Conception de sites web dynamique

HTML – CSS – JAVASCRIPT – PHP
BASE DE DONNÉES

IF04U050

David Bouchet

david.bouchet.paris5@gmail.com

Conception de sites web dynamique

PHP
6e partie

PDO

PDO = *PHP Data Objects*

- Interface d’abstraction d’accès une base de données depuis PHP.
- Plusieurs bases de données sont prises en charge (*MySQL*, *PostgreSQL*, *Microsoft SQL Server*, *IBM DB2*, *Oracle*, etc.)
- Mêmes fonctions pour toutes les bases de données.
- Chaque base peut garder ses spécificités.
- **Disponible uniquement à partir de PHP 5.**

PDO – Connexion (1)

Vous devez vous munir des informations suivantes :

- Nom de l'hôte.
- Nom de la base de données.
- Nom d'utilisateur.
- Mot de passe.

Ces informations sont relatives à la base de données sur laquelle vous souhaitez vous connecter.

PDO – Connexion (2)

La connexion s'effectue par la création d'un objet PDO.

```
new PDO('<prefix>:host=<host>;dbname=<dbname>', '<login>', '<password>');
```

Le préfixe est un préfixe DSN (*Data Source Name*). Il est fonction du type de base de données.

Quelques exemples de préfixes DSN :

- Préfixe DSN pour PostgreSQL : pgsql
- Préfixe DSN pour MySQL : mysql
- Préfixe DSN pour Oracle : oci

Le constructeur renvoie un objet qui permettra d'accéder à la base de données à travers ses différentes méthodes et attributs.

La gestion des erreurs se fait à l'aide d'une exception.

PDO – Connexion (3)

Connexion à une base de données *PostgreSQL*

```
try
{
    $pgDB = new PDO('pgsql:host=opale…paris5.fr;dbname=BDgdupont', 'gdupont', 'uiot5434');
}

catch (Exception $e)
{
    exit($e->getMessage());
}
```

Connexion à une base de données *MySQL*

```
try
{
    $myDB = new PDO('mysql:host=opale…paris5.fr;dbname=gdupont', 'gdupont', 'uiot5434');
}

catch (Exception $e)
{
    exit($e->getMessage());
}
```

PDO – Déconnexion

La déconnexion à une base de données s'obtient en attribuant la valeur *NULL* à l'objet PDO de la connexion. Elle est automatique en fin de script.

Exemple :

```php
// Connexion.
try
{
    $pgDB = new PDO('pgsql:host=…dbnamame=…', 'gdupont', 'uiot5434');
}

catch (Exception $e)
{
    exit($e->getMessage());
}

// Traitement...

// Déconnexion (facultative en fin de script).
$pgDB = null;
```

PDO – Exécuter une requête

Il existe deux méthodes différentes pour exécuter une requête.

- La méthode ***exec()*** qui doit être utilisée lorsqu'il ne s'agit pas d'une requête d'extraction de données (ex. CREATE TABLE). Elle renvoie le nombre de lignes modifiées ou effacées par la requête ou *FALSE* si une erreur survient.

- La méthode ***query()*** qui doit être utilisée lorsqu'il s'agit d'une requête d'extraction de données (ex. SELECT). Elle renvoie un objet sur la table de résultat ou *FALSE* si une erreur survient.

La méthode *errorInfo()* renvoie un tableau qui contient différentes informations sur la dernière erreur survenue.

PDO – Créer une table

```
$query = "CREATE TABLE customer
          (id INT AUTO_INCREMENT PRIMARY KEY,
           name VARCHAR(50),
           age SMALLINT,
           email VARCHAR(64) UNIQUE)";

if ($pgDB->exec($query) === false)
{
        print_r($pgDB->errorInfo());
        exit;
}
```

PDO – Remplir une table

```php
$query[] = "INSERT INTO customer VALUES(DEFAULT, 'roger', 24, 'roger@gm.com')";
$query[] = "INSERT INTO customer VALUES(DEFAULT, 'david', 35, 'david@yh.com')";
$query[] = "INSERT INTO customer VALUES(DEFAULT, 'max', 24, 'max@gm.com')";
$query[] = "INSERT INTO customer VALUES(DEFAULT, 'roger', 15, 'roger.b@ht.fr')";
$query[] = "INSERT INTO customer VALUES(DEFAULT, 'jade', 32, 'jade15@ht.fr')";
$query[] = "INSERT INTO customer VALUES(DEFAULT, 'léa', 24, 'lea@gm.com')";
$query[] = "INSERT INTO customer VALUES(DEFAULT, 'jade', 27, 'jade.s@gm.com')";

foreach ($query as $q)
    $pgDB->exec($q);
```

PDO – Extraction de données (1)

Méthode	Description
fetch($fetch_style)	Récupère une ligne de résultat. La forme de renvoie dépend de la valeur entière *$fetch_style* : PDO::FETCH_NUM : Tableau indicé. PDO::FETCH_ASSOC: Tableau associatif. PDO::FETCH_BOTH : Tableau associatif et indicé (valeur par défaut). PDO::FETCH_OBJ : Objet.
closeCursor()	Ferme le curseur, permettant à la requête d'être de nouveau exécutée
rowCount()	Nombre de lignes du résultat
columnCount()	Nombre de colonnes du résultat

PDO – Extraction de données (2)

```php
$query = "SELECT * FROM customer";
$result = $pgDB->query($query);

while ($row = $result->fetch())
{
    echo "id = ".$row['id']."; ";
    echo "name = ".$row['name']."; ";
    echo "age = ".$row['age']."; ";
    echo "email = ".$row['email'];
    echo "<br />";
}

$result->closeCursor();
```

PHP

```
id = 1; name = roger; age = 24; email = roger@gm.com
id = 2; name = david; age = 35; email = david@yh.com
id = 3; name = max; age = 24; email = max@gm.com
id = 4; name = roger; age = 15; email = roger.b@ht.fr
id = 5; name = jade; age = 32; email = jade15@ht.fr
id = 6; name = léa; age = 24; email = lea@gm.com
id = 7; name = jade; age = 27; email = jade.s@gm.com
```

Affichage

PDO – Requêtes préparées (1)

Méthode	Description
PDO::prepare()	Prépare une requête à l'exécution et retourne un objet *PDOStatement*.
PDOStatement::execute()	Exécute une requête préparée

Les paramètres de la requête sont passés soit par :

- **Des marqueurs interrogatifs (?).** Dans ce cas, les valeurs seront passées par un tableau indicé dans la méthode *execute()*. L'ordre des marqueurs devra correspondre avec l'ordre des éléments du tableau.

- **Des marqueurs nommés (précédés par deux points (:))**. Dans ce cas les valeurs seront passées par un tableau associatif dans la méthode *execute()*. L'ordre n'a plus d'importance.

PDO – Requêtes préparées (2)

Paramètres : marqueurs interrogatifs

```php
$query = "SELECT * FROM customer WHERE name = ? AND age < ?";
$result = $pgDB->prepare($query);

$result->execute(array("roger", 30)); DisplayResult($result);
$result->execute(array("jade", 30)); DisplayResult($result);

$result->closeCursor();

function DisplayResult($result)
{
    while ($row = $result->fetch())
    {
        echo "id = ".$row['id']."; ";
        echo "name = ".$row['name']."; ";
        echo "age = ".$row['age']."; ";
        echo "email = ".$row['email'];
        echo "<br />";
    }
    echo "<br />";
}
```

PDO – Requêtes préparées (3)

Paramètres : marqueurs nommés

```php
$query = "SELECT * FROM customer WHERE name = :nom AND age < :age";
$result = $pgDB->prepare($query);

$param = array(
         ':nom' => 'roger',
         ':age' => 30);

$result->execute($param);
DisplayResult($result);

$param = array(
         ':nom' => 'jade',
         ':age' => 30);

$result->execute($param);
DisplayResult($result);

$result->closeCursor();
```

PDO – Requêtes préparées (4)

Il est également possible de lier des variables PHP aux différents paramètres de la requête préparée.

```
bool bindParam ($parameter , &$variable [, int $data_type = PDO::PARAM_STR])
```

$parameter : Identifiant. Pour une requête préparée utilisant des marqueurs nommés, ce sera le nom du paramètre sous la forme *:name*. Pour une requête préparée utilisant les marqueurs interrogatifs, ce sera la position indexé du paramètre (indice de départ = 1).

$variable : Nom de la variable PHP à lier au paramètre de la requête SQL.

$data_type : Type explicite de données pour le paramètre (PDO::PARAM_<type>).
PDO::PARAM_INT → Représente le type de données INTEGER SQL.
PDO::PARAM_STR → Représente les types de données CHAR, VARCHAR etc.
Les autres types sont disponibles sur la page suivante :
http://php.net/manual/fr/pdo.constants.php

Valeur de retour : *TRUE* si aucune erreur, *FALSE* dans le cas contraire.

PDO – Requêtes préparées (5)

Paramètres : marqueurs interrogatifs

```php
$query = "SELECT * FROM customer WHERE name = ? AND age < ?";
$result = $pgDB->prepare($query);

$result->bindParam(1, $nom, PDO::PARAM_STR);
$result->bindParam(2, $age, PDO::PARAM_INT);

$nom = "roger";
$age = 30;
$result->execute();
DisplayResult($result);

$nom = "jade";
$result->execute();
DisplayResult($result);

$result->closeCursor();
```

PDO – Requêtes préparées (6)

Paramètres : marqueurs nommés

```php
$query = "SELECT * FROM customer WHERE name = :nom AND age < :age";
$result = $pgDB->prepare($query);

$result->bindParam(":nom", $nom, PDO::PARAM_STR);
$result->bindParam(":age", $age, PDO::PARAM_INT);

$nom = "roger";
$age = 30;
$result->execute();
DisplayResult($result);

$nom = "jade";
$result->execute();
DisplayResult($result);

$result->closeCursor();
```

Frameworks PHP

Avec un peu d'expérience, on se rend compte que les développements PHP se ressemblent beaucoup.

On retrouve régulièrement les mêmes problématiques :

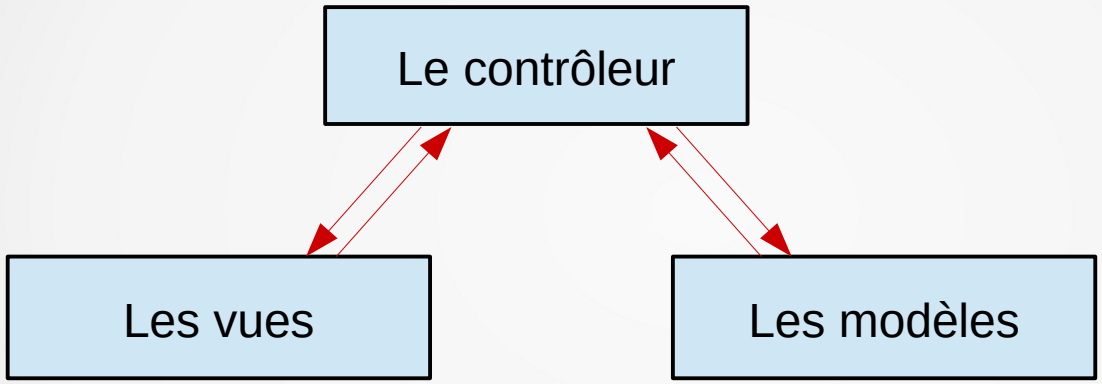
- Gestion de formulaires ;
- Gestion de bases de données ;
- Authentification avec identifiant et mot de passe ;
- Internationalisation ;
- Etc.

L'objectif des *frameworks* PHP est de rendre plus rapide et moins répétitif le développement de scripts en proposant une gestion des problématiques récurrentes.

Il existe de nombreux *frameworks* PHP : *Zend Framework, Symfony,* ***CodeIgniter****, Laravel,* etc.

Architecture MVC

La plupart des *frameworks* proposent une architecture MVC (Modèle Vue Contrôleur)



- **Les vues** représentent les pages ou les fragments de pages présentés à l'utilisateur.
- **Les modèles** représentent les structures de données et permettent d'interagir directement avec une base de données.
- **Le contrôleur** est un intermédiaire entre les vues et les modèles. C'est lui qui génère la page HTML à afficher à partir des vues et des modèles.

CodeIgniter - Présentation

Principales caractéristiques de *CodeIgniter* :

- Excellentes performances ;
- Très peu de configurations ;
- Pas de règles contraignantes ;
- Très bien documenté ;
- Prise en main rapide.

https://www.codeigniter.com/

CodeIgniter - Installation

Télécharger la dernière version sur le site officiel :
Par exemple : CodeIgniter-3.0.6.zip

Décompresser le fichier sur votre serveur web :
Par exemple : ~/public_html/

Après décompression, un répertoire par défaut a été créé :
Par exemple : ~/public_html/CodeIgniter-3.0.6

Renommer le répertoire par défaut :
Par exemple : ~/public_html/myProject

CodeIgniter – Arborescence principale

Le répertoire *application* :
Il contiendra tous les fichiers de votre application. Il contient déjà un certain nombre de fichiers et de répertoires.

Le répertoire *system* :
C'est le cœur de *CodeIgniter*, ce répertoire ne devra jamais être modifié.

Le répertoire *user_guide* :
Il contient la documentation de *CodeIgniter*. Il peut être effacé sans conséquence si l'on n'en a pas besoin.

Le fichier *index.php* :
Fichier principal de *CodeIgniter* qui doit être appelé. C'est lui qui effectuera la redirection vers une page de votre site.

CodeIgniter – Le répertoire *application*

Pour l'instant nous nous intéresserons uniquement aux répertoires suivants :

Le répertoire *config* :
Il contient les fichiers de configuration.

Le répertoire *controllers* :
Il contient les contrôleurs de votre application.

Le répertoire *models* :
Il contient les modèles de votre application.

Le répertoire *views* :
Il contient les vues de votre application.

CodeIgniter – Configuration de base

Dans le fichier *application/config/config.php* :
Il faut initialiser l'adresse de base de l'URL du site.
Par exemple :

```
$config['base_url'] = 'http://host/~login/myProject/';
```

Dans le fichier *application/config/database.php* :
Il faut initialiser les paramètres d'accès à la base de données.
Par exemple :

```
'hostname' => 'opale.ens.math-info.univ-paris5.fr',
'username' => 'gdupont',
'password' => 'uiot5434',
'database' => 'BDgdupont',
'dbdriver' => 'mysqli',
```

CodeIgniter – Hello World (1)

Créer le fichier *application/views/HelloWorld.html* :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World</title>
        <meta charset="utf-8" />
    </head>
    <body>
        <p>Hello World</p>
    </body>
</html>
```

CodeIgniter – Hello World (2)

Créer le fichier *application/controllers/HelloWorld.php* :

```php
<?php

class HelloWorld extends CI_Controller
{
    public function index()
    {
        $this->load->view('HelloWorld.html');
    }
}
```

- Il faut créer une classe et la dériver de la classe *CI_Controller*.
- Il faut créer une méthode *index()* (elle sera appelée automatiquement par *CodeIgniter*).
- La classe *CI_Controller* permet de charger une vue du répertoire *views* à l'aide de la méthode : `$this->load->view('nom du fichier de la vue');`
- La page est alors accessible à l'URL suivante : http://.../myProject/index.php/HelloWorld

CodeIgniter – URL (1)

Lorsqu'une classe dérive de *CI_Controller*, il est possible d'accéder à l'une de ses méthodes à l'aide de l'URL suivante :

`http://hôte/myProject/index.php/classe/méthode`

Par exemple, soit le code ci-dessous :

```
class HelloWorld extends CI_Controller
{
    public function index()
    {
        $this->load->view('HelloWorld.html');
    }

    public function search()
    {
        echo "Nous sommes dans la fonction search().";
    }

    public function result()
    {
        echo "Nous sommes dans la fonction result().";
    }
}
```

CodeIgniter – URL (2)

Les deux URL ci-dessous appelle la méthode *index()* de la classe *HelloWorld* :

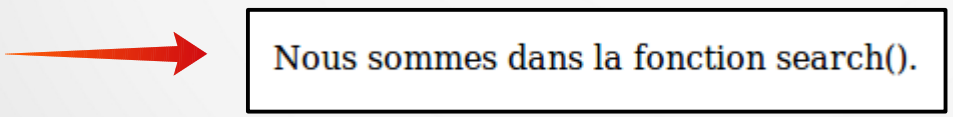
`http://hôte/myProject/index.php/HelloWorld`

`http://hôte/myProject/index.php/HelloWorld/index`

→ Si aucun nom de méthode n'est précisé, c'est la méthode *index()* qui est appelée par défaut.

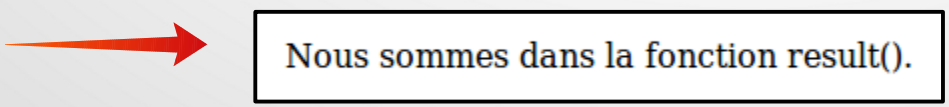
L' URL ci-dessous appelle la méthode *search()* de la classe *HelloWorld* :

`http://hôte/myProject/index.php/HelloWorld/search`



L' URL ci-dessous appelle la méthode *result()* de la classe *HelloWorld* :

`http://hôte/myProject/index.php/HelloWorld/result`



CodeIgniter – URL (3)

Il est également possible de passer des arguments aux méthodes d'une classe à l'aide de l'URL suivante :

`http://hôte/myProject/index.php/classe/méthode/arg_1/arg_2/…/arg_n`

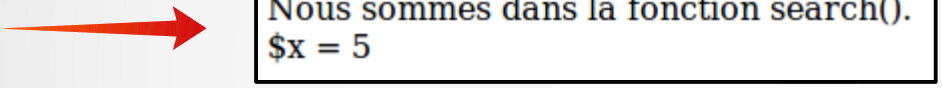
Par exemple, soit le code ci-dessous :

```php
class HelloWorld extends CI_Controller
{
    public function index()
    {
        $this->load->view('HelloWorld.html');
    }

    public function search($x)
    {
        echo "Nous sommes dans la fonction search().<br>";
        echo '$x = '.$x;
    }

    public function result($a, $b)
    {
        echo "Nous sommes dans la fonction result().<br>";
        echo '$a = '.$a."<br>";
        echo '$b = '.$b."<br>";
    }
}
```

CodeIgniter – URL (4)

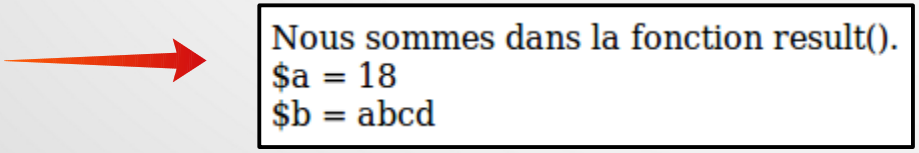
L' URL ci-dessous appelle la méthode *search()* de la classe *HelloWorld* avec l'argument « 5 » → `search(5)` :

`http://hôte/myProject/index.php/HelloWorld/search/5`



L' URL ci-dessous appelle la méthode *result()* de la classe *HelloWorld* avec les arguments « 18 » et « abcd » → `result(18, "abcd")` :

`http://hôte/myProject/index.php/HelloWorld/result/18/abcd`



CodeIgniter – URL (5)

Il est possible de configurer le serveur afin de supprimer l'occurrence « index.php ». Nous ne nous en servirons pas dans ce cours, mais cela donnerait pour les exemples précédents :

http://hôte/myProject/~~index.php~~/HelloWorld/search/5

http://hôte/myProject/HelloWorld/search/5

```
Nous sommes dans la fonction search().
$x = 5
```

http://hôte/myProject/~~index.php~~/HelloWorld/result/18/abcd

http://hôte/myProject/HelloWorld/result/18/abcd

```
Nous sommes dans la fonction result().
$a = 18
$b = abcd
```

CodeIgniter – Les vues (1)

CodeIgniter permet de modifier une vue de façon dynamique. La vue doit alors être un fichier PHP contenant du code HTML.

Considérons par exemple la vue ci-dessous :

~/public_html/myProject/application/views/DemoView.php

```
<!DOCTYPE html>
<html>
    <head>
        <title><?php echo $title?></title>
        <meta charset="utf-8" />
    </head>
    <body>
        <h1><?php echo $title?></h1>
        <p>Bonjour <?php echo $prenom?></p>
    </body>
</html>
```

CodeIgniter – Les vues (2)

Le contrôleur sera le suivant :

~/public_html/myProject/application/controllers/DemoView.php

```php
class DemoView extends CI_Controller
{
    public function index()
    {
        // Intialise dans un tableau
        // les valeurs à remplacer dans la vue.
        $data['title'] = "DemoView";
        $data['prenom'] = "David";

        // Charge la vue en remplaçant les valeurs.
        // (Si la vue est un fichier PHP, il est inutile
        // de préciser l'extension PHP.)
        $this->load->view('DemoView', $data);
    }
}
```

http://.../myProject/index.php/DemoView →

DemoView

Bonjour David

CodeIgniter – Les *helpers* (1)

Les *helpers* sont un regroupement de fonctions permettant de simplifier le développement de certaines tâches. Elles sont regroupées par catégories et doivent être chargées dans le constructeur.

CodeIgniter contient déjà un certain nombre de *helpers*, mais il est également possible d'ajouter ses propres *helpers* ou encore de modifier le comportement des *helpers* existants.

Exemples de *helpers* proposés par *CodeIgniter* :

- *Array Helper* → Manipulation de tableaux
- *Cookie Helper* → Manipulation de cookies
- *Date Helper* → Manipulation de dates
- *Directory Helper* → Manipulation de répertoires
- *File Helper* → Manipulation de fichiers
- *URL Helper* → Manipulation d'URL
- Etc.

CodeIgniter – Les *helpers* (2)

Exemple : Utilisation de la fonction *time_zone_menu()* du *Date Helper*.

Vue (…/views/DemoDate.php) :

```php
<!DOCTYPE html>
<html>
  <head>
    <title><?php echo $title?></title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1><?php echo $title?></h1>
    <p>Affichage d'un TimeZoneMenu :</p>
    <p><?php echo $menu?></p>
  </body>
</html>
```

Affichage :

Contrôleur (…/controllers/DemoDate.php) :

```php
class DemoDate extends CI_Controller
{
    // Constructeur.
    public function __construct()
    {
        // Appel du constructeur parent
        // (obligatoire).
        parent::__construct();

        // Chargement du "Date Helper".
        $this->load->helper("date");
    }

    public function index()
    {
        $data['title'] = "DemoDate";
        $data['menu'] = timezone_menu();
        $this->load->view('DemoDate', $data);
    }
}
```

CodeIgniter – Les bibliothèques (1)

Les bibliothèques sont un ensemble de classes qui permettent de simplifier le développement de certaines tâches. Tout comme les *helpers* elles doivent être chargées dans le constructeur.

CodeIgniter contient déjà un certain nombre de bibliothèques, mais il est également possible d'ajouter ses propres bibliothèques ou encore de modifier le comportement de celles existantes.

Exemples de bibliothèques proposées par *CodeIgniter* :

- *Session Library* → Manipulation de sessions
- *HTML Table Class* → Manipulation de tableaux HTML
- *Form Validation* → Aide à la validation de formulaires
- *Image Manipulation Class* → Manipulation d'images
- *Template Parser Class* → Gestion de modèles de vues
- Etc.

CodeIgniter – Les bibliothèques (2)

Exemple : Utilisation de la bibliothèque *Template Parser Class*.

→ *Possibilité d'utiliser des vues au format purement HTML.*

Vue (…/views/DemoTemplate.html) :

```
<!DOCTYPE html>
<html>
    <head>
        <title>{TITLE}</title>
        <meta charset="utf-8" />
    </head>
    <body>
        <h1>{TITLE}</h1>
        <p>Bonjour {PRENOM}</p>
    </body>
</html>
```

Affichage :

DemoTemplate

Bonjour David

Contrôleur (…/controllers/DemoTemplate.php) :

```
class DemoTemplate extends CI_Controller
{
  // Constructeur.
  public function __construct()
  {
    // Appel du constructeur parent
    // (obligatoire).
    parent::__construct();

    // Chargement de la bibliothèque.
    $this->load->library("parser");
  }

  public function index()
  {
    $data['TITLE'] = "DemoTemplate";
    $data['PRENOM'] = "David";
    $this->parser->parse(
    'DemoTemplate.html', $data);
  }
}
```

CodeIgniter – Les modèles (1)

Les modèles servent à se connecter à une base de données.

Pour les exemples qui suivent, nous utiliserons une base de données MySql qui possédera les caractéristiques suivantes :

- Hôte : localhost
- Nom de la base : demo_db
- Utilisateur : demo_user
- Mot de passe : demo_pw

Pour commencer, il faut initialiser les paramètres d'accès à la base de données MySql dans le fichier de configuration :

…/application/config/database.php

```
'hostname' => 'localhost',
'username' => 'demo_user',
'password' => 'demo_pw',
'database' => 'demo_db',
'dbdriver' => 'mysqli',
```

CodeIgniter – Les modèles (2)

La base de données *demo_db* contient la table *user*.
Cette dernière possède les enregistrements suivants :

id	login	password	first_name	last_name	mail
1	matt	matt	Matt	Salem	m.s@yh.com
2	goldorak	goldorak	Roger	Cadix	r.c@hm.com
3	hiro	hiro	David	Hiro	d.h@yh.com
4	coco	coco	Corine	Kara	c.k@hm.com
5	soso	soso	Sophie	Berthier	s.b@yh.com
6	chapi	chapi	Gérard	Gautier	g.g@gm.com
7	superman	superman	Charles	Kent	c.k@gm.com
8	kayak	kayak	Raphaël	Demont	r.d@yh.com
9	lolo	lolo	Laurence	Maxin	l.m@hm.com
10	teo	teo	Théodre	Pratt	t.p@gm.com

CodeIgniter – Les modèles (3)

Un modèle se place dans le répertoire …/application/models
Il doit dériver de la classe *CI_Model*
Le constructeur doit appeler le constructeur parent et doit charger le gestionnaire de la base de données (l'objet *db* est alors accessible).
Il est possible de lancer une requête à l'aide de la méthode `$this->db->query()`

…/myProject/models/Model_user.php

```
class Model_user extends CI_Model
{
    public function __construct()
    {
        parent::__construct();
        $this->load->database();
    }

    public function GetAll()
    {
        return $this->db->query("SELECT * FROM user");
    }

    public function Delete($id)
    {
        return $this->db->query("DELETE FROM user WHERE id='$id'");
    }
}
```

CodeIgniter – Les modèles (4)

Pour notre exemple, la vue se contentera d'afficher un tableau HTML qui contiendra tous enregistrements de la table *user*.
Le tableau sera généré dynamiquement par le contrôleur et remplacera l'occurrence `{TABLE}`.

…/myProject/views/DemoDB.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>DemoDB</title>
        <meta charset="utf-8" />
        <style>
            table, tr, th, td {border: solid 1px black;}
        </style>
    </head>
    <body>
        <h1>DemoDB</h1>
        <h2>Contenu de la table "user" :</h2>
        {TABLE}
    </body>
</html>
```

CodeIgniter – Les modèles (5)

Le contrôleur peut alors charger le modèle à l'aide de : `$this->load->model()`

- Le 1er argument est le nom du fichier sans l'extension PHP.
- Le 2nd argument est le nom de la variable qui permettra d'accéder au modèle.

```php
class DemoDB extends CI_Controller
{
    public function __construct()
    {
        parent::__construct();
        $this->load->library("parser");                    // Template Parser Class
        $this->load->library("table");                     // HTML Table Class
        $this->load->model("Model_user", "user");          // Chargement du modèle
    }

    public function index()
    {
        $result = $this->user->GetAll();                   // Accès au modèle
        $data['TABLE'] = $this->table->generate($result); // Génère un tableau HTML
        $this->parser->parse("DemoDB.html", $data);        // et le place dans la vue
    }

    public function DeleteUser($id)
    {
        $result = $this->user->Delete($id);                // Accès au modèle
        $this->index();                                    // Affichage
    }
}
```

CodeIgniter – Les modèles (6)

Affichage : `http://…/myProject/index.php/DemoDB`

DemoDB

Contenu de la table "user" :

id	login	password	first_name	last_name	mail
1	matt	matt	Matt	Salem	m.s@yh.com
2	goldorak	goldorak	Roger	Cadix	r.c@hm.com
3	hiro	hiro	David	Hiro	d.h@yh.com
4	coco	coco	Corine	Kara	c.k@hm.com
5	soso	soso	Sophie	Berthier	s.b@yh.com
6	chapi	chapi	Gérard	Gautier	g.g@gm.com
7	superman	superman	Charles	Kent	c.k@gm.com
8	kayak	kayak	Raphaël	Demont	r.d@yh.com
9	lolo	lolo	Laurence	Maxin	l.m@hm.com
10	teo	teo	Théodre	Pratt	t.p@gm.com

Suppression d'un enregistrement (par exemple celui dont le champ *id* vaut 6) :

`http://…/myProject/index.php/DemoDB/DeleteUser/6`