

	INSTITUT SUPERIEUR D'INFORMATIQUE DU KEF المعهد العالي للإعلامية بالكاوف	Technologies et programmation Web
TP n°7 : CRUD avec PHP, PDO et MySQL		

Objectifs :

- Créer des enregistrements MySQL.
- Lire les enregistrements MySQL et les afficher dans un tableau HTML.
- Mettez à jour les enregistrements MySQL existants.
- Confirmer et Supprimer les enregistrements MySQL.
- Envoyez des données à notre application à partir d'un formulaire HTML et de paramètres d'URL (Requêtes GET et POST).
- Sécurisez nos instructions SQL avec des instructions préparées

Rappel :

- **CRUD** est un acronyme pour **Create**, **Read**, **Update** et **Delete** (Créer, Lire, Mettre à jour et Supprimer). C'est un ensemble d'opérations de base qui peuvent être effectuées sur une base de données.
- En programmation PHP, une extension est un module logiciel qui étend les fonctionnalités de base de PHP. Les extensions sont généralement fournies sous la forme de bibliothèques dynamiques (fichiers .dll sous Windows ou .so sous Linux), qui sont chargées en mémoire lorsque PHP démarre ou lorsque l'extension est utilisée pour la première fois dans un script PHP. Les extensions PHP peuvent fournir des fonctionnalités supplémentaires telles que la prise en charge de bases de données, la manipulation de fichiers, le cryptage, la communication réseau, etc. Les extensions les plus couramment utilisées dans PHP incluent MySQL, SQLite, cURL, GD (pour la manipulation d'images), OpenSSL (pour la sécurité), et bien d'autres.
- Les différences entre **PDO** (PHP Data Objects) et **MySQLi** :

PDO (PHP Data Objects)	MySQLi
PDO (PHP Data Objects) et MySQLi sont deux extensions PHP qui permettent aux développeurs de se connecter à une base de données MySQL à partir de PHP.	
est une extension orientée objet	est une extension qui propose à la fois une interface orientée objet et procédurale.
PDO prend en charge plusieurs types de bases de données.	MySQLi est spécifiquement conçu pour MySQL.
PDO est considéré comme plus sûr que MySQLi car il prend en charge les requêtes préparées et liées, ce qui permet de réduire le risque d'injection SQL.	MySQLi est considéré comme moins sûr que PDO.

Tableau 1: les différences entre PDO et MySQLi

NB :

Le choix entre PDO et MySQLi dépend des besoins spécifiques du projet. Si vous avez besoin d'une portabilité maximale ou si vous travaillez avec plusieurs types de bases de données, PDO peut être le meilleur choix. Si vous travaillez uniquement avec MySQL et que vous avez besoin d'une interface orientée objet et procédurale, MySQLi peut être une bonne option.

1. Mise en route

Avant de nous lancer dans la programmation de notre application CRUD, nous devons activer notre serveur Web et configurer notre application.

1.2. Exigences

- **Serveur Web** : le package de serveur XAMPP comprend MySQL, PHP, phpMyAdmin et l'extension PDO.
- **PHP** : utiliser la dernière version de PHP, mais les anciennes versions devraient fonctionner correctement (ignorez si vous avez installé XAMPP).
- **Extension PDO** : Doit être activée par défaut si vous utilisez XAMPP, mais si ce n'est pas le cas, vous devrez l'activer/l'installer. Pour vérifier si le pilote PDO MySQL est activé, vous devez ouvrir le fichier **php.ini** et décommenter les lignes suivantes en supprimant le point-virgule (;) au début de l'entrée:

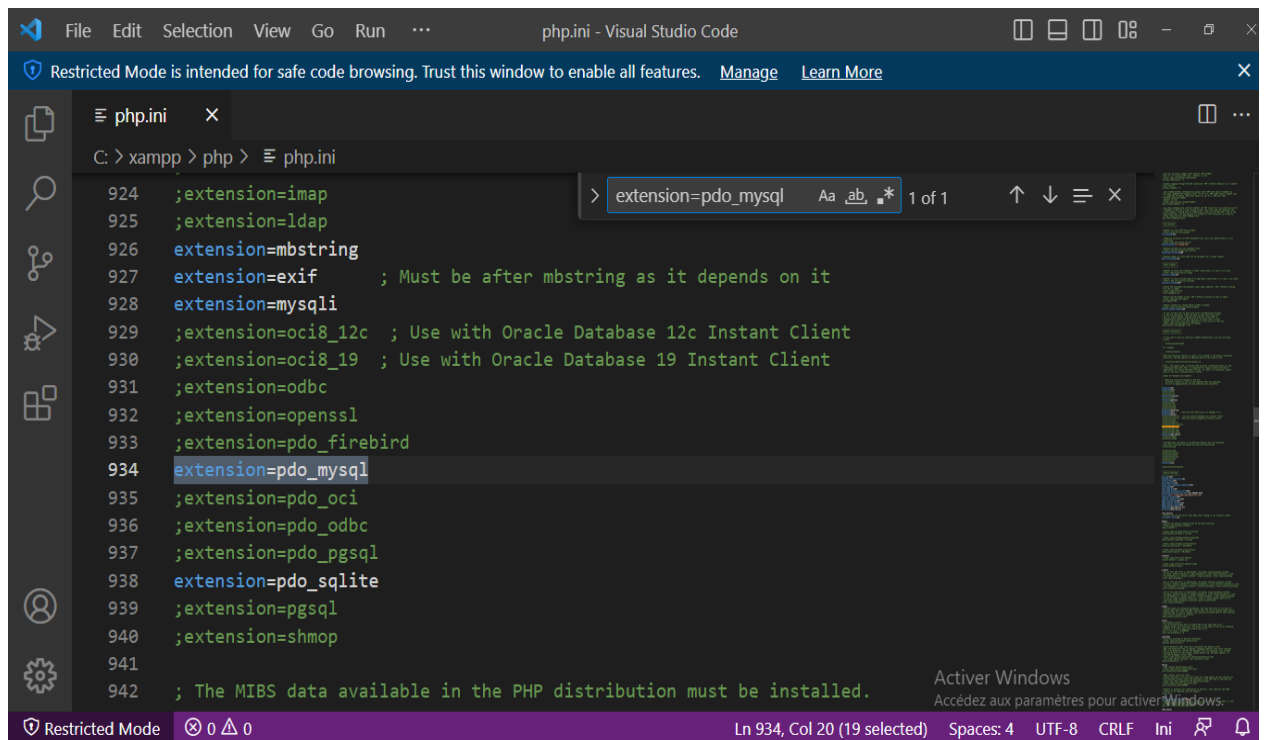


Figure 1: vérifier si le pilote PDO MySQL est activé

1.3. Structure et configuration des fichiers

Accédez à <C:\xampp\htdocs> (XAMPP) et créez les répertoires et fichiers ci-dessous.

Structure du fichier

```
\-- phpcrud
|-- index.php
|-- create.php
|-- read.php
|-- update.php
|-- delete.php
|-- functions.php
|-- style.css
```

Ce que chaque fichier contiendra :

- [index.php](#) : Page d'accueil de notre application CRUD.
- [create.php](#) : Créez de nouveaux enregistrements avec un formulaire HTML et envoyez des données au serveur avec une requête POST.
- [read.php](#) : Affichez les enregistrements de notre table de base de données et naviguez avec la pagination.

- [update.php](#) : Met à jour les enregistrements existants avec un formulaire HTML et envoie les données au serveur avec une requête POST.
- [delete.php](#) : Confirme et supprime les enregistrements par ID (requête GET pour obtenir l'ID).
- [functions.php](#) : Fonctions de base de modèles et fonction de connexion MySQL (nous n'avons donc pas à répéter le code dans chaque fichier).
- [style.css](#) : La feuille de style de notre application, cela changera l'apparence de notre application.

2. Création de la base de données et configuration des tables

Suivez les instructions ci-dessous.

- Accédez à <http://localhost/phpmyadmin/>
- Cliquez sur [Bases de données](#) en haut
- Sous [Créer une base de données](#), entrez [phpcrud](#) et sélectionnez [utf8_general_ci](#) comme collation
- Cliquez sur [Créer](#)
- Sélectionnez la base de données nouvellement créée
- Cliquez sur l'onglet [SQL](#) et exécutez le SQL ci-dessous :

```
CREATE TABLE IF NOT EXISTS `contacts` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `phone` varchar(255) NOT NULL,
  `title` varchar(255) NOT NULL,
  `created` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8;

INSERT INTO `contacts` (`id`, `name`, `email`, `phone`, `title`, `created`)
VALUES
```

```
(1, 'John Doe', 'johndoe@example.com', '2026550143', 'Lawyer', '2019-05-08
17:32:00'),

(2, 'David Deacon', 'daviddeacon@example.com', '2025550121', 'Employee',
'2019-05-08 17:28:44'),

(3, 'Sam White', 'samwhite@example.com', '2004550121', 'Employee', '2019-05-08
17:29:27'),

(4, 'Colin Chaplin', 'colinchaplin@example.com', '2022550178', 'Supervisor',
'2019-05-08 17:29:27'),

(5, 'Ricky Waltz', 'rickywaltz@example.com', '7862342390', '', '2019-05-09
19:16:00'),

(6, 'Arnold Hall', 'arnoldhall@example.com', '5089573579', 'Manager', '2019-
05-09 19:17:00'),

(7, 'Toni Adams', 'alvah1981@example.com', '2603668738', '', '2019-05-09
19:19:00'),

(8, 'Donald Perry', 'donald1983@example.com', '7019007916', 'Employee', '2019-
05-09 19:20:00'),

(9, 'Joe McKinney', 'nadia.doole0@example.com', '6153353674', 'Employee',
'2019-05-09 19:20:00'),

(10, 'Angela Horst', 'angela1977@example.com', '3094234980', 'Assistant',
'2019-05-09 19:21:00'),

(11, 'James Jameson', 'james1965@example.com', '4002349823', 'Assistant',
'2019-05-09 19:32:00'),

(12, 'Daniel Deacon', 'danieldeacon@example.com', '5003423549', 'Manager',
'2019-05-09 19:33:00');
```

Le SQL ci-dessus créera la table : [contacts](#), nous utiliserons cette table dans notre application, incluse dans le SQL est un exemple de données, ces données seront utilisées à des fins de test pour s'assurer que tout fonctionne comme il se doit, vous pouvez supprimer ça plus tard. Il y a 6 colonnes dans la table des [contacts](#) (id, nom, email, téléphone, titre et créé).

3. Création de la feuille de style (CSS3)

La feuille de style changera l'apparence de notre application, modifiera le fichier [style.css](#) et ajoutera le code suivant :

```
* {
  box-sizing: border-box;
  font-family: -apple-system, BlinkMacSystemFont, "segoe ui", roboto,
oxygen, ubuntu, cantarell, "fira sans", "droid sans", "helvetica neue", Arial,
sans-serif;
  font-size: 16px;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
body {
  background-color: #FFFFFF;
  margin: 0;
}
.navtop {
  background-color: #3f69a8;
  height: 60px;
  width: 100%;
  border: 0;
}
.navtop div {
  display: flex;
  margin: 0 auto;
  width: 1000px;
  height: 100%;
}
.navtop div h1, .navtop div a {
  display: inline-flex;
  align-items: center;
}
.navtop div h1 {
  flex: 1;
  font-size: 24px;
  padding: 0;
  margin: 0;
  color: #ecf0f6;
  font-weight: normal;
}
.navtop div a {
```

```
padding: 0 20px;
text-decoration: none;
color: #c5d2e5;
font-weight: bold;
}
.navtop div a i {
padding: 2px 8px 0 0;
}
.navtop div a:hover {
color: #ecf0f6;
}
.content {
width: 1000px;
margin: 0 auto;
}
.content h2 {
margin: 0;
padding: 25px 0;
font-size: 22px;
border-bottom: 1px solid #ebebeb;
color: #666666;
}
.read .create-contact {
display: inline-block;
text-decoration: none;
background-color: #38b673;
font-weight: bold;
font-size: 14px;
color: #FFFFFF;
padding: 10px 15px;
margin: 15px 0;
}
.read .create-contact:hover {
background-color: #32a367;
}
.read .pagination {
display: flex;
justify-content: flex-end;
```

```
}  
.read .pagination a {  
    display: inline-block;  
    text-decoration: none;  
    background-color: #a5a7a9;  
    font-weight: bold;  
    color: #FFFFFF;  
    padding: 5px 10px;  
    margin: 15px 0 15px 5px;  
}  
.read .pagination a:hover {  
    background-color: #999b9d;  
}  
.read table {  
    width: 100%;  
    padding-top: 30px;  
    border-collapse: collapse;  
}  
.read table thead {  
    background-color: #ebeeef;  
    border-bottom: 1px solid #d3dae0;  
}  
.read table thead td {  
    padding: 10px;  
    font-weight: bold;  
    color: #767779;  
    font-size: 14px;  
}  
.read table tbody tr {  
    border-bottom: 1px solid #d3dae0;  
}  
.read table tbody tr:nth-child(even) {  
    background-color: #fbfcfc;  
}  
.read table tbody tr:hover {  
    background-color: #376ab7;  
}  
.read table tbody tr:hover td {
```



```
    color: #FFFFFF;
}
.read table tbody tr:hover td:nth-child(1) {
    color: #FFFFFF;
}
.read table tbody tr td {
    padding: 10px;
}
.read table tbody tr td:nth-child(1) {
    color: #a5a7a9;
}
.read table tbody tr td.actions {
    padding: 8px;
    text-align: right;
}
.read table tbody tr td.actions .edit, .read table tbody tr td.actions .trash
{
    display: inline-flex;
    text-align: right;
    text-decoration: none;
    color: #FFFFFF;
    padding: 10px 12px;
}
.read table tbody tr td.actions .trash {
    background-color: #b73737;
}
.read table tbody tr td.actions .trash:hover {
    background-color: #a33131;
}
.read table tbody tr td.actions .edit {
    background-color: #37afb7;
}
.read table tbody tr td.actions .edit:hover {
    background-color: #319ca3;
}
.update form {
    padding: 15px 0;
    display: flex;
```

```
    flex-flow: wrap;
}
.update form label {
    display: inline-flex;
    width: 400px;
    padding: 10px 0;
    margin-right: 25px;
}
.update form input {
    padding: 10px;
    width: 400px;
    margin-right: 25px;
    margin-bottom: 15px;
    border: 1px solid #cccccc;
}
.update form input[type="submit"] {
    display: block;
    background-color: #38b673;
    border: 0;
    font-weight: bold;
    font-size: 14px;
    color: #FFFFFF;
    cursor: pointer;
    width: 200px;
    margin-top: 15px;
}
.update form input[type="submit"]:hover {
    background-color: #32a367;
}
.delete .yesno {
    display: flex;
}
.delete .yesno a {
    display: inline-block;
    text-decoration: none;
    background-color: #38b673;
    font-weight: bold;
    color: #FFFFFF;
}
```

```
padding: 10px 15px;
margin: 15px 10px 15px 0;
}
.delete .yesno a:hover {
    background-color: #32a367;
}
```

N'hésitez pas à changer le style, c'est ce que j'ai mis en place pour rendre l'application CRUD plus attrayante.

4. Création de l'application CRUD

4.1. Création des fonctions

Le fichier *functions.php* contiendra des fonctions que nous pourrons exécuter dans tous nos fichiers PHP, c'est pour ne pas avoir à écrire le même code dans chaque fichier PHP, plus le code est court, mieux c'est, n'est-ce pas ? Nous allons créer 3 fonctions, 1 fonction se connectera à la base de données, les 2 autres seront les modèles pour l'en-tête et le pied de page qui apparaîtront sur chaque page que nous créerons et contiendront la mise en page HTML.

Editez le fichier *functions.php* et ajoutez le code suivant :

```
<?php
//Fonction de configuration et de connexion à une base de données
MySQL
function pdo_connect_mysql() {
    // Configuration des informations de la base de données
    $DATABASE_HOST = 'localhost';
    $DATABASE_USER = 'root';
    $DATABASE_PASS = '';
    $DATABASE_NAME = 'phpcrud';
    //saisie d'une exception potentielle
    try {
        // Connexion à la base de données mysql avec PDO
```

```

        return new PDO('mysql:host=' . $DATABASE_HOST . ';dbname=' .
$DATABASE_NAME . ';charset=utf8', $DATABASE_USER, $DATABASE_PASS);
    } catch (PDOException $exception) {
        // S'il y a une erreur avec la connexion, arrêtez le script
et affichez l'erreur.
        exit('Failed to connect to database!');
    }
}

```

/*Le fonction template_header(\$title) est communément utilisé dans les sites web pour inclure le code HTML et les éléments visuels qui sont répétés sur toutes les pages du site, tels que la barre de navigation, le logo du site, les liens vers les réseaux sociaux, etc.*/

```

function template_header($title) {
    /* EOT est un délimiteur de chaîne de caractères en PHP.
    Il est utilisé pour définir une chaîne de caractères
multilignes appelée heredoc.*/
echo <<<EOT
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>$title</title>
        <link href="style.css" rel="stylesheet" type="text/css">
                                <link                rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.7.1/css/all.css">
    </head>
    <body>
        <nav class="navtop">
            <div>

```

```

        <h1>Website Title</h1>
        <a href="index.php"><i class="fas fa-home"></i>Home</a>
            <a href="read.php"><i class="fas fa-address-
book"></i>Contacts</a>
        </div>
    </nav>
EOT;
}

/*La fonction template_footer() est communément utilisé dans les
sites web pour inclure les éléments communs au pied de page
de chaque page du site, tels que les liens de navigation
secondaire,
les coordonnées de contact, etc.*/
function template_footer() {
echo <<<EOT
    </body>
</html>
EOT;
}
?>

```

Nous utilisons PDO pour nous connecter à MySQL, PDO nous facilitera l'interaction avec notre base de données MySQL.

4.2. Création de la page d'accueil

Lorsque vous naviguez vers <http://localhost/phpcrud/> il servira le fichier [index.php](#), cette page sera notre page d'accueil.

Modifiez le fichier [index.php](#) et ajoutez le code suivant :

```

<?php
include 'functions.php';
// Votre code PHP ici.

```

```
// Modèle de page d'accueil ci-dessous.

?>

<?=template_header('Home')?>

<!--<?php template_header('Home'); ?>-->

<div class="content">
    <h2>Home</h2>
    <p>Welcome to the home page!</p>
</div>

<?=template_footer()?>
```

Nous pouvons utiliser cette page pour naviguer vers les autres pages. Comme vous pouvez le voir, nous incluons le fichier *functions.php* et exécutons les fonctions de modèle que nous avons créées, rappelez-vous que ces fonctions ajouteront le code d'en-tête et de pied de page à notre page d'accueil.

Maintenant, si nous naviguons vers <http://localhost/phpcrud/> ou <http://127.0.0.1/phpcrud/>, nous verrons ce qui suit :



Figure 2:page index.php

Cette page est juste pour que nous puissions naviguer vers les autres pages, n'hésitez pas à ajouter votre propre contenu.

4.3. Création de la page de lecture

Cette page remplira les enregistrements de notre table *de contacts* dans une table HTML.

Modifiez le fichier *read.php* et ajoutez le code suivant :

```
<?php
include 'functions.php';
// Connexion à la base de données MySQL
$pdo = pdo_connect_mysql();
/* Récupère la page via la requête GET (param URL : page),
   si elle n'existe pas, la page est définie par défaut sur 1*/
$page = isset($_GET['page']) && is_numeric($_GET['page']) ?
(int)$_GET['page'] : 1;
// Nombre d'enregistrements à afficher sur chaque page
$records_per_page = 5;
```

Une fois de plus, nous incluons le fichier de fonctions, mais cette fois nous nous connectons à notre base de données MySQL en exécutant la fonction : **pdo_connect_mysql**. Nous créons 2 variables, la **\$page** déterminera la page sur laquelle l'utilisateur se trouve actuellement, la **\$records_per_page** sera utilisé pour limiter le nombre d'enregistrements à afficher sur chaque page, par exemple, si nous limitons le nombre d'enregistrements à 5 et que nous avons 10 enregistrements dans notre table *de contacts*, alors il n'y aura que 2 pages et 5 enregistrements sur chaque page, l'utilisateur pourra naviguer entre les pages.

Ajoutez le code suivant au fichier *read.php* :

```
// Préparez l'instruction SQL et obtenez les enregistrements de
notre table de contacts, LIMIT déterminera la page

/*

Cette ligne de code prépare une requête SQL pour sélectionner toutes
les données de la table "contacts",
```

triées par ordre croissant de l'identifiant (ID), avec une limitation du nombre de résultats retournés.

La limitation est déterminée par deux paramètres qui sont `":current_page"` : détermine la page de résultats actuelle à afficher,

en commençant par 0 pour la première page. `":record_per_page"` : détermine le nombre maximum de résultats à retourner par page.

```
*/
```

```
$stmt = $pdo->prepare('SELECT * FROM contacts ORDER BY id LIMIT :current_page, :record_per_page');
```

//effectuer une pagination dans une liste de résultats en récupérant les enregistrements correspondant à la page demandée

```
$stmt->bindValue(':current_page', ($page-1)*$records_per_page, PDO::PARAM_INT);
```

```
$stmt->bindValue(':record_per_page', $records_per_page, PDO::PARAM_INT);
```

```
$stmt->execute();
```

// Récupérez les enregistrements afin que nous puissions les afficher dans notre modèle.

```
$contacts = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Nous utilisons également une instruction préparée pour la requête ci-dessus, cela garantira que notre requête est sécurisée (échappe aux données d'entrée de l'utilisateur). La méthode **bindValue** lie une valeur à un paramètre nommé de la requête préparée. Dans ce cas, la valeur est calculée en fonction du numéro de page demandé (**\$page**) et du nombre de résultats à afficher par page (**\$records_per_page**), et est passée à la requête SQL pour récupérer les résultats correspondants. Plus précisément, la ligne de code lie la valeur "**(\$page-1)*\$records_per_page**" à un paramètre nommé **":current_page"** dans la requête préparée.

La constante **PDO::PARAM_INT** spécifie que la valeur liée est un entier (integer), afin d'éviter les risques d'injection SQL.

Ajoutez le code suivant au fichier *read.php* :

```
// Obtenez le nombre total de contacts, afin que nous puissions
déterminer s'il devrait y avoir un bouton suivant et précédent
$num_contacts = $pdo->query('SELECT COUNT(*) FROM contacts')-
>fetchColumn();
?>
```

La requête SQL ci-dessus obtiendra le nombre total d'enregistrements dans la table *des contacts*, nous n'avons pas besoin d'utiliser une instruction préparée ici car la requête n'inclut pas de variables d'entrée utilisateur.

Ajoutez le code suivant au fichier *read.php* :

```
<?=template_header('Read')?>

<div class="content read">
  <h2>Read Contacts</h2>
  <a href="create.php" class="create-contact">Create Contact</a>
  <table>
    <thead>
      <tr>
        <td>#</td>
        <td>Name</td>
        <td>Email</td>
        <td>Phone</td>
        <td>Title</td>
        <td>Created</td>
        <td></td>
      </tr>
    </thead>
    <tbody>
      <?php foreach ($contacts as $contact): ?>
        <tr>
```

```

        <td><?=$contact['id']?></td>
        <td><?=$contact['name']?></td>
        <td><?=$contact['email']?></td>
        <td><?=$contact['phone']?></td>
        <td><?=$contact['title']?></td>
        <td><?=$contact['created']?></td>
        <td class="actions">
            <a href="update.php?id=<?=$contact['id']?>"
class="edit"><i class="fas fa-pen fa-xs"></i></a>
            <a href="delete.php?id=<?=$contact['id']?>"
class="trash"><i class="fas fa-trash fa-xs"></i></a>
        </td>
    </tr>
<?php endforeach; ?>
</tbody>
</table>
<div class="pagination">
    <?php if ($page > 1): ?>
        <a href="read.php?page=<?=$page-1?>"><i class="fas fa-angle-double-
left fa-sm"></i></a>
    <?php endif; ?>
    <?php if ($page*$records_per_page < $num_contacts): ?>
        <a href="read.php?page=<?=$page+1?>"><i class="fas fa-angle-double-
right fa-sm"></i></a>
    <?php endif; ?>
</div>
</div>

<?=template_footer()?>

```

Ceci est le modèle de la page de lecture, le code itère les contacts et les ajoute au tableau HTML, nous pourrions lire les enregistrements sous forme de tableau lorsque nous naviguerons vers la page de lecture. La pagination est ajoutée afin que nous puissions naviguer entre les pages de la page lue (page 1, page 2, etc.).

Et maintenant, si nous naviguons vers <http://localhost/phpcrud/read.php>, nous verrons ce qui suit :

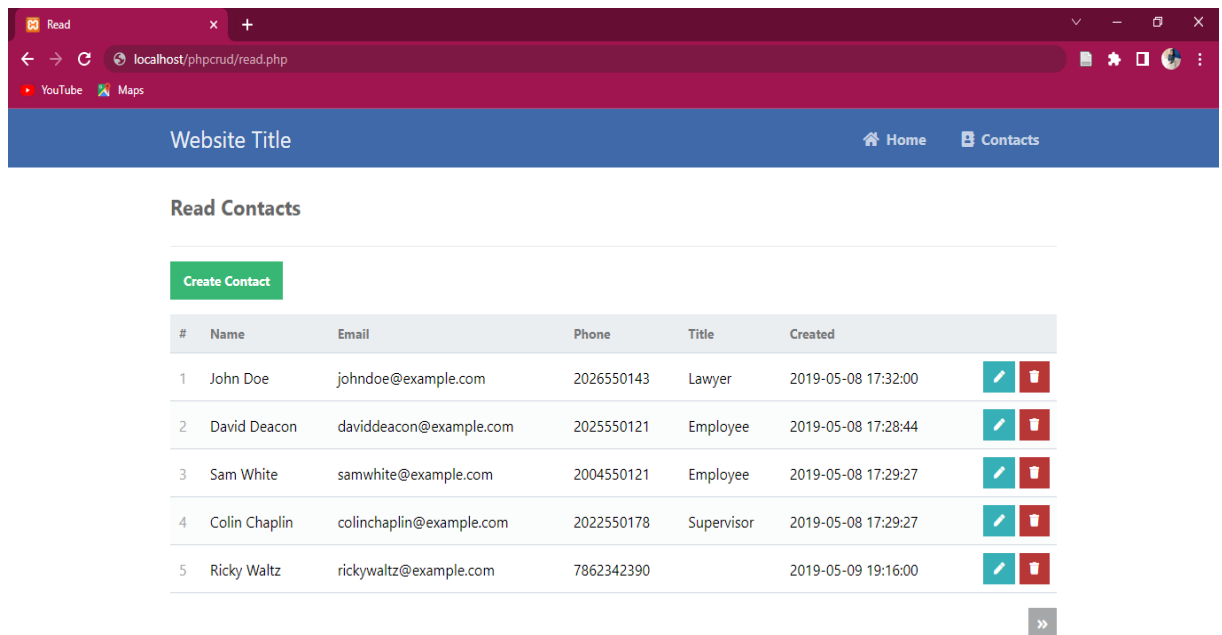


Figure 3:page read.php

4.4. Création de la page de création

La page de création sera utilisée pour créer de nouveaux enregistrements et les insérer dans notre table *Contacts*.

Modifiez le fichier *create.php* et ajoutez :

```
<?php
include 'functions.php';
$pdo = pdo_connect_mysql();
$msg = '';
// Vérifiez si les données POST ne sont pas vides
if (!empty($_POST)) {
    // Données de publication non vides insérer un nouvel enregistrement
    // Configurez les variables qui vont être insérées, nous devons vérifier
    // si les variables POST existent sinon nous pouvons les vider par défaut
    $id = isset($_POST['id']) && !empty($_POST['id']) && $_POST['id'] !=
'auto' ? $_POST['id'] : NULL;
    // Vérifiez si la variable POST "nom" existe, sinon la valeur par défaut
    // est vide, fondamentalement la même pour toutes les variables
    $name = isset($_POST['name']) ? $_POST['name'] : '';
    $email = isset($_POST['email']) ? $_POST['email'] : '';
```

```

    $phone = isset($_POST['phone']) ? $_POST['phone'] : '';
    $title = isset($_POST['title']) ? $_POST['title'] : '';
    $created = isset($_POST['created']) ? $_POST['created'] : date('Y-m-d
H:i:s');

    // Insérer un nouvel enregistrement dans la table des contacts
    $stmt = $pdo->prepare('INSERT INTO contacts VALUES (?, ?, ?, ?, ?, ?)');
    $stmt->execute([$id, $name, $email, $phone, $title, $created]);
    // Message de sortie
    $msg = 'Created Successfully!';
}
?>

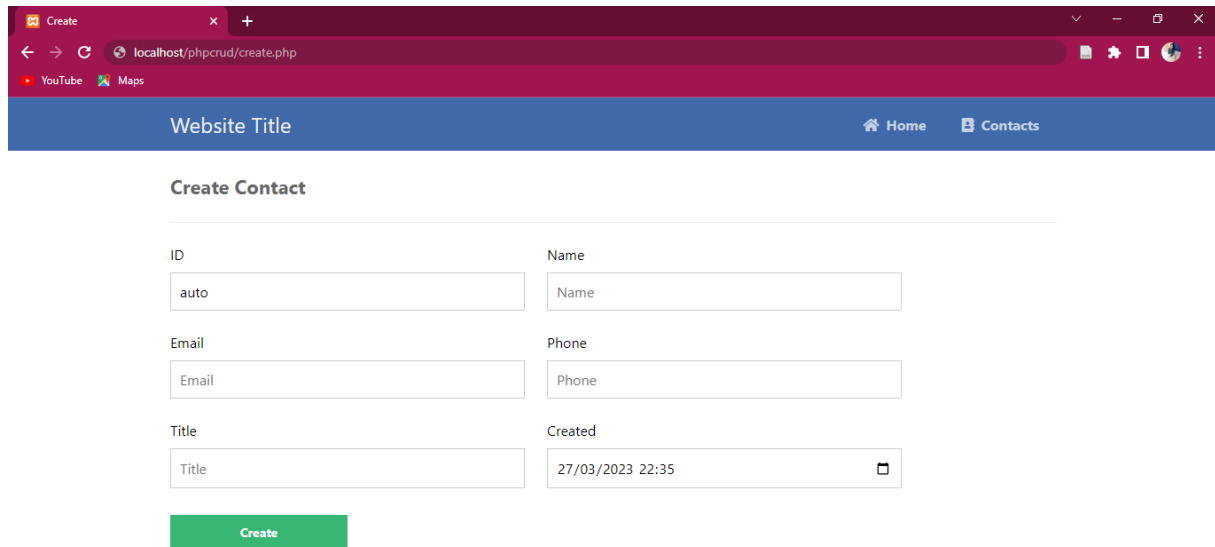
<?=template_header('Create')?>

<div class="content update">
    <h2>Create Contact</h2>
    <form action="create.php" method="post">
        <label for="id">ID</label>
        <label for="name">Name</label>
        <input type="text" name="id" placeholder="26" value="auto" id="id">
        <input type="text" name="name" placeholder="Name" id="name">
        <label for="email">Email</label>
        <label for="phone">Phone</label>
        <input type="text" name="email" placeholder="Email" id="email">
        <input type="text" name="phone" placeholder="Phone" id="phone">
        <label for="title">Title</label>
        <label for="created">Created</label>
        <input type="text" name="title" placeholder="Title" id="title">
        <input type="datetime-local" name="created" value="<?=date('Y-m-
d\TH:i')?>" id="created">
        <input type="submit" value="Create">
    </form>
    <?php if ($msg): ?>
    <p><?=$msg?></p>
    <?php endif; ?>
</div>

<?=template_footer()?>

```

Et maintenant, si nous naviguons vers <http://localhost/phpcrud/create.php> ou cliquons sur le bouton **Create Contact** sur la page de lecture, nous verrons ce qui suit :



The screenshot shows a web browser window with the address bar displaying 'localhost/phpcrud/create.php'. The page has a dark blue header with 'Website Title' on the left and 'Home' and 'Contacts' links on the right. Below the header, the main content area is titled 'Create Contact'. It contains a form with six input fields arranged in two columns. The left column has fields for 'ID' (containing 'auto'), 'Email' (containing 'Email'), and 'Title' (containing 'Title'). The right column has fields for 'Name' (containing 'Name'), 'Phone' (containing 'Phone'), and 'Created' (containing '27/03/2023 22:35' with a calendar icon). A green 'Create' button is positioned at the bottom left of the form.

Figure 4:page create.php

4.5. Création de la page de mise à jour

La page de mise à jour sera utilisée pour mettre à jour les enregistrements dans notre table **Contacts**, cette page est similaire à la page de création mais au lieu d'insérer un nouvel enregistrement, nous mettrons à jour les enregistrements existants. Nous pourrions obtenir l'ID d'enregistrement avec une requête GET.

Modifiez le fichier **update.php** et ajoutez :

```
<?php
include 'functions.php';
$pdo = pdo_connect_mysql();
$msg = '';
// Vérifiez si l'identifiant du contact existe, par exemple update.php?id=1
// obtiendra le contact avec l'identifiant 1
if (isset($_GET['id'])) {
    if (!empty($_POST)) {
        // Cette partie est similaire à create.php, mais à la place, nous
        // mettons à jour un enregistrement et n'insérons pas
        $id = isset($_POST['id']) ? $_POST['id'] : NULL;
        $name = isset($_POST['name']) ? $_POST['name'] : '';
```

```

        $email = isset($_POST['email']) ? $_POST['email'] : '';
        $phone = isset($_POST['phone']) ? $_POST['phone'] : '';
        $title = isset($_POST['title']) ? $_POST['title'] : '';
        $created = isset($_POST['created']) ? $_POST['created'] : date('Y-m-d
H:i:s');

        // Update the record
        $stmt = $pdo->prepare('UPDATE contacts SET id = ?, name = ?, email =
?, phone = ?, title = ?, created = ? WHERE id = ?');
        $stmt->execute([$id, $name, $email, $phone, $title, $created,
$_GET['id']]);
        $msg = 'Updated Successfully!';
    }
    // Obtenir le contact à partir du tableau des contacts
    $stmt = $pdo->prepare('SELECT * FROM contacts WHERE id = ?');
    $stmt->execute($_GET['id']);
    $contact = $stmt->fetch(PDO::FETCH_ASSOC);
    if (!$contact) {
        exit('Contact doesn\'t exist with that ID!');
    }
} else {
    exit('No ID specified!');
}
?>

<?=template_header('Read')?>

<div class="content update">
    <h2>Update Contact #<?=$contact['id']?></h2>
    <form action="update.php?id=<?=$contact['id']?>" method="post">
        <label for="id">ID</label>
        <label for="name">Name</label>
            <input type="text" name="id" placeholder="1"
value="<?=$contact['id']?>" id="id">
            <input type="text" name="name" placeholder="John Doe"
value="<?=$contact['name']?>" id="name">
        <label for="email">Email</label>
        <label for="phone">Phone</label>

```

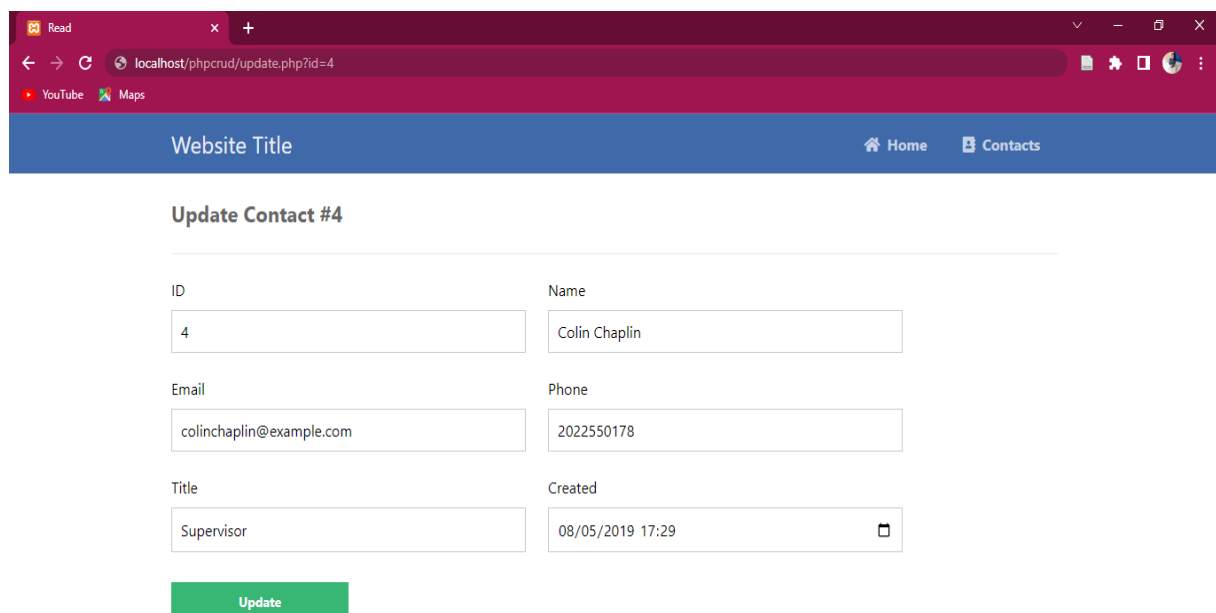
```

        <input type="text" name="email" placeholder="johndoe@example.com"
value="<?=$contact['email']?>" id="email">
        <input type="text" name="phone" placeholder="2025550143"
value="<?=$contact['phone']?>" id="phone">
        <label for="title">Title</label>
        <label for="created">Created</label>
        <input type="text" name="title" placeholder="Employee"
value="<?=$contact['title']?>" id="title">
        <input type="datetime-local" name="created" value="<?=date('Y-m-
d\TH:i', strtotime($contact['created']))?>" id="created">
        <input type="submit" value="Update">
    </form>
    <?php if ($msg): ?>
    <p><?=$msg?></p>
    <?php endif; ?>
</div>


<?=template_footer()?>

```

Sur la page de lecture (Contacts), nous devrions pouvoir cliquer sur l'icône de mise à jour à côté d'un enregistrement et le mettre à jour, nous devrions voir quelque chose comme ceci :



The screenshot shows a web browser window with the URL `localhost/phpcrud/update.php?id=4`. The page has a blue header with 'Website Title' and navigation links for 'Home' and 'Contacts'. The main content area is titled 'Update Contact #4' and contains a form with the following fields:

ID	Name
4	Colin Chaplin
Email	Phone
colinchaplin@example.com	2022550178
Title	Created
Supervisor	08/05/2019 17:29 

At the bottom of the form is a green 'Update' button.

Figure 5:page update.php

4.6. Création de la page de suppression

La page de suppression sera utilisée pour supprimer des enregistrements du tableau *Contacts*. Avant qu'un utilisateur puisse supprimer un enregistrement, il devra le confirmer, cela empêchera une suppression accidentelle.

Modifiez le fichier *delete.php* et ajoutez :

```
<?php
include 'functions.php';
$pdo = pdo_connect_mysql();
$msg = '';
// Vérifier que l'ID de contact existe
if (isset($_GET['id'])) {
    // Sélectionnez l'enregistrement qui va être supprimé
    $stmt = $pdo->prepare('SELECT * FROM contacts WHERE id = ?');
    $stmt->execute([$_GET['id']]);
    $contact = $stmt->fetch(PDO::FETCH_ASSOC);
    if (!$contact) {
        exit('Contact doesn\'t exist with that ID!');
    }
    // Assurez-vous que l'utilisateur confirme avant la suppression
    if (isset($_GET['confirm'])) {
        if ($_GET['confirm'] == 'yes') {
            // L'utilisateur a cliqué sur le bouton "Oui", supprimer
            l'enregistrement
            $stmt = $pdo->prepare('DELETE FROM contacts WHERE id = ?');
            $stmt->execute([$_GET['id']]);
            $msg = 'You have deleted the contact!';
        } else {
            // L'utilisateur a cliqué sur le bouton "Non", le redirige vers la
            page de lecture
            header('Location: read.php');
            exit;
        }
    }
} else {
    exit('No ID specified!');
```



```
}  
?>
```

Pour supprimer un enregistrement, le code vérifiera si la variable de requête GET "*id*" existe, si c'est le cas, vérifiez si l'enregistrement existe dans la table *Contacts* et confirmez à l'utilisateur s'il souhaite supprimer le contact ou non, une simple requête GET déterminera sur quel bouton l'utilisateur a cliqué (Oui ou Non).

Ajouter après :

```
<?=template_header('Delete')?>  
  
<div class="content delete">  
    <h2>Delete Contact #<?=$contact['id']?></h2>  
    <?php if ($msg): ?>  
    <p><?=$msg?></p>  
    <?php else: ?>  
    <p>Are you sure you want to delete contact #<?=$contact['id']?>?</p>  
    <div class="yesno">  
        <a href="delete.php?id=<?=$contact['id']?>&confirm=yes">Yes</a>  
        <a href="delete.php?id=<?=$contact['id']?>&confirm=no">No</a>  
    </div>  
    <?php endif; ?>  
</div>  
  
<?=template_footer()?>
```

Le code ci-dessus est le modèle de la page de suppression, cela inclut les boutons *Oui* et *Non* (confirmation de suppression) et le message de sortie. Les boutons *Oui* et *Non* créeront une nouvelle requête GET qui confirmera le choix de l'utilisateur.

Sur la page de lecture (Contacts), cliquez sur le bouton de suppression de l'un des enregistrements, vous devriez voir quelque chose comme ceci :

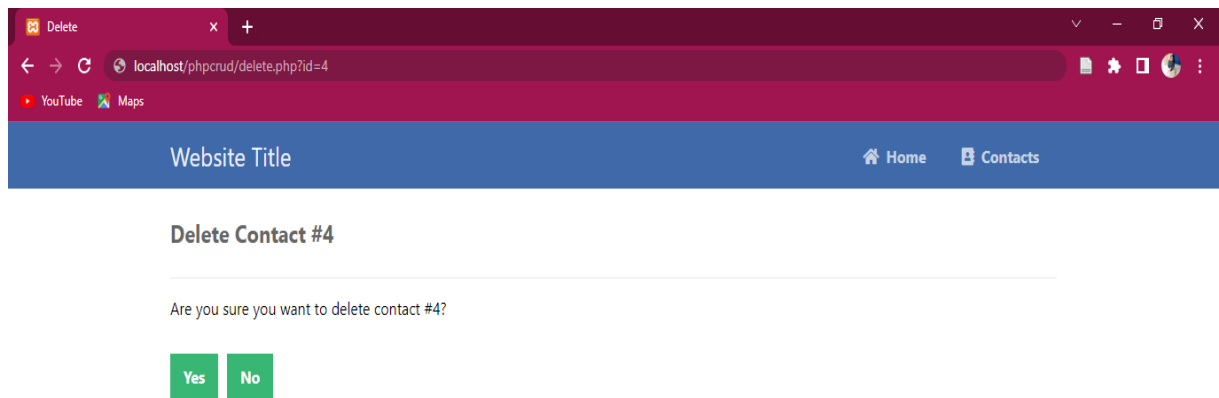


Figure 6:page delete.php