16

Accès objet à MySQL avec PHP

Une tendance évidente qui a prévalu dans le développement de PHP 5 est la programmation objet ; l'accès à MySQL n'y a pas échappé, et cela n'a fait que s'accentuer dans les versions successives de PHP 5. Cette possibilité est liée à l'utilisation de l'extension `mysqli`, dite « mysql améliorée », qui comprend les trois classes suivantes :

- La classe `mysqli` qui permet de créer des objets de type `mysqli_object`. Cette dernière possède pas moins de 33 méthodes et 15 propriétés pouvant remplacer ou compléter les fonctions de l'extension `mysql` comme la connexion à la base, l'envoi de requêtes, les transactions ou les requêtes préparées.
- La classe `mysqli_result`, permettant de gérer les résultats d'une requête SQL effectuée par l'objet précédent. Ses méthodes et propriétés sont également les équivalents des fonctions de l'extension `mysql` vues au chapitre 15.
- La classe `mysql_stmt` qui représente une requête préparée. Il s'agit là d'une nouveauté par rapport à l'extension `mysql`.

Nous allons maintenant reprendre des différents exemples du chapitre 15 et voir comment obtenir les mêmes résultats via un accès objet. Ce chapitre peut donc être abordé indépendamment du chapitre précédent.

Connexion au serveur MySQL

Comme il se doit, la première chose à faire est de se connecter au serveur MySQL. Pour cela nous créons un objet de la classe `mysqli` selon la syntaxe suivante :

```
$idcom = new mysqli (string $host, string $user, string $pass [,string $base]) ;
```

`$idcom` est un objet `mysqli` et `$host`, `$user` et `$pass` sont, comme dans le chapitre 15, le nom du serveur MySQL, le nom de l'utilisateur et le mot de passe. Le paramètre facultatif `$base` permet de choisir d'emblée la base sur laquelle seront effectuées les commandes SQL.

Nous pouvons également réutiliser le fichier `myparam.inc.php`, créé au chapitre 15, contentant les paramètres de connexions (ci-dessous en local) :

Exemple 16-1. Le fichier *myparam.inc.php*

```
<?php
define("MYHOST","localhost");
define("MYUSER","root");
define("MYPASS","");
?>
```

L'objet `$idcom` représentant la connexion sera utilisé directement ou indirectement pour toutes les opérations à effectuer sur la base. Si la connexion n'est pas effectuée, la variable `$idcom` contiendra la valeur `FALSE`, permettant ainsi de tester si la connexion est bien réalisée.

La connexion prend fin quand l'exécution du script PHP est terminée, mais on peut y mettre fin explicitement pour observer le serveur MySQL. Si les résultats d'une requête sont entièrement récupérés, la connexion peut en effet être coupée avec la méthode `close()` selon la syntaxe suivante :

```
boolean $idcom->close()
```

Si le serveur comporte plusieurs bases de données et que le paramètre `$base` a été employé lors de la création de l'objet `$idcom`, il est possible de changer de base sans interrompre la connexion en cours en appelant la méthode `select_db()` avec la syntaxe suivante :

```
boolean $idcom->select_db (string $base)
```

Cette méthode retourne un booléen qui permet de tester la bonne fin de l'opération. Si le paramètre `$base` n'a pas été précisé lors de la création de l'objet `mysqli`, c'est cette méthode qui permet de choisir la base.

Sélection de la base via SQL

Comme nous l'avons vu au chapitre 15, vous pouvez aussi sélectionner une base en envoyant la requête `"USE nom_base"` au serveur à l'aide de la méthode `query()` détaillée dans les sections suivantes.

En cours de script, vous pouvez à tout moment tester si la connexion est encore active en appelant la méthode `ping()` selon la syntaxe suivante :

```
boolean $idcom->ping()
```

Celle-ci renvoie `TRUE` si la connexion est active, sinon elle renvoie `FALSE` et effectue une reconnexion avec les paramètres initiaux.

La structure d'un script accédant à MySQL est donc la suivante :

```
<?php
//Inclusion des paramètres de connexion
include_once("myparam.inc.php");
//Connexion au serveur
$idcom = new mysqli(MYHOST,MYUSER,MYPASS,"ma_base");
//Affichage d'un message en cas d'erreurs
if(!$idcom)
{
    echo "<script type=text/javascript>";
    echo "alert('Connexion Impossible à la base)</script>";
}
//*********************************
//Requêtes SQL sur la base choisie
//Lecture des résultats
//*********************************
//Fermeture de la connexion
$idcom->close();
?>
```

Comme nous l'avons fait pour l'accès procédural à MySQL, nous avons intérêt à créer une fonction de connexion au serveur que nous réutiliserons systématiquement dans tous les exemples qui suivent. C'est l'objet de l'exemple 16-2 qui crée la fonction `connexobjet()` dont les paramètres sont le nom de la base dans la variable `$base` et le nom du fichier `.inc.php` contenant les paramètres de connexion dans la variable `$param`.

La fonction inclut d'abord les paramètres de connexion (repère ❶) puis crée un objet `mysqli` (repère ❷) ; elle vérifie ensuite que la connexion est bien réalisée (repère ❸), affiche un message d'alerte JavaScript et sort de la fonction en cas de problème (repère ❹). Si la connexion est bien réalisée elle retourne l'objet `$idcom` (repère ❺).

☛ Exemple 16-2. Fonction de connexion au serveur

```
<?php
function connexobjet($base,$param)
{
    include_once($param.".inc.php"); ←❶
    $idcom = new mysqli(MYHOST,MYUSER,MYPASS,$base);  ←❷
    if (!$idcom)  ←❸
    {
        echo "<script type=text/javascript>";
        echo "alert('Connexion Impossible à la base')</script>";
        exit(); ←❹
    }
    return $idcom;  ←❺
}
?>
```

Chacun de vos scripts d'accès à la base doit donc contenir les lignes suivantes :

```
include("connexobjet.inc.php");
$idcom = connexobjet ("nom_base","myparam ");
```

L'opération de connexion est donc gérée en deux lignes.

Envoi de requêtes SQL au serveur

Les différentes opérations à réaliser sur la base MySQL impliquent l'envoi de requêtes SQL au serveur.

Pour envoyer une requête, il faut employer la méthode `query()` de l'objet `mysqli` dont la syntaxe est :

```
divers $idcom->query (string $requete [,int mode ])
```

La requête est soit directement une chaîne de caractères, soit une variable de même type. Le paramètre `mode` est une constante qui prend la valeur `MYSQLI_USE_RESULT` ou `MYSQLI_STORE_RESULT`, cette dernière étant la valeur par défaut. Avec `MYSQLI_STORE_RESULT`, il est possible d'envoyer plusieurs requêtes sans libérer la mémoire associée à un premier résultat, tandis qu'avec l'autre méthode, il faut d'abord libérer cette mémoire à l'aide de la méthode `free_result()` appliquée à l'objet `$result` (de type `mysqli_result`).

La méthode `query()` retourne `TRUE` en cas de réussite et `FALSE` sinon et un objet de type `mysqli_result` pour les commandes SQL de sélection comme `SELECT`. Nous pouvons donc tester la bonne exécution d'une requête. En résumé, un script d'envoi de requête a la forme suivante :

Exemple 16-3 Envoi de requête type

```
<?php
include_once("connexobjet.inc.php"); ←❶
$idcom=connexobjet("magasin","myparam"); ←❷
$requete="SELECT * FROM article ORDER BY categorie"; ←❸
$result=$idcom->query($requete); ←❹
if(!$result)
{
  echo "Lecture impossible"; ←❺
}
else
{
  //Lecture des résultats ←❻
  while ($row = $result->fetch_array(MYSQLI_NUM))
  {
   foreach($row as $donn)
   {
     echo $donn," ";
   }
   echo "<hr />";
```

```
    }
    //Destruction de l'objet $result
    $result->close();
}
// Fermeture de la connexion
$idcom->close(); ←❼
?>
```

Ce script effectue successivement l'inclusion du fichier `connexobjet.inc.php` (repère ❶), la connexion au serveur (repère ❷), l'écriture de la requête SQL dans la variable `$requete` (repère ❸), l'envoi de la requête et la récupération du résultat (repère ❹) puis l'affichage d'un message d'erreur éventuel (repère ❺) ou bien des résultats, procédure que nous détaillerons dans le paragraphe suivant (repère ❻) et, enfin, la libération de la mémoire occupée par l'objet `mysqli_result` (repère ❼).

Lecture du résultat d'une requête

Pour les opérations d'insertion, de suppression ou de mise à jour de données dans une base, il est simplement utile de vérifier si la requête a bien été exécutée.

Par contre, lorsqu'il s'agit de lire le résultat d'une requête contenant la commande `SELECT`, la méthode `query()` retourne un objet de type `mysqli_result`, identifié dans nos exemples par la variable `$result`. La classe `mysqli_result` offre une grande variété de méthodes permettant de récupérer des données sous des formes diverses, la plus courante étant un tableau. Chacune de ces méthodes ne récupérant qu'une ligne du tableau à la fois, il faut recourir à une ou plusieurs boucles pour lire l'ensemble des données.

Lecture à l'aide d'un tableau

La méthode des objets instances de la classe `mysqli_result` la plus perfectionnée pour lire des données dans un tableau est `fetch_array()`, dont la syntaxe est :

```
array $result->fetch_array (int type)
```

Elle retourne un tableau qui peut être indicé (si la constante `type` vaut `MYSQLI_NUM`), associatif (si `type` vaut `MYSQLI_ASSOC`), dont les clés sont les noms des colonnes ou les alias de la table interrogée, ou encore mixte, contenant à la fois les indices et les clés (si `type` vaut `MYSQLI_BOTH`). Pour lire toutes les lignes du résultat, il faut écrire une boucle (`while` par exemple) qui effectue un nouvel appel de la méthode `fetch_array()` pour chaque ligne. Cette boucle teste s'il existe encore des lignes à lire, la méthode `fetch_array()` retournant la valeur `NULL` quand il n'y en a plus. Pour lire et afficher chaque ligne nous utilisons ensuite une boucle `foreach`. Notez encore une fois que si le tableau retourné est indicé, l'indice 0 correspond au premier attribut écrit dans la requête et ainsi de suite.

Les méthodes suivantes permettent également de récupérer une ligne de résultat à la fois :

```
array $result->fetch_assoc(void)
```

retourne un tableau associatif dont les clés sont les noms des colonnes de la table.

```
array $result->fetch_row(void)
```

donc les indices de 0 à N sont les positions des attributs dans la requête SQL.

L'exemple 16-4 met cette méthode en pratique dans le but d'afficher le contenu de la table `article` de la base `magasin` dans un tableau XHTML. Après l'inclusion de la fonction de connexion (repère ❶) puis son appel sur la base `magasin` (repère ❷) nous écrivons la requête SQL de sélection (repère ❸). L'envoi de la requête avec la méthode `query()` permet de récupérer un objet `$result` ou `FALSE` en cas d'échec (repère ❹). Un test sur la variable `$result` (repère ❺) permet d'afficher un message d'erreur (repère ❻) ou le contenu de la table (repère ❼). Nous récupérons d'abord le nombre d'articles dans la table grâce à la propriété `num_rows` de l'objet `mysqli_result` (repère ❽) et affichons ce nombre dans un titre `<h4>` (repère ❾). Une boucle `while` permet de lire une ligne à la fois dans un tableau indicé (repère ❿), puis une boucle `foreach` permet d'afficher chacune des valeurs du tableau dans un tableau XHTML (repère ⓫). L'objet `$result` est alors supprimé (repère ⓬) et la connexion fermée (repère ⓭). La figure 16-1 montre l'affichage obtenu dans un navigateur.

Exemple 16-4. Lecture de la table article

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Lecture de la table article</title>
  <style type="text/css" >
  table {border-style:double;border-width: 3px;border-color:red;
  ➥background-color: yellow;}
  </style>
 </head>
 <body>
<?php
include("connexobjet.inc.php"); ←❶
$idcom=connexobjet("magasin","myparam"); ←❷
$requete="SELECT * FROM article ORDER BY categorie"; ←❸
$result=$idcom->query($requete); ←❹
if(!$result) ←❺
{
  echo "Lecture impossible"; ←❻
}
else ←❼
{
  $nbcol=$result->field_count;
  $nbart=$result->num_rows; ←❽
  echo "<h3> Tous nos articles par cat&#233;gorie</h3>";
  echo "<h4> Il y a $nbart articles en magasin </h4>"; ←❾
```

```
  echo "<table border=\"1\">";
  echo "<tr><th>Code article</th> <th>Description</th> <th>Prix</th>
  ➥<th>Cat&#233;gorie</th></tr>";
  while($ligne=$result->fetch_array(MYSQLI_NUM)) ←⑩
  {
    echo "<tr>";
    foreach($ligne as $valeur) ←⑪
    {
      echo "<td> $valeur </td>";
    }
    echo "</tr>";
  }
  echo "</table>";
}
$result->close(); ←⑫
$idcom->close(); ←⑬
?>
</body>
</html>
```

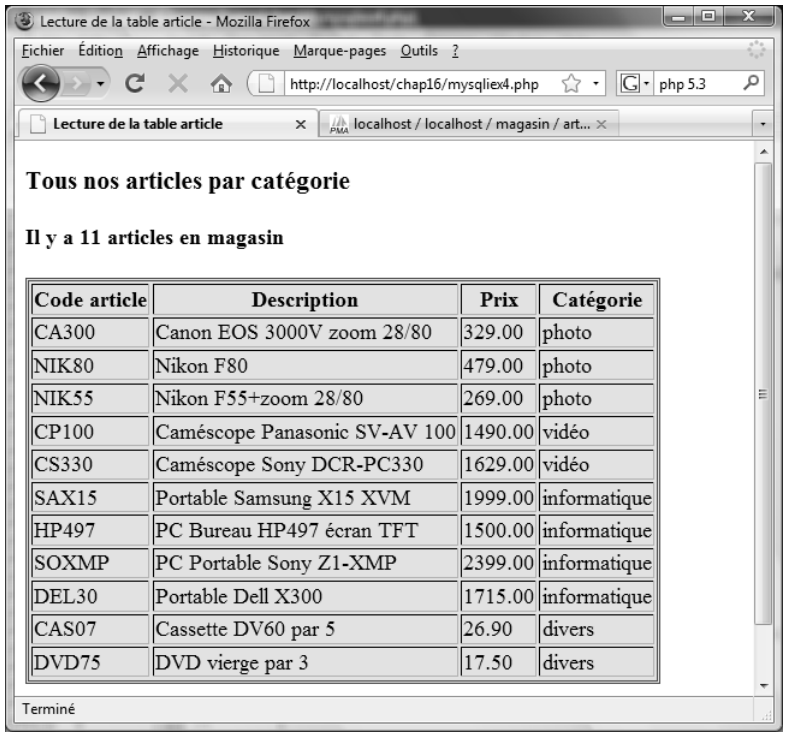

Figure 16-1

Lecture de la table article

Lecture des noms de colonnes.

Dans l'exemple précédent, les titres des colonnes du tableau XHTML étant écrits à l'avance dans le script, nous pouvons automatiser cette opération en récupérant les noms des colonnes de la table interrogée ou les alias figurant dans les requête SQL. La méthode `fetch_fields()` d'un objet `mysqli_result` nous fournit ces informations et bien d'autres concernant la table. Sa syntaxe est :

```
array $result->fetch_fields(void)
```

Le tableau retourné contient autant d'objets qu'il existe de colonnes dans la requête SQL. Ces derniers ont tous les mêmes propriétés, détaillées dans le tableau 16-1, permettant d'obtenir des informations sur les attributs de la table.

Tableau 16-1 Définition des propriétés

Propriété	Définition
`name`	Nom de la colonne
`orgname`	Nom initial de la colonne si un alias a été créé
`table`	Nom de la table à laquelle la colonne appartient (s'il n'a pas été obtenu dynamiquement par concaténation par exemple)
`orgtable`	Nom initial de la table si un alias a été créé
`def`	Valeur par défaut de l'attribut, donnée dans une chaîne de caractères
`max_length`	Longueur maximale du champ pour le jeu de résultats
`length`	Longueur du champ dans la définition de table
`charsetnr`	Jeu de caractères pour cet attribut
`flags`	Entier représentant le bit-flags pour cet attribut
`type`	Type de données utilisées pour l'attribut
`decimals`	Nombre de décimales utilisées (pour les attributs de type entier)

Ces informations peuvent nous permettre de reconstituer la structure de la table à laquelle nous avons accès sans en connaitre les détails.

Nous allons utiliser une de ces propriétés pour créer automatiquement les en-têtes d'un tableau XHTML à partir des noms des colonnes ou des alias éventuels définis dans la requête. L'exemple 16-5 illustre cette possibilité à partir d'une requête SQL sélectionnant les articles de la table `article` dont la désignation contient le mot « Sony » en définissant des alias pour les noms des colonnes (repère ❶). Après l'envoi de la requête par la méthode `query()`, nous récupérons le résultat (repère ❷) puis son nombre de lignes (repère ❸) et le tableau d'objets `$titres` contenant les informations présentées dans le tableau 16-1 (repère ❹). Les noms des colonnes ou des alias employés ici sont contenus dans les propriétés `$titres->name` et lus un par un à l'aide d'une boucle `foreach` (repère ❺), puis affichés dans les en-têtes par des éléments `<th>` du tableau XHTML

(repère ❻). L'affichage des données sélectionnées est réalisé dans une boucle `while` avec la méthode `fetch_array()` comme dans l'exemple précédent. Le tableau retourné étant ici indicé (repère ❼), sa lecture est effectuée par une boucle `foreach`. Le résultat et l'objet connexion sont ensuite supprimés.

☛ Exemple 16-5. Lecture des noms des colonnes

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Lecture de la table article</title>
  <style type="text/css" >
  table {border-style:double;border-width: 3px;border-color:red;
  ➥background-color: yellow;}
  </style>
 </head>
<body>
<?php
include("connexobjet.inc.php");
$idcom=connexobjet("magasin","myparam");
//
$requete="SELECT id_article AS 'Code article',designation AS 'Désignation',prix
➥AS 'Prix Unitaire',categorie AS 'Catégorie' FROM article WHERE designation
➥LIKE '%Sony%' ORDER BY categorie"; ←❶
//
$result=$idcom->query($requete); ←❷
//
if(!$result)
{
    echo "Lecture impossible";
}
else
{
  $nbart=$result->num_rows; ←❸
  $titres=$result->fetch_fields(); ←❹
  echo "<h3> Tous nos articles de la marque Sony</h3>";
  echo "<h4> Il y a $nbart articles en magasin </h4>";
  echo "<table border=\"1\"> <tr>";
  //Affichage des titres
  foreach($titres as $colonne) ←❺
  {
      echo "<th>", htmlentities($colonne->name) ,"</th>"; ←❻
  }
   echo "</tr>";
   //Lecture des lignes de résultat
  while($ligne=$result->fetch_array(MYSQLI_NUM)) ←❼
  {
    echo "<tr>";
    foreach($ligne as $valeur)
```

```
    {
      echo "<td> $valeur </td>";
    }
    echo "</tr>";
  }
  echo "</table>";
}
$result->free_result();
$idcom->close();
?>
</body>
</html>
```

La figure 16-2 illustre les résultats obtenus.

Figure 16-2

Lecture des noms des colonnes

Récupération des valeurs dans un objet

Les objets de type `mysqli_result` possèdent la méthode `fetch_object()` dont la syntaxe est :

```
object $result->fetch_object()
```

À chaque appel de cette méthode, l'objet retourné représente une ligne de résultat et possède autant de propriétés qu'il existe d'attributs dans la requête SQL `SELECT` ; les noms de ces propriétés sont ceux des colonnes de la table ou des alias éventuels.

Si nous récupérons une ligne de résultat dans la variable `$ligne` :

```
$ligne=$result->fetch_object()
```

La valeur d'un attribut `nom` est lue avec la ligne de code :

```
$ligne->nom
```

L'exemple 16-6 réalise, en employant cette méthode, la recherche et l'affichage dans un tableau XHTML de tous les clients qui habitent Paris. Par contre, pour afficher les titres du tableau nous n'allons pas utiliser la méthode `fetch_fields()` comme précédemment. Dans la requête SQL nous définissons des alias qui vont être les en-têtes (repère ❶). Pour les lire, nous appelons une première fois la méthode `fetch_object()` pour l'objet `$result` (repère ❷) et récupérons un objet `$titres`. Ici, ce ne sont pas les valeurs de ses propriétés qui nous intéressent mais leurs noms. Ces derniers sont récupérables dans la variable `$colonne` en appliquant une boucle `foreach` à l'objet `$titres` (repère ❸). Nous intégrons alors ces valeurs dans des éléments `<th>` (repère ❹). Nous pourrions maintenant appeler de nouveau la méthode `fetch_object()` pour lire les données mais, à ce niveau, il se pose un problème. En effet, cette méthode ayant déjà été appelée, un deuxième appel lirait la deuxième ligne de résultat et la première serait perdue. Une solution est d'utiliser la méthode `data_seek()` (repère ❺), dont la syntaxe est :

```
boolean $result->data_seek(int N)
```

Le paramètre `N` désigne la ligne de résultat sur laquelle nous voulons pointer. Le booléen retourné permet de vérifier que l'opération est bien réalisée et que la ligne N existe bien, par exemple. Nous pouvons maintenant utiliser une boucle `while` pour lire chacune des lignes du résultat (repère ❻) et afficher les données. Remarquez que, comme nous l'avons déjà signalé, les noms des propriétés de l'objet `$ligne` sont ici les alias définis dans la requête SQL (repère ❼). Le tableau XHTML obtenu est représenté à la figure 16-3.

☛ Exemple 16-6. Lecture des données dans un objet

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lecture de la table client</title>
<style type="text/css" >
table {border-style:double;border-width:3px;border-color:red;
➥background-color:yellow;}
</style>
</head>
<body>
<?php
include("connexobjet.inc.php");
$idcom=connexobjet("magasin","myparam");
$requete="SELECT id_client AS 'Code_client',nom,prenom,adresse,age,mail
➥FROM client WHERE ville ='Paris' ORDER BY nom"; ←❶
$result=$idcom->query($requete);
if(!$result)
{
  echo "Lecture impossible";
}
```

```
else
{
  $nbart=$result->num_rows;
  echo "<h3> Il y a $nbart clients habitant Paris</h3>";
  //Affichage des titres du tableau
  $titres=$result->fetch_object(); ←2
  echo "<table border=\"1\"> <tr>";
  foreach($titres as $colonne=>$val) ←3
    {
      echo "<th>", $colonne ,"</th>"; ←4
    }
  echo "</tr>";
  //Affichage des valeurs du tableau
  echo "<tr>";
  $result->data_seek(0); ←5
  while ($ligne = $result->fetch_object()) ←6
  {
    echo"<td>", $ligne->Code_client,"</td>", "<td>", $ligne->nom,"</td>","<td>",
    ➥$ligne->prenom,"</td>","<td>", $ligne->adresse,"</td>","<td>", $ligne->age,
    ➥"</td>","<td>", $ligne->mail,"</td></tr>"; ←7
  }
echo "</table>";
$result->free_result();
$idcom->close();
}
?>
</body>
</html>
```

Figure 16-3

Lecture des noms des colonnes

Insertion de données dans la base

Dans un site interactif, il faut pouvoir enregistrer dans la base de données les informations saisies par les visiteurs dans un formulaire XHTML en vue d'une réutilisation ultérieure, comme dans le cas des coordonnées complètes d'un client.

En vous situant, comme au chapitre 15, dans la perspective d'un site de e-commerce, vous allez réaliser la saisie puis l'insertion des coordonnées d'un client dans la table `client` de la base `magasin`. Dans un second temps, nous lui permettrons de mettre à jour ces informations.

Insertion des données

Le formulaire XHTML est l'outil privilégié pour saisir de données et les envoyer vers le serveur PHP/MySQL. Nous disposons désormais de la fonction `connexobjet()` pour effectuer la connexion et de la méthode `query()` pour l'envoi des requêtes. Seule la commande SQL `INSERT` distingue cette opération de celle de lecture de données. Le script de l'exemple 16-7 réalise ce type d'insertion en récupérant les données saisies par le client dans un formulaire lors d'une commande. Nous commençons par vérifier l'existence des saisies obligatoires correspondant aux variables `$_POST['nom']`, `$_POST['adresse']` et `$_POST['ville']` (repère ❶). Quand une requête est formée en utilisant les saisies faites par l'utilisateur, il est préférable d'utiliser le caractère d'échappement pour les caractères spéciaux des chaines récupérées dans le tableau `$_POST`, en particulier les guillemets, qui peuvent poser problème dans la requête. Nous disposons pour cela de la méthode `escape_string()` des objets `mysqli` dont la syntaxe est :

```
string $idcom->escape_string(string $chaine)
```

La chaine obtenue contient le caractère d'échappement / devant les caractères spéciaux `NULL`, `\n`, `\r`, `'`, `"` et `Control-Z`.

Le script récupère toutes les saisies et les protège (repères ❸ à ❽). La colonne `id_client` de la table `client` ayant été déclarée avec l'option `AUTO_INCREMENT`, il faut y insérer la valeur `NULL` en lui donnant la valeur `"\N"` (repère ❷). Le résultat de la requête est ici un booléen permettant de vérifier la bonne insertion des données dans la table (repère ❾). Le script doit communiquer son identifiant au client pour qu'il puisse modifier éventuellement ses données. La valeur de la colonne `id_client` est récupérable à l'aide de la propriété `insert_id` de l'objet `mysqli`. Ce nombre entier est affiché dans une boite d'alerte JavaScript (repère ❿).

☛ **Exemple 16-7 Insertion de données**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title>Saisissez vos coordonnées</title>
```

```
 </head>
 <body>
  <form action= "<?php echo $_SERVER['PHP_SELF'];?>" method="post"
enctype="application/x-www-form-urlencoded">
<fieldset>
<legend><b>Vos coordonnées</b></legend>
<table>
<tr><td>Nom : </td><td><input type="text" name="nom" size="40" maxlength="30"/>
➥</td></tr>
<tr><td>Prénom : </td><td><input type="text" name="prenom" size="40"
➥maxlength="30"/></td></tr>
<tr><td>Age : </td><td><input type="text" name="age" size="40" maxlength="2"/>
➥</td></tr>
<tr><td>Adresse : </td><td><input type="text" name="adresse" size="40"
➥maxlength="60"/></td></tr>
<tr><td>Ville : </td><td><input type="text" name="ville" size="40" maxlength="40"/>
➥</td></tr>
<tr><td>Mail : </td><td><input type="text" name="mail" size="40" maxlength="50"/>
➥</td></tr>
<tr>
<td><input type="reset" value=" Effacer "></td>
<td><input type="submit" value=" Envoyer "></td>
</tr>
</table>
</fieldset>
</form>
<?php
include("connexobjet.inc.php");
$idcom=connexobjet('magasin','myparam');
if(!empty($_POST['nom'])&& !empty($_POST['adresse'])&&
➥!empty($_POST['ville'])) ←❶
{
  $id_client="\N"; ←❷
  $nom=$idcom->escape_string($_POST['nom']); ←❸
  $prenom=$idcom->escape_string($_POST['prenom']); ←❹
  $age=$idcom->escape_string($_POST['age']); ←❺
  $adresse=$idcom->escape_string($_POST['adresse']); ←❻
  $ville=$idcom->escape_string($_POST['ville']); ←❼
  $mail=$idcom->escape_string($_POST['mail']); ←❽
  //Requête SQL
  $requete="INSERT INTO client VALUES('$id_client','$nom','$prenom','$age',
  ➥'$adresse','$ville','$mail')";
  $result=$idcom->query($requete); ←❾
  if(!$result)
  {
    echo $idcom->errno;
    echo $idcom->error;
    echo "<script type=\"text/javascript\">
```

```
    alert('Erreur : ".$idcom->error."')</script>";
  }
  else
  {
    echo "<script type=\"text/javascript\">
    alert('Vous êtes enregistré Votre numéro de client est : ".
    ➥$idcom->insert_id."')</script>"; ←⑩
  }
}
else {echo "<h3>Formulaire à compléter!</h3>";}
?>
</body>
</html>
```

La figure 16-4 illustre la page de saisie et la figure 16-5 la boîte d'alerte JavaScript qui donne son identifiant au client.

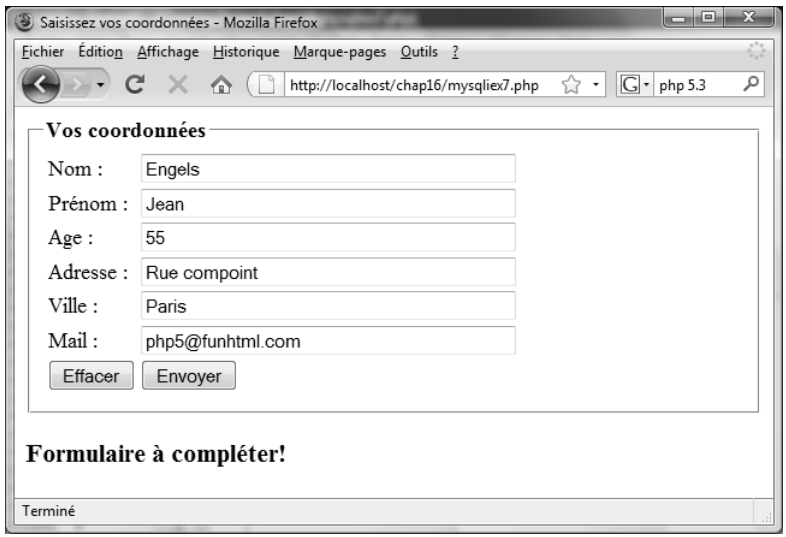

Figure 16-4

Formulaire d'insertion de données

Figure 16-5

Boite d'alerte JavaScript donnant le numéro de client

Mise à jour d'une table

Un client doit pouvoir modifier ses coordonnées, comme son adresse de livraison ou son e-mail. L'exemple 16-8 crée une page contenant un formulaire qui permet la saisie du code client dans une zone de texte XHTML (repère ❶). L'attribut `action` de l'élément `<form>` renvoie le traitement de la saisie au fichier `mysqliex9.php` de l'exemple 16-9. La page créée est conforme à la figure 16-6.

Exemple 16-8 Page de saisies des modifications

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
   <title>Modifiez vos coordonnées</title>
</head>
<body>
  <form action= "mysqliex9.php" method="post"
  ➥enctype="application/x-www-form-urlencoded">
  <fieldset>
  <legend><b>Saisissez votre code client pour modifier vos coordonnées</b></legend>
  <table><tbody>
  <tr>
  <td>Code client : </td>
  <td><input type="text" name="code" size="20" maxlength="10"/></td> ←❶
  </tr>
  <tr>
  <td>Modifier : </td>
  <td><input type="submit" value="Modifier"/></td>
  </tr>
  </tbody></table>
  </fieldset>
  </form>
</body>
</html>
```

Figure 16-6

Page de saisie du code client

La mise à jour des coordonnées du client est réalisée par le script de l'exemple 16-9.

La première inclusion de code PHP renvoie le client vers la page de saisie du code s'il a validé le formulaire sans avoir effectué de saisie (repère ❶). Rappelons, comme nous l'avons déjà vu par ailleurs, que cette partie de code PHP doit figurer en tête du fichier car elle utilise la fonction `header()` pour effectuer la redirection.

La suite du fichier comporte deux parties distinctes. La première crée dynamiquement un formulaire permettant la modification des données et la seconde enregistre ces données dans la base.

Lors du premier appel du fichier de l'exemple 16-9, la condition de l'instruction `if` (repère ❷) est nécessairement vérifiée car la variable `$_POST['modif']` ne contient rien. Elle correspond à la valeur associée au bouton `submit` du formulaire qui n'est pas encore créé. Le script crée une connexion au serveur MySQL pour y lire les coordonnées actuelles du client, dont le code est contenu dans la variable `$code` issue de la page de saisie de l'exemple 16-8 (repère ❸).

La requête SQL sélectionne alors toutes les colonnes de la table `client` dont l'identifiant client (colonne `id_client` de la table `client`) correspond à la valeur de la variable `$code` (repère ❹), dans le but de compléter le formulaire avec les données actuelles. Cela permet de ne saisir que les modifications éventuelles de coordonnées du client, sans devoir ressaisir l'ensemble. Ces coordonnées sont lues à l'aide de la méthode `fetch_row()` de l'objet `$result` de type `mysqli_result`, puisque le résultat de la requête `SELECT` ne comporte qu'une seule ligne. Elles sont alors contenues dans la variable `$coord` de type `array`. Pour afficher les coordonnées dans le formulaire, vous devez attribuer les valeurs de ses éléments aux attributs `value` des différents champs `<input />` (repère ❻).

La figure 16-7 montre un exemple de création dynamique de formulaire pour le client dont l'identifiant vaut 12. Le champ caché `code` du formulaire permet de passer la valeur du code client à la partie du script chargée de l'enregistrement des données modifiées (repère ❼).

L'envoi du formulaire utilise la deuxième partie du script, qui met à jour les données du visiteur dans la table `client` après avoir vérifié l'existence de valeurs pour les champs obligatoires du formulaire (repère ❽). Seules les colonnes `nom`, `adresse`, `ville` et `mail` peuvent être mises à jour à l'aide de la requête suivante (repère ❾) le reste étant forcément inchangé :

```
UPDATE client SET nom='$nom',adresse='$adresse',ville='$ville',
➥mail='$mail'
WHERE id_client='$code'
```

La vérification du résultat de la requête (repère ❿) permet d'afficher une boîte d'alerte JavaScript contenant soit un message d'erreur, soit la confirmation de l'enregistrement. La page ne devant pas être une impasse, le visiteur est redirigé d'office vers la page d'accueil du site `index.html` (repères ⓫ et ⓬).

☛ Exemple 16-9 Mise à jour de données

```
<?php
  if(empty($_POST['code'])){header("Location:mysqliex8.php");} ←❶
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Modifiez vos coordonnées</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head>
<body>
<?php
include('connexobjet.inc.php');
$idcom=connexobjet('magasin','myparam');
if($_POST['modif']!='Enregistrer') ←❷
{
  $code=$idcom->escape_string($_POST['code']); ←❸
  //Requête SQL
  $requete="SELECT * FROM client WHERE id_client='$code' "; ←❹
  $result=$idcom->query($requete);
  $coord=$result->fetch_row(); ←❺
  //Création du formulaire complété avec les données existantes ←❻
  echo "<form action= \"". $_SERVER['PHP_SELF']."\"
  ➥method=\"post\"enctype=\"application/x-www-form-urlencoded\">";
  echo "<fieldset>";
  echo "<legend><b>Modifiez vos coordonnées</b></legend>";
  echo "<table>";
  echo "<tr><td>Nom : </td><td><input type=\"text\" name=\"nom\"
  ➥size=\"40\" maxlength=\"30\" value=\"$coord[1]\"/> </td></tr>";
  echo "<tr><td>Prénom : </td><td><input type=\"text\" name=\"prenom\"
  ➥size=\"40\" maxlength=\"30\" value=\"$coord[2]\"/></td></tr>";
  echo "<tr><td>Age : </td><td><input type=\"text\" name=\"age\"
  ➥size=\"40\" maxlength=\"2\" value=\"$coord[3]\"/></td></tr>";
  echo "<tr><td>Adresse : </td><td><input type=\"text\" name=\"adresse\"
  ➥size=\"40\" maxlength=\"60\" value=\"$coord[4]\"/></td></tr>";
  echo "<tr><td>Ville : </td><td><input type=\"text\" name=\"ville\"
  ➥size=\"40\" maxlength=\"40\" value=\"$coord[5]\"/></td></tr>";
  echo "<tr><td>Mail : </td><td><input type=\"text\" name=\"mail\"
  ➥size=\"40\" maxlength=\"50\" value=\"$coord[6]\"/></td></tr>";
  echo "<tr><td><input type=\"reset\" value=\" Effacer \"></td> <td><input
  ➥type=\"submit\" name=\"modif\" value=\"Enregistrer\"></td></tr></table>";
  echo "</fieldset>";
  echo "<input type=\"hidden\" name=\"code\"  value=\"$code\"/>"; ←❼
  echo "</form>";
  $result->close();
  $idcom->close();
}
```

```
elseif(isset($_POST['nom'])&& isset($_POST['adresse'])&&
➥isset($_POST['ville'])) ←8
{
  //ENREGISTREMENT
  $nom=$idcom->escape_string($_POST['nom']);
  $adresse=$idcom->escape_string($_POST['adresse']);
  $ville=$idcom->escape_string($_POST['ville']);
  $mail=$idcom->escape_string($_POST['mail']);
  $age=(integer)$_POST['age'];
  $code=$idcom->escape_string($_POST['code']);
  //Requête SQL
  $requete="UPDATE client SET nom='$nom',adresse='$adresse', ville=
  ➥'$ville',mail='$mail',age=$age WHERE id_client='$code'"; ←9
  $result=$idcom->query($requete);
  if(!$result) ←10
  {
    echo "<script type=\"text/javascript\">
    alert('Erreur : ".$result->error."')</script>"; ←11
  }
  else
  {
    echo "<script type=\"text/javascript\"> alert('Vos modifications
   ➥sont enregistrées');window.location='index.html';</script>"; ←12
  }
  $result->close();
  $idcom->close();
}
else
{
  echo "Modifier vos coordonnées!";
}
?>
</body>
</html>
```

Vérification de l'insertion

La propriété `affected_rows` de l'objet `mysqli` contient le nombre de lignes affectées par une mise à jour. Elle peut donc nous permettre de savoir si la modification est bien réalisée, car elle doit ici être égale à 1. Au repère 10 de l'exemple 16-9, vous pourriez donc écrire :

```
if($idcom->affected_rows()!=1)
{//Affichage de l'erreur
   }
else
{//Message de confirmation
   }
```

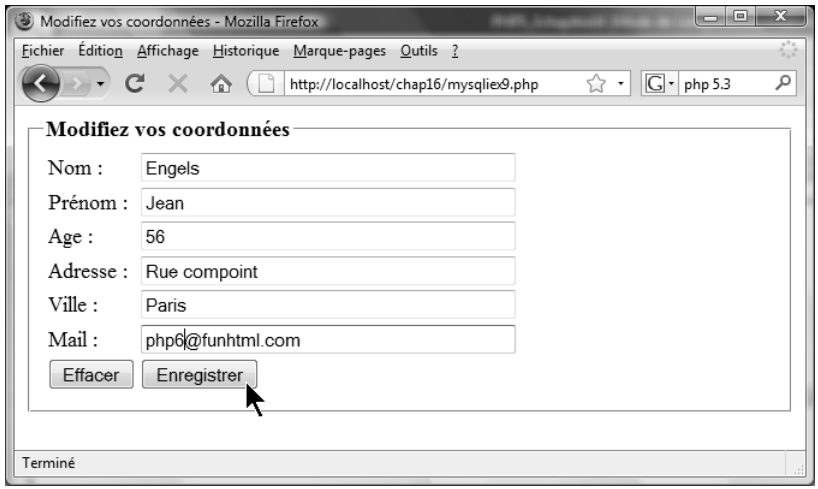

Figure 16-7

Formulaire de saisie des coordonnées créé dynamiquement

Recherche dans la base

Un site de commerce en ligne doit permettre à ses visiteurs et futurs clients d'effectuer des recherches dans la base de données afin d'accéder plus rapidement à l'information sur le produit recherché. Il doit en outre permettre d'effectuer des statistiques marketing à l'usage du propriétaire du site. Ces recherches concernent aussi bien les sites de commerce en ligne que les annuaires et moteurs de recherche des sites de contenu.

L'exemple 16-10 crée un formulaire classique permettant de saisir un mot-clé et d'effectuer des choix de tri des résultats. Les critères de tri selon le prix, la catégorie ou l'identifiant d'article sont affichés sous forme de liste déroulante. Le choix de l'ordre croissant ou décroissant s'effectue au moyen de deux boutons radio ayant le même attribut `name`, ce qui les rend exclusifs l'un de l'autre.

Le script contrôle d'abord que le visiteur a saisi un mot-clé dans le formulaire en vérifiant que la variable `$_POST['motcle']` n'est pas vide (repère ❶). Il récupère ensuite le mot-clé, la catégorie, le critère de tri et l'ordre d'affichage, respectivement dans les variables `$motcle`, `$categorie`, `$ordre` et `$tri` (repères ❷ à ❺).

Si la catégorie choisie est `"tous"`, la partie de la commande `WHERE` concernant cette catégorie est vide. Pour les autres choix, elle est égale à `"AND categorie=$categorie"` (repère ❻). La requête de sélection suivante (repère ❼) est alors créée par le code :

```
"SELECT id_article AS 'Code article',
designation AS 'Description',
prix,categorie AS 'Cat&#233;gorie'
FROM article WHERE lower(designation) LIKE'%$motcle%'".$reqcategorie.
"ORDER BY $tri $ordre";
```

On peut remarquer l'utilisation de la fonction MySQL `lower()` qui permet d'effectuer une recherche insensible à la casse. De cette façon, que l'utilisateur cherche les mots-clés « `sony` » ou « `Sony` », il obtiendra bien les résultats présents dans la table `article`.

L'utilisation d'alias donne un meilleur affichage des titres du tableau de résultats. Après la connexion au serveur, vous récupérez le résultat de la requête dans la variable `$result`. La lecture des résultats et l'affichage de toutes les lignes retournées sont réalisés avec la méthode `fetch_row()` (repère ❾). La figure 16-8 illustre la page créée après la recherche du mot-clé `portable`.

☛ Exemple 16-10. Page de recherche d'articles

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Rechercher un article dans le magasin</title>
 </head>
 <body>
  <form action= "<?php echo $_SERVER['PHP_SELF']?>" method="post"
  ➥enctype="application/x-www-form-urlencoded">
  <fieldset>
  <legend><b>Rechercher un article en magasin</b></legend>
  <table>
  <tbody>
  <tr> <td>Mot-cl&#233;: </td>
  <td><input type="text" name="motcle" size="40" maxlength="40"
  ➥value="<?php $_POST['motcle']?>"/></td>
  </tr>
  <tr>
  <td>Dans la cat&#233;gorie : </td>
  <td>
  <select name="categorie">
  <option value="tous">Tous</option>
  <option value="vidéo">Vid&#233;o</option>
  <option value="informatique">Informatique</option>
  <option value="photo">Photo</option>
  <option value="divers">Divers</option>
  </select>
  </td>
  </tr>
  <tr>
  <td>Trier par : </td>
  <td>
  <select name="tri">
  <option value="prix">Prix</option>
  <option value="categorie">Cat&#233;gorie</option>
  <option value="id_article">Code</option>
  </select>
  </td>
```

```
  </tr>
  <tr><td>En ordre: </td>
  <td>Croissant<input type="radio" name="ordre" value="ASC" checked="checked"/>
  ➥D&#233;croissant<input type="radio" name="ordre" value="DESC" />
  </td> </tr>
  <tr><td>Envoyer</td><td><input type="submit" name="" value="OK"/></td></tr>
  </tbody>
  </table>
  </fieldset>
  </form>
<?php
if(!empty($_POST['motcle'])) ←❶
{
  include('connexobjet.inc.php');
  $motcle=($_POST['motcle']); ←❷
  $categorie=($_POST['categorie']); ←❸
  $ordre=($_POST['ordre']); ←❹
  $tri=($_POST['tri']); ←❺

  //Requête SQL
  $reqcategorie=($_POST['categorie']=="tous")?"":"AND categorie='$categorie'"; ←❻
  $requete="SELECT id_article AS 'Code article',designation AS 'Description',prix,
  ➥categorie AS 'Cat&#233;gorie' FROM article WHERE lower(designation)
  ➥LIKE'%$motcle%'".$reqcategorie."ORDER BY $tri $ordre"; ←❼
  $idcom=connexobjet('magasin','myparam');
  $result=$idcom->query($requete); ←❽
  if(!$result) ←❾
  {
    echo "Lecture impossible";
  }
  else
  {
  $nbcol=$result->field_count;
  $nbart=$result->num_rows;
  $titres=$result->fetch_fields();
  echo "<h3> Il y a $nbart articles correspondant au mot-cl&#233; : $motcle</h3>";
    //Affichage des titres du tableau
    echo "<table border=\"1\"> <tr>";
    foreach($titres as $nomcol=>$val)
    {
      echo "<th>", $titres[$nomcol]->name ,"</th>";
    }
    echo "</tr>";
    //Affichage des valeurs du tableau
    for($i=0;$i<$nbart;$i++)
    {
      $ligne=$result->fetch_row();
      echo "<tr>";
      for($j=0;$j<$nbcol;$j++)
      {
        echo "<td>",$ligne[$j],"</td>";
```

```
        }
        echo "</tr>";
      }
      echo "</table>";
      $result->close();
      $idcom->close();
    }
}
?>
</body>
</html>
```

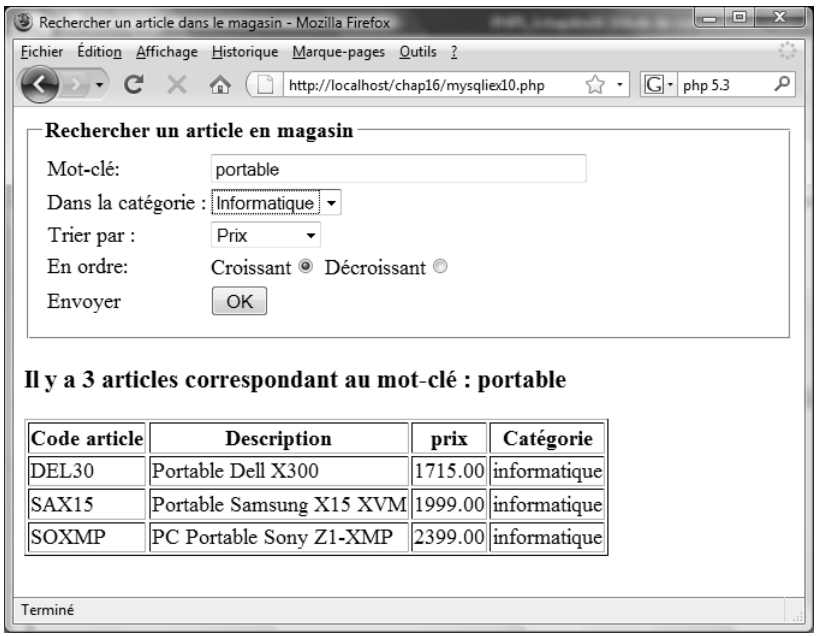

Figure 16-8

Formulaire de recherche et résultats obtenus

Les requêtes préparées

Nous avons déjà construit des requêtes SQL dynamiquement à partir d'informations saisies par l'utilisateur dans l'exemple précédent. Les requêtes préparées permettent de créer des requêtes SQL qui ne sont pas directement utilisables mais qui contiennent des paramètres auxquels on peut donner des valeurs différentes en fonction des besoins, pour des appels répétitifs par exemple. Une requête préparée peut se présenter sous la forme suivante :

```
SELECT prenom,nom FROM client WHERE ville=? AND id_client>=?
```

Dans cette requête, les caractères ? vont être remplacés par des valeurs quelconques en fonction des besoins du visiteur.

La démarche à suivre pour utiliser une requête préparée est la suivante :

1. Écrire la chaîne de requête comme paramètre de la méthode `prepare()` de l'objet `mysqli`. Cette dernière retourne un objet de type `mysqli_stmt` qui représente la requête préparée.
2. Lier les paramètres dans l'ordre de leur apparition avec des valeurs ou des variables à l'aide de la méthode `bind_param()` de l'objet `mysqli_stmt`, selon la syntaxe : `$mysqli_stmt->bind_param(string $types, $param1,...$paramN)`.

   La chaîne `$types` est la concaténation de caractères indiquant le type de chacun des paramètres. La signification de ces caractères est présentée au tableau 16-2.

Tableau 16-2. Signification des caractères de la chaîne $types

Caractère	Définition
`i`	Entier (integer)
`d`	Décimal
`s`	Chaîne de caractères (string)
`b`	Blob (texte long)

3. Pour trois paramètres qui seraient, dans l'ordre d'apparition dans la requête, un décimal, une chaîne et un entier, la chaîne `$types` serait par exemple `"dsi"`.
4. Exécuter la requête en appelant la méthode `execute()` de l'objet `mysqli_stmt`.
5. Pour les requêtes préparées qui retournent des résultats, comme `SELECT`, il faut ensuite lier les résultats à des variables PHP, avec la méthode `bind_result()` selon la syntaxe : `mysqli_stmt->bind_result($var1,...$varN)` avec autant de paramètres qu'il existe de colonnes lues dans la requête. L'objet `mysqli_result` obtenu contient alors toutes les lignes du résultat.
6. Lire ces lignes à l'aide d'une boucle en appliquant la méthode `fetch()` à cet objet et utiliser ces résultats pour un affichage avec les noms des variable définies à l'étape 4.
7. Libérer la mémoire occupée par l'objet `mysqli_stmt` avec la méthode `free_result()`.

L'exemple 16-11 présente une application de requête préparée dans laquelle ce sont les saisies d'un utilisateur qui déterminent les valeurs des paramètres et donc la requête finale. Un formulaire classique demande la saisie d'un nom de ville et d'un numéro de client (repères ❶ et ❷) dans le but de trouver tous les clients habitant cette ville et dont l'identifiant est supérieur ou égal à la valeur saisie. Ces saisies sont d'abord récupérées dans les variables `$ville` et `$id_client` (repères ❸ et ❹). Après la connexion à la base, nous écrivons la requête préparée avec la méthode `prepare()` (repère ❺), nous lions les paramètres de type `string` et `integer` aux variables `$ville` et `$id_client` (repère ❻), puis

nous exécutons la requête ainsi complétée en appelant la méthode `execute()` (repère ❼). Comme il s'agit d'une requête `SELECT`, elle retourne une ou plusieurs lignes contenant chacune deux valeurs que nous lions aux variables `$prenom` et `$nom` (repère ❽). Les différentes lignes sont alors lues et affichées avec une boucle `while` en appelant la méthode `fetch()` (repère ❾), les valeurs cherchées étant contenues dans les variables `$prenom` et `$nom`. La mémoire est ensuite libérée (repère ❿) et l'objet `mysqli` supprimé (repère ⓫).

☛ Exemple 16-11 Utilisation des requêtes préparées

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title>Recherche de client</title>
  <style type="text/css" >
    div{font-size: 16px;}
  </style>
 </head>
 <body>
  <form method="post" action="mysqliex11.php">
  <fieldset>
    <legend>Recherche de client</legend>
    <label>Ville </label>
    <input type="text" name="ville" /><br /> ←❶
    <label>Id_client</label>
    <input type="text" name="id_client" /> ←❷
    <input type="submit" value="Envoyer" />
  </fieldset>
  </form>
 </body>
</html>
<?php
if(isset($_POST['ville']) &&  isset($_POST['id_client']))
{
  $ville=strtolower($_POST['ville']); ←❸
  $id_client=$_POST['id_client']; ←❹
  include('connexobjet.inc.php');
  $idcom=connexobjet('magasin','myparam');
  $reqprep=$idcom->prepare("SELECT prenom,nom FROM client WHERE lower(ville)=?
  ➥AND id_client>=? "); ←❺
  $reqprep->bind_param("si",$ville,$id_client); ←❻
  $reqprep->execute(); ←❼
  $reqprep->bind_result($prenom,$nom); ←❽
  echo "<div><h3>Client(s) habitant à ", ucfirst($ville)," et dont l'identifiant est
  ➥supérieur à $id_client</h3>";
  //Affichage des résultats
  while ($reqprep->fetch()) ←❾
  {
    echo "<h3> $prenom $nom</h3>";
```

```
    }
    echo "</div>";
    $reqprep->free_result(); ←⓾
    $idcom->close(); ←⓫
}
?>
```

Avec notre base de données magasin et les paramètres « Paris » et « 3 », nous obtenons par exemple l'affichage présenté à la figure 16-9

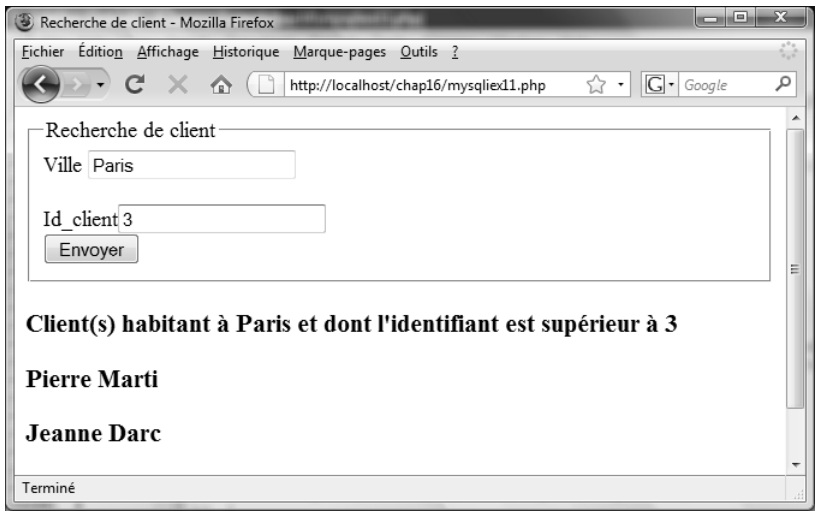

Figure 16-9
Résultats d'une requête préparée

Les transactions

Dans l'exemple de notre base magasin, si un client effectue un achat, il faut réaliser simultanément deux insertions : une dans la table `commande` et une dans la table `ligne`. Si, pour une raison quelconque, matérielle ou logicielle, la seconde insertion n'est pas réalisée alors que la première l'est déjà, la base contiendra une incohérence car il existera une commande ne comportant aucune ligne. L'inverse ne serait guère préférable car, dans ce cas, il existerait une ligne qui ne serait reliée à aucune commande. Les transactions permettent de gérer ce genre de situation en effectuant des commandes « tout ou rien » ce qui, en pratique pour notre exemple ci-dessus, signifie que si l'une des deux requêtes n'est pas exécutée, aucune ne l'est. L'intégrité de la base s'en trouve préservée.

Le langage SQL possède des commandes qui permettent de gérer les transactions, mais l'extension `mysqli` nous fournit des méthodes qui permettent de gérer les transactions sans y faire appel.

Dans l'exemple 16-12, nous illustrons la procédure à suivre pour effectuer deux requêtes `INSERT` dans la table `article`. Par défaut, le mode `autocommit` de MySQL est activé, ce qui signifie que chaque requête est automatiquement validée. Il nous faut donc le désactiver au moyen de la méthode `autocommit()` de l'objet `mysqli` avec pour paramètre la valeur `FALSE` (repère ❶). À partir de cet instant, les deux requêtes d'insertion que nous voulons réaliser (repères ❷ et ❸) ne seront validées que si nous effectuons explicitement la validation au moyen de la méthode `commit()` (repère ❻), ou bien l'ensemble sera annulé en appelant la méthode `rollback()` de l'objet `mysqli` (repère ❼). Pour choisir de valider ou annuler les insertions, nous comptons le nombre total de lignes insérées dans la base en lisant la propriété `affected_rows` (repères ❹ et ❺). S'il est bien égal à 2, la validation est effectuée (repère ❻), sinon tout est annulé et un message d'information s'affiche (repère ❼).

Pour tester l'efficacité du mécanisme, il suffit de créer une erreur en écrivant par exemple dans la variable `$requete2` le nom `articles`, au lieu de `article`, comme nom de table inexistante. L'affichage de la table `article` avec phpMyAdmin permet de vérifier que même la première requête, qui était correcte, n'a pas été exécutée.

Exemple 16-12 Insertions avec transaction

```
<?php
include('connexobjet.inc.php');
$idcom=connexobjet('magasin','myparam');
$idcom->autocommit(FALSE); ←❶
$requete1="INSERT INTO article VALUES ('AZERT', 'Lecteur MP3', 59.50,
➥'divers');"; ←❷
$requete2="INSERT INTO article VALUES ('QSDFG', 'Bridge Samsung 10 Mo', 358.90,
➥'photo');"; ←❸
//pour empêcher la validation écrire articles au lieu de article
//*****************************************************************
$idcom->query($requete1);
$nb=$idcom->affected_rows; ←❹
echo "LIGNES INSEREES",$nb,"<hr />";
$idcom->query($requete2);
$nb+=$idcom->affected_rows; ←❺
if($nb==2)
{
  $idcom->commit(); ←❻
  echo $nb," lignes insérées";
}
else
{
  $idcom->rollback(); ←❼
  echo "transaction annulée";
}
?>
```

Mémo des méthodes et propriétés

Classe mysqli : méthodes

```
mysqli ( [string $host [, string $username [, string $passwd [, string $base [, int $port [,
string $socket]]]]]] )
}
```

Crée un objet mysqli.

```
boolean autocommit ( boolean $mode )
```

Active (`$mode =TRUE`) ou désactive (`$mode=FALSE`) le mode autocommit.

```
boolean change_user ( string $user, string $password, string $base )
```

Change l'utilisateur de la connexion et retourne `TRUE` en cas de réussite et `FALSE` sinon.

```
boolean close (void)
```

Ferme la connexion et retourne `TRUE` en cas de réussite et `FALSE` sinon.

```
boolean commit ( void )
```

Valide la transaction courante et retourne `TRUE` en cas de réussite et `FALSE` sinon.

```
string escape_string ( string $chaine )
```

Crée une chaîne SQL valide qui pourra être utilisée dans une requête SQL. La chaîne de caractères `$chaine` est encodée en une chaîne SQL échappée, en tenant compte du jeu de caractères courant de la connexion.

```
boolean kill ( int $processid )
```

Demande au serveur de terminer un thread MySQL identifié par son identifiant.

```
boolean multi_query ( string $query )
```

Exécute une ou plusieurs requêtes, rassemblées dans le paramètre `query` par des points-virgules.

```
boolean ping ( void )
```

Teste la connexion pour s'assurer que le serveur est bien en fonctionnement. S'il ne fonctionne pas et que l'option globale `mysqli.reconnect` est activée, une connexion automatique sera tentée avec les derniers paramètres utilisés.

```
objet mysqli_stmt prepare ( string $query )
```

Prépare une requête SQL et retourne un objet `mysqli_stmt`.

```
divers query ( string $query [, int $mode] )
```

Exécute une requête sur la base de données. `$mode` est une constante qui vaut `MYSQLI_USE_RESULT` ou `MYSQLI_STORE_RESULT` (par défaut), suivant le comportement désiré. Retourne `TRUE` en cas de succès, `FALSE` en cas d'échec. Pour `SELECT`, `SHOW`, `DESCRIBE` ou `EXPLAIN` retourne un objet `mysqli_result`.

```
boolean rollback ( void )
```

Annule la transaction courante.

```
boolean select_db ( string $base )
```

Sélectionne une base de données.

```
string stat ( void )
```

Retourne une chaîne de caractères contenant des informations sur la connexion ouverte : le temps de fonctionnement, exprimé en secondes, le nombre de threads courant, le nombre de commandes, les tables rechargées et ouvertes.

`objet mysqli_result store_result ( void )`

Stocke un ensemble de résultats dans un objet `mysqli_result` à partir de la dernière requête.

Classe mysqli : propriétés

`integer affected_rows`

Contient le nombre de lignes affectées par la dernière requête `INSERT`, `UPDATE`, `REPLACE` ou `DELETE`.

`integer errno`

Contient un code d'erreur pour la dernière commande SQL.

`integer error`

Contient une chaîne décrivant la dernière erreur.

`integer field_count`

Contient le nombre de colonnes concernées dans la dernière requête.

`integer insert_id`

Contient l'identifiant automatiquement généré pour un attribut déclaré `AUTO_INCREMENT` ou `0` sinon.

`thread_id`

Contient l'identifiant du thread pour la connexion en cours.

`integer warning_count`

Contient le nombre d'avertissements générés par la dernière requête.

Classe mysqli_result : méthodes

`boolean close ( void )`

Supprime l'objet résultat.

`boolean data_seek(integer N)`

Déplace le pointeur interne de résultat à la position `N`.

`divers fetch_array ( [integer $type] )`

Retourne un tableau qui correspond à la ligne récupérée, ou `NULL` s'il n'y a plus de ligne dans le résultat. Le paramètre `$type` détermine le type du tableau ; il vaut `MYSQLI_ASSOC`, `MYSQLI_NUM` ou `MYSQLI_BOTH` respectivement pour obtenir un tableau associatif, indicé ou mixte.

`array fetch_assoc ( void )`

Retourne une ligne de résultat sous forme de tableau associatif.

`object fetch_field ( void )`

Retourne un objet qui contient les caractéristiques d'un attribut de table.

`array fetch_fields ( void )`

Retourne un tableau d'objets qui contient les caractéristiques de tous les attributs de table présents dans une requête.

`object fetch_object (void)`

Retourne un objet dont les propriétés sont les noms des colonnes utilisées dans la requête ou `NULL` s'il n'y a plus de ligne dans le résultat.

`divers fetch_row ( void )`

Retourne un tableau indicé contenant une ligne de résultat ou `NULL` s'il n'y a plus de ligne de résultat.

`boolean field_seek ( int N )`

Place le pointeur de résultat sur le champ `N`.

`void free_result ( void ) ou void close ( void )`

Libère la mémoire associée à l'objet résultat.

Classe mysqli_result : propriétés

`integer field_count`

Contient le nombre de colonnes pour la dernière requête.

`int num_rows`

Retourne le nombre de lignes d'un résultat.

Classe mysqli_stmt : méthodes

`boolean bind_param ( string $types, divers $var1 [...divers $varN] )`

Lie des variables à une requête SQL préparée avec la méthode `prepare()`.

`boolean bind_result (divers $var1 [,...divers $varN] )`

Associe des variables à un résultat de requête préparée et retourne `TRUE` en cas de réussite ou `FALSE` sinon.

`boolean close ( void )`

Termine une requête préparée et retourne `TRUE` en cas de réussite ou `FALSE` sinon.

`void data_seek ( integer N)`

Déplace le pointeur de résultat à la position `N`.

`boolean execute ( void )`

Exécute une requête préparée et retourne `TRUE` en cas de réussite ou `FALSE` sinon.

`boolean fetch ( void )`

Lit des résultats depuis une requête MySQL préparée dans les variables liées.

`void free_result ( void )`

Libère le résultat de la mémoire.

`divers prepare ( string $requete )`

Prépare une requête SQL et retourne `TRUE` en cas de succès, `FALSE` en cas d'échec.

`boolean reset ( void )`

Annule une requête préparée et retourne `TRUE` en cas de succès, `FALSE` en cas d'échec.

Classe mysqli_stmt : propriétés

`integer affected_rows`

Contient le nombre total de lignes concernées par la dernière requête.

`integer errno`

Contient un code erreur pour la dernière requête préparée.

`string error`

Contient la description de la dernière erreur.

`integer field_count`

Contient le nombre de colonnes dans la requête.

`integer insert_id`

Contient l'identifiant généré par la dernière requête `INSERT` pour la colonne `AUTO_INCREMENT`.

`integer num_rows`

Contient le nombre de lignes d'un résultat de requête préparée.

`integer param_count`

Contient le nombre de paramètres nécessaires dans la requête préparée.

Exercices

Tous les exercices ci-dessous portent sur la base de données `voitures` créée aux chapitres 13 et 14. Ils sont identiques à ceux du chapitre 15, mais vous devez les réaliser uniquement avec l'extension `mysqli` objet.

Exercice 1

Créez un script permettant d'afficher le contenu de la table `modele` dans un tableau XHTML. Les résultats doivent être triés par marque.

Exercice 2

Créez un formulaire permettant l'insertion de nouvelles données dans la table `modele`.

Exercice 3

Créez un formulaire permettant l'insertion simultanée des coordonnées d'une personne dans les tables `proprietaire` et `cartegrise`. Il doit contenir les zones de saisie des coordonnées de la personne et la liste des modèles d'une marque créée dynamiquement à partir de la saisie de la marque.

Exercice 4

Créez un formulaire de recherche permettant de retrouver tous les propriétaires d'un type de véhicule de marque et de modèle donnés. Affichez les résultats sous forme de tableau XHTML.

Exercice 5

Créez un formulaire de recherche permettant de retrouver tous les véhicules possédés par une personne donnée. Affichez les résultats sous forme de tableau XHTML.

Exercice 6

Réécrivez entièrement le code de l'exercice 5 en récupérant tous les résultats dans des objets et en manipulant leurs propriétés.

Exercice 7

Refaire l'exercice 4 en utilisant une requête préparée.

Exercice 8

Refaire l'exercice 3 en utilisant une transaction pour s'assurer que les données sont bien insérées dans les différentes tables.