Base de données :compléments PDO, SQLite, Mysqli

J-F Dazy

J-F Berger

PDO

PHP Data Objects

L'extension *PHP Data Objects* (PDO)

interface d'abstraction à l'accès à une base de données :

- **Standard d'accès à des bases de données pour les applications PHP**
  - PDO requiert les nouvelles fonctionnalités OO fournies par PHP 5 et donc, ne fonctionne pas avec les versions antérieures de PHP.
  - PDO est fourni avec PHP 5.1 (extension PECL pour php5.0)
  - Documentation : **http://fr.php.net/pdo/**

- Ainsi une application PHP ne sera plus liée à une seule base (le plus souvent MySQL), mais à PDO, **et donc à n'importe quel SGBD (SQLite , MySQL, PostgreSQL, Oracle, Sybase, SQL Server, ...)**

- **Pourquoi PDO ?**
  - les fonctions PHP d'utilisation des SGBD sont très différentes tant dans leur implémentation que dans leurs noms.
    PDO permet de se connecter à différentes bases de données avec une même fonction, (y compris en émulant certaines fonctions absentes des SGBD les moins fournis) **=> PORTABILITE ACCRUE**

Installation PDO

Il faut ,soit par l'administration php, soit dans php.ini directement

- **Activer dans php.ini extension=php_pdo.dll : l' extension offrant une couche d'abstraction d'accès à des bases de données.**
  - 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 :
- **Activer les extensions spécifiques des BD désirées parmi les disponibles:**
  - extension=php_pdo.dll
  - extension=php_pdo_firebird.dll
  - extension=php_pdo_mssql.dll
  - extension=php_pdo_mysql.dll
  - extension=php_pdo_oci.dll
  - extension=php_pdo_oci8.dll
  - extension=php_pdo_odbc.dll
  - extension=php_pdo_pgsql.dll
  - extension=php_pdo_sqlite.dll

NB : Les extensions disponibles sont visibles dans

- **wamp/bin/php/php2.5.2/ext**
- **easyPHP 2.0/php5/ext** (ne pas y toucher…)

PDO : Installation : php.ini exemple

EXEMPLE **php.ini** (parti de)

```
.....
extension=php_pdo.dll //extension pdo activée
;extension=php_pdo_firebird.dll  //non activée (;)
;extension=php_pdo_mssql.dll     //non activée (;)
extension=php_pdo_mysql.dll  //driver pdo mysql activé
;extension=php_pdo_oci.dll       //non activée (;)
;extension=php_pdo_oci8.dll
;extension=php_pdo_odbc.dll     //non activée (;)
extension=php_pdo_pgsql.dll //driver pdo postgresql activé
;extension=php_pdo_sqlite.dll   //non activée (;)


....
```

Avec cette configuration php ,on pourra utiliser les deux types de bases sous l'interface PDO

PDO utilisation : connexion

Une fois cette configuration faite, on connectera à différentes bases de données avec une même fonction, en ne modifiant que certains arguments : **un changement de SGBD se fait simplement en modifiant l'instanciation de l'objet de la classe PDO !**

- **Connexion :Les connexions sont établies en créant des instances de la classe de base de PDO**

Exemple :

```
// MySQL
    $dbh = new PDO('mysql:host=localhost;dbname=basededonnees',
    identifiant, motdepasse);
// PostgreSQL
    $dbh = new PDO('pgsql:host=localhost port=5432
    dbname=basededonnees user=identifiant password=motdepasse');
```

- **Fermeture de la connexion**

```
    $dbh = NULL;
```

Gestions des exceptions

```
try
 {
    ……code……
   ……..
   ……..
 }
Catch(PDOException $monexception)
   {
        echo 'Error de connection à MySQL!: '.$monexception->getMessage();
         exit()
   }
```

Pour des connexions persistantes, on utilisera

```
<?php
    $dbh = new PDO ('mysql:host=localhost;dbname=basededonnees',
    identifiant,motdepasse, array(  PDO::ATTR_PERSISTENT => true) );
    ?>
```

Transactions et validation automatique

- **PDO gère les transactions avant d'exécuter des requêtes**
- **Les transactions :**
  - 4 fonctionnalités : Atomicité, Consistance, Isolation et Durabilité (ACID).
  - Ne sont pas toujours supportées dans les SGBD
    - **PDO doit s'exécuter en mode "auto-commit" lorsqu'on ouvre pour la première fois la connexion, cad :**
      - toutes les requêtes ont leurs transactions implicites, si la base de données le supporte
      - aucune transaction si la base de données ne les supporte pas.
- **Quand on a besoin d'une transaction:**
  - méthode PDO->beginTransaction() pour l'initialiser

(Si le driver utilisé ne supporte pas les transactions, une exception PDO sera lancée)

  - Et la méthode PDO->commit() (valide) ou PDO->rollBack() (annule) pour la terminer, suivant le succès de votre code durant la transaction.
  - catch (Exception $e) {
    $dbh->rollBack(); echo "Échec : " . $e->getMessage();
    }

Intérêt des requêtes préparées

- **La requête ne doit être analysée (ou préparée) qu'une seule fois, mais peut être exécutée plusieurs fois avec des paramètres identiques ou différents.**
  - **Lorsque la requête est préparée, la base de données va analyser, compiler et optimiser son plan pour exécuter la requête.. En utilisant les requêtes préparées, vous évitez ainsi de répéter le cycle analyser/compilation/optimisation.**
- **Les paramètres pour préparer les requêtes n'ont pas besoin d'être entre guillemets ; le driver le gère pour vous.** Si votre application utilise exclusivement les requêtes préparées, vous pouvez être sûr qu'aucune injection SQL n'est possible
  - **(Cependant, si vous construisez d'autres parties de la requête en vous basant sur des entrées utilisateurs, vous continuez à prendre un risque).**

Requêtes préparées

requête normale :

1 : envoi de la requête par le client vers le serveur
2 : compilation de la requête
3 : plan d'exécution par le serveur
4 : exécution de la requête
5 : résultat du serveur vers le client

requête préparée :

Phase 1 :

1 : envoi de la requête à préparer
2 : compilation de la requête
3 : plan d'exécution par le serveur
4 : stockage de la requête compilée en mémoire
5 : retour d'un identifiant de requête au client

Phase 2 :

1 : le client demande l'exécution de la requête avec l'identifiant
2 : exécution
3 : résultat du serveur au client

En PDO : prepare

Requêtes préparées en PDO :exemple

Utilisation de prepare :

marqueurs ?

```
<?php
    $stmt = $dbh->prepare("INSERT INTO TAB (nom, valeur) VALUES (?, ?)");
    $stmt->bindParam(1, $nom);
    $stmt->bindParam(2, $valeur);
    // insertion d'une ligne
    $nom = 'one';
    $valeur = 1;
    $stmt->execute();
    // insertion d'une autre ligne avec différentes valeurs
    $nom = 'two';
    $valeur = 2;
    $stmt->execute();
    ?>
```

Pour des marqueurs **nommés** ,on aura

```
$stmt = $dbh->prepare ("INSERT INTO TAB (nom, valeur) VALUES (:nom, :valeur)");
$stmt->bindParam(':nom', $nom)
$stmt->bindParam(':valeur', $valeur);
```

Méthode query (1)

- **PDO->query() :**

Exécute une requête SQL, retourne un jeu de résultats en tant qu'objet PDOStatement

- Pour une requête dont on a besoin plusieurs fois,
  - **préparer un objet PDOStatement avec la fonction PDO->prepare()**
  - **exécuter la requête avec la fonction PDOStatement->execute().**
  - **Il faudra lier les variables PHP aux marqueurs de positionnement soit par PDOStatement->bindParam() , soit par un tableau, en paramètre de PDOStatement->execute().**

- Si on ne récupère pas toutes les données du jeux de résultats avant d'exécuter le prochain appel à PDO->query(), l'appel peut échouer.
  - Il faut appeler PDOStatement->closeCursor() pour libérer les ressources associées à l'objet PDOStatement avant d'exécuter le prochain appel PDO->query().

PDOStatement->fetch()

- **Récupère la ligne suivante d'un jeu de résultat PDO,**

Liste de paramètres

- *fetch_style* détermine la façon dont PDO retourne la ligne.
  - ***PDO::FETCH_ASSOC*: tableau indexé par le nom de la colonne**
  - ***PDO_FETCH_BOTH* (défaut): retourne un tableau indexé par les noms de colonnes mais aussi par les numéros de colonnes (commençant à l'indice 0),**
  - ***PDO::FETCH_BOUND*: retourne** TRUE **et assigne les valeurs des colonnes de votre jeu de résultats dans les variables PHP à laquelles elles sont liées avec la méthode PDOStatement->bindParam()**
  - ***PDO::FETCH_LAZY*: combine *PDO::FETCH_BOTH* et *PDO::FETCH_OBJ*, créant ainsi les noms des variables de l'objet, comme elles sont accédées**
  - ***PDO::FETCH_NUM*: retourne un tableau indexé par le numéro de la colonne comme elle est retourné dans votre jeu de résultat, commençant à 0**
  - ***PDO_FETCH_OBJ*: retourne un objet anonyme avec les noms de propriétés qui correspondent aux noms des colonnes retournés dans le jeu de résultats**
  - **…**
- *cursor_orientation* quelle ligne sera retournée à l'appelant. Par défaut **PDO::FETCH_ORI_NEXT.**
- *Offset* : pour un objet PDOStatement représentant un curseur scrollable pour lequel le paramètre *cursor_orientation* est défini à *PDO::FETCH_ORI_ABS*, cette valeur spécifie le numéro absolu de la ligne

Exemple query (2)

```
$sql = "SELECT * FROM personnes";
      foreach ($dbh->query($sql) as $row)
          {
          print $row[‘nom] .' - '. $row[‘prenom] . '<br />';
          }
```

Le *fetch_style* est celui par défaut (tableau indexé par les noms de colonnes mais aussi par les numéros de colonnes)

PDOStatement->fetchAll()

Retourne un tableau contenant toutes les lignes du jeu d'enregistrements

- ***Paramètre***
  - *: fetch_style* détermine la façon dont PDO retourne les lignes
  - *column_index* [optionnel] pour n'obtenir qu'une seule colonne

Exemple :

```
<?php
    $sth = $dbh->prepare("SELECT nom, couleur FROM fruit");
    $sth->execute();

    /* Récupération de toutes les lignes d'un jeu de résultats */
    $result = $sth->fetchAll();
    print_r($result);
    ?>
```

Méthode exec

PDO->exec(statement) : Exécute une requête SQL et retourne le nombre de lignes affectées

- **Liste de paramètres**
  - *statement* : **requête à préparer et à exécuter.**

- **Exemple:**

```
$delrows=$dbh->exec('DELETE FROM tab WHERE id<20');
echo ‘nombre d’enregistrements supprimés :'.$delrows;
```

PDOStatement->execute()

- **Exécute une requête préparée.**

Si la requête préparée inclut des marqueurs de positionnement, on peut :

- appeler la fonction PDOStatement->bindParam() pour lier les variables PHP aux marqueurs de positionnement :
- ou passer un tableau de valeurs de paramètres, uniquement en entrée

Exemple

```
<?php
    /* Exécute une requête préparée en passant un tableau de valeurs */
    $calories = 150;
    $colour = 'rouge';
    $sth = $dbh->prepare('SELECT nom, couleur, calories
        FROM fruit
        WHERE calories < ? AND couleur = ?');
    $sth->execute(array($calories, $couleur));
```

SQLite

SQLite : un moteur SQL simplifié

- **petite bibliothèque écrite en C : un moteur de base de données SQL (partie de SQL92) et les propriétés ACID,**
  - dans la distribution de base depuis la version 5 de php
- **SQLite n'est pas client/serveur :**
  - pas de procédure d'installation , de configuration, de gestion de comptes et de droits utilisateurs.
  - toute la base est stockée dans un seul fichier (directement sur le disque dur du serveur www).
  - le type de chaque donnée stockée en base est une propriété de la donnée, pas de la colonne. Une colonne peut donc contenir des données de type différent.
  - Adapté aux petits sites, applications embarquées, mais non conçu pour gérer de nombreux accès simultanés
  - Intégré dans mozilla firefox, skype, mac, téléphones symbian et apple (sans doute plus de 300 millions de déploiement au total)
- **D. Richard Hipp, le créateur de SQLite, a choisi de distribuer cette bibliothèque dans le domaine public.**

SQLite et php5

Voir aussi par exemple : http://fr2.php.net/SQLite/

- Depuis PHP 5.1.0, SQLite (3) dépend de PDO, alors PDO doit être activé, en ajoutant à *php.ini* (dans l'ordre) :
  - **extension=php_pdo.dll**
  - **extension=php_pdo_sqlite.dll**
- SQLite est embarqué et compilé par défaut, ce qui signifie que le simple fait d'installer PHP5 permet de l'utiliser.

Donc pas de processus indépendant comme MySQLserver

SQLite 2 ou SQLite3? création de base

- **SQLite2**

```
$dbhandle = new SQLiteDatabase('/www/support/usersqlite2.db');
```

  - il suffit de donner un nom de fichier (création BD ou ouverture) sous réserve des droits lecture/écriture sur le répertoire
  - On peut aussi créer la base en mémoire vive destination :memory

- **SQLite3** (cf. exposé PHP 5 et PDO)

```
try {
   $dbHandle = new
  PDO('sqlite:'.$_SERVER['DOCUMENT_ROOT'].'/../usersqlite3.db');
   }
catch( PDOException $exception ){
    die($exception->getMessage());
}
```

SqLite interface OO

```
// connection à une db ou création si elle n'existe pas
$db = new SQLiteDatabase("db.sqlite");
//transaction implicite
$db->query("CREATE TABLE foo(id INTEGER PRIMARY KEY, name
    UNIQUE)");
//transaction
$db->query("BEGIN;
   INSERT INTO foo (name) VALUES('machin');
   INSERT INTO foo (name) VALUES('truc');
   INSERT INTO foo (name) VALUES('chouette');
   COMMIT;");
$result = $db->query("SELECT * FROM foo"); // de classe 'SQLiteResult'

while ($result->valid()) { // reste-t-il des lignes ?
   $row = $result->current();
    print_r($row); echo "<p />";
   $result->next();  }

unset($db); // détruire la connexion (pas très utile en PHP...)
```

SQLite en procédural

```
// connection à une db ou création si elle n'existe pas
$db = sqlite_open("db.sqlite");
//transaction implicite
sqlite_query($db ,
     "CREATE TABLE foo(id INTEGER PRIMARY KEY, name UNIQUE)");
//transaction
sqlite_query("BEGIN;
    INSERT INTO foo (name) VALUES('machin1');
    INSERT INTO foo (name) VALUES('truc1');
    INSERT INTO foo (name) VALUES('chouette1');
    COMMIT;");

$result = sqlite_query($db,"SELECT * FROM foo");
while ($row = sqlite_fetch_array($result)) {
    print_r($row); echo "<p />";
}

sqlite_close($db);
```

SQLite: typage

Par new SQLiteDatabase("db.sqlite"); ou sqlite_open("db.sqlite");

Création de la db "db.sqlite" dans le répertoire du script qui exécute cette instruction dans UN seul fichier.

Rappel :

SQLite est très faiblement typée, toutes les données sont enregistrées comme des chaînes terminées par 'NULL'.

Pas de type pour les colonnes/champs. Pour des raisons de compatibilité SQLite 'supporte' les spécifications INT, CHAR, FLOAT, TEXT, etc dans les déclarations de table mais ne s'en sert pas.

SQLite fait seulement la distinction entre les strings et les integers pour les tris. Donc si vous n'avez pas l'intention de trier des données vous pouvez vous dispenser de typer les colonnes dans les instructions CREATE TABLE !!!

SQLite's typeless nature makes sorting and comparing data somewhat slow !!!

SQLite : INTEGER PRIMARY KEY

L'habitude dans les tables SQL d'avoir un premier champ numérique clé primaire en "AUTOINCREMENT" (pour faciliter des recherches ou l'accès à la dernière ligne) a en SQLite une syntaxe particulière :

INTEGER PRIMARY KEY.

(c'est tout...)

ATTENTION :

AUTOINCREMENT provoque une erreur de syntaxe

Autre erreur de syntaxe : IF NOT EXISTS

SQLite : requêtes chaînées , transactions

```
$db = new SQLiteDatabase(":memory:");
// transaction
$db->query("BEGIN;
   CREATE TABLE bar ( id INTEGER PRIMARY KEY, id2 );
   INSERT INTO bar (id2) VALUES('v1');
   INSERT INTO bar (id2) VALUES('v2');
  COMMIT;");

// requête chaînées
$db->query("INSERT INTO bar (id2) VALUES('v3');
        INSERT INTO bar (id2) VALUES('v4');
        INSERT INTO bar (id2) VALUES('v5');
        INSERT INTO bar (id2) VALUES('v6');");
```

SQLite : class SQLiteDatabase

```
{
__construct ( string filename [, int mode [, string &error_message]] );

SQLiteResult query ( string query [, int result_type [, string &error_msg]] );
bool queryExec ( string query [, string &error_msg] );
array arrayQuery ( string query [, int result_type [, bool decode_binary]] );
array singleQuery ( string query [, bool first_row_only [, bool decode_binary]] );

SQLiteUnbuffered unbufferedQuery ( string query [, int result_type
                                                    [, string &error_msg]] );
int lastInsertRowid ( );
int changes ( );
void createAggregate ( string function_name, callback step_func,
                                          callback finalize_func [, intnum_args] );
void createFunction ( string function_name, callback callback [, int num_args] );
void busyTimeout ( int milliseconds );
int lastError ( );
array fetchColumnTypes ( string table_name [, int result_type] )
}
```

SQLite : class SQLiteDatabase(1)

- **__construct** (string filename [, int mode [, string &error_message]]);

Ouvre une db SQLite ou la crée si elle n'existe pas.

Paramètres :

• filename : le nom du fichier qui contient la db

• mode : mode d'accès (unix) au fichier de la db. (pour le moment NON utilisé par SQLite). Le mode par défaut est la valeur octale 0666.

• error_message : Passé par référence pour proposer un message d'erreur expliquant pourquoi la db n'a pu être ouverte.

•Return Values : Retourne une ressource (database handle) en cas de succès, FALSE en cas d'erreur

SQLite : class SQLiteDatabase(2)

SQLiteResult query(string query [, int result_type [, string &error_msg]]);

Exécute une requête SQL donné par query sur la connexion à la db (database handle).

Remarque : la connexion n'est pas demandé dans l'interface OO.

result_type : **SQLITE_ASSOC** , **SQLITE_NUM , SQLITE_BOTH** (par défaut)

SQLiteUnbuffered unbufferedQuery (string query [, int result_type [, string &error_msg]]);

identique à **query**( ) mais le résultat peut seulement être utilisé en lisant une ligne après l'autre; idéale pour la génération de tables HTML où on traite le résultat ligne par ligne.

bool **queryExec (string query [, string &error_msg]);**

exécuter une requête qui ne retourne rien.

array **arrayQuery (string query [, int result_type [, bool decode_binary]]);**

Exécute une requête et retourne les résultats dans un tableau

array **singleQuery (string query [, bool first_row_only [, bool decode_binary]]);**

Exécute une requête et retourne soit un tableau pour une seule colonne, soit la valeur de la première ligne

SQLite : class SQLiteResult (3)

```
{
array fetch ( [int result_type [, bool decode_binary]] );
object fetchObject ( [string class_name [, array ctor_params
                                                [, bool decode_binary]]] );
string fetchSingle ( [bool decode_binary] );
array fetchAll ( [int result_type [, bool decode_binary]] );
mixed column ( mixed index_or_name [, bool decode_binary] );
int numFields ( );
string fieldName ( int field_index );
array current ( [int result_type [, bool decode_binary]] );
int key ( );
bool next ( );
bool valid ( );
bool rewind ( );
bool prev ( );
bool hasPrev ( );
int numRows ( );
bool seek ( int rownum );
}
```

SQLite : class SQLiteUnbuffered

```
{
array fetch ( [int result_type [, bool decode_binary]] );
object fetchObject ( [string class_name [, array ctor_params
                                    [, bool decode_binary]]] );
string fetchSingle ( [bool decode_binary] );
array fetchAll ( [int result_type [, bool decode_binary]] );
mixed column ( mixed index_or_name [, bool decode_binary] );
int numFields ( );
string fieldName ( int field_index );
array current ( [int result_type [, bool decode_binary]] );

bool next ( );
bool valid ( );
}
```

SQLite : inconvénients, problèmes

ATTENTION : pendant la mise à jour de la BD en ajoutant une nouvelle donnée SQLite doit verrouiller le fichier qui contient la BD jusqu'à ce que la modification soit terminée…

- **sqlite_escape_string()**

ATTENTION : utiliser sqlite_escape_string( ) pour les problèmes de simple quote et autres caractères spéciaux. Sinon une chaine comme abc'123 provoquera une erreur syntaxique.
Ne pas utiliser addslashes( ) à la place de sqlite_escape_string( ), car les deux fonctions ne sont pas équivalentes.

- **Exemple :**

```
$username = sqlite_escape_string($username);
$password = sqlite_escape_string($password);
sqlite_query($db, "INSERT INTO users VALUES
                        ('$username', '$password')");
```

SQLite : sqlite_fetch_array()

```
$r = sqlite_query($db, 'SELECT username FROM users');
while ($row = sqlite_fetch_array($r)) {
// do something with $row
}
```

Par défaut, sqlite_fetch_array( ) retourne un tableau dont les éléments sont indexés numériquement **ET** comme un tableau associatif .

Par exemple

si la requête ci dessus retourne une ligne avec le username rasmus, on obtient :

```
Array (
[0] => rasmus
[username] => rasmus
)
```

Développement Web (2) NFA017

PHP5 : extension mysqli

J-F Dazy

J-F Berger

Mysqli : Extension MySQL "améliorée"

Voir aussi par exemple :http://fr3.php.net/manual/ref.mysql.html

Installation : Pour installer l'extension mysqli dans WAMP, Activer l'extension php_mysqli dans le menu 'PHP extensions' et désactivez aussi l'extension standard php_mysql

Options de configuration MySQLi : mysqli.max_links , mysqli.default_port , mysqli.default_socket , mysqli.default_host , mysqli.default_user , mysqli.default_pw cf. le fichier 'php.ini' dans wamp

Classes pré-définies : Mysqli

mysqli : représente la connexion entre php et le serveur MySQL
Contructeur :

Style orienté objet (méthode)
class **mysqli** {

__construct ([string host [, string username [, string passwd [, string dbname [, int port [, string socket]]]]]])
…
}

Style procédural
mysqli **mysqli_connect** ([string host [, string username [, string passwd [, string dbname [, int port [, string socket]]]]]])

Mysqli : connexion 'style Objet'

```
<?php
$mysqli = new mysqli("localhost", "root", "", "world");

/* Vérification de la connexion */
if (mysqli_connect_errno()) {
   echo "Echec de la connexion: ".mysqli_connect_error()."<br/>";
   exit();
}
 echo " connexion OK …<br/>";
echo "Information sur le serveur : ".$mysqli->host_info ."<br/>";

/* Fermeture de la connexion */
$mysqli->close();
?>
```

Mysqli : connexion 'style procedurale'

```
// connexion à une db
<?php
$link = mysqli_connect("localhost", "root", "", "world");

/* Vérification de la connexion */
if (!$link) {
   printf("Echec de la connexion : %s\n", mysqli_connect_error());
   exit();
}

printf("Information sur le serveur : %s\n", mysqli_get_host_info($link));

/* Fermeture de la connexion */
mysqli_close($link);
?>
```

Mysqli : requête

Style orienté objet (méthode)

class **mysqli** {

mixed **query** (string query [, int resultmode])

}

Style procédural

mixed **mysqli_query** (mysqli link, string query [, int resultmode])

Mysqli : requête 'style Objet'

```
<?php
$mysqli = new mysqli("localhost", "root", "", "world");
/* Vérification de la connexion …*/
/* Requète "Select" retourne un objet de type mysqli */
if ($result = $mysqli->query("SELECT Name FROM City LIMIT 10")) {
   printf("Select a retourné %d lignes.\n", $result->num_rows);
   /* Libération du jeu de résultats */
   $result->close();
}
/* Si beaucoup de données, utiliser MYSQLI_USE_RESULT */
if ($result = $mysqli->query("SELECT * FROM City",
MYSQLI_USE_RESULT)) {
…   $result->close();
}

$mysqli->close();
?>
```

Mysqli : requête 'style procédural'

```
<?php
$mysqli = mysqli_connect("localhost", "root", "", "world");
/* Vérification de la connexion …*/
/* Requète "Select" retourne un objet de mysqli */
if ($result = mysqli_query($mysqli ,
                "SELECT Name FROM City LIMIT 10")) {
   printf("Select a retourné %d lignes.<br/>", mysqli_num_rows);
   /* Libération du jeu de résultats */
   mysqli_free_result($result);
}
…
mysqli_close();
?>
```

Classes pré-définies : mysqli_result

Représente le résultat retourné par le serveur.

Quelques Méthodes :

close - Termine le jeu de résultat MySQL
data_seek - Déplace le pointeur de lignes de résultat
fetch_field - Lit les informations de colonnes dans un résultat
fetch_fields - Lit les informations de toutes les colonnes d'un résultat
fetch_field_direct - Lit les informations de colonne
fetch_array - Lit une ligne de résultat dans un tableau associatif ou numérique

fetch_assoc - Lit une ligne de résultat dans un tableau associatif
fetch_object - Lit une ligne de résultat dans un objet (construit à la volée !!!)

fetch_row - Lit une ligne de résultat dans un tableau (indices numériques)
fetch_seek - Place le pointeur de résultat à un offset valide

mysqli_result : fetch_assoc

Style procédural

array **mysqli_fetch_assoc** (mysqli_result result)

Style orienté objet (méthode)

class **mysqli_result** {

array **fetch_assoc** (void)

}

mysqli_result : fetch_assoc *exemple*

```
$query = "SELECT Nom, CodePays FROM Ville ORDER by
ID DESC LIMIT 50,5";

if ($result = $mysqli->query($query)) {

   /* Tableau associatif de la ligne sélectionnée */
   while ($row = $result->fetch_assoc()) {
       printf ("%s (%s)\n", $row["Nom"], $row["CodePays"]);
   }

   /* Libération du jeu de résultats */
   $result->close();
}
```

mysqli_result : fetch_object

Style orienté objet (méthode)

```
class mysqli_result {

    mixed fetch_object ( void )

}
```

Style procédural

```
mixed mysqli_fetch_object ( mysqli_result result )
```

mysqli_result : fetch_object exemple

```
$query = "SELECT Nom, CodePays FROM Ville ORDER
by ID DESC LIMIT 50,5";

if ($result = $mysqli->query($query)) {

   /* Récupération du tableau d'objet */
   while ($obj = $result->fetch_object()) {
       printf ("%s (%s)\n", $obj->Nom, $obj->CodePays);
   }

   /* Libération du jeu de résultats */
   $result->close();
}
```