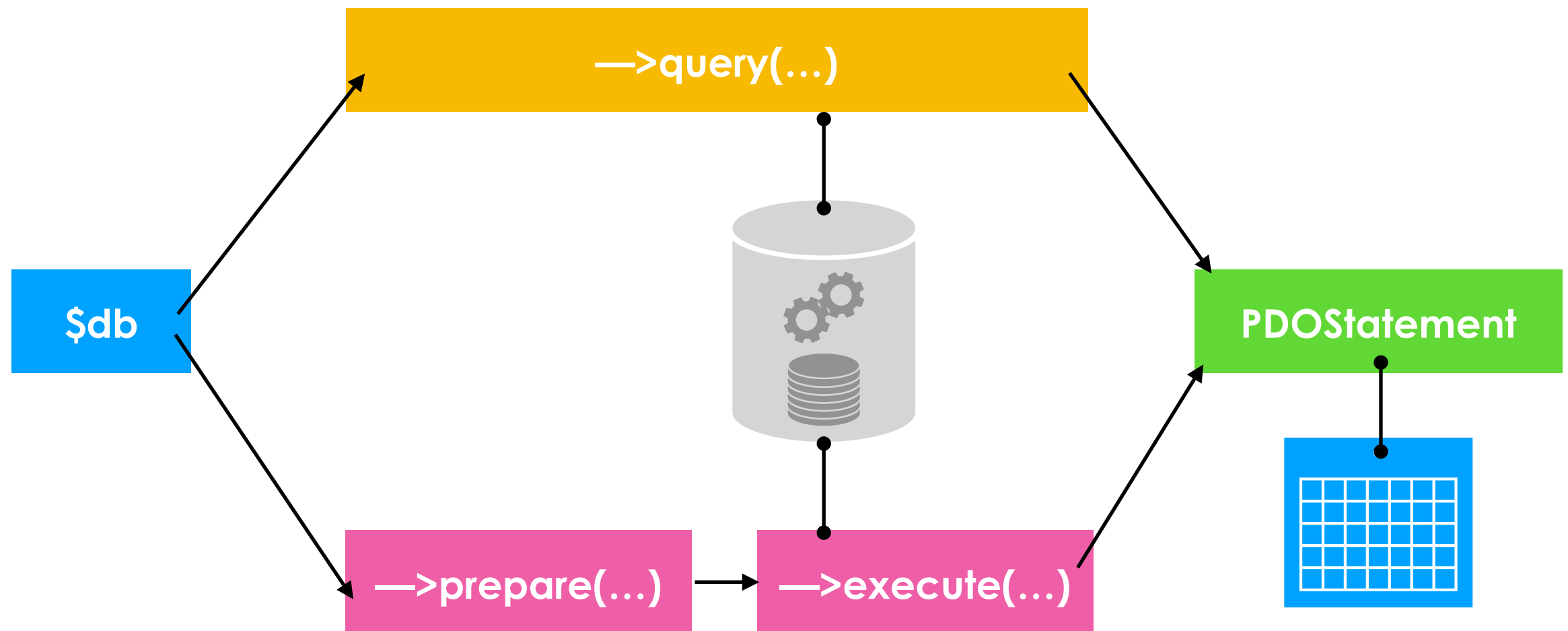
# PDO

## exécution avec retour

- il s'agit d'envoyer une requête au serveur, et attendre un jeu de résultats, typiquement une sélection dans la base
- le jeu de résultat est retourné sous la forme d'un objet **PDOStatement**
- les requêtes peuvent être **directes** ou **préparées**

# PDO

## Jeu de résultats



# PDO

## requêtes directes

- on les utilise lorsque le code SQL n'inclut pas de paramètres « injectés », c'est à dire qu'il est écrit en brut, non modifiable

```
<?php
```

```
//Exécution d'une requête figée
```

```
$stmt = $db->query("select * from genres  
                    where nom = 'Jazz'");
```

```
?>
```

# PDO

## requêtes préparées

- on les utilise lorsque les requêtes ont des valeurs injectées et/ou lorsque la requête est susceptible d'être appelée plusieurs fois avec des valeurs différentes
- Les requêtes préparées s'effectuent en deux temps :
  - **préparation de la requête** : cette phase analyse et 'compile' la requête pour optimiser les invocations ultérieures. Le résultat est un objet **PDOStatement**
  - **exécution de la requête** avec d'éventuels paramètres : cette phase exécute la requête précédemment analysée, et y insère les arguments s'ils existent.

# PDO

## requêtes préparées

```
<?php
//définition de la requête et des paramètres à injecter
$stmt = $db->prepare("insert into categorie values(:id,:nom)");

//association des paramètres et de leur valeur

$stmt ->execute(array(' :id'=>'1', ' :nom'=>'Jazz' ));

$stmt ->execute(array(' :id'=>'2', ' :nom'=>'Musette' ));

$stmt ->execute(array(' :id'=>'3', ' :nom'=>'Metal' ));

?>
```

# PDO

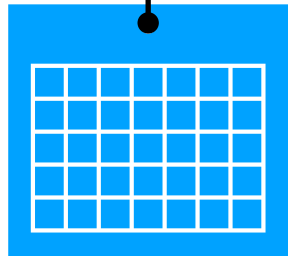
## requêtes préparées

```
<?php
//définition de la requête et des paramètres à injecter
$stmt = $db->prepare("select * from categorie where nom = :nom");
```

```
//association des paramètres et de leur valeur
```

```
$stmt->execute(array(' :nom' => 'Jazz' ));
```

?>



- l'objet **PDOStatement** récupéré contient le jeu de résultats (qui peut être vide)
- il contient également d'autres informations sur l'exécution de la requête : taille du jeu d'enregistrements, erreurs éventuelles



# PDO

## Informations sur les résultats

**\$stmt** —> **rowCount**( ) : nombre de résultats disponibles

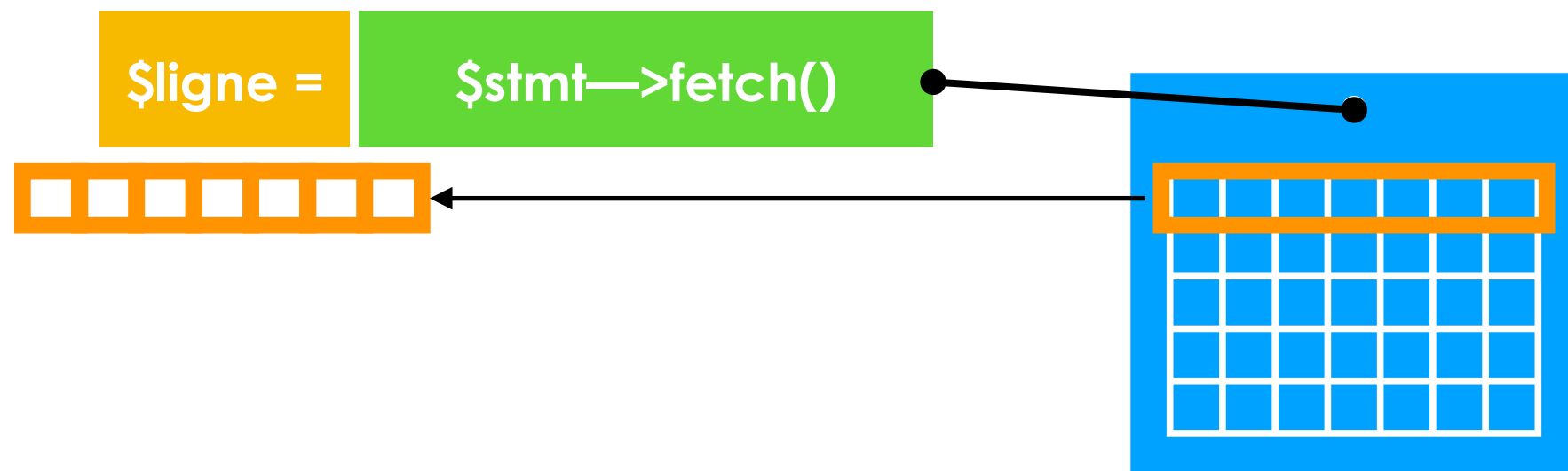
**\$stmt** —> **errorCode**( ) : code d'erreur de l'exécution précédente

**\$stmt** —> **errorInfo**( ) : tableau d'erreurs contenant le code et le message de débogage de l'exécution précédente

# PDO

## Récupération des résultats

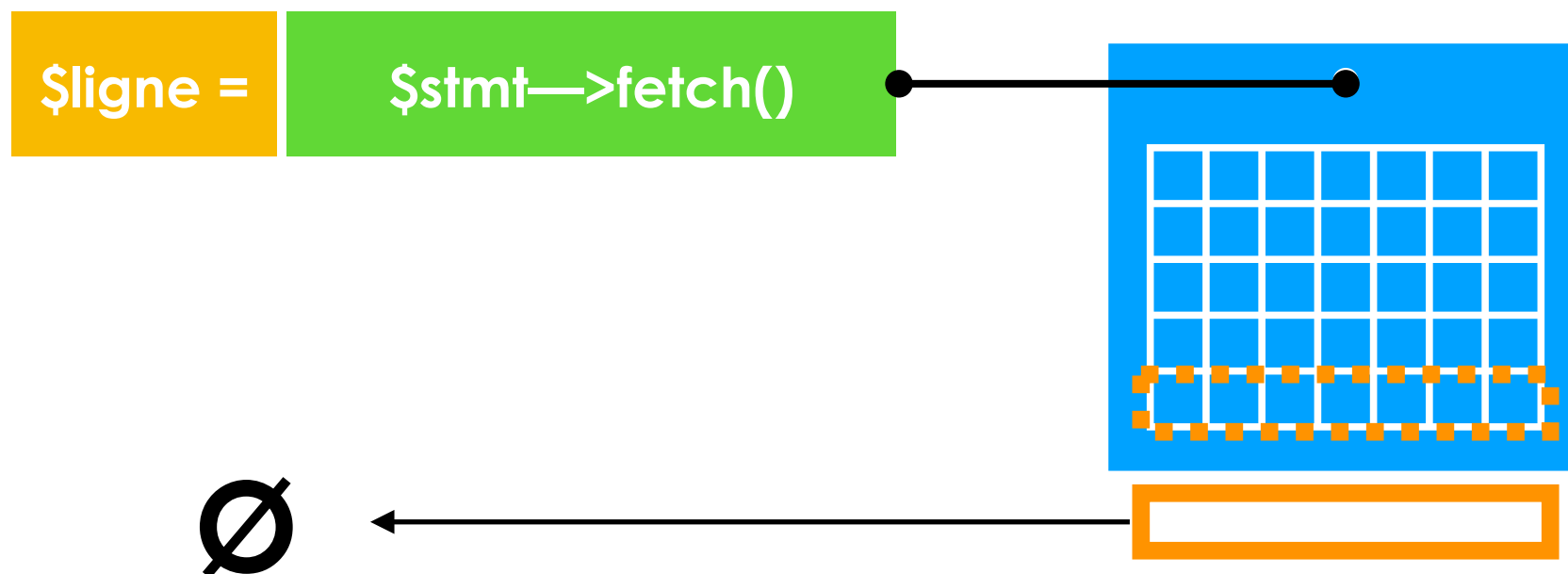
- Le jeu de résultats est récupéré en parcourant ligne à ligne les enregistrements
- La méthode **fetch()** extrait la prochaine ligne disponible dans le jeu de résultats



# PDO

## Récupération des résultats

- On parcourt les lignes jusqu'à ce que le dernier enregistrement soit atteint



# PDO

## Récupération des résultats

- La ligne peut être récupérée selon différents formats, en fonction du paramètre passé à **fetch**
  - **tableau indicé** (0...n) où chaque indice est le numéro de la colonne dans la requête
  - **tableau associatif** (champ1, ..., champ.n) où chaque clé est le nom de la colonne dans la requête
  - **objet** (—>attr1, ..., —>attrn) où chaque attribut est le nom de la colonne dans la requête

# PDO

## Parcours des résultats

- Pour itérer sur la liste d'enregistrements, on peut utiliser :
  - une boucle **while** terminée sur une ligne *nulle*
  - une boucle **foreach** sur l'objet **PDOStatement** (ce dernier implémente l'interface *traversable* dont se sert foreach)

# PDO

## Parcours des résultats

- while avec tableau indicé

<?php

```
$db = new PDO(...);
```

```
$stmt = $db -> query("select * from genres  
                        where nom = 'Jazz'");
```

```
while ( $ligne = $stmt -> fetch(PDO::FETCH_NUM) ) {  
  
    $data = $ligne[0] . " " . $ligne[1] . "<br />";  
    print $data;  
  
}
```

?>

# PDO

## Parcours des résultats

- while avec tableau associatif

<?php

```
$db = new PDO(...);
```

```
$stmt = $db -> query("select * from genres  
                        where nom = 'Jazz'");
```

```
while ( $ligne = $stmt -> fetch(PDO::FETCH_ASSOC) ) {  
  
    $data = $ligne['id'] . " " . $ligne['nom'] . "<br />";  
    print $data;  
  
}
```

?>

# PDO

## Parcours des résultats

- foreach avec tableau indicé

```
<?php
...
foreach( $stmt as $ligne ) {
    $data = $ligne[0] . " " . $ligne[1] . "<br />";
    print $data;
}
?>
```



# PDO

## Gestion des erreurs

- Les erreurs peuvent être manuellement (cf. **errorInfo( )**, **errorCode( )** )
- PDO peut être configuré pour masquer les erreurs (comportement par défaut), générer des Warning PHP ou bien lancer des exceptions.

```
<?php
//configuration
//lancement d'exception si erreur
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

```
//aucun message
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);
```

```
?>
```

# PDO

## Gestion des erreurs

- Les exceptions sont rattrapables par une fonction dédiée (mécanisme similaire en Java et VB)

```
<?php
function ma_gestion_d_exception($exception) {
    //gestion des erreurs
    //affichage d'un message
    //redirection...
    //log système, mail administrateur
}

set_exception_handler('ma_gestion_d_exception');
?>
```

# BD

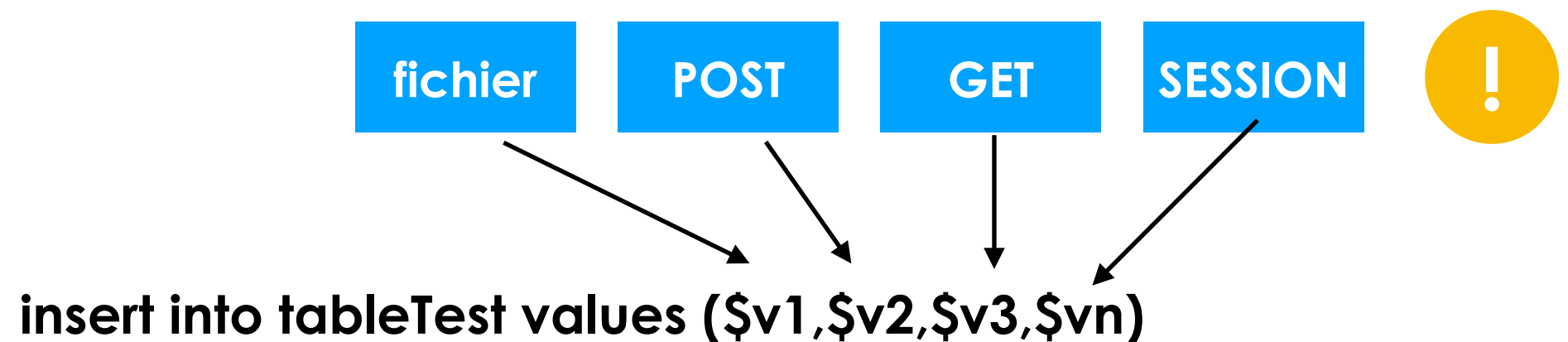
## Considérations autour de la sécurité

- Les bases contiennent potentiellement des données sensibles
- Elles sont des cibles privilégiées d'attaques : soit pour récupérer les données, soit pour les corrompre, soit pour les supprimer

# BD

## Considérations autour de la sécurité

- Il est **essentiel** de sécuriser le code, notamment pour se prémunir des **injections**
- chaque donnée provenant de **l'extérieur** et utilisée dans une requête est potentiellement dangereuse



# BD

## Considérations autour de la sécurité



# BD

## Considérations autour de la sécurité

- exemple : injection de code SQL à travers un formulaire

Votre identifiant : `admin' OR 1==1#`

POST[login]

POST[pass]

`select * from utilisateurs where login = '$login' and pass='$pass'`

`select * from utilisateurs where login = 'admin' OR 1==1# ' and pass='$pass'`

# BD

## Considérations autour de la sécurité

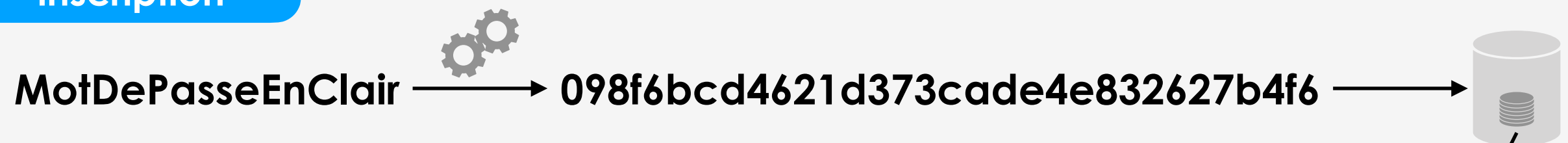
- les **mots de passe** utilisateur : ils représentent une cible privilégiée des attaques
- ils doivent être **chiffrés** dans la base, jamais en clair !
- le chiffrement consiste à générer une **empreinte** du mot de passe en clair

# BD

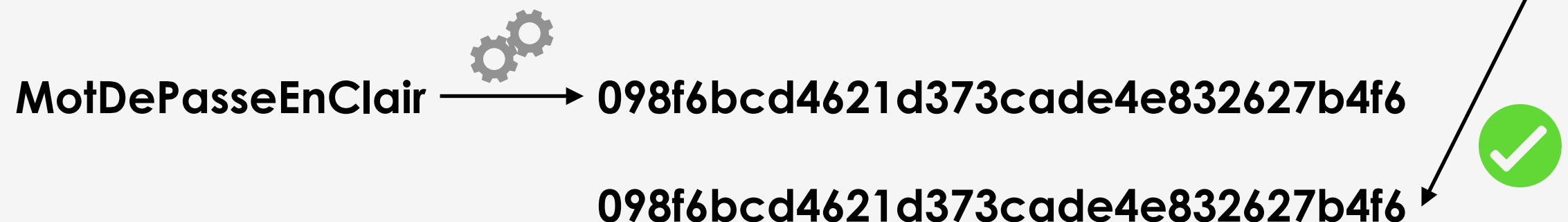
## Considérations autour de la sécurité

- pour vérifier un mot de passe d'utilisateur, on compare les versions chiffrées (les empreintes)

### inscription



### authentification

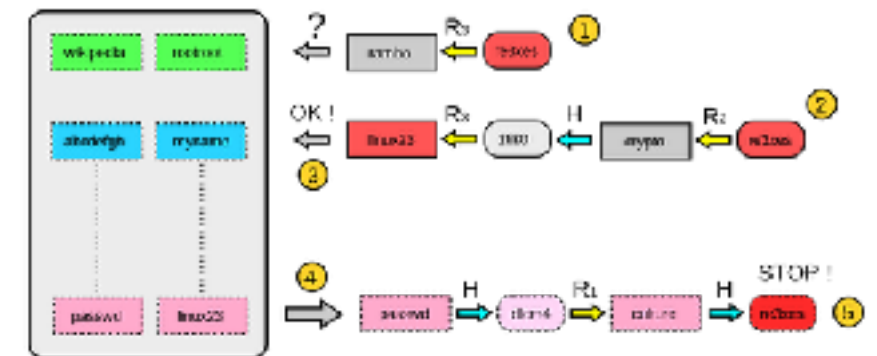
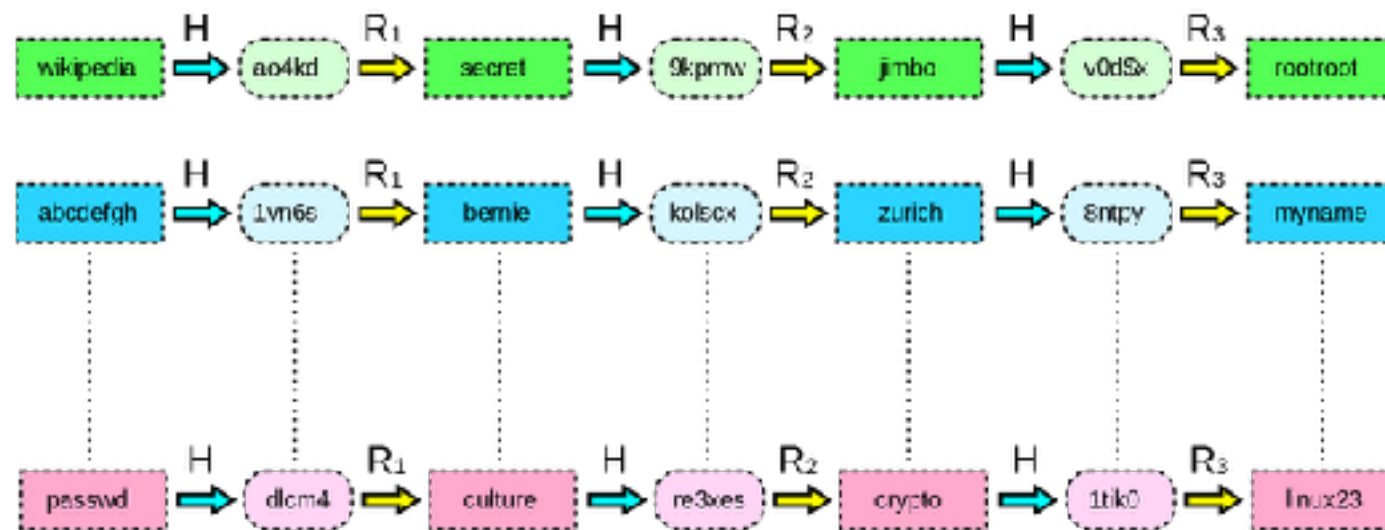




# BD

## Considérations autour de la sécurité

- privilégier un algorithme de chiffrement **aléatoire**, afin d'éviter des attaques de type brute-force ou *Rainbow table*



source wikipedia

# BD

## Chiffrement « sans sel »

- exemple : **md5**, longtemps utilisé, n'est plus assez robuste
- génère toujours la même **empreinte**

**md5(test) = 098f6bcd4621d373cade4e832627b4f6**

**md5(test) = 098f6bcd4621d373cade4e832627b4f6**

**md5(test) = 098f6bcd4621d373cade4e832627b4f6**

- si l'**empreinte** est connue (déjà calculé dans une table), on retrouve facilement le mot de passe en clair

# BD

## Chiffrement « avec sel »

- actuellement utilisé en PHP, la fonction **crypt** génère une empreinte **différente** à chaque appel, grâce à un **sel** (aléatoire) incorporé dans la chaîne générée

```
crypt(test) = $2y$12$1anH5RDH1GU3wj7q/JzS9ePaQGatUxxqdZHB.kfTbFTcHb..WCFau  
crypt(test) = $2y$12$4NSBBgP0ZQCkvSFU0AWm1.e/TMprPBaNNvyaMyNXwNLOV4kLSwL1.  
crypt(test) = $2y$12$qmgnlb4Y4dpaqot4eLgwOce0onlgGNUAHUHg/YcOzSFrdqauD9mm
```

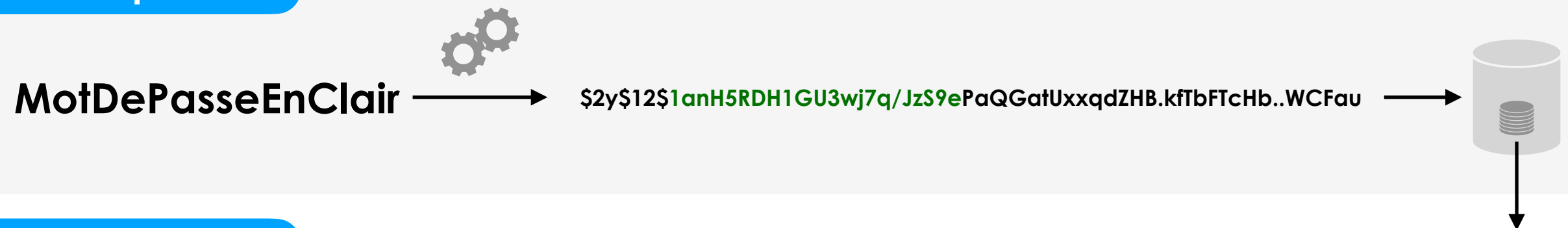
- un appel à crypt avec le même sel génère la même empreinte

```
crypt(test,1anH5RDH1GU3wj7q/JzS9e) =  
$2y$12$1anH5RDH1GU3wj7q/JzS9ePaQGatUxxqdZHB.kfTbFTcHb..WCFau
```

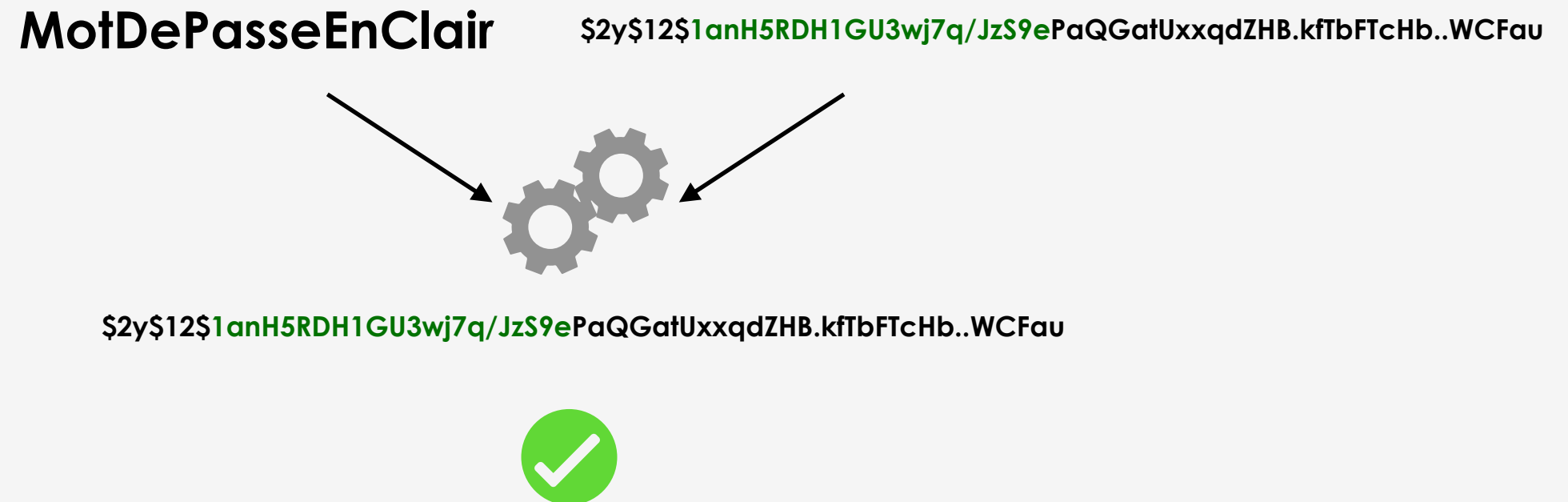
# BD

## Chiffrement « avec sel »

inscription



authentification



# BD

# Authentication

- **password\_hash** et **password\_verify** simplifient le chiffrement et la vérification des mots de passe, ils utilisent actuellement utilisent l'algorithme **blowfish** par défaut

- chiffrement :

```
$motdepassechiffre = password_hash("toto", PASSWORD_DEFAULT);
```

```
—> $2y$10$it5s/tbVCAnlZkJ/4cLT5.mQnDmAf2B7rT/ee.Lq85RiqxNurF2D2
```

- vérification

```
$resultat = password_verify ( "toto" , $motdepassechiffre );
```

```
—> true
```