

# Tutoriel de site dynamique - Classes d'abstraction

par Guillaume Rossolini ([Tutoriels Web](#))

Date de publication : 19 mai 2006

Dernière mise à jour : 1 septembre 2006

Pierre-Baptiste Naigeon a écrit un tutoriel pour apprendre à concevoir un site au menu dynamique. Le tutoriel qui suit est l'étape suivante dans l'élaboration d'un site Web : utiliser des **classes** d'**abstraction**. Je me contenterai de reprendre l'exemple qu'il donne sans le modifier en substance : je ne prévois d'en modifier que l'organisation. Le produit final sera 100% identique à l'original mais le code sera mieux organisé, ce qui est fondamental pour permettre à un projet d'évoluer.

- I - Introduction
  - I-A - Remerciements
  - I-B - Prélude
  - I-C - Problématique
  - I-D - Que sont les classes d'abstraction ?
  - I-E - Arborescence dans le système de fichiers
- II - La base de données
  - II-A - Introduction
  - II-B - Principe
  - II-C - Procédure
- III - Les gabarits
  - III-A - Introduction
  - III-B - Gabarit fondamental
  - III-C - Script PHP fondamental
  - III-D - Les "blocks"
  - III-E - Le pied de page
- IV - Les langues
  - IV-A - Le texte "hard-coded"
  - IV-B - La base de données
- V - Conclusion
  - V-A - Épilogue
  - V-B - Liens

## I - Introduction

### I-A - Remerciements

Je tiens à remercier **mathieu**, **Jérôme Usal** et **MrDuChnok** pour leurs conseils, ainsi bien sûr que **Pierre-Baptiste Naigeon** pour avoir fourni le tutoriel de départ.


### I-B - Prélude

Ce tutoriel traite de l'organisation d'un site Web en séparant au maximum les couches impliquées, ce qui permet de rendre le code aisément maintenable. Je n'ai pas la prétention de mettre des exemples pour toutes les solutions possibles (car elles sont nombreuses). Non, je me contenterai de vous mettre sur la voie en vous donnant un exemple concret.

Avant de commencer, voyons ce que sont les classes d'abstraction et pourquoi il peut être intéressant de les utiliser.

L'article de Pierre-Baptiste Naigeon auquel je me réfère se trouve ici :

 <http://pbnaigeon.developpez.com/tutoriel/PHP/conception-site-dynamique/>

 *Je ne remettrai absolument pas en cause ses choix. Je me contenterai ici de réécrire son code en utilisant des techniques d'abstraction. La sécurité doit être pensée dès le départ, sous peine d'être inefficace. La réécriture du code que nous allons effectuer serait un bon moment pour y porter son attention mais ce n'est pas mon propos.*

### I-C - Problématique

Lorsque nous créons un site Web en PHP, nous nous rendons généralement compte qu'il y a deux éléments étrangers au code PHP. Il s'agit de la source de données (fichiers, BDD, etc.) et du destinataire des données (page Web, image, fichier PDF, animation Flash, etc.).


PHP permet de tout gérer mais cela donne un mélange détonnant, dans le code source. C'est malgré tout (et malheureusement) la méthode suivie par un très grand nombre de développeurs.

Les éditeurs de code pallient parfois ce problème en proposant une colorisation syntaxique intelligente. Cependant, c'est peu satisfaisant.

Je propose de réfléchir à séparer toutes les couches : cela aurait le mérite de rendre le code source du site Web plus facile à maintenir.



Il s'agit de déléguer les tâches qui conviennent peu à PHP. C'est un peu comme déléguer la présentation aux feuilles de style plutôt que de tout spécifier dans le code HTML.

### Quelles sont les couches d'un site Web, du point de vue du navigateur ?


- La structure de la page et le contenu ( **HTML**,  **XHTML**,  **XML**, etc.);
- La présentation du contenu ( **CSS**,  **XSL**, etc.).

Lorsque nous utilisons une feuille de style CSS, nous faisons exactement la même chose que ce que je m'appête à expliquer : déléguer une tâche.




## Quelles sont les couches d'un site Web, du point de vue du serveur ?

- Les informations brutes : la base de données (MySQL, PostgreSQL, Oracle, etc.) ;
- Les traitements : le langage de script ( **PHP**,  **ASP.NET**, etc.) ;
- L'affichage : la page Web affichée par le navigateur (HTML, XHTML, XML, etc.).

Il est clair ici que chaque couche peut être gérée par une technologie ou une autre. Nous allons nous situer dans la couche centrale en utilisant PHP.

Pour nous, les deux autres couches seront des entités abstraites. Nous allons nous détacher des cas particuliers en utilisant des  **objets** qui se chargeront d'effectuer les tâches comme bon leur semble. Comment ils le feront ne sera pas notre préoccupation.

## I-D - Que sont les classes d'abstraction ?

Les  **classes** d' **abstraction** sont les classes qui nous permettent d'instancier les  **objets** que nous allons utiliser.

Dans notre cas, puisque nous avons deux couches desquelles nous souhaitons nous détacher, nous allons utiliser deux classes d'abstraction. En réalité, je donnerai plusieurs exemples de chaque cas pour que vous puissiez faire votre choix, mais l'application finale n'en utilisera qu'une de chaque : il s'agit d'une classe d'abstraction de base de données et d'un moteur de templates.

La classe d'abstraction de la base de données permet d'extraire les informations de la base de données sans utiliser de fonctions spécifiques à un système de gestion de base de données. Nous les demandons, le moteur se charge de les récupérer en fonction du SGBD utilisé.

La classe moteur de gabarit permet d'envoyer nos informations à la page de destination sans à avoir à nous préoccuper de l'organisation de ces informations dans la page de destination. Nous les envoyons, le moteur se charge de les organiser en fonction du gabarit défini.

Je vais en profiter pour présenter une méthode permettant de rendre une application multilingue.

## I-E - Arborescence dans le système de fichiers

Pierre-Baptiste a parlé de l'arborescence du menu dans le site. Pour ma part, je vais traiter de l'arborescence des fichiers que nous enregistrons sur le serveur.


### Le répertoire du site


- **/** - Répertoire racine ;
- **/includes** - Répertoire des scripts à inclure ;
- **/language** - Répertoire des fichiers de langue ;
- **/templates** - Répertoire des différents gabarits

## II - La base de données

### II-A - Introduction

Il existe de nombreuses classes d'abstraction de bases de données. Parmi les plus connues, on compte *PEAR::DB* et *PDO* ; de plus, chaque collection de scripts (comme *phpBB*) a développé sa propre classe... Je vais vous proposer d'utiliser **PDO** (PHP Data Objects).

 *PDO est disponible à partir de PHP 5.0 ou 5.1 suivant votre système d'exploitation.*

 *Pour charger l'extension, il faut activer deux lignes de votre fichier `php.ini` : "`extension=php_pdo`" ainsi que "`extension=php_pdo_mysql`" (suivant votre système de gestion de base de données, ici MySQL).*

### II-B - Principe

Commençons par déterminer la problématique.

Schéma classique d'interrogation d'une base de données avec MySQL :

```
<?php

$id = 10;

mysql_connect("server", "login", "password")
    or die(basename(__FILE__)."<br />".__LINE__."<br />".mysql_error());
mysql_select_db("database")
    or die(basename(__FILE__)."<br />".__LINE__."<br />".mysql_error());

$sql = "SELECT `field` FROM `table` WHERE `id` < ".$id;
$result = mysql_query($sql)
    or die(basename(__FILE__)."<br />".__LINE__."<br />".mysql_error());

while($row = mysql_fetch_assoc($result)
{
    echo "field: ".$row["field"]."<br />";
}

mysql_close();

?>
```

Le problème est le suivant : si je souhaite donner ce script à quelqu'un ou bien si je change de serveur de bases de données, je serai contraint de réécrire tout le code qui concerne la base de données (puisque les fonctions MySQL ne seront plus disponibles).

La solution est d'utiliser ce que l'on appelle un *wrapper*, c'est-à-dire des fonctions ou une classe qui me permettent de faire la même chose sans appeler aucune fonction spécifique à un système en particulier (ici : MySQL).

Schéma classique d'interrogation d'une base de données avec PDO :

```
<?php

$id = 10;
try
{
```

Schéma classique d'interrogation d'une base de données avec PDO :

```
$db = new PDO("mysql:host=server;dbname=database", "login", "password");

$sql = 'SELECT `field` FROM `table` WHERE `id` < :id';
$stmt = $db->prepare($sql);
$stmt->execute(array(":id" => $id));

if($rows = $stmt->fetchAll())
{
    foreach($rows as $row)
    {
        echo "field: ".$row["field"]."<br />";
    }
}

$db = NULL;
}
catch (PDOException $exception)
{
    die("Erreur : " . $exception->getMessage() . "<br/>");
}
?>
```

Avec cette technique, un changement de SGBD se fait simplement en modifiant l'instanciation de l'objet de la classe PDO !

## II-C - Procédure

Quelle est la procédure pour réécrire un script ?

Il faut trouver tous les appels à des fonctions spécifiques au SGBD utilisé et les remplacer par leur équivalent en utilisant la classe d'abstraction.

Trouver dans le fichier : index.php

```
connexion_DB("developpez");
```

Remplacer par :

```
define("BDD_CONNECTION", "mysql:host=localhost;dbname=BASE_TEST");

set_exception_handler("exception_handler");
$db = new PDO(BDD_CONNECTION, "root", "");
```

Nous venons d'instancier l'objet de base de données. C'est cet objet **\$db** que nous utiliserons tout au long du script.

Nous avons aussi assigné la gestion des erreurs à une fonction. Cela nous permet d'éviter d'écrire les blocs *try-catch* comme ci-dessus. Chacun sa méthode de faire : ici, j'ai opté pour le plus simple. Dans une application réelle, il est conseillé d'attraper l'exception et de la gérer correctement. D'une manière ou d'une autre, il est fortement conseillé de gérer l'erreur (au moyen du gestionnaire ou bien au moyen de *try-catch*) car, en cas d'erreur non gérée, PHP afficherait toutes les informations nécessaires au débogage (y compris le mot de passe de l'utilisateur de la base de données) !

La fonction de gestion d'erreurs est la suivante :

Ajouter dans le fichier : mes-fonctions.php

```
function exception_handler($exception)
```

Ajouter dans le fichier : mes-fonctions.php

```
{
    $message = "Wrong query : " . $exception->getMessage() . "<br>\n";
    $message .= "Please, send this message to the webmaster";
    die($message);
}
```

Nous n'aurons plus besoin de la fonction "connexion\_DB", nous pouvons donc la supprimer. Il en va de même pour la fonction "deconnexion\_DB".

Voici maintenant comment procéder à l'exécution d'une requête :

Trouver dans le fichier : mes-fonctions.php

```
$strSQL = "SELECT * FROM PAGES WHERE Id_page = ".$_ENV["id_page"];
$resultat = requete_SQL($strSQL);
$tabl_result = mysql_fetch_array($resultat);
```

Remplacer par :

```
global $db;

$sql = "SELECT * FROM PAGES WHERE Id_page = :id_page";
$stmt = $db->prepare($sql);
$stmt->execute(array(":id_page" => $_ENV["id_page"]));
$tabl_result = $stmt->fetch();
```

Le processus est très similaire à ce que nous connaissons déjà. Il s'agit de préparer la requête, de l'exécuter et de récupérer les résultats. L'étape de préparation peut être évitée mais elle a l'avantage d'optimiser automatiquement la requête tout en protégeant contre les *injections SQL* : il serait dommage de s'en passer.

C'est lors de l'exécution que nous envoyons les paramètres à la requête. Cela a l'avantage de permettre de préparer une seule fois la requête mais en l'exécutant plusieurs fois avec des paramètres différents.

La méthode **fetch()** permet de récupérer uniquement l'enregistrement suivant, tandis que la méthode **fetchAll()** permet de tous les récupérer en une fois.

Il faut modifier toutes les requêtes de cette manière. C'est la première fois que je le fais et j'y suis parvenu sans être vraiment à ce que je fais, donc j'imagine que vous n'aurez pas trop de problèmes non plus.


Voici le code complet de Pierre-Baptiste converti à cette étape : [ [FTP](#) ] ou [ [HTTP](#) ].

## III - Les gabarits

### III-A - Introduction

Ici, il y a davantage de travail. En effet, pour afficher du HTML, nous n'utilisons aucune fonction spéciale. Il s'agit donc de mettre en place un système qui mette en évidence le fait que nous souhaitons afficher du HTML.

La démarche peut sembler complexe mais je vous assure qu'elle a d'énormes avantages, notamment de permettre d'extraire tout le code HTML du fichier PHP. Les avantages sont évidents : il est plus facile de trouver un bug dans le PHP (puisque'il n'y a plus de HTML parasite) et il est plus facile de modifier l'apparence de la page (puisque le code PHP n'est pas dans le même fichier).

Tout comme pour les classes d'abstraction de base de données, il existe de nombreux moteurs de  templates comme Smarty, PHPLib, etc. De plus, chaque collection de script telle que *phpBB* dispose là aussi de son propre moteur.

J'ai choisi d'utiliser le moteur de templates de phpBB2 car je l'utilise souvent et je l'apprécie. Je l'ai un peu modifié pour mes besoins : je lui fais gérer la conversion des variables en entités HTML. Pour cela, j'ai modifié la classe pour qu'elle appelle **htmlentities()** avec tous ses paramètres. La différence se fait lors de l'instanciation car il est possible de modifier le comportement par défaut (**ENT\_QUOTES** et **"iso-8859-1"**) :

#### Exemples de déclaration de l'objet \$template

```
$template = new Template("./templates/default/");  
$template = new Template("./templates/titoumimi", "ISO-8859-1");  
$template = new Template("./templates/titoumimi", "UTF-8", ENT_QUOTES);
```

Voici le source du moteur de templates, à décompresser et à placer dans le répertoire `"/includes"` : [ **FTP** ] ou [ **HTTP** ].

Pour terminer cette introduction, voici le schéma minimum d'utilisation d'un template :

```
/index.php  
define("TEMPLATE_FOLDER", "./templates/titoumimi");  
  
$template = new Template(TEMPLATE_FOLDER);  
$template->set_filenames(array("index" => "index.tpl"));  
  
// Mettre ici le code de la page  
  
$template->pparse("index");
```

### Globalement, les modifications à apporter aux scripts de Pierre-Baptiste sont :

- Les portions de code qui envoient du HTML et/ou du texte de manière répétitive seront remplacées par un appel à **assign\_block\_vars()** chacune ;
- Les portions de code qui n'envoient du HTML et/ou du texte qu'une fois dans la page seront remplacées par un ou plusieurs appels à **assign\_vars()**.

### III-B - Gabarit fondamental

Commençons par écrire notre gabarit général, c'est-à-dire la structure de la page d'index :

/templates/default/index.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <!-- Insère les mots-clés extraits de la DB dans les meta -->
  <META NAME="keywords" lang="fr" CONTENT="{HEAD_MOTS_CLES}">
  <!-- Insère la description extraite de la DB dans les meta -->
  <META NAME="Description" CONTENT="{HEAD_DESCRIPTION}">
  <meta http-equiv="Content-Type" content="text/html; charset={TPL_CHARSET}">
  <!-- Insère le titre extrait de la DB dans la balise correspondante -->
  <title>{HEAD_TITRE}</title>
  <link rel="stylesheet" type="text/css" href="{THEMES_FOLDER}/final.css">
</head>
<body>
  <div id="menu_horizontal">
  </div>
  <div id="chemin_fer">
  </div>
  <div id="bloc_central">
  <div id="menu_vertical">
  </div>
  <div id="contenu">
    {PAGE_CONTENU}
  </div>
</div>
```

## Il est intéressant de noter plusieurs choses :

- Nous n'avons plus aucune des balises `<?php et ?>` ;
- Toutes les portions `<?php echo $var; ?>` ont été remplacées par des sortes de variables entre accolades comme `{HEAD_TITRE}` ;
- J'ai laissé vides les blocs *DIV* pour le moment car ils ont besoin de davantage d'attention ;
- J'ai mis une variable `{TPL_CHARSET}` qui semble provenir de nulle part : c'est un cas particulier assigné par le moteur de templates (suite aux modifications apportées par mes soins) ;

## III-C - Script PHP fondamental

Il faut maintenant modifier le script PHP afin qu'il utilise notre gabarit :

/index.php

```
<?php

// Active tous les warning. Utile en phase de développement
// En phase de production, remplacer E_ALL par 0
error_reporting(E_ALL | E_STRICT);

define("BDD_CONNECTION", "mysql:host=localhost;dbname=developpez");
define("TEMPLATE_FOLDER", "./templates/titoumimi");
define("THEMES_FOLDER", "./themes/titoumimi");

// Inclusions diverses
include_once("./includes/mes-fonctions.php");
include_once("./includes/template.php");

// Construction de l'objet de la classe de gabarit
$template = new Template(TEMPLATE_FOLDER);
```

```
/index.php
```

```
$template->set_filenames(array("index" => "index.tpl"));

// Construction de l'objet de la classe de connexion à la base de données
set_exception_handler("exception_handler");
$db = new PDO(BDD_CONNECTION, "root", "");

// Définit l'Id de la page d'accueil (1 dans cet exemple)
// Pensez à le modifier si ce n'est pas le cas chez vous.
$id_page_accueil = 1;

// Récupère l'id de la page courante passée par l'URL
// Si non défini, on considère que la page est la page d'accueil
if (isset($_GET["id_page"]))
{
    $_ENV["id_page"] = $_GET["id_page"];
}
else
{
    $_ENV["id_page"] = $id_page_accueil;
}

// Extrait les informations correspondantes à la page en cours de la DB
extraction_infos_DB();

// Le reste des traitements viendra ici

// Envoi des variables 'globales' au gabarit
$template->assign_vars(array(
    "THEMES_FOLDER" => THEMES_FOLDER,
    "HEAD_MOTS_CLES" => $_ENV["mots_cles"],
    "HEAD_DESCRIPTION" => $_ENV["description"],
    "HEAD_TITRE" => $_ENV["titre"],
    "PAGE_CONTENU" => $_ENV["contenu"]
));

$template->pparse("index");

?>
```

## Il est intéressant de noter plusieurs choses :

- Nous ouvrons la balise `<?php` uniquement en début de script et la refermons avec `?>` uniquement en fin de script (nous n'avons plus de code HTML brut) ;
- Les divers affichages des variables de la superglobale `$_ENV`; ont été déplacés dans l'appel à `assign_vars()`, ce qui permet de les regrouper et donc de mieux les gérer.

### III-D - Les "blocks"

Ensuite, occupons-nous des blocs qui se répètent. Nous aurons besoin de notre objet **\$template** dans les deux fonctions concernées : **affiche\_chemin\_fer()** et **affiche\_menu()**. Je propose de l'inclure à la liste des globales comme nous l'avons fait pour **\$db** :

```
/includes/mes-fonctions.php
```

```
function affiche_chemin_fer($idpage)
{
    global $db, $template;
```

```
/includes/mes-fonctions.php
```

```
function affiche_menu($idpage)
{
    global $db, $template;
```

Tout d'abord, le chemin de fer :

```
/templates/index.tpl
<div id="chemin_fer">
  Vous &ecirc;tes ici :
<!-- BEGIN un_niveau -->
  -&gt; <a href="index.php?id_page={un_niveau.ID}">{un_niveau.TITRE}</a>
<!-- END un_niveau -->
</div>
```

Le moteur de templates saura qu'il devra répéter ce lien à chaque fois qu'on lui parlera du bloc "un\_niveau".

Voici le code PHP associé :

```
/includes/mes-fonctions.php
function affiche_chemin_fer($idpage)
{
    global $db, $template;

    // Si l'id de la page en cours est différent de 0
    // (0 = page parente de la page racine = inexistante)
    if($idpage != 0)
    {
        $pages = array();

        $sql = "SELECT * FROM PAGES WHERE Id_page = :id_page";
        $statement = $db->prepare($sql);

        do
        {
            // on récupère les informations de la page en cours dans la DB
            $statement->execute(array(":id_page" => $idpage));
            $tabl_results = $statement->fetch();

            // Envoi au template
            $pages[] = $tabl_results;

            $idpage = $tabl_results["Id_parent"];
        } while($idpage);

        if(!empty($pages))
        {
            $nb_pages = count($pages);
            for($i = $nb_pages-1; $i >= 0; --$i)
            {
                $template->assign_block_vars("un_niveau", array(
                    "ID" => $pages[$i]["Id_page"],
                    "TITRE" => $pages[$i]["Titre"]
                ));
            }
        }
    }
}
```

Le principe est d'appeler **assign\_block\_vars()** à chaque itération. Ici, j'ai déporté ces appels dans une autre boucle afin de pouvoir afficher le menu à l'envers (comme Pierre-Baptiste l'a fait dans son article). J'ai enlevé la récursivité pour me simplifier la tâche.

Au passage, nous pouvons noter que PDO nous permet de ne pas préparer la requête avant chaque exécution. En effet, une fois la requête préparée, elle peut être exécutée plusieurs fois à la suite avec des paramètres différents. Il faut simplement prendre garde à toujours récupérer tous les tuples avant de l'exécuter à nouveau.

Voyons ensuite les deux menus :

/templates/index.tpl

```
<div id="menu_horizontal">
<!-- BEGIN menu_horizontal -->
  <ul>
<!-- BEGIN un_niveau -->
  <li><a
    href="index.php?id_page={menu_horizontal.un_niveau.ID}">{menu_horizontal.un_niveau.TITRE}</a></li>
<!-- END un_niveau -->
  </ul>
<!-- END menu_horizontal -->
</div>
```

/templates/index.tpl

```
<div id="bloc_central">
  <div id="menu_vertical">
<!-- BEGIN menu_vertical -->
  <ul>
<!-- BEGIN un_niveau -->
  <li>
    <a
      href="index.php?id_page={menu_vertical.un_niveau.ID}">{menu_vertical.un_niveau.TITRE}</a>
  </li>
<!-- END un_niveau -->
  </ul>
<!-- END menu_vertical -->
</div>
  <div id="contenu">
    {PAGE_CONTENU}
  </div>
</div>
```

/includes/mes-fonctions.php

```
function affiche_menu($idpage, $menu)
{
  global $db, $template;

  $sql = "SELECT * FROM PAGES WHERE Id_parent = :id_page";
  $statement = $db->prepare($sql);
  $statement->execute(array(":id_page" => $idpage));
  $tabl_results = $statement->fetchAll();

  // Si la page n'a pas de page fille, alors on modifie la requête pour obtenir ses pages soeurs.
  if (count($tabl_results) == 0)
  {
    $statement->execute(array(":id_page" => $_ENV["id_parent"]));
    $tabl_results = $statement->fetchAll();
  }

  if(count($tabl_results))
  {
    $template->assign_block_vars($menu, array());

    foreach($tabl_results as $tabl_result)
    {
      $template->assign_block_vars($menu."un_niveau", array(
        "ID" => $tabl_result["Id_page"],
        "TITRE" => $tabl_result["Titre"]
      ));
    }
  }
}
```

```
/includes/mes-fonctions.php
```

```
    }  
  }  
}
```

Ici, nous devons imbriquer deux blocs : le bloc racine ("**menu\_horizontal**" ou "**menu\_vertical**") permet d'initialiser la liste à puces dans le template et le bloc interne ("**un\_niveau**") contient les éléments de cette liste.

Il faut maintenant appeler les fonctions de Pierre-Baptiste pour que nos blocks soient générés :

```
/index.php
```

```
// Le reste des traitements viendra ici  
affiche_menu($id_page_accueil, 'menu_horizontal');  
affiche_menu($_ENV['id_page'], 'menu_vertical');  
affiche_chemin_fer($_ENV['id_page']);
```

### III-E - Le pied de page

Terminons par le pied de page, ultra simple :

```
/templates/default/pied-page.tpl
```

```
<hr />  
Ceci est le pied de page...  
</body>  
</html>
```

```
/pied-page.php
```

```
<?php  
  
$template->set_filenames(array("pied-page" => "pied-page.tpl"));  
$template->pparse("pied-page");  
  
?>
```

Voici le code complet de Pierre-Baptiste converti à cette étape : [ [FTP](#) ] ou [ [HTTP](#) ].

## IV - Les langues

Dans cette partie, je vais vous présenter la méthode que j'utilise pour traduire un site Web.

### IV-A - Le texte "hard-coded"

Dans l'exemple de Pierre-Baptiste, il y a plusieurs messages inclus directement dans le code. Nous allons créer des fichiers de langue pour les contenir :

```
/language/fr.php
```

```
<?php  
  
define("L_YOURE_HERE", "Vous êtes ici : ");  
define("L_FOOTER", "Ceci est le pied de page...");  
define("L_DIE", "Mauvaise requête : %s<br />\nVeuillez envoyer ce message à un  
administrateur");  
  
?>
```

```
/language/en.php
```

```
<?php  
  
define("L_YOURE_HERE", "You are here: ");  
define("L_FOOTER", "This is the footer...");  
define("L_DIE", "Wrong query: %s<br />\nPlease send this message to the webmaster");  
  
?>
```

Vous remarquerez l'utilisation du préfixe "L\_" : cela permet de s'assurer que les constantes que nous définissons ici n'entreront pas en conflit avec d'autres constantes. Cela permet également d'utiliser les fonctionnalités d'autocomplétion de code de votre éditeur favori.

Il faut évidemment inclure ce fichier dans notre script :

```
/index.php
```

```
include_once("../includes/mes-fonctions.php");  
include_once("../includes/template.php");  
include_once("../language/fr.php");
```

L'intérêt de cette méthode est de permettre de changer la langue simplement en incluant tel fichier (de langue) ou tel autre. Vous pourriez par exemple conserver l'information de langue dans une session ou dans un profil utilisateur.

Il faut maintenant utiliser les nouvelles variables :

```
/templates/default/index.tpl
```

```
<div id="chemin_fer">  
{L_YOURE_HERE}
```

```
/templates/default/pied-page.tpl
```

```
<hr />  
{L_FOOTER}
```

```


/index.php
// Envoi des variables 'globales' au gabarit
$template->assign_vars(array(
    "THEMES_FOLDER" => THEMES_FOLDER,
    "HEAD_MOTS_CLES" => $_ENV["mots_cles"],
    "HEAD_DESCRIPTION" => $_ENV["description"],
    "HEAD_TITRE" => $_ENV["titre"],
    "PAGE_CONTENU" => $_ENV["contenu"],
    "L_YOURE_HERE" => L_YOURE_HERE,
    "L_FOOTER" => L_FOOTER
));

```

```

/includes/mes-fonctions.php
function exception_handler($exception) {
    if(defined("L_DIE"))
    {
        die(sprintf(L_DIE, $exception->getMessage()));
    }
    else
    {
        die($exception->getMessage());
    }
}

```

 À ce stade, le code du site proposé par Pierre-Baptiste est complètement réécrit. L'objectif que je m'étais proposé en début d'article est atteint. Ce qui vient ensuite est un bonus, une réflexion plus poussée.

Voici le code complet de Pierre-Baptiste converti jusqu'à cette étape : [ **FTP** ] ou [ **HTTP** ].

## IV-B - La base de données

"Oui, mais là il reste encore plein de mots en français !"

Ahhh, en effet. Nous n'avons pas traduit la base de données...

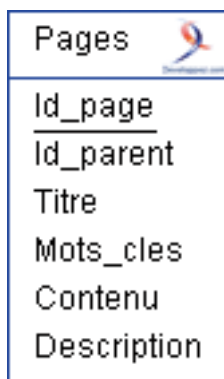
Traduire la base de données implique une charge de travail importante en permanence. En effet, il faut enregistrer chaque message dans chaque langue. Imaginez un site d'annonces dans cinq langues : vos utilisateurs devraient envoyer chaque annonce (titre + contenu) dans ces cinq langues, sans quoi elles ne seraient pas affichées par tous les visiteurs. Quelle galère !

Je vous conseille fortement de bien réfléchir à ce que cela implique.

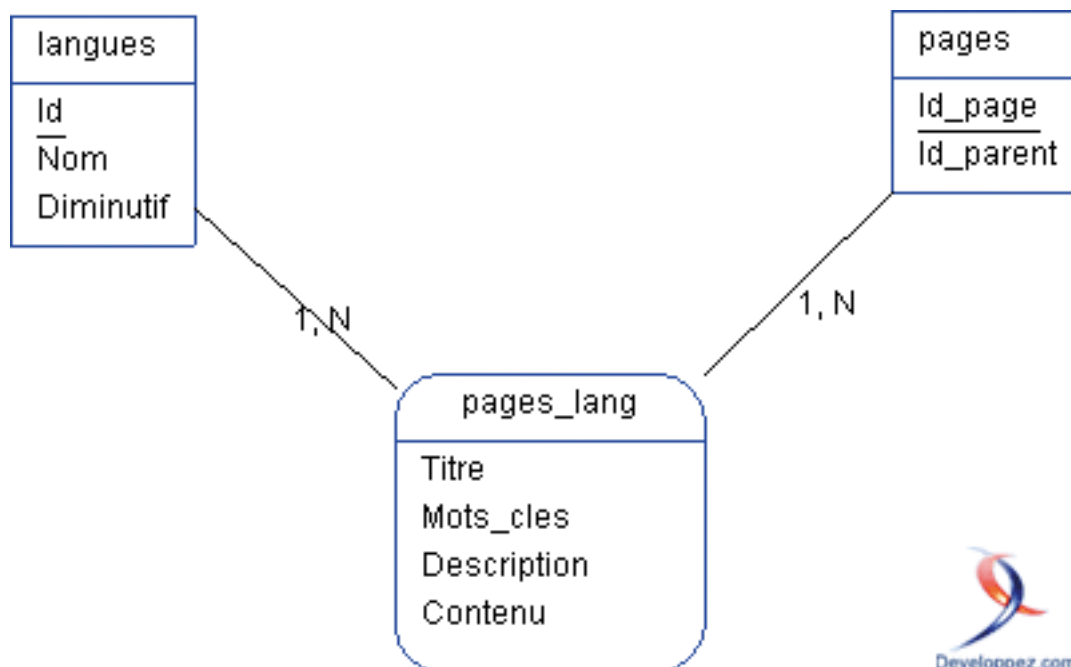
Anecdote : j'ai développé un site qu'un client voulait en plusieurs langues. Résultat : les produits sont ok dans sa langue maternelle mais cela fait plusieurs mois que les traductions sont vides. Il est parfois préférable de revoir ses ambitions à la baisse...

Bref.

Voici comment je m'y prends : je définis une table "langage" dans la base de données. Ensuite, je sépare les champs de type texte de chaque table pour laquelle je souhaite proposer une traduction. Il vous faudra des notions de bases de données relationnelles pour aborder cette partie, notamment ce qui touche aux clés étrangères.



*Avant : Schéma de la base de données de Pierre-Baptiste*



*Maintenant : Schéma de la base de données avec mes modifications*

Exemple de modification

Nous souhaitons traduire la page "Accueil" (*Id\_page=1, Id\_parent=0*) en français (*Id\_langue=1*) et en anglais (*Id\_langue=2*).

Il nous faut utiliser la nouvelle table *pages\_lang* pour effectuer la jointure entre la *page* et la *langue*. Nous y introduisons donc les identifiants des tuples à relier (il y a deux langues donc deux étapes) : ***Id\_page=1*** et ***Id\_langue=1*** avec ***Titre="Accueil"***, puis ***Id\_page=1*** et ***Id\_langue=2*** avec ***Titre="Home"***.

Voici maintenant la totalité des modifications

**SQL de création de la base de données**

```

CREATE TABLE `langues` (
  `Id_langue` int(11) NOT NULL auto_increment,
  `Nom` varchar(255) NOT NULL,
  `Diminutif` varchar(5) NOT NULL,

```

## SQL de création de la base de données

```
PRIMARY KEY (`Id`)
);

CREATE TABLE `pages` (
  `Id_page` int(11) NOT NULL auto_increment,
  `Id_parent` int(11) NOT NULL default '1',
  PRIMARY KEY (`Id_page`)
);

CREATE TABLE `pages_lang` (
  `Id_page` int(11) NOT NULL,
  `Id_langue` int(11) NOT NULL default '1',
  `Titre` varchar(255) NOT NULL,
  `Mots_cles` varchar(255) NOT NULL,
  `Description` varchar(255) NOT NULL,
  `Contenu` TEXT NOT NULL,
  PRIMARY KEY (`Id_page`,`Id_langue`)
);

INSERT INTO `langues`
(`Id_langue`,`Nom`,`Diminutif`)
VALUES
(1, 'Français', 'fr'),
(2, 'English', 'en');

INSERT INTO `pages`
(`Id_page`,`Id_parent`)
VALUES
(1, 0),
(2, 1),
(3, 1),
(4, 1),
(5, 1),
(6, 2),
(7, 2),
(8, 2),
(9, 3),
(10, 3),
(11, 10),
(12, 10);

INSERT INTO `pages_lang`
(`Id_page`,`Id_langue`,`Titre`,`Mots_cles`,`Description`,`Contenu`)
VALUES
(1, 1, 'Accueil', 'Accueil', 'Accueil', 0x4163637565696c),
(1, 2, 'Home', 'Home', 'Home', 0x486f6d65),
(2, 1, 'Mon corps de rêve', 'Mon corps de rêve', 'Mon corps de rêve',
0x4d6f6e20636f7270732064652072c3aa7665),
(2, 2, 'My heavenly body', 'My heavenly body', 'My heavenly body',
0x4d792068656176656e6c7920626f6479),
(3, 1, 'Mes vacances', 'Mes vacances', 'Mes vacances', 0x4d657320766163616e636573),
(3, 2, 'My holidays', 'My holidays', 'My holidays', 0x4d7920686f6c6964617973),
(4, 1, 'Mes loisirs', 'Mes loisirs', 'Mes loisirs', 0x4d6573206c6f6973697273),
(4, 2, 'My hobbies', 'My hobbies', 'My hobbies', 0x4d7920686f6262696573),
(5, 1, 'Me contacter', 'Me contacter', 'Me contacter', 0x4d6520636f6e746163746572),
(5, 2, 'Contact me', 'Contact me', 'Contact me', 0x436f6e74616374206d65),
(6, 1, 'Mes abdos Kro', 'Mes abdos Kro', 'Mes abdos Kro', 0x4d6573206162646f73204b726f),
(6, 2, 'My Kro muscles', 'My Kro muscles', 'My Kro muscles', 0x4d79204b726f206d7573636c6573),
(7, 1, 'Mes mains', 'Mes mains', 'Mes mains', 0x4d6573206d61696e73),
(7, 2, 'My hands', 'My hands', 'My hands', 0x4d792068616e6473),
(8, 1, 'Mon profil Grec', 'Mon profil Grec', 'Mon profil Grec', 0x4d6f6e2070726f66696c2047726563),
(8, 2, 'My greek profile', 'My greek profile', 'My greek profile',
0x4d7920677265656b2070726f66696c65),
(9, 1, 'Au ski', 'Au ski', 'Au ski', 0x417520736b69),
(9, 2, 'Skying', 'Skying', 'Skying', 0x536b79696e67),
(10, 1, 'A la mer', 'A la mer', 'A la mer', 0x41206c61206d6572),
(10, 2, 'At the beach', 'At the beach', 'At the beach', 0x417420746865206265616368),
```

**SQL de création de la base de données**

```
(11, 1, 'Mer du Nord', 'Mer du Nord', 'Mer du Nord', 0x4d6572206475204e6f7264),  
(11, 2, 'North Sea', 'North Sea', 'North Sea', 0x4e6f72746820536561),  
(12, 1, 'Mer Morte', 'Mer Morte', 'Mer Morte', 0x4d6572204d6f727465),  
(12, 2, 'Dead Sea', 'Dead Sea', 'Dead Sea', 0x4465616420536561);
```

Alors oui, évidemment, cela fait des chiffres partout, ce n'est pas lisible.

Vous savez quoi ? Une base de données n'est pas faite pour être lisible à l'oeil nu. Elle doit contenir des données aussi segmentées que possible. C'est aux requêtes d'assembler les données de manière à leur donner du sens

C'est ici qu'il devient particulièrement intéressant de conserver dans une session les informations sur la langue en cours.

Voici le code complet de Pierre-Baptiste converti à cette étape : [ [FTP](#) ] ou [ [HTTP](#) ].

## V - Conclusion

### V-A - Épilogue

#### Voici ce à quoi nous sommes parvenus :

- Nous utilisons une base de données sans pour autant utiliser aucune fonction spécifique à un SGBD ;
- Notre base de données ne contient que des informations élémentaires qu'il serait futile de chercher à segmenter davantage ;
- Nous envoyons les informations de la base de données à un moteur de templates qui se charge de tout l'affichage à l'aide de gabarits ;
- Nous avons des scripts PHP dont le code n'est pas parasité par des incises en HTML ;
- Nous avons des gabarits écrits en HTML pur, que nous pourrions aisément donner à un graphiste ;
- Nos gabarits ne contiennent ni information ni même texte : ils ne contiennent que la structure du document HTML final et permettent uniquement de savoir où sont positionnés les éléments ;
- Nous avons une feuille de style CSS pour gérer la présentation (par opposition à "structure") des informations dans notre page Web ;
- **Simplement en modifiant quelques petites constantes, nous avons la possibilité de changer le SGBD, le gabarit, le style et la langue de notre site !**

Je souhaite attirer votre attention sur un élément supplémentaire qui me tient à coeur : l'organisation du code.

Lorsque l'on utilise ce genre de classes d'abstraction, il est généralement intéressant d'y réfléchir afin de ne pas s'y perdre.





#### Personnellement, je vous recommande cette structure pour chacun de vos scripts :

- Inclure un script **common.php** qui s'occupe d'inclure tous les scripts nécessaires (configuration, classes, fonctions), d'instancier les classes comme PDO et Template, de déterminer la langue par défaut (en incluant le fichier correspondant) et d'effectuer les éventuelles modifications de la base de données (tous types de requêtes sauf *SELECT*) ;
- Inclure un script **header.php** couplé à un gabarit **header.tpl** pour avoir un affichage d'en tête identique sur toutes vos pages sans à avoir à répéter le code ;
- Le code de votre script actuel ;
- Inclure un script **footer.php** couplé à un gabarit **footer.tpl** pour avoir un affichage de pied de page identique sur toutes vos pages sans à avoir à répéter le code.

À titre de démonstration, voici une version plus ou moins finale du projet de site ci-dessus : [ [FTP](#) ] ou [ [HTTP](#) ]. Il ne vous reste plus qu'à gérer tout cela à l'aide de sessions et vous aurez un site totalement dynamique !

## V-B - Liens

### Tutoriels de Developpez

-  [Exemple de conception d'un site dynamique](#) par Pierre-Baptiste Naigeon ;
-  [Tutoriel : Abstraction de l'accès aux données avec PEAR::MDB2](#) par Hugo ;
-  [Comparatif des systèmes de template pour PHP](#) par Hugo Étievant ;
-  [Le système de template de phpBB](#) par Genova.

## Ressources externes

-  [Documentation officielle de PDO](#) ;
-  [Information on how the phpBB Template system works.](#)

