PHP マニュアル
実践 PHP for Windows, Linux
PHP と MySQL の連携

14.トランザクション処理
15.トランザクションの活用
- exec('BEGIN');
- exec('COMMIT');
- exec('ROLLBACK');
- beginTransaction();
- commit();
- rollback();

16.メタデータ
- テーブル情報の取得

1.データベースの種類

データベースといってもデータ構造によって、さまざまな形式のものに分類することができます。古くは、ひとつの情報を1枚のカードとして扱うカード型データベース。関連するデータ同士を網状に関連付けるネットワーク型データベース。現在ではデータベースといったら、ほとんどがリレーショナルデータベースと思って間違いないでしょう。

リレーショナルデータベースにおいて、関連する一連のデータはテーブルと呼ばれる区画に格納されます。概念的には、表計算ソフトを思い浮かべるとわかりやすいと思います。テーブルが表計算のひとつのシートにあたります。そして、表計算シートの列にあたるものをリレーショナルデータベースではカラムと呼びます。同様に、行にあたるものをレコードと呼びます。

リレーショナルデーターベースでは目的にあわせてデータを管理しています。たくさんのシートがあり目的別にわかれていると思ってください。それぞれのテーブルには、別個にわけたテーブルどうしを関連付けするためのキーとなる情報を決めます。データベースを操作する場合はこのキーをもとに別個の表を結合させてほしい情報をとってきます。このキーで元となるキー（テーブルのなかでそれぞれのレコードを判別できるデータ）を主キーと呼び、その主キーを参照するキーを外部キーと呼びます。現在、リレーショナルデータベースでは以下のような製品があります。

データベース名	概要
MySQL	パフォーマンス（とくにデータの読み込み）に定評があり、世界的にも普及しているデータベース。速度重視であるため、対応機能が他と比べやや限られている。
PostgreSQL	オープンソースで開発が続けられているデータベース。数値演算系の関数が豊富で、分散処理などの機能など機能面が豊富。
SQLite	PHP5から標準で利用できるようになった軽量データベース。簡易的なデータベースなので機能面はそれほど高くない。
Oracle	Oracle 社開発の商用データベースサーバ。中規模から大規模まで採用されているデータベース。
SQL Server	Microsoft 社開発の商用データベースサーバ。小規模から大規模まで採用されている。
DB2	IBM 社開発の商用データベースサーバ。
Access	Microsoft 社開発のパーソナル用途向けのデータベース。ODBC ドライバを使用することで他データベースのフロントエンドとして利用可能。

2.データベース抽象化レイヤー

PHP では主要なデータベースサーバに接続するための窓口（関数）が標準で装備されています。しかし、実際に開発を行うプログラマの目から見た場合、それぞれに専用関数が用意されているということは、使用するデータベースが変われば、そのつど、新しい関数を習得しなければならないということです。それぞれのデータベース関数は使い方も似ていれば、命名規則にも共通点が数多くありますが、細部で多くの違いがあるのも事実です。こうした細部の違いはプログラマを混乱させるだけでなく、バグの要因、開発の遅延をもたらす恐れすらあります。

そうした問題を解決するために開発されたのが、データベース抽象化レイヤーです。データーベース抽象化レイヤーは、データベースごとの違いを吸収し、アプリケーションがデータベースに対して接続する際に、共通的なインターフェイスを提供するためのソフトウェアです。これを利用することにより、データベースごとに異なる関数を内部的に吸収統合し、共通化された機能を提供してくれます。PHP で利用できる主なデータベースレイヤーは以下の通りです。

ライブラリ名	長所	短所
PEAR::DB	多くのデータベースに対応 普及率が高く、参考文献資料も豊富にある	処理速度は遅い
PEAR::MDB MDB2	多くのデータベースに対応 スキーマを抽象化	処理速度は遅い
PEAR::DB_DataObject	多くのデータベースに対応 O/R マッピング機能を提供	処理速度は遅い
PDO	PHP5の例外処理に対応 シンプルな構文仕様 処理速度は速い	対応データベースが少ない ドライバによっては未実装機能多いものもあり
dbx	多くのデータベースに対応 処理速度は速い	対応機能が少ない

3.PDO 関数の導入、PHP と MySQL 連携の準備

今回は、PDO 関数を使用したいと思います。PDO 関数は拡張モジュールとなっており、使用するにあたっては、あらかじめ PDO のコアモジュールとドライバモジュールを有効にしておく必要があります。

- Windows の場合

Windows フォルダの php.ini 設定ファイルを開いて、

```
630 extension=php_mysql.dll↵
631 extension=php_mysqli.dll↵
632 ;extension=php_oci8.dll↵
633 ;extension=php_openssl.dll↵
634 extension=php_pdo.dll↵
635 ;extension=php_pdo_firebird.dll↵
636 ;extension=php_pdo_mssql.dll↵
637 extension=php_pdo_mysql.dll↵
```

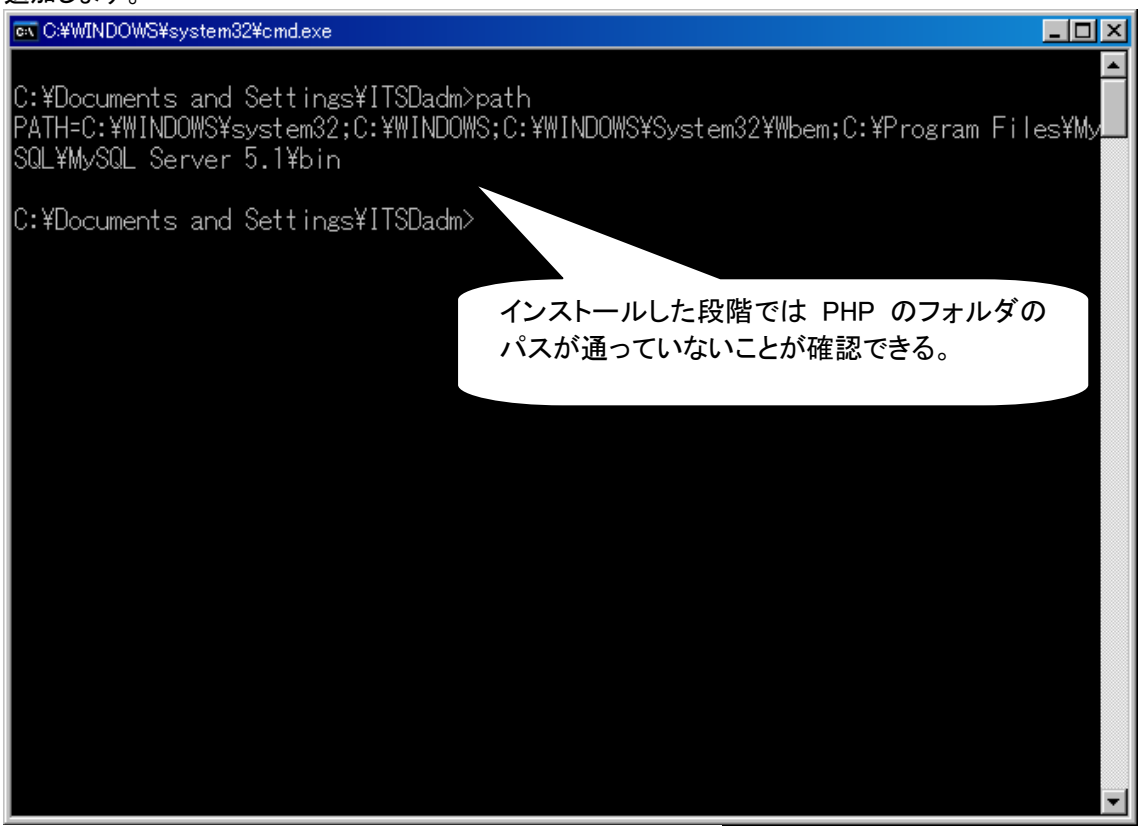
extension=php_mysql.dll
extension=php_mysqli.dll
extension=php_pdo.dll
extension=php_pdo_mysql.dll
のコメント表記（セミコロン）をはずして保存します。
設定変更した際は Apache を再起動してください。

- Linux の場合

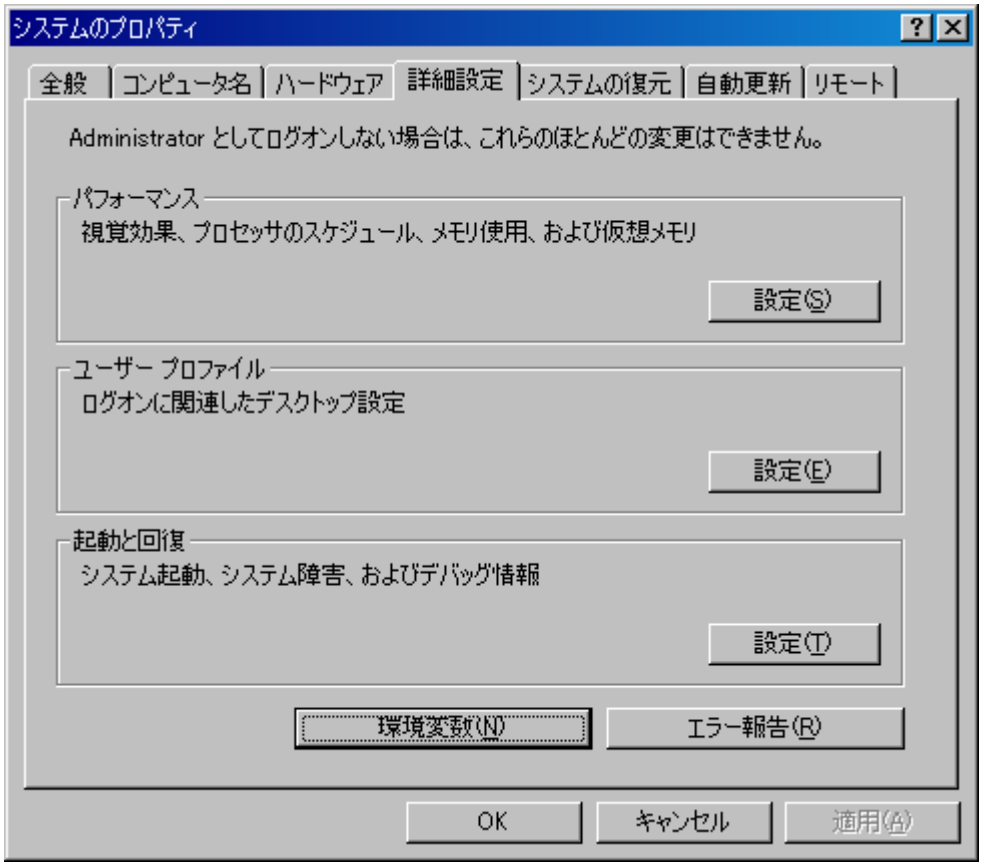
Linux 環境では、デフォルトで静的に PDO 関数が組み込まれていますので、特別な設定は必要ありません。もしも動的モジュールとして組み込みたい場合には、configure 時に--enable-pdo=shared オプションを付与して、生成された.so ファイルを php.ini 上で設定してください。動的モジュールとしておくと、後から PDO ドライバのみ変更となった場合でも、PHP を再コンパイルせずに PDO のみ更新できます。

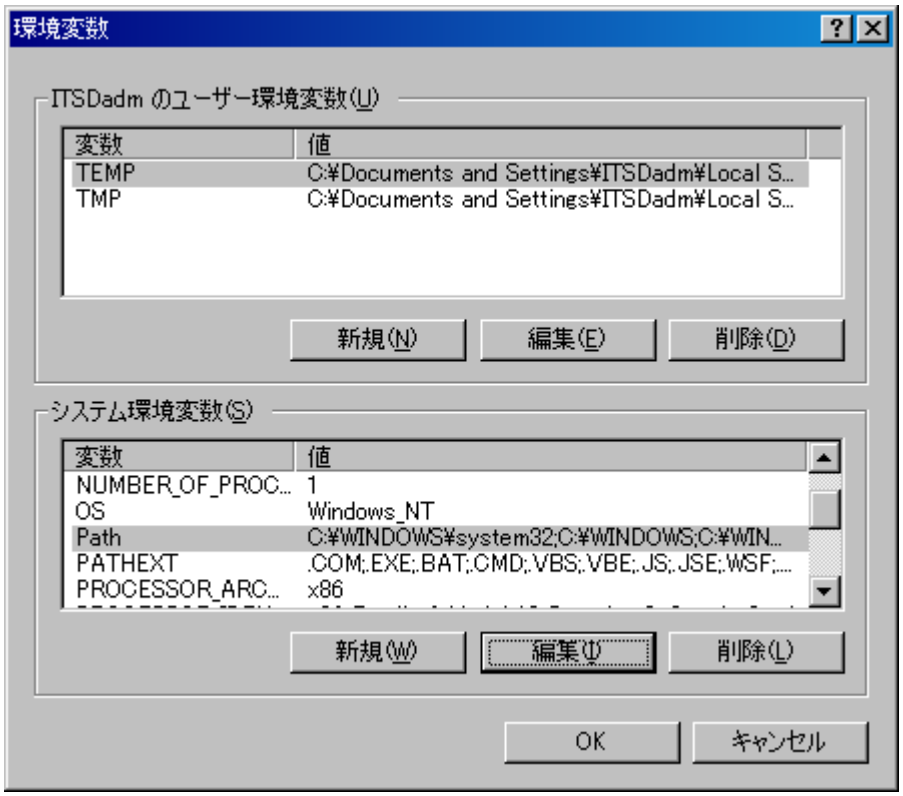
➢ PHP と MySQL 連携の準備

Windows では、PHP をインストールした時点でまだ PHP へのパスが通っていません。これを設定しないと、PHP から MySQL を操作することができないので設定します。パスの確認は以下の通り。コマンドプロンプトを開いて、patch と入力します。すると、今現在の通っているパスが確認できます。ここに、C:¥PHP フォルダのパスを追加します。

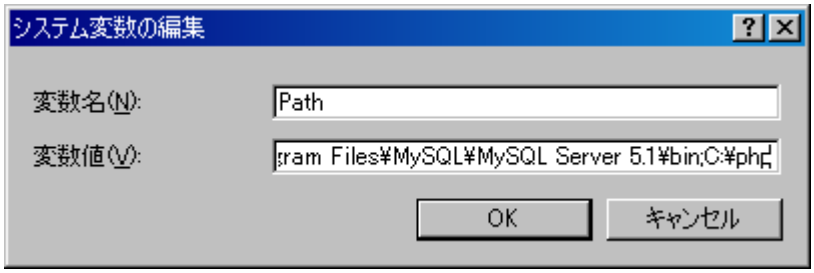


マイコンピュータを右クリックしてプロパティを開きます。

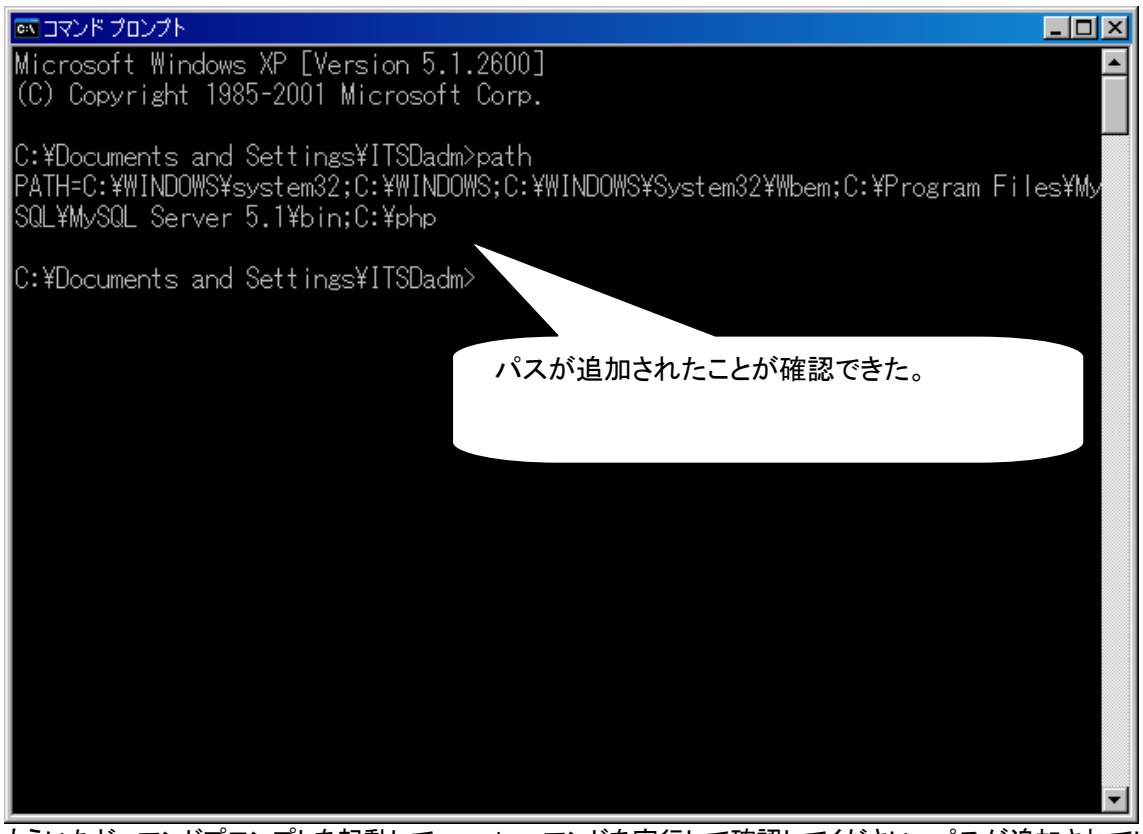




詳細設定、環境設定と進み、Path 編集を選びます。



変数値の後ろに;C:¥PHP（セミコロン PHP のインストールしたパス）と入力します。
これでパスの追加は終わりです。設定を有効にするためにパソコンを再起動してください。



もういちどコマンドプロンプトを起動して patch コマンドを実行して確認してください。パスが追加されていることが確認できます。これで PHP をインストールしたフォルダ内にある libmysql.dll というファイルを Windows が読み込めるようになり、MySQL 関連のオブジェクトが利用することができるようになります。

4.データベース接続の準備

PHP アプリケーションからデータベースを操作するには、まずデータベースに対して接続を確立しなければなりません。データベースへの接続に際して、接続先のデータベースの種類やホスト名などを宣言する文字列情報のことを DSN（Data Source Name : データベース接続文字列）といいます。

PDO 経由でデータベース接続文字列を定義するには、php.ini 上で次のように宣言する必要があります。データベース接続文字列の書式は、接続先のデータベースによって異なるので注意してください。

接続先のデータベース	接続文字列（例）
SQL Server	mssql:host=localhost; dbname=samples
Sybase	sybase:host=localhost; dbname=samples
FireBird	firebird:User=yamada; Password=myPass; Database=samples.gde; DataSource=localhost;Port=3050
MySQL	mysql:host=localhost; dbname=samples; charset=ujis
Oracle	oci:samples (tnsnames.ora 使用時) oci:dbname=//localhost: 1521/samples (instantclient 使用時)
ODBC	odbc:DSN=SAMPLES; UID=yamada; PWD=myPass
PostgreSQL	pgsql:host=localhost port=5432 dbname=samples user=yamada password=myPass
SQLite 3.x	sqlite:samples.sqlite （ファイルシステムにデータベースを作成する場合） sqlite::memory: （メモリ上にデータベースを作成する場合）
SQLite 2.x	sqlite2:samples.sqlite （ファイルシステムにデータベースを作成する場合） sqlite2::memory:: （メモリ上にデータベースを作成する場合）

```
656 pdo.dsn.park="mysql:host=localhost;dbname=park;charset=sjis"↵
```

今回は MySQL への接続なのでこんな感じで extension ブロックの下に入力します。

※park データベースは MySQL の操作 SQL 全般を参照してつくってください。

5.データベースへの接続

さっそく接続をして見ます。PDO では、PDO クラスのコンストラクタに DSN 名を引き渡すだけで、データベースへの接続を確立できます。たとえば、先に php.ini で宣言した DSN 名は park だったので、次のように記述します。

```
$db = new PDO ('park');
```

PDO の構文は次のとおり、

```
PDO PDO::__construct(string dsn [,string username [,string password [,array driver_options]]])
dsn : データベース接続文字列
username : データベース接続時に使用するユーザ名
password : データベース接続時に使用するパスワード
driver_options : データベース固有のオプション
```

前述した表のとおり、データベースごとに接続文字列の書式は違います。ユーザ名、パスワードはデータベース接続文字列に含めることも可能です。データベース接続文字列で明示されている場合、引数ユーザーネーム、パスワードを個別に指定する必要はありません。なにかしらの理由でデータベースへの接続に失敗した場合、PDO クラスは PDOException 例外を発生させることができます。前述したように、PDO では PHP5の例外処理に対応しているのが特徴です。後続の処理で不用意なエラー発生を防ぐためにも try～catch ブロックで接続の成否を確認することは重要です。

・connect_pdo.php

```
<?php
  try {
    $db=new PDO('park','root' ,'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
    print('データベースへの接続を確立しました');
  }catch (PDOException $e) {
    die('エラーメッセージ : '.$e->getMessage());
  }
?>
```

上の例では、何かしらの理由で（たとえば、データベースを新規に生成するための書き込み権限を持たない、など）データベースへの接続を確立できなかった場合、エラーメッセージを表示して、スクリプトの実行を停止します。ちなみに、PDOException クラスで利用できるメソッドには、getMessage メソッドの他に次のようなものがあります。

メソッド	概要
string getFile()	例外が発生したファイル名を取得

int getLine()	例外が発生した行番号を取得
int getCode()	エラーコードを取得
string getMessage()	例外メッセージを取得
string getTraceAsString()	バックトレースを文字列として取得

その他、データベース接続に際して必要なパラメータは、PDO::setAttribute メソッドで設定することが可能です。たとえば、データベースサーバによっては取得したテーブルのフィールド名をそのまま返さず、自動的に大文字（小文字）に変換するものもありますが、そのような挙動の不具合は、時としてプログラマを混乱させます。しかし、あらかじめ接続パラメータを次のように設定しておくことで、こうしたデータベースごとの違いを吸収することが可能となります。

```
$db = new PDO('sample');
$db -> setAttribute(PDO::ATTR_CASE, PDO::CASE_LOWER);
```

PDO::ATTR_CASE はカラム名の処理を決めるための属性を表す定数です。ここでは、データベースから取得したフィールド名を一律、小文字に変換処理（PDO::CASE_LOWER）することを指定しています。その他、setAttribute メソッドで設定可能な接続パラメータについては以降を参照してください。

定数	概要
PDO::ATTR_AUTOCOMMIT	オートコミットを有効化
PDO::ATTR_TIMEOUT	タイムアウト値
PDO::ATTR_ERRMODE	エラーモード
PDO::ATTR_ERRMODE_SILENT	エラー出力を行わない
PDO::ATTR_ERRMODE_WARNING	警告を出力
PDO::ATTR_ERRMODE_EXCEPTION	PDOException 例外を発生
PDO::ATTR_SERVER_VERSION	サーバのバージョン（読み込み専用）
PDO::ATTR_CLIENT_VERSION	クライアントのバージョン（読み込み専用）
PDO::ATTR_SERVER_INFO	サーバ情報（読み込み専用）
PDO::ATTR_CONNECTION_STATUUS	サーバへの接続状態（読み込み専用）
PDO::ATTR_CASE	取得したカラム名を大文字/小文字変換するか
PDO::CASE_LOWER	カラム名を小文字に変換
PDO::CASE_NATURAL	ドライバからの戻り値のまま変換しない
PDO::CASE_UPPER	カラム名を大文字に変換
PDO::ATTR_CURSOR_NAME	カーソル名
PDO::ATTR_ORACLE_NULLS	空文字を null に変換するか
PDO::ATTR_PERSISTENT	持続的接続を有効にするか

エラーモード（PDO::ATTR_ERRMODE）に PDO::ERRMODE_SILENT を指定した場合、PHP はエラー発生時にエラーコードを設定するだけでエラー出力は行いません。もしもこの設定で発生したエラーを確認したい場合には、PDO::errorCode, errorInfo メソッドを使用してください。errorInfo メソッドはエラー情報を配列として返します。

errorInfo に含まれるエラー情報

インデックス	概要
0	共通のエラーコード
1	ドライバ固有のエラーコード
2	ドライバ固有のエラーメッセージ

errorInfo メソッドに含まれる共通エラーコードは、getCode/errorCode メソッドによって返される値と等価です。

6.実践 PHP・データベースに SQL クエリを発行する

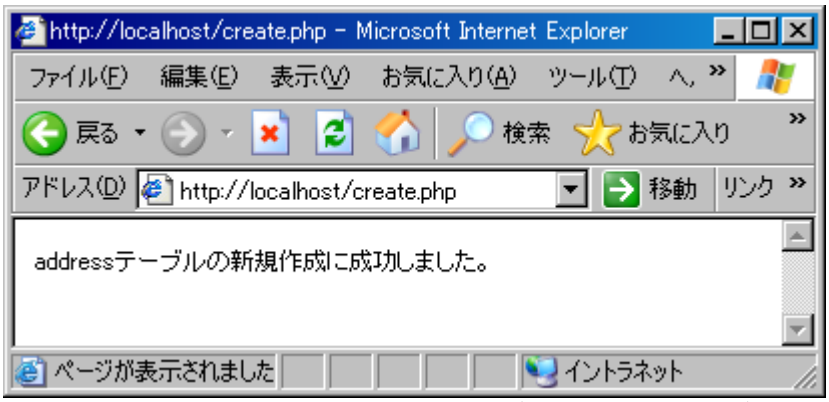
次のコードは、新規にデータベースを作成し、かつ、その配下に address テーブルを定義します。address テーブルのフィールドレイアウトは以下のものをつかいます。

フィールド名	データ型	概要
id	INTEGER	メンバ ID（主キー、連番）
name	VARCHAR(50)	名前
address	VARCHAR(150)	住所
tel	VARCHAR(20)	電話番号
email	VARCHAR(100)	E-Mail アドレス

・create.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->exec('CREATE TABLE address(id INTEGER AUTO_INCREMENT PRIMARY KEY, name VARCHAR(50), address VARCHAR(150), tel VARCHAR(20), email VARCHAR(100))');
    print('address テーブルの新規作成に成功しました。');
}catch (PDOException $e) {
    die('エラーメッセージ：'.$e->getMessage());
}
?>
```

※赤字が実際の SQL 文になります。



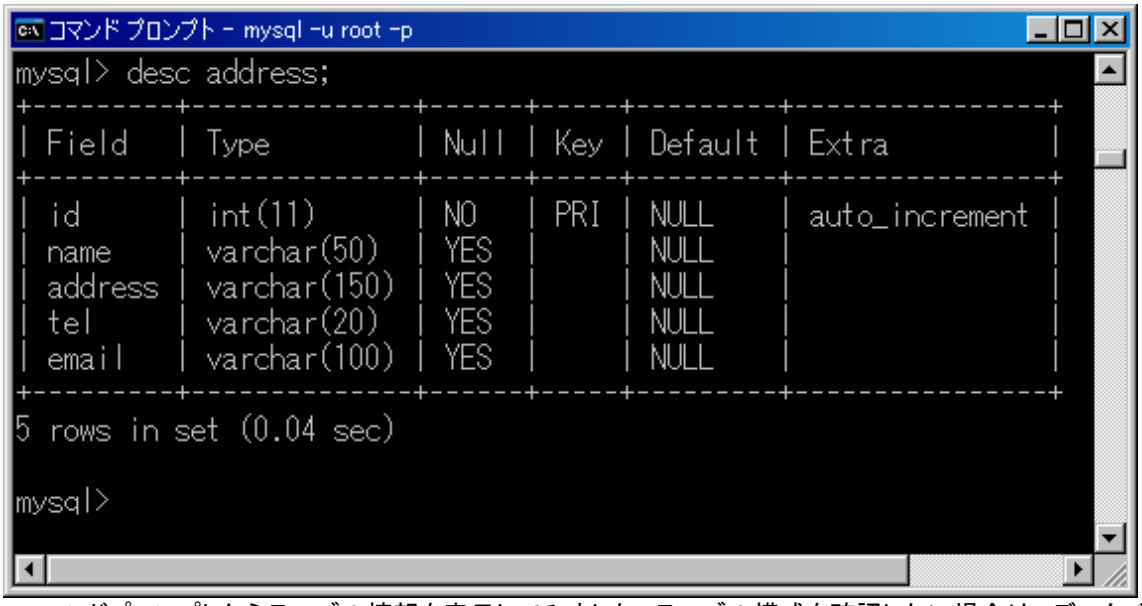
成功しました。というメッセージが表示されれば作成されているはずです。

➢ ちなみにエラーが返ってきたときの意味を書いておきます。

ユーザー名が間違っていた場合
SQLSTATE[28000] [1045] Access denied for user 'testuserr'@'localhost' (using password: YES)

ホスト名が間違っていた場合
SQLSTATE[HY000] [2005] Unknown MySQL server host 'localhosta' (11001)

MySQL のサーバが停止していた場合
SQLSTATE[HY000] [2003] Can't connect to MySQL server on 'localhost' (10061)



コマンドプロンプトからテーブル情報を表示してみました。テーブル構成を確認したい場合は、データベースを選んだ後、desc テーブル名 と入力します。CREATE 文が実行されたことが確認できます。

7.SQL、DML、DDL

データベースへの接続を確立した後は、exec, query メソッドを使用して、SQL クエリを発行することができます。SQL(Structured Query Language)は、リレーショナルデータベースを操作するための標準的な問い合わせ言語で、INSERT(登録)や UPDATE(更新)、DELETE(削除)、SELECT(検索)のようにデータそのものを操作するための DML(Data Manipulation Language : データ操作言語)と、CREATE TABLE(テーブル生成)や ALTER TABLE(構造変更)などのようにデータベースの構造を操作するための DDL(Data Definition Language : データ定義言語)に大別できます。

SQL	
DML	DDL
INSERT UPDATE DELETE SELECT その他	CREATE TABLE ALTER TABLE その他

8.exec メソッド

exec メソッドは指定された SQL クエリを実行し、その結果、影響を受けたレコード数（行数）を戻り値として返します。つまり、exec メソッドはレコードに対してなにかしらの影響を及ぼす INSERT、UPDATE、DELETE のような命令、または、結果を返さない CREATE TABLE のような DDL を発行することができます。SQL クエリが結果を返さない、または、SQL クエリの実行によって影響を受けたレコードが存在しない場合、exec メソッドは 0 を返します。ただし、exec メソッドは SQL クエリが「結果セット」を返す場合には利用できません。つまり、SELECT（検索）命令を発行するには、query メソッドを使用する必要があります。query メソッドは、戻り値として PDOStatement オブジェクトを返します。

exec, query メソッド
long PDO::exec (string statement)
object PDO::query (string statement)
statement : 任意の SQL クエリ

9.実践 PHP・入力値を基にデータベースに登録する

前述で作成した address テーブルにデータを登録してみましょう。次のコードは insert_form.php で入力された内容を insert_process で登録処理します。

・insert_process.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO address(name, address, tel, email) VALUES(" . $db->quote($_POST['name']) . "," . $db->quote($_POST['address']) . "," . $db->quote($_POST['tel']) . "," . $db->quote($_POST['email']) . ")";
    $db->exec("SET NAMES sjis");
    $db->exec($sql);
    header('Location: http://'.$_SERVER['HTTP_HOST'].dirname($_SERVER['PHP_SELF']).'insert_form.php');
}catch (PDOException $e) {
    print('エラー発生:'.$e->getMessage());
}
?>
```

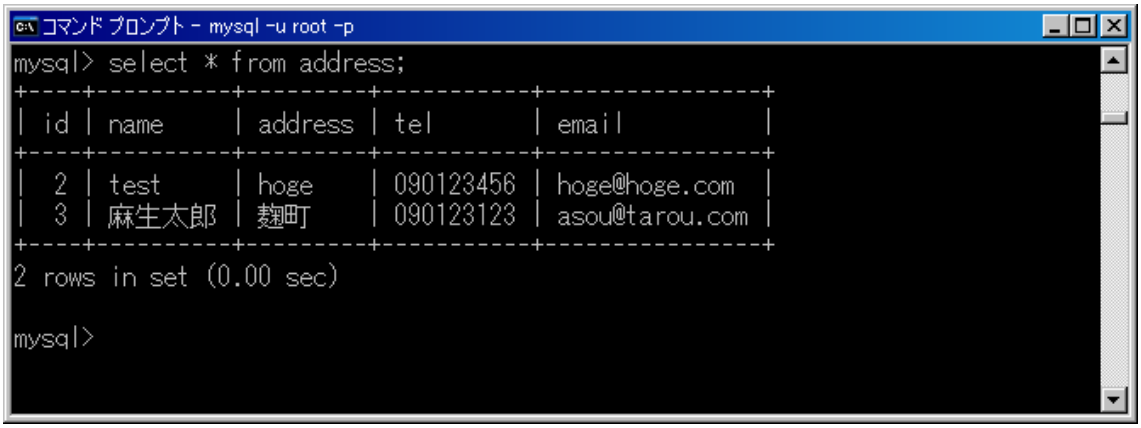
※赤字が実際の SQL 文になります。

※SET NAMES sjis はクライアントのキャラクタセットを SJIS に変更させる SQL です。データベースで他の言語設定をしている場合は変えてください。

・insert_form.php

```
<html>
<head>
<title>住所の登録</title>
</head>
<body>
<form method="POST" action="insert_process.php">
   <table border="0">
      <tr>
         <th align="right">名前:</th>
         <td><input type="text" name="name" size="15" maxlength="50" /></td>
      </tr>
      <tr>
         <th align="right">住所:</th>
         <td><input type="text" name="address" size="35" maxlength="170" /></td>
      </tr>
      <tr>
         <th align="right">電話番号:</th>
         <td><input type="text" name="tel" size="20" maxlength="20" /></td>
      </tr>
      <tr>
         <th align="right">E-Mail アドレス:</th>
         <td><input type="text" name="email" size="50" maxlength="100" /></td>
      </tr>
      <tr>
         <td colspan="2">
            <input type="submit"value="登録" />
            <input type="reset" value="クリア" />
         </td>
         <td></td>
      </tr>
   </table>
</form>
</body>
</html>
```

実行結果は以下の通り、ブラウザから insert_form.php を開き、データを入力して登録を押してみましょう。コマンドプロンプトから確認してデータがきちんと登録されたことを見てみます。



登録は大丈夫なようです。id が2番目からオートインクリメントになっているのは、著者の方でテストのために1番目をデリートしたためです。オートインクリメントとは、値を自動的にプラスしてくれる機能のことです。テーブル作成時に指定したので覚えておきましょう。主キーになるものにオートインクリメントはよく使われます。例えばここでいうと、1番、2 番と登録して1番のレコードを削除してそれからあらたにレコードを登録とすると、id の値は 3 番目になります。今回は成功した例のエビデンスを貼り付けていますが、もしエラーコードがブラウザ上に表示された場合は、そのエラーコードをもとにデバッグしていきます。具体的には返ってきたエラーコードを検索エンジンで検索してみましょう。その他、マニュアルや設定を見直してみてください。

10.quote メソッド

以上、実践でやってもらいましたが、ここでは insert_process.php ソースコードにある quote メソッドについて解説していきたいとおもいます。ユーザ入力から動的に SQL クエリを生成する場合、入力値は必ずクォート処理する必要が出てきます。クォート処理とは、指定された文字列に含まれるシングルクォーテーションをエスケープ処理（無効化）した上で、文字列全体を引用符で括ることをいいます。PDO 関数でクォート処理を担当するのは、quote メソッドの役割です。

quote メソッド

string PDO::quote(string string [, int parameter_type])

string: クォート対象の文字列

parameter_type: パラメータの型（PDO::PARAM_STR | PDO::PARAM_NULL | PDO::PARAM_INT | PDO::PARAM_LOB）

これによって、文字列に不正な文字列（例えば、データを抜き取ったり、消し去ったりする悪質な SQL 文）が故意に入力されたとしても、あらかじめそれらの文字列を無効化することが可能になります。ユーザからの入力文字列を基に SQL クエリを生成する場合には、必ずクォート処理を行うようにしてください。

処理後は、header 関数で insert_from.php へ自動的にページ遷移しています。このような自動リンク（ジャンプ）のことをリダイレクトといいます。リダイレクトを行うには、次の形式で記述してください。

```
hader('Location : <リダイレクト先の URL>')
```

リダイレクト先の URL は「http://」などで始まる絶対 URL で指定してください。絶対 URL は、サンプルでもあるように次の記述で取得することが可能です。

```
http://'.$_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . 'insert_from.php'
```

ホスト名（www など）　ホスト内のパス　リダイレクト先のファイル名

$_SEVER['PHP_SELF']は現在実行中のファイル名を、dirname 関数は指定されたパスの親ディレクトリを取得します。

11.SQL インジェクションとプレイスホルダ

ユーザ入力から動的に SQL クエリを生成する場合、入力値をそのままセットするということはセキュリティの観点から好ましくありません。というのも、入力文字列に SQL クエリの挙動を変更するような予約文字列が含まれていた場合、開発者が想定しない命令が勝手に実行される可能性があるためです。

イメージが湧きにくいと思いますので簡単な例を挙げてみることにします。たとえば、次のようなコードがあると仮定します。次のコードは、ユーザから入力されたユーザ ID/パスワードのセットでユーザテーブルを検索し、ユーザ ID/パスワードの妥当性を確認するものです。

```
$sql="SELECT * usr WHERE id = ' ". $_POST['id'] . "' AND passwd=' " . $_POST['passwd'] . "' ";
```

この SQL 命令に対して、もしもエンドユーザがユーザ ID として「admin';-」という文字列を入力したらどうでしょう。「'」は文字列の終了、「;」は命令の終了、そして「-」はコメントを表す、それぞれ SQL クエリにおける予約語です。ちなみにパスワードは基本的に何を入力してもかまいません。

すると、文字列連結によって生成される SQL クエリは、次のようになります。

```
SELECT * usr WHERE id= 'admin' ;--' AND passwd= 'hogehoge'
```

「;-」以降は無視され、いつのまにかユーザテーブルを id フィールドだけで（passwd フィールドはまったく無関係で）検索していることがわかると思います。もしこのようなコードがログイン処理の局面で使われてしまえば、ユーザ名だけで管理者権限のログインができてしまうことになり、大きなトラブルの原因になります。このようなセキュリティホールのことを「SQL インジェクション」と言い、最近でも、不正アクセス、情報漏えいなどで話題になっています。

このような SQL インジェクションへの対抗手段として有効なのが、前述した quote メソッドです。quote メソッドを使用すると、たとえば上の例では次のような SQL クエリが生成されることになります。

```
SELECT * usr WHERE id= 'admin'' ;--' AND passwd= 'hogehoge'
```

quote メソッドにより「'」を無効化できるので、与えられた文字列がひとつのパラメータであることが保証されるというわけです。

- プレイスホルダの基本

以上のようなクォート処理は SQL インジェクションを防止する上で有効な（かつ基本的な）手法ですが、ひとつ問題もあります。入力データひとつひとつに対してクォート処理を施さなければならないため、自ずとコードが冗長になりやすい点です。また、そもそも文字列結合による SQL クエリの生成は、コードを読みにくくする原因ともなります。そこで、SQL クエリを動的に生成するには、「プレイスホルダ」というしくみを利用するのが一般的です。前述の insert_process.php をプレイスホルダを利用したコードに置き換えてみましょう。

・insert_process2.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
    $sql = $db->prepare('INSERT INTO address(name, address, tel, email) VALUES(?, ?, ?, ?)');
    $db->exec("SET NAMES sjis");
    $sql->execute(array($_POST['name'], $_POST['address'], $_POST['tel'], $_POST['mail']));
    header('Location: http://'.$_SERVER['HTTP_HOST'].dirname($_SERVER['PHP_SELF']).'insert_form2.php');
}catch (PDOException $e) {
    print('エラー発生：'.$e->getMessage());
}
?>
```

※赤字は前のコードからの変更点とプレースホルダコード部分

・insert_form2.php

```
<html>
<head>
<title>住所の登録</title>
</head>
<body>
<form method="POST" action="insert_process2.php">
   <table border="0">
      <tr>
         <th align="right">名前:</th>
         <td><input type="text" name="name" size="15" maxlength="50" /></td>
      </tr>
      <tr>
         <th align="right">住所:</th>
         <td><input type="text" name="address" size="35" maxlength="170" /></td>
      </tr>
      <tr>
         <th align="right">電話番号:</th>
         <td><input type="text" name="tel" size="20" maxlength="20" /></td>
      </tr>
      <tr>
         <th align="right">E-Mail アドレス:</th>
         <td><input type="text" name="email" size="50" maxlength="100" /></td>
      </tr>
      <tr>
         <td colspan="2">
            <input type="submit"value="登録" />
            <input type="reset" value="クリア" />
         </td>
         <td></td>
      </tr>
   </table>
</form>
</body>
</html>
```

prepare メソッドは、データベースサーバに発行するための SQL クエリを準備するメソッドで、一般的には次のような構文で記述します。prepare メソッドは、SQL 命令の準備に成功すると、「プリペアドステートメント（準備済みの SQL）」を表す PDOStatement オブジェクトを返します。

prepare メソッド
object PDO::prepare(string statement [,array driver_options])
statement : 任意の SQL クエリ
driver_options : オプション（キー=>値の形式の連想配列）

prepara メソッドで指定する引数 statement には、「:<name>」、または「?」という形式でプレイスホルダを含めることができます。プレイスホルダとは、その名のとおり、パラメータの置き場所（プレイスホルダ）です。プレイスホルダには、実行時に動的にパラメータを埋め込むことができます。プレイスホルダにパラメータが埋め込まれる際には、パラメータが自動的にクォート処理されるので、前述のようにいちいち quote メソッドによる処理を行う必要がなくコードをよりシンプルに記述できます。

パラメータへの値の引渡しは、PDOStatement::execute メソッドを使用して実行時に行うか、PDOStatement::bindParam メソッドを使用して事前に行う必要があります。execute, bindParam メソッドの一般的な構文は次のとおりです。

bindParam メソッド
bool bindParam(mixed parameter, mixed variable [, int data_type [, int length]])
bool execute([array input_parameters])
parameters : パラメータ名（またはインデックス名）
variable : パラメータ名
data_type : データ型
length : データ長
input_parameters : パラメータ（名前と値のセット）

先ほどの例では execute メソッドを使用してパラメータを引き渡しましたが、パラメータのデータ型やデータ長を明示的に指定したい場合には、bindParam メソッドを使用してください。引数 data_type に指定できる値を以降に示します。

設定値	概要
PDO::PARAM_NULL	NULL 型
PDO::PARAM_INT	整数型
PDO::PARAM_STR	CHAR/VARCHAR など文字列型
PDO::PARAM_LOB	ラージオブジェクト型
PDO::PARAM_STMT	SQL クエリ
PDO::PARAM_INPUT_OUTPUT	入出力パラメータ

先ほどプレイスホルダには「?」と「:<name>」という形式があると述べましたが、「?」のことを「名前なしパラメータ」、「:<name>」のことを「名前付きパラメータ」と言います。

名前なしパラメータはよりシンプルにプレイスホルダを表現できますが、パラメータと値の対応関係がわかりにくくなるという問題があります。一方、名前付きパラメータはややコードは冗長になりますが、パラメータと値の対応関係は明確になります。比較的パラメータ数が少ない SQL クエリに対しては名前なしパラメータを、パラメータ数が多い SQL クエリに対しては名前付きパラメータを使用することをお勧めします。

名前なしパラメータを利用する場合、bindParam メソッドの引数 parameter には 1 をスタートとするインデックス番号を、execute メソッドには数値インデックスを持つ通常配列を指定する必要があります。

たとえば、以下は insert_process2.php を名前なしパラメータと bindParam メソッドを利用して書き換えたものです。

・process_bind1.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
    $sql = $db->prepare('INSERT INTO address(name, address, tel, email) VALUES(?, ?, ?, ?)');
    $db->exec("SET NAMES sjis");
    $sql->bindParam(1, $_POST['name'],PDO::PARAM_STR, 50);
    $sql->bindParam(2, $_POST['address'],PDO::PARAM_STR, 150);
    $sql->bindParam(3, $_POST['tel'],PDO::PARAM_STR, 20);
    $sql->bindParam(4, $_POST['email'],PDO::PARAM_STR, 100);
    $sql->execute();
    header('Location:
http://'.$_SERVER['HTTP_HOST'].dirname($_SERVER['PHP_SELF']).'/insert_form2.php');
}catch (PDOException $e) {
    print('エラー発生：'.$e->getMessage());
}
?>
```

bindParam メソッドを使用した場合は、引き渡すパラメータの数が増えても bindParam メソッドを並列に連ねていくだけなので、よりコードがすっきりと見やすくなっていることが分かるでしょう。今度は、同じスクリプトを名前付きパラメータと bindParam メソッドを使って書き換えてみましょう。

・process_bind2.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
      $sql = $db->prepare('INSERT INTO address(name, address, tel, email)
       VALUES(:name, :address, :tel, :email)');
    $db->exec("SET NAMES sjis");
    $sql->bindParam(':name', $_POST['name'],PDO::PARAM_STR, 50);
    $sql->bindParam(':address', $_POST['address'],PDO::PARAM_STR, 150);
    $sql->bindParam(':tel', $_POST['tel'],PDO::PARAM_STR, 20);
    $sql->bindParam(':email', $_POST['email'],PDO::PARAM_STR, 100);
    $sql->execute();
    header('Location:http://'.$_SERVER['HTTP_HOST'].dirname($_SERVER['PHP_SELF']).'/insert_form2.php');
}catch (PDOException $e) {
    print('エラー発生:'.$e->getMessage());
}
?>
```

名前付きパラメータを使用すると、パラメータと入力値の対応関係がより分かりやすくなります。パラメータの数が多い SQL クエリを生成するようなケースでは、名前付きパラメータを使用することで、コードの可読性を向上させ、コーディングの誤りを減らすことが期待できます。

12.結果セットの取得

結果セットとは、SQL-SELECT 命令によって、1個または複数のテーブルから取り出されたレコード群を保存するために、メモリ上に用意された「仮想テーブル」のことを言います。結果セットは、query メソッド、またはprepare/execute メソッドのセットを利用することで取得できます。WHERE 条件句のパラメータのみを変更して繰り返し実行するようなケース、あるいは、ユーザからの入力値に基づいて動的に SQL クエリを発行するようなケースでは、prepare/execute メソッドを利用するとよいでしょう。前述しましたが、exec メソッドはあくまで「結果を返さない SQL クエリ」を発行するためのメソッドであり、結果セットを返す SELECT 命令を実行する場合は「利用できない」ということに注意してください。

・result.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->exec("SET NAMES sjis");
    $rs=$db->query('SELECT * FROM address ORDER BY id ASC');
    print('<table border="1">');
    print('<tr><th>名前</th><th>住所</th><th>TEL</th><th>E-Mail アドレス</th></tr>');
    while ($row=$rs->fetch(PDO::FETCH_ASSOC)){
?>
    <tr>
        <td><?php print($row['name']); ?></td>
        <td><?php print($row['address']); ?></td>
        <td><?php print($row['tel']); ?></td>
        <td><?php print($row['email']); ?></td>
    </tr>
<?php
    }
}catch (PDOException $e) {
    print('エラーメッセージ:'.$e->getMessage());
}
                                                    ?>
```

名前	住所	TEL	E-Mailアドレス
test	hoge	090123456	hoge@hoge.com
麻生太郎	麹町	090123123	asou@tarou.com

取得した結果セットからデータを取り出すことを「フェッチする」と言います。データをフェッチするために、PDOStatement オブジェクトには次の3つのメソッドが用意されています。

メソッド	概要
fetch (int mode)	結果セットから次の行を取得
fetchAll (int mode)	結果セットからすべての行を含むデータを取得
fetchColumn (int col_num)	結果セットの最初のフィールドからデータを取得

最も一般的に利用されるのは、fetch メソッドでしょう。fetch メソッドは（デフォルトで）次の行を指定のフェッチモードで取り出します。fetch メソッドは、次の行が存在しない場合に FALSE を返します。result.php では fetch メソッドのこの性質を利用して、fetch メソッドが FALSE を返すまで while ループを繰り返すことで、結果セット内のすべての値を取得しています。

結果セット内の現在のレコードのことを「カレントレコード」、現在位置を示す内部的な情報のことを「レコードポインタ」と言います。フェッチモードとは結果セットから取り出したデータをどのような形式の変数に格納するかを表すもので、次のような値を設定することができます。

定数	概要	コード（例）
PDO::FETCH_NUM	一般配列にフェッチ	$row[0]
PDO::FETCH_ASSOC	連想配列にフェッチ	$row['name']
PDO::FETCH_OBJ	オブジェクトにフェッチ	$row->name
PDO::FETCH_BOTH	通常/連想配列にフェッチ（デフォルト）	$row[0]/$row['name']
PDO::FETCH_BOUND	フィールド値は bindColumn メソッドで個別にバインド	$name
PDO::FETCH_CLASS	指定クラスにフェッチ	$row->$name
PDO::FETCH_INTO	指定インスタンスに対してフェッチ	$row->$name

たとえば、フェッチモードが PDO::FETCH_NUM の場合は各フィールドの値を「$row[0]」のようにフィールド番号で取り出すことができますし、PDO::FETCH_ASSOC の場合は「$row['name']」のようにフィールド名をキーにして取り出し、PDO::FETCH_OBJ の場合は「$row->name」のようにフィールド名をプロパティとして取り出します。PDO::FETCH_BOUND の場合は取得したパラメータを PDOStatement::bindColumn メソッドで個別にバインドする必要があります。

- bindColumn メソッド

bool PDOStatement::bindColumn(mixed column ,mixed param [,int type [,int maxlen [,mixed driver_options]]])
coloumn : 結果セット内のフィールド番号、またはフィールド名
param : バインドする変数
type : パラメータのデータ型
maxlen : パラメータの最大データ長
driver_options : ドライバ固有のオプション

フェッチモードは、個別のフェッチ関数で実行のたびに指定するほか、PDOStatement::setFetchMode メソッドでデフォルト値を宣言することも可能です。

- setFetchMode メソッド

bool PDOStatement::setFetchMode(int mode [,string clazz | object obj])
mode : フェッチモード
clazz : フェッチするクラス名（フェッチモードが PDO::FETCH_CLASS の場合）
obj : フェッチするインスタンス（フェッチモードが PDO::FETCH_INTO の場合）

fechAll メソッドは、結果セットに含まれるデータを多次元配列としてフェッチするためのメソッドです。fechAll メソッドは、得られる結果セットに対してランダムにアクセスしたい場合などに便利なメソッドですが、反面、結果セットの内容をいったん配列にコピーしなければならないため結果セットが大きい場合はメモリを消費しやすいという問題もあります。特別な目的がない場合には、fetch メソッドを使用することをお勧めします。

そして、最後のフェッチメソッドが fetchColumn メソッドです。fetchColumn メソッドは結果セット内の特定のカラム（デフォルトは先頭カラム）をフェッチするためのメソッドです。SQL 集計関数を利用しているなどで、最初から結果セットに1行しか存在していないことが分かっている場合には、より直感的にデータを取り出すことができます。

13.さまざまなフェッチの利用方法

- fetchAll メソッドで特定カラムをリスト出力する

  fetchAll メソッドを使って、address テーブルから名前と住所のセットを取り出し、リスト出力します。

・fetchAll.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->exec("SET NAMES sjis");
    $db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
    $rs=$db->query('SELECT name, address FROM address ORDER BY id ASC');
    $data=$rs->fetchAll(); //全レコードを配列として取得
    print('<ul>');
    foreach($data as $column){  //配列の内容を順に出力
        print('<li>');
        print($column['name'].' : '.$column['address']);
        print('</li>');
    }
    print('</ul>');
} catch (PDOException $e) {
    print('エラーメッセージ:'.$e->getMessage());
}
?>
```

➢ fetchColumn メソッドで特定カラムの平均値を出力する

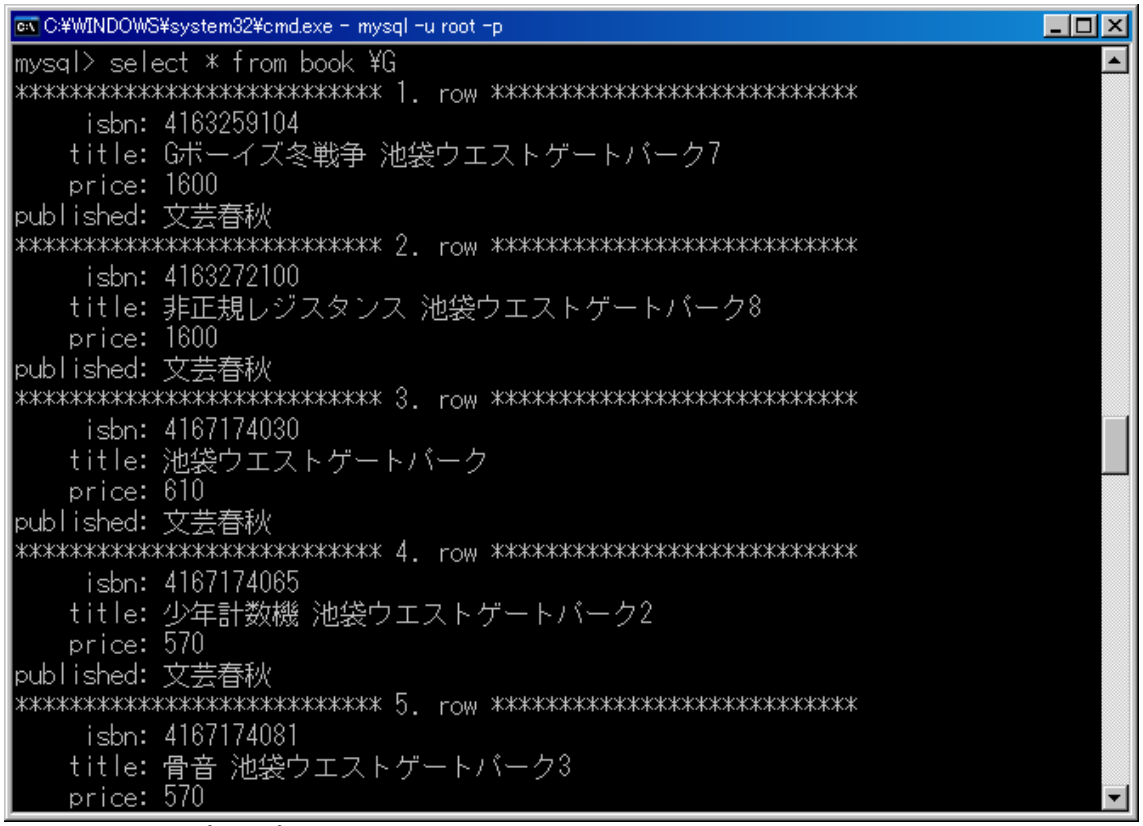
fetchColumn メソッドを使って、book テーブルに含まれる書籍単価の平均を求め、出力します。book テーブルのフィールドレイアウトは下記のとおりです。

フィールド名	データ型	概要
isbn	VARCHAR(25)	ISBN コード
title	VARCHAR(100)	書籍名
price	INT	価格
published	VARCHAR(20)	出版会社

・book_create.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->exec('CREATE TABLE book(isbn VARCHAR(25) PRIMARY KEY, title VARCHAR(100), price INT(10), published VARCHAR(20))');
    print('book テーブルの新規作成に成功しました。');
    $db->exec("SET NAMES sjis");
    $sql = $db->prepare('INSERT INTO book(isbn, title, price, published) VALUES(?, ?, ?, ?)');
    $sql->execute(array('4167174030', '池袋ウエストゲートパーク', 610, '文芸春秋'));
    $sql->execute(array('4167174065', '少年計数機 池袋ウエストゲートパーク 2', 570, '文芸春秋'));
    $sql->execute(array('4167174081', '骨音 池袋ウエストゲートパーク 3', 570, '文芸春秋'));
    $sql->execute(array('416717409X', '電子の星 池袋ウエストゲートパーク IV', 540, '文芸春秋'));
    $sql->execute(array('416717412X', '反自殺クラブ 池袋ウエストゲートパーク 5', 520, '文芸春秋'));
    $sql->execute(array('4167174138', '灰色のピーターパン 池袋ウエストゲートパーク 6', 570, '文芸春秋'));
    $sql->execute(array('4163259104', 'G ボーイズ冬戦争 池袋ウエストゲートパーク 7', 1600, '文芸春秋'));
    $sql->execute(array('4163272100', '非正規レジスタンス 池袋ウエストゲートパーク 8', 1600, '文芸春秋'));
    print('book テーブルのレコード追加に成功しました。');
}catch (PDOException $e) {
    die('エラーメッセージ：'.$e->getMessage());
}
?>
```

以上のソースを実行してまとめてレコードを登録してみましょう。



ちなみにコマンドプロンプトから確認するとこんな感じになります。SELECT 文の最後に￥G(バックスラッシュ＋G キー)を追加することで、結果レコードを縦表示にすることができます。

・fetchColumn.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql=$db->prepare('SELECT AVG(price) FROM book');
    $sql->execute();
    print('平均額:'.$sql->fetchColumn().'円');
} catch (PDOException $e){
    print('エラーメッセージ:'.$e->getMessage());
}
?>
```

- **フェッチモード PDO::FETCH_BOUND を使用する**

result.php をフェッチモード PDO : : FETCH_BOUND を使用して取得してみましょう。

bindColumn メソッドを利用することで、ここのフィールド値を指定された変数に明示的に割り当てることができます。bindColumn メソッドによる割り当ては、可搬性を最大化するため必ず execute メソッドを実行した後に行うべきです。

・bindColumn.php

```
<table border="1">
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->exec("SET NAMES sjis");
    $sql=$db->prepare('SELECT * FROM address ORDER BY id ASC');
    $sql->execute();
    $sql->bindColumn(1, $name, PDO::PARAM_STR, 50);
    $sql->bindColumn(2, $address, PDO::PARAM_STR, 150);
    $sql->bindColumn(3, $tel, PDO::PARAM_STR, 20);
    $sql->bindColumn(4, $email, PDO::PARAM_STR, 100);
    while ($row=$sql->fetch(PDO::FETCH_BOUND)) {
?>
    <tr>
      <td><?php print($name); ?></td>
      <td><?php print($address); ?></td>
      <td><?php print($tel); ?></td>
      <td><?php print($email); ?></td>
    </tr>
<?php
    }
    print('</table>');
}catch (PDOException $e) {
    print('エラーメッセージ : '.$e->getMessage());
}
?>
</table>
```

14.トランザクション処理

トランザクション処理とは、関連する複数の処理をクループ化したもの、と考えればよいでしょう。たとえば、銀行でのお金の振込みを想定してみましょう。振込み処理は、大きく次の2つの処理から構成されます。

- 振り込み元口座の出勤
- 振り込み先口座への入金

もしも、この振り込み処理で振り込み元からの出金には成功したのに、（システム障害などが原因で）振り込み先への入金が失敗してしまったとしたらどうでしょう。振り込みもとの残高は減っているのに、振り込み先の残高は増えていないというおかしなことになってしまいます。逆に、振り込み先の入金は成功したのに、振り込み元からの出金が失敗したとしたら、振り込み元の残高は減らないのに、振り込み先の残高だけが増えるという、ユーザにとっては嬉しいことが起こってしまいます。ただ、こうした不整合は、システム的には絶対にあってはならない問題です。入金/出金というふたつの処理はかならず「両方ともに成功」するか、さもなければ「両方とも失敗」すなければなりません。

ここで登場するのが「トランザクション処理」という考え方です。先ほど述べたように、トランザクション処理とは関連する複数の処理をひとつにグループ化するためのしくみです。つまり、ここでは入金処理と出金処理とをひとつのトランザクションとしてみなします。トランザクションでは、最初に行われる（たとえば）出金処理をその段階では確定せず、仮に登録された状態とみなします。そして、入金処理が成功したタイミングで初めて入金/出金という双方の処理を確定します。このような確定処理のことを「コミット（Commit）」と言います。

ちなみに、出金あるいは入金処理が失敗した場合には、トランザクションは仮登録の状態になっている処理を元に戻します。このような処理のことを「ロールバック（RollBack）」と言い、トランザクションに属するすべての処理を「なかったこと」にします。

トランザクションを利用することで、すべての処理が「すべて成功」か「すべて失敗」であることを保証できるというわけです。複数の関連する処理を行う場合、トランザクションの利用は必須です。

15.トランザクションの活用

トランザクションを利用する場合、PDO::exec メソッドで「BEGIN」、「COMMIT」、「ROLLBACK」といった SQL コマンドを直接に発行することも可能です。

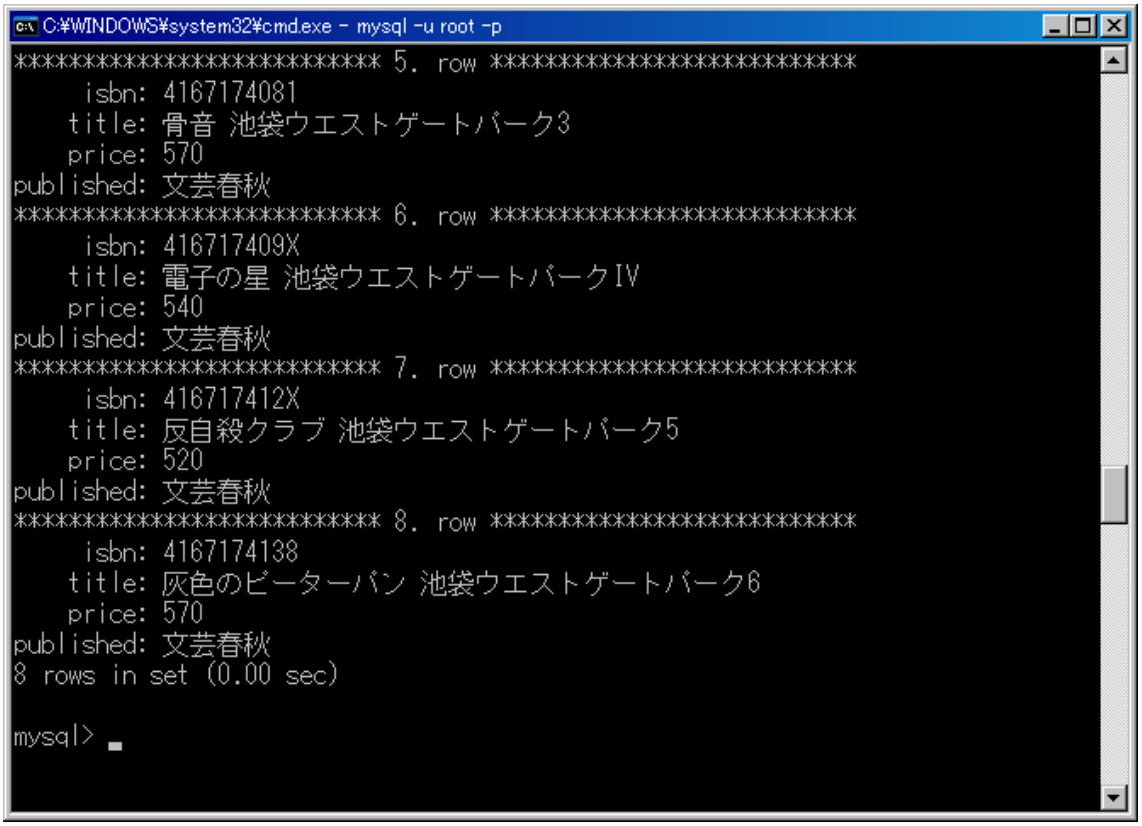
たとえば、次のコードは、トランザクションを利用して2つの INSERT 命令を発行する例です。次の例では、1つ目の命令は成功しますが、2つ目の命令がキー重複のために失敗します。その結果、ROLLBACK 命令が呼び出され、命令は2つともロールバックされます。そのため、transaction.php を実行した後、データベースの中身を確認しても、データの更新は反映されていないことが確認できるはずです。ちなみに、いずれかの値を変更すると、2つの命令は正しく実行され、COMMIT 命令が呼び出されます。データベースの中身を確認すると、2件のデータが追加されているのが確認できるようになります。

・transaction.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->exec("SET NAMES sjis");
    $db->exec('BEGIN');   //トランザクションの開始
    $db->exec('INSERT INTO book(isbn) VALUES("0000-0000-00-0")');
    $db->exec('INSERT INTO book(isbn) VALUES("0000-0000-00-0")');
    $db->exec('COMMIT');   //トランザクションの確定
    print('登録に成功しました。');
}catch (PDOException $e) {
    $db->exec('ROLLBACK');   //トランザクションのキャンセル
    print('エラーページ:'.$e->getMessage());
}
?>
```

```
エラーページ:SQLSTATE[23000]: Integrity constraint violation: 1062 Duplicate entry '0000-0000-00-0' for key
'PRIMARY'
```

エラーページが表示されたのでコンソールからチェックしてみましょう。



こんなふうにエラー発生したため、例外処理により ROLLBACK が実行され処理を取り消すことができます。いまいちわからないという方は、一連のトランザクションの処理をしている行を取り外して再度、実行して確認してみてください。違いがわかると思います。

しかし、PDO には専用の PDO::beginTransaction, commit, rollBack メソッドが用意されています。PDO を利用している場合には、コードの可読性を良くするという意味でも、こちらを優先して利用することをお勧めします。以下は、専用のメソッドを利用して、上の transaction.php を書き換えてみたものです。

・transaction2.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->exec("SET NAMES sjis");
    $db->beginTransaction();   //トランザクションの開始
    $db->exec('INSERT INTO book(isbn) VALUES("0000-0000-00-0")');
    $db->exec('INSERT INTO book(isbn) VALUES("0000-0000-00-0")');
    $db->commit();   //トランザクションの確定
    print('登録に成功しました。');
}catch (PDOException $e) {
    $db->rollback();
    print('エラーページ:'.$e->getMessage());
}
?>
```

beginTransaction メソッドは BEGIN 命令に対応するメソッドであり、トランザクションを開始します。トランザクションを開始しない場合、発行された SQL クエリは直接にデータベースに反映されるので、注意してください。トランザクションが成功した場合には commit メソッドで変更を確定し、失敗した場合には rollback メソッドで変更をロールバックしてください。commit, rollback メソッドが発行されるとトランザクションは終了し、次にトランザクションを開始する際には、もう一度、beginTransaction メソッドを呼び出さなければなりません。

16.メタデータ

データベースとは、ただ単にデータを漠然と蓄積するものではありません。データを体系的に格納し、また必要なときに効率よく取り出せるように、データを規定するさまざまな構成情報を盛っています。このような「データの構成を表すためのデータ」のことを「メタデータ」または、「メタ情報」と言います。

PDO では、結果セットのメタデータを取得するために、PDOStatement::getColumnMeta メソッドを用意しています。getColumnMeta メソッドの構文は、次のとおりです。

getColumnMeta メソッド
mixed PDOStatement::getColumnMeta (int column)
column : カラム番号

getColumnMeta メソッドは結果セット内の指定されたカラムに関する情報を連想配列として返します。getColumnMeta メソッドが返す連想配列に含まれる要素は以下のとおりです。

要素	概要
native_type	データ型
driver.decl_type	SQL データ型
flags	フィールドを規定する全フラグ
name	フィールド名
len	フィールドのデータ長（浮動小数点以外は－1）
precision	フィールドの数値精度（浮動小数点数以外は 0）
pdo_type	データ型（PDO::PARAM_*定数）

要求されたフィールドが結果セットに存在しない場合、またはそもそも結果セットが存在しない場合、getColumnMeta メソッドは FALSE を返します。

➢ テーブル情報の取得

getColumnMeta メソッドを使用して、address テーブルに含まれるフィールド情報を一覧表示してみましょう。

・meta.php

```
<?php
try {
    $db=new PDO('park', 'root', 'passwd');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $rs=$db->query('SELECT * FROM address');
    print('<table border="1">');
    print('<tr><th>フィールド名</th><th>データ型</th><th>サイズ</th></tr>');
    for($i=0;$i<$rs->columnCount();$i++){
        $data=$rs->getColumnMeta($i);   //$i 列目のメタ情報を取得
?>
    <tr>
      <!-- メタ情報を出力 -->
      <td><?php print($data['name']); ?></td>
      <td><?php print($data['native_type']); ?></td>
      <td><?php print($data['len']); ?></td>
    </tr>
<?php
    }
        print('</table>');
}catch (PDOException $e) {
    print('エラーメッセージ:'.$e->getMessage());
}
?>
```

フィールド名	データ型	サイズ
id	LONG	11
name	VAR_STRING	50
address	VAR_STRING	150
tel	VAR_STRING	20
email	VAR_STRING	100

ColumnCount メソッドは結果セット内に含まれるフィールドの数を返すためのメソッドで、ここでは 0 番目から「フィールド数－1」番目までのフィールド情報を順番に出力しているわけです。ちなみに、結果セット内に含まれるレコード数を取得するには、rowCount メソッドを使用してください。