ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Partie 4 : PHP

Filière : Ingénierie Informatique et Réseaux
ECOLE MAROCAINE DES SCIENCES DE L'INGÉNIEUR – EMSI
Honoris United Universities

Niveau : 3°IIR

Pr. NAZIH Marouane

La persistance

La persistance

- Tableaux associatifs
- Modularité: inclusion du code
- Bases de données
  - ➔ Accès aux BDD avec PDO
  - ➔ La lecture
  - ➔ L'écriture
  - ➔ Les transactions

Tableaux

Tableaux indexé

PHP (Modèle)

```
$name = array();                          # Create
$name = array(value0, ..., valueN);
$name[index]                              # Get element value
$name[index] = value;                     # Set element value
$name[] = value;                          # Append PHP
```

PHP (Exemple)

```
$a = array();                             # Empty array (length 0)
$a[0] = 23;                               # Stores 23 at index 0 (length 1)
$a2 = array("some", "strings", "in", "an", "array");
$a2[] = "Ooh!";                           # Add string to end (at index 5)
```

NB : Un tableau peut contenir des éléments de différents types.

Tableaux associatifs

PHP (Modèle)

```
$name = array();                          # Create
$name = array(key0 => value0, ..., keyN => valueN);
$name[key]                                # Get element value
$name[key] = value;                       # Set element value
$name[newKey] = value;                    # Append PHP
```

PHP (Exemple)

```
$p = array();                 # Empty array (length 0)
$p["firstmane"] = "Allan";     # Stores "Allan" at key "firstname"
$p["lastname"] = "Turing";    # Stores "Turing" at key "lastname"
```

Les indices sont des strings et les clefs sont **toujours** des chaînes de caractères.

La boucle foreach

PHP (Modèle)

```
foreach ($array as $key => $value) {
   statements ;
}
```

PHP (Exemple)

```
$dict = array [ "oeuf" => "egg", "pomme" => "apple", "pain" => "bread"
];
foreach ($dict as $fr => $en) {
  print "$fr se dit $en" en anglais;
}
```

NB : Elle Permet d'itérer sur les couples (clef, valeur) d'un tableau associatif.

Modularité

Modularité: inclusion du code

PHP (Modèle)

```
include(filename);
```

PHP (Exemple)

```
include("header.html");
include("shared-code.php");
```

- Insère le contenu du fichier paramètre dans le fichier courant.
- Permet une certaine modularité.
- Permet de partager des fonctions utiles dans plusieurs scripts.
- Voir aussi : include_once, require, et require_once.

Bases de données

Accès aux BDD avec PDO

- PDO : PHP Data Objects
- Extension PHP fournissant une interface pour accéder à une base de données
- Fournit une interface d'abstraction pour l'accès aux données
- Ne fournit PAS une abstraction de base de données
  - SQL spécifique au moteur du SGBD
  - Fonctionnalités présentes / absentes
- Interface orientée objet
- Compatible avec plusieurs SGBD et dispose de différents drivers, dont : MySQL, Oracle, PostgreSQL, SQLite, ODBC et DB2, MS SQL Server, Informix, Firebird, IBM ...

MySQLi vs. PDO

MySQLi (MySQL Improved)
Interface Orientée Objet (OO) et Procédurale

Offre une interface orientée objet (OO) et une approche procédurale.

Les fonctions peuvent être appelées de manière procédurale ou à travers des objets.
Prise en charge des fonctionnalités spécifiques à MySQL

Lié spécifiquement à MySQL, ce qui peut rendre le code moins portable entre différentes bases de données.

PDO (PHP Data Objects)
Interface Orientée Objet uniquement

Fournit une interface uniquement orientée objet.

Offre une abstraction de la base de données, rendant le code plus portable entre différentes bases de données.
Support de plusieurs bases de données

Compatible avec plusieurs types de bases de données, dont MySQL, Oracle, PostgreSQL, SQLite, etc.

Connexion PDO – MySQL

PHP (Modèle)

```php
$db = new PDO($dsn, $username=null, $password=null, $options=null);
```

PHP (Exemple MySQL)

```php
$db = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
```

Les informations de la basse de données (host + nom de BDD + utilisateur + mot de passe) ainsi que la commande de connexion ci-dessus sont généralement isolées dans un fichier à part pour être incluses dans tous les scripts où on va avoir besoin de faire des opération sur la BDD.

Gestion des erreurs – Connexion

PHP (Exemple MySQL)

```php
try{
  $db = new PDO("mysql:host=localhost;dbname=test", $user,
$pass);
} catch(PDOException $e){
  die( "Erreur: ".$e->getMessage()."<br>");
}
```

La fonction die(...) permet d'envoyer un texte au client (comme 'echo') et d'arrêter l'interprétation du script.

Gestion des erreurs – Hormis connexion

PHP (Exemple)

```php
$db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
```

- PDO::ERRMODE_SILENT (par défaut)
  - Mode silencieux, mise en place d'un code d'erreur
  - $db->errorCode() et $db->errorInfo()
- PDO::ERRMODE_WARNING
  - Émission d'une erreur de type E_WARNING
- PDO::ERRMODE_EXCEPTION
  - Déclanchement d'une exception de type PDOException
  - La technique la plus propre pour gérer les erreurs

Opérations de base : query(...)

PHP (Exemple)

```php
$sql="SELECT id,nom,prenom,email FROM client";
$rs=$db->query($sql);
echo "<table>";
while ($rec=$rs->fetch(PDO::FETCH_ASSOC)) {
  echo "<tr data-id=\"".$rec["id"]."\"><td>".$rec["nom"].
" "
.$rec["prenom"]."</td><td>".$rec["email"]."</td></tr>";
}
echo "</table>";
```

Nous avons séléctionné tous les enregistrements dans la table "client" et nous les avons affichés sous forme de tableau HTML.

- La méthde fetch permet de récupérer un enregistrement et pointer sur le suivant.

Opérations de base : fetch(...)

- On peut récupérer l'enregistrement courant dans différents formats :
  - Tableau associatif : PDO::FETCH_ASSOC
  - Tableau indexé : PDO::FETCH_NUM
  - Les deux à la fois : PDO::FETCH_BOTH (par defaut)
  - Objet : PDO::FETCH_OBJ
  - ...
- La méthde fetchAll() permet de récupérer tous les enregistrements d'un coups.

PHP (Exemple)

```
$allRecs = $rs->fetchAll();
foreach ( $allRecs as $rec ) {
  echo "<tr data-id=\"".$rec["id"]."\"><td>".$rec["nom"]. "
" .$rec["prenom"]."</td><td>".$rec["email"]."</td></tr>";
}
```

Opérations de base : prepare(...)

Quand on veut lancer plusieurs fois la même requete en changeant seulement quelques paramètres, il est préférable de la "préparer".

- C'est plus "propre"
- C'est plus performant
- Inclut une protection contre les injections SQL

Pour lancer une requête "préparée" il faut :

1. La préparer avec
   `prepare("requete sql contenant des marqueurs '?' ou ':marqueur'")`
2. L'executer avec la méthode
   `execute([tableau, de, paramètres])`

Opérations de base : prepare(...)

PHP (Exemple)

```php
$sql="SELECT nom,calories FROM fruit WHERE calories < ?
AND couleur = ?";
$rs=$db->prepare($sql);
$rs->execute([100,"rouge"]);
$allRed = $rs->fetchAll();
$rs->execute([120,"jaune"]);
$allYellow = $rs->fetchAll();
$rs->execute([90,"vert"]);
$allGreen = $rs->fetchAll();
```

Les requêtes "executées" après avoir été "préparées" sont plus rapides que celles lancées par un simple "query".

Opérations de base : exec(...)

La méthode PDO `exec("requête SQL")` permet d'executer une requête SQL en écriture et retourner le nombre d'enregistrements affectés.

PHP (prototype)

```
public PDO::exec(string $statement): int|false
```

PHP (Exemple)

```
$nb = $db->exec("DELETE FROM fruit WHERE calories > 200");
$nb = $db->exec("UPDATE fruit SET calories='160' WHERE
nom='avocat'");
$nb = $db->exec("INSERT INTO client (nom, prenom, email)
VALUES ('DOE','John','john@doe.com')");
$clientId = $db->lastInsertId();
```

Opérations de base

- La méthode PDO `lastInsertId()` donne l'ID du dernier enregistrement inséré. Elle est très utile quand la table dispose d'une clé primaire numérique de type AUTO_INCREMENT.
- Les requêtes d'écriture (INSERT, UPDATE, DELETE ...) peuvent elles aussi être "préparées" et "executées"

PHP (Exemple)

```php
$sql = "DELETE FROM fruit WHERE calories > ? AND couleur
<> ?";
$rs = $db->prepare($sql);
$rs->execute([100,"rouge"]);
```

Les transactions

- Une transaction consiste à confirmer plusieurs requêtes SQL à la fois. On utilise donc la methode PDO `commit()`
- Si on a un problème au cours du traitement de nos requêtes, on peut annuler toutes les requêtes précédantes de la transaction en cours avec `rollBack()`
- Une transaction commance toujours par `beginTransaction()`
- Les transactions sont importantes dans les cas où l'on ne doit rien faire si une requete parmi plusieurs retourne un résultat non désiré
- Par exemple: dans une BDD MySQL de type MyISAM, quand je supprime enregistrement, je dois supprimer toutes ses relations dans les autres tables (clès étrangères pointant vers cet enregistrement). Si l'une de ces requêtes n'aboutit pas, on doit annuler toutes les requêtes de suppressions précédantes.

Les transactions

```
// Début de la transaction
$connexion->begin_transaction();

try {
    // Opération 1 : Insérer un utilisateur
    $utilisateurNom = "John Doe";
    $utilisateurEmail = "john.doe@example.com";

    $requeteUtilisateur = "INSERT INTO utilisateurs (nom, email) VALUES ('$utilisateurNom', '$utilisateurEmail')";
    $connexion->query($requeteUtilisateur);

    // Opération 2 : Insérer une commande
    $montantCommande = 100.00;

    $requeteCommande = "INSERT INTO commandes (utilisateur_id, montant) VALUES ('$connexion->insert_id',
'$montantCommande')";
    $connexion->query($requeteCommande);

    // Valider la transaction
    $connexion->commit();
    echo "Transaction réussie !";
} catch (Exception $e) {
    // En cas d'erreur, annuler la transaction
    $connexion->rollback();
    echo "Erreur de transaction : " . $e->getMessage();
}
```

Fin de Partie 3 : PHP