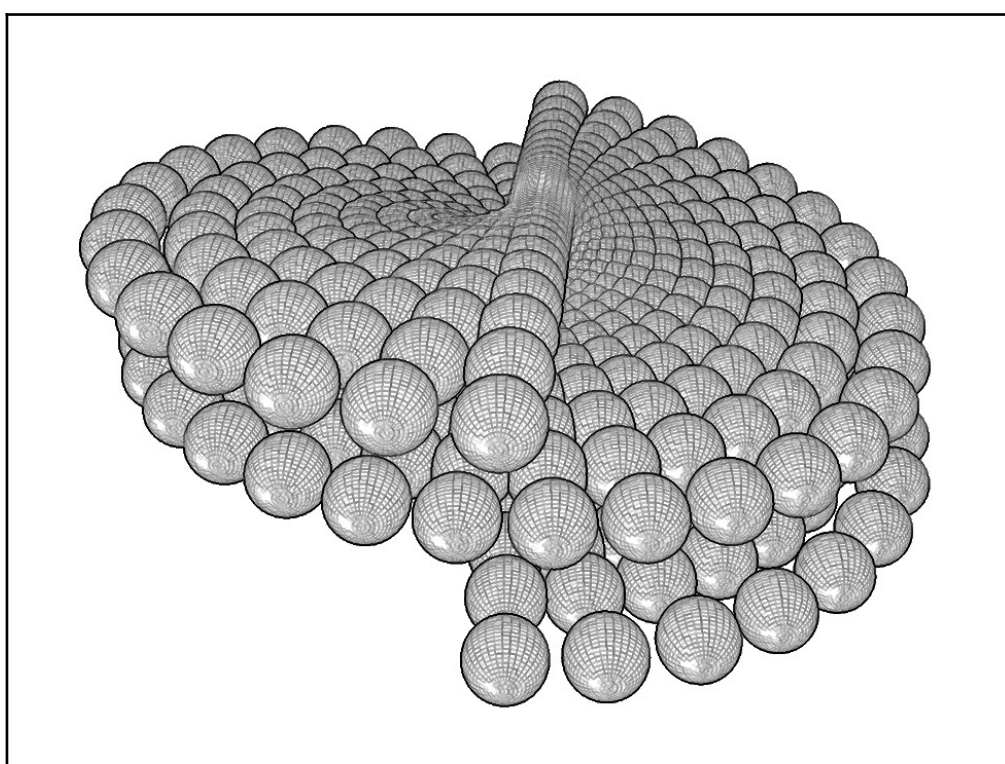


***Utilisation de PHP
en environnement IBM i
via un stack PHP sur PC***



Livre blanc sur l'utilisation de PHP en environnement IBM i

*Donne un poisson à un homme, il fait un repas.
Apprends-lui à pêcher, il mangera toute sa vie.
Confucius*

1	Introduction.....	4
2	Premiers contacts avec PHP.....	6
3	Préparation d'une table DB2 contenant la liste des pays.....	7
4	Premier script PHP affichant la liste des pays.....	10
5	Variante du 1 ^{er} script PHP.....	14
5.1	Variante avec critères de tri (ORDER BY).....	14
5.2	Variante avec critères de sélection (WHERE).....	15
6	Approfondissements autour de DB2.....	18
6.1	N'abusez pas du parseur SQL.....	18
6.2	Principe du SQL dynamique.....	18
6.3	Avantages des paramètres marqueurs.....	18
6.4	Outils d'analyse des performances SQL.....	19
7	Premier programme de reprise de données en PHP.....	21
8	Pagination, ou comment simuler un sous-fichier en PHP.....	25
9	Considérations sur la portabilité du code SQL.....	33
9.1	Sensibilité à la casse.....	33
9.2	Particularités syntaxiques de MySQL.....	34
9.3	Simuler la fonction LastInsertID() sous DB2.....	34
9.4	Gérer les attributs d'exécution de PDO.....	36
9.5	L'échappement des quotes.....	38
9.6	PDO et Object Relational Mapping (ORM).....	39
9.7	Pistes pour améliorer la gestion des erreurs.....	41
9.8	La gestion des dates et heures.....	47
10	La gestion des transactions sous PDO.....	52
11	Dialogue entre PHP et RPG via les procédures stockées DB2.....	53
12	Nouveautés de la V7.....	64
13	Portage de code PHP vers Zend Server for i.....	67
14	Règles de nommage de DB2 for i.....	68
15	Frameworks PHP.....	71
16	Liens et bouquins, pour aller plus loin.....	72
17	Conclusion.....	74
18	Annexe.....	75
18.1	Affichage de la liste des pays avec la clause DB2 OVER.....	75
18.2	Sécurisez votre environnement WAMP ou LAMP.....	78
18.3	Générer un script SQL à partir d'un fichier CSV.....	81
18.4	Configuration de PHP sur Laragon.....	83

Présentation de l'auteur :



Grégory Jarrige

Chef de Projet et Développeur Freelance

Je totalise plus de 30 années d'expérience dans le développement, la maintenance et la modernisation de logiciels de gestion, essentiellement sur plateforme IBM i. J'ai travaillé pour différents éditeurs de logiciels et différentes sociétés de services informatiques, avant de franchir le pas du freelancing en 2016, en créant ma propre société, Le Défrichoir.

Mes domaines de compétences : SQL DB2 (dont SQL-PSM), RPG III, RPG IV et RPG Free, PHP, Javascript et Node.js, MySQL et MariaDB, Oracle, etc.

J'ai rédigé de nombreux articles techniques – sur RPG et SQL DB2 - pour le site [Foothing.net](https://foothing.net) (anciennement XDocs400).

J'ai également publié de nombreux articles – sur SQL et Javascript - dans GNU/Linux Magazine : <https://connect.ed-diamond.com/auteur/jarrige-gregory>

Je travaille actuellement sur plusieurs sujets avec la modernisation d'applications IBM i, telle que :

- la mise en place d'environnements de type DevOps pour le pilotage de projets IBM i,
- la conversion de bases DB2 de type DDS vers SQL,
- la conversion de code RPG et Adelia vers RPG Free et SQL-PSM.

Passionné par la programmation en général, et les outils de développement open source en particulier, j'effectue une veille technologique permanente autour de différents sujets : HTML5 et Javascript, Graphisme 3D, Machine Learning. Blockchain, etc.

Co-fondateur et co-animateur, depuis 2016, du meetup CreativeCodeParis, meetup dédié au coding créatif et aux arts numériques :

<https://github.com/gregja/CreativeCodingParis>

Et parce qu'il n'y a pas que la programmation dans la vie, je jongle avec le peu de temps qui me reste pour composer de la musique, et pratiquer des arts martiaux.

Pour me contacter :

<https://www.linkedin.com/in/gregoryjarrige/>

Document publié sous licence Creative Commons N° 6 BY SA

La mise à jour en version 3 (avril 2022) du présent document vous est offerte par [LeDefrichoir.com](https://ledefrichoir.com) et www.foothing.net.

Image de couverture : forme 3D paramétrique de type hélicoïdale générée sous WebGL

Dépôt Github d'archivage de ce dossier : <https://github.com/gregja/phpLibrary4i>

1 Introduction

Le présent dossier explique comment utiliser un stack PHP installé sur un poste développeur (sous Windows, Linux ou Mac), avec un (ou plusieurs) serveur(s) de type IBM i.

Dans ce dossier, vous allez apprendre à configurer une connexion à la base de données DB2 for i, et vous allez apprendre à interagir avec votre IBM i au travers de requêtes SQL DB2 exécutées à partir de votre stack PHP.

Cette version 3 du présent dossier est une réactualisation, effectuée en avril 2022, à partir de la version 2 qui avait été publiée en 2009 sur le site Foothing.net (anciennement XDocs400). En 2009, on parlait encore de serveur iSeries (plutôt que IBM i) et la V7 de l'OS IBM i n'était pas encore arrivée (si mes souvenirs sont bons, elle arriva vers 2010). Cette V7 apporta – et continue à apporter au travers de ses différentes releases (V7R1, V7R2, etc.) - un certain nombre de nouveautés, qui rendent la vie des développeurs plus facile. La version 3 du présent dossier me donnera l'occasion de présenter certaines de ces nouveautés.

Pourquoi réactualiser ce vieux dossier poussiéreux ? Etant l'auteur je me suis posé la question de nombreuses fois, et je vous avoue que j'ai longtemps pensé que cette mise à jour était superflue, et n'intéresserait pas grand monde. Plus de 10 ans après la rédaction de la version 2 de ce dossier, je travaille aujourd'hui beaucoup plus avec Node.js qu'avec PHP. Mais je continue à utiliser un stack PHP pour de nombreux usages : pour tester des techniques, pour valider des concepts, pour maintenir du code, etc. Bref, cette configuration consistant en un stack PHP, relié aux serveurs IBM i via ODBC, me rend, aujourd'hui encore, de nombreux services. Or, je rencontre de plus en plus de jeunes développeurs IBM i, dont certains arrivent dans le monde IBM i avec un background de développeurs web. Ces jeunes développeurs (et développeuses) seraient intéressés par l'utilisation d'un stack PHP connecté à l'IBM i, mais faute de bien connaître l'écosystème IBM i, ils-elles ne savent pas exactement comment le mettre en place (quel stack utiliser ? quel connecteur ODBC utiliser ?). C'est en pensant à ces jeunes développeurs et développeuses, que j'ai décidé de retrousser mes manches, et de réactualiser le présent dossier.

Dans ce dossier, nous verrons quelques exemples d'utilisation du langage de développement PHP avec la base de données DB2 pour plateforme IBM i, au travers d'une connexion ODBC. Nous n'aborderons pas ici l'utilisation de PHP sur un stack Zend Server for IBM i, mais nous évoquerons brièvement certaines adaptations que vous devrez prévoir si vous envisagez de déplacer votre code, développé initialement sur un stack PHP sur PC, vers un stack PHP sur IBM i (sur le fameux Zend Server for i).

Les techniques présentées dans le présent dossier sont relativement faciles à mettre en œuvre. Pour pouvoir réaliser vous-même ce qui est présenté dans ce dossier, vous avez besoin :

- d'un serveur IBM i équipé d'un système d'exploitation, de préférence en V7 (V7R1, V7R2, ou plus), même si parmi les techniques que nous allons voir, beaucoup fonctionnent aussi sur des versions antérieures à la V7.
- d'un serveur web de type WAMP (Windows Apache MySQL PHP), ou LAMP (Linux Apache MySQL PHP), installé en local sur votre poste de développement. Il existe plusieurs produits concurrents. Nous allons en reparler dans instant.
- du logiciel ACS (Access Client Solutions) d'IBM : nous en avons besoin pour deux raisons :

Livre blanc sur l'utilisation de PHP en environnement IBM i

1. il embarque avec lui le driver « IBM i Access ODBC Driver », dont nous avons besoin pour connecter nos scripts PHP à DB2 for i.
 2. il fournit des assistants très pratiques, notamment pour le transfert de fichiers, la création de procédures stockées DB2, etc... ainsi que d'excellents outils de monitoring permettant d'analyser les performances des requêtes SQL.
- d'un éditeur de code source PHP, de préférence de type IDE (environnement de développement intégré). Il existe une multitude de très bons produits, dont certains sont gratuits. Parmi les très bons éditeurs gratuits, Visual Studio Code de Microsoft se situe dans le haut du classement. Parmi les solutions payantes de très bon niveau, on peut citer le logiciel PHPStorm de la société JetBrains.

Concernant les stacks PHP pour PC :

- sur Linux : sauf cas particulier, je vous recommande de privilégier le stack PHP fourni par défaut avec votre distribution Linux

- sur Mac : prenez le temps de vous documenter et de lire la doc fournie par php.net
<https://www.php.net/manual/en/install.macosx.bundled.php>

- sur Windows :

- le projet XAMPP de ApacheFriends est un très bon choix :
<https://www.apachefriends.org>
- le projet Laragon, que j'ai découvert en 2020, est très intéressant également, d'autant qu'il propose une version « portable » (pouvant tourner sur une simple clé USB). Mais surtout, il permet de swapper très facilement entre différentes versions du langage PHP, ce qui permet de s'aligner facilement avec le PHP d'un Zend Server for i (ce que j'explique en annexe) : <https://laragon.org>

Quelques remarques :

- Ce dossier n'est pas un vrai tutorial, dans le sens où il ne vous apprendra pas à utiliser un stack PHP, pas plus qu'à programmer en PHP, ou en SQL, ou en RPG. L'utilisation optimale de ce dossier suppose que vous ayez un minimum de connaissances préalable dans les domaines précités vous permettant d'utiliser ces différents outils et langages. La non connaissance de l'un de ces domaines n'est cependant pas rédhibitoire, elle ne devrait pas vous empêcher de comprendre les concepts présentés dans ce dossier, même si vous n'êtes pas en mesure de les appliquer immédiatement.
- Le style de programmation utilisé pour les scripts PHP est « old school », je veux dire par là qu'il est écrit de manière très procédurale, et qu'il n'obéit pas à une stricte séparation entre couche d'accès aux données et couche de présentation. J'ai laissé volontairement ces scripts, en l'état, parce que ce qui importe ici, c'est de maîtriser la mécanique de communication entre PHP et DB2 for i. Le style de programmation PHP utilisé pour le rendu final est ici purement accessoire. Le point essentiel, c'est que les requêtes SQL soient écrites dans les règles de l'art (c'est-à-dire correctement sécurisées).
- les exemples de programme RPG présentés dans ce dossier sont également « old school ». Par manque de temps, et parce que cela ne présentait pas un grand intérêt pour la compréhension du sujet, je ne les ai pas convertis en RPG Free.

2 Premiers contacts avec PHP

Pour nos accès à la base de données DB2 for i avec le langage PHP, nous allons utiliser PDO. La documentation officielle de PDO propose la définition suivante :

« L'extension PHP Data Objects (PDO) définit une excellente interface pour accéder à une base de données depuis PHP. Chaque pilote de base de données implémenté dans l'interface PDO peut utiliser des fonctionnalités spécifiques de chacune des bases de données en utilisant des extensions de fonctions. Notez que vous ne pouvez exécuter aucune fonction de base de données en utilisant l'extension PDO par elle-même ; vous devez utiliser un driver PDO spécifique à la base de données pour accéder au serveur de base de données.

PDO fournit une interface d'abstraction à l'accès de données, ce qui signifie que vous utilisez les mêmes fonctions pour exécuter des requêtes ou récupérer les données quelle que soit la base de données utilisée. PDO ne fournit pas une abstraction de base de données : il ne réécrit pas le SQL, n'émule pas des fonctionnalités manquantes. Vous devriez utiliser une interface d'abstraction complète si vous avez besoin de cela.

PDO est fourni avec PHP 5.1 et est disponible en tant qu'extension PECL pour PHP 5.0 ; PDO requiert les nouvelles fonctionnalités OO fournies par PHP 5 et donc, ne fonctionne pas avec les versions antérieures de PHP. »

Pour pouvoir utiliser PDO avec notre stack PHP, il faut que l'extension « php_pdo_odbc » soit activée (c'est à dire décommentées) dans le fichier php.ini de votre stack PHP. La plupart des stacks pour PHP sous Windows fournissent une interface graphique permettant de modifier la configuration du fichier php.ini, mais si vous avez localisé l'emplacement du fichier, vous pouvez aussi le modifier manuellement.

Dans un fichier de configuration « php.ini » type, on trouve les lignes suivantes :

```
...
extension=php_mysql.dll
extension=php_mysqli.dll
;extension=php_oci8_12c.dll ; Use with Oracle Database 12c Instant Client
extension=php_openssl.dll
;extension=php_pdo_firebird.dll
extension=php_pdo_mysql.dll
;extension=php_pdo_oci.dll
extension=php_pdo_odbc.dll
;extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
;extension=php_pgsql.dll
extension=php_shmop.dll
...
```

Décommentez la ligne en rouge, si ce n'est pas déjà fait, et redémarrez votre stack PHP (en particulier Apache), pour que la modification soit prise en compte.

NB : dans la version 2 de ce dossier, j'indiquais qu'il fallait activer aussi l'extension « php_pdo », mais ce n'est plus vrai aujourd'hui, car le composant PDO est désormais embarqué en standard dans le noyau de PHP, et il ne nécessite plus d'activation spécifique.

3 Préparation d'une table DB2 contenant la liste des pays

Pour nos premiers scripts PHP, je vous propose de créer sur votre serveur IBM i une nouvelle table DB2 contenant la liste des pays. J'ai récupéré cette liste de pays sur Wikipedia, sous la rubrique ISO_3166-1. Après avoir copié-collé le tableau des pays dans un éditeur texte pour le nettoyer de ses attributs HTML, je l'ai collé dans Excel, pour ne récupérer que les colonnes suivantes :

	A	B	C	D
1	Nom français	Nom ISO	Norme	Désignation
2	AFG	AF	(ISO 3166-2)	AFGHANISTAN
3	ZAF	ZA	(ISO 3166-2)	AFRIQUE DU SUD
4	ALA	AX	(ISO 3166-2)	ALAND, ILES
5	ALB	AL	(ISO 3166-2)	ALBANIE
6	DZA	DZ	(ISO 3166-2)	ALGERIE
7	DEU	DE	(ISO 3166-2)	ALLEMAGNE
8	AND	AD	(ISO 3166-2)	ANDORRE
9	AGO	AO	(ISO 3166-2)	ANGOLA
10	AIA	AI	(ISO 3166-2)	ANGUILLA
11	ATA	AQ	(ISO 3166-2)	ANTARCTIQUE
12	ATG	AG	(ISO 3166-2)	ANTIGUA-ET-BARBUDA
13	ANT	AN	(ISO 3166-2)	ANTILLES NEERLANDAISES

Vous pouvez créer la table des pays sur votre IBM i en utilisant DDS, mais je vous propose de le faire plutôt en passant par SQL. Pour cela, démarrez une session SQL (via la commande STRSQL), puis exécutez les requêtes suivantes (remplacez « MABIB » par la bibliothèque que vous souhaitez utiliser pour vos tests) :

```
CREATE TABLE MABIB/LSTPAYS (  
  CODFRA CHAR (3 ) NOT NULL WITH DEFAULT,  
  CODISO CHAR (2 ) NOT NULL WITH DEFAULT,  
  LIBELLE CHAR (50 ) NOT NULL WITH DEFAULT  
)
```

Vous pouvez si vous le souhaitez ajouter un libellé à votre table LSTPAYS avec la requête suivante :

```
COMMENT ON TABLE LSTPAYS IS 'TABLE DES PAYS' ;
```

Je vous propose également d'ajouter à votre table les indexs ci-dessous, qui vous permettront d'obtenir des requêtes SQL plus performantes :

```
CREATE INDEX MABIB/LSTPAYSL01 ON MABIB/LSTPAYS (CODFRA) ;  
CREATE INDEX MABIB/LSTPAYSL02 ON MABIB/LSTPAYS (CODISO) ;  
CREATE INDEX MABIB/LSTPAYSL03 ON MABIB/LSTPAYS (LIBELLE, CODFRA) ;
```

Vous pouvez exécuter ces requêtes de création sur DB2 Express C, en remplaçant par un « . », le « / » qui est spécifique à DB2 for i.

A titre d'information, le code SQL pour la création de la même table sous MySQL se présente de la façon suivante (remplacer « mabase » par la BD MySQL de votre choix) :

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
CREATE TABLE mabase.lstpays (
    codfra CHAR( 3 ) NOT NULL DEFAULT ' ',
    codiso CHAR( 2 ) NOT NULL DEFAULT ' ',
    libelle CHAR( 50 ) NOT NULL DEFAULT ' '
) ENGINE = MYISAM CHARACTER SET utf8 COMMENT = 'table des pays' ;

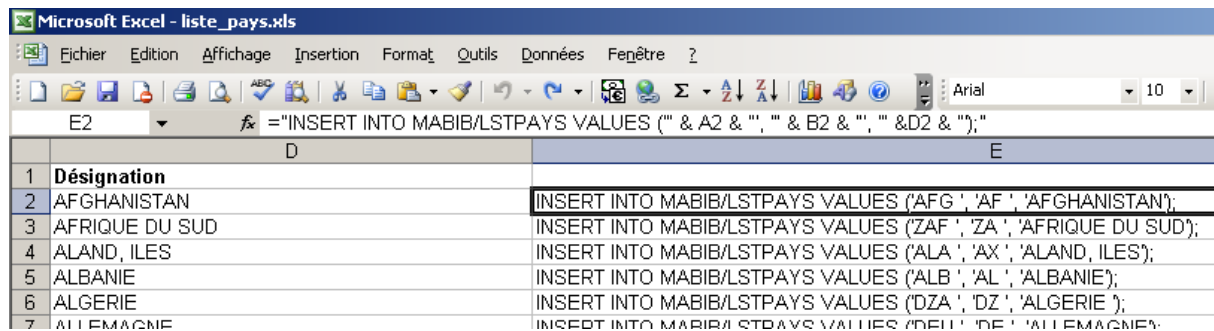
CREATE INDEX lstpaysl01 ON mabase.lstpays(codfra);
CREATE INDEX lstpaysl02 ON mabase.lstpays(codiso);
CREATE INDEX lstpaysl03 ON mabase.lstpays(libelle, codfra);
```

Revenons maintenant à notre fichier Excel pour le finaliser : nous souhaitons récupérer les colonnes A, B et D. Pour ce faire, nous allons créer une colonne supplémentaire dans laquelle nous allons utiliser une formule de concaténation, de manière à créer des requêtes INSERT pour chaque ligne du tableau Excel. Concrètement, la requête de concaténation se présente de la façon suivante :

```
= "INSERT INTO MABIB/LSTPAYS VALUES ('" & A2 & "', '" & B2 & "', '" & D2 & "');" "
```

Dès que la requête fonctionne pour une ligne, vous pouvez la dupliquer sur toutes les lignes en-dessous.

Il me semble que c'est plus parlant quand on le voit en action sur Excel :



	D	E
1	Désignation	
2	AFGHANISTAN	INSERT INTO MABIB/LSTPAYS VALUES ('AFG ', 'AF ', 'AFGHANISTAN');
3	AFRIQUE DU SUD	INSERT INTO MABIB/LSTPAYS VALUES ('ZAF ', 'ZA ', 'AFRIQUE DU SUD');
4	ALAND, ILES	INSERT INTO MABIB/LSTPAYS VALUES ('ALA ', 'AX ', 'ALAND, ILES');
5	ALBANIE	INSERT INTO MABIB/LSTPAYS VALUES ('ALB ', 'AL ', 'ALBANIE');
6	ALGERIE	INSERT INTO MABIB/LSTPAYS VALUES ('DZA ', 'DZ ', 'ALGERIE ');
7	ALLEMAGNE	INSERT INTO MABIB/LSTPAYS VALUES ('DEU ', 'DE ', 'ALLEMAGNE');

Il faut ensuite copier le contenu de la colonne E dans le presse-papier, puis effectuer un « collage spécial » vers une colonne vierge d'Excel en utilisant l'option « par valeurs ». Vous obtenez ainsi un script SQL d'insertion de toutes les lignes du tableau Excel.

J'allais oublier un détail important : le problème des quotes, ou apostrophes. Je vous invite à regarder la colonne E se situant en face de la ligne de la « COTE D'IVOIRE ». Normalement, vous devriez avoir ceci :

```
INSERT INTO MABIB/LSTPAYS VALUES ('CIV ', 'CI ', 'COTE D'IVOIRE ');
```

Si vous y regardez de près, vous constaterez que vous avez un nombre d'apostrophes impair entre les parenthèses du mot-clé SQL VALUES. Donc cette requête SQL ne pourra pas fonctionner, pas plus sous DB2 que sous MySQL. Et on retrouve le même problème sur tous les libellés contenant une ou plusieurs apostrophes.

Vous pouvez corriger facilement le problème en utilisant les options de remplacement de chaînes d'Excel. Ainsi, vous allez demander à Excel de remplacer les apostrophes simples par 2 apostrophes (attention, j'ai bien dit « par 2 apostrophes », et non pas « par des guillemets »).

Après correction, la requête d'insertion pour la Côte d'Ivoire ressemblera à ceci :

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
INSERT INTO MABIB/LSTPAYS VALUES ('CIV ', 'CI ', 'COTE D'IVOIRE ');
```

La correction étant effectuée pour l'ensemble des lignes du tableau, vous pouvez recopier la colonne E dans le presse-papier Windows, puis effectuer un « collage spécial par valeur » dans une colonne vierge d'une autre feuille Excel.

Vous disposez maintenant d'un script SQL d'insertion prêt à l'emploi vous permettant d'alimenter la table SQL des pays, que vous pouvez sauvegarder au format texte. Vous pouvez ainsi le transférer sur votre serveur IBM i par tout moyen à votre convenance (par exemple FTP). Pensez à remplacer dans le script SQL « MABIB » par la bibliothèque de votre choix avant d'effectuer le transfert sur votre plateforme i. Copiez le contenu du fichier dans un membre de fichier source (par exemple : MABIBSRC/QSQLSRC), dans le membre LSTPAYS, puis exécutez le script via la commande OS/400 RUNSQLSTM, de la façon suivante :

```
RUNSQLSTM SRCFILE(MABIBSRC/QSQLSRC) SRCMBR(LSTPAYS) COMMIT(*NONE)
```

Vous risquez de rencontrer quelques problèmes dûs à des requêtes INSERT trop longues, que la commande RUNSQLSTM ne supporte pas. Certaines requêtes devront donc être légèrement modifiées de façon à être saisies sur 2 lignes, comme par exemple :

```
INSERT INTO LSTPAYS VALUES ('COD ', 'CD ',  
                             'CONGO, LA REPUBLIQUE DEMOCRATIQUE DU ');
```

Si tout s'est bien passé, vous disposez donc d'une table DB2 contenant la liste des pays. Dans le chapitre suivant, nous allons voir comment afficher son contenu de manière dynamique via PHP.

Avant de poursuivre, je crois important de souligner que les exemples que vous allez voir dans la suite de ce dossier fonctionnent aussi bien sur des tables DB2 créées via SQL, que sur des fichiers DB2 créés via DDS. Vous n'avez donc aucune obligation de travailler sur une base de données « full SQL ». Cependant, vous pouvez être mis en difficulté par 2 types de fichiers DB2/400 :

- les fichiers multi-formats : incompatibles avec SQL.
- les fichiers multi-membres : SQL ne sait pas gérer directement les fichiers multi-membres, mais on peut contourner le problème en utilisant des alias SQL (mais ce point est hors du périmètre du présent dossier).

Dans ce chapitre, j'ai souhaité démarrer très progressivement en vous montrant une méthode de création de script SQL très simple et très « manuelle ». On peut aboutir au même résultat en recourant au langage PHP de différentes manières, comme par exemple :

- *solution 1 : à partir de la liste des pays au format CSV, développer un script PHP effectuant une reprise des données vers une table MySQL ou DB2. Vous trouverez plusieurs variantes de cette technique dans l'annexe, au chapitre 14.3.*
- *solution 2 : exporter la liste des pays au format XML via Excel, et développer un script PHP effectuant une reprise des données vers une table MySQL ou DB2. Pour que cela fonctionne, l'export XML doit être effectué avec une version 2003 ou supérieure d'Excel. PHP fournit en standard des mécanismes simples et puissants de traitement du format XML. Je ne vous donnerai pas d'exemple ici car il s'agit de techniques avancées (quoique pas très compliquées) qui dépassent le cadre du présent dossier.*
- *solution 3 : extraire la liste des pays directement à partir de la page HTML. C'est légèrement plus compliqué que le traitement de fichier XML, mais c'est néanmoins tout à fait réalisable en PHP.*

4 Premier script PHP affichant la liste des pays

Entrons tout de suite dans le vif du sujet avec le script « testlstpays.php » ci-dessous :

```
<?php
1 $user      = 'xxxxxxxxx'; // profil utilisateur AS/400
  $password  = 'yyyyyyyyy'; // mot de passe du profil utilisateur AS/400
  $database  = 'zzzzzzzzz' ; // nom de la bib contenant la table LSTPAYS
  $system    = '999.999.999.999' ; // adresse IP de l'AS/400
  $trace     = 0 ; // mode trace DB2 désactivé par défaut
  $dsn = "odbc:DRIVER={IBM i Access ODBC Driver};
        SYSTEM=$system;DBQ=$database;TRACE=$trace";

2 try {
    $conn = new PDO($dsn, $user, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  } catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage() . '<p/>';
  }

3 echo 'Liste des pays : <p/>' ;

  echo <<<BOTABLE
    <table border='1'>
      <tr>
        <td>CODE FRANCE</td>
        <td>CODE ISO</td>
        <td>LIBELLE</td>
      </tr>
    BOTABLE;

4 $sql = 'SELECT CODFRA, CODISO, LIBELLE FROM LSTPAYS ' ;

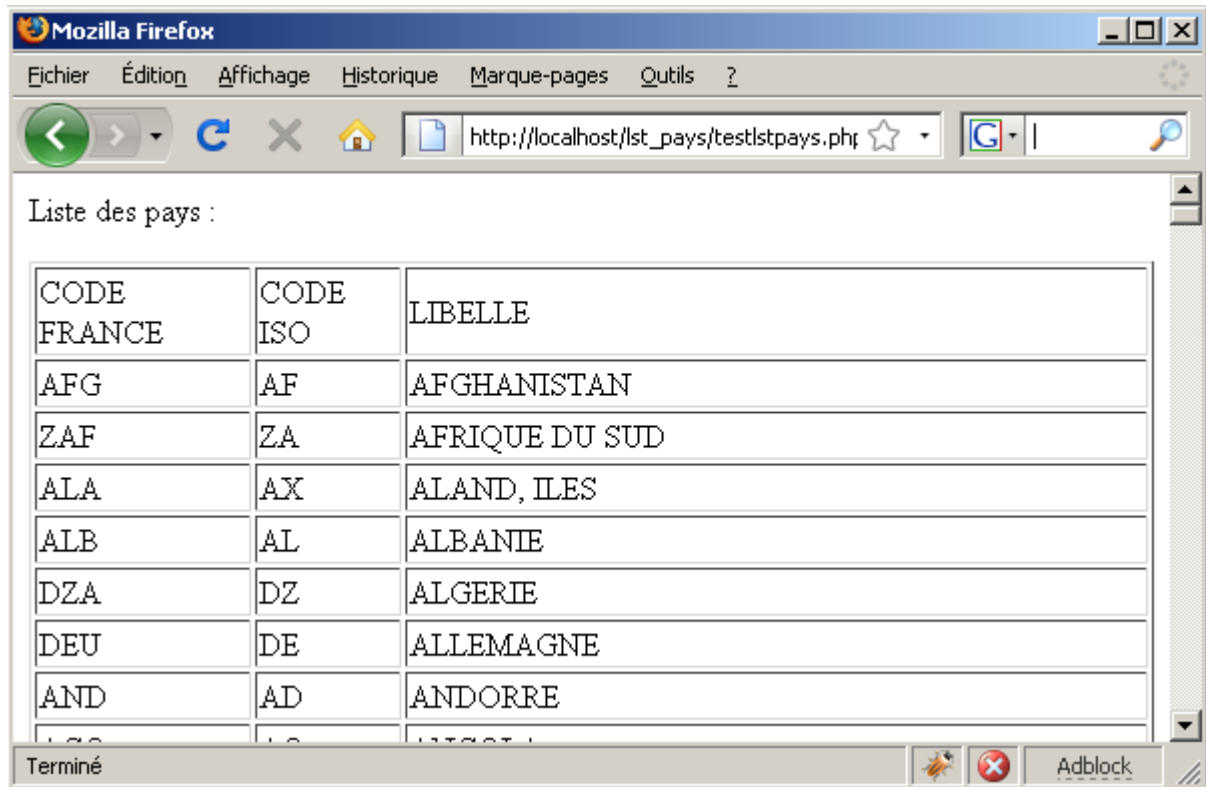
5 try {
    $st = $conn->query($sql);
    while($row_data = $st->fetch(PDO::FETCH_ASSOC)) {
      echo <<<LOTABLE
        <tr>
          <td>{$row_data['CODFRA']}</td>
          <td>{$row_data['CODISO']}</td>
          <td>{$row_data['LIBELLE']}</td>
        </tr>
      LOTABLE;
    }
  } catch (PDOException $e) {
6   echo 'Error : ' . $e->getMessage() . '<br/>';
   echo 'Code : ' . $e->getCode() . '<br/>';
   echo 'File : ' . $e->getFile() . '<br/>';
   echo 'Line : ' . $e->getLine() . '<br/>';
   echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
  }

7 echo <<<EOTABLE
    </table>
  EOTABLE;
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

?>

Le résultat obtenu dans le navigateur ressemble à ceci :



...

Chacune des parties du script de la page précédente est détaillé ci-dessous :

1	<p>Cette partie contient les paramètres de connexion à la base de données. La connexion est de type ODBC, la variable \$dsn regroupe l'ensemble des paramètres de connexion, dont le nom du driver ODBC, le système, la base de données et le mode trace DB2 (activé ou non).</p> <pre>\$dsn = "odbc:DRIVER={IBM i Access ODBC Driver}; SYSTEM=\$system;DBQ=\$database;TRACE=\$trace";</pre> <p>A noter : le mode TRACE est spécifique au driver DB2 for i, il n'existe pas d'équivalent sur DB2 LUW (Linux Unix Windows).</p> <p>Si je souhaite me connecter à une base de données DB2 pour LUW, installée en local sur mon poste, le contenu de \$dsn sera légèrement différent :</p> <pre>\$dsn = "odbc:DRIVER={IBM DB2 ODBC DRIVER};HOSTNAME=localhost; PORT=50000;PROTOCOL=TCPIP;DATABASE=\$database ";</pre> <p>On remarque qu'avec DB2 pour LUW, le nom de la base de données est défini sur le paramètre DATABASE (et non DBQ).</p> <p>Le script de la page précédente peut être réutilisé quasiment à l'identique avec la même table LSTPAYS créée sur une base de données MySQL. Pour cela, il suffit de modifier légèrement les paramètres de connexion. Par exemple, avec Wampserver exécuté en local, les paramètres de connexion deviennent :</p> <pre>\$user = 'root'; // profil utilisateur « root » par défaut \$password = ''; // password du profil « root » sur MySQL \$database = 'mabase'; // nom de la bd MySQL sélectionnée \$system = 'localhost'; // utilisation de Wampserver en local</pre>
---	--

	<code>\$dsn = "mysql:dbname=\$database;host=\$system";</code>
2	<p>La connexion à la base de données est réalisée par le code situé à l'intérieur du paragraphe <code>try { }</code>. En cas d'échec de la connexion, PHP « débranche » automatiquement sur le paragraphe <code>catch { }</code> et renvoie un message d'erreur. Cette manière de gérer les exceptions est apparue avec la V5.1 de PHP, et elle se révèle extrêmement puissante à l'usage. Nous la retrouverons plus loin dans le même script.</p> <p>Les paramètres <code>PDO::ATTR_ERRMODE</code> et <code>PDO::ERRMODE_EXCEPTION</code> sont importants à noter. Leur invocation via la méthode <code>setAttribute()</code> permet d'activer la gestion des exceptions sur PDO. Cela permet de récupérer un maximum d'informations en cas d'échec de la requête SQL, via la gestion des exceptions de PHP (<code>try/catch</code>).</p>
3	<p>Code PHP encapsulant un peu de code HTML pour l'affichage de l'entête de la liste des pays. On utilise ici une particularité syntaxique de PHP, à savoir la définition de chaîne par bloc. Le début du bloc est symbolisé par <code><<<</code> suivi du nom que j'ai attribué arbitrairement à ce bloc, en l'occurrence « BOTABLE » pour « Begin Of Table ». La fin du bloc est symbolisée par le nom du même bloc placé obligatoirement sur le premier caractère de la ligne de fin de bloc (<code>BOTABLE;</code>).</p> <pre>echo <<<BOTABLE // début du bloc ... BOTABLE; // fin du bloc</pre> <p>Cette manière de définir une chaîne de caractères n'est pas la seule façon de faire, il en existe beaucoup d'autres en PHP. Je la trouve très pratique, notamment sur de petits programmes comme celui-ci. Mais elle peut poser des problèmes avec certains éditeurs de code source PHP, dont les options de formatage de code ne savent pas indenter correctement ce type de code. Il est vrai également que la contrainte consistant à placer la fin du bloc sur le premier caractère de la ligne a pour effet de créer une rupture dans l'indentation du code, et nuit quelque peu à sa lisibilité.</p>
4	<p>Définition de la requête SQL dans la variable <code>\$sql</code>. La requête présentée ici est très simple. Nous verrons dans la suite de ce document différentes variantes et la manière de les implémenter sous PDO.</p>
5	<p>La boucle de chargement à l'intérieur du paragraphe « <code>try { }</code> » pilote un curseur SQL sur la table <code>LSTPAYS</code>. La fonction <code>fetch()</code> permet de manipuler chaque ligne renvoyée par le curseur individuellement. Chaque colonne de la table est identifiée par PHP comme un poste de tableau grâce à la présence du paramètre « <code>FETCH_ASSOC</code> » dans la ligne suivante :</p> <pre>while(\$row_data = \$st->fetch(PDO::FETCH_ASSOC))</pre> <p>cela permet d'adresser et d'afficher chaque colonne renvoyée par le curseur, en utilisant le nom des colonnes comme des postes de tableau, comme dans l'exemple ci-dessous :</p> <pre>\$resultat = \$row_data['CODFRA'] ;</pre> <p>A noter que chaque itération de la boucle affiche une ligne du curseur au moyen d'une chaîne délimitée par le bloc que j'ai appelé arbitrairement « LOTABLE »</p>

	<p>(pour « Line Of Table »)</p> <pre><<<LOTABLE ... LOTABLE ;</pre> <p>Il est intéressant de noter que l'on peut remplacer le paramètre <code>FETCH_ASSOC</code> par <code>FETCH_LAZY</code> comme dans l'exemple ci-dessous :</p> <pre>while(\$row_data = \$st->fetch(PDO::FETCH_LAZY))</pre> <p>Le paramètre <code>FETCH_LAZY</code> a pour effet de créer une instance d'objet dite anonyme. En pratique, cela permet d'adresser et d'afficher chaque colonne renvoyée par le curseur comme une propriété de l'instance <code>\$row_data</code> de l'objet <code>PDORow</code>. On peut dès lors adresser chaque propriété de cette instance de la façon suivante :</p> <pre>\$resultat = \$row_data->CODFRA ;</pre>
6	<p>Le paragraphe <code>catch{ }</code> permet ici de voir l'ensemble des informations que l'on peut récupérer en cas d'anomalie lors de l'ouverture ou de l'exécution du curseur (ou de toute autre requête SQL). Un bon moyen pour voir ce que ce paragraphe est en mesure d'afficher consiste à déclencher une exception en glissant une erreur de syntaxe à l'intérieur de la requête SQL (cf. variable <code>\$sql</code>).</p>
7	<p>Affichage d'une chaîne délimitée par un nouveau bloc pour « fermer » proprement le tableau HTML affiché par le script PHP.</p>

Dans ce chapitre, j'ai présenté les syntaxes de définition des connexions ODBC pour MySQL et DB2. Il s'agit là de techniques utilisables pour des connexions temporaires. Rien ne vous interdit de définir une connexion ODBC permanente sur votre PC de développement. Ainsi, si votre connexion ODBC permanente s'appelle « MABASE », l'alimentation de la variable \$dsn s'en trouvera simplifiée puisque l'on écrira simplement ceci :

```
$dsn = "odbc:MABASE" ;
```

A noter que, dans le script présenté ici, je suis parti du principe que le mot de passe du profil "root" était à blanc. Mais par sécurité, je vous recommande de modifier ce mot de passe comme indiqué au chapitre 17.2.

5 Variante du 1^{er} script PHP

5.1 Variante avec critères de tri (ORDER BY)

Supposons que l'on souhaite faire évoluer le script précédent pour afficher les pays sur l'une ou l'autre des colonnes de la table LSTPAYS, on pourrait le faire très facilement en ajoutant quelques lignes de code. Voici un exemple possible consistant à moduler la requête SQL par concaténation, en fonction du contenu de la variable \$choix_tri (alimentée en amont en fonction du choix de l'utilisateur) :

```
...

$sql = 'SELECT CODFRA, CODISO, LIBELLE FROM LSTPAYS ' ;

switch ( $choix_tri ) {
    case 'FRA': {
        // tri sur le code FRANCE
        $sql .= 'ORDER BY CODFRA' ;
        break;
    }
    case 'ISO': {
        // tri sur le code ISO
        $sql .= 'ORDER BY CODISO';
        break;
    }
    case '3': {
        // tri sur le libellé
        $sql .= 'ORDER BY LIBELLE';
        break;
    }
    default:{
        // pas de tri par défaut, ce pavé "default" est en fait inutile
    }
}

try {
    $st = $conn->query($sql);
    while($row_data = $st->fetch(PDO::FETCH_ASSOC)) {
        etc...
    }
}

...
```

5.2 Variante avec critères de sélection (WHERE)

Supposons que – suite à une subite envie d'exotisme - on souhaite afficher la liste des pays contenant le mot « ILE », on pourrait le faire très facilement avec les quelques lignes de code suivantes :

```
...
/* les variables $choix_col et $choix_val pourraient être récupérées
   via un formulaire, mais pour la simplicité de l'exemple, on les
   alimente ici arbitrairement */
$choix_col = 'LIBELLE' ;
$choix_rech = 'LIKE' ;
$choix_val = '%ILE%' ;

$sql = 'SELECT CODFRA, CODISO, LIBELLE FROM LSTPAYS ' ;

if ($choix_col != '') {
    $sql .= ' WHERE ' . $choix_col .
           ' ' . $choix_rech . ' ? ' .
           ' ORDER BY CODFRA ' ;
}

try {
    $st = $conn->prepare($sql);
    $st->execute(array($choix_val));
    while($row_data = $st->fetch(PDO::FETCH_ASSOC)) {
        echo <<<EOM
        <tr>
            <td>{$row_data['CODFRA']}
```

Notre requête SQL a donc la forme suivante, après concaténation :

```
SELECT CODFRA, CODISO, LIBELLE FROM LSTPAYS WHERE LIBELLE LIKE ?
```

Mais toute la beauté de la chose réside dans les 2 lignes suivantes :

```
$st = $conn->prepare($sql);
$st->execute(array($choix_val));
```

On retrouve ici les mêmes ordres SQL PREPARE et EXECUTE qu'en SQLRPGLE. Et on s'appuie sur les mécanismes de parsing et de substitution des paramètres marqueurs de DB2 SQL (qui sont d'ailleurs identiques dans le principe à ceux de MySQL).

Livre blanc sur l'utilisation de PHP en environnement IBM i

La fonction PDO prepare() demande donc à DB2 de « parser » la requête SQL et de rechercher le chemin d'accès le plus adéquat pour son exécution. La fonction execute() reçoit un tableau en paramètre, qui peut être composé de 0, 1 ou x éléments. Dans l'exemple ci-dessus, ce tableau contient un seul poste qui est la variable \$choix_val. La fonction execute() indique en substance à l'analyseur de requête de DB2 de remplacer le point d'interrogation par le contenu de la variable \$choix_val.

Si on avait eu 2 critères de sélection, comme dans l'exemple suivant :

```
SELECT CODFRA, CODISO, LIBELLE FROM LSTPAYS WHERE CODFRA = ? OR CODISO = ?
```

... alors on aurait pu écrire ceci :

```
$st = $conn->prepare($sql);  
$st->execute(array('FRA', 'FR'));
```

(cette fois-ci j'ai choisi de passer un tableau contenant 2 chaînes plutôt que 2 variables, simplement pour vous montrer que c'est possible. Je vous laisse deviner ce que cette requête aurait renvoyé).

A titre de comparaison, voici à quoi pourrait ressembler, converti en SQLRPGLE, le script PHP de la page précédente :

```
H DECEDIT('0,') DATEDIT(*YMD)  DEBUG  
*-----*  
*  DECLARATION DES VARIABLES  
*-----*  
D $Requete          S          1000  
D $Col_Rech         S           20  
D $Lib_Rech         S           20  
D $Typ_Rech         S           20  
D WK_DS_PAYS        E DS              extname(LSTPAYS)  
//  
// Préparation des directives de compilation et d'exécution SQL  
C/Exec SQL  
C+ SET OPTION DATFMT = *ISO, TIMFMT = *ISO , CLOSQLCSR = *ENDMOD  
C/End-Exec  
*-----*  
*  TRAITEMENT PRINCIPAL  
*-----*  
/free  
    $Col_Rech = 'LIBELLE' ;  
    $Typ_Rech = 'LIKE' ;  
    $Lib_Rech = '%ILE%';  
    $Requete = 'Select * From LSTPAYS ' +  
                'Where ' + %trim($Col_Rech) + ' ' +  
                %trim($Typ_Rech) + ' ? ' +  
                'Order By CODFRA ' ;  
  
/end-free  
*  
*  Préparation du curseur  
C/EXEC SQL  
C+ PREPARE REQ1 FROM :$Requete  
C/END-EXEC  
*  Déclaration du curseur  
C/EXEC SQL  
C+ DECLARE CUR1 CURSOR FOR REQ1  
C/END-EXEC  
*  Ouverture et exécution du curseur avec parsing de la requête  
*  et remplacement du ? par la variable $Lib_Rech  
C/EXEC SQL
```


Livre blanc sur l'utilisation de PHP en environnement IBM i

```
C+ OPEN CUR1 USING :$Lib_Rech
C/END-EXEC
*
* Lecture de la 1ère ligne renvoyée par le curseur
C/EXEC SQL
C+ FETCH FROM CUR1 INTO :WK_DS_PAYS
C/END-EXEC
C          Z-add      0          WI          1200
C          Dow        SQLCOD = *zero
* Pour les besoins du test, on a ici placé un simple compteur
* mais on pourrait tout aussi bien alimenter le sous-fichier
* d'un programme interactif ou le format d'édition d'un
* programme batch
C          Add        1          WI
*
* Ligne suivante du curseur (équivalent d'un Read en RPG)
C/EXEC SQL
C+ FETCH FROM CUR1 INTO :WK_DS_PAYS
C/END-EXEC
C          Enddo
*
* Dump pour contrôler la valeur du compteur WI en fin de traitement
C          Dump
*
C          Seton          LR
```

Le programme RPG ci-dessus est écrit dans un style compatible avec le compilateur RPG de la V5R3 de l'OS/400. Je me rends compte en me relisant que je n'ai pas utilisé exactement les mêmes noms de variables que dans le script PHP, j'espère que vous ne m'en tiendrez pas rigueur. Pour ne pas surcharger le code source du programme RPG, je n'ai pas intégré de gestion d'exception SQL.

Au-delà des différences de syntaxe et de déclaration, on voit bien ici que la méthode de programmation du curseur est strictement la même en PHP et en RPG. Quand vous savez programmer un curseur SQL dynamique en RPG, vous savez le faire aussi en PHP, et inversement. De plus, comme ce script PHP fonctionne également avec DB2 pour LUW (Linux Unix Windows) et avec MySQL, simplement en modifiant les paramètres de connexion ODBC, vous êtes dès maintenant en mesure de travailler avec des bases de données autres que DB2 for i.

6 Approfondissements autour de DB2

Dans le présent chapitre, je souhaite revenir sur les exemples PHP et RPG du chapitre précédent, en apportant quelques précisions et recommandations.

6.1 N'abusez pas du parseur SQL

On remarque dans les exemples présentés ici, aussi bien en PHP qu'en RPG, que je n'utilise jamais les paramètres marqueurs (?) pour faire des substitutions de noms de tables, de noms de colonnes, ou de critères de comparaison. Ces éléments doivent toujours être ajoutés à la requête SQL par concaténation, et surtout pas en utilisant le parseur de SQL. Il serait par exemple tentant d'écrire la requête précédente ainsi :

```
Select * From LSTPAYS Where ? = ? Order By CODFRA
```

... ou pire encore :

```
Select * From ? Where ? = ? Order By ?
```

... mais le parseur SQL a ses limites. Si le parseur de DB2 a toutes les chances de vous renvoyer un code erreur SQL sur chacune des 2 requêtes ci-dessus, le parseur d'autres bases de données sera peut être plus tolérant. Mais en règle générale, vous risquez d'obtenir des requête peu performantes si vous les programmez de cette façon.

6.2 Principe du SQL dynamique

Quand on utilise les mécanismes de préparation et d'exécution différées du SQL dynamique, comme c'est le cas dans nos deux programmes PHP et RPG, on donne un signal important à l'analyseur de requête du moteur de base de données : lors de la phase de préparation, l'analyseur de requête recherche dans sa base de données interne - appelée « cache de plan d'accès » dans le jargon DB2 - s'il a déjà exécuté une requête similaire. Si c'est le cas, il va réutiliser le chemin d'accès utilisé lors de l'exécution précédente. Dans le cas contraire, l'optimiseur SQL va calculer un chemin d'accès et le stocker dans le cache de plan d'accès en vue d'une utilisation ultérieure. Le mécanisme que je viens de décrire sommairement pour DB2 fonctionne à peu près de la même manière avec MySQL, du moins dans ses versions supérieures à la version 4. Il existe bien évidemment des exceptions au principe que je viens de décrire : par exemple, si votre requête adresse successivement la même table dans des bibliothèques différentes, le chemin d'accès à la table ne cessant de changer d'une exécution à l'autre, DB2 ne sera pas en mesure d'optimiser l'exécution de la requête à partir du cache de plan d'accès.

6.3 Avantages des paramètres marqueurs

Les paramètres marqueurs offrent une grande souplesse dans l'écriture de requêtes SQL, en vous épargnant la tâche fastidieuse consistant à concaténer les critères de recherche à l'intérieur de la requête (l'analyseur de requête de DB2 fait le travail à votre place)..

De plus, dès lors qu'une requête doit être exécutée plusieurs fois, le fait d'utiliser des paramètres marqueurs permet de préparer la requête une fois, et de l'exécuter plusieurs fois avec des valeurs différentes. Il en résulte un gain de temps appréciable puisque le temps de préparation de la requête n'est « consommé » qu'une fois. On reverra ce principe avec le programme de reprise présenté au chapitre suivant.

Enfin un autre avantage à signaler, et non des moindres, est lié au fait que l'utilisation des paramètres marqueurs vous permet de protéger vos requêtes des attaques par injection SQL, qui sont une vraie plaie pour la sécurité des sites internet. Vous trouvez sur internet de nombreux sites traitant du problème des attaques par injection SQL, je ne m'étendrai donc pas sur le sujet ici.

Il est également intéressant de savoir que vous pouvez remplacer les points d'interrogations à l'intérieur de la requête par des paramètres nommés, associés à la méthode PDO bind_param. Cette technique peut améliorer la lisibilité des requêtes, quand elle utilisent beaucoup de colonnes en substitution (comme c'est souvent le cas avec les requêtes d'insertion). Nous parlerons brièvement de la méthode bind_param dans le chapitre consacré au « dialogue » entre script PHP et programme RPG. Pour de plus amples informations sur cette méthode, je vous invite à vous reporter aux différents sites et livres que je vous recommande à la fin de ce dossier.

6.4 Outils d'analyse des performances SQL

DB2 pour IBM i offre plusieurs outils d'analyse de performances. A commencer par le paramètre TRACE, que nous avons défini dans la variable \$dsn utilisée comme paramètre de connexion ODBC de notre script PHP. Ce paramètre TRACE est à priori une spécificité du driver ODBC pour IBM i, je n'ai pas connaissance d'un paramètre équivalent pour les drivers des autres versions de DB2.

Dans l'exemple de script que je vous ai proposé au chapitre précédent, la variable \$trace était fixée arbitrairement à zéro. Les valeurs possibles pour ce paramètre sont les suivantes :

- 0 : pas de tracing
- 1 : active le driver de tracing interne de DB2
- 2 : active le moniteur de base de données
- 4 : active le débogueur (STRDBG)
- 16 : active le tracing du job

La valeur du paramètre TRACE peut être le cumul de plusieurs des valeurs ci-dessus. Par exemple, si j'utilise la valeur 18, cela signifie que j'active à la fois le moniteur de base de données (2) et le tracing de job (16).

Le fait d'indiquer une valeur différente de zéro dans le paramètre TRACE a pour effet de générer dans la bibliothèque QUSRSYS un fichier DB2 dont le nom est « QODB » suivi du numéro de travail. Le contenu de ce fichier peut être analysé via SQL.

Exemple de requête :

```
SELECT * FROM QUSRSYS/QODB385372 WHERE QQRID = 1000
```

Livre blanc sur l'utilisation de PHP en environnement IBM i


La colonne QQRID peut prendre l'une des valeurs suivantes :

- 1000 Record: SQL statement summary
- 3000 Record: Arrival sequence
- 3001 Record: Using existing index
- 3002 Record: Temporary index created
- 3003 Record: Query sort
- 3004 Record: Temporary file
- 3006 Record: Access plan rebuild
- 3007 Record: Summary optimization data
- 3010 Record: Host variable and ODP implementation
- 3021 Record: Bitmap created
- 3022 Record: Bitmap merge
- 3023 Record: Temporal hash table created
- 3026 Record: Union merge
- 3027 Record: Subquery merge
- 3028 Record: Grouping
- 3029 Record: Index ordering
- 3005 Record: Table locked
- 3008 Record: Subquery processing
- 3014 Record: Generic query information
- 3018 Record: STRDBMON and ENDDDBMON data
- 3019 Record: Retrieved detail (only with *DETAIL)
- 3025 Record: DISTINCT processing
- 3030 Record: Query step processing
- 5002 Record: SQL request executed by SQL Query Engine (SQE)

Vous pouvez également utiliser l'analyseur de performances SQL intégré au logiciel ACS d'IBM. Pour de plus amples précisions sur les problématiques de performances de DB2 for i, je vous invite à télécharger et à étudier l'excellent redbook (gratuit) d'IBM dont la référence est SG246654 (cf. lien au chapitre 16).

7 Premier programme de reprise de données en PHP

La base de données MySQL intègre en standard plusieurs moteurs de bases de données. Les plus couramment utilisés sont MyISAM (moteur par défaut) et InnoDB. La liste des moteurs disponibles évolue en fonction de la version de MySQL que vous utilisez. On peut afficher la liste des moteurs disponibles via PHPMyAdmin. Avec la version 5.1.24 de MySQL, j'obtiens la liste suivante :

 Moteurs de stockage	
Moteur de stockage	Description
MEMORY	Hash based, stored in memory, useful for temporary tables
InnoDB	Supports transactions, row-level locking, and foreign keys
MyISAM	Default engine as of MySQL 3.23 with great performance
BLACKHOLE	/dev/null storage engine (anything you write to it disappears)
MRG_MYISAM	Collection of identical MyISAM tables
CSV	CSV storage engine
ARCHIVE	Archive storage engine

On constate qu'on dispose notamment d'un moteur MEMORY qui est en quelque sorte l'équivalent de QTEMP pour l'AS/400. A partir de la V5.1 de MySQL, on dispose également d'un moteur de base de données CSV. Eh oui, vous avez bien lu, c'est bien le fameux format CSV qui est couramment utilisé notamment pour les échanges de données inter-entreprises (catalogues électroniques par exemple). La présence de ce moteur CSV vous permet d'importer, et surtout de gérer dans MySQL, les fichiers CSV comme s'il s'agissait de tables MySQL.

A partir de là, si la table des codes pays nous avait été fournie en format CSV, on aurait pu l'intégrer dans MySQL directement dans son format d'origine. Il nous resterait à écrire un petit programme de reprise en PHP permettant de lire ce fichier CSV et de recopier son contenu dans la table LSTPAYS de l'AS/400. C'est d'ailleurs ce que je vous propose de faire avec le script ci-dessous :

```
<?php
// définition de la connexion MySQL
$user      = 'root'; // profil « root » utilisé en développement
$password  = '';     // password par défaut du profil « root » (à adapter)
$database  = 'basetest'; // nom de la base MySQL
$system    = 'localhost'; // localisation de Wampserver
$dsn = "mysql:dbname=$database;host=$system";//creation du dsn pour mysql";

try {
    $conn_mysql = new PDO($dsn, $user, $password);
    $conn_mysql->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage() . '<p/>';
}

// définition de la connexion DB2
$user      = 'xxxxxxxx'; // paramètre à personnaliser
$password  = 'yyyyyyyyyy'; // paramètre à personnaliser
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
$database = 'zzzzzzzzzz' ; // nom de la bibliothèque IBM i à personnaliser
$system   = '999.999.999.999' ; // adresse IP de l'IBM i à personnaliser
$trace    = 0 ; // paramètre de trace désactivé par défaut
$dsn = "odbc:DRIVER={IBM i Access ODBC Driver};
        SYSTEM=$system;DBQ=$database;TRACE=$trace";

try {
    $conn_db2 = new PDO($dsn, $user, $password);
    $conn_db2->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage() . '<p/>';
}

// requête de balayage de la table MySQL LSTPAYSCSV
$req_mysql = 'SELECT CODFRA, CODISO, LIBELLE FROM LSTPAYSCSV ' ;

// requête d'insertion dans la table DB2 de l'AS/400
$req_db2 = 'INSERT INTO LSTPAYS (CODFRA, CODISO, LIBELLE)
           VALUES (?, ?, ?)';

try {
    // exécution de la requête de lecture de la table CSV
    $st1 = $conn_mysql->query($req_mysql) ;
    // préparation de la requête d'insertion sur la table DB2
    $st2 = $conn_db2->prepare($req_db2) ;
    while($row_data1 = $st1->fetch(PDO::FETCH_LAZY)) {
        // exécution de la requête d'insertion sur la table DB2
        // avec les valeurs renvoyées par la requête MySQL
        $st2->execute(array($row_data1->CODFRA,
                           $row_data1->CODISO,
                           $row_data1->LIBELLE
                           )) ;
    }
} catch (PDOException $e) {
    echo 'Error : ' . $e->getMessage() . '<br/>';
    echo 'Code : ' . $e->getCode() . '<br/>';
    echo 'File : ' . $e->getFile() . '<br/>';
    echo 'Line : ' . $e->getLine() . '<br/>';
    echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
}

?>
```

Dans le programme que nous venons d'écrire, nous ouvrons en début de script 2 connexions à 2 bases de données en parallèle (une sur MySQL, l'autre sur DB2 for i). Un peu plus loin, la requête d'insertion sur la table DB2 LSTPAYS est préparée une fois, en amont de la boucle de balayage du curseur MySQL qui traite la table LSTPAYSCSV. Ce temps de préparation est nécessaire à DB2 pour rechercher le meilleur chemin d'accès possible. On prépare donc la requête une seule fois, en amont de la boucle, et on l'exécute à l'intérieur de la boucle, autant de fois qu'il y a de pays dans la table LSTPAYSCSV.

La mise au point de ce programme de reprise nous donne une bonne occasion d'aborder les problématiques de performance et de nous interroger sur la manière dont nous pouvons mesurer le temps de traitement d'un script tel que celui-ci. J'ai trouvé une réponse intéressante à ce problème dans un livre de poche qui s'intitule « PHP avancé » (auteur : Arnaud Gadal, éditions Micro Application). Vous aurez sans doute du mal à vous procurer cet excellent livre qui ne figure plus au catalogue de l'éditeur. Pour mesurer le temps d'exécution d'un script, l'auteur nous proposait, entre autres, le petit script suivant :

```
<?php
function getmicrotime() {
    list($usec, $sec) = explode(" ", microtime());
    return ((float)$usec + (float)$sec);
}
?>
```

Vous pouvez utiliser ce script pour mesurer le temps de traitement du programme de reprise présenté dans ce chapitre. Pour cela, je vous propose de stocker ce script dans un fichier PHP que vous appellerez « fonctions.php ». Il ne vous reste plus qu'à modifier votre programme de reprise de la façon suivante (les modifications sont indiquées en gras) :

```
<?php
// inclusion de la fonction getmicrotime()
require_once("fonctions.php") ;

// définition de la connexion MySQL
$user      = 'root'; // profil « root » utilisé en développement
$password  = '';     // password par défaut du profil « root »
$database  = 'basetest' ; // nom de la base MySQL
$system    = 'localhost' ; // localisation de Wampserver
$dsn       = "mysql:dbname=$database;host=$system";//creation du dsn pour mysql";

try {
    $conn_mysql = new PDO($dsn, $user, $password);
    $conn_mysql->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage() . '<p/>';
}

// définition de la connexion DB2
$user      = 'xxxxxxxxx'; // paramètre à personnaliser
$password  = 'yyyyyyyyyy'; // paramètre à personnaliser
$database  = 'zzzzzzzzzz' ; // nom de la bibliothèque IBM i à personnaliser
$system    = '999.999.999.999' ; // adresse IP de l'IBM i
$trace     = 0; // paramètre de trace désactivé par défaut
$dsn       = "odbc:DRIVER={IBM i Access ODBC Driver};
              SYSTEM=$system;DBQ=$database;TRACE=$trace";

try {
    $conn_db2 = new PDO($dsn, $user, $password);
    $conn_db2->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage() . '<p/>';
}

// requête de balayage de la table MySQL
$req_mysql = 'SELECT CODFRA, CODISO, LIBELLE FROM LSTPAYSCSV ' ;

// requête d'insertion dans la table DB2 de l'AS/400
$req_db2   = 'INSERT INTO MABIB.LSTPAYS (CODFRA, CODISO, LIBELLE) VALUES (?,
?, ?)';

// premier appel de la fonction getmicrotime()
$time_start = getmicrotime() ;
// compteur du nombre de lignes reprises
$wi = 0 ;

try {
    $st1 = $conn_mysql->query($req_mysql) ;
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
// préparation de la requête d'insertion
$stmt2 = $conn_db2->prepare($req_db2) ;
while($row_data1 = $stmt1->fetch(PDO::FETCH_LAZY)) {
    $wi += 1 ;
    // exécution de la requête d'insertion
    $stmt2->execute(array($row_data1->CODFRA,
                        $row_data1->CODISO,
                        $row_data1->LIBELLE
                    )) ;
}
} catch (PDOException $e) {
    echo 'Error : ' . $e->getMessage() . '<br/>';
    echo 'Code : ' . $e->getCode() . '<br/>';
    echo 'File : ' . $e->getFile() . '<br/>';
    echo 'Line : ' . $e->getLine() . '<br/>';
    echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
}

// second et dernier appel de la fonction getmicrotime()
$time_stop = getmicrotime() ;
$time_dif = $time_stop - $time_start ;
$time_dif_unit = $time_dif / $wi ;
echo "<p><h2>Temps d'exécution de la reprise :</h2></p>";
echo "{$time_dif} secondes pour {$wi} enregistrement(s)</p>" ;
echo "Temps de traitement moyen pour 1 enregistrement : {$time_dif_unit}
secondes</p>" ;

?>
```

Résultat obtenu dans le navigateur après exécution du script ci-dessus :

Temps d'exécution de la reprise :

1.1584198474884 secondes pour 246 enregistrement(s)

Temps de traitement moyen pour 1 enregistrement : 0.0047090237702781 secondes

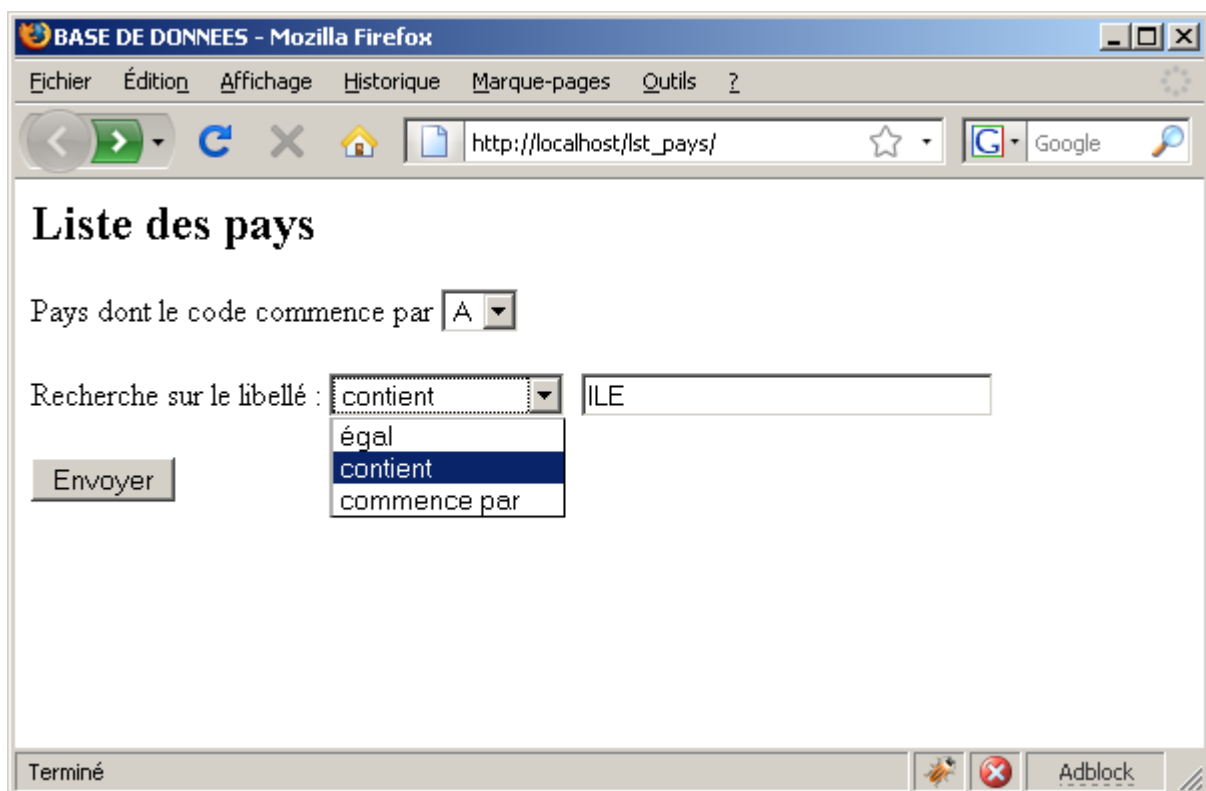
Bien évidemment, ces temps de traitement dépendent pour l'essentiel de la puissance de traitement du serveur IBM i, et dans une moindre mesure, de la puissance du processeur et la mémoire disponible sur le poste sur lequel s'exécute PHP, ainsi que de la bande passante du réseau reliant le serveur IBM i au poste sur lequel s'exécute PHP.

8 Pagination, ou comment simuler un sous-fichier en PHP

Les développeurs RPG sont souvent amenés à développer des programmes interactifs de type 5250 avec listes. En règle générale ils utilisent pour gérer les listes la technique des sous-fichiers.

Avec PHP, on peut obtenir dans une interface web un comportement similaire à un sous-fichier RPG. Je vous propose dans ce chapitre une mini-application, basée sur notre table des pays, qui nous permettra d'afficher la liste des pays selon différents critères. On pourra :

- Soit afficher la liste des pays dont le code commence par une lettre (A à Z)
- Soit afficher la liste des pays dont le nom est égal à une chaîne, ou contient une chaîne, ou commence par une chaîne donnée.



Le code source de ce premier écran est relativement simple. Le nom de ce fichier source est index.php :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>LISTE DES PAYS</title>
</head>
<body>
  <?php
    print '<h2> Liste des pays </h2>';

    echo '<form action="listepays.php" method="get">';

    $alphabet = array( "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K",
                      "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V",
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
        "X", "Y", "Z") ;

    echo '<label>Pays dont le code commence par </label>'
    <select name=code_alpha id=code_alpha>;

    for($index = 0; $index < count($alphabet); $index++) {
        $char = $alphabet[$index] ;
        echo "<option value=\"\$char\">$char</option>";
    }

    echo "</select><br/><br/>";
    echo 'Recherche sur le libellé :
        <select name="choix_recherche">
        <option value=1>écart</option>
        <option value=2>contient</option>
        <option value=3>commence par</option>
        </select>&nbsp;
        <input type="text" name="nom_recherche" value="" size="30" /><br/><br/>'
;

    echo '<input type="submit"> ' ;
    echo '</form>' ;

    ?>
</body>
</html>
```

Si je demande à afficher tous les pays dont le nom contient « ILE », j'obtiens ceci :



The screenshot shows a Mozilla Firefox browser window with the title "LISTE DES PAYS - Mozilla Firefox". The address bar shows the URL "http://localhost/1st_pays/listepays.php?code_alpha=A&choix_recherche=2&nom_recherche=ILE". The page content displays a table titled "Liste des pays dont le libellé contient : ILE". The table has three columns: "CODE FRANCE", "CODE ISO", and "DESCRIPTION". It lists 20 countries, all of which have "ILE" in their names. At the bottom of the page, there are navigation links: "<< Préc. | 1-15 | 16-20 | Suiv. >>" and "(Affichage 1 - 15 sur 20)". The browser's status bar at the bottom shows "Terminé" and an "Adblock" extension icon.

CODE FRANCE	CODE ISO	DESCRIPTION
ALA	AX	ALAND, ILES
BVT	BV	BOUVET, ILE
CYM	KY	CAIMANES, ILES
CXR	CX	CHRISTMAS, ILE
CCK	CC	COCOS (KEELING), ILES
COK	CK	COOK, ILES
FLK	FK	FALKLAND, ILES (MALVINAS)
FRO	FO	FEROE, ILES
SGS	GS	GEORGIE DU SUD ET LES ILES SANDWICH DU SUD
HMD	HM	HEARD, ILE ET MCDONALD, ILES
IMN	IM	ILE DE MAN
UMI	UM	ILES MINEURES ELOIGNEES DES ETATS-UNIS
VGB	VG	ILES VIERGES BRITANNIQUES
VIR	VI	ILES VIERGES DES ETATS-UNIS
MNP	MP	MARIANNES DU NORD, ILES

Livre blanc sur l'utilisation de PHP en environnement IBM i

Le principe de pagination utilisé ici consiste à afficher les pays par série de 15, chaque série s'affichant dans une page distincte en cliquant sur la fourchette indiquée en bas de page (par exemple : 1 à 15 ou 16 à 20).

L'algorithme de pagination est composé de 2 parties étroitement imbriquées qui sont :

- une requête SQL basée sur un curseur scrollable, sur laquelle je reviendrai un peu plus loin
- l'algorithme de pagination proprement dit, que j'ai trouvé dans un excellent livre, le « PHP Cookbook », dont je vous donne les références au chapitre 16. J'ai apporté de très légères modifications à l'algorithme initial, que j'ai indiquées dans les commentaires de chaque script.

Le code source se présente ainsi :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>LISTE DES PAYS</title>
  </head>
  <body>
    <?php
      // configuration de la base de données
      require_once 'config.php' ;
      // chargement des fonctions de pagination
      require_once 'pagination.php' ;

      // récupération des paramètres du $_GET
      if (isset($_GET)) {
        $params = $_GET ;
      } else {
        $params = array() ;
      } ;

      // récupération ou initialisation de l'offset
      $offset = isset($_GET['offset']) ? intval($_GET['offset']) : 1;
      if (!isset($offset)) { $offset = 1; }

      // récupération du critère de recherche par lettre
      if (isset($params['code_alpha'])) {
        $car_de_recherche = trim($params['code_alpha']) ;
      } else {
        $car_de_recherche = '*';
      } ;
      if ($car_de_recherche != '*') {
        $lib_titre = " dont le code commence par " . $car_de_recherche ;
      }

      // récupération du critère de recherche par libellé
      unset ($type_recherche) ;
      if (isset($params['choix_recherche']) && isset($params['nom_recherche'])) {
        $nom_recherche = trim($params['nom_recherche']) ;
        if ($nom_recherche != '') {
          $choix_recherche = trim($params['choix_recherche']) ;
          switch ( $choix_recherche ) {
            case '1': {
              // recherche de type "égal"
              $type_recherche = " = ";
              $lib_titre = " dont le libellé est " .
                "&acute;gal &agrave; : " . $nom_recherche ;
              break;
            }
            case '2': {
              // recherche de type "contient"
              $type_recherche = " LIKE ";
              $lib_titre = " dont le libellé contient : "
```

```
        . $nom_recherche;
        $nom_recherche = '%' . trim($nom_recherche) . '%';
        break;
    }
    case '3': {
        // recherche de type "commence par"
        $type_recherche = " LIKE ";
        $lib_titre = " dont le libellé; commence par : "
            . $nom_recherche ;
        $nom_recherche = trim($nom_recherche) . '%';
        break;
    }
    default:{
        $msg_erreur = "Critères de sélection "
            . " non définis";
        error_log($msg_erreur) ;
        die($msg_erreur);
    }
}
}

// affichage du titre
print "<h2>Liste des pays {$lib_titre}</h2>" ;

// dessiner le début du tableau
echo <<<BOTABLE
    <table border="1" width="100%" cellspacing="0" cellpadding="5">
    <thead>
    <tr>
        <td>CODE FRANCE</td>
        <td>CODE ISO</td>
        <td>DESCRIPTION</td>
    </tr>
    </thead>
    <tbody>
BOTABLE;

// nombre de lignes par page
$nbl_par_page = 15 ;

// préparation des requêtes 1 et 2
// la requête 1 sert au comptage du nombre d'occurrences
$requete1 = "SELECT COUNT(*) AS TOTAL FROM LSTPAYS " ;
if (!isset($type_recherche)) {
    $car_de_recherche2 = $car_de_recherche . '%';
    // recherche sur le code pays
    $requete1 = trim($requete1) . " WHERE CODFRA LIKE ? " ;
    $val_de_recherche = $car_de_recherche2 ;
} else {
    // recherche sur le libellé des pays
    $requete1 = trim($requete1) . " WHERE LIBELLE ".$type_recherche." ? " ;
    $val_de_recherche = $nom_recherche ;
} ;

// la requête 2 sert au chargement de la liste des pays
$requete2 = "SELECT CODFRA, CODISO, LIBELLE FROM LSTPAYS " ;
if (!isset($type_recherche)) {
    // recherche sur le nom des entités logiques
    $requete2 = trim($requete2) . " WHERE CODFRA LIKE ? " ;
} else {
    // recherche sur le libellé des entités logiques
    $requete2 = trim($requete2) . " WHERE LIBELLE ".$type_recherche." ? " ;
} ;

// comptage du nombre de lignes total pour le critère de recherche
// considéré (en vue de calculer le nombre de pages total)
try {
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
$requete = $requete1 ;
$stmt = $db->prepare($requete);
$stmt->execute(array($val_de_recherche));
$data = $stmt->fetch(PDO::FETCH_LAZY);
$total = $data->TOTAL;
} catch (PDOException $e) {
    echo 'Database Problem: ' . $e->getMessage() . '<br/>';
    echo 'Code : ' . $e->getCode() . '<br/>';
    echo 'File : ' . $e->getFile() . '<br/>';
    echo 'Line : ' . $e->getLine() . '<br/>';
    echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
}

// Boucle de chargement du tableau utilisant la technique du Curseur
// Scrollable de DB2 (fonctionne aussi bien sur DB2 pour IBM i
// que sur DB2 pour LUW).
try {
    $requete = $requete2 ;
    $stmt = $db->prepare($requete,
        array(PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL));
    $stmt->execute(array($val_de_recherche));
    for ($stofetch = $nbl_par_page,
        $row_data = $stmt->fetch(PDO::FETCH_LAZY,
            PDO::FETCH_ORI_REL, $offset);
        $row_data !== false && $stofetch-- > 0;
        $row_data = $stmt->fetch(PDO::FETCH_LAZY)) {

        $lastRowNumber++;
        echo <<<LOTABLE
            <tr>
                <td>{$row_data->CODFRA}</td>
                <td>{$row_data->CODISO}</td>
                <td>{$row_data->LIBELLE}</td>
            </tr>
LOTABLE;
    }
} catch (PDOException $e) {
    echo 'Database Problem: ' . $e->getMessage() . '<br/>';
    echo 'Code : ' . $e->getCode() . '<br/>';
    echo 'File : ' . $e->getFile() . '<br/>';
    echo 'Line : ' . $e->getLine() . '<br/>';
    echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
}

// dessiner la fin du tableau
echo <<<EOTABLE
    </tbody>
</table>
<br/>
EOTABLE;

// récupération de la page courante et appel de la fonction de pagination
$page_en_cours = $_SERVER['PHP_SELF'] ;
pc_indexed_links($total, $offset, $nbl_par_page, $page_en_cours, $params);
print "<br/>";
print "(Affichage $offset - $lastRowNumber sur $total)";
?>

</body>
</html>
```

Le code source du script config.php est le suivant :

```
<?php

// définition BD et identifiants de connexion sur DB2 pour IBM i
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
$usr    = 'xxxxxxx'; // profil utilisateur à personnaliser
$pwd    = 'yyyyyyy'; // mot de passe à personnaliser
$dtb    = 'mabase ' ; // bibliothèque contenant la tables des pays à personnaliser
$system = '999.999.999.99' ; // adresse IP du serveur IBM i
$trace  = 0 ; // mode trace désactivé par défaut
$dsn    = "odbc:DRIVER={IBM i Access ODBC Driver};
          SYSTEM=$system;DBQ=$dtb;TRACE=$trace";

// connexion à la base de destination
try {
    $db = new PDO($dsn, $usr, $pwd);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo 'Database Problem: ' . $e->getMessage() . '<br/>';
    echo 'Code   : ' . $e->getCode() . '<br/>';
    echo 'File   : ' . $e->getFile() . '<br/>';
    echo 'Line   : ' . $e->getLine() . '<br/>';
    echo 'Trace  : ' . $e->getTraceAsString() . '<br/>';
} ;

?>
```

Le code source du script pagination.php est le suivant :

```
<?php
// Fonctions de pagination extraites du livre suivant :
//   PHP Cookbook, 2nd Edition,
//   par Adam Trachtenberg et David Sklar, O'Reilly (2006)
// Légère modif de l'algo initial pour passer un tableau $params permettant
// de transmettre les critères de sélection saisis sur le premier formulaire

function pc_print_link($inactive,$text,$offset='1',$current_page, $params_page) {
    // on prépare l'URL avec tous les paramètres sauf "offset"
    $url = '' ;
    $params_page['offset'] = $offset;
    $url = '?' . http_build_query($params_page) ;
    if ($inactive) {
        print "<span class='inactive'>$text</span>";
    } else {
        print "<span class='active'>".
            "<a href='" . htmlentities($current_page) .
            "$url'>$text</a></span>";
    }
}

function pc_indexed_links($total,$offset,$per_page,$curpage,$parmpage) {
    $separator = ' | ';

    // print "<<Prev" link
    pc_print_link($offset == 1, '<< Pr&eacute;c.', $offset - $per_page,
        $curpage, $parmpage);

    // ajout compteur pour ajout d'un <BR> tous les 10 groupes
    $compteur = 0 ;

    // print all groupings except last one
    for ($start = 1, $end = $per_page;
        $end < $total;
        $start += $per_page, $end += $per_page) {
        $compteur += 1 ;
        if ($compteur == 15) {
            $compteur = 0 ;
            print "<br>" ;
        } else {
            print $separator;
        } ;
        pc_print_link($offset == $start, "$start-$end", $start,
            $curpage, $parmpage);
    }
}
```

```
}

/* print the last grouping -
 * at this point, $start points to the element at the beginning
 * of the last grouping
 */

/* the text should only contain a range if there's more than
 * one element on the last page. For example, the last grouping
 * of 11 elements with 5 per page should just say "11", not "11-11"
 */
$end = ($total > $start) ? "-$total" : '';

print $separator;
pc_print_link($offset == $start, "$start$end", $start, $curpage, $parmpage);

// print "Next>>" link
print $separator;
pc_print_link($offset == $start, 'Suiv. >>', $offset + $per_page,
    $curpage, $parmpage);
}
?>
```

En fait la technique du curseur scrollable n'allait pas de soi au départ. L'exemple d'algorithme de pagination fourni dans le livre « PHP Cookbook » s'appuyait sur la base de données SQLite avec laquelle on peut écrire ceci :

```
SELECT * FROM LSTPAYS ORDER BY CODFRA LIMIT 20 OFFSET 10; -- lit 20 lignes à
partir de la ligne 10
```

Avec MySQL, on peut écrire la même chose avec une syntaxe légèrement différente :

```
SELECT * FROM LSTPAYS ORDER BY CODFRA LIMIT 10, 20; -- lit 20 lignes à partir
de la ligne 10
```

Jusqu'à l'arrivée de la V7R1, DB2 for i n'offrait pas d'équivalent des clauses LIMIT et OFFSET. Il a fallu attendre l'arrivée de la V7R1 TR11, et de la V7R2 TR3, pour enfin bénéficier de ce mécanisme. Ce petit tableau extrait d'une documentation IBM vous donne le principe :

Syntax	Alternative Syntax	Action
LIMIT x	FETCH FIRST x ROWS ONLY	Return the first x rows
LIMIT x OFFSET y	OFFSET y ROWS FETCH FIRST x ROWS ONLY	Skip the first y rows and return the next x rows
LIMIT y,x	OFFSET y ROWS FETCH FIRST x ROWS ONLY	Skip the first y rows and return the next x rows

Il devient donc plus facile de porter du code SQL provenant notamment de MySQL ou de PostgreSQL par exemple.

Attention, il y a une petite restriction : la clause OFFSET n'est autorisée que dans le cadre d'une requête de type Full-Select externe appliqué à un DECLARE CURSOR, ou d'un « prepared statement » sur une requête de type SELECT. Vous n'avez pas compris ? Ne vous inquiétez pas, à vrai moi non plus. De toute façon, on s'en moque, utilisez la 3ème solution (LIMIT x, y), et c'est marre.

Livre blanc sur l'utilisation de PHP en environnement IBM i

Il faut quand même noter que, avant l'arrivée de la clause LIMIT (en V7), on pouvait quand même gérer une pagination en SQL sans passer par un curseur scrollable. C'était juste un peu plus compliqué à écrire. Dans notre contexte de pagination sur la liste des pays, on devait écrire la requête suivante :

```
SELECT * FROM (
    SELECT CODFRA, CODISO, LIBELLE,
           ROW_NUMBER() OVER (ORDER BY CODFRA ASC) AS RN
    FROM LSTPAYS
    WHERE CODFRA LIKE ?
) AS FOO WHERE RN BETWEEN ? AND ?
```

Attention : si la technique SQL ci-dessus fonctionne bien sur des tables SQL de taille raisonnable (de l'ordre de quelques dizaines milliers de lignes), j'ai par le passé – vers 2010 ou 2011 - rencontré des problèmes de latence en appliquant cette technique sur des tables de tailles supérieures. A contrario, le curseur scrollable répondait sans latence, quelle que soit la taille du fichier testé. Je n'ai pas eu l'occasion de refaire de tests récemment pour vérifier si ce problème est encore d'actualité. Et peut être que la nouvelle clause LIMIT est suffisamment optimisée pour ne pas présenter les mêmes faiblesses que la technique basée sur la clause OVER (présentée dans la dernière requête ci-dessus).

9 Considérations sur la portabilité du code SQL

9.1 Sensibilité à la casse

On l'a vu précédemment, le paramètre PDO::FETCH_ASSOC est très pratique pour récupérer dans un FETCH, sous la forme d'un tableau, les différentes colonnes renvoyées par une requête SQL. Mais il convient d'exploiter le résultat renvoyé par le FETCH avec prudence.

Exemple : soit une requête déclarant 2 fois la même colonne dans le WHERE, une fois en majuscule, une fois en minuscule :

```
$requete = "SELECT libelle, LIBELLE FROM lstpays WHERE codfra = ?" ;
$st = $db->prepare($requete);
$st->execute(array('FR'));
$data = $st->fetch(PDO::FETCH_ASSOC);
echo ' contenu de $data récupéré :' ;
print_r ($data) ; // affichage du contenu du tableau $data
```

- résultat obtenu avec MySQL :

```
Array ( [libelle] => FRANCE [LIBELLE] => FRANCE )
```

- résultat obtenu avec DB2 :

```
Array ( [LIBELLE] => FRANCE )
```

On constate que :

- Le moteur de MySQL étant sensible à la casse, il renvoie à PHP 2 postes de tableau de nom identique, l'un en majuscule, l'autre en minuscule (conformément à ce qui est demandé dans la requête SQL),
- Le moteur de DB2 n'étant pas sensible à la casse, il renvoie à PHP un seul poste de tableau dont le nom est codé en majuscule.

Maintenant, que se passe-t-il si la requête utilise un SELECT * au lieu d'un SELECT déclarant explicitement les colonnes ? Eh bien, là encore, DB2 renverra un jeu de données avec des noms de colonnes en majuscules, alors que les noms des colonnes renvoyées par MySQL seront par défaut en minuscules. La solution à ce problème se trouve dans la doc officielle de PDO (cf. lien au chapitre 16).

La partie de la documentation qui nous intéresse est celle-ci :

- PDO::ATTR_CASE : Force les noms de colonnes en une casse spécifique.
 - PDO::CASE_LOWER : Force les noms de colonnes en minuscule.
 - PDO::CASE_NATURAL : Laisse les noms des colonnes en la casse définie par le driver de la base de données.
 - PDO::CASE_UPPER : Force les noms de colonnes en majuscule.

Donc, en définissant une connexion PDO avec la syntaxe suivante :

```
$db = new PDO($dsn, $usr, $mdp, array(PDO::ATTR_CASE => PDO::CASE_LOWER));
```

... la requête MySQL de la page précédente ne renvoie plus qu'un seul poste qui est le suivant:

```
Array ( [libelle] => FRANCE )
```

Le jeu de données récupéré par la méthode PDO::FETCH_ASSOC sera donc toujours en minuscule, ce qui **pourrait** garantir une bonne portabilité du code, dans le cas où l'on souhaiterait développer une application portable vers plusieurs SGBD. En fait, il n'en est rien, car **il s'avère que DB2 n'accepte pas le paramètre PDO::CASE_LOWER**. En effet, DB2 n'est capable de renvoyer que des noms de colonnes en majuscules (le paramètre PDO::CASE_LOWER entraînant un échec de la connexion). Donc si on souhaite garantir une bonne portabilité du code, il conviendra de privilégier le mot clé PDO::CASE_UPPER, qui est accepté par la plupart des SGBD (dont MySQL et DB2). Il faudra dès lors travailler, au sein du code PHP, avec des noms de postes de tableaux en majuscule pour garantir une bonne portabilité du code.

9.2 Particularités syntaxiques de MySQL

Avertissement : certaines des considérations ci-dessous étaient valables en 2009, elles ne le sont peut être plus autant en 2022. Mais si vous êtes amené à travailler sur la maintenance et le portage vers DB2 de vieilles applications PHP, ces points feront peut être sens pour vous.

Quelques remarques complémentaires sur les problèmes de portage de requêtes MySQL vers DB2 :

- Certains développeurs MySQL ont pour habitude d'encadrer les noms de tables et de colonnes entre apostrophes inverses (`). Cela ne sert à rien, mais surtout cela nuit à la portabilité du code SQL vers DB2, car l'apostrophe inverse est un caractère non toléré par DB2. Il est donc nécessaire de « nettoyer » les requêtes SQL de ce caractère quand vous le rencontrez.

- Certains développeurs MySQL, quand ils créent des requêtes par concaténation, ont l'habitude d'encadrer toutes les valeurs utilisées dans les clauses WHERE avec des apostrophes, même lorsqu'il s'agit de valeurs numériques. Cela s'explique par le fait que MySQL est beaucoup plus tolérant que DB2 quant au typage des colonnes comparées dans les clauses WHERE. Si vous rencontrez des requêtes écrites de cette façon, vous devrez les « nettoyer » en supprimant les apostrophes encadrant les valeurs numériques, pour pouvoir les utiliser sous DB2.

NB : Le chapitre 16 contient le lien vers un intéressant redbook d'IBM, relatif au portage de bases MySQL vers DB2.

9.3 Simuler la fonction LastInsertID() sous DB2

Un autre point à prendre en considération lors du portage de script PHP de MySQL vers DB2, est l'absence de la fonction LastInsertID() sous DB2. PDO offre une méthode LastInsertID() émulant la fonction MySQL du même nom, mais cette méthode ne fonctionne pas avec DB2.

Exemple d'utilisation de la fonction LastInsertID en MySQL :

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
$sql = "SELECT LAST_INSERT_ID()";
```

Exemple d'utilisation de la méthode LastInsertID sous PDO :

```
echo 'ID de la dernière insertion : ' . $db->lastInsertId();
```

La méthode LastInsertId() renvoie le message suivant si on l'utilise avec DB2 (difficile de faire plus explicite) :

```
ID of last insert: Error : SQLSTATE[IM001]: Driver does not support this
function: driver does not support lastInsertId()
Code : IM001
```

Je vous propose ci-dessous un moyen de contourner le problème sous DB2. Pour cela nous allons repartir de notre table des pays, en lui ajoutant un Identifiant interne, que nous appelons IDAUTO, et qui est incrémenté automatiquement par DB2 à chaque insertion de ligne :

```
CREATE TABLE MABIB/LSTPAYS (
  CODFRA CHAR (3 ) NOT NULL WITH DEFAULT,
  CODISO CHAR (2 ) NOT NULL WITH DEFAULT,
  LIBELLE CHAR (50 ) NOT NULL WITH DEFAULT,
  IDAUTO INTEGER NOT NULL
    GENERATED ALWAYS AS IDENTITY
    ( START WITH 1 , INCREMENT BY 1 , CACHE 1 )
)
```

Vous pouvez maintenant ajouter les lignes suivantes à la fin du programme de reprise que nous avons étudié au chapitre 7, ce qui nous permettra d'afficher dans le navigateur le numéro du dernier ID inséré dans la table LSTPAYS :

```
$sql2 = 'SELECT IDENTITY_VAL_LOCAL() AS DERNIERID FROM SYSIBM.SYSDUMMY1';
$stmt2 = $conn_db2->query($sql2);
$row = $stmt2->fetch(PDO::FETCH_LAZY);
echo 'ID of last insert: ', $row->DERNIERID;
```

A noter : dans la version 1 du présent dossier, j'avais fixé le pas d'incrémentation du cache de la colonne IDAUTO à 10. Si cela peut se justifier pour des raisons de performance, dans le cadre d'un programme RPG par exemple, ce n'est pas une très bonne idée dans le contexte d'exécution d'un script PHP, et il vaut mieux fixer cette valeur à 1 dans ce cas. Je vous invite à consulter la documentation de DB2 pour de plus amples informations sur ce sujet.

9.4 Gérer les attributs d'exécution de PDO

On l'a vu avec le problème de la sensibilité à la casse (cf. chapitre 9.1), il est parfois nécessaire de gérer les attributs d'exécution de PDO. On se sert également de ces attributs pour activer l'interception des erreurs bases de données par la gestion des exceptions PHP (try/catch) lors de l'ouverture d'une connexion base de données, comme dans l'exemple ci-dessous :

```
try {
    $conn = new PDO($dsn, $user, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage() . '<p/>';
}
```

On trouve la liste des attributs d'exécution de PDO dans la documentation officielle (cf. lien au chapitre 16).

Il est intéressant de noter que l'on peut également récupérer les attributs d'exécution de la connexion courante au moyen de la méthode PDO `getAttribute()`. La documentation officielle propose un petit script permettant de récupérer la valeur de chaque attribut. Ce script est le suivant :

```
<?php
$conn = new PDO('odbc:sample', 'db2inst1', 'ibmdb2');
$attributes = array(
    "AUTOCOMMIT", "ERRMODE", "CASE", "CLIENT_VERSION", "DRIVER_NAME",
    "CONNECTION_STATUS", "ORACLE_NULLS", "PERSISTENT", "PREFETCH",
    "SERVER_INFO", "SERVER_VERSION", "TIMEOUT"
);

foreach ($attributes as $val) {
    echo "PDO::ATTR_$val: ";
    echo $conn->getAttribute(constant("PDO::ATTR_$val")) . "\n";
}
?>
```

J'ai testé ce script sur DB2 pour LUW et DB2 for i, et je me suis aperçu que plusieurs attributs ne fonctionnaient pas avec ces 2 versions de DB2. Pour que le script ci-dessus fonctionne avec DB2, il faudrait réduire le nombre de postes du tableau `$attributes` aux seules valeurs suivantes :

```
$attributes = array(
    "ERRMODE", "CASE", "CLIENT_VERSION", "DRIVER_NAME", "ORACLE_NULLS",
    "PERSISTENT"
);
```

Ce qui nous donne les valeurs suivantes :

```
DB2 : PDO::ATTR_ERRMODE: 2
DB2 : PDO::ATTR_CASE: 0
DB2 : PDO::ATTR_CLIENT_VERSION: ODBC-Win32
DB2 : PDO::ATTR_DRIVER_NAME: odbc
DB2 : PDO::ATTR_ORACLE_NULLS: 0
DB2 : PDO::ATTR_PERSISTENT:
```

Par comparaison, MySQL est beaucoup plus bavard, puisqu'il nous renvoie la liste d'attributs suivants (qui dépendent bien évidemment de votre version de MySQL) :

```
MySQL : PDO::ATTR_AUTOCOMMIT: 1
MySQL : PDO::ATTR_ERRMODE: 2
MySQL : PDO::ATTR_CASE: 0
MySQL : PDO::ATTR_CLIENT_VERSION: 5.0.51a
MySQL : PDO::ATTR_DRIVER_NAME: mysql
MySQL : PDO::ATTR_CONNECTION_STATUS: localhost via TCP/IP
MySQL : PDO::ATTR_ORACLE_NULLS: 0
MySQL : PDO::ATTR_PERSISTENT:
MySQL : PDO::ATTR_SERVER_INFO: Uptime: 7374 Threads: 1 Questions: 32
    Slow queries: 0 Opens: 14 Flush tables: 1 Open tables: 0
    Queries per second avg: 0.4
MySQL : PDO::ATTR_SERVER_VERSION: 5.1.24-rc-community-log
```

Attention : ce n'est pas parce qu'un attribut PDO est présent, qu'il fonctionne réellement avec votre base de données, nous avons d'ailleurs vu le problème avec l'attribut de gestion de la casse. Je vous recommande donc, si vous souhaitez utiliser certains attributs de PDO, de bien tester leur comportement avec votre base de données, avant de les utiliser dans vos développements.

Notez également que cette liste d'attributs est susceptible d'évoluer en fonction des nouvelles versions de PHP. De plus, vous pouvez rencontrer des attributs spécifiques à une base de données en particulier. Par exemple, je me suis heurté à un problème de bufferisation de requêtes sur MySQL (pour une raison qui m'échappe encore aujourd'hui). Ce problème ne se posait pas tant que je travaillais sur Wampserver, mais dès que j'installais mes scripts sur un serveur de production sous Linux, mes requêtes MySQL ne fonctionnaient plus. La solution à ce problème m'a été apportée par l'attribut `MYSQL_ATTR_USE_BUFFERED_QUERY` que je devais obligatoirement activer sur le serveur de production. J'ai donc aménagé mon script PHP de connexion à la base de données MySQL de la façon suivante (en définissant une constante PHP que j'ai appelée `TYPE_ENVIR` et que j'alimentais en amont en fonction de mes besoins) :

```
if (TYPE_ENVIR == 'prod') {
    $db = new PDO($dsn, $usr, $mdp ,
        array(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY=>true));
} else {
    $db = new PDO($dsn, $usr, $mdp);
}
```

J'aurais pu écrire également ceci (ce qui revient au même) :

```
$db = new PDO($dsn, $usr, $mdp);
if (TYPE_ENVIR == 'prod') {
    $db->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, true);
}
```

9.5 L'échappement des quotes

Les méthodes `prepare()` et `execute()`, que nous avons vues notamment dans le programme de reprise, effectuent un échappement implicite et automatique des quotes. Cet échappement est indispensable pour bénéficier d'une bonne protection contre les attaques par injection SQL. Comme j'utilise les méthodes `prepare()` et `execute()` de manière systématique, je n'ai quasiment jamais besoin d'utiliser la méthode `PDO::quote()`.

Pour information, la méthode `quote()` est en fait l'équivalent pour PDO de la méthode `mysql_real_escape_string()` pour la bibliothèque de fonctions MySQL de PHP.

La documentation officielle de PDO indique ceci :

***PDO::quote()** place des guillemets simples autour d'une chaîne d'entrées (si nécessaire) et échappe les caractères spéciaux présents dans la chaîne d'entrées, en utilisant le style de protection approprié au driver courant.*

*Si vous utilisez cette fonction pour construire des requêtes SQL, vous êtes vivement invité à utiliser **PDO::prepare()** pour préparer les requêtes SQL avec des paramètres liés au lieu d'utiliser **PDO::quote()** pour interpréter les entrées utilisateurs dans la requête SQL. Les requêtes préparées avec des paramètres liés ne sont pas seulement plus portables, plus malléables et plus sécurisées mais bien plus rapides à exécuter que d'interpréter les requêtes, étant donné que les côtés client et serveur peuvent mettre en cache une version compilée de la requête.*

Tous les drivers PDO n'implémentent pas cette méthode (comme PDO_ODBC). Utilisez les requêtes préparées à la place.

Eh oui, vous avez bien lu, PDO_ODBC n'implémente pas cette méthode qui est donc susceptible de nuire à la portabilité de votre code PHP. Donc si j'étais vous...

9.6 PDO et Object Relational Mapping (ORM)

A noter : ce chapitre a été entièrement réécrit pour la v2 du présent document.

PDO fait partie de la famille des outils de type ORM (pour « Object Relational Mapping ») puisqu'il permet de mettre en relation très facilement des données SQL avec des données PHP, dans un sens comme dans l'autre.

On peut qualifier PDO de couche ORM de premier niveau, dans le sens où PDO ne masque pas le code SQL sous-jacent.

Certaines couches ORM peuvent être qualifiées de couches ORM de second niveau, car elles apportent une couche d'abstraction supplémentaire, permettant de masquer le code SQL au moyen d'un méta-langage. Dans cette catégorie, on trouve ActiveRecord, qui est la couche ORM standard du framework de développement Ruby On Rails. Très concrètement, voici comment on pourrait lire la table des pays avec ActiveRecord :

- pour la lecture de la ligne dont la colonne IDAUTO est égale à 1 :

```
pays = Pays.find_by_IDAUTO(1)
puts « Pays #{pays.CODISO}, Libellé = #{pays.LIBELLE} »
```

- pour une boucle de lecture de tous les pays dont le libellé contient « ILE » :

```
Pays.find(:all, :conditions=> « LIBELLE LIKE '%ILE%' ») do |pays|
  puts pays.LIBELLE
end
```

L'un des gros points forts d'ActiveRecord réside dans sa capacité à traduire les requêtes ci-dessus (qui sont en fait écrites en pure syntaxe Ruby) dans la syntaxe propre au SGBD sous-jacent (MySQL, Oracle, DB2, etc...) tel qu'il a été configuré dans le fichier de configuration de Ruby On Rails. Le développeur est ainsi déchargé des problématiques de portabilité du code SQL. Bien évidemment cette portabilité a un prix, car le code SQL généré ne sera sans doute pas aussi optimisé que le code écrit par un développeur SQL (c'est l'éternel problème des générateurs de code...).

Pour en revenir à PHP, il est intéressant de noter que plusieurs projets de couches ORM existent, que vous pouvez considérer dans vos choix de développement. Parmi les projets les plus connus, on trouve :

- le projet Propel : propel.phpdb.org
- le projet Doctrine : www.doctrine-project.org

Je vous invite à consulter la documentation de ces 2 projets, car leurs possibilités respectives sont réellement attractives.

Concrètement, utiliser une couche ORM consiste à générer des classes PHP encapsulant chacune de vos tables SQL. On aura généralement une classe par table, mais il peut y avoir des exceptions. En règle générale, chaque instance d'une classe PHP correspondra à une ligne de table SQL (ou à un regroupement de lignes selon les cas).

Mais peut être vous demandez-vous si l'utilisation d'un outil de type ORM présente un réel intérêt pour vous. Il n'y a pas de réponse toute faite à cette question. Tout dépend du contexte et de l'importance de ce que vous souhaitez développer.

Si vous avez besoin d'écrire quelques programmes de reprise tels que celui que nous avons vu pour l'alimentation de la table des pays, ou quelques extractions de données à des fins statistiques ou d'interfaçage, utiliser un outil de type ORM ne présente pas un grand intérêt.

Si en revanche vous souhaitez développer une application « métier » dont vous savez qu'elle sera amenée à « grossir », vous avez tout intérêt à envisager très tôt l'utilisation d'une couche ORM. Cette couche vous permettra d'obtenir une meilleure séparation entre logique métier, stockage des données, gestion des transactions, et présentation des informations. On parle dans ce cas d'architecture de type MVC (pour « Modèle - Vue - Contrôleur »). Employée à bon escient, une architecture MVC permet de garantir à vos applications un bon niveau de maintenabilité et une grande robustesse, tout au long de leur cycle de vie. Il existe de nombreux ouvrages sur le sujet, et de bons articles sur Wikipedia, je ne m'étendrai donc pas sur le sujet.

Un autre aspect à prendre en compte dans la décision d'utiliser ou pas une couche ORM est liée au fait qu'il est plus facile de faire de l'ORM sur une base de données bien normalisée (ce qui est généralement le cas quand on a la possibilité de partir de zéro), que sur une base de données existante dans laquelle les colonnes ont des noms hétéroclites. En effet, l'un des intérêts des outils de type ORM est de permettre la génération plus ou moins automatisée des classes encapsulant les tables SQL. Mais dans le monde AS/400, il est fréquent de rencontrer des tables dont les clés primaires sont composées de plusieurs colonnes (j'ai même vu des tables dont la clé primaire était composée d'une quinzaine de colonnes... no comment !). Ce genre de table nécessite l'écriture de classes PHP personnalisées, ou d'importantes modifications dans le code des classes générées, car les générateurs automatiques ne peuvent donner de réponse satisfaisante sur de telles structures.

Ce chapitre est très frustrant pour moi, car je suis tenté de vous donner des exemples concrets. Mais si je fais ça, je suis obligé de partir dans l'écriture d'un livre blanc « bis » d'au moins 70 pages, tant le sujet est vaste et passionnant. De plus, si vous n'avez pas de connaissances en matière de POO, vous n'allez rien comprendre à ce que je raconte, à moins que je n'ajoute un cours d'introduction sur le sujet (et là on va certainement dépasser les 100 pages). Donc je préfère en rester là, et vous proposer, à la fin du chapitre 16, une sélection de livres consacrés à PHP, qui sont très complémentaires et contiennent tous de très bons chapitres sur la POO.

9.7 Pistes pour améliorer la gestion des erreurs

A noter : ce chapitre est un nouveau chapitre écrit spécifiquement pour la v2 du présent dossier.

PHP offre plusieurs moyens de gérer les erreurs, particulièrement avec PDO.

La première, que nous allons étudier rapidement, et écarter tout aussi rapidement, consiste à utiliser les méthodes PDO::errorCode() et PDOStatement::errorCode().

La documentation de PHP est très bien faite et indique ceci :

PDO::errorCode() retourne un SQLSTATE, un identifiant alphanumérique de cinq caractères défini dans le standard AINSI SQL. Brièvement, un SQLSTATE consiste en une valeur de classe de deux caractères suivi par une valeur de sous-classe de trois caractères. Une valeur de classe de 01 indique une alerte et est accompagnée par un code de retour SQL_SUCCESS_WITH_INFO. Les valeurs de classes autre que '01', mis à part la classe 'IM', indiquent une erreur. La classe 'IM' est spécifique aux alertes et aux erreurs qui sont issus de l'implémentation elle-même de PDO (ou peut-être ODBC, si vous utilisez le driver ODBC). La valeur de sous-classe '000' dans n'importe quelle classe, indique qu'il n'y a pas de sous-classe pour cet SQLSTATE.

PDO::errorCode() retourne uniquement les codes erreurs pour les opérations exécutées directement sur le gestionnaire de la base de données. Si vous créez un objet PDOStatement avec la fonction PDO::prepare() ou la fonction PDO::query() et que vous invoquez une erreur sur le gestionnaire de requête, **PDO::errorCode()** ne retournera pas cette erreur. Vous devez appeler PDOStatement::errorCode() pour retourner le code erreur pour une opération exécutée sur un gestionnaire de requête particulier.

Concrètement, dans un programme RPG, pour travailler avec SQLSTATE, vous pourriez écrire ceci :

```
* CARTES D
* (seule WSQLCLAS doit être déclarée, SQLSTATE étant implicite)
DWSQLCLAS          S          2

... execution de la requête SQL ...

C          EVAL      WSQLCLAS = %SUBST(SQLSTATE:1:2)
C          IF        WSQLCLAS = '00' OR WSQLCLAS = '01'
C* OK (requête exécutée)
C          ELSEIF    WSQLCLAS = '02'
C* EOF (fin de fichier)
C          ELSE
C* KO (erreur d'exécution)

C          END
```

Mais revenons à nos moutons, je vous disais au début de ce chapitre que je préférais privilégier une autre solution qui est la suivante :

```
try {  
    // requête SQL à exécuter  
} catch (PDOException $e) {  
    echo 'Error : ' . $e->getMessage() . '<br/>';  
    echo 'Code : ' . $e->getCode() . '<br/>';  
    echo 'File : ' . $e->getFile() . '<br/>';  
    echo 'Line : ' . $e->getLine() . '<br/>';  
    echo 'Trace : ' . $e->getTraceAsString() . '<br/>';  
}
```

Vous avez déjà vu cette forme d'écriture dans plusieurs des codes sources proposés dans ce dossier. Elle consiste à déclencher une exception au moyen de la classe PDOException, qui est une classe standard de PHP. La variable \$e est en fait une instance de la classe PDOException. Elle permet de récupérer différentes informations au moyen des méthodes getMessage(), getCode(), getFile(), getLine() et getTraceAsString().

La manière dont j'ai écrit le paragraphe « catch » déclenché par une erreur PDO est un exemple à caractère purement pédagogique. Vous avez tout intérêt à ne pas l'écrire de cette façon sur une application destinée à « tourner » en production. La raison est toute simple : en cas d'anomalie, les informations affichées ici au moyen des différentes méthodes de PDO vont s'afficher dans le navigateur de l'utilisateur. Et ces informations sont trop sensibles pour la sécurité de votre application pour que vous puissiez vous le permettre.

En fait, il est intéressant de pouvoir afficher les erreurs dans votre navigateur, tant que vous êtes en environnement de développement ou de recette, mais il faut pouvoir les cacher aux utilisateurs dès que vous êtes en production. Cela nous amène à considérer la notion d'environnement, notion que l'on retrouve sur de nombreux frameworks de développement (PHP ou non).

Pour pouvoir orienter l'affichage des erreurs vers la bonne sortie, je vous propose la fonction suivante :

```
function MyPDOException ($pdoexc, $reqsql = '') {  
    if (TYPE_ENVIR != "prd") {  
        echo "<table border=\"1\">  
        <tr>  
            <td> Code SQL </td>  
            <td> {$reqsql} </td>  
        </tr>  
        <tr>  
            <td> Msg </td>  
            <td> {$pdoexc->getMessage()} </td>  
        </tr>  
        <tr>  
            <td> Code </td>  
            <td> {$pdoexc->getCode()} </td>  
        </tr>  
        <tr>  
            <td> File </td>  
            <td> {$pdoexc->getFile()} </td>  
        </tr>  
        <tr>  
            <td> Line </td>  
            <td> {$pdoexc->getLine()} </td>  
        </tr>  
        <tr>
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
<td> Trace</td>
<td> {$pdoexc->getTraceAsString()} </td>
</tr>
</table><br />";
}
error_log ("code SQL -> " . $reqsql);
error_log ("getMessage -> " . $pdoexc->getMessage());
error_log ("getCode -> " . $pdoexc->getCode());
error_log ("getFile -> " . $pdoexc->getFile());
error_log ("getLine -> " . $pdoexc->getLine());
error_log ("getTrace -> " . $pdoexc->getTraceAsString());
die ('Application momentan&eacute;ment indisponible') ;
}
```

Le principe est le suivant :

- si je suis dans un environnement autre que « prd » (qui correspond à la production), alors j’affiche les erreurs dans le navigateur sous la forme d’un tableau HTML tel que celui ci-dessous :

Code SQL	SELECT CODFRA, CODISO, LIBELLEx FROM LSTPAYS WHERE CODFRA = 'FRA'
Msg	SQLSTATE[42S22]: Column not found: 1054 Unknown column 'LIBELLEx' in 'field list'
Code	42S22
File	C:\wamp\www\ testpdo.php
Line	7
Trace	#0 C:\wamp\www\ testpdo.php(7): PDOStatement->execute() #1 {main}

- de plus, et ce quel que soit l’environnement d’exécution, j’envoie les mêmes informations d’erreur dans la log d’erreur de PHP, via la fonction PHP standard `error_log()`. Cette log est facilement consultable sous Wampserver (cf. menu « PHP », option « PHP Error Log »).

Pour que cette fonction s’exécute correctement, nous avons besoin de définir en amont une constante `TYPE_ENVIR`, de la façon suivante :

```
/**
 * Définition du type d'environnement BD
 * "dev" = développement
 * "rec" = recette
 * "prd" = production
 */
define('TYPE_ENVIR', 'dev') ;
```

Désormais, au lieu d’écrire ceci dans vos programmes :

```
try {
    // requête SQL à executer
} catch (PDOException $e) {
    echo 'Error : ' . $e->getMessage() . '<br/>';
    echo 'Code : ' . $e->getCode() . '<br/>';
    echo 'File : ' . $e->getFile() . '<br/>';
    echo 'Line : ' . $e->getLine() . '<br/>';
}
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
        echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
    }
}
```

... vous écrirez plutôt ceci :

```
try {
    $sql = 'SELECT * FROM xxx ...' // requête SQL à exécuter
} catch (PDOException $e){
    MyPDOException ($e, $sql) ;
}
```

Pour conclure sur le sujet, voici le contenu de mon script configdb.php, tel que je l'utilise dans mes développements :

```
<?php
/**
 * Définition du type d'environnement BD
 * "dev" = développement
 * "rec" = recette
 * "prd" = production
 */
define('TYPE_ENVIR', 'dev') ;

/**
 * Création d'une instance PDO stockée dans la variable globale $db
 * (si pas déjà définie)
 */
If (!isset($db)) {
    if (TYPE_ENVIR == "prd") {
        /**
         * Paramètres de connexion pour la production
         */
        $mypdo_username = 'xxx';
        $mypdo_password = 'yyy';
        $mypdo_dsnconfig = "odbc:zzz";
        $mypdo_options  = array() ;
    } else {
        if (TYPE_ENVIR == "dev") {
            /**
             * Paramètres de connexion pour le développement
             */
            $mypdo_username = 'xxx';
            $mypdo_password = 'yyy';
            $mypdo_dsnconfig = "odbc:zzz";
            $mypdo_options  = array() ;
        } else {
            if (TYPE_ENVIR == "rec") {
                /**
                 * Paramètres de connexion pour la recette
                 */
                $mypdo_username = 'xxx';
                $mypdo_password = 'yyy';
                $mypdo_dsnconfig = "odbc:zzz";
                $mypdo_options  = array() ;
            } else {
                die ("BD inaccessible");
            }
        }
    }
}
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
// création de la connexion BD via PDO
$db = new PDO($mypdo_dsnconfig, $mypdo_username ,
    $mypdo_password, $mypdo_options) or die("");
// activation du mode de gestion des erreurs avancées de PDO
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try {
    // forçage des sorties de MYSQL en mode UTF8 (pour exemple)
    $sql = "SET NAMES 'utf8'" ;
    $db->query($sql) ;
} catch (PDOException $e){
    MyPDOException ($e, $sql);
}

}

/**
 * fonction utilisée avec la classe PDOException permettant un meilleur
 * contrôle sur la présentation des erreurs renvoyées par cette classe
 */
function MyPDOException ($pdoexc, $reqsql = '')
{
    if (TYPE_ENVIR != "prd") {
        echo "<table border=\"1\">
        <tr>
            <td> Code SQL </td>
            <td> {$reqsql} </td>
        </tr>
        <tr>
            <td> Msg </td>
            <td> {$pdoexc->getMessage()} </td>
        </tr>
        <tr>
            <td> Code </td>
            <td> {$pdoexc->getCode()} </td>
        </tr>
        <tr>
            <td> File </td>
            <td> {$pdoexc->getFile()} </td>
        </tr>
        <tr>
            <td> Line </td>
            <td> {$pdoexc->getLine()} </td>
        </tr>
        <tr>
            <td> Trace</td>
            <td> {$pdoexc->getTraceAsString()} </td>
        </tr>
        </table><br />";
    }
    error_log ("code SQL -> " . $reqsql);
    error_log ("getMessage -> " . $pdoexc->getMessage());
    error_log ("getCode -> " . $pdoexc->getCode());
    error_log ("getFile -> " . $pdoexc->getFile());
    error_log ("getLine -> " . $pdoexc->getLine());
    error_log ("getTrace -> " . $pdoexc->getTraceAsString());
    die ('Application momentanément indisponible') ;
}
?>
```

Vous noterez au début du code ci-dessus la présence du test suivant :

```
if (!isset($db)) {
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

Il est bien pratique puisqu'il me permet de contrôler que je n'ouvre la connexion base de données qu'une seule fois. A noter que \$db est ici une variable de type « ressource » affectée à une instance de la classe PDO.

Vous pouvez, si vous le souhaitez, reprendre et adapter ce code à vos propres besoins.

9.8 La gestion des dates et heures

A noter : ce chapitre est un nouveau chapitre écrit spécifiquement pour la v2 du présent dossier.

Dans les différents articles que j'ai pu consulter concernant l'utilisation de PHP avec DB2 (notamment ceux publiés sur le site Developerworks d'IBM), j'ai constaté que la gestion des dates (et notamment leur mise à jour) était un sujet rarement abordé. Il est vrai que c'est un sujet relativement ambitieux pour plusieurs raisons :

- PHP offre un panel impressionnant de fonctions de gestion de dates (environ une cinquantaine), incluant notamment la gestion des fuseaux horaires. Il convient d'étudier cette liste de fonctions avec attention afin de pouvoir les utiliser à bon escient. Il faut noter également que certaines fonctions ne sont pas implémentées dans la version Windows de PHP.
- Les types de données temporelles diffèrent d'un SGBD à l'autre. Par exemple, DB2 fournit les types de données DATE, TIME et TIMESTAMP. MySQL fournit les types DATE, DATETIME, TIME, TIMESTAMP et YEAR. Si on se focalise sur le type TIMESTAMP, on s'aperçoit que ce format n'est pas identique entre les 2 SGBD.
- Certains développeurs privilégieront l'utilisation des fonctions de manipulation de date du SGBD quand d'autres développeurs privilégieront l'utilisation des fonctions de manipulation de dates de PHP. Les 2 approches me semblent tout à fait valables (elles ont chacune leurs avantages et leurs inconvénients), et je me garderais bien de prendre position pour l'un ou l'autre choix.

Je vous recommande vivement d'étudier attentivement les fonctions de manipulations de dates de PHP, ainsi que celles de votre SGBD, pour déterminer celles que vous choisirez d'utiliser (vous pouvez très bien choisir d'utiliser un « mix » des deux). Si vous êtes amené à travailler avec plusieurs SGBD en parallèle, vous avez tout intérêt à bien maîtriser les formats de date propres à chaque SGBD, pour pouvoir naviguer d'une base à l'autre sans difficulté.

Je vous propose page suivante un petit script PHP qui permet de tester plusieurs manières de mettre à jour des dates dans une table DB2. J'ai créé la table en question avec la requête SQL suivante :

```
CREATE TABLE TESTDATES (  
  DATDAT DATE NOT NULL WITH DEFAULT,  
  TIMTIM TIME NOT NULL WITH DEFAULT,  
  TIMSTAMP TIMESTAMP NOT NULL WITH DEFAULT,  
  DATNUM NUMERIC ( 8, 0) NOT NULL WITH DEFAULT,  
  HEUNUM NUMERIC ( 6, 0) NOT NULL WITH DEFAULT  
)
```

Dans le monde IBM i, on utilise encore fréquemment des dates et heures définies dans un simple format numérique, c'est la raison pour laquelle j'ai créé les colonnes DATNUM et HEUNUM, histoire de vous montrer comment vous pouvez les alimenter avec PHP.

Dans le script PHP suivant, les deux premières requêtes sont une mise en jambe pour rappeler simplement les 2 syntaxes de détermination de l'heure et de la date système. Ces deux premières requêtes ne mettent à jour que les 3 premières colonnes de la table.

Livre blanc sur l'utilisation de PHP en environnement IBM i

La quatrième requête met à jour également DATNUM et HEUNUM avec la date et l'heure système, en utilisant strictement la syntaxe SQL.

La cinquième requête met à jour l'ensemble des colonnes à partir de données calculées par PHP, en respectant les formats de données attendus par SQL DB2. La plupart des colonnes ne présentent pas de difficulté, hormis la colonne Timestamp, dont le format attendu par DB2 est le suivant : YYYY-MM-DD-HH.M.SS.xxxxxx.

Or le Timestamp fourni par PHP en standard ne contient pas les 6 chiffres de droite (représentés par « xxxxxx »), qui sont en fait des microsecondes. Je vous propose de compléter le Timestamp de PHP en y ajoutant les microsecondes calculées par la fonction PHP *microtime()*. Cette dernière renvoie le temps exprimé en secondes et microsecondes (je rappelle que nous avons déjà utilisé cette fonction pour calculer le temps d'exécution d'un script PHP).

Voici le script PHP « brut de fonderie ». Ce n'est pas un modèle du genre, c'est juste un point de départ pour vous aider à tester différentes solutions :

```
<?php
$usr   = 'xxxxxxxxxx'; // profil utilisation AS/400
$pwd   = 'YYYYYYYYYY'; // mot de passe
$dtb   = 'zzzzzzzzzz' ; // base de données
$system = 'xxx.xxx.xxx.xxx' ; adresse IP
$trace = 0 ; // mode trace désactivé par défaut
$dsn = "odbc:DRIVER={IBM i Access ODBC Driver};
        SYSTEM=$system;DBQ=$dtb;TRACE=$trace";

try {
    $db = new PDO($dsn, $usr, $pwd);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
} ;

/* requête 1 pour test d'insertion en "full SQL" */
$requete = "INSERT INTO TESTDATES (DATDAT, TIMTIM, TIMSTAMP)
            VALUES(current date, current time, current timestamp)" ;
$st = $db->prepare($requete);
$st->execute();
$nbinsert = $st->rowCount();
if ($nbinsert != 1) {
    echo "<br/>insertion 1 KO" ;
} else {
    echo "<br/>insertion 1 OK" ;
}

/* requête 2 identique à la précédente (mais avec appel des fonctions
équivalentes aux mots réservés) */
$requete = "INSERT INTO TESTDATES (DATDAT, TIMTIM, TIMSTAMP)
            VALUES(curdate(), curtime(), now())" ;
$st = $db->prepare($requete);
$st->execute();
$nbinsert = $st->rowCount();
if ($nbinsert != 1) {
    echo "<br/>insertion 2 KO" ;
} else {
    echo "<br/>insertion 2 OK" ;
}

/* requête 3 avec mise à jour des colonnes DATNUM et HEUNUM en "full SQL"
*/
```


Livre blanc sur l'utilisation de PHP en environnement IBM i

```
$requete = <<<_SQL
INSERT INTO TESTDATES (DATDAT, TIMTIM, TIMSTAMP, DATNUM, HEUNUM)
VALUES(curdate(), curtime(), now(),
       integer(replace(char(curdate()),ISO), '-', '')),
       integer(replace(char(curtime()), ISO), '.', '')))
_SQL;
$st = $db->prepare($requete);
$st->execute();
$nbinsert = $st->rowCount();
if ($nbinsert !== 1) {
    echo "<br/>insertion 3 KO" ;
} else {
    echo "<br/>insertion 3 OK" ;
}

/* requête 4 : les données temporelles sont intégralement calculées par PHP
avant insertion dans la table DB2 */
$requete = "INSERT INTO TESTDATES (DATDAT, TIMTIM, TIMSTAMP, DATNUM,
HEUNUM) VALUES(?, ?, ?, ?, ?)" ;

// récupération du temps exprimé en microsecondes
$microtime1 = microtime(true);
// récupération de la partie décimale pour l'utiliser sur un Timestamp DB2
$microtime2 = intval(($microtime1 - intval($microtime1)) * 1000000) ;
// date système au format ISO
$datesys = date("Y-m-d");
// heure système au format ISO
$heursys = date("H:i:s");
// Timestamp système avec ajout de la partie décimale des microsecondes
$stampsys = date("Y-m-d-H.i.s.") . $microtime2 ;
// Date système au format numérique simple (AAAAMMJJ)
$datenum = date("Ymd");
// Heure système au format numérique simple (HHMMSS)
$heurnum = date("His");

$st = $db->prepare($requete);
$st->execute(array($datesys, $heursys, $stampsys, $datenum, $heurnum));
$nbinsert = $st->rowCount();
if ($nbinsert !== 1) {
    echo "<br/>insertion 4 KO" ;
} else {
    echo "<br/>insertion 4 OK" ;
}

/* affichage des données calculées pour la dernière requête, pour contrôle
avec les données insérées dans la table DB2 */
echo '<br/> $datesys = ' . $datesys ;
echo '<br/> $heursys = ' . $heursys ;
echo '<br/> $stampsys = ' . $stampsys ;
echo '<br/> $microtime1 = ' . $microtime1 ;
echo '<br/> $microtime2 = ' . $microtime2 ;
echo '<br/> $datenum = ' . $datenum ;
echo '<br/> $heurnum = ' . $heurnum ;
?>
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

Un rapide coup d'œil au contenu de la table après une première exécution du script me donne les résultats suivants :

DATDAT	TIMTIM	TIMSTAMP	DATNUM	HEUNUM
25/04/09	14:56:31	2009-04-25-14.56.31.483456	0	0
25/04/09	14:56:31	2009-04-25-14.56.31.537176	0	0
25/04/09	14:56:31	2009-04-25-14.56.31.597320	20.090.425	145.631
25/04/09	14:58:36	2009-04-25-14.58.36.484379	20.090.425	145.836

A noter : dans la 3^{ème} requête SQL, j'ai utilisé la syntaxe suivante :

```
INSERT INTO TESTDATES (DATDAT, TIMTIM, TIMSTAMP, DATNUM, HEUNUM)
VALUES (curdate(), curtime(), now(),
        integer(replace(char(curdate()), ISO, '-', '')),
        integer(replace(char(curtime()), ISO, '.', '')))
```

En fait l'usage de la fonction *integer()* n'est plus indispensable à partir de la V5R3 de l'OS/400, car en V5R3, DB2 opère une conversion implicite, si le format renvoyé par la formule est compatible avec la colonne destinataire.

Et dans l'autre sens ?

Si je lis une ligne de la table TESTDATES, comment faire pour afficher dans un script PHP le contenu des colonnes numériques DATNUM et HEUNUM, dans les formats « date » et « heure » tels que PHP les comprend ?

La solution se trouve du côté des fonctions *strtotime()* et *date_parse()* :

Code PHP	Résultat affiché dans le navigateur
<pre>\$a = strtotime('20090425'); echo strftime('%Y-%m-%d', \$a);</pre>	2009-04-25
<pre>\$a = strtotime('140556'); echo strftime('%H:%M:%S', \$a);</pre>	14:05:56
<pre>\$a = strtotime('20090425140556'); echo strftime('%Y-%m-%d %H:%M:%S', \$a);</pre>	2009-04-25 14:05:56
<pre>\$a = date_parse('20090428140556'); print_r(\$a);</pre>	<pre>Array ([year] => 2009 [month] => 4 [day] => 28 [hour] => 14 [minute] => 5 [second] => 56 [fraction] => 0 [warning_count] => 0 [warnings] => Array () [error_count] => 0 [errors] => Array () [is_localtime] =>)</pre>

Vous pouvez utiliser l'une ou l'autres des solutions proposées dans le tableau ci-dessus, en fonction du format d'origine des dates et heures stockées dans vos tables DB2.

Livre blanc sur l'utilisation de PHP en environnement IBM i

Pour les données de type « Timestamp DB2 », c'est un petit plus compliqué, mais on peut s'en sortir en utilisant les expressions régulières. Exemple :

```
// Timestamp tel qu'il est renvoyé par DB2
$date = '2009-04-25 14.05.56.123456';
// Application d'un masque pour extraire les éléments du Timestamp
preg_match('/(\d{4})-(\d{2})-(\d{2}) (\d{2}).(\d{2}).(\d{2}).(\d{6})/',
    $date, $date_parts);
// affichage du contenu du tableau
print_r($date_parts);
```

On obtient le tableau `$date_parts` ci-dessous, il est alors facile de recombinaison les éléments à sa guise :

```
Array (
[0] => 2009-04-25-14.05.56.123456
[1] => 2009
[2] => 04
[3] => 25
[4] => 14
[5] => 05
[6] => 56
[7] => 123456
)
```

Vous pouvez vous amuser à relire la dernière ligne de la table TESTDATES, et à tester les conversions de date et d'heure au moyen d'un script tel que celui ci-dessous :

```
$requete = "select * from testdates where
            rrn(testdates) = (select count(*) from testdates)" ;
$st = $db->query($requete);
$data = $st->fetch(PDO::FETCH_LAZY);

// affichage des données « brut de fonderie »
echo $data->DATDAT, "<br/>", $data->TIMTIM, "<br/>" ;
echo $data->TIMSTAMP, "<br/>", $data->DATNUM, "<br/>" ;
echo $data->HEUNUM, "<br/>" ;

// extraction des éléments du Timestamp
preg_match('/(\d{4})-(\d{2})-(\d{2}) (\d{2}).(\d{2}).(\d{2}).(\d{6})/',
    $data->TIMSTAMP, $date_parts);
print_r($date_parts);
echo '<br/>';

// Conversion de DATNUM au format « date »
$a = strtotime($data->DATNUM);
echo strftime('%Y-%m-%d', $a), '<br/>';

// Conversion de HEUNUM au format « time »
$a = strtotime($data->HEUNUM);
echo strftime('%H:%M:%S', $a), '<br/>';
```

10 La gestion des transactions sous PDO

Pour pouvoir gérer des transactions avec PHP et DB2, il faut que les fichiers concernés soient journalisés. La solution la plus simple pour y parvenir, est de créer la bibliothèque via SQL, au moyen de l'ordre CREATE COLLECTION "mabib", ce qui a pour effet d'activer la journalisation par défaut.

En reprenant notre programme de reprise des pays (cf. extrait ci-dessous), et en supposant que la table LSTPAYS soit bien journalisée, l'intégration d'un mécanisme transactionnel se traduirait par l'ajout des lignes en gras ci-dessous :

```
try {
    $conn_db2->beginTransaction();

    $st1 = $conn_mysql->query($req_mysql) ;
    // préparation de la requête d'insertion
    $st2 = $conn_db2->prepare($req_db2) ;
    while($row_data1 = $st1->fetch(PDO::FETCH_LAZY)) {
        $wi += 1 ;
        // exécution de la requête d'insertion
        $st2->execute(array($row_data1->CODFRA,
                        $row_data1->CODISO,
                        $row_data1->LIBELLE
                        )) ;
    }
    $conn_db2->commit();
} catch (PDOException $e) {
    $conn_db2->rollBack();

    echo 'Error : ' . $e->getMessage() . '<br/>';
    echo 'Code : ' . $e->getCode() . '<br/>';
    echo 'File : ' . $e->getFile() . '<br/>';
    echo 'Line : ' . $e->getLine() . '<br/>';
    echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
}
```

N.B. : dans l'exemple ci-dessus, le Rollback est placé à l'intérieur du paragraphe Catch, pour annuler la mise à jour en cas d'erreur. Le Rollback peut aussi être placé à l'intérieur du paragraphe Try, pour annuler une mise à jour (ou création) pour des raisons fonctionnelles plutôt que techniques.

Mais attention, on ne peut pas enchaîner coup sur coup, et dans un même bloc de code, 2 Commit, ou encore 2 Rollback, ou encore 1 Rollback et 1 Commit, car ces ordres ont pour effet de clore la transaction en cours. Il convient donc de mettre en place des tests appropriés permettant d'éviter ce cas de figure, par exemple :

```
$conn_db2->beginTransaction();
$transac = 'OK' ;
... traitement dans lequel la valeur de $transac peut changer ...
if ($transac == 'OK' ) {
    $conn_db2->commit();
} else {
    $conn_db2->rollBack();
}
```

11 Dialogue entre PHP et RPG via les procédures stockées DB2

Nous arrivons bientôt à la fin de ce livre blanc. Je pense que vous en savez maintenant suffisamment sur l'utilisation de PHP avec DB2 pour pouvoir aborder la technique que je souhaite vous présenter ici. Nous allons voir qu'il est possible pour un script PHP d'envoyer des informations à un programme RPG, et qu'il est également possible, pour ce même script PHP, de récupérer des informations en sortie du programme RPG.

Pour pouvoir mettre en place ce dialogue, il nous faut un intermédiaire, capable de dialoguer avec PHP et RPG, et cet intermédiaire est tout trouvé puisqu'il s'agit de la base de données DB2 for i. Cette dernière offre en standard un puissant mécanisme qui nous permet de réaliser ce petit miracle : les procédures stockées externes. Les procédures stockées externes constituent un sous ensemble du langage PL/SQL qui est intégré en standard dans DB2. Une procédure stockée externe est un objet DB2 qui encapsule l'appel d'un programme natif IBM i. Ce programme peut être de type RPGLE, ou autre (cf. liste plus loin dans ce chapitre). L'encapsulation du programme RPGLE passe par la création et la compilation d'un script SQL dans lequel nous définissons les paramètres d'entrée/sortie du programme encapsulé, avec en prime la possibilité de définir un ou plusieurs « result sets » (jeu de données résultantes).

Une fois la procédure stockée externe créée sur votre base de données DB2, vous pouvez l'invoquer en SQL via la commande CALL. Après la théorie, je vous propose de passer à la pratique. Pour l'exemple qui suit, j'ai choisi de créer un petit programme RPG qui vous retourne les références croisées d'un programme natif IBM i reçu en paramètre. Pour obtenir ces références croisées, nous nous appuyons sur la commande OS/400 DSPPGMREF que nous avons présentée au travers de différents articles sur le site Footthing.net.

Notre programme RPG va effectuer les actions suivantes :

1. réception du nom et de la bibliothèque d'un programme dont on souhaite extraire les références croisées
2. extraction via la commande DSPPGMREF des références croisées du programme souhaité, et stockage de ces références croisées dans un fichier temporaire
3. renvoi en sortie du programme, via la technique dite du « result set », du fichier temporaire extrait via la commande DSPPGMREF. On ne renverra pas la totalité des colonnes de la table temporaire, mais seulement quelques unes de ces colonnes, que nous avons jugé utile d'afficher via le script PHP appelant. De plus, on ne renvoie dans le « Result Set » que les objets de type « F » (pour « fichier ») qui sont référencés dans le programme analysé.

Code source du programme RPG PGMREFPROC :

```
*****
* DESCRIPTION          DSPPGMREF d'un pgm reçu en paramètre      *
* NOM DU PROGRAMME     PGMREFPROC                                *
* CREATION LE          20/11/2008 par Gregory JARRIGE             *
*****
H DECEDIT('0,') DATEDIT(*YMD)  DEBUG
*-----*
* DECLARATION DES VARIABLES
*-----*
D Requete              S              300
D Commande             S              200
D PGMREF               PR              Extpgm('QCMDEXC')
D String               1000          Const
D                      Options(*Varsize)
D Len                  15P 5 Const
*-----*
C* PARAMETRES D'ENTREES :
C  *ENTRY              PLIST
C                      PARM              CODBIB              10
C                      PARM              CODPGM              10
*-----*
/free
  Commande = 'DSPPGMREF PGM(' + %trim(CODBIB) + '/' + %trim(CODPGM)
    + ') OUTPUT(*OUTFILE) OBJTYPE(*ALL) '
    + ' OUTFILE(QTEMP/PGMREF1) OUTMBR(*FIRST*REPLACE) ' ;

  CALLP(E) PGMREF(Commande:%len(Commande));
  if not(%error()) ;
    Requete = 'SELECT WHFNAM, WHLNAM, WHSNAM, WHRFNO, WHFUSG, '
      + ' WHRFNM, WHRFSN, WHRFFN, WHOBJT '
      + ' FROM QTEMP/PGMREF1 WHERE WHOBJT = 'F'' ;

/end-free

* Préparation du jeu de données à renvoyer à la procédure stockée appelante
C/EXEC SQL
C+ PREPARE REQ1 FROM :Requete
C/END-EXEC

C/EXEC SQL
C+ DECLARE C1 CURSOR FOR REQ1
C/END-EXEC

C/EXEC SQL
C+ OPEN C1
C/END-EXEC

C/EXEC SQL
+ SET RESULT SETS CURSOR C1
C/END-EXEC

/free
  endif ;
  *InLR = *On;
/End-Free
```

Points importants : l'envoi d'un « Result Set » en sortie du programme passe obligatoirement par les étapes suivantes :

- Préparation du curseur (PREPARE...)
- Déclaration du curseur (DECLARE...)
- Ouverture du curseur (OPEN...)
- Génération du Result Set (SET RESULT SETS...)

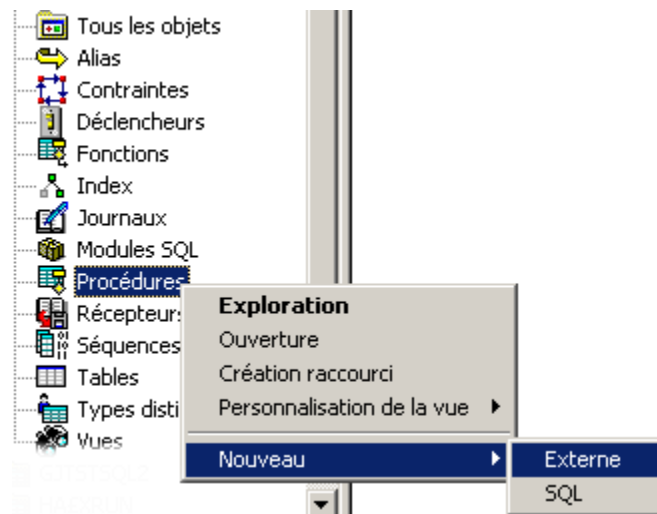
A noter qu'un même programme RPG peut renvoyer plusieurs « Result Sets ». Chaque « Result Set » est identifié par son nom (par exemple : C1, C2, etc...).

Il nous faut maintenant créer la procédure stockée DB2 externe qui va encapsuler notre programme RPG.

IBM propose un excellent redbook, librement téléchargeable au format PDF, qui explique dans le détail la manière de créer des procédures stockées DB2 (cf. lien au chapitre 16). Je vous recommande de lire attentivement le chapitre qui est consacré à la création de procédures stockées externes.

Il est important de souligner que le logiciel ACS d'IBM fournit de très bons assistants permettant de faciliter la création de procédures stockées. On peut y choisir de créer une procédure de type "SQL" (on s'appuie intégralement sur le langage normalisé PL/SQL qui est intégré à DB2), ou une procédure de type "externe" (on s'appuie alors sur un programme développé dans un langage autre que SQL, tel que le RPGLE, le C, le Cobol, etc...).

Création d'une procédure stockée externe sous ACS :



Dans les copies d'écran suivantes, on crée une procédure stockée de type "externe", s'appuyant sur le programme SQLRPGLE PGMREFPROC.

Premier onglet : la procédure stockée va renvoyer 1 jeu de données (result set), ce qui explique qu'on ait saisi "1" dans le nombre maximal de fichiers de résultats. Les autres options sont laissées avec leurs valeurs par défaut.

Général Paramètres Programme externe

Procédure : PGMREFPRC

Description : Procédure cataloguée effectuant un DSPPGMREF

Nombre maximal de fichiers de résultats : 1

☐ Même valeur renvoyée à partir d'appels successifs pour des paramètres identiques (fonction déterministe)

☐ Validation des modifications lors du renvoi du contrôle à l'appelant

☐ Début d'un nouveau point de sauvegarde lors de l'appel

Accès aux données : Modifie des données SQL

Nom spécifique :

Deuxième onglet : définition des paramètres d'entrée/sortie du programme SQLRPGLE utilisé par la procédure stockée.

Général Paramètres Programme externe

Nom du paramètre	Type	Longueur	CCSID	E-S	Relev...	Description
codbib	CHARACTER	10		IN		
codpgm	CHARACTER	10		IN		

Insertion

Suppression

Style de paramètre :

☒ SQL

☐ Simple, valeurs indéfinies admises

☐ Simple, valeurs indéfinies non admises

☐ Java

Troisième onglet : on y précise simplement que le programme encapsulé sera de type « RPGLE ». A titre d'information, la liste des types de programmes pouvant être encapsulés dans une procédure stockée externe est la suivante :

C
C++
CL
COBOL
COBOLLE
PLI
REXX
RPG
RPGLE

Livre blanc sur l'utilisation de PHP en environnement IBM i

La validation du 3ème onglet a pour effet de générer automatiquement la procédure stockée sur l'IBM i. Attention : l'objet créé est un objet de type DB2, et non pas un objet de type OS/400, donc ne vous étonnez pas de ne pas le voir par un simple WRKOBJ. Si on souhaite le supprimer, on peut le faire en mode SQL avec la requête suivante :

```
DROP PROCEDURE.PGMREFPRC
```

On peut également générer le source SQL de création de la procédure stockée, par exemple pour pouvoir le réexécuter sur un autre IBM i. Pour faire cela sous ACS, il suffit de faire un clic-droit sur la procédure, et de sélectionner l'option "Génération d'instructions SQL...". On peut choisir de générer le source dans un fichier texte (PC) ou dans un membre de fichier source IBM i.

Une fois généré dans un membre de fichier source, le script SQL peut être retravaillé sur l'IBM i par exemple via PDM, il peut également être transféré sur une autre machine, il peut également être exécuté via la commande OS/400 RUNSQLSTM, ce qui aura pour effet de recréer la procédure. Attention, si elle existe déjà, il faut la supprimer au préalable via un DROP PROCEDURE, ou mieux encore, vous pouvez remplacer l'ordre « CREATE PROCEDURE » par l'ordre « CREATE OR REPLACE PROCEDURE », ce qui facilitera la recompilation (le paramètre « OR REPLACE » est apparu avec la V7R1 de DB2 for i).

Il est important de souligner que l'on peut tester sous ACS le bon fonctionnement de la procédure stockée. Par exemple, en passant par l'option « Exécution de scripts SQL » de ACS, on peut saisir la requête suivante (en supposant que la procédure se trouve dans la bibliothèque MABIB) :

```
CALL MABIB.PGMREFPRC ('MABIB', 'MONPGM')
```

Si ma procédure stockée fonctionne bien, une fenêtre doit s'afficher dans ACS, me renvoyant le contenu du « Result Set », soit les références croisées du programme « MONPGM » de la bibliothèque « MABIB ».

Voici le code source de la procédure stockée PGMREFPRC :

(Cette procédure a été générée en s'appuyant sur l'assistant de création d'ACS)

```
-- Procédure stockée générée via ACS,
-- dont le principe est d'effectuer un DSPPGMREF d'un programme
-- reçu en paramètre, et de renvoyer un RESULT SET correspondant
-- au contenu du fichier généré par le DSPPGMREF
--
-- Procédure créée avec la commande suivante :
-- RUNSQLSTM SRCFILE(MABIB/QSQLSRC)
-- SRCMBR(PGMREFPRC) COMMIT(*NONE) NAMING(*SQL)
--
-- Commande d'exécution de la procédure :
-- CALL MABIB.PGMREFPRC ('xxMABIBxx', 'xxMONPGMxx');
--
CREATE PROCEDURE MABIB.PGMREFPRC (
  IN CODBIB CHAR(10),
  IN CODPGM CHAR(10))
DYNAMIC RESULT SETS 1
LANGUAGE RPGLE
SPECIFIC MABIB.PGMREFPRC
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
EXTERNAL NAME 'MABIB/PGMREFPROC'
PARAMETER STYLE SQL ;
--
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
COMMENT ON SPECIFIC PROCEDURE MABIB.PGMREFPRC  
IS 'Procédure encapsulant un DSPPGMREF' ;
```

A partir de maintenant, nous sommes en mesure d'invoquer cette procédure stockée, et de récupérer son « Result Set » à l'intérieur d'un script PHP. C'est exactement ce que fait le script suivant :

Code source du script PHP testpgmrefprc.php :

```
<html>  
<head>  
    <title>R&eacute;f&eacute;rences crois&eacute;es d'un programme</title>  
</head>  
<body>  
<?php  
  
    // configuration de la connexion à la base de données  
    require_once 'config.php' ;  
  
    // chargement de la fonction getmicrotime  
    require_once 'fonctions.php' ;  
  
    // récupération des paramètres du $_GET  
    if (isset($_GET)) {  
        $params = $_GET ;  
    } else {  
        $params = array() ;  
    } ;  
  
    // paramètres d'appel de la procédure cataloguée  
  
    $codbib = trim($params['id_database']) ;  
    $codpgm = trim($params['id_pgm']) ;  
  
    echo 'Liste des fichiers utilis&eacute;s par le programme : ' . trim($codbib) .  
    '/' . trim($codpgm) . '<p/>' ;  
  
    echo <<<EOM  
        <table border='1'>  
            <tr>  
                <td>WHFNAM</td>  
                <td>WHLNAM</td>  
                <td>WHSNAM</td>  
                <td>WHRFNO</td>  
                <td>WHFUSG</td>  
                <td>WHRFNM</td>  
                <td>WHRFSN</td>  
                <td>WHRFFN</td>  
                <td>WHOBJT</td>  
            </tr>  
        </table>  
    EOM;  
  
    $time_start = getmicrotime() ;  
  
    // requête d'appel de la procédure stockée  
    $sql = "CALL MABIB.PGMREFPRC (?, ?)" ;  
    try {  
        $st = $db->prepare($sql);  
        $st->bindParam(1, $codbib);  
        $st->bindParam(2, $codpgm);  
        $st->execute() ;  
        if ($st) {  
            do {  
                $row_data = $st->fetchAll(PDO::FETCH_ASSOC);
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
        if ($row_data) {
            for($yIndex = 0; $yIndex < count($row_data); $yIndex++) {
                echo <<<EOM
                <tr>
                <td>{$row_data[$yIndex]['WHFNAM']}</td>
                <td>{$row_data[$yIndex]['WHLNAM']}</td>
                <td>{$row_data[$yIndex]['WHSNAM']}</td>
                <td align="right">{$row_data[$yIndex]['WHRFNO']}</td>
                <td align="right">{$row_data[$yIndex]['WHFUSG']}</td>
                <td>{$row_data[$yIndex]['WHRFNM']}</td>
                <td>{$row_data[$yIndex]['WHRFNS']}</td>
                <td align="right">{$row_data[$yIndex]['WHRFFN']}</td>
                <td align="center">{$row_data[$yIndex]['WHOBJT']}</td>
                </tr>
EOM;
            }
        }
    } while ($st->nextRowset());
}
} catch (PDOException $e) {
    if ($e->getCode() == 24000) {
        echo 'Aucune donn&eacute;e disponible pour le programme indiqu&eacute;;<br/>'
;
    } else {
        echo 'Erreur grave autre que 24000 :<br/>' ;
        echo 'Error : ' . $e->getMessage() . '<br/>';
        echo 'Code : ' . $e->getCode() . '<br/>';
        echo 'File : ' . $e->getFile() . '<br/>';
        echo 'Line : ' . $e->getLine() . '<br/>';
        echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
    }
}

echo <<<EOTABLE
    </table>
EOTABLE;

$time_stop = getmicrotime() ;
$time_ecart = $time_stop - $time_start ;

echo "<p><h2>Temps d'ex&eacute;cution de la proc&eacute;dure stock&eacute;e
:</h2></p>";
echo "{$time_ecart} secondes </p>" ;

?>
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

Le résultat obtenu dans le navigateur internet pour le programme DESFICP est le suivant :

Liste des fichiers utilisés par le programme : MABIB/DESFICP

WHFNAME	WHLNAME	WHSNAME	WHRFNO	WHFUSG	WHRFNM	WHRFSN	WHRFFN	WHOBJT
DESFICPR	MABIB	DESFICPR	10	2	ENTETE	11820368BE66F	1	F
DESFICPR	MABIB	DESFICPR	10	2	ENTCOM	1E8098CA56998	9	F
DESFICPR	MABIB	DESFICPR	10	2	ENTLF	073DFA2E14470	3	F
DESFICPR	MABIB	DESFICPR	10	2	ENTLF2	07BA9B3B317A7	3	F
DESFICPR	MABIB	DESFICPR	10	2	LSTENT	0142323C5D624	0	F
DESFICPR	MABIB	DESFICPR	10	2	LSTDET	1667768810ED5	2	F
DESFICPR	MABIB	DESFICPR	10	2	LSTTOT	130D6E6AA588D	1	F
DESFICPR	MABIB	DESFICPR	10	2	LSTBAS	0142323C2C223	0	F
DESFICPR	MABIB	DESFICPR	10	2	ENTZON	0061623E9D716	0	F
DESFICPR	MABIB	DESFICPR	10	2	LIGDET	1A6C0978A3115	10	F
DDSBAS	QTEMP	DDSBAS	1	1	QWHFDBAS	4A3530A4B7CA3	41	F
DDSFMT	QTEMP	DDSFMT	1	1	QWHFDFMT	4A8BB90D72FCD	53	F
DDSMBR	QTEMP	DDSMBR	1	1	QWHFDMML	5B61CBE120082	32	F
DDSZON	QTEMP	DDSZON	1	1	QWHDRFFD	48CE605002B02	96	F
DDSDBR	QTEMP	DDSDBR	1	1	QWHDRDBR	4BD9868FD2CF2	16	F
DDSKEY	QTEMP	DDSKEY	1	1	QWHFDACP	4C91E57AA1783	32	F
DDSDEP	QTEMP	DDSDEP	1	1	QWHFDMBR	351AE2B8355B3	146	F

Temps d'exécution de la procédure stockée :

0.14506101608276 secondes

Points importants :

Le temps d'exécution est calculé au moyen de la fonction `getmicrotime()` que nous avons vue dans un précédent chapitre.

Vous noterez que pour la première fois, j'ai utilisé la méthode `bindParam()` de PDO pour encapsuler les paramètres passés à la requête SQL :

```
$sql = "CALL MABIB.PGMREFPRC (?, ?)" ;  
...  
$st = $db->prepare($sql);  
$st->bindParam(1, $codbib);  
$st->bindParam(2, $codpgm);  
$st->execute() ;  
...
```

Ce n'était pas indispensable, mais je voulais profiter de l'occasion pour vous montrer cette technique. On aurait tout aussi bien pu écrire la variante suivante :

```
$sql = "CALL MABIB.PGMREFPRC (:bib, :pgm)" ;
```

```
...
$stmt = $db->prepare($sql);
$stmt->bindParam(':bib', $codbib);
$stmt->bindParam(':pgm', $codpgm);
$stmt->execute();
...
```

Et bien sûr la version plus simple, sans bindParam(), qui donne également le même résultat :

```
$sql = "CALL MABIB.PGMREFPRC (?, ?)" ;
...
$stmt = $db->prepare($sql);
$stmt->execute(array($codbib, $codpgm) ;
...
```

La boucle principale du script PHP consiste à parcourir la liste des « Result Set » renvoyés par la procédure stockée. Comme le programme RPG utilisé ici ne renvoie qu'un seul « Result Set », cette boucle ne s'exécutera qu'une seule fois.

```
do { ... } while ($stmt->nextRowset());
```

Le « Result Set » récupéré par le script PHP est un tableau à deux dimensions (stocké dans la variable \$row_data). Si on affiche le contenu de ce tableau avec la fonction PHP print_r(), on obtient le résultat suivant :

```
[0] => Array (
    [WHFNAM] => DESFICPR
    [WHLNAM] => MABIB
    [WHSNAM] => DESFICPR
    [WHRFNO] => 10
    [WHFUSG] => 2
    [WHRFNM] => ENTETE
    [WHRFSN] => 11820368BE66F
    [WHRFFN] => 1
    [WHOBJT] => F )
[1] => Array (
    [WHFNAM] => DESFICPR
    [WHLNAM] => MABIB
    [WHSNAM] => DESFICPR
    [WHRFNO] => 10
    [WHFUSG] => 2
    [WHRFNM] => ENTCOM
    [WHRFSN] => 1E8098CA56998
    [WHRFFN] => 9
    [WHOBJT] => F )
etc....
```

On pourrait donc parcourir le tableau au moyen de 2 boucles imbriquées, la première boucle parcourant les lignes (0, 1, etc...), la seconde parcourant les colonnes. Comme les colonnes sont nommées explicitement (et non pas seulement numérotées), et que je souhaite contrôler l'ordre d'affichage des colonnes, j'ai opté pour une approche plus « rigide », en utilisant une syntaxe du type :

```
row_data[<numéro de ligne>][<nom de colonne>]
```

A noter que c'est la fonction `fetchAll()`, associée au paramètre `PDO::FETCH_ASSOC`, qui permet de récupérer les noms des colonnes renvoyées par le « Result Set ».

Livre blanc sur l'utilisation de PHP en environnement IBM i

On aboutit donc à la boucle ci-dessous, qui parcourt les lignes du « Result Set », et nous permet d'afficher le tableau présenté page précédente :

```
for($yIndex = 0; $yIndex < count($row_data); $yIndex++) {
    echo <<<EOM
    <tr>
    <td>{$row_data[$yIndex]['WHFNAM']}
```

ATTENTION : les tests que j'ai effectués sur PHP en version 5.2.6 semblent indiquer que la fonction `nextRowset()` ne donne pas tout à fait le résultat attendu : il ne m'a pas été possible de parcourir plusieurs Result Sets renvoyés par une même procédure stockée. Ce problème semble connu également des développeurs MySQL, comme en témoignent certaines questions posées dans les forums. Ceci étant dit, le problème pourrait provenir tout aussi bien de DB2 (qui ne serait pas en mesure de traiter plus d'un « Result Set »), ou du compilateur RPGLE (éventuellement pour les mêmes raisons). Il convient donc d'être prudent dans le diagnostic, et en l'absence de certitude, de limiter l'échange de données à un seul « Result Set ». De plus, le problème peut être contourné en utilisant conjointement :

- des paramètres de sortie (ou d'entrée/sortie) pour récupérer des valeurs en sortie de la procédure stockée (par exemple : un prix brut et un prix net, dans le cadre d'un composant de calcul de prix),
- un Result Set pour récupérer des informations complémentaires (par exemple : le détail des remises cumulées ayant permis de calculer le prix net).

Quelques variantes sont présentées ci-dessous démontrant la possibilité d'utiliser des paramètres d'entrée/sortie en complément du « Result Set ».

L'utilisation de paramètres d'E/S sur la procédure stockée nécessite de modifier légèrement :

- la procédure stockée,
- le programme RPG,
- le script PHP.

Seules les parties modifiées sont indiquées **en gras** ci-dessous :

- pour la procédure stockée :

```
CREATE PROCEDURE MABIB.PGMREFPR2 (
    IN CODBIB CHAR(10) ,
    IN CODPGM CHAR(10) ,
    OUT CODRET CHAR(10) )
DYNAMIC RESULT SETS 1
...
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

- pour le programme RPG :

```
C      *ENTRY          PLIST
C                      PARM                CODBIB          10
C                      PARM                CODPGM          10
C      PARM                CODRET          10
...
/free
    endif ;
    CODRET = 'SUPER' ;
    *InLR = *On;
/End-Free
```

- pour le script PHP :

```
$sql = "CALL MABIB.PGMREFPR2 (?, ?, ?)" ;
try {
    $stmt = $conn->prepare($sql);
    $stmt->bindParam(1, $codbib);
    $stmt->bindParam(2, $codpgm);
    $stmt->bindParam(3, $codret, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT, 11) ;
    $stmt->execute() ;
    echo 'code retour = ', $codret , '<br/>';
    if ($stmt) {
        ...
    }
}
```

Si tout s'est bien passé, le script PHP va afficher « SUPER » à l'exécution.

A noter : ce sont les paramètres 3 et 4 de la méthode « bindparam() » qui conditionnent la possibilité de récupérer un paramètre provenant de la procédure stockée. Le 4^{ème} paramètre correspond **au minimum à la longueur du paramètre retourné + 1**.

12 Nouveautés de la V7

Avec l'arrivée de la V7, et en plus de ses différentes releases (V7R1, V7R2, etc.), IBM a ajouté un nouveau process de livraison permettant de délivrer plus rapidement des nouveautés à ses clients IBM i. Ce process est désigné sous le terme TR, pour « Technology Refresh ».

Au travers des différentes releases, et des TR successives intercalées entre les releases, IBM a commencé à intégrer dans le SQL de DB2 for i, tout un tas de nouvelles fonctions et procédures stockées, qui couvrent de nombreux sujets et répondent à de nombreux besoins.

Regroupées sous le terme générique de « DB2 for i Services », ces nouveaux objets DB2 permettent d'accéder facilement à un grand nombre de ressources de l'IBM i.

Je vous propose juste quelques exemples, et vous pourrez approfondir le sujet au travers des trois liens que je vais vous fournir à la fin de ce chapitre.

Premier exemple :

La procédure stockée QCMDXEC permet d'exécuter des commandes systèmes à partir de SQL. Ce n'est pas vraiment une nouveauté, mais ce qui est nouveau, c'est que - depuis la TR7 de la V7R1 - on n'est plus obligé de préciser la longueur de la commande système à exécuter, en complément de la commande elle-même. La procédure stockée est en mesure de déterminer elle-même la longueur de la commande, ce qui permet au développeur de créer à la volée des commandes systèmes complexes, et de les transmettre à la procédure sans se préoccuper de la longueur de ces commandes.

Deux exemples d'utilisation :

- Ajout d'une bibliothèque dans la "library list" :

```
CALL QSYS2.QCMDXEC('ADDLIBL PRODLIB2');
```

- La même chose dans une procédure stockée, avec une "expression" concaténée à la volée :

```
DECLARE V_LIBRARY_NAME VARCHAR(10);  
...  
SET V_LIBRARY_NAME = 'PRODLIB2';  
...  
CALL QSYS2.QCMDXEC('ADDLIBL ' || CONCAT V_LIBRARY_NAME);
```

Pour vous simplifier la vie, dans vos projets de tous les jours, je vous recommande de créer une fonction PHP `genCmdSys` qui aura pour unique fonction d'encapsuler votre commande IBM i dans le CALL de la procédure QCMDXEC :

```
function genCmdSys ($cmd) {  
    return "CALL QCMDXEC ('{$cmd}')" ;  
}
```


Livre blanc sur l'utilisation de PHP en environnement IBM i

A partir de là, il sera facile de créer, par exemple, une fonction générant une commande de type OVRDBF, de manière à pouvoir lire le contenu d'un fichier DB2 multi-membre à partir d'un script PHP :

```
function genOverlayDBFile($file, $library, $member) {
    $library = strtoupper(trim($library));
    $file = strtoupper(trim($file));
    $member = strtoupper(trim($member));
    $cmd = "OVRDBF FILE({$file}) TOFILE({$library}/{ $file}) MBR({$member})
OVRSCOPE(*JOB)";
    return genCmdSys($cmd);
}

echo genOverlayDBFile('myfile', 'mylib', 'mymember');

// Résultat obtenu en sortie :
// CALL QCMDEXC ('OVRDBF FILE(MYFILE) TOFILE(MYLIB/MYFILE) MBR(MYMEMBER)
// OVRSCOPE(*JOB) ')
```

Attention : le paramètre OVRSCOPE(*JOB) pourrait vous surprendre, mais je vous garantis qu'il est indispensable dans ce contexte.

Et bien sûr, quand il y a un OVRDBF, il y a forcément un DLTOVR derrière :

```
function dropOverlayDBFile($file='*ALL') {
    $file = strtoupper(trim($file));
    $cmd = "DLTOVR FILE({$file}) LVL(*JOB)";
    return genCmdSys($cmd);
}
```

Attention : le paramètre LVL(*JOB) est indispensable pour pouvoir enchaîner plusieurs OVRDBF sur un même fichier et dans un même travail.

Second exemple :

Très pratique, la nouvelle fonction DELIMIT_NAME renvoie une valeur avec des délimiteurs (guillemets et/ou apostrophes) répondant à différentes problématiques rencontrées par les développeurs SQL (et par les développeurs webs). Le paramètre d'entrée est une chaîne de 128 caractères maximum (en cas de dépassement, la valeur renvoyée est tronquée à cette longueur). La valeur renvoyée est un VARCHAR contenant une chaîne correctement délimitée.

Exemple :

```
SELECT
DELIMIT_NAME('ABC'), -- ABC
DELIMIT_NAME('ABC') , -- "ABC"
DELIMIT_NAME('TEST"NAME'), -- "TEST"NAME"
DELIMIT_NAME('TEST'NAME2'), -- "TEST'NAME2"
DELIMIT_NAME('NEW') -- "NEW"
FROM SYSIBM.SYSDUMMY1;
```

Troisième exemple :

La nouvelle vue DB2 LIBRARY_LIST_INFO permet de récupérer la liste des bibliothèques courante du travail en cours d'exécution.

Exemple :

```
SELECT * FROM QSYS2.LIBRARY_LIST_INFO;
```

Quatrième et dernier exemple :

La nouvelle vue DB2 SCHEDULED_JOB_INFO permet de consulter en temps réel le contenu du planning de travaux de l'IBM i (auquel on accède habituellement au moyen de la commande WRKJOBSCDE).

Exemple :

```
SELECT * FROM QSYS2.SCHEDULED_JOB_INFO A
WHERE A.STATUS IN ('HELD', 'SAVED')
ORDER BY SCHEDULED_BY;
```

Ce n'était qu'un petit aperçu, il y a beaucoup d'autres fonctions à découvrir.

Ces nouveaux objets DB2 étant accessibles via SQL, vous pouvez les « consommer » via PHP et votre connexion ODBC. Je vous laisse imaginer l'étendue des possibilités que cela offre...

Pour faire le tour des nombreuses nouveautés apparues sur DB2 for i en V7R1, vous avez cette synthèse assez complète que j'ai rédigée il y a quelques années maintenant :

https://github.com/gregja/SQLMasters/blob/devel/SQL_DBTwo_NewsV7.pdf

Pour un tour d'horizon des nouveautés de DB2 for i, plus orientées sur la gestion des travaux et des PTF, vous avez cette synthèse publiée par la société Gaïa :

<https://www.gaia.fr/wpfb-file/s46-les-nouveautes-v7-de-la-gestion-des-travaux-et-des-ptfs-pdf/>

Le slide ci-dessous, publié par IBM en mars 2022 propose une synthèse des nouveautés apparues sur DB2 for i en V7R3 et V7R4 :

<https://www.itheis.com/wp-content/uploads/2018/08/Webinar-ITHEIS-IBM-du-17-mars-2022-Partie-IBM-nouveautes-DB2.pdf>

Parmi les nouveautés très intéressantes que j'ai eu l'occasion d'expérimenter récemment, il y a notamment les fonctions DB2 dédiées à la gestion de l'IFS. Grâce à ces fonctions, on peut lire ou écrire des fichiers dans des répertoires de l'IFS :

<https://www.ibm.com/docs/en/i/7.3?topic=services-ifs>

13 Portage de code PHP vers Zend Server for i

Le portage de code PHP développé sur un stack PHP sur PC, vers un stack PHP sur IBM i (Zend Server), ne pose généralement pas de problème, sauf dans un cas précis : la connexion à la base de données DB2.

Nous avons vu que, pour nous connecter à DB2 à partir de notre stack PHP sur PC, nous devons utiliser l'extension `pdo_odbc`. Mais sur un Zend Server for i, pour connecter PHP et DB2, nous devons passer par l'extension `ibm_db2`. Ce qui veut dire que, potentiellement tout le code PHP utilisant `pdo_odbc` doit être révisé pour fonctionner sur IBM i. Enfin, ça c'était vrai autrefois... car il semble que depuis 4 ou 5, IBM ait fait des efforts pour faciliter le portage de code, comme l'indique la page ci-dessous du site YoungiProfessionals (Yip) :

<http://yips.idevcloud.com/wiki/index.php/PHP/DB2Documents>

En effet, sur la page ci-dessus, il est précisé qu'il est possible d'utiliser « `pdo_odbc` » aussi sur IBM i, ce qui devrait éliminer les problèmes que j'ai connus il y a 10 ans. Vous noterez que j'emploie le conditionnel car je n'ai pas eu l'occasion de tester cette possibilité.

Cette problématique m'avait conduit à développer une petite couche base de données, que j'avais appelée MacaronDB, et qui me permettait, entre autres choses, de swapper librement entre les extensions « `pdo_odbc` » et « `ibm_db2` ». Si le sujet vous intéresse, vous trouverez le projet MacaronDB et sa documentation en suivant ce lien :

<https://github.com/gregja/macarondb/blob/master/DOCUMENTATION.md>

Cette forte contrainte concernant l'extension DB2 - différente selon l'environnement d'exécution (sur IBM i ou hors IBM i) - a pesé lourdement, durant une longue période, sur le choix des frameworks et bibliothèques open-source pouvant être utilisés en environnement IBM i. En effet, de nombreux frameworks et bibliothèques (notamment de type ORM) utilisaient exclusivement PDO pour gérer les accès bases de données. Le fait de ne pas pouvoir utiliser PDO sur IBM i rendait donc le portage de certains projets open-source très compliqué, donc trop coûteux, donc impossible.

Rares étaient en effet les frameworks offrant une couche d'accès base de données permettant de swapper de PDO à `db2_connect`. En fait, il n'y en avait qu'un, c'était le Zend Framework dans sa version 1, grâce à son composant `Zend_DB`.

Pour de plus amples précisions sur la problématique des frameworks, je vous renvoie au chapitre 15.

14 Règles de nommage de DB2 for i

Concernant les règles de nommage, DB2 for i a une convention particulière, qui n'existe pas sur les autres versions de DB2.

En effet, avec DB2 for i, on a le choix entre 2 modes de fonctionnement :

- le nommage "système" : dans ce mode, le caractère de séparation entre noms de bibliothèque et de table - dans les requêtes SQL - est un slash (BIBL/TABL), et on peut définir une liste de bibliothèques au moment où on se connecte à la base de données
- le nommage "SQL" : dans ce mode, le caractère de séparation entre noms de bibliothèque et de table est un point (BIBL.TABL), et on ne peut définir qu'une bibliothèque par défaut

En réalité, cette convention de nommage a été assouplie à partir de la V7R3 si mes souvenirs sont bons (ou était-ce la V7R2 ?).

Quoi qu'il en soit, si vous êtes en V7R3 ou version supérieure, vous pouvez utiliser le point comme caractère séparateur (entre bibliothèque et table), même si vous utiliser le nommage "système". Je vous garantis que ce petit changement est un vrai soulagement tant cette problématique de slash et de point était pénible à gérer au quotidien.

On a vu dans un chapitre précédent qu'une "connection string" se présentait sous la forme suivante :

```
Driver={IBM i Access ODBC Driver};System=mySystem;Uid=myUser;Pwd=myPassword;
```

Mais on peut la compléter avec un jeu de paramètres complémentaires, comme dans l'exemple fictif suivant :

```
Driver={IBM i Access ODBC Driver}; System=mySystem; Uid=myUser; Pwd=myPassword;  
CCSID=1208;Port=1234;DATABASE=XXX;CMT=0;DFT=5;NAM=1;DSP=1;DEC=0;TFT=0;TSP=0
```

- **CCSID** (encodage) : 1208 pour l'UTF-8, 297 pour la France, pour les autres voir le lien en référence (ci-dessous)
- **PORT** (port IP) : sur IBM i, la valeur par défaut est généralement 1234 (donc valeur facultative), en cas de doute contacter son admin. réseau
- **NAM** (convention de nommage) : 0 (convention SQL), 1 (convention système)
- **DATABASE** : base de données par défaut (à utiliser exclusivement en convention SQL)
- **DBQ** : liste de bibliothèques séparées par des blancs (à utiliser exclusivement en convention système)
- **CMT** (commit mode) : valeurs possible => voir ci-après
- **DFT** (date format) : valeurs possible => voir ci-après
- **DSP** (car. séparateur / date): valeurs possible => voir ci-après
- **DEC** (séparateur / données décimales): valeurs possible => voir ci-après
- **TFT** (time format): valeurs possibles => voir ci-après
- **TSP** (car. séparateur / time): valeurs possibles => voir ci-après

Lien pour connaître les différents encodages possibles :

<https://www.ibm.com/docs/fr/zos-connect/zosconnect/3.0?topic=properties-coded-character-set-identifiers>

Livre blanc sur l'utilisation de PHP en environnement IBM i

Pour le paramètre CMT ("commit mode" ou 'niveau d'isolation'), les valeurs possibles sont les suivantes :

- 0 = Commit immediate (*NONE), valeur par défaut
- 1 = Read committed (*CS)
- 2 = Read uncommitted (*CHG)
- 3 = Repeatable read (*ALL)
- 4 = Serializable (*RR)

Concernant les paramètres DFT à TSP, les valeurs par défaut sont satisfaisantes dans la plupart des cas, mais si vous avez besoin de les personnaliser, suivez le guide.

Pour les paramètres DFT (date format) et TST (time format), les valeurs possibles sont les suivantes :

- 1 = ISO (par défaut)
- 2 = USA
- 3 = EUR
- 4 = JIS
- 5 = MDY
- 6 = DMY
- 7 = YMD
- 8 = JUL
- 10 = JOB (format du Job IBM i)

Pour le paramètre DEC (séparateur de décimale) :

- 1 = SLASH (par défaut)
- 2 = DASH
- 3 = PERIOD
- 4 = COMMA
- 5 = BLANK
- 6 = COLON
- 7 = JOB (job IBM i)

Pour les paramètres DSP (car. séparateur / date) et TSP (car. séparateur / time) :

- 1 = COLON (par défaut)
- 2 = PERIOD
- 3 = COMMA
- 4 = BLANK
- 5 = JOB (job IBM i)

D'autres paramètres (optionnels) sont disponibles avec le driver « IBM i Access ODBC Driver », paramètres qui sont détaillés dans cette page :

<https://www.connectionstrings.com/ibm-i-access-odbc-driver/>

Mais soyez prudents avec leur utilisation, si vous envisagez de porter votre code sur Zend Server for i, car certains de ces paramètres n'ont pas d'équivalent avec l'extension ibm_db2.

Livre blanc sur l'utilisation de PHP en environnement IBM i

Si vous voulez connaître les équivalences de paramétrage entre PDO et ibm_db2, vous pouvez vous reporter sur la documentation de l'extension ibm_db2 :

<https://www.php.net/manual/fr/function.db2-connect.php>

Cette problématique de paramétrage fait partie des points que je m'étais efforcé de gérer dans mon projet MacaronDB, comme on peut le voir dans le source des classes de connexion (pour PDO et ibm_db2). Vous pouvez vous en inspirer si besoin :

<https://github.com/gregja/macarondb/blob/master/ODBC/IBMi/DBConnex.php>

<https://github.com/gregja/macarondb/blob/master/DB2/IBMi/DBConnex.php>

15 Frameworks PHP

On l'a vu au chapitre 13, le choix d'un framework PHP capable de fonctionner librement sur IBM i a été longtemps contraint par les limitations relatives aux connecteurs bases de données pouvant être utilisés sur la plateforme.

En dehors du Zend Framework 1, le choix était quasi inexistant. Cela n'aurait pas été un problème si Zend n'avait pas décidé d'abandonner, vers 2010 ou 2011, le projet ZF1, au profit d'un ZF2 assez différent, incompatible avec ZF1, et dont le composant Zend_DB était très incomplet par rapport à celui de ZF1. Cette décision de Zend a été très mal vécue par un certain nombre de développeurs ZF (moi y compris).

Mais en ce début d'année 2022, une très bonne nouvelle est tombée : tel le Phénix, ZF1 renaît de ses cendres, avec le projet "ZF1 Future". Car au travers de « ZF1 Future », non seulement le projet ZF1 est à nouveau maintenu, mais en plus il a été mis à jour pour fonctionner sans problème avec PHP 7 et PHP 8. L'annonce a été faite par le Seiden Group, une société américaine qui réalise un salubre travail de défrichage pour les développeurs IBM i :

<https://www.seidengroup.com/2022/03/28/how-to-upgrade-zend-framework-1-to-php-7-4-or-8-x/>

Grâce à ce retour en force de ZF1, ces 2 slides de présentation de Alan Seiden, quoiqu'un peu anciens, redeviennent d'actualité :

- "From Zero to ZF" on IBM i Your first Zend Framework project (seidengroup.com)
<https://www.seidengroup.com/presentation%20slides/Your-first-Zend-Framework-Project-on-IBM-i.pdf>

- Confessions of an RPG programmer: Why use Zend Framework? (seidengroup.com)
<https://www.seidengroup.com/presentation%20slides/Zend%20Framework%20RPG%20Confessions.pdf>

Lien vers une doc d'intro à ZF1 Future : <https://zf1future.com/manual>

Lien vers le dépôt Github : <https://github.com/Shardj/zf1-future>

Pour les développeurs désireux de travailler avec un framework léger de type MVC, dans lequel ils pourraient coder leur propre couche d'accès base de données, le micro-framework Slim est une solution très intéressante à envisager. Slim est utilisé avec succès par certaines sociétés développant des projets PHP sur IBM i :

<https://www.slimframework.com/>

En complément de ce chapitre, je vous invite à lire cet article proposé par le Seiden Group :
<https://www.seidengroup.com/2022/03/17/how-to-choose-a-php-api-framework/>

16 Liens et bouquins, pour aller plus loin..

Un certain nombre de ressources que j'avais présentées dans la version 2 du présent dossier ont disparu aujourd'hui, comme par exemple celles fournies par le défunt – et regretté - site Developerworks d'IBM. J'ai donc été au regret de retirer un certain nombre de liens pourtant très intéressants.

Si le site Developerworks a disparu, le site YoungiProfessionals lui est toujours présent et actif. Il propose de nombreuses ressources pour les développeurs désireux d'utiliser des technologies open-source sur IBM i :

<http://yips.idevcloud.com/wiki/index.php/Main/HomePage>

Plusieurs très bons redbooks IBM, pour compléter vos connaissances sur différents sujets :

- Bringing PHP to Your IBM eServer IBM iSeries Server :

www.redbooks.ibm.com/abstracts/redp3639.html

- Developing PHP Applications for IBM Data Servers :

www.redbooks.ibm.com/abstracts/sg247218.html

- Migration de MySQL vers DB2 :

www.redbooks.ibm.com/abstracts/sg247093.html

- Procédures stockées, triggers... avec DB2 for i:

www.redbooks.ibm.com/abstracts/sg246503.html

- Diagnostic de perfs sur DB2 for i :

www.redbooks.ibm.com/abstracts/sg246654.html

A noter que la plupart de ces redbooks ne sont malheureusement plus mis à jour par les équipes d'IBM.

Pour connaître les paramètres de connexion ODBC vers la plupart des bases de données du marché, je vous recommande ce site : www.connectionstrings.com

... avec notamment cette page concernant la connexion à DB2 for i :

<https://www.connectionstrings.com/ibm-i-access-odbc-driver/standard/>

... et celle-ci si vous avez besoin de vous connecter à DB2 LUW :

<https://www.connectionstrings.com/ibm-db2-odbc/>

Pour de plus amples précisions sur le fonctionnement de la classe PHP PDO :

<https://www.php.net/manual/fr/book.pdo.php>

Attention : certaines des techniques de scrapping présentées dans le livre suivant m'ont tiré d'affaire plus d'une fois. Je vous recommande vivement ce livre, qui est disponible chez l'éditeur en PDF :

Webbots, Spiders, and Screen Scrapers, 2nd Edition

A Guide to Developing Internet Agents with PHP/CURL

by Michael Schrenk

NoStarch 2012

<https://nostarch.com/webbots2>

Livre blanc sur l'utilisation de PHP en environnement IBM i

Je n'ai pas fait d'inventaire de la littérature disponible pour PHP 7 et 8 aujourd'hui, d'autant que je travaille beaucoup plus avec Node.js qu'avec PHP actuellement. Mais j'ai conservé dans cette page quelques références anciennes (pour PHP5), car ce sont de très bons ouvrages, inspirants et riches en informations, qui m'ont beaucoup aidé à monter en compétence sur PHP. La plupart de ces livres sont épuisés chez leurs éditeurs respectifs, mais on arrive à les trouver sur le web en version numérique, ou d'occasion en version papier.

- *PHP and MySQL® Web Development, Fourth Edition*
par Luke Welling; Laura Thomson, (Addison-Wesley Professional)
- *PHP 5 in Practice*
par Elliot White III; Jonathan Eisenhamer (Publisher: Sams)
- *PHP Cookbook, 2nd Edition*,
par David Sklar and Adam Trachtenberg (O'Reilly)
- *PHP Hacks*
par Jack D. Herrington (O'Reilly)
- *Object-Oriented PHP*
par Peter Lavin (No Starch Press)

Parmi cette sélection de livres, j'attribuerais une mention spéciale à celui de Jack Herrington (PHP Hacks), qui fournit une excellente introduction aux design patterns, et plein d'exemples inspirants, comme par exemple un générateur de classe PHP à partir de table SQL.

17 Conclusion

PHP a mis du temps à s'imposer dans le monde IBM i, mais aujourd'hui (début 2022), on peut constater que PHP est bien implanté dans de nombreuses sociétés utilisatrices de la plateforme IBM i.

Ce n'était pas gagné pourtant. Durant les années 2000, le PHP avait mauvaise presse dans de nombreuses DSI (Directions des Systèmes d'Informations), qui lui préféraient très souvent le langage Java pour les projets de réécriture d'applications.

Et au début des années 2010, alors que le regard des décideurs sur PHP commençait à changer (en mieux), beaucoup de DSI étaient fortement impactées par les conséquences désastreuses de la crise financière de 2008, et étaient obligées de mettre en sommeil des projets de modernisations applicatives.

Aujourd'hui, la situation est beaucoup plus sereine. Les DSI ont maintenant suffisamment de recul pour apprécier pleinement les qualités de PHP, et elles ont des projets métiers stratégiques qu'elles doivent moderniser. De plus, par le jeu des départs en retraite, elles renouvellent progressivement leurs équipes en intégrant de jeunes recrues qui sont déjà formées aux technos et langages open-source, et qui seront plus enclines à mixer les technologies historiques (RPG, Cobol, etc.) avec des technos plus récentes.

Si PHP a réussi petit à petit à s'imposer face à Java, il est aujourd'hui concurrencé par Node.js (solution apparue en 2009), qui va inévitablement lui ravir des parts de marché. Je ne suis pas inquiet pour PHP, cependant. Je pense que Node.js, arrivé plus récemment dans le monde IBM i, va trouver sa place, et cohabiter pacifiquement avec PHP. Chacune de ces solutions a des avantages et des inconvénients, à charge pour les développeurs de les utiliser à bon escient.

Pour ma part, en ce début d'année 2022, je continue à développer et à maintenir du code PHP avec plaisir, mais pour mes nouveaux projets, je privilégie Node.js quand j'en ai la possibilité. C'est d'abord une affaire de goût, car j'aime beaucoup coder en Javascript et j'aime beaucoup le principe de fonctionnement de Node.js. Mais c'est aussi une question purement pratique. En effet, de plus en plus d'échanges inter-applicatifs se font aujourd'hui avec le format de données JSON... Or la manipulation de données au format JSON est infiniment plus facile en Javascript qu'en PHP.

Mais je vous avoue que l'annonce du projet ZF1 Future (cf. chapitre 15) a fait plus qu'éveiller ma curiosité. Elle m'a carrément donné envie de me ré-intéresser à l'écosystème PHP, et à ZF1 en particulier. Je pense que je ne vais pas tarder à exhumer mes vieux projets ZF1 et à vérifier s'ils fonctionnent correctement avec ZF1 Future.

J'espère que vous aurez pris plaisir à lire ce dossier, et surtout qu'il vous aura apporté des clés pour comprendre les caractéristiques spécifiques de PHP par rapport à l'écosystème IBM i. Si vous êtes amené à reprendre en maintenance du code PHP écrit il y a une dizaine d'années pour l'IBM i, vous serez mieux à même de comprendre les choix de codage et d'architecture faits par vos prédécesseurs.

18 Annexe

18.1 Affichage de la liste des pays avec la clause DB2 OVER

Le script ci-dessous est une variante de listepays.php utilisant l'ordre SQL OVER. Ce script est compatible avec DB2 Express C (version 9 pour LUW), et DB2 pour IBM i en V5R4 ou supérieure :

```
<html>
  <head>
    <title>LISTE DES PAYS</title>
  </head>
  <body>
    <?php
      // configuration de la base de données
      require_once 'config.php' ;
      // chargement des fonctions de pagination
      require_once 'pagination.php' ;

      // récupération des paramètres du $_GET
      if (isset($_GET)) {
        $params = $_GET ;
      } else {
        $params = array() ;
      } ;

      // récupération ou initialisation de l'offset
      $offset = isset($_GET['offset']) ? intval($_GET['offset']) : 1;
      if (!isset($offset)) { $offset = 1; }

      // récupération du critère de recherche par lettre
      if (isset($params['code_alpha'])) {
        $car_de_recherche = trim($params['code_alpha']) ;
      } else {
        $car_de_recherche = '*';
      } ;
      if ($car_de_recherche != '*') {
        $lib_titre = " dont le code commence par " . $car_de_recherche ;
      }

      // récupération du critère de recherche par libellé
      unset ($type_recherche) ;
      if (isset($params['choix_recherche']) && isset($params['nom_recherche'])) {
        $nom_recherche = trim($params['nom_recherche']) ;
        if ($nom_recherche != '') {
          $choix_recherche = trim($params['choix_recherche']) ;
          switch ( $choix_recherche ) {
            case '1': {
              // recherche de type "égal"
              $type_recherche = " = ";
              $lib_titre = " dont le libellé est égal "
                . $car_de_recherche . " . $nom_recherche ;
              break;
            }
            case '2': {
              // recherche de type "contient"
              $type_recherche = " LIKE ";
              $lib_titre = " dont le libellé contient : "
                . $nom_recherche;
              $nom_recherche = '%' . trim($nom_recherche) . '%';
            }
          }
        }
      }
    >
  </body>
</html>
```

```
                break;
            }
            case '3': {
                // recherche de type "commence par"
                $type_recherche = " LIKE ";
                $lib_titre = " dont le libellé; commence par : "
                    . $nom_recherche ;
                $nom_recherche = trim($nom_recherche) . '%';
                break;
            }
            default:{
                $msg_erreur = "Critères de sélection " .
                    "non définis" ;
                error_log($msg_erreur) ;
                die($msg_erreur);
            }
        }
    }
}

// affichage du titre
print "<h2>Liste des pays {$lib_titre}</h2>" ;

// dessiner le début du tableau
echo <<<BOTABLE
    <table border="1" width="0" cellpadding="5">
    <thead>
    <tr>
        <td>CODE FRANCE</td>
        <td>CODE ISO</td>
        <td>DESCRIPTION</td>
    </tr>
    </thead>
    <tbody>

BOTABLE;

// nombre de lignes par page
$nbl_par_page = 15 ;

// préparation des requêtes 1 et 2
// la requête 1 sert au comptage du nombre d'occurrences
$req1 = "SELECT COUNT(*) AS TOTAL FROM LSTPAYS " ;
if (!isset($type_recherche)) {
    $car_de_recherche = $car_de_recherche . '%';
    // recherche sur le code pays
    $req1 = trim($req1) . " WHERE CODFRA LIKE ? " ;
    $val_de_recherche = $car_de_recherche ;
} else {
    // recherche sur le libellé des pays
    $req1 = trim($req1) . " WHERE LIBELLE ".$type_recherche." ? " ;
    $val_de_recherche = $nom_recherche ;
} ;

// la requête 2 sert au chargement de la liste des pays
$req2 = "SELECT * FROM ( " ;
$req2 .= " SELECT CODFRA, CODISO, LIBELLE,
                ROW_NUMBER() OVER (ORDER BY CODFRA ASC) AS RN
                FROM LSTPAYS" ;
if (!isset($type_recherche)) {
    // recherche sur le nom des entités logiques
    $req2 = trim($req2) . " WHERE CODFRA LIKE ? " ;
} else {
    // recherche sur le libellé des entités logiques
    $req2 = trim($req2) . " WHERE LIBELLE ".$type_recherche." ? " ;
} ;
$req2 = trim($req2) . ") AS FOO WHERE RN BETWEEN ? AND ? " ;
```

Livre blanc sur l'utilisation de PHP en environnement IBM i

```
// comptage du nombre de lignes total pour le critère de recherche
// considéré (en vue de calculer le nombre de pages total)
try {
    $requete = $requete1 ;
    $st = $db->prepare($requete);
    $st->execute(array($val_de_recherche));
    $data = $st->fetch(PDO::FETCH_LAZY);
    $total = $data->TOTAL;
} catch (PDOException $e) {
    echo 'Database Problem: ' . $e->getMessage() . '<br/>';
    echo 'Code : ' . $e->getCode() . '<br/>';
    echo 'File : ' . $e->getFile() . '<br/>';
    echo 'Line : ' . $e->getLine() . '<br/>';
    echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
}

// définition des bornes pour la requête DB2
$limit_min = $offset ;
$limit_max = $offset + $nbl_par_page - 1 ;

// récupération du dernier numéro de ligne chargée
$lastRowNumber = $offset - 1;

// Boucle de chargement du tableau utilisant la clause OVER de DB2
try {
    $requete = $requete2 ;
    $st = $db->prepare($requete);
    $st->execute(array($val_de_recherche, $limit_min, $limit_max));
    while ($row_data = $st->fetch(PDO::FETCH_LAZY)) {
        $lastRowNumber++;
        echo <<<LOTABLE
            <tr>
                <td>{$row_data->CODFRA}</td>
                <td>{$row_data->CODISO}</td>
                <td>{$row_data->LIBELLE}</td>
            </tr>
LOTABLE;
    }
} catch (PDOException $e) {
    echo 'Database Problem: ' . $e->getMessage() . '<br/>';
    echo 'Code : ' . $e->getCode() . '<br/>';
    echo 'File : ' . $e->getFile() . '<br/>';
    echo 'Line : ' . $e->getLine() . '<br/>';
    echo 'Trace : ' . $e->getTraceAsString() . '<br/>';
}

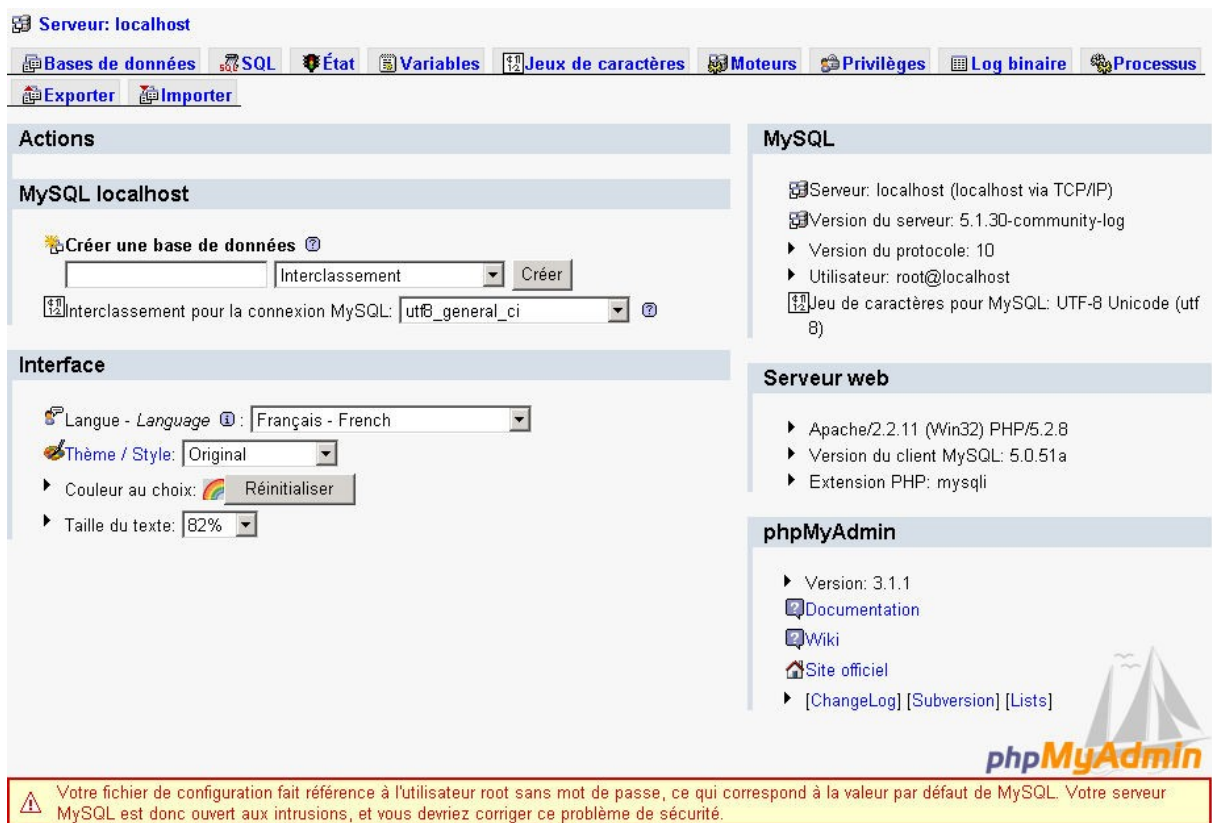
// dessiner la fin du tableau
echo <<<EOTABLE
    </tbody>
</table>
<br/>
EOTABLE;

// récupération de la page courante et appel de la fonction de pagination
$page_en_cours = $_SERVER['PHP_SELF'] ;
pc_indexed_links($total, $offset, $nbl_par_page, $page_en_cours, $params);
print "<br/>";
print "(Affichage $offset - $lastRowNumber sur $total)";
?>

</body>
</html>
```

18.2 Sécurisez votre environnement WAMP ou LAMP


Sur la plupart des stacks PHP pour PC, le profil MySQL par défaut est le profil « root », et dans presque 100 % des cas, ce profil « root » a un mot de passe utilisateur à blanc, ce qui entraîne un risque pour la sécurité de votre machine et donc de votre réseau. Certaines versions de PhpMyAdmin vous avertissent de ce risque par un message en rouge en bas d'écran :



Que vous utilisiez ou pas le profil « root » pour vos connexions à la base de données MySQL, et même si vous n'utilisez que rarement MySQL, au profit d'une autre base de données telle que DB2 par exemple, il est de toute façon impératif de sécuriser le profil « root » en lui attribuant un mot de passe.

Dans la suite de ce chapitre, nous allons voir comment remédier à ce problème.

Commencez par cliquer sur l'onglet « Privileges »

Sur la page suivante, sur la ligne de l'utilisateur « root », cliquez sur l'icône  qui est liée à l'infobulle « changer les privileges » :

The screenshot shows the 'Privileges' tab in phpMyAdmin. At the top, there are navigation tabs: 'Bases de données', 'SQL', 'État', 'Variables', 'Jeux de caractères', 'Moteurs', and 'Privileges'. Below these are 'Exporter' and 'Importer' buttons. The main heading is 'Vue d'ensemble des utilisateurs'. A horizontal menu of letters (A-Z) is visible, with 'R' selected. Below this is a table with columns: 'Utilisateur', 'Serveur', 'Mot de passe', 'Privileges globaux 1', and 'Grant'. The table has one row for the user 'root' on 'localhost', with a password status of 'Non' and 'ALL PRIVILEGES' granted. A checkbox is present in the first column. Below the table, there is a link 'Tout cocher / Tout décocher' and a button 'Changer les privileges'.

Utilisateur	Serveur	Mot de passe	Privileges globaux 1	"Grant"
<input type="checkbox"/> root	localhost	Non	ALL PRIVILEGES	Oui

Dans la page qui suit, descendez jusqu'au pavé qui s'intitule « modifier le mot de passe » :

The screenshot shows the 'Modifier le mot de passe' form. It has two radio buttons: 'aucun mot de passe' and 'Mot de passe:'. The 'Mot de passe:' option is selected. There are two input fields: one for the current password and one for the new password, with a label 'Entrer à nouveau:'. Below these is a section for 'Hachage du mot de passe' with two radio buttons: 'MySQL 4.1+' (selected) and 'compatible MySQL 4.0'. An 'Exécuter' button is at the bottom right.

Saisissez un mot de passe identique dans les 2 cases puis cliquez sur le bouton « Exécuter ».
Petite astuce : si vous manquez d'imagination pour définir le mot de passe, vous pouvez utiliser le bouton « générer un mot de passe » situé dans un autre pavé, un peu plus bas dans la même page :

The screenshot shows the 'Changement des informations de connexion / Copie d'utilisateur' form. It has a section 'Information pour la connexion' with fields for 'Nom d'utilisateur:' (set to 'root'), 'Serveur:' (set to 'localhost'), and 'Mot de passe:'. There are also fields for 'Entrer à nouveau:' and a 'Générer un mot de passe:' section with 'Générer' and 'Copier' buttons. The generated password 'uaxwftMKS9cWdJby' is shown next to the 'Copier' button.

Une fois le mot de passe « généré », vous pouvez le copier-coller dans les 2 cases du pavé de modification du mot de passe présenté plus haut, puis cliquer sur le bouton « Exécuter ».
Le mot de passe généré doit être conservé précieusement car, si vous l'égarez, vous ne pourrez pas le récupérer par la suite (car il est crypté).

Si tout s'est bien passé, phpMyAdmin vous confirmera que la modification du mot de passe a bien été effectuée avec l'affichage suivant :

The screenshot shows a confirmation message in a green box: 'Le mot de passe de 'root'@'localhost' a été changé.' Below this, a SQL statement is displayed: 'SET PASSWORD FOR 'root'@'localhost' = PASSWORD('*****');'.

Il reste une dernière étape à effectuer : mettre à jour le fichier de configuration « config.inc.php » de phpMyAdmin pour qu'il prenne en compte le nouveau mot de passe du profil « root ». En effet, ce profil est utilisé par défaut par le programme phpMyAdmin, donc si vous ne faites pas ce qui suit, vous ne pourrez plus accéder à ce logiciel par la suite.

Le fichier de configuration à modifier se trouve dans :

C:\wamp\apps\phpmyadmin3.1.1\config.inc.php

(bien évidemment, si vous utilisez une version légèrement différente de phpMyAdmin, vous devrez ajuster le chemin d'accès en conséquence).

Ouvrez le fichier de configuration avec votre éditeur préféré, et insérez le mot de passe généré précédemment comme dans l'exemple ci-dessous. Sauvegardez le fichier modifié :

```
1 <?php
2 /*
3  * Generated configuration file
4  * Generated by: phpMyAdmin 3.1.1 setup script by Piotr Przybylski <piotrprz@
5  * Date: Tue, 16 Dec 2008 09:58:49 +0100
6  */
7
8 /* Servers configuration */
9 $i = 0;
10
11 /* Server: localhost [1] */
12 $i++;
13 $cfg['Servers'][$i]['verbose'] = 'localhost';
14 $cfg['Servers'][$i]['host'] = 'localhost';
15 $cfg['Servers'][$i]['port'] = '';
16 $cfg['Servers'][$i]['socket'] = '';
17 $cfg['Servers'][$i]['connect_type'] = 'tcp';
18 $cfg['Servers'][$i]['extension'] = 'mysqli';
19 $cfg['Servers'][$i]['auth_type'] = 'config';
20 $cfg['Servers'][$i]['user'] = 'root';
21 $cfg['Servers'][$i]['password'] = 'uaxwftMKS9cWdJby';
22 $cfg['Servers'][$i]['AllowNoPasswordRoot'] = true;
23
24 /* End of servers configuration */
25
26 $cfg['DefaultLang'] = 'en-utf-8';
27 $cfg['ServerDefault'] = 1;
28 $cfg['UploadDir'] = '';
29 $cfg['SaveDir'] = '';
30
31 ?>
```

Vous pouvez maintenant utiliser ce nouveau mot de passe dans vos propres scripts PHP, si bien évidemment vous utilisez le profil « root » pour vos connexions bases de données. On recommande en général de réserver le profil « root » pour l'administration des bases de données, et d'utiliser des profils dédiés à chaque application, pour l'accès aux bases de données..

18.3 Générer un script SQL à partir d'un fichier CSV

Le script PHP présenté dans ce chapitre a pour but de vous montrer qu'il est très facile de lire un fichier CSV en PHP.

Pour fonctionner correctement, ce script part de l'hypothèse que votre fichier CSV utilise pour ses entête de colonnes les noms des colonnes de la table destinataire. Le début du fichier CSV ressemblerait donc à ceci :

```
CODFRA;CODISO;LIBELLE
AFG;AF;AFGHANISTAN
ZAF;ZA;AFRIQUE DU SUD
ALA;AX;ALAND, ILES
ALB;AL;ALBANIE
...
```

Je rappelle qu'à l'arrivée on doit obtenir ceci :

```
INSERT INTO ( lstpays ) ( CODFRA, CODISO, LIBELLE ) VALUES ( 'AFG', 'AF', 'AFGHANISTAN ' );
INSERT INTO ( lstpays ) ( CODFRA, CODISO, LIBELLE ) VALUES ( 'ZAF', 'ZA', 'AFRIQUE DU SUD ' );

INSERT INTO ( lstpays ) ( CODFRA, CODISO, LIBELLE ) VALUES ( 'ALA', 'AX', 'ALAND, ILES ' );
INSERT INTO ( lstpays ) ( CODFRA, CODISO, LIBELLE ) VALUES ( 'ALB', 'AL', 'ALBANIE ' );
...
```

Voici le code source du script PHP :

```
<?php
function encode( $text ) {
    // on double les apostrophes simples
    $text = preg_replace( "'/'", "'", $text );
    // et on encadre le contenu de $text par des apostrophes de part et d'autre
    return "'".$text."'";
}
$file = fopen('lstpays.csv', 'r'); // fichier CSV indiqué en dur pour l'exemple
$name = 'lstpays'; // nom de la table SQL en dur car non fourni dans le fichier CSV

// en revanche les noms de colonnes sont extraits de la 1ère ligne du fichier CSV
$cols = array();
$i = 0;
while (!feof($file)) {
    $fields = fgetcsv($file, 0, ';');
    $data = array();
    foreach( $fields as $field ) {
        if ($i == 0) {
            $cols []= $field ;
        } else {
            $data []= $field ;
        } ;
    }
    if ($i == 0) {
        $sqlcols = implode( " , ", $cols ) ;
    } else {
        $sqldata = implode( " , ", array_map( "encode", $data ) );
        echo "INSERT INTO ( $name ) ( $sqlcols ) VALUES ( $sqldata ); <br/>";
    }
    $i++;
};
fclose($file);
```

Maintenant que vous connaissez le principe, vous pouvez modifier ce programme pour qu'il alimente directement une table DB2 ou MySQL à partir du fichier CSV. Cela nous donne d'ailleurs un script plus simple que le script précédent :

```
<?php
// paramètres de configuration de la base de données
require_once("config.php");

$reqsql = 'INSERT INTO LSTPAYS (CODFRA, CODISO, LIBELLE) VALUES (?, ?, ?)';

$stmt = $pdo->prepare($reqsql) ;

// compteur destine à "sauter" la 1ère ligne (entête du fichier CSV)
$wi = 0 ;

$fd = fopen('lstpays.csv', 'r');

$wi = 0;
$fields = fgetcsv($fd, 0, ';');
while (!feof($fd)) {
    $data = array() ;
    if ($wi != 0) {
        $stmt->execute(array($fields[0], $fields[1], $fields[2])) ;
    }
    $fields = fgetcsv($fd, 0, ';');
    $wi++ ;
};
fclose($fd);
```

Pour conclure, je vais vous montrer comment générer un fichier CSV avec PHP, à partir d'une table SQL. Pour ce dernier exemple, j'utilise encore une fois mon fichier des pays, que j'extrais cette fois-ci d'une table SQL. Pour chaque ligne de la table SQL je génère une ligne au format CSV dans un fichier que j'ai appelé « paygen.csv ». L'exemple ci-dessous fonctionne aussi bien avec DB2 qu'avec MySQL, il suffit de personnaliser les paramètres de connexion de PDO dans le fichier config.php :

```
<?php
// paramètres de configuration de la base de données
require_once 'config.php' ;

echo 'Generation liste des pays en CSV : <p/>' ;

$sql = 'SELECT CODFRA, CODISO, LIBELLE FROM LSTPAYS ' ;
$file = fopen('payngen.csv', 'w') or die("Impossible d'ouvrir le fichier payngen.csv");

$stmt = $pdo->query($sql);
while($row_data = $stmt->fetch(PDO::FETCH_ASSOC)) {
    if (fputcsv($file, $row_data, ';') === false) {
        die("Can't write CSV line");
    }
}
fclose($file) or die("Impossible de fermer le fichier payngen.csv");
```

18.4 Configuration de PHP sur Laragon

Je l'ai évoqué au début de ce dossier, le projet Laragon est un stack PHP qui a deux caractéristiques intéressantes :

- on peut l'utiliser en version « stand alone » ou en version « portable »
- quelle que soit la version utilisée, on peut y installer plusieurs versions du langage PHP, et swapper très facilement d'une version à l'autre.

Pour installer plusieurs versions de PHP, c'est très facile. Voici les étapes :

Etape 1 : allez sur le site officiel de PHP et téléchargez le Zip de la version qui vous intéresse <https://windows.php.net/download>

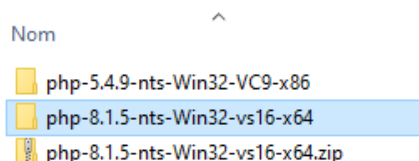
Par exemple, pour télécharger PHP 8.1.5, cliquez sur le lien que j'ai surligné en jaune



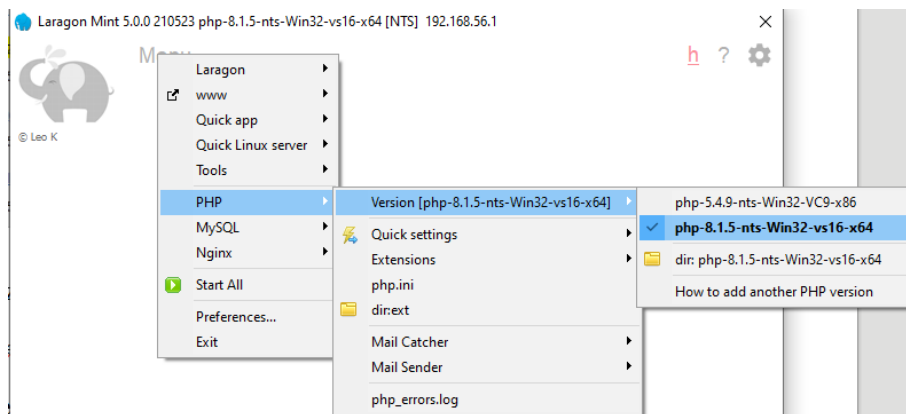
Etape 2 : copiez le fichier zip dans le répertoire suivant de votre instance Laragon (dans mon cas c'est la version « portable ») :

...\laragon-portable\bin\php

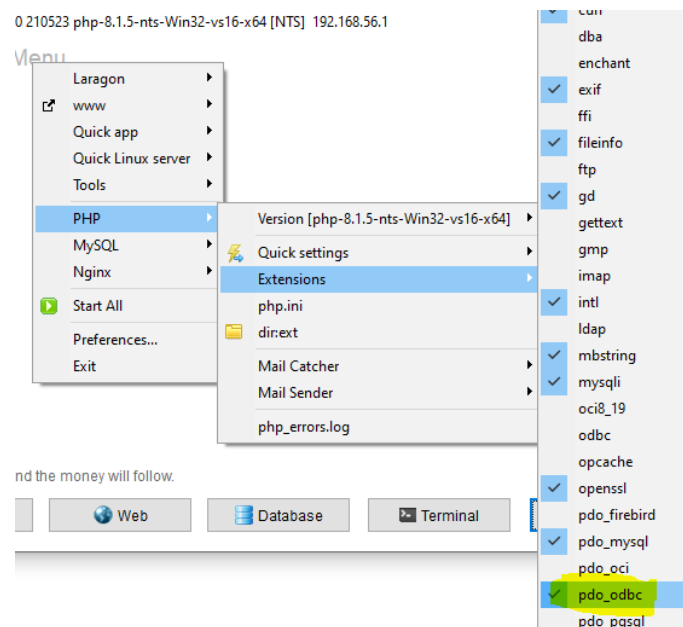
ce qui devrait vous donner ceci, sachant que la 5.4.9 est la version embarquée en standard avec Laragon version portable :



Etape 3 : dans le menu de configuration de Laragon, sélectionnez la version de PHP qui vous intéresse



Etape 4 : activez l'extension « pdo_odbc » pour la version de PHP que vous avez activée :



Etape 5 : cliquez sur le bouton « Start All » et vous pouvez commencer à travailler.