TP PHP n°3 : *framework* et *templates*

Sommaire

Les objectifs de ce tp sont de mettre en œuvre un *framework* pour réaliser une application PHP basée sur des *templates*.

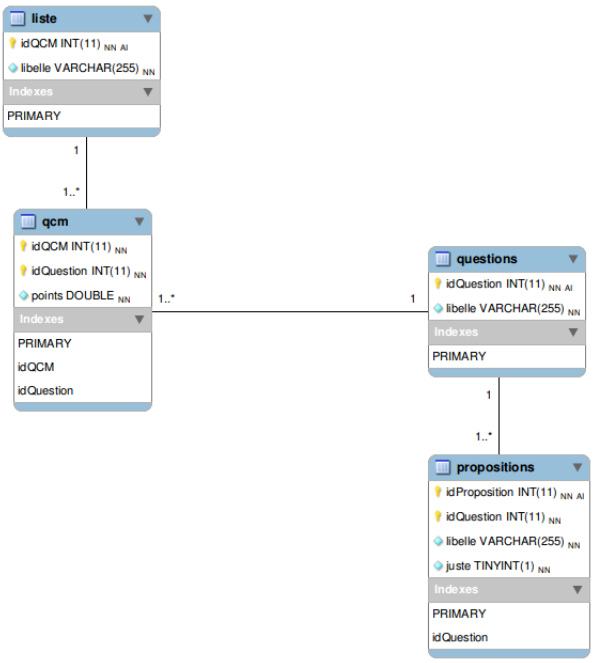
Il existe de très nombreux sites dédiés au PHP ! Il faut au moins l'accès au manuel (notamment pour les fonctions) : www.php.net/manual/fr/index.php

Un QCM en PHP

Objectifs

L'objectif est de réaliser une application PHP permettant de passer des QCM en utilisant le *framework* **TinyMVC** et le gestionnaire de *templates* **Smarty**.

Les QCM sont gérés à partir d'une base de données :



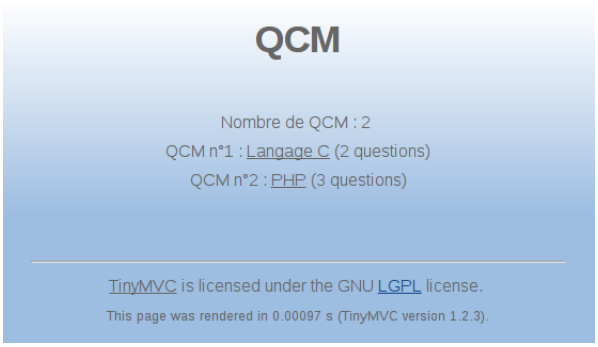
Comme on peut le voir, il n'est pas prévu pour l'instant de gérer des participants.

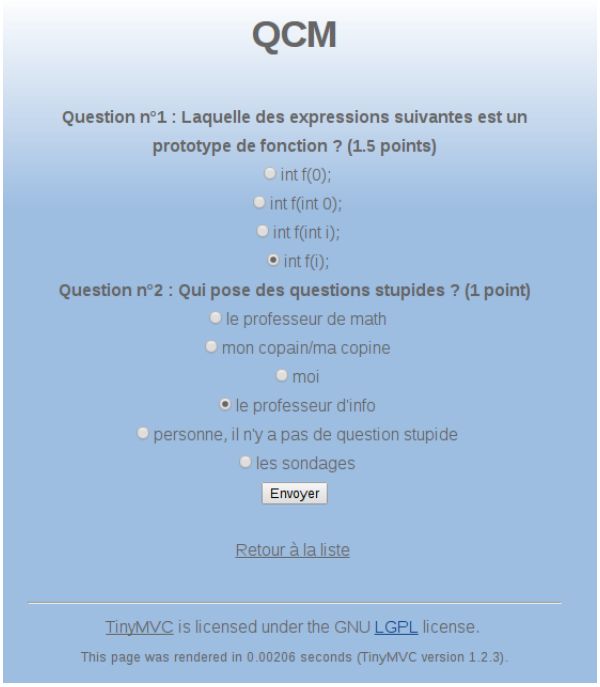
Explications :

- Un QCM comprend un identifiant `idQCM` et un `libelle`. Ils sont stockés dans la table `liste`.
- La table `qcm` fournit le contenu d'un QCM : il contient une ou plusieurs questions. On attribue un nombre de `points` à chaque question identifiée par `idQuestion`.
- Chaque question possède un `libelle` et un ensemble de propositions. Chaque proposition est définie par un `libelle` et il est indiqué si c'est une réponse `juste` ou non.

➠ Voir le détail de la base de données QCM en Annexe 1 page 15.

Exemple : http://localhost/tp-php/htdocs/index.php/qcm/





➠ Voir l'Annexe 2 page 18 pour les réécriture d'URL.

Framework et *Templates*

Le *framework* TinyMVC

TinyMVC est un *framework* MVC (*Model-View-Controller*) pour des applications PHP. Il fournit une séparation claire entre la base de données (*Model*), la présentation (*View*), et le lien entre les deux (*Controller*). TinyMVC n'est pas un *framework* complet. Il est une simple structure MVC avec une couche de support de base de données (PDO). Par contre, il propose une extension de ce cadre via des *plugins*.

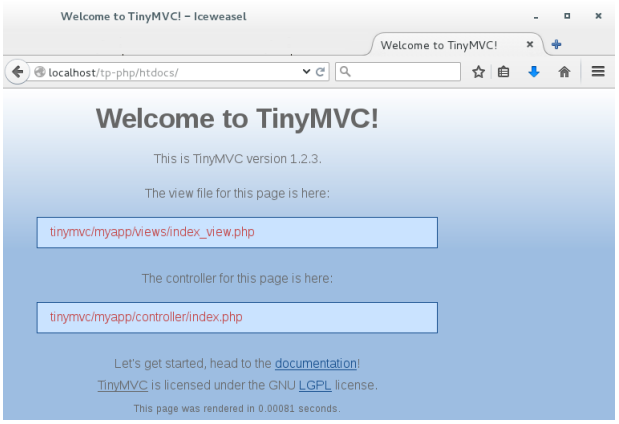
Liens :
– Téléchargement : http://www.tinymvc.com/download/
– Installation : http://www.tinymvc.com/documentation/index.php/Documentation:Installation
– Documentation : http://www.tinymvc.com/documentation/index.php/Documentation

Installation

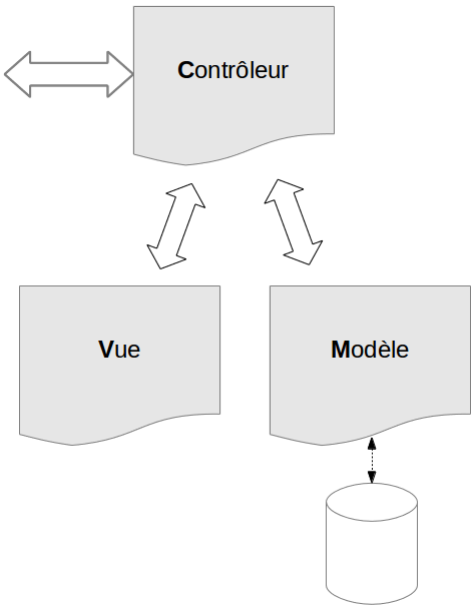
Question 1. Installer TinyMVC en suivant la procédure de la documentation et Tester.

```
$ wget -c http://www.tinymvc.com/downloads/TinyMVC-1.2.3.zip

$ unzip TinyMVC-1.2.3.zip

$ mkdir -p /var/www/tp-php

$ mv TinyMVC-1.2.3/tinymvc/ /var/www/tp-php/

$ mv TinyMVC-1.2.3/htdocs/ /var/www/tp-php/

$ ls -l /var/www/tp-php/
total 8
drwxr-xr-x 3 tvaira tvaira 4096 mai 3 2013 htdocs
drwxr-xr-x 5 tvaira tvaira 4096 mai 3 2013 tinymvc
```

Avec le navigateur, on va à l'adresse http://localhost/tp-php/htdocs/ :



Il reste maintenant à découvrir le cadre de développement du *framework* TinyMVC :



Mise en œuvre du contrôleur

Question 2. À partir des exemples de la documentation http://www.tinymvc.com/documentation/index.php/Documentation:Controllers, créer une classe `Hello_Controller` qui hérite de `TinyMVC_Controller` dans le fichier `hello.php` :

```
myapp/controllers/
    hello.php
```

```php
<?php
class Hello_Controller extends TinyMVC_Controller
{
  function index()
  {
    echo "Hello World!";
  }
}
?>
```

Question 3. Tester avec l'adresse http://localhost/tp-php/htdocs/index.php/hello.

Question 4. Ajouter une méthode à la classe et tester avec l'adresse http://localhost/tp-php/htdocs/index.php/hello/time.

```php
<?php
class Hello_Controller extends TinyMVC_Controller
{
  function index()
  {
    echo "Hello World.";
  }
  function time()
  {
    echo "The time is now.";
  }
}
?>
```

Évidemment, on va s'interdire de produire des "sorties" en faisant des "`echo`" à partir du contrôleur. Pour la partie présentation, on va utiliser une **vue**.

Mise en œuvre de la vue

Question 5. À partir des exemples de la documentation http://www.tinymvc.com/documentation/index.php/Documentation:Views, créer une vue dans le fichier `hello_view.php` :

```
myapp/views/
    hello_view.php
```



```
<html>
  <head><title>Hello</title></head>
  <body>
    Hello World.
  </body>
</html>
```

On modifie la classe `Hello_Controller` :

```
<?php
class Hello_Controller extends TinyMVC_Controller
{
  function index()
  {
    $this->view->display('hello_view');
  }
}
?>
```

Question 6. Tester avec l'adresse http://localhost/tp-php/htdocs/index.php/hello.

La vue assure une séparation mais il est possible d'assigner des variables via le contrôleur :

```
<html>
  <head><title><?=$title?></title></head>
  <body>
    <?=$body_text?>
  </body>
</html>
```

Et dans la classe `Hello_Controller` :

```
<?php
class Hello_Controller extends TinyMVC_Controller
{
  function index()
  {
    $this->view->assign('title','Hello');
    $this->view->assign('body_text','Hello World.');
    $this->view->display('hello_view');
  }
}
?>
```

Évidemment, on va s'interdire de "fabriquer des variables" à partir du contrôleur. Pour la partie données, on va utiliser un **modèle**.

Mise en œuvre du modèle

Question 7. À partir des exemples de la documentation http://www.tinymvc.com/documentation/index.php/Documentation:Models, créer une classe `Page_Model` qui hérite de `TinyMVC_Model` dans `page_model.php` :

```
myapp/models/
    page_model.php
```

```php
<?php
class Page_Model extends TinyMVC_Model
{
  function get_title()
  {
    return 'Hello';
  }
  function get_body_text()
  {
    return 'Hello World.';
  }
}
?>
```

On modifie la classe `Hello_Controller` :

```php
<?php
class Hello_Controller extends TinyMVC_Controller
{
  function index()
  {
    // load the model
    $this->load->model('Page_Model', 'page');

    // use the model to gather data
    $title = $this->page->get_title();
    $body_text = $this->page->get_body_text();

    $this->view->assign('title', $title);
    $this->view->assign('body_text', $body_text);
    $this->view->display('hello_view');
  }
}
?>
```

Question 8. Tester avec l'adresse http://localhost/tp-php/htdocs/index.php/hello.

Pour un accès vers une base de données, il faut préalablement configurer le fichier `myapp/configs/config_database.php` :

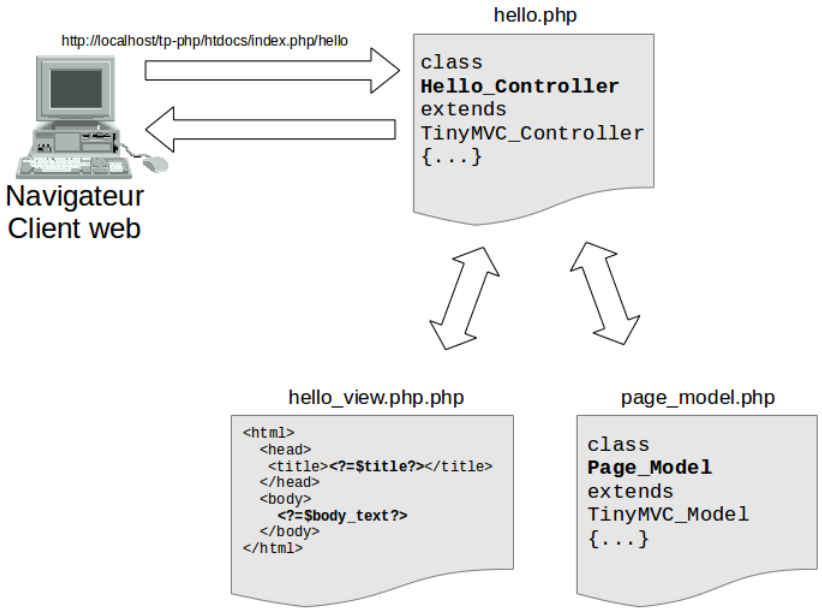
```php
<?php
$config['default']['plugin'] = 'TinyMVC_PDO'; // plugin for db access
$config['default']['type'] = 'mysql'; // connection type
```



```php
$config['default']['host'] = 'localhost'; // db hostname
$config['default']['name'] = 'QCM'; // db name
$config['default']['user'] = 'root'; // db username
$config['default']['pass'] = 'password'; // db password
$config['default']['persistent'] = false; // db connection persistence?
?>
```

Bilan

Ces premières manipulations sont toujours très importantes dans la découverte d'un *framework*. Celles-ci permettent d'assimiler le fonctionnement du cadre de développement :



Le gestionnaire de *templates* Smarty

Une architecture basée sur un système de *template* permet de séparer dans des fichiers distincts le code PHP de la mise en page HTML. L'affichage prévoit avec des zones prédéfinies où seront placées les données générées par le code PHP. L'avantage évident est de pouvoir travailler uniquement sur la mise en page, sans modifier quoi que ce soit dans le code PHP et inversement, ou de diviser efficacement le travail à faire, le **programmeur** s'occupant uniquement de la partie code et le **designeur** de la mise en page.

Il existe de très nombreux moteurs de template en PHP : **Smarty**, Twig, ...

Ici, on va intégrer le gestionnaire de *templates* Smarty au *framework* TinyMVC.

Liens :
- Téléchargement : http://www.smarty.net/download
- Installation : http://www.smarty.net/quick_install
- Tutoriel : http://www.smarty.net/crash_course
- Documentation : http://www.smarty.net/docs/en/

Installation

Question 9. Installer Smarty en suivant la procédure de la documentation et Tester.

```
$ wget -c https://github.com/smarty-php/smarty/archive/v3.1.29.zip

$ unzip smarty-3.1.29.zip

$ sudo cp -r smarty-3.1.29/libs /usr/share/php/smarty/

$ sudo mkdir -p /usr/share/php/smarty/templates/
$ sudo mkdir -p /usr/share/php/smarty/templates_c/
$ sudo mkdir -p /usr/share/php/smarty/configs/
$ sudo mkdir -p /usr/share/php/smarty/cache/

$ sudo chown nobody:www-data /usr/share/php/smarty/templates_c/
$ sudo chown nobody:www-data /usr/share/php/smarty/cache/
$ sudo chmod 775 /usr/share/php/smarty/cache/
$ sudo chmod 775 /usr/share/php/smarty/templates_c/
```

On crée un *template* dans `/usr/share/php/smarty/templates/index.tpl` :

```
<html>
  <head>
    <title>Smarty</title>
  </head>
  <body>
    Hello, {$name}!
  </body>
</html>
```

Et un programme de test :

```
<?php

// put full path to Smarty.class.php
require('/usr/share/php/Smarty/Smarty.class.php');
$smarty = new Smarty();

$smarty->setTemplateDir('/usr/share/php/smarty/templates');
$smarty->setCompileDir('/usr/share/php/smarty/templates_c');
$smarty->setCacheDir('/usr/share/php/smarty/cache');
$smarty->setConfigDir('/usr/share/php/smarty/configs');

$smarty->assign('name', 'Ned');
$smarty->display('index.tpl');

?>
```

Mise en œuvre de Smarty dans TinyMVC

Lire la documentation sur l'intégration de Smarty dans TinyMVC : http://www.tinymvc.com/documentation/index.php/Documentation:Templates

On va créer un *plugin* pour Smarty pour pouvoir l'utiliser de base dans TinyMVC :

```
myapp/plugins/
   tinymvc_library_smarty_wrapper.php
```



```php
<?php

define('SMARTY_SPL_AUTOLOAD', 1);

require('/usr/share/php/smarty/libs/Autoloader.php');
require('/usr/share/php/smarty/libs/Smarty.class.php');

class TinyMVC_Library_Smarty_Wrapper Extends Smarty
{
    function __construct()
    {
        parent::__construct();
        //$this->setTemplateDir('/usr/share/php/smarty/templates/');
        $this->setTemplateDir('/var/www/tp-php/tinymvc/myapp/views/');
        $this->setCompileDir('/usr/share/php/smarty/templates_c/');
        $this->setConfigDir('/usr/share/php/smarty/configs/');
        $this->setCacheDir('/usr/share/php/smarty/cache/');
    }
}

?>
```

Ensuite, on configure l'autochargement dans `myapp/configs/config_autoload.php` :

```php
$config['libraries'] = array(
  array('Smarty_Wrapper', 'smarty')
);

/* auto-loaded scripts */
$config['scripts'] = array();
```

On crée maintenant un template `index_view.html` dans `/var/www/tp-php/tinymvc/myapp/views/` :

```html
<html>
  <head>
    <title>{$title}</title>
  </head>
  <body>
    {$body_text}
  </body>
</html>
```

Puis, on teste avec le contrôleur :

```php
<?php

class Hello_Controller extends TinyMVC_Controller
{
  function test()
  {
      $this->load->library('Smarty_Wrapper', 'smarty');
      $this->load->model('Page_Model', 'page');

      $title = $this->page->get_title();
```



```
        $body_text = $this->page->get_body_text();

        $this->smarty->assign('title', $title);
        $this->smarty->assign('body_text', $body_text);

        $this->smarty->display('index_view.html');
    }
}

?>
```

Travail demandé

Question 10. En utilisant **TinyMVC** et **Smarty**, réaliser l'application **QCM** demandée.

On créera **trois méthodes** (`index()`, `start()` et `correction()`) dans la classe `QCM_Controller` :

```
<?php

/**
 * qcm.php
 *
 * QCM application controller
 *
 * @package TinyMVC
 * @author Thierry Vaira
 */

class QCM_Controller extends TinyMVC_Controller
{
    function index()
    {
        tmvc::timer('tmvc_app_start');
        $this->load->library('Smarty_Wrapper', 'smarty');
        $this->load->model('QCM_Model', 'page');
        $titre = $this->page->getPageTitre();
        $body_titre = $this->page->getTitre();
        $nbQCM = $this->page->getNbQCM();
        $QCM = $this->page->getQCM();
        for($i=0;$i<count($QCM);$i++)
        {
            $nbQuestions = $this->page->getNbQuestions($QCM[$i]['idQCM']);
            $QCM[$i]["nb"] = $nbQuestions['nb'];
        }
        $this->smarty->assign('titre', $titre);
        $this->smarty->assign('body_titre', $body_titre);
        $this->smarty->assign('nb_qcm', $nbQCM['nb']);
        $this->smarty->assign('liste_qcm', $QCM);
        $this->smarty->assign('TMVC_VERSION', TMVC_VERSION);
        tmvc::timer('tmvc_app_end');
        $TMVC_TIMER = sprintf('%0.5f', tmvc::timer('tmvc_app_start', 'tmvc_app_end'));
        $this->smarty->assign('TMVC_TIMER', $TMVC_TIMER);
```

```
        $this->smarty->display('index_view.html');
    }

    function start()
    {
        $this->load->library('Smarty_Wrapper', 'smarty');
        $this->load->model('QCM_Model', 'page');

        //echo "<pre>"; var_dump($_GET); echo "</pre>";

        // TODO

        $this->view->display('start_view.html');
    }

    function correction()
    {
        $this->load->library('Smarty_Wrapper', 'smarty');
        $this->load->model('QCM_Model', 'page');

        //echo "<pre>"; var_dump($_POST); echo "</pre>";

        // TODO

        $this->view->display('correction_view.html');
    }

}

?>
```

Vous pouvez modifier le fichier de configuration de l'application dans `myapp/configs/config_application.php` pour indiquer le nom par défaut du contrôleur :

```
<?php

// ...

/* name of default controller/method when none is given in the URL */
$config['default_controller'] = 'qcm';
$config['default_action'] = 'index';

// ...

?>
```

Voici un exemple de *template* basique `index_view.html` :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
    strict.dtd">
<html>
        <head>
                <title>{$titre}</title>
                <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

```
            <style type="text/css">
                    body {
                      background:#9dbde1 url(http://www.tinymvc.com/images/bg-gradient.gif
                          ) top repeat-x;
                            color: #666666;
                            font-family: arial, sans;
                            font-size: 100%;
                            line-height: 1.7em;
                            margin: 0 auto;
                            text-align: center;
                            width: 500px;
                    }
                    h1 {
                            font-size: 2.18em;
                            letter-spacing: -0.01em;
                    }
                    a:link {
                            color: #134c8c;
                    }
                    a:visited {
                            color: #666666;
                    }
                    #bottom {
                            border-top: 1px solid #134c8c;
                            margin-top: 1em;
                            padding-top: 1em;
                            font-size: 0.8em;
                    }
            </style>
        </head>
        <body>
        <h1>{$body_titre}</h1>
        <br />
        Nombre de QCM : {$nb_qcm}
        <br />
        {foreach $liste_qcm as $qcm}
            QCM n°{$qcm.idQCM} : <a href="start?idQCM={$qcm.idQCM}">{$qcm.libelle}</a> ({$qcm
                .nb} questions)<br />
        {/foreach}
        <br />
        <br />
        <hr />
        <div id="bottom ">
            <a href="http://www.tinymvc.com/">TinyMVC</a> is licensed under the GNU <a rel="
                license" href="http://www.gnu.org/licenses/lgpl.html">LGPL</a> license.
        <br />
        <span style="font-size: 0.8em">This page was rendered in {$TMVC_TIMER} s (TinyMVC
            version {$TMVC_VERSION}).</span>
        </div>
        </body>
</html>
```

Il faudra créer une classe `QCM_Model` qui manipulera la base de données `QCM` :

```php
<?php
/**
 * qcm-model.php
 *
 * @package TinyMVC
 * @author Thierry Vaira
 */

class QCM_Model extends TinyMVC_Model
{
    function getPageTitre()
    {
        return 'QCM';
    }

    function getTitre()
    {
        return 'QCM';
    }

    function getNbQCM()
    {
        return $this->db->query_one('select COUNT(*) as nb FROM liste');
    }

    function getQCM()
    {
        return $this->db->query_all('SELECT * FROM liste');
    }

    function getNbQuestions($idQCM)
    {
        return $this->db->query_one('select COUNT(*) as nb FROM qcm,questions where idQCM='.
            $idQCM.' and qcm.idQuestion=questions.idQuestion');
    }

    // TODO
}
?>
```

N'oubliez de configurer le fichier `myapp/configs/config_database.php` :

```php
<?php
$config['default']['plugin'] = 'TinyMVC_PDO'; // plugin for db access
$config['default']['type'] = 'mysql'; // connection type
$config['default']['host'] = 'localhost'; // db hostname
$config['default']['name'] = 'QCM'; // db name
$config['default']['user'] = 'root'; // db username
$config['default']['pass'] = 'password'; // db password
$config['default']['persistent'] = false; // db connection persistence?
?>
```

Annexe 1 : la base de données QCM

```sql
CREATE DATABASE IF NOT EXISTS QCM;

USE QCM;

-- --------------------------------------------------------

--
-- Structure de la table `liste`
--

CREATE TABLE IF NOT EXISTS `liste` (
  `idQCM` int(11) NOT NULL AUTO_INCREMENT,
  `libelle` varchar(255) NOT NULL,
  PRIMARY KEY (`idQCM`)
);

-- --------------------------------------------------------

INSERT INTO `liste` (`idQCM`, `libelle`) VALUES
("1", "Langage C"),
("2", "PHP");
-- --------------------------------------------------------

--
-- Structure de la table `questions`
--

CREATE TABLE IF NOT EXISTS `questions` (
  `idQuestion` int(11) NOT NULL AUTO_INCREMENT,
  `libelle` varchar(255) NOT NULL,
  PRIMARY KEY (`idQuestion`)
);

-- --------------------------------------------------------

INSERT INTO `questions` (`idQuestion`, `libelle`) VALUES
("", "Laquelle des expressions suivantes est un prototype de fonction ?"),
("", "Qui pose des questions stupides ?"),
("3", "À quel endroit s'exécute un script PHP ?"),
("4", "Peut-on exécuter un script PHP en ligne de commande ?"),
("5", "Peut-on faire de la programmation orientée objet en PHP ?");

-- --------------------------------------------------------

--
-- Structure de la table `qcm`
--

CREATE TABLE IF NOT EXISTS `qcm` (
  `idQCM` int(11) NOT NULL,
  `idQuestion` int(11) NOT NULL,
```



```
  `points` double NOT NULL,
  PRIMARY KEY (`idQCM`,`idQuestion`),
  KEY `idQCM` (`idQCM`),
  KEY `idQuestion` (`idQuestion`)
);

ALTER TABLE `qcm`
  ADD CONSTRAINT `fk_questions_id` FOREIGN KEY (`idQuestion`) REFERENCES `questions` (`
      idQuestion`) ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE `qcm`
  ADD CONSTRAINT `fk_liste_id` FOREIGN KEY (`idQCM`) REFERENCES `liste` (`idQCM`) ON DELETE
      NO ACTION ON UPDATE CASCADE;
-- --------------------------------------------------------

INSERT INTO `qcm` (`idQCM`, `idQuestion`, `points`) VALUES
("1", "1", 1.5),
("1", "2", 1),
("2", "3", 1),
("2", "4", 1),
("2", "5", 1);

-- --------------------------------------------------------

--
-- Structure de la table `propositions`
--

CREATE TABLE IF NOT EXISTS `propositions` (
  `idProposition` int(11) NOT NULL AUTO_INCREMENT,
  `idQuestion` int(11) NOT NULL,
  `libelle` varchar(255) NOT NULL,
  `juste` BOOLEAN NOT NULL,
  PRIMARY KEY (`idProposition`,`idQuestion`),
  KEY `idQuestion` (`idQuestion`)
);

ALTER TABLE `propositions`
  ADD CONSTRAINT `fk_questions_id1` FOREIGN KEY (`idQuestion`) REFERENCES `questions` (`
      idQuestion`) ON DELETE NO ACTION ON UPDATE CASCADE;

-- --------------------------------------------------------

INSERT INTO `propositions` (`idProposition`, `idQuestion`, `libelle`, `juste`) VALUES
("", "1", "int f(0);", false),
("", "1", "int f(int 0);", false),
("", "1", "int f(int i);", true),
("", "1", "int f(i);", false),
("", "2", "le professeur de math", false),
("", "2", "mon copain/ma copine", false),
("", "2", "moi", false),
("", "2", "le professeur d'info", false),
("", "2", "personne, il n'y a pas de question stupide", false),
("", "2", "les sondages", true),
```

```
("", "3", "Sur le serveur web", true),
("", "3", "Dans le navigateur du client", false),
("", "4", "Oui", true),
("", "4", "Non", false),
("", "5", "Oui", true),
("", "5", "Non", false);

-- --------------------------------------------------------

--
-- Quelques requêtes
--

-- Les questions du QCM n°1
SELECT * FROM qcm,questions WHERE idQCM=1 AND qcm.idQuestion=questions.idQuestion
-- idQCM idQuestion points idQuestion libelle
-- 1 1 1.5 1 Laquelle des expressions suivantes est un prototyp...
-- 1 2 1 2 Qui pose des questions stupides ?

-- Le nombre de questions du QCM n°1
SELECT COUNT(*) FROM qcm,questions WHERE idQCM=1 AND qcm.idQuestion=questions.idQuestion
-- 2

--
SELECT * FROM qcm,questions,propositions WHERE idQCM=1 AND qcm.idQuestion=questions.
    idQuestion and qcm.idQuestion=propositions.idQuestion

-- --------------------------------------------------------
```


Annexe 2 : Moteur de réécriture d'URL

Apache fournit un moteur de réécriture à base de règles permettant de réécrire les URLs des requêtes à la volée (http ://httpd.apache.org/docs/2.2/fr/mod/mod_rewrite.html). Il accepte un nombre illimité de règles, ainsi q'un nombre illimité de conditions attachées à chaque règle, fournissant ainsi un mécanisme de manipulation d'URL vraiment souple et puissant. Les manipulations d'URL peuvent dépendre de nombreux tests, des variables du serveur, des variables d'environnement, des en-têtes HTTP ou de l'horodatage.

Il faut commencer par activer le module `mod_rewrite` et redémarrer le serveur Apache :

```
$ sudo a2enmod rewrite

$ sudo service apache2 restart
```

Vous trouverez de nombreux exemples d'utilisation courante (et moins courante) dans la documentation spécifique à la réécriture.

Exemple : supposons qu'on a récemment renommé la page `default.html` en `index.html` et que l'on désire que les accès à l'ancienne URL restent compatibles. Cependant, on veut que les utilisateurs de l'ancienne URL ne puissent pas reconnaître que les pages ont été renommées. Pour cela, on utilise les directives :

- `RewriteEngine` qui active ou désactive l'exécution du moteur de réécriture.
- `RewriteRule` qui définit les règles pour le moteur de réécriture en utilisant des expressions rationnelles compatible perl.

```
<Directory "/srv/www/www.intra.net">
   ...
   RewriteEngine On
   RewriteRule ^default\.html$ index.html
</Directory>
```

Il est donc possible de réécrire l'url http://localhost/tp-php/htdocs/index.php/qcm/ en http://localhost/tp-php/htdocs/qcm/ avec les règles suivantes :

```
<Directory "/var/www/tp-php/htdocs">
 RewriteEngine On
 RewriteCond %{REQUEST_FILENAME} !-f
 RewriteCond %{REQUEST_FILENAME} !-d
 RewriteRule ^(.*)$ index.php/$1 [L]
</Directory>
```

