



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



# Langage de Script PHP

---

## Chapitre 2

Pr. Zainab OUFQIR

Z.Oufkir@emsi.ma



# Passage de données (URL et formulaire)

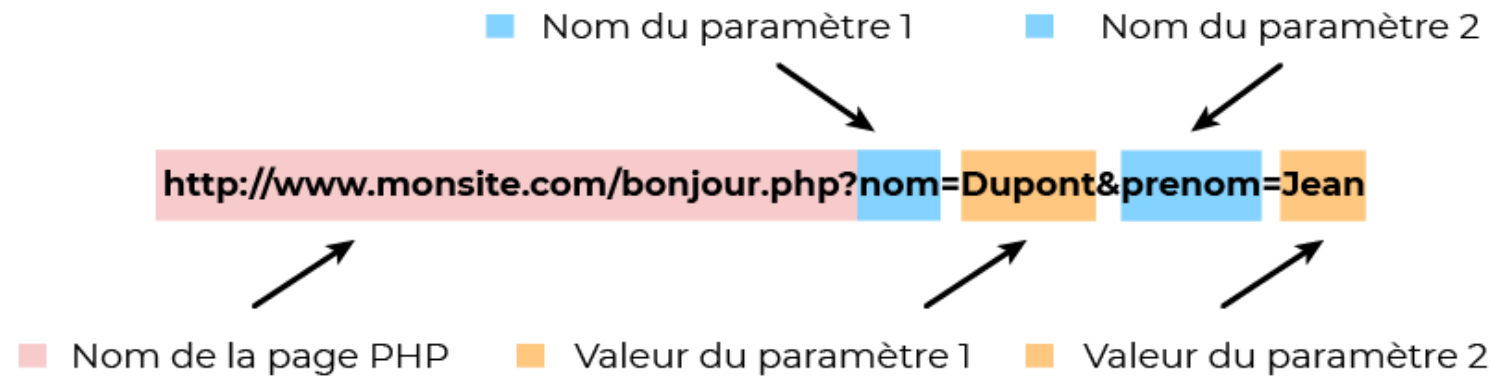
---

- **URL** signifie **U**niform **R**esource **L**ocator. C'est en fait **une adresse sur le Web**.
- Toutes les adresses en haut de votre navigateur comme : <https://www.google.com> sont des **URL**.
- Lorsque vous faites une recherche sur Google pour trouver le site d'EMSI en tapant le mot « EMSI", la barre d'adresse contient une URL un peu longue qui ressemble à ceci:

```
https://www.google.com/search?q=emsi&sca_esv=572530057
```

# Passage de données (URL et formulaire)

- Il est possible d'envoyer des informations lors d'une requête à un serveur:



- Le point d'interrogation sépare le nom de la page PHP des **paramètres**. Celle-ci peut récupérer ces informations dans des **variables**.

# Passage de données (URL et formulaire)

---

- Les paramètres s'enchaînent selon la forme **nom=valeur** et sont séparés les uns des autres par le symbole **&** .

```
page.php?param1=valeur1&param2=valeur2&param3=valeur3&param4=valeur4...
```

- **La seule limite** est **la longueur de l'URL**. En général, il n'est pas conseillé **de dépasser les 256 caractères**, mais les navigateurs arrivent parfois à gérer des URL plus longues.

# Passage de données (URL et formulaire)

- Nous voulons faire un lien de **index.php** à **bonjour.php** pour transmettre des informations dans l'URL :



# Passage de données (URL et formulaire)

---

- Pour cela, nous ouvrez `index.php` (puisque c'est lui qui contiendra le lien) et insérez le code suivant :

```
1 <a href="bonjour.php?nom=Dupont&prenom=Jean">Dis-moi bonjour !</a>
```

- Ce lien appelle la page `bonjour.php` et lui envoie deux paramètres :
  - `nom` : Dupont ;
  - `prenom` : Jean.

# Passage de données (URL et formulaire)

---

- La deuxième solution pour faire passer des informations dans l'URL, c'est de proposer à l'utilisateur de soumettre **un formulaire avec la méthode HTTP GET**.
- Nous utilisons une balise **<form></form>** qui a pour attribut **method** avec la valeur **GET**.

```
1 <form action="contact.php" method="GET">
2     <!-- données à faire passer à l'aide d'inputs -->
3     <input name="nom">
4     <input name="prenom">
5     <input type="submit" value="Envoyer">
6 </form>
```

# Récupérez les paramètres en PHP

- Nous avons un formulaire de contact – `contact.php` – que nous allons soumettre sur une autre page, et qui affichera un message de bonne réception : `submit_contact.php`

```
1 <form action="submit_contact.php" method="GET">
2     <div>
3         <label for="email">Email</label>
4         <input type="email" name="email">
5     </div>
6     <div>
7         <label for="message">Votre message</label>
8         <textarea placeholder="Exprimez vous" name="message"></textarea>
9     </div>
10    <button type="submit">Envoyer</button>
11 </form>
```



# Récupérez les paramètres en PHP

- Le formulaire va alors être converti en lien vers :

```
submit_contact.php?email=utilisateur%40exemple.com&message=Bonjour
```

- Et ces informations pourront être **récupérées** par PHP dans le fichier **submit\_contact.php**.
- Lors de la soumission, **une variable superglobale** appelée **\$\_GET** va **contenir les données envoyées**:

Nom	Valeur
<code>\$_GET['email']</code>	<u><a href="mailto:utilisateur@exemple.com">utilisateur@exemple.com</a></u>
<code>\$_GET['message']</code>	Bonjour

# Récupérez les paramètres en PHP

---

- On peut donc récupérer ces informations, les traiter, les afficher, etc...
- Pour l'exemple, nous créons un nouveau fichier PHP `submit_contact.php` et nous y plaçons le code suivant :

```
1 <h1>Message bien reçu !</h1>
2
3 <div class="card">
4
5     <div class="card-body">
6         <h5 class="card-title">Rappel de vos informations</h5>
7         <p class="card-text"><b>Email</b> : <?php echo $_GET['email']; ?></p>
8         <p class="card-text"><b>Message</b> : <?php echo $_GET['message']; ?></p>
9     </div>
10 </div>
```

# Récupérez les paramètres en PHP

---

➤ Nous obtenons le résultat suivant :

**Message bien reçu !**

**Rappel de vos informations**

**Email** : utilisateur@exemple.com

**Message** : Bonjour

# Récupérez les paramètres en PHP

- Si la méthode est **POST** (bonne pratique), alors c'est la **supervariable** **\$\_POST** qui recevra les données.

```
1 <form action="submit_contact.php" method="POST">
2     <div>
3         <label for="email">Email</label>
4         <input type="email" name="email">
5     </div>
6     <div>
7         <label for="message">Votre message</label>
8         <textarea placeholder="Exprimez vous" name="message"></textarea>
9     </div>
10    <button type="submit">Envoyer</button>
11 </form>
```

Nom	Valeur
<code>\$_POST['email']</code>	<a href="mailto:utilisateur@exemple.com">utilisateur@exemple.com</a>
<code>\$_POST['message']</code>	Bonjour

# Transfert de fichiers

---

- Pour envoyer un fichier, il faut ajouter l'attribut `enctype="multipart/form-data"` à la balise `<form>` :

```
1 <form action="submit_contact.php" method="POST" enctype="multipart/form-data">
2     <!-- champs de formulaire -->
3 </form>
```

- Grâce à `enctype`, le navigateur du visiteur sait qu'il s'apprête à envoyer des fichiers.

# Transfert de fichiers

- Nous ajoutons à l'intérieur du formulaire une balise permettant d'envoyer un fichier, c'est une balise très simple de type `<input type="file" />` .
- Il faut donner un **nom** à ce champ de formulaire (grâce à l'attribut **name**) pour que PHP puisse reconnaître le champ par la suite.

```
1 <form action="submit_contact.php" method="POST" enctype="multipart/form-data">
2   <!-- Ajout des champs email et message -->
3   [...]
4   <!-- Ajout champ d'upload ! -->
5   <div class="mb-3">
6       <label for="screenshot" class="form-label">Votre capture d'écran</label>
7       <input type="file" class="form-control" id="screenshot" name="screenshot" />
8   </div>
9   <!-- Fin ajout du champ -->
10  <button type="submit" class="btn btn-primary">Envoyer</button>
11 </form>
```

# Transfert de fichiers

---

- Au moment où la page PHP s'exécute, le fichier a été envoyé sur le serveur mais il est stocké dans un **dossier temporaire**.
- C'est à nous de décider si nous acceptons définitivement le fichier ou non.
- Pour chaque fichier envoyé, une variable `$_FILES['nom_du_champ']` est créée. Dans notre cas, la variable s'appellera `$_FILES['screenshot']`.


```
<input type="file" class="form-control" id="screenshot" name="screenshot" />
```

# Transfert de fichiers

- Cette variable est un tableau qui contient plusieurs informations sur le fichier :

Variable	Signification
<code>\$_FILES['sc reenshot'] ['name']</code>	Contient le nom du fichier envoyé par le visiteur.
<code>\$_FILES['sc reenshot'] ['type']</code>	Indique le type du fichier envoyé. Si c'est une image gif par exemple, le type sera <code>image/gif</code>



Variable	Signification
<code>\$_FILES['screenshot']['size']</code>	<p>Indique la taille du fichier envoyé.</p> <div>  <b>Attention</b> : cette taille est en octets. Il faut environ 1 000 octets pour faire 1 Ko, et 1 000 000 d'octets pour faire 1 Mo.            La taille de l'envoi est limitée par PHP. Par défaut, impossible d'uploader des fichiers de plus de 8 Mo.         </div>
<code>\$_FILES['screenshot']['tmp_name']</code>	Juste après l'envoi, le fichier est placé dans un répertoire temporaire sur le serveur en attendant que votre script PHP décide si oui ou non il accepte de le stocker pour de bon. Cette variable contient l'emplacement temporaire du fichier (c'est PHP qui gère ça).
<code>\$_FILES['screenshot']['error']</code>	Contient un code d'erreur permettant de savoir si l'envoi s'est bien effectué ou s'il y a eu un problème et si oui, lequel. La variable vaut 0 s'il n'y a pas eu d'erreur.

# Transfert de fichiers

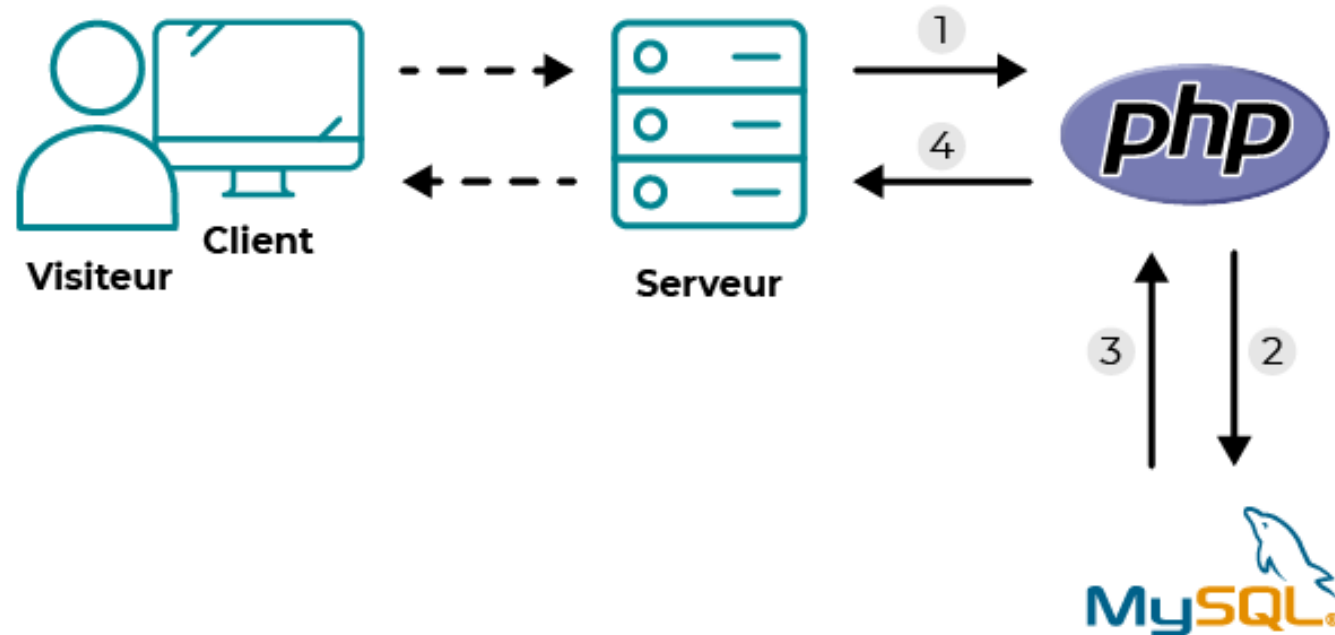
---

- Pour accepter le fichier, nous utilisons fonction `move_uploaded_file`, elle permet d'enregistrer le fichier d'une manière définitive.
- Comme `$_FILES['screenshot']['name']` contient le chemin entier vers le fichier d'origine ( `C:\dossier\fichier.png` , par exemple), il nous faudra extraire le nom du fichier.
- On peut utiliser pour cela la fonction `basename` qui renverra juste « `fichier.png` ».

```
// On peut valider le fichier et le stocker définitivement  
move_uploaded_file($_FILES['screenshot']['tmp_name'], 'uploads/' . basename($_FILES['screenshot']['name']));
```

# Interaction avec la base de données

➤ PHP va faire l'intermédiaire entre le client et MySQL.



# Interaction avec la base de données

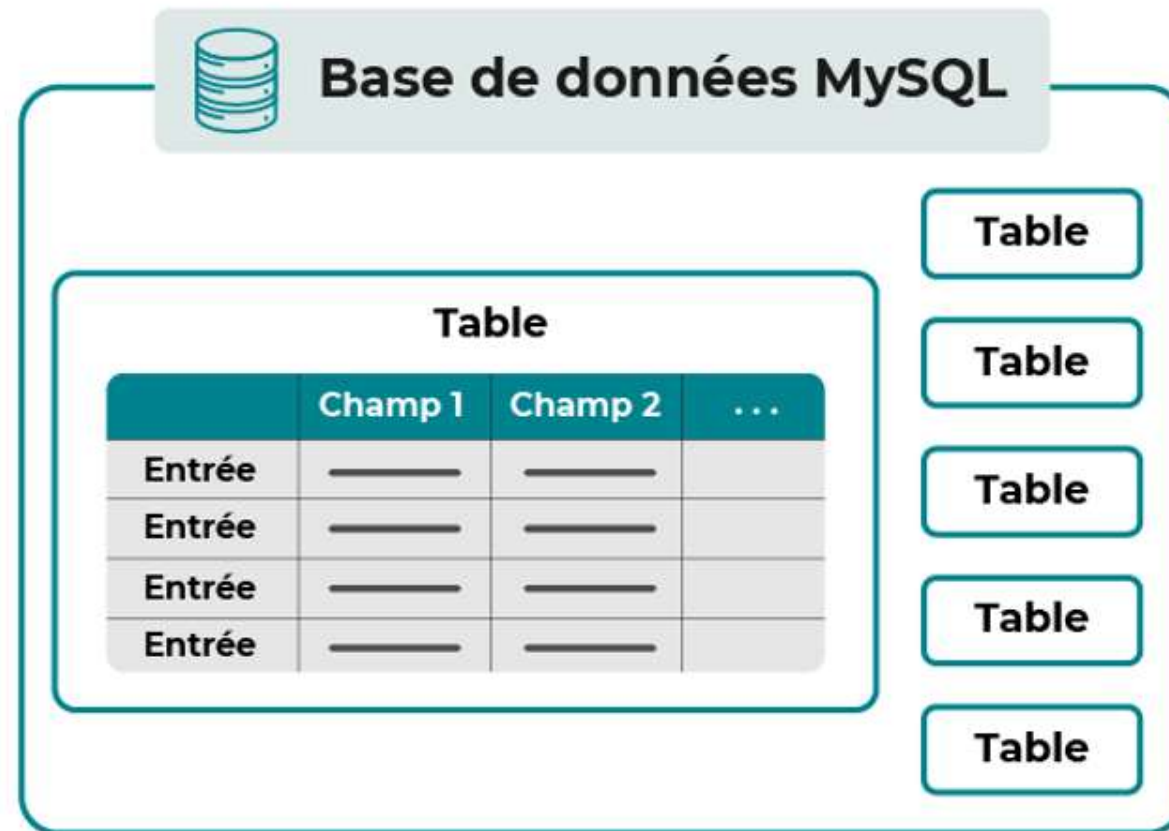
---

➤ Voici ce qui peut se passer lorsque le serveur reçoit une demande d'un client qui veut poster un message :

1. Le serveur utilise toujours **PHP**, il lui fait donc passer le message.
2. **PHP** effectue les actions demandées et se rend compte qu'il a besoin de **MySQL**. En effet, le code **PHP** contient à un endroit "Va demander à **MySQL** d'enregistrer ce message". Il fait donc passer le travail à **MySQL**.
3. **MySQL** fait le travail que **PHP** lui a soumis et lui répond "OK, c'est bon !".
4. **PHP** renvoie au serveur que **MySQL** a bien fait ce qui lui était demandé.

# Interaction avec la base de données

- La **base de données** contient plusieurs **tables**, chacune est en fait **un tableau** où les **colonnes** sont appelées champs et où les **lignes** sont appelées entrées.



# Interaction avec la base de données

- Pour se **connecter à une base de données MySQL**, vous allez devoir utiliser **une extension PHP** appelée **PDO** ("PHP Data Objects"). Cette extension est fournie avec PHP, mais parfois il vous faudra activer l'extension.



# Interaction avec la base de données

---

- Maintenant que nous sommes certains que **PDO** est **activé**, nous pouvons nous connecter à **MySQL**. Nous allons avoir besoin de quatre renseignements:
  - **Le nom de l'hôte** : c'est l'adresse **IP** de l'ordinateur où MySQL est installé. Le plus souvent, MySQL est installé sur le même ordinateur que PHP : dans ce cas, mettez la valeur **localhost**.
  - **La base** : c'est le nom de la base de données à laquelle vous voulez vous connecter. Dans notre cas, la base s'appelle **my\_recipes** . Nous pouvons la créer avec phpMyAdmin.
  - **L'identifiant et le mot de passe** : ils permettent de vous identifier. Sur **WAMP**, la valeur de l'identifiant est **root** et le mot de passe est **vide**.

# Interaction avec la base de données

---

- Dans cet exemple, nous créons une connexion à la base de données.
- Pour créer la connexion, on indique dans l'ordre dans les paramètres :
  - le nom d'hôte : `localhost` ;
  - la base de données : `my_recipes` ;
  - l'identifiant : `root` ;
  - le mot de passe `vide`

```
1 <?php
2 // Souvent on identifie cet objet par la variable $conn ou $db
3 $mysqlConnection = new PDO(
4     'mysql:host=localhost;dbname=my_recipes;charset=utf8',
5     'root',
6     ''
7 );
8 ?>
```



# Interaction avec la base de données

- En cas d'erreur, **PDO** renvoie ce qu'on appelle une **exception**, qui permet de « capturer » **l'erreur**.

```
1 <?php
2 try
3 {
4     $db = new PDO('mysql:host=localhost;dbname=my_recipes;charset=utf8', 'root', '');
5 }
6 catch (Exception $e)
7 {
8     die('Erreur : ' . $e->getMessage());
9 }
10 ?>
```

- Voilà encore un code un peu nouveau pour nous :
  - S'il y a **une erreur**, il rentre dans le bloc catch et fait ce qu'on lui demande (ici, on arrête l'exécution de la page en affichant un message décrivant l'erreur).
  - Si au contraire **tout se passe bien**, PHP poursuit l'exécution du code et ne lit pas ce qu'il y a dans le bloc catch .

# Interaction avec la base de données

- Pour **afficher des détails sur l'erreur** d'une requête SQL mal écrit, il faut activer les erreurs lors de la connexion à la base de données via PDO:

```
1 <?php
2 $db = new PDO(
3     'mysql:host=localhost;dbname=my_recipes;charset=utf8',
4     'root',
5     '',
6     [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION],
7 );
8 ?>
```

- Désormais, toutes les requêtes SQL qui comportent des erreurs les afficheront avec un message beaucoup plus clair.

# Interaction avec la base de données

---

- Nous allons apprendre à lire des informations dans la base de données, L'objectif consiste à récupérer la liste des recettes qui sont maintenant stockées dans votre base de données.
- Effectuons la requête à l'aide de l'objet PDO :

```
1 <?php
2 $recipesStatement = $db->prepare('SELECT * FROM recipes');
3 ?>
```

- `$recipesStatement` est un objet de type `PDOStatement`. Cet objet va contenir la requête SQL que nous devons exécuter, et par la suite, les informations récupérées en base de données.

# Interaction avec la base de données

---

- Pour récupérer les données, nous demandons à l'objet `$recipesStatement` d'exécuter la requête SQL et de récupérer toutes les données dans un format "exploitable", c'est-à-dire sous forme d'un tableau PHP.

```
1 <?php
2 $recipesStatement->execute();
3 $recipes = $recipesStatement->fetchAll();
4 ?>
```

- "fetch" en anglais signifie « va chercher ».

```

1 <?php
2 try
3 {
4     // On se connecte à MySQL
5     $mysqlClient = new PDO('mysql:host=localhost;dbname=my_recipes;charset=utf8', 'root', '');
6 }
7 catch(Exception $e)
8 {
9     // En cas d'erreur, on affiche un message et on arrête tout
10    die('Erreur : '.$e->getMessage());
11 }
12
13 // Si tout va bien, on peut continuer
14
15 // On récupère tout le contenu de la table recipes
16 $sqlQuery = 'SELECT * FROM recipes';
17 $recipesStatement = $mysqlClient->prepare($sqlQuery);
18 $recipesStatement->execute();
19 $recipes = $recipesStatement->fetchAll();
20
21 // On affiche chaque recette une à une
22 foreach ($recipes as $recipe) {
23 ?>
24     <p><?php echo $recipe['author']; ?></p>
25 <?php
26 }
27 ?>

```

# Interaction avec la base de données

---

- 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 défaut)
  - Objet : `PDO::FETCH_OBJ`

```
15 // On récupère tout le contenu de la table recipes
16 $sqlQuery = 'SELECT * FROM recipes';
17 $recipesStatement = $mysqlClient->prepare($sqlQuery);
18 $recipesStatement->execute();
19 $recipes = $recipesStatement->fetchAll(PDO::FETCH_ASSOC);
```

# Interaction avec la base de données

---

- Il est possible de filtrer et trier les données en modifiant la requête SQL.
- On souhaite récupérer les recettes avec le champ `is_enabled` à `TRUE`, alors la requête au début sera la même qu'avant, mais vous ajouterez à la fin `WHERE is_enabled = TRUE`.

```
1 <?php
2 $sqlQuery = 'SELECT * FROM recipes WHERE is_enabled = TRUE';
```

# Interaction avec la base de données

- Les **marqueurs** sont des identifiants reconnus par PDO pour être remplacés lors de la préparation de la requête par les variables PHP :

```
1 <?php
2 $sqlQuery = 'SELECT * FROM recipes WHERE author = :author AND is_enabled = :is_enabled';
3
4 $recipesStatement = $db->prepare($sqlQuery);
5 $recipesStatement->execute([
6 'author' => 'mathieu.nebra@exemple.com',
7 'is_enabled' => true,
8 ]);
9 $recipes = $recipesStatement->fetchAll();
10 ];
```

On ne concatène **JAMAIS** une requête SQL pour passer des variables, au risque de créer des **injections SQL** !



# Interaction avec la base de données

---

- On peut utiliser **?** pour la remplacer lors de la préparation de la requête par les variables en respectant l'ordre:

```
1 <?php
2 $sqlQuery = 'SELECT * FROM recipes WHERE author = ? AND is_enabled = ?';
3
4 $recipesStatement = $db->prepare($sqlQuery);
5 $recipesStatement->execute(['mathieu.nebra@exemple.com',true]);
6 $recipes = $recipesStatement->fetchAll();
7 ]);
```

# Traitement avancé des données

---

- Pour **ajouter une entrée**, vous aurez besoin de connaître la requête SQL. En voici une par exemple qui ajoute une recette :
  - D'abord, on commence par les mots-clés **INSERT INTO** qui indiquent que nous voulons insérer une entrée.
  - Préciser ensuite **le nom de la table** (ici **recipes**), puis listez entre parenthèses les noms des champs dans lesquels nous souhaitons placer des informations.
  - Enfin, écrire **VALUES** suivi des valeurs à insérer dans **le même ordre que les champs**.

```
1 <?php
2 $sqlQuery = 'INSERT INTO recipes(title, recipe, author, is_enabled) VALUES (:title, :recipe,
   :author, :is_enabled)';
3
```

```

2 try
3 {
4     $db = new PDO('mysql:host=localhost;dbname=my_recipes;charset=utf8', 'root', '');
5 }
6 catch (Exception $e)
7 {
8     die('Erreur : ' . $e->getMessage());
9 }
10
11 // Ecriture de la requête
12 $sqlQuery = 'INSERT INTO recipes(title, recipe, author, is_enabled) VALUES (:title, :recipe,
    :author, :is_enabled)';
13
14 // Préparation
15 $insertRecipe = $db->prepare($sqlQuery);
16
17 // Exécution ! La recette est maintenant en base de données
18 $insertRecipe->execute([
19     'title' => 'Cassoulet',
20     'recipe' => 'Etape 1 : Des flageolets ! Etape 2 : Euh ...',
21     'author' => 'contributeur@exemple.com',
22     'is_enabled' => 1, // 1 = true, 0 = false
23 ]);

```

# Traitement avancé des données

---

- Pour **modifier** une recette, nous aurons besoin de **UPDATE** et **SET** .
  - Tout d'abord, le mot-clé **UPDATE** permet de dire qu'on va modifier une entrée.
  - Ensuite, le nom de la table ( **recipes** ).
  - Le mot-clé **SET** sépare le nom de la table de la liste des champs à modifier.
  - Viennent ensuite les champs qu'il faut modifier, séparés par des virgules. Ici, on modifie le champ **title** et le champ **recipe**.
  - Enfin, le mot-clé **WHERE** permet de dire à MySQL quelle entrée il doit modifier (sinon, toutes les entrées seraient affectées !). On se base très souvent sur le champ **recipe\_id** pour indiquer quelle entrée doit être modifiée.

```
1 UPDATE recipes SET title = :title, recipe = :recipe WHERE recipe_id = :id
```

# Traitement avancé des données

---

- Voici comment on **supprime** par exemple une recette à partir de son identifiant :
  - **DELETE FROM** : pour dire « supprimer dans » ;
  - **recipes** : le nom de la table ;
  - **WHERE** : indispensable pour indiquer quelles entrées doivent être supprimées.

```
1 DELETE FROM recipes WHERE recipe_id=:id
```

Après suppression, **il n'y a aucun moyen de récupérer les données**, alors faites bien attention !

# Les expressions régulières

---

- Les **expressions régulières** constituent un système très puissant et très rapide pour **faire des recherches dans des chaînes de caractères**. C'est une sorte de fonctionnalité **Rechercher / Remplacer** très poussée.
- Voici quelques exemples pratiques de ce que nous pouvons faire avec les expressions régulières:
  - Vérifier automatiquement si l'adresse e-mail entrée par le visiteur a une forme valide (dupont@gmail.com).
  - Modifier une date que vous avez au format américain (08-05-1985) pour la mettre dans le bon ordre en français (05/08/1985).

# Les expressions régulières

---

- `preg_match` est une fonction qui renvoie un booléen : true ou false. Elle renvoie **true** (vrai) si elle a trouvé le mot que vous cherchiez dans la chaîne, **false** (faux) si elle ne l'a pas trouvé.
- Cette fonction prend **deux paramètres** : votre **regex** (expression régulière) et **la chaîne dans laquelle vous faites une recherche**.

```
<?php
if (preg_match("** Votre REGEX **", "Ce dans quoi vous faites la
recherche"))
{
    echo 'Le mot que vous cherchez se trouve dans la chaîne';
}
else
{
    echo 'Le mot que vous cherchez ne se trouve pas dans la chaîne';
}
?>
```

# Les expressions régulières

- une regex (Expression régulière) est toujours entourée de caractères spéciaux appelés **délimiteurs (#)**.

```
<?php
if (preg_match("#guitare#", "J'aime jouer de la guitare."))
{
    echo 'VRAI';
}
else
{
    echo 'FAUX';
}
?>
```

Chaîne	Regex	Résultat
J'aime jouer de la guitare.	#guitare#	VRAI
J'aime jouer de la guitare.	#piano#	FAUX

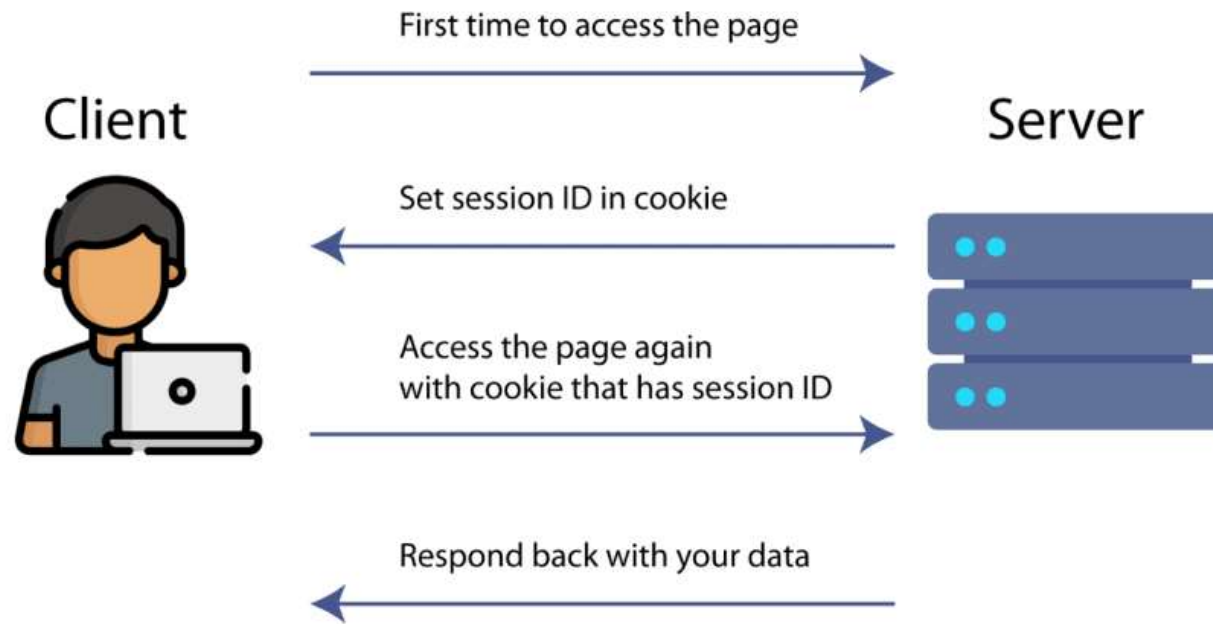


# Sessions

---

- Nous savons comment **passer des variables** de page en page à l'aide d'**URL** ou de **formulaires**.  
Mais dès qu'on **charge** une autre page, ces informations sont « **oubliées** ».
- **Les sessions** permettent de **conserver** des variables **sur toutes les pages** de votre site.
- Une **session** est représentée par **un identifiant unique** (généré par le serveur).
- Cet **identifiant** est placé dans un **cookie** stocké chez le client.
- A chaque **requête**, le **cookie** de session est **envoyé automatiquement** par le navigateur au **serveur**.

# Sessions



# Sessions

---

## ➤ **Étape 1** : création d'une session unique

1. Un visiteur arrive sur votre site.
2. On demande à créer une session pour lui.
3. PHP génère alors un numéro unique.

➤ Ce numéro est souvent très grand. Exemple : a02bbffc6198e6e0cc2715047bc3766f. Ce numéro sert **d'identifiant** ; c'est ce qu'on appelle un « **ID de session** » ou **PHPSESSID**

➤ PHP transmet **automatiquement** cet **ID** de page en page, en utilisant généralement un **cookie**.

# Sessions

---

- **Étape 2** : création de variables pour la session
- Une fois la session générée, on peut créer une infinité de variables de session pour nos besoins. Par exemple, on peut créer :
  - une variable qui contient le nom du visiteur : `$_SESSION['nom']`
  - une autre qui contient son prénom : `$_SESSION['prenom']`
  - etc.
- Le serveur **conserve** ces variables même lorsque la page PHP a fini d'être générée. Autrement dit : quelle que soit la page de votre site, vous pourrez récupérer le nom et le prénom du visiteur via la superglobale **`$_SESSION`** !

# Sessions

---

## ➤ **Étape 3** : suppression de la session

- Lorsque le visiteur **se déconnecte** de votre site, **la session est fermée** et PHP « **oublie** » alors **toutes les variables de session** que vous avez créées.
- Ou on attend quelques minutes **d'inactivité** pour le **déconnecter automatiquement** : on parle alors de "**timeout**". Le plus souvent, le visiteur est déconnecté par un timeout.

# Sessions

---

- Pour activer ou détruire une session, deux fonctions sont à connaître :
  - `session_start()` : démarre le système de sessions. Si le visiteur vient d'arriver sur le site, alors un numéro de session est généré pour lui.
  - `session_destroy()` : ferme la session du visiteur. Cette fonction est automatiquement appelée lorsque le visiteur ne charge plus de page de votre site pendant plusieurs minutes (c'est le timeout), mais vous pouvez aussi créer une page « Déconnexion » si le visiteur souhaite se déconnecter manuellement.
- Il faut appeler `session_start()` sur chacune de vos pages **AVANT** d'écrire le moindre code HTML ou PHP (avant même la balise `<!DOCTYPE>` ).
- Si vous oubliez de lancer `session_start()`, vous ne pourrez pas accéder à la variable superglobale `$_SESSION` .

# Sessions

---

- Si on appelle `session_start()` peu de temps après avoir **détruit la session**, PHP réutilise parfois **le même identifiant** de session.
- Pour démarrer une nouvelle session et éviter ce problème il faut utiliser la fonction `session_regenerate_id()`, elle utilise un nouvel id de session.
- Il est possible de **réinitialiser** le tableau **`$_SESSION`** avec la fonction **`session_unset()`**.