Auteur : Grégory JARRIGE

Bonnes pratiques PHP et DB2 for i

Macaron DB, une toolbox pour vos applications de gestion

Une toolbox pour vos applications de gestion Préambule

- J'ai présenté ce slide en juin 2012, dans le cadre d'un webinar organisé par Zend France. Le sujet présenté durant ce webinar venait compléter une présentation que j'avais faite, en avril 2012, lors d'un colloque organisé par IBM France.
- Durant le colloque IBM, j'avais présenté un sujet qui s'intitulait « DB2 et PHP, bonnes pratiques sous IBM i ». Mon objectif durant cette session était de présenter un certain nombre de techniques qu'il me semblait nécessaire de maîtriser pour utiliser le couple PHP - DB2 for i, de manière efficace.

Une toolbox pour vos applications de gestion
Préambule

Parmi les sujets que j'avais traités lors du colloque IBM, il y avait :

- Présentation des connecteurs PDO et ibm_db2
- Syntaxe système vs syntaxe SQL
- Requêtes paramétrées
- Paginations « full SQL » et « scroll cursor »
- Exécution de commandes système IBM i via DB2 et PHP
- Données avec dates d'effet
- Procédures stockées DB2 externes (pour l'appel de programmes RPG)
- Verrouillage optimiste
- **Créer son propre « wrapper » base de données**
- **Design pattern Active Record**
- **C.R.U.D. (Create – Retrieve – Update – Delete)**

Une toolbox pour vos applications de gestion Préambule

Ce que nous allons voir dans ce slide, c'est donc un complément du sujet présenté chez IBM.

Mon objectif avec le webinar organisé par Zend était de présenter un exemple d'utilisation de PHP dans le cadre de projets IBM, et en particulier de présenter le projet MacaronDB, une librairie PHP open source que j'avais développée pour faciliter le développement d'applications avec les bases DB2 (IBM i et Windows), et MySQL.

J'avais profité de cette session pour présenter aussi le projet HTML_Quickform2. Ce composant PHP open-source était une solution idéale pour générer des formulaires robustes et sécurisés. On pouvait combiner HTML_Quickform2 avec un plugin jQuery de validation de formulaire, pour obtenir des formulaires plus réactifs et améliorer le confort des utilisateurs. J'ai donc présenté cette solution, et j'avais conclu ce slide par un exemple de CRUD constitué à partir des composants pré-cités.

Avertissement

Les techniques de développement ont considérablement évolué depuis 2012, aussi les solutions évoquées dans ce slide ne sont plus forcément pertinentes en 2022.

Le projet MacaronDB peut néanmoins servir de source d'inspiration dans le cas de projets PHP pour lesquels on a besoin d'une couche d'accès base de données capable de fonctionner indifféremment sur plusieurs stacks PHP (sur et hors IBM i).

Librairie pour l'accès aux bases de données

Le projet MacaronDB

Le projet MacaronDB

Ce projet que j'ai développé est né d'un constat : la plupart des librairies pour l'accès aux bases de données sont décevantes. Parmi les griefs que je peux énumérer sur les nombreux projets que j'ai audités :

- Projets utilisant peu – voire pas du tout – la technique des requêtes SQL paramétrées, donc faible niveau de sécurité et de performance,
- Support inexistant - ou au mieux incomplet - pour les bases DB2 (DB2 for i est généralement le grand « oublié »)

Et MacaronDB est né.

Petit constat : développer sa propre couche d'accès base de données, ce n'est pas très compliqué, c'est même très intéressant, mais ça prend du temps (en développement, en test…), comme n'importe quel projet de développement.

Le projet MacaronDB

- MacaronDB est une librairie fournissant un jeu de classes destinées à faciliter le travail avec les bases de données DB2 for i, DB2 Express C, et MySQL.
- Cette librairie - dont j'ai assuré le développement – m'a rendu service sur plusieurs projets IBM i, et elle est maintenant suffisamment stable et mâture pour être mise à la disposition des développeurs, sous licence open source (new BSD).
- Vous pouvez télécharger le code source du projet et la documentation, directement sur mon blog :
  - https://github.com/gregja/macarondb/

Le projet MacaronDB - Exemple

- Exemple de liste simple avec pagination :

Code	Libellé
R10	Agence Lisieux
R30	Agence Paris
R32	Agence Marseille
R35	Agence Bordeaux
R37	Agence Pantin
R38	Agence Lyon
R40	Agence Perpignan
<< Préc. | 1-20 | 21 | Suiv. >>	

Le projet MacaronDB - Exemple

- Code PHP pour l'ouverture de la connexion à une base de données DB2 for i (à mettre dans un script que vous « appellerez » via la fonction require_once) :

```php
$usr = 'votre profil de connexion';
$pwd = 'votre mot de passe';
$options = array ();
$options['i5_naming'] = true ;
$options['i5_libl'] = array('libl1', 'libl2', 'libl3' ;
$options['DB2_ATTR_CASE'] = 'UPPER'  ;
/*
 * Ouverture d'une connexion BD sur un serveur IBM i, avec DB2 Connect
 */
require_once 'DB2/IBMi/DBWrapper.php';
require_once 'DB2/IBMi/DBConnex.php';
require_once 'DB2/IBMi/DBInstance.php';
$cnx_db04 = new DB2_IBMi_DBInstance('*LOCAL', $usr, $pwd, $options ) ;
```

Le projet MacaronDB - Exemple

```php
$sql = 'select * from matable where code = ? ' ;
$criteres = $_GET['code'] ; /* dans cet exemple simplifié,
    la donnée issue du formulaire n'est pas filtrée */
$nb_lignes_total = $cnx_db04->countNbRowsFromSQL ( $sql,
    $criteres );
if (is_null($nb_lignes_total) || $nb_lignes_total <= 0)
{
    echo 'pas de données trouvées';
} else {
    $max_lines_by_page = 7 ;
    $datas = $cnx_db04->getPagination ( $sql,
     $criteres, $offset, $max_lines_by_page,
     'BASE.code' );
    echo '<table border="1" width="100%"';
    echo 'cellspacing="0" cellpadding="5" >'.PHP_EOL;
    echo '<thead>'.PHP_EOL;
    echo '<tr class="header-row">'.PHP_EOL;
    $list_cols = array('Code', 'Libellé') ;
    echo '<th>'.implode('</th><th>',$list_cols) .
    '</th>'.PHP_EOL;
    echo '</tr>'.PHP_EOL;
    echo
```

```php
echo '<tbody>'.PHP_EOL;
foreach ( $datas as $data ) {
    echo '<tr>'.PHP_EOL;
    echo '<td>' . htmlentities(trim($data ['code'])) .
          '</td>'.PHP_EOL;
    echo '<td>' . htmlentities(trim($data ['libelle']))
    . '</td>'.PHP_EOL;
    echo '</tr>'.PHP_EOL;
}
echo '</tbody>'.PHP_EOL;
echo '<tfoot>'.PHP_EOL;
echo '<tr class="footer-row">' ;
echo '<td colspan="5">';
DBPagination::pcIndexedLinks ( $nb_lignes_total,
    $offset, MAX_LINES_BY_PAGE, $_SERVER ['PHP_SELF'],
    $params );
echo '</td>';
echo '</tr>';
echo '</tfoot>'.PHP_EOL;
echo '</table>'. PHP_EOL;
echo '<br/>'. PHP_EOL;
```

```php
} /* fin du bloc correspondant au « else »
```

Le design pattern Active Record

- Le design pattern Active Record est particulièrement bien adapté au développement de programmes de gestion, et notamment de modules de type C.R.U.D. (dont nous parlerons juste après).
- L'implémentation de ce design pattern au sein du projet MacaronDB est l'une des pièces maîtresses permettant de développer rapidement des écrans de gestion.
- Lors de la session d'avril 2012 chez IBM, j'avais abordé la théorie de ce design pattern, je ne la redétaillerai pas ici.
- La documentation actuelle de MacaronDB est très incomplète en ce qui concerne Active Record, ce n'est pas délibéré, c'est juste par manque de temps. Je vous prie de m'en excuser, et je vais m'efforcer de combler cette lacune rapidement.

La création de formulaire HTML

Le projet PEAR::HTML_Quickform2

Le projet PEAR::HTML_Quickform2

- Le projet HTML_Quickform est un composant destiné à faciliter la génération de formulaires, ainsi que leur validation.
- Le projet HTML_Quickform2 est une réécriture complète du composant, en PHP 5 (la version précédente était écrite en PHP 4). Les auteurs de HTML_Quickform2 se sont efforcés de maintenir une compatibilité ascendante avec la version précédente, mais quelques différences nécessitent d'effectuer de légères modifications (pour ceux qui utilisaient l'ancienne version du composant). La lecture de la procédure de migration – proposée dans la documentation officielle - est donc vivement recommandée.
- HTML_Quickform2 est officiellement en bêta, mais je considère qu'il est aujourd'hui suffisamment robuste pour des applications de production (pour tout ce qui concerne la génération du HTML et la validation côté serveur).
- Initialement disponible sur PEAR, on retrouve aujourd'hui plusieurs déclinaisons de ce projet sur packagist.org, donc celui-ci qui a été adapté pour PHP 7 :
  - mistralys/html_quickform2 - Packagist

Le projet PEAR::HTML_Quickform2

Les principaux « rules » du composant :

Nom de "rule"	Description du "rule"
nonempty	Contrôle qu'un champ n'est pas vide
empty	Contrôle qu'un champ est vide
required	Idem " nonempty " mais le champ est "marqué" comme obligatoire ("required") dans le formulaire généré
compare	Compare la valeur d'un champ avec celle d'un autre en utilisant l'opérateur de comparaison indiqué en paramètre
eq	Idem "compare" mais avec l'opérateur "===" codé en dur (Les valeurs sont comparées comme des chaînes)
neq	Rule inverse de "eq" (opérateur "!==")
lt	Idem "compare" mais avec l'opérateur "<" codé en dur (Les valeurs sont comparées comme des numériques)
lte	Idem "lt" avec opérateur "<="
gt	Idem "lt" avec opérateur ">"
gte	Idem "lt" avec opérateur ">="
regex	Contrôle que le champ "matche" avec l'expression régulière transmise en paramètre
email	Contrôle que la valeur du champ est une adresse email valide.
callback	Contrôle la valeur d'un champ via l'appel d'une fonction (ou méthode) externe (qui renverra TRUE si l'élément est valide).
length	Contrôle que la longueur de la valeur du champ est conforme aux limites fixées.
minlength	Contrôle que la longueur de la valeur du champ est au moins égale à la limite minimale fixée.
maxlength	Contrôle que la longueur de la valeur du champ est au maximum égale à la limite maximale fixée.
notcallback	Checks the value using a provided callback function (method). It is expected to return FALSE if the element is valid.
notregex	Rule inverse de "regex"

Source : http://pear.php.net/manual/en/package.html.html-quickform2.rules.list.php

Exemple de formulaire avec HTML_Quickform2

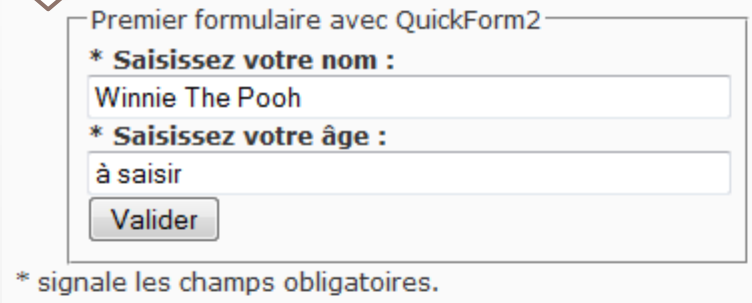
```php
// Chargement de la classe principale
require_once 'HTML/QuickForm2.php';
// Instanciation de l'objet HTML_QuickForm2
$form = new HTML_QuickForm2 ( 'tutorial' );
// Définition des valeurs par défaut pour les champs du formulaire
$form->addDataSource ( new HTML_QuickForm2_DataSource_Array ( array (
    'nom' => 'Winnie The Pooh', 'age' => 'à saisir'
) ) );
// Ajout de 2 champs au formulaire (nom et âge)
$fieldset = $form->addElement ( 'fieldset' ) ;
$fieldset->setLabel ( 'Premier formulaire avec QuickForm2' );
$nom = $fieldset->addElement ( 'text', 'nom', array (
    'size' => 50,  'maxlength' => 255
) )->setLabel ( 'Saisissez votre nom :' );
$age = $fieldset->addElement ( 'text', 'age', array (
    'size' => 50, 'maxlength' => 255
) )->setLabel ( 'Saisissez votre âge :' );
$fieldset->addElement ( 'submit', 'validform', array (
'value' => 'Valider'
) );
// La suite sur la diapo suivante …
```

Exemple de formulaire avec HTML_Quickform2

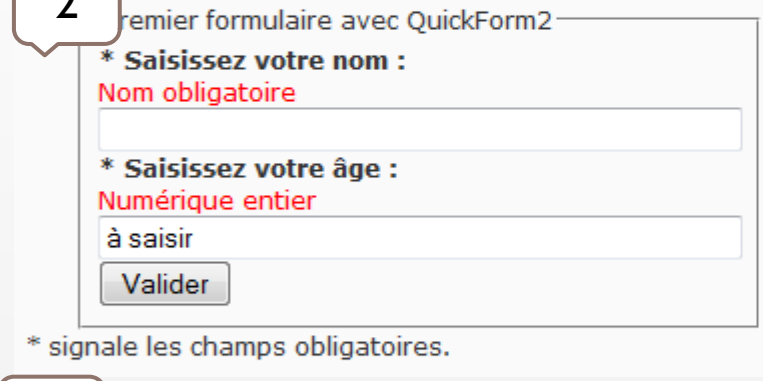
```
// Définition des règles de validation (les "rules" dans le jargon de HTMLQuickform)
$nom->addRule ( 'required', 'Nom obligatoire' );
$age->addRule ( 'required', 'Age obligatoire' );
$age->addRule ( 'callback', 'Numérique entier', array(
    'callback'  => 'filter_var',   'arguments' => array(FILTER_VALIDATE_INT)
));
$form->addRecursiveFilter('trim'); // Elimination des blancs sur tous les champs
// On tente de valider le formulaire
if ($form->validate ()) {
    echo '<h2>Bonjour, ' . htmlspecialchars ( $nom->getValue () ) . 'Vous avez '. $age->getValue
    () . ' ans </h2>';
    echo '<pre>' ; // Affichage d'un « dump » de l'instance formulaire
    $form_submit_values = $form->getValue() ;
    var_dump($form);
    echo '</pre>' ;
} else {
    // Affichage du formulaire
    echo '<div style="width:25em;">';
    echo $form;
    echo '</div>';
}
```

Exemple de formulaire avec HTML_Quickform2

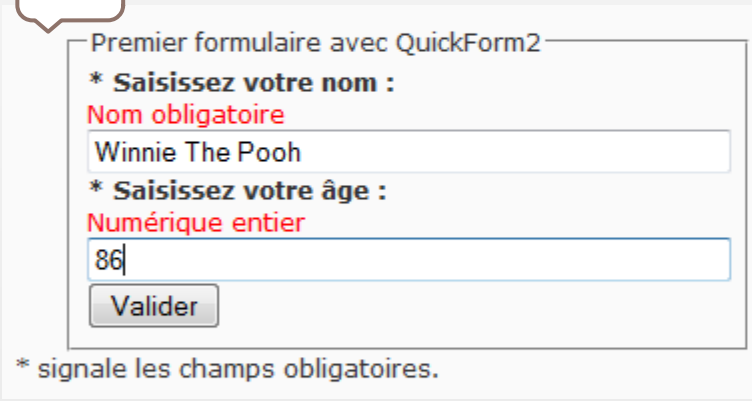
1



2



3



4

PEAR::HTML_Quickform2 – les « rules »

Par défaut les contrôles effectués par les « rules » s'exécutent côté serveur, comme dans l'exemple ci-dessous :

```
$nom->addRule ( 'required', 'Nom obligatoire');
```

Mais on peut souhaiter bénéficier d'une double validation, avec des contrôles effectués à la fois côté serveur (en PHP) et côté poste client (en JavaScript), comme dans l'exemple suivant :

```
$nom->addRule ( 'required', 'Nom obligatoire', null,
   HTML_QuickForm2_Rule::SERVER | HTML_QuickForm2_Rule::CLIENT );
```

La validation des données côté poste client n'est pas à mon avis le point fort de HTML_Quickform2. Pour la mise en œuvre de contrôles côté poste client, je préfère une autre approche, via l'utilisation de jQuery, dont nous reparlerons dans quelques instants.

PEAR::HTML_Quickform2 – les « rules »

On peut créer ses propres « rules », des plus simples aux plus complexes :

```
/**
 * Classe additionnelle pour traiter le "rule" de contrôle d'unicité de clé
 * avec l'aide de la classe ActiveRecord de MacaronDB
 */
class HTML_QuickForm2_Rule_UniqueKey extends HTML_QuickForm2_Rule {
  protected function validateOwner()  {
    $value  = $this->owner->getValue();
    $config = $this->getConfig();
    if (isset($config['model']) && $config['model'] instanceof DBActiveRecord) {
       if ($config['model']->isKeyUsed($value)) {
           return false;
       } else {
           return true;
       }
    } else {
       return false;
    }
  }
}
```

… dans votre code de génération d'un formulaire …

```
// déclaration d'un nouveau "rule" pour vérifier l'unicité d'une donnée dans
// le formulaire en cours de traitement
HTML_QuickForm2_Factory::registerRule('crud_key_used',
       'HTML_QuickForm2_Rule_UniqueKey');
```

… plus loin, après avoir instancié votre champ de saisie …

```
// affectation du nouveau "rule" à un champ du formulaire
$element_form->addRule('crud_key_used', 'Cette valeur est déjà utilisée',
       array('model'=> $this->model_item));
```

Le projet PEAR::HTML_Quickform2

Pourquoi j'aime bien HTML_Quickform2 :

- Projet écrit en PHP 5, code de bonne qualité, lisible et maintenable (si besoin)
- Support par email – en anglais – très réactif (j'ai testé pour vous),
- Système de filtres et de règles de validation (« rules ») simple et puissant
- Possibilité de créer facilement des « rules » spécifiques pour traiter des problématiques « métier »
- Double système de validation des formulaires possible (côté serveur en PHP et côté poste client en JavaScript), mais je recommande de n'utiliser que la validation côté serveur
- Possibilité d'effectuer des contrôles conditionnels (selon la valeur de certains champs du formulaire (cf. « chaining the rules » dans la documentation en ligne).
- Possibilité d'exporter le formulaire généré sous forme de tableau PHP, pour en personnaliser plus facilement l'affichage (ce n'était pas possible avec l'ancien HTLM_Quickform).

La validation de formulaire côté « poste client »

…du bon usage de jQuery

Validation de formulaire côté poste client

AVERTISSEMENT

Si vous décidez de mettre en œuvre des contrôles côté poste client (donc écrits en JavaScript), il est impératif que ces contrôles soient doublés par des contrôles équivalents côté serveur (en PHP).

Seuls les contrôles effectués côté serveur peuvent permettre de garantir la qualité et la sécurité des informations transmises par les formulaires. Un formulaire validé uniquement côté poste client est susceptible de mettre en péril la sécurité de votre système d'informations.

Le framework jQuery

- Le framework jQuery est un framework JavaScript puissant et malgré tout simple d'emploi, qui connaît un succès mérité, tant auprès des développeurs que des webdesigners.
- Couplé avec la librairie complémentaire jQueryUI, il permet de produire de nombreux effets graphiques, dont certains dédiés aux formulaires (champs de type « slider », calendriers, auto-complétion, onglets, etc…).
- Sites officiels :

http://jquery.com http://jqueryui.com

Validation de formulaire côté poste client

- Pourquoi utiliser jQuery :
  - Pour sa puissance et sa simplicité d'utilisation
  - Pour sa documentation officielle - et officieuse - riche et de qualité
  - Pour ses « thèmes » préconfigurés faciles à utiliser qui permettent de produire des effets « sexy » sans être un pro du webdesign.
  - Pour la facilité dans la mise en œuvre de contrôles de formulaire « non intrusifs ».
- Que signifie « non intrusif ? »
  - Un code JavaScript non intrusif est un code dont l'application peut se passer (en cas de problème) tout en continuant à fonctionner normalement :
    - Par exemple si le navigateur de l'internaute est configuré pour « bloquer » toute exécution du code JavaScript
    - Ou si le code JavaScript présente un dysfonctionnement – ou une incompatibilité avec un navigateur particulier – qui aurait pour effet de le rendre inopérant

Le plugin jQuery Validation

De nombreux plugins de qualité sont disponibles pour jQuery, dont le plugin « Validation », qui permet de mettre en œuvre des contrôles de formulaires sophistiqués. Il utilise pour ce faire un système de « rules » très proche de celui utilisé par HTML_Quickform2.

Site officiel du Plugin jQuery « Validation »

http://bassistance.de/jquery-plugins/jquery-plugin-validation/

Page de démonstration :

http://jquery.bassistance.de/validate/demo/

Le plugin jQuery Validation

Pourquoi j'aime bien jQuery et le plugin Validation :

- L'utilisation de jQuery en complément de HTML_Quickform2 est intéressante car jQuery permet de produire certains types de champs de saisie que HTML_Quickform2 ne sait pas produire (sliders et calendriers notamment).
- Avec jQuery, il est facile de modifier le comportement du formulaire en fonction de la saisie de l'utilisateur, sans avoir besoin de recharger l'intégralité de la page contenant le formulaire (réduction de la consommation de bande passante, formulaire plus réactif).
- Le système de « rules » du plugin Validation, très proche de celui de HTMLQuickform2, permet d'utiliser les 2 composants conjointement sans difficulté majeure (mais avec un peu de travail quand même).
- Le système de « rules » du plugin Validation permet de mettre en place des contrôles conditionnels (activation ou personnalisation d'un contrôle sur un champ selon la valeur d'un autre champ), dont certains sont difficiles à réaliser sans l'aide du JavaScript.

Le plugin jQuery Validation

La validation de formulaire côté poste-client n'est pas indispensable, mais elle permet de réduire les « échanges » entre postes clients (navigateurs) et serveur, les formulaires n'étant transmis au serveur qu'une fois les champs validés. Les saisies sont ainsi plus rapides et cela améliore sensiblement le confort de saisie pour l'utilisateur de votre application.

Je crois néanmoins utile de souligner que je préfère cantonner le JavaScript aux contrôles les plus simples, c'est-à-dire les « zones obligatoires », ainsi que les validations de types (numérique, email…) et de longueur. Du côté serveur, ces mêmes contrôles doivent être répétés (j'insiste !!!), et complétés par les contrôles plus complexes liés à vos règles métier, ce qui se fait aisément en utilisant le mécanisme des « rules » personnalisées de HTML_Quickform2.

Mise en pratique avec le développement d'un « C.R.U.D. »

N.B. : Les diapos qui suivent ont été écrites après le webinar, en utilisant les éléments produits durant la démonstration.

Mise en pratique – la table DB2 Contrat_TB

Voici le code source SQL de la table DB2 utilisée durant la démonstration (table créée à la volée via le logiciel « System i Navigator » d'IBM) :

```
CREATE TABLE GJABASE.CONTRAT_TB (
ID INTEGER
GENERATED ALWAYS AS IDENTITY
(START WITH 1, INCREMENT BY 1),
REF_CNT    CHAR(15) NOT NULL ,
LIB_CNT    CHAR(30) NOT NULL ,
DATE_CNT   DATE NOT NULL,
DATE_ANI   DATE NOT NULL,
NOM_RESP   CHAR(30) NOT NULL default '',
TYPE_CNT   CHAR(4)  NOT NULL ,
INFO_CNT   CHAR(80) NOT NULL default '',
ACC_CNT    CHAR(6)  NOT NULL default '',
MONT_CNT   DEC (11, 2) NOT NULL default 0,
VALID_CNT  CHAR(1) NOT NULL default '',
KBQ_CNT    CHAR(6) NOT NULL default '',
CRE_DATE DATE NOT NULL DEFAULT CURRENT_DATE ,
CRE_TIME TIME NOT NULL DEFAULT CURRENT_TIME ,
CRE_USID CHAR(20) NOT NULL DEFAULT USER ,
UPD_DATE DATE NOT NULL DEFAULT CURRENT_DATE ,
UPD_TIME TIME NOT NULL DEFAULT CURRENT_TIME ,
UPD_USID CHAR(20) NOT NULL DEFAULT USER ,
STATUT CHAR (1 ) NOT NULL WITH DEFAULT ' ' ,
DEL_DATE DATE DEFAULT null,
DEL_TIME TIME DEFAULT null ,
DEL_USID CHAR(20) DEFAULT null ,
CONSTRAINT GJABASE.Q_CONTRAT PRIMARY KEY( ID )
)  ;
```

```
CREATE INDEX GJABASE.CONTRAT_TB01 ON GJABASE.CONTRAT_TB (REF_CNT, ID) ;
CREATE INDEX GJABASE.CONTRAT_TB02 ON GJABASE.CONTRAT_TB (LIB_CNT, ID) ;
LABEL ON TABLE GJABASE.CONTRAT_TB IS 'Table des contrats' ;
LABEL ON COLUMN GJABASE.CONTRAT_TB (
ID        IS 'Id contrat' ,
REF_CNT   IS 'Référence contrat' ,
LIB_CNT   IS 'Libellé contrat' ,
DATE_CNT  IS 'Date ouverture contrat' ,
DATE_ANI  IS 'Date anniversaire' ,
NOM_RESP  IS 'Nom du responsable' ,
TYPE_CNT  IS 'Type contrat' ,
INFO_CNT  IS 'Info contrat' ,
ACC_CNT   IS 'Code client associé' ,
MONT_CNT  IS 'Montant contrat' ,
VALID_CNT IS 'Statut contrat' ,
KBQ_CNT   IS 'Compte banque' ,
CRE_DATE  IS 'Date création       ' ,
CRE_TIME  IS 'Heure création      ' ,
CRE_USID  IS 'ID user création    ' ,
UPD_DATE  IS 'Date modif          ' ,
UPD_TIME  IS 'Heure modif         ' ,
UPD_USID  IS 'ID user modif       ' ,
    STATUT    IS 'Code statut     ' ,
DEL_DATE  IS 'Date suppression       ' ,
DEL_TIME  IS 'Heure suppression      ' ,
DEL_USID  IS 'ID user suppression    '
) ;
```

Mise en pratique – la classe ContratModel

On peut créer la classe ContratModel manuellement, mais on peut aussi la créer au moyen d'un script PHP qui extrait la structure de la table et génère automatiquement la classe correspondante (cf. extrait ci-dessous) :

```
require_once (dirname ( __FILE__ ) . '/../macaronDB/DBActiveRecord.php');

class ContratModel extends DBActiveRecord implements intDBActiveRecord {

public function __construct($base) {
$this->table_name = 'CONTRAT_TB'; // à modifier après génération
$this->schema_name = 'GJABASE'; // à modifier après génération

// liste des colonnes de la table
$this->fields_name = array ('id', 'ref_cnt', 'lib_cnt', 'date_cnt', 'date_ani', 'nom_resp', 'type_cnt',
'info_cnt', 'acc_cnt', 'mont_cnt', 'valid_cnt', 'kbq_cnt', 'cre_date', 'cre_time', 'cre_usid', 'upd_date',
'upd_time', 'upd_usid', 'statut', 'del_date', 'del_time', 'del_usid');

$this->key_name = 'id'; // nom de la colonne "id"
$this->key_value = null; // valeur par défaut de la colonne "id" (avant utilisation de la méthode load() )

// nom de la colonne "clé" d'un point de vue utilisateur (peut être la colonne "id" ou un "identifiant manuel")
$this->user_key = 'ref_cnt' ;   // à modifier après génération seulement si nécessaire

$this->incr_id_mode = '*auto' ;
```

N.B. : Ceci est un extrait de la classe générée automatiquement durant la démonstration. Le code complet n'est pas fourni ici faute de place, mais il peut être fourni par email sur simple demande, et sera incorporé à la prochaine version de la documentation, dont la livraison est prévue pour septembre 2012.

Mise en pratique – la classe ContratModel

- Pour la génération automatique de la classe ContratModel – tel que je l'ai fait en « live », j'ai utilisé une nouvelle classe du projet MacaronDB, que j'ai appelée DBForgeActiveRecord, et qui sera incorporée dans la prochaine version de MacaronDB (le temps de compléter la documentation du projet).
- Une fois la classe générée, on peut lui appliquer quelques modifications, comme par exemple :
  - Suppression des zones mouchards du tableau des éléments, si on ne souhaite pas les voir apparaître dans les formulaires de mise à jour
  - Personnalisation de certains attributs HTML, de certains filtres, et/ou de certains « rules »
  - Ajout d'effets jQuery tels que :
    - L'effet « datepicker » pour les champs de type « date »,
    - L'effet d'auto-complétion sur des champs liés à des listes trop importantes pour pouvoir être incorporées dans des champs de type « Select » (liste déroulante),
    - L'appel de fonctions de type « popup » pour la génération de fenêtre de recherche avec sélection (l'équivalent de la touche F4 des bons vieux écrans verts),
    - Etc.

Mise en pratique – le PHP

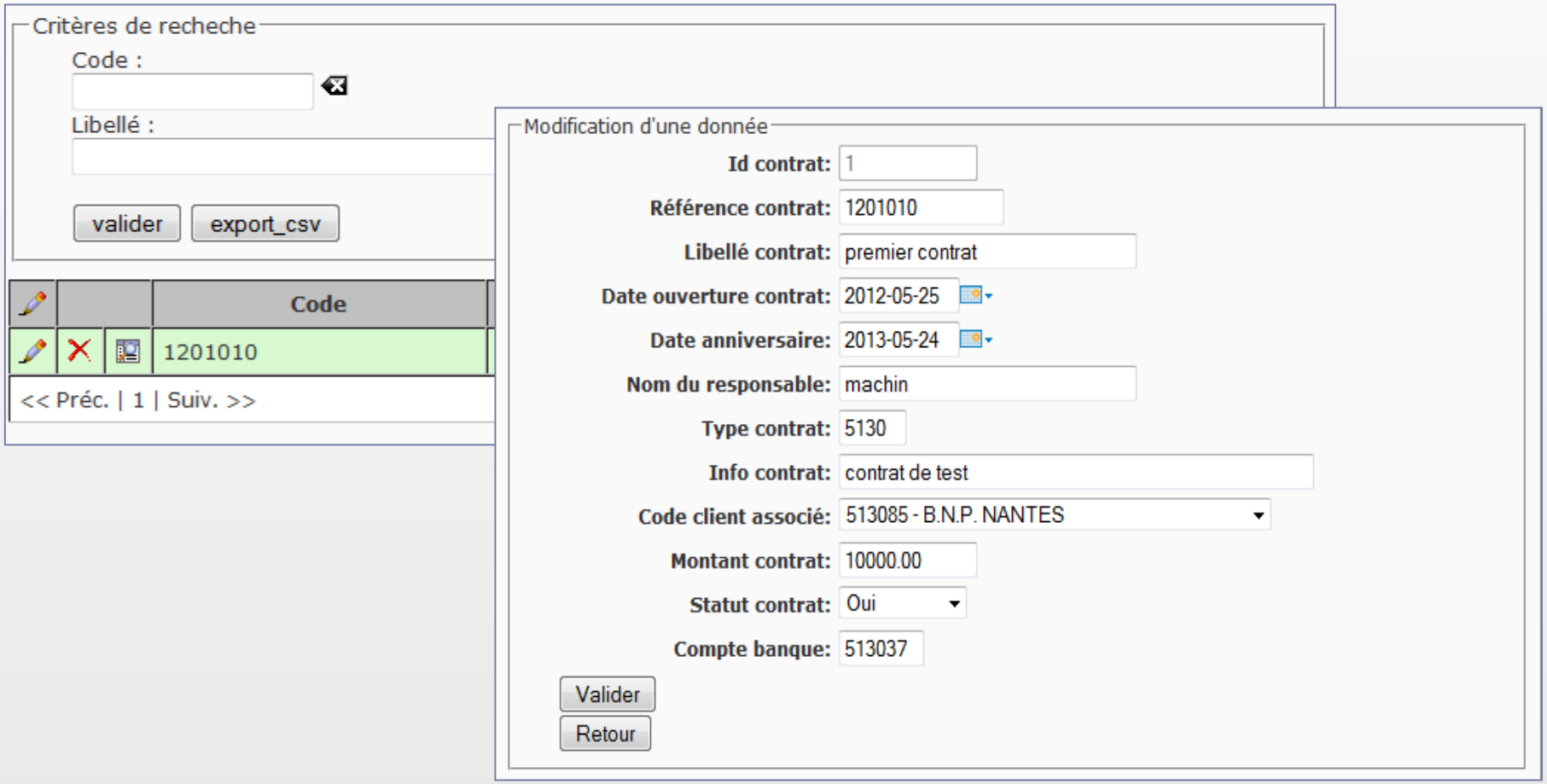
- Il reste enfin à écrire un script pour l'affichage de la liste des contrats (avec les options C.R.U.D), et un script pour l'affichage du formulaire dans les différents modes d'affichage (Affichage – Modification – Suppression). Ces scripts ne sont pas présentés ici – faute de place - mais ils seront incorporés à la prochaine version de la documentation.
- Pour faire le lien entre la classe ContratModel et le composant HTML_Quickform2, j'ai développé une classe que j'ai appelée CrudManager2 (*), qui est dédiée à cette tâche. Elle sera incorporée au projet MacaronDB dans la prochaine version prévue pour septembre 2012.
- Grâce à CrudManager2, l'affichage et la gestion du formulaire peut s'écrire en seulement 4 lignes :

```
$crud_instance = new CrudManager2($cnx_db, 'ContratModel', __FILE__) ;
$crud_instance->setTemplate('html_div') ;
$crud_instance->process() ;
$crud_instance->display() ;
```

(*) pourquoi CrudManager2 ? Parce que j'ai aussi une version dédiée à l'utilisation de HTML_Quickform, que j'ai appelée CrudManager, et que je n'envisage pas de diffuser, préférant travailler sur l'amélioration de CrudManager2.

Mise en pratique – le résultat

- Au final, on aboutit en quelques minutes au résultat suivant :



En conclusion

- Mon objectif avec cette présentation était de vous démontrer qu'il est possible de développer rapidement – en s'appuyant sur des composants open source – des applications d'entreprise robustes.
- Le module C.R.U.D. présenté ici peut être enrichi par l'ajout de fonctions de sécurité, permettant de définir des niveaux d'autorisations. Cet aspect n'a pas été abordé ici faute de temps, mais il ne présente pas de difficulté majeure à développer.
- La solution présentée ici n'est pas forcément adaptée à tous les types d'applications webs mais elle me semble particulièrement bien adaptée pour le développement (ou redéveloppement) de back-offices. La solution est facilement reproductible et industrialisable.
- La version actuelle de la classe ActiveRecord proposée dans MacaronDB est conçue pour travailler avec des tables DB2 ayant une seule colonne en identifiant. Mais elle peut être adaptée à des tables ayant des identifiants composés de plusieurs colonnes, moyennant quelques Modifications. Cela fait partie des évolutions que j'ai prévu d'implémenter assez rapidement (au plus tard durant le second semestre 2012).

Une toolbox pour vos applications de gestion

Merci de votre attention.

A bientôt.