SCI6306 Bases de données documentaires (Automne 2023)

Christine Dufour, EBSI, UdeM

A2023 1er décembre 2023

Cours 12 : Publication dynamique

SCI6306

Christine Dufour, EBSI, UdeM

Table des matières

Publication dynamique de contenu d'une base de données sur le Web avec PHP

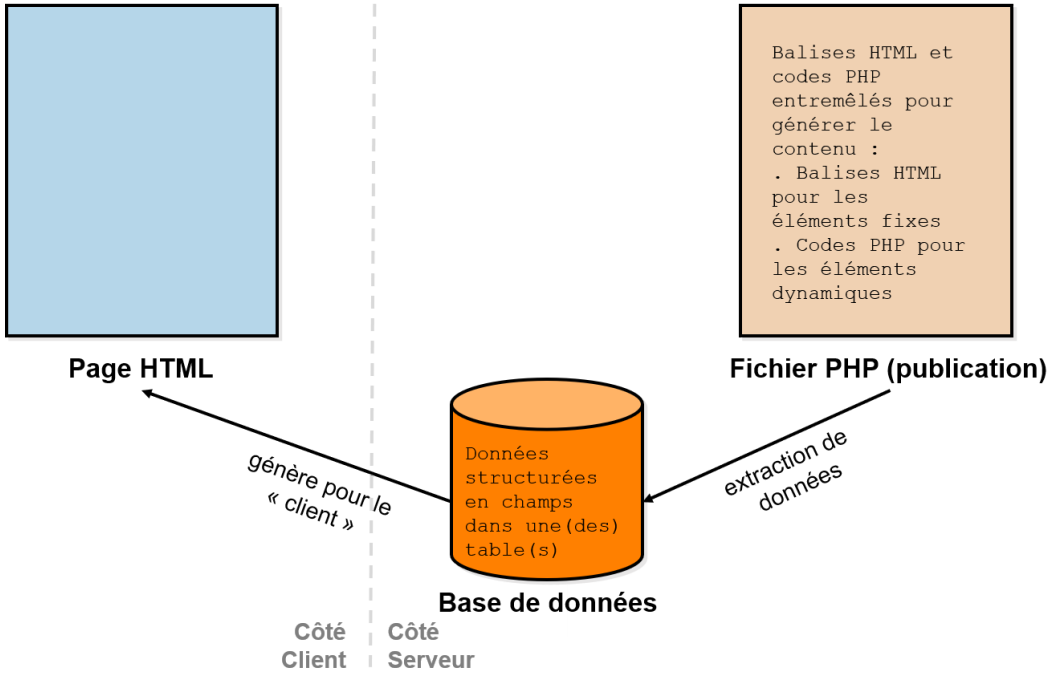
- + Au programme aujourd'hui
- Processus général pour la publication dynamique
- Structures algorithmiques de base
    - SI ... ALORS ... SINON (aiguillage simple = 1 seule condition)
    - CHOIX MULTIPLES (aiguillage complexe = plusieurs conditions)
    - TANT QUE (boucle avec nombre de répétitions inconnues)
    - POUR (boucle avec nombre de répétitions connues)
- Code PHP pour publier : processus
- Ressources en lien avec le cours

1. + Au programme aujourd'hui

- **Publication Web**
    - Processus général
    - Structures algorithmiques
    - Code PHP pour publier : processus et exemples
    - Exercices

2. Processus général pour la publication dynamique

Comme pour la saisie de données dans une BD à partir d'une page Web, la publication dynamique du contenu d'une base de données sur le Web implique les trois couches de l'architecture impliquée, soit (1) du côté **client**, la **visualisation de la page Web** dans le navigateur, (2) du côté **serveur**, l'interaction avec la **BD** pour extraire les données, et (3) encore du côté **serveur**, l'exécution d'**instructions PHP** permettant de générer la page Web.



Processus pour la publication Web

Dans ce scénario, les instructions PHP sont exécutées dès qu'un ou une internaute demande à afficher un certain contenu dynamique. Comme nous le verrons, ces **instructions PHP** seront **entrelacées** avec des éléments **HTML** dans une page, les éléments **HTML** servant pour les **éléments fixes** (par exemple, un titre de page, des directives, un texte d'introduction), tandis que les instructions **PHP** auront pour objectif d'aller **chercher** et **traiter** l'information provenant de la base de données.

Le fichier PHP pour la publication Web sera différent de celui pour l'enregistrement (saisie) entre autres parce qu'on y retrouvera non pas des requêtes SQL pour manipuler les données (en ajouter, en modifier ou en supprimer), mais plutôt des **requêtes SELECT** pour extraire des données. Nous commencerons par voir les principales structures algorithmiques qui peuvent être exploitées pour faire l'affichage pour, par la suite, examiner plus avant la construction d'une page pour afficher des données.

3. Structures algorithmiques de base

Différentes **structures algorithmiques** peuvent être exploitées pour faciliter la **publication dynamique** de contenu sur le Web. En fait, nous en avons même vu une dans le contexte de la saisie via le Web, soit une **structure conditionnelle** afin de gérer l'enregistrement des valeurs nulles! Ces structures algorithmes existent dans tous les langages procéduraux, leur syntaxe pouvant différer. Nous verrons les structures suivantes :

- **Structures logiques (conditionnelles)**
  - Aiguillage simple (SI ... ALORS ... SINON)
  - Aiguillage complexe (CHOIX MULTIPLES)
- **Structures répétitives (boucles)**
  - Nombre de cycles inconnus (TANT QUE)
  - Nombre de cycles prédéterminés (POUR)

3.1. SI ... ALORS ... SINON (aiguillage simple = 1 seule condition)

Fondamental **: Fonctionnement**

Une structure conditionnelle simple exécute minimalement un bloc d'**instructions** si la **condition est vraie**. Elle peut aussi inclure un bloc d'**instructions** si la condition est **fausse**.

¶ *Syntaxe* : Structure générique (forme complète en PHP*)

```
IF (condition)
{//Bloc d'instructions à exécuter si la condition est vraie}
ELSE
{//Bloc d'instructions - facultatif - à exécuter si la condition est fausse}
```

* Rappel de la forme compacte pour affichage : (*condition?élément retourné si vrai:élément retourné si faux*)

🔎 *Exemple* :

On peut utiliser une structure conditionnelle simple pour vérifier la valeur d'un contrôle (par exemple, vérifier si la note obtenue est inférieure à un certain seuil) et définir un affichage différent en fonction de cette valeur.

Exemple : https://cours.ebsi.umontreal.ca/sci6306/algo/condition.php - pour l'affichage du **statut échec/réussite**

[cf. condition.pdf]Extrait du code (lignes 37-46 du code de la page [cf. condition.pdf]) :

```
if ($donnees['note']<60)
{
  // Si la note est sous 60, écriture dans le fichier HTML, en gras, de "a eu un échec pour"
  echo '<strong> a eu un échec pour </strong>';
}
else
{
  // Si la note est plus grande ou égale à 60, écriture, en gras, de "a réussi"
  echo '<strong> a réussi </strong>';
}
```

- On retrouve à la ligne 1, la condition qui compare la note obtenue au seuil de 60 pour voir si elle est sous ce seuil.
- Les lignes 2 à 5 représentent les instructions si la condition est vraie. On y retrouve une instruction `echo` afin d'afficher un message d'échec.
- Les lignes 7 à 10 regroupent les instructions si la condition est fausse (`else`). On y retrouve l'instruction `echo` qui affiche un message de réussite.

3.2. CHOIX MULTIPLES (aiguillage complexe = plusieurs conditions)

💡 *Fondamental* : Fonctionnement

La structure conditionnelle complexe permet d'exécuter plus de deux blocs d'instructions en fonction des valeurs prises par une variable.

¶ *Syntaxe* : Structure générique (PHP)

```
SWITCH (variable) {
CASE valeur1:
//Bloc d'instructions à exécuter si variable = valeur1;
break;
[//Autant de CASE que nécessaire]
DEFAULT:
//Bloc d'instructions à exécuter si variable = autres valeurs - facultatif;
}
```

🔎 *Exemple* :

Une structure conditionnelle complexe peut permettre de personnaliser l'affichage des données en fonction des différentes valeurs prises par un champ, comme, par exemple, d'attribuer à une note numérique son équivalent en note littérale. *Note* : Cela pourrait aussi se faire par une série de IF imbriqués mais cela devient rapidement compliqué lorsqu'il y a plusieurs valeurs à tester...

Exemple : https://cours.ebsi.umontreal.ca/sci6306/algo/condition.php - pour l'affichage de la **note littérale**

Extrait du code (lignes 53-93 du code de la page [cf. condition.pdf]) :

```php
switch (true) {
    case $donnees['note'] >= 90:
        echo "A+";
        break;
    case $donnees['note'] >= 85:
        echo "A";
        break;
    case $donnees['note'] >= 80:
        echo "A-";
        break;
    case $donnees['note'] >= 77:
        echo "B+";
        break;
    case $donnees['note'] >= 73:
        echo "B";
        break;
    case $donnees['note'] >= 70:
        echo "B-";
        break;
    case $donnees['note'] >= 65:
        echo "C+";
        break;
    case $donnees['note'] >= 60:
        echo "C";
        break;
    case $donnees['note'] >= 57:
        echo "C-";
        break;
    case $donnees['note'] >= 54:
        echo "D+";
        break;
    case $donnees['note'] >= 50:
        echo "D";
        break;
    case $donnees['note'] >= 35:
        echo "E";
        break;
    Default:
        echo "F";
        break;
}
```

- La ligne 1 établit la base de la conditionnelle complexe qui consiste ici à chercher les cas où la condition énoncée est vraie.
- Par exemple, à la ligne 2, on vérifie si la note est plus grande ou égale à 90. Si cette condition est vraie, la ligne 55 fera afficher un A+.
- La ligne 4 est importante : l'instruction "break" permet d'arrêter la structure conditionnelle afin d'éviter que les autres conditions qui suivent ne soient validées, ce qui causerait en ce cas problème. En effet, si la note est plus grande ou égale à 90, elle est plus grande ou égale à 85 (la condition suivante à la ligne 5).

3.3. TANT QUE (boucle avec nombre de répétitions inconnues)

***Fondamental* : Fonctionnement**

Une boucle TANT QUE permet d'exécuter un bloc d'**instructions tant que** la condition demeure **vraie**. On ne connaît pas en amont le nombre exact de fois où la boucle sera répétée.

¶ *Syntaxe* : Structure générique (PHP)

```
1 WHILE (condition)
2 {//Bloc d'instructions à exécuter}
```

🔍 *Exemple* :

Ce type de boucle est particulièrement utile pour passer à travers les enregistrements dans une table de résultats retournée par une requête SQL. On ne sait en effet pas à l'avance le nombre de lignes retournées; on voudra en ce cas faire la boucle jusqu'à ce que l'on ait terminé de passer au travers les différentes lignes de la table de résultats.

Exemple : https://cours.ebsi.umontreal.ca/sci6306/algo/repetition.php - pour afficher la liste des titres des cours

[cf. repetition.pdf]Extrait du code (lignes 30-34 du code de la page [cf. repetition.pdf]) :

```
1 while ($donnees = $req->fetch())
2     {
3         // Écriture dans le fichier HTML des balises d'ouverture et de fermeture de paragraphe encadrant l'identifiant du cours (alias TITRE_COURS qui correspond à la concaténation NO_COURS TITRE (voir SQL)
4         echo '<p>'.$donnees['titre_cours'].'</p>';
5     }
```

- La ligne 1 précise la condition de la boucle : tant qu'il y aura des lignes qui demeurent dans la table des résultats. L'instruction fait en sorte de mettre dans la variable `$donnees` une ligne que l'on va chercher (`fetch`) dans la requête SQL qui a été exécutée (`$req`). À chaque tour de la boucle la ligne suivante est mise en mémoire et, lorsqu'il n'y a plus de lignes, cela retourne "Faux" ce qui termine la boucle.
- lignes 3 et 4 seront répétées pour toutes les lignes de la table des résultats ce qui affichera ainsi, dans un paragraphe, le titre des cours.

3.4. POUR (boucle avec nombre de répétitions connues)

💡 *Fondamental* : Fonctionnement

Une boucle POUR exécute un bloc d'**instructions** de manière répétitive pour un **nombre de fois prédéterminé**, c'est-à-dire à l'intérieur d'une boucle allant d'une valeur de départ à une valeur de fin avec une certaine valeur d'incrémentation entre ces deux extrêmes. Par exemple, on peut aller de 1 à 10 à pas de 10 ou aller de 2 à 8 à pas de 2!

¶ *Syntaxe* : Structure générique (PHP)

```
1 FOR (valeur de départ; valeur de fin; valeur de l'incrémentation)
2 {//Bloc d'instructions à exécuter}
```

🔍 *Exemple* :

Une boucle POUR est une autre manière pour passer à travers l'ensemble des enregistrements dans une table de résultats retournée par une requête SQL lorsque l'on connaît le nombre de lignes que l'on veut. On pourrait par exemple vouloir afficher uniquement les 3 premières lignes de la table des résultats.

Exemple : [1] https://cours.ebsi.umontreal.ca/sci6306/algo/repetition.php - pour afficher les trois premiers cours

[cf. repetition.pdf]Extrait du code (lignes 50-57 du code de la page [cf. repetition.pdf]) :

```
1 for ($i=1;$i<4;$i++)
2     {
3         // Mise en mémoire d'une ligne dans la variable $donnees
4         $donnees = $req->fetch();
5
```

[1] http://cours.ebsi.umontreal.ca/sci6306/algo/condition.php

```
6           // Affichage de l'identifiant du cours et des balises LI associées
7           echo '<li>'.$donnees['titre_cours'].'</li>';
8       }
```

- La boucle est définie à la ligne 1. On veut aller de 1 jusqu'à 4 (non inclus) à pas de 1 (`$i++`), donc afficher trois lignes.
- À la ligne 4, on demande de mettre en mémoire (dans `$donnees` grâce à `fetch`) une ligne de la table des résultats (`$req`).
- À la ligne 7, on fait afficher comme un item de liste (`li`) le titre du cours qui avait été mis en mémoire à la ligne 4.
- Lorsque la boucle recommence, la ligne 4 met en mémoire la ligne suivante de la table des résultats.
- La boucle continue jusqu'à ce que le compteur (`$i`) soit égal à 4.

4. Code PHP pour publier : processus

Six blocs sont nécessaires pour afficher des données extraites d'une base de données X à partir d'une requête Y :

1. **Connexion** à la **base de données X**
2. **Définition** d'une **requête Y**
3. **Exécution** de la **requête Y**
4. **Affichage des résultats** de la requête Y
5. **Fermeture** de la **requête Y**
6. **Fermeture** de la connexion à la **base de données X**

Ainsi, par comparaison aux blocs nécessaires pour enregistrer des données dans une base de données, on ne retrouve qu'un **nouveau bloc**, soit celui de l'**affichage** des résultats (**bloc 4**). Nous n'insisterons que sur ce dernier, les autres demeurant très similaires.

Lorsque l'on débute dans le développement de pages permettant de publier du contenu dynamiquement, il peut être utile de **systématiser** le processus de préparation des pages en le découpant en **4 étapes** :

Processus pour la préparation du code PHP pour publier des données

No de l'étape	Objectif de l'étape	Correspondance aux blocs	Commentaires
1	Préparer la **requête SQL** pour extraire les données à publier ainsi que les instructions **PHP** correspondantes pour l'exécuter	Blocs 1, 2, 3	Il faut ici réfléchir aux données dont nous aurons besoin pour l'affichage. Il faut y inclure **tout** ce qui est nécessaire, mais **seulement** ce qui est nécessaire. Le défi ici est plus souvent sur le plan **SQL** que PHP.
2	Préparer le **code HTML et PHP** pour afficher **une ligne** de la table des résultats	Bloc 4	On pourrait se demander pourquoi on ne se concentre que **sur une seule ligne** de la table des résultats. Il faut se rappeler que la table des résultats est lue de **manière linéaire**, ligne après ligne. Ainsi, si on comprend comment afficher les résultats d'une ligne, il sera facile de répéter par la suite pour les autres lignes.

No de l'étape	Objectif de l'étape	Correspondance aux blocs	Commentaires
			Il faut décider du **type d'affichage** désiré pour les données : en paragraphe? en tableau? dans une liste à puces? Une fois ce choix fait, il faut préparer le code **HTML** en conséquence. **Ne pensez à ce moment-ci qu'en HTML avec les données fixes** provenant d'une ligne de la table des résultats. Lorsque l'on débute, il est plus facile de procéder ainsi. Par la suite, repérez dans le code HTML **les données dynamiques** (celles qui changeront à chacune des lignes de la table des résultats) et **remplacez-les** par les instructions **PHP** correspondantes.
3	Identifier dans le code créé les éléments qui se **répéteront** pour **chaque ligne** de la table des résultats et les **encadrer** par les instructions **PHP** permettant de **parcourir la table des résultats**	Bloc 4	Il faut cibler maintenant ce qu'il est nécessaire, dans notre code **HTML et PHP**, de **répéter** pour **chacune des lignes de la table des résultats**. Par exemple, si on décide d'afficher le résultat sous forme d'une liste à puces, nous voudrons répéter pour chacune des lignes l'élément LI, mais nous ne voudrons pas le faire pour l'élément OL ou UL. Il faut **ajouter** dans le code les **instructions PHP** pour que ces éléments soient **répétés** pour chacune des lignes de la table (boucle WHILE).
4	Insérer le code **PHP** pour **fermer** la requête et la connexion à la BD	Blocs 5, 6	Finalement, il ne restera qu'à préparer les instructions **PHP** pour **fermer** tant la **requête** que la connexion à la **BD**.

Deux exemples seront maintenant présentés pour illustrer ce processus.

Exemple : Affichage de données sous forme d'un paragraphe

Soit la base de données INSCRIP. Imaginons que nous aimerions faire **afficher** pour **chacun des cours**, dans un **paragraphe**, son **horaire** et son **statut** de cette manière :

*« Le cours optionnel **2001 Histoire 101** se donne le 1^{e} jour de la semaine, à 09:00:00 heure. »*

Pour mieux visualiser le résultat, vous pouvez consulter la page à l'URL https://cours.ebsi.umontreal.ca/sci6306/demo_c12/info_cours.php.

Reprenons une à une les étapes préalablement proposées.

ÉTAPE 1

Objectif de l'étape : Préparer la **requête SQL** pour extraire les données à publier ainsi que les instructions **PHP** correspondantes

Résultat :

La requête SQL devra aller chercher les champs STATUT, NO_COURS, TITRE, JOUR et HEURE de la table COURS :

```
1 SELECT no_cours, titre, jour, heure, statut FROM cours ORDER BY no_cours;
```

Cette requête viendra s'inscrire dans le bloc 2, le bloc 1 visant à établir la connexion et le bloc 3, à exécuter la requête :

```
1 //Bloc 1
2 include "App_Data/ouverture_bd.php";
3
4 //Bloc2
5 $req_cours = $bdd->prepare('SELECT no_cours, titre, jour, heure, statut FROM cours ORDER
  BY no_cours;');
6
7 //Bloc 3
8 $req_cours->execute();
```

Remarquez ici la forme de l'instruction `execute`. Comme on ne retrouve aucune variable dans la requête SQL, il n'y a pas à passer en arguments/paramètres les valeurs de ces dernières comme nous avions à le faire pour l'enregistrement des données. C'est ce qui explique qu'on ne retrouve rien entre les parenthèses.

Une fois ces trois lignes exécutées, nous aurons mis en mémoire, dans la variable `$req_cours`, la table de résultats retournée par la requête SQL. Par la suite, nous pourrons aller chercher une ligne à la fois avec la variable `$donnees` qui contiendra tous les champs pour cette ligne que nous pourrons cibler individuellement avec la syntaxe `$donnees['statut']` où `statut` est le nom de la colonne dans la requête SQL (détails plus tard à l'étape 3).

ÉTAPE 2

Objectif de l'étape : Préparer le code **HTML et PHP** pour afficher **une ligne** de la table des résultats

Résultat :

Le code HTML nécessaire pour afficher la première ligne du tableau, si on le pense **statique** et non dynamique, serait le suivant :

```
1 <p>Le cours optionnel <strong>20001 Histoire 101</strong> se donne le 1<sup>e</sup> jour
  de la semaine, à 09:00:00 heure.</p>
```

Ainsi, si on veut généraliser ce code à toutes les lignes (donc le penser **dynamique**; les commentaires représentent les éléments dynamiques génériques) :

```
1 <p>Le cours <!-- statut --> <strong><!-- no_cours titre --></strong> se donne le <!--
  jour --><sup>e</sup> jour de la semaine, à <!-- heure --> heure.</p>
```

Finalement, la "dynamisation" en PHP donnerait le code suivant :

```
1 <p>Le cours <?php echo $donnees['statut'];?> <strong><?php echo $donnees['no_cours'];?>
  <?php echo $donnees['titre'];?></strong> se donne le <?php echo $donnees['jour'];?>
  <sup>e</sup> jour de la semaine, à <?php echo $donnees['heure'];?> heure.</p>
```

Ce qu'il faut comprendre ici, c'est que chacun des **éléments dynamiques** nécessitera un morceau de **code PHP** pour l'afficher, donc une instruction `echo`. De plus, il faudra bien nommer ces éléments, par exemple `$donnees['statut']`.

ÉTAPE 3

Objectif de l'étape : Identifier dans le code créé les éléments qui se **répéteront** pour **chaque ligne** de la table des résultats et les **encadrer** par les instructions **PHP** permettant de **parcourir la table des résultats**

Résultat :

Comme nous voulons obtenir un paragraphe par ligne, tout le code `<p>...</p>` sera répété pour chacune des lignes de la table des résultats. Il faudra ainsi l'inscrire dans une boucle (`WHILE`) qui permettra de parcourir une à la suite de l'autre toutes les lignes de la table des résultats :

```
1 <?php
2 while ($donnees = $req_cours->fetch())
3 {
4 ?>
5    <p>Le cours <?php echo $donnees['statut'];?> <strong><?php echo $donnees['no_cours'];?
  > <?php echo $donnees['titre'];?></strong> se donne le <?php echo $donnees['jour'];?>
  <sup>e</sup> jour de la semaine, à <?php echo $donnees['heure'];?>.</p>
6 <?php
7 }
```

```
?>
```

La condition de la boucle **`WHILE`** peut se comprendre ainsi : on **verse** (instruction `fetch`) dans la variable **`$donnees`** une **ligne de la table de résultats** (cette table ayant été mise à l'étape 1 dans la variable `$req_cours`) et on **exécute** les **instructions** qui se retrouvent dans la **boucle** (soit l'affichage du paragraphe avec les données de la ligne mise en mémoire). On continue tant qu'il reste des lignes dans la table de résultats (chaque passage dans la boucle va chercher la ligne suivante). La boucle s'**arrête** lorsqu'il n'y a **plus de lignes** non traitées dans la table des résultats.

ÉTAPE 4

Objectif de l'étape : Insérer le code **PHP** pour **fermer** la requête et la connexion à la BD

Résultat :

Les instructions PHP pour fermer la requête et la connexion à la BD sont les mêmes que pour l'enregistrement de données :

```php
$req_cours->closeCursor();
$bdd=null;
```

Ainsi, au final, la publication en paragraphe des informations sur les cours nécessitera le code suivant :

```php
<?php
    //Bloc 1
    include "App_Data/ouverture_bd.php";

    //Bloc 2
    $req_cours = $bdd->prepare('SELECT no_cours, titre, jour, heure, statut FROM cours ORDER BY no_cours;');

    //Bloc 3
    $req_cours->execute();

    //Bloc 4
    while ($donnees = $req_cours->fetch())
    {
?>

    <p>Le cours <?php echo $donnees['statut'];?> <strong><?php echo $donnees['no_cours'];?> <?php echo $donnees['titre'];?></strong> se donne le <?php echo $donnees['jour'];?>e jour de la semaine, à <?php echo $donnees['heure'];?>.</p>

<?php
    }
    //Bloc 5
    $req_cours->closeCursor();

    //Bloc 6
    $bdd=null;
?>
```

Rappel : Il faut faire attention à bien identifier les moments où l'on "**parle PHP**" en les encadrant par les balises d'ouverture et de fermeture PHP, soit `<?php` et `?>`.

Exemple : Affichage de données sous forme d'une liste à puces

Soit la base de données INSCRIP. Imaginons que nous aimerions cette fois faire afficher la **liste des étudiant.e.s inscrit.e.s à des cours** sous forme d'une **liste à puces** comme suit :

- Bretécher, Claire (10002)
- Asimov, Isaac (10004)

Pour mieux visualiser le résultat, vous pouvez le consulter à l'URL https://cours.ebsi.umontreal.ca/sci6306/demo_c12/liste_etudiants.php.

Reprenons à nouveau les différentes étapes.

ÉTAPE 1

Objectif de l'étape : Préparer la **requête SQL** pour extraire les données à publier ainsi que les instructions **PHP** correspondantes

Résultat :

La requête SQL devra aller chercher les champs NO_ETUD et NOM de la table ETUD en vérifiant que les étudiant.e.s sont bien inscrit.e.s à un cours (table SUIT) :

```
SELECT DISTINCT etud.no_etud, nom FROM etud, suit WHERE etud.no_etud=suit.no_etud ORDER BY etud.no_etud;
```

Les blocs 1, 2 et 3 deviendront ainsi :

```
//Bloc 1
include "App_Data/ouverture_bd.php";

//Bloc 2
$req_etud = $bdd->prepare('SELECT DISTINCT etud.no_etud, nom FROM etud, suit WHERE etud.no_etud=suit.no_etud ORDER BY etud.no_etud;');

//Bloc 3
$req_etud->execute();
```

ÉTAPE 2

Objectif de l'étape : Préparer le code **HTML et PHP** pour afficher **une ligne** de la table des résultats

Résultat :

Le code HTML **statique** nécessaire pour afficher la première puce, serait le suivant :

```
<ul>
   <li>Bretécher, Claire (10002)</li>
</ul>
```

Ainsi, si on veut généraliser ce code à toutes les lignes (donc le penser **dynamique**; les commentaires représentent les éléments dynamiques génériques) :

```
<ul>
   <li><!-- nom --> (<!-- no_etud -->)</li>
</ul>
```

Finalement, la "dynamisation" en PHP donnerait le code suivant :

```
<ul>
   <li><?php echo $donnees['nom'];?> (<?php echo $donnees['no_etud'];?>)</li>
</ul>
```

ÉTAPE 3

Objectif de l'étape : Identifier dans le code créé les éléments qui se **répéteront** pour **chaque ligne** de la table des résultats et les **encadrer** par les instructions **PHP** permettant de **parcourir la table des résultats**

Résultat :

Ce que nous voudrons répéter pour toutes les lignes est uniquement l'**élément LI** et son **contenu**. Les balises d'ouverture et de fermeture de la liste à puces (UL) ne devraient pas se retrouver dans la boucle sinon elles seront démultipliées :

```
<ul>
  <?php
  while ($donnees = $req_etud->fetch())
  {
  ?>
    <li><?php echo $donnees['nom'];?> (<?php echo $donnees['no_etud'];?>)</li>
  <?php
  }
  ?>
</ul>
```

ÉTAPE 4

Objectif de l'étape : Insérer le code **PHP** pour fermer la requête et la BD

Résultat :

Les instructions PHP pour fermer la requête et la BD sont les mêmes que pour l'enregistrement de données :

```php
$req_etud->closeCursor();
$bdd=null;
```

Ainsi, au final, la publication en liste à puces des informations sur les étudiant.e.s inscrit.e.s à un cours nécessitera le code suivant :

```php
<?php
    //Bloc 1
    include "App_Data/ouverture_bd.php";

    //Bloc 2
    $req_etud = $bdd->prepare('SELECT DISTINCT etud.no_etud, nom FROM etud, suit WHERE etud.no_etud=suit.no_etud ORDER BY etud.no_etud;');

    //Bloc 3
    $req_etud->execute();

    //Bloc 4
?>

    <ul>
        <?php
        while ($donnees = $req_etud->fetch())
        {
        ?>
           <li><?php echo $donnees['nom'];?> (<?php echo $donnees['no_etud'];?>)</li>
        <?php
        }
        ?>
    </ul>

<?php
    //Bloc 5
    $req_etud->closeCursor();

    //Bloc 6
    $bdd=null;
?>
```

5. Ressources en lien avec le cours

Matériel de cours

- *Notes de cours [cf. sci6306_cours12_notes.pdf]*