Konstruktor klasy PDO

Konstruktor klasy PDO przyjmuje następujące parametry:

```
$dsn =
'rodzaj_bazy:host=adres_serwera;port=port;encoding=kodowanie_znaków
; dbname=nazwa_bazy';

$dbh = new PDO($dsn, $user,  $pass,  $opcje);
```

gdzie:

- $dsn - podstawowe parametry połączenia
- $user – nazwa użytkownika bazy
- $pass – hasło użytkownika bazy
- $opcje – tablica opcji [argument nie wymagany]

Należy zwrócić uwagę na składnię - najpierw podajemy rodzaj bazy (np. mysql), następnie po dwukropku podajemy konfigurację połączenia, gdzie parametry rozdzielamy średnikiem. Argument $opcje nie jest wymagany, jednakże warto zapoznać się z listą opcji jakie możemy tutaj ustawić. Wszystkie opisane zostały dokładnie w manualu, który znajduje się tutaj: http://php.net/manual/pt_BR/pdo.setattribute.php

Opcje dla połączenia możemy również ustawić później, służy do tego metoda statyczna:

```
PDO::setAttribute ($opcja, $wartość);
```

Warto jednak pamiętać, iż opcja ustawiona za pomocą setAttribute() zaczyna działać od miejsca umieszczenia jej w kodzie.

Wartość ustawionego atrybutu pobieramy natomiast za pomocą:

```
$wartość = PDO::getAttribute ($opcja);
```

Połączenie

Przykładowy kod połączenia z bazą danych może wyglądać tak jak na listingu poniżej.
Dane konfiguracyjne zapisane zostają tutaj przykładowo w tablicy $db_config:

```
<?php
$db_config = array(
        'host' => 'localhost',
        'port' => '3306',
        'user' => 'root',
        'pass' => 'password',
        'db' => 'myDatabase',
        'db_type' => 'mysql',
        'encoding' => 'utf-8'
  );
// próba połączenia
try
{
 $dsn = $db_config['db_type'] .
 ':host=' . $db_config['host'] .
 ';port=' . $db_config['port'] .
 ';encoding=' . $db_config['encoding'] .
 ';dbname=' . $db_config['db'];
 // opcje, tutaj ustawienie trybu reagowania na błędy
 $options = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
 // tworzymy obiekt klasy PDO inicjując tym samym połączenie
 $dbh = new PDO($dsn, $db_config['user'],  $db_config['pass'],
$options);

 // w przypadku błędu, poniższe się już nie wykona:


 define('DB_CONNECTED', true);
 echo '<h1>Connection success!</h1>';
  // łapiemy ewentualny wyjątek:
} catch(PDOException $error)
{
die('Unable to connect: ' . $error->getMessage());
}
?>
```

Jeśli uda się nam połączyć z bazą to wyświetlona zostanie informacja Connection success! oraz zadeklarowana zostanie stała DB_CONNECTED *o* wartość TRUE, jeśli nie uda się uzyskać połączenia – rzucony zostanie wyjątek klasy PDOException i wyświetlony zostanie komunikat błędu.

Uwaga: za pomocą opcji:

```
$options = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
```

Ustawiliśmy tutaj tryb reagowania na błędy. Bardzo ważnym jest ustawienie tej opcji na rzucanie wyjątków, gdyż bez tego ustawienia w przypadku błędu PDO może nam wyrzucić na ekran informacje takie jak szczegółowe parametry połączenia (login do bazy etc...).

Możliwe ustawienia opcji PDO::ATTR_ERRMODE to:

- PDO::ERRMODE_SILENT - zwraca jedynie kody błędów
- PDO::ERRMODE_WARNING -wyświetla ostrzeżenie E_WARNING.
- PDO::ERRMODE_EXCEPTION - rzuca wyjątek

Try, catch

Bardzo ważną rzeczą związaną z powyższym jest, aby kod połączenia umieścić w bloku try{} i wyłapać ewentualnie rzucony wyjątek – w przeciwnym razie klasa PDO może wyrzucić na ekran parametry połączenia, a więc zdradzić informacje, takie jak nazwa użytkownika bazy i jego hasło.

```
try
 {
    // kod połączenia
 } catch(PDOException $error)
 {
  die('Komunikat błędu: ' . $error->getMessage());
 }
```

Rozłączenie

Rozłączenie z bazą jest banalnie proste i polega na przypisaniu wartości NULL do obiektu:

```
$dbh = null;
```

Obsługiwane bazy danych

Poniżej pełna lista wspieranych na chwilę obecną baz danych, zaczerpnięta z manuala:

- CUBRID - cubrid:
- MS SQL Server - dblib:
- Firebird - firebird:
- IBM - ibm:
- Informix - informix:
- MySQL - mysql:
- MS SQL Server (PDO) - sqlsrv:
- Oracle - oci:
- ODBC i DB2 - odbc:
- PostgreSQL - pgsql:
- SQLite - sqlite:
- 4D - 4d:

Wysyłanie zapytania - query()

Wysłanie zapytania do bazy może odbyć się na dwa sposoby. Pierwszym jest standardowa metoda query(), która wysyła do bazy całe zapytanie. Jest to najkrótsza metoda do szybkiego pobrania rekordów. Tą metodę przedstawię jedynie w celu pokazania, że takowa jest. Podczas pracy nie będziemy z niej korzystać. Dlaczego? O tym za chwilę. Na początek pobierzmy kilka rekordów za pomocą zwykłego zapytania. Pobierzemy je za pomocą prostej metody:

```
$dbh->query($zapytanie)
```

Załóżmy teraz, że naszym celem jest pobranie z tabeli wszystkich rekordów zawierających informacje o klientach. Korzystając z query() zapytanie wyglądać będzie następująco:

```
$stmt = $dbh->query('SELECT * FROM customers');
```

Metoda $dbh->query() zwraca jako zbiór wyników zapytania obiekt klasy PDOStatement (stąd skrótowa nazwa obiektu - $stmt). Po obiekcie tym możemy się przelecieć jak po zwykłej tablicy:

1) za pomocą pętli foreach:

```
foreach($stmt as $row)
{
   echo 'id: ' . $row['customer_id']  .
   ', imię:' . $row['first_name']  .
   ', nazwisko:' . $row['last_name'] .
   '<br />';
 }
```

2) za pomocą pętli while i metody $stmt->fetch():

```
while($row = $stmt->fetch())
 {
   echo 'id: ' . $row['customer_id']  .
   ', imię:' . $row['first_name']  .
  ', nazwisko:' . $row['last_name'] .
  '<br />';
 }
```

W obu powyższych rozwiązaniach wylistowana zostanie lista wszystkich użytkowników zapisanych w tabeli customers.

Przykład kodu:

```
<?php
// konfiguracja
$conn_config = array(
'host' => 'localhost',
'port' => '3306',
'user' => 'root',
'pass' => 'password',
'db' => 'myDatabase',
'db_type' => 'mysql',
'encoding' => 'utf-8'
);

// próba połączenia

try
{
$dsn = $conn_config['db_type'] .
':host=' . $conn_config['host'] .
';port=' . $conn_config['port'] .
';encoding=' . $conn_config['encoding'] .
';dbname=' . $conn_config['db'];

$dbh = new PDO($dsn, $conn_config['user'],  $conn_config['pass']);

// ustawienie trybu raportowania błędów

$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
define('DB_CONNECTED', true);
echo '<h1>Connection success!</h1>';

// zapytanie do bazy

$stmt = $dbh->query('SELECT * FROM customers');
```

```
// pobranie/wyświetlenie wyników

while($row = $stmt->fetch())
{
echo 'id: ' . $row['customer_id']  .
', imię:' . $row['first_name']  .
', nazwisko:' . $row['last_name'] .
'<br />';
}

// łapiemy ewentualny wyjątek:

} catch(PDOException $e)
{
  die('Unable to connect: ' . $e->getMessage());
}
```

Wynik:

```
id: 0, imię: Jan, nazwisko: Kowalski

id: 1, imię: Paweł, nazwisko: Nowak

[...]
```

Pobieranie wierszy

a) Metoda fetch()

Jak już zapewne zauważyliśmy w kodzie, w PDO do pobrania rekordu służy metoda:

```
$stmt->fetch()
```

Domyślnie pobiera ona wiersz jako tablicę asocjacyjną + index. Pozwala ona jednak na lekkie przekonfigurowanie swojej pracy, np. tak, aby zamiast tablicy zwracała nam obiekty, gdzie każdy wiersz (rekord) to oddzielny obiekt, a kolumny tabeli to jego właściwości.

Ustawienia typu zwracanych przez fetch() danych dokonujemy za pomocą metody:

```
$stmt->setFetchMode($parametr);
```
lub
```
$dbh->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, $parametr);
```

gdzie $parametr może przybrać następujące wartości:

- PDO::FETCH_ASSOC - zwraca wiersz jako tablicę asocjacyjną, gdzie kluczem są nazwy kolumn
- PDO::FETCH_NAMED - podobnie jak FETCH_ASSOC, ale w przypadku kilku kolumn o tych samych nazwach zwraca tablicę ze wszystkimi wartosciami
- PDO::FETCH_NUM - zwraca wiersz jako tablicę z indeksem numerycznym
- PDO::FETCH_BOTH - zwraca wiersz jako tablicę z indeksem nazwowym oraz numerycznym (ta opcja jest domyślna)
- PDO::FETCH_OBJ - zwraca wiersz jako anonimowy objekt ustawiając jego właściwości na wartości kolumn
- PDO::FETCH_LAZY - połączenie FETCH_BOTH i FETCH_OBJ - tworzy właściwości obiektu nazywając właściwości obiektu jako nazwy kolumn lub za pomocą indeksu numerycznego, tak jak zostają wywołane
- PDO::FETCH_BOUND - zwraca wartości kolumn do zmiennych ustawionych za pomocą bindValue()
- PDO::FETCH_CLASS - zwraca wiersz jako instancję zadanej klasy, ustawiajac pola jako wlaściwości obiektu
- PDO::FETCH_INTO - aktualizuje obiekt zadanej klasy ustawiajac odpowiednie właściwości

Ustawienie danej opcji wpływa na metodę fetch() definiując rodzaj zwracanych przez nią danych od miejsca ustawienia opcji. Domyślnie ustawiona jest opcja FETCH_BOTH, czyli zwracamy tablicę asocjacyjną oraz z indeksem, co za tym idzie, do wiersza możemy dostać się za pomocą:

```
$row['first_name']
```

ale możemy też i poprzez index numeryczny:

```
$row[3]
```

W przypadku natomiast obiektu (PDO::FETCH_OBJ):

```
$row->first_name
```

Parametr możemy też podać bezpośrednio jako argument metody fetch():

```
while($row = $stmt->fetch(PDO::FETCH_ASSOC))
 {
  echo 'id: ' . $row['customer_id']  .
  ', imię:' . $row['first_name']  .
  ', nazwisko:' . $row['last_name'] .
  '<br />';
 }
```

b) Metoda fetchAll()

Różnica pomiędzy metodą fetch(), a fetchAll() jest taka, iż o ile metoda fetch() pobiera za jednym razem jedynie jeden wiersz, na który ustawiony jest kursor, to metoda fetchAll() pobiera na raz do tablicy wszystkie wiersze. Przy dużej ilości wierszy może to więc powodować niemałe obciążenie.

Przyładowo, metody fetchAll() użyć możemy tak:

```
$result = $stmt->fetchAll();

foreach($result as $row)
  {
   echo 'id: ' . $row['customer_id']  .
   ', imię:' . $row['first_name']  .
   ', nazwisko:' . $row['last_name'] .
   '<br />';
  }
```

Wysyłanie zapytania - prepare() i execute()

I w tym miejscu doszliśmy do miejsca, gdzie żegnamy się już całkowicie z metodą query().
Na czym polega różnica pomiędzy metodą query(), a metodą prepare()? Za pomocą:

```
$stmt = $dbh->prepare($query);
```

nie wysyłamy całego zapytania od razu do bazy, a jedynie definiujemy jego szkielet.
Zapytanie takie wykonywane jest dopiero po wywołaniu metody:

```
$stmt->execute();
```

Konstrukcja taka pozwala na kilka ciekawych zabiegów, takich jak parametryzowanie zapytań, które omówimy w następnym artykule, a także sprawia, iż oszczędzamy na czasie połączenia z serwerem bazy. Zapomnijmy o query().

Zwolnienie kursora - closeCursor()

Teraz bardzo ważna sprawa. Wartością zwracaną przez zapytanie jest zbiór wyników będący obiektem klasy PDOStatement. Specyfika działania PDO pozwala na jednoczesną pracę jedynie na JEDNYM otwartym zbiorze. Czym jednak jest ten tzw. kursor? Otóż w momencie pobierania wyników ze zbioru, np. w pętli za pomocą metody fetch() - podczas każdego kolejnego przebiegu pętli ustawiany jest wskaźnik, czyli nasz kursor. Wskaźnik ten ustawiany jest na kolejny rekord ze zbioru - dzięki temu mozemy pobierać kolejne rekordy w pętlach. Chcąc więc pracować na kolejnym zbiorze wyników zwróconym przez kolejne zapytanie musimy ten kursor zwolnić i ustawić go z

powrotem na pierwszy element z nowego zbioru wyników. Służy do tego metoda:

`$stmt->closeCursor():`

którą wywołujemy ZAWSZE po skończonej pracy na danym zbiorze wyników.

Przykład użycia closeCursor():

```
<?php

// konfiguracja

$conn_config = array(
 'host' => 'localhost',
 'port' => '3306',
 'user' => 'root',
 'pass' => 'password',
 'db' => 'myDatabase',
 'db_type' => 'mysql',
 'encoding' => 'utf-8'
);

// próba połączenia

try
{
  $dsn = $conn_config['db_type'] .
  ':host=' . $conn_config['host'] .
  ';port=' . $conn_config['port'] .
  ';encoding=' . $conn_config['encoding'] .
  ';dbname=' . $conn_config['db'];

 $dbh = new PDO($dsn,
$conn_config['user'],  $conn_config['pass']);

 // ustawienie trybu raportowania błędów

 $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

 define('DB_CONNECTED', true);

 echo '<h1>Connection success!</h1>';

 // zapytanie do bazy

 $stmt = $dbh->prepare('SELECT * FROM customers');

 $stmt->execute();
```

```
// pobranie/wyświetlenie wyników

 while($row = $stmt->fetch())
  {
   echo 'id: ' . $row['customer_id']  .
   ', imię:' . $row['first_name']  .
   ', nazwisko:' . $row['last_name'] .
   '<br />';
  }

 // zwolnienie kursora

 $stmt->closeCursor();
 unset($stmt);

  // [...] dalsze zapytania do bazy możliwe - kursor został
zwolniony...

 // zamknięcie połączenia z bazą

 $dbh = null;

// wyłapanie wyjątku w przypadku błędu połączenia z bazą

} catch(PDOException $e)
{
  die('Unable to connect: ' . $e->getMessage());
}
?>
```

Na koniec jeszcze dwie sprawy: jeśli chcemy sobie potestować rodzaje zwracanych przez fetch() danych, możemy sobie to zrobić za pomocą takiego wzoru:

```
$stmt = $dbh->prepare('SELECT * FROM customers');
$stmt->execute();

echo('PDO::FETCH_ASSOC: ');
echo('Wiersz indeksowany przez nazwę kolumny<br />');
$result = $stmt->fetch(PDO::FETCH_ASSOC);
print_r($result);
echo(' <br />');

echo('PDO::FETCH_BOTH: ');
echo('Wiersz indeksowany przez nazwę kolumny i indeks numeryczny
<br />');
$result = $stmt->fetch(PDO::FETCH_BOTH);
print_r($result);
echo(' <br />');
```

```
echo('PDO::FETCH_LAZY: ');
echo('Wiersz zwracany jako obiekt - nazwy kolumn jako właściwości
obiektu lub indeks numerowy <br />');
$result = $stmt->fetch(PDO::FETCH_LAZY);
print_r($result);
echo(' <br />');

echo('PDO::FETCH_OBJ: ');
echo('Wiersz zwracany jako obiekt - nazwy kolumn jako właściwości
obiektu <br/>');
$result = $stmt->fetch(PDO::FETCH_OBJ);
echo $result->LOGIN;
echo(' <br />');
```

Bindowanie zmiennych i zapytania sparametryzowane

Jak to działa?

Przyjrzyjmy się zapytaniu:

```
$stmt = $dbh->query('SELECT login, password FROM customers WHERE
customer_id =' . $id);
```

W zapytaniu tym próbujemy pobrać login i hasło klienta o numerze id pobieranym ze zmiennej $id. Co jest złego w tym zapytaniu? Generalnie nic. Ale popatrzmy teraz na poniższe:

```
$stmt = $dbh->prepare('SELECT login, password FROM customers WHERE
customer_id = :id');
```

W powyższym zapytaniu zniknęła nam zmienna $id, a zamiast niej pojawił się zapis **:**id.
Dodatkowo nie korzystamy już z query(), a z prepare().

Przeanalizujmy teraz poniższy kod:

```
$stmt = $dbh->prepare('SELECT login, password FROM customers WHERE
customer_id = :id');  //1
$stmt->bindValue(':id', $id, PDO::PARAM_INT);  //2
$result = $stmt->execute();  //3
```

Co się tutaj wyrabia? - bindValue(), prepare() i execute()

1) Za pomocą metody prepare() przygotowujemy jedynie SZKIELET zapytania, zamiast całego zapytania uzupełnionego od razu wartością pobraną ze zmiennej $id. Użyliśmy tutaj zapisu :id, który jest placeholderem dla zmiennej. Samą zmienną prześlemy do bazy dopiero w kolejnym kroku. Nazwy placeholderów mogą być dowolne (nie muszą być takie same jak nazwy zmiennej), byleby zostały pokryte w metodach bindValue().

2) Metodą:

```
$dbh->bindValue(':id', $id, PDO::PARAM_INT )
```

przypisujemy (bindujemy) wartość zmiennej $id do placeholdera :id. Zauważmy, że w trzecim parametrze podajemy typ naszej zmiennej, w tym przypadku jest to INTEGER informując tym samym bazę jakiego typu wartości się spodziewać. Podanie tutaj do zmiennej $id wartości typu STRING spowoduje NIEWYKONANIE zapytania, gdyż wymuszony jest typ INTEGER.

3) Dopiero w tym kroku wysyłamy całość do bazy metodą execute(). Jednocześnie do zmiennej $result zwracamy jest wynik wykonania zapytania (true|false). Oczywiście pobrane rekordy wylistować możemy tak jak poprzednio za pomocą $stmt->fetch(). Poza sposobem wykonania zapytania nic się tutaj innego w tym momencie nie zmieniło.

4) Zwalniamy kursor.

Lista możliwych parametrów podawanych w trzecim argumencie bindValue() znajduje się poniżej:

- PDO::PARAM_BOOL – boolean
- PDO::PARAM_NULL – null
- PDO::PARAM_INT – integer
- PDO::PARAM_STR – char, varchar, string
- PDO::PARAM_LOB – SQL large object datatype

Co ciekawe - brakuje tutaj typów zmiennoprzecinkowych.
Zmienne np. typu FLOAT dodajemy poprzez PDO::PARAM_STR i rzutowanie, np.:

```
bindValue(':cena', (float) $cena, PDO::PARAM_STR);
```

```
$stmt->closeCursor();  //4
```

Zwracanie liczby wierszy jakich dotyczyło zapytanie - rowCount()

Czasem konieczne jest pobranie liczby wierszy, których dotyczyło ostatnie zapytanie, np. podczas usuwania rekordów wg zadanego warunku chcemy wiedzieć ile wierszy zostało usuniętych. Metoda execute() zwróci nam jedynie wartość TRUE w przypadku wykonania zapytania, lub FALSE w przypadku niepowodzenia. Aby uzyskać liczbę zaafektowanych wierszy używamy metody $stmt->rowCount(), która pobiera informację o ilości wierszy jakich dotyczyło ostatnie wykonane zapytanie. Metoda ta zwraca poprawną ilość dla zapytań INSERT, UPDATE i DELETE. Z zapytaniem SELECT bywa tutaj różnie. Przykład użycia:

```
$id = 1;
$stmt = $dbh->prepare('DELETE FROM customers WHERE customer_id >
:id');
$stmt->bindValue(':id', $id, PDO::PARAM_INT);

$result = $stmt->execute();

$count = $stmt->rowCount();
echo 'Usuniętych wierszy: ' . $count;

$stmt->closeCursor();
```

Różnica pomiędzy bindValue(), a bindParam()

Do przypisania zmiennej do placeholdera służą dwie bardzo podobne do siebie metody, na pierwszy rzut oka identyczne:

```
bindValue(placeholder, zmienna, typ)
```
i
```
bindParam(placeholder, zmienna, typ)
```

Różnia pomiędzy tymi obiema metodami jednak jest i to dość znaczna.
Popatrzmy na przykłady:

bindValue():

```
$id = 2;
$stmt = $dbh->prepare('SELECT * FROM customers WHERE customer_id
= :id');
$stmt->bindValue(':id', $id, PDO::PARAM_INT);
$id = 6;
$result = $stmt->execute();
// wykona się zapytanie: 'SELECT * FROM customers WHERE
customer_id = 2'
```

bindParam():

```
$id = 2;

$stmt = $dbh->prepare('SELECT * FROM customers WHERE customer_id
= :id');

$stmt->bindParam(':id', $id, PDO::PARAM_INT);

$id = 6;

$result = $stmt->execute();

// wykona się zapytanie: 'SELECT * FROM customers WHERE
customer_id = 6'
```

W drugim przypadku w zmiennej $id została użyta wartość 6.
Stało się tak dlatego, iż bindParam() pobiera wartość zmiennej przez REFERENCJĘ - warto o tym pamiętać podczas korzystania z bindParam(), gdyż czasem fakt ten prowadzić może do trudnych do wykrycia błędów w logice aplikacji.

Szybkie bindowanie - znak zapytania (?)

Istnieje jeszcze jeden sposób przypisania zmiennych do placeholdera, który nie wymaga wywoływania metod bindValue(), ani bindParam(). Polega on na zastąpieniu wszystkich placeholderów znakami zapytania - ?, a następnie na przypisaniu tablicy zmiennych bezpośrednio w metodzie execute(). Zobaczmy na przykładzie:

```
$country = 'Polska';

$city = 'Warszawa';

$stmt = $dbh->prepare('SELECT * FROM customers WHERE country=?
AND city=?');

$result = $stmt->execute(array($country, $city));

// wykona się zapytanie: 'SELECT * FROM customers WHERE
country="Polska" AND city="Warszawa"'
```

W przypadku takim jak powyżej wszystkie placeholdery podajemy jako znak zapytania, nastepnie pokrywamy ich wartości w tablicy przekazywanej jako argument do metody execute(). Warto pamiętać, że ważna jest kolejność - zmienne podane w tablicy muszą być podawane w takiej samej kolejności w jakiej występują w zapytaniu. Trzeba też pamiętać, iż zmienne do execute() zawsze podajemy w tablicy, nawet jeśli jest to tylko jedna zmienna.

Przykładowy kod

Wróćmy jednak do metody z użyciem bindValue() i przyjrzyjmy się jak wyglądać może całość naszego kodu. Dla przykładu wyobraźmy sobie, że potrzebujemy pobrać z bazy wszystkich klientów, którzy mieszkają w Polsce i którzy zarejestrowali się w bazie wcześniej niż godzinę temu. Czas rejestracji trzymany jest w bazie w polu created_at i zapisany jako linuxowa liczba sekund, musimy więc od obecnego czasu odjąć ilość sekund jaka upłynęła przez godzinę. i wyświetlić klientów, których czas rejestracji jest mniejszy od tej wartości. Nasz kod wyglądać będzie tak:

```
<?php

// konfiguracja

$conn_config = array(

  'host' => 'localhost',
  'port' => '3306',
  'user' => 'root',
  'pass' => 'password',
  'db' => 'myDatabase',
  'db_type' => 'mysql',
  'encoding' => 'utf-8'
);

// próba połączenia

try
{
 $dsn = $conn_config['db_type'] .

 ':host=' . $conn_config['host'] .

 ';port=' . $conn_config['port'] .

 ';encoding=' . $conn_config['encoding'] .

 ';dbname=' . $conn_config['db'];

  $dbh = new PDO($dsn,
$conn_config['user'],  $conn_config['pass']);

  // ustawienie trybu raportowania błędów

  $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  define('DB_CONNECTED', true);

  echo '<h1>Connection success!</h1>';
```

```
  $now = time();

  $minus_hour = $now - 60*60;

  $country = 'Polska';

  $stmt = $dbh->prepare('SELECT * FROM customers WHERE country =
:country AND created_at < :hour');

 // przygotowanie szkieletu zapytania

  $stmt->bindValue(':country', $country, PDO::PARAM_STR); //
przypisanie zmiennej $country do :country

  $stmt->bindValue(':hour', $minus_hour, PDO::PARAM_INT); //
przypisanie zmiennej $minus_hour do :hour, jak widzimy nazwa
placeholdera nie musi się równać nazwie zmiennej

  $result = $stmt->execute(); // wykonanie zapytania

  if($result !== false)
  {
    // wyświetlenie wyniku;

    while($row = $stmt->fetch())
    {
      echo 'id: ' . $row['customer_id']  .

     ', imię:' . $row['first_name']  .

     ', nazwisko:' . $row['last_name'] .

     '<br />';
    }
  }

  // zwalniamy kursor

  $stmt->closeCursor();
  unset($stmt);

  // zamknięcie połączenia z bazą

  $dbh = null;
```

```
  // wyłapanie wyjątku w przypadku błędu połączenia z bazą

} catch(PDOException $e)
{
  die('Unable to connect: ' . $e->getMessage());
}

?>
```

Jakie z tego płyną korzyści?

1. Szkielet zapytania jest wysyłany jedynie raz, zatem oszczędzamy na ilości zapytań, jeśli np. w tym miejscu mamy zapytanie dodające nowe rekordy – to zyskujemy znacznie na prędkości. Do wysłanego RAZ szkieletu zapytania dosyłane są jedynie wartości zmiennych odpowiadające danym placeholderom.
2. Elastyczność – ten sam szkielet zapytania możemy wykorzystać wielokrotnie, podmieniając jedynie wartości zmiennych.
3. Informujemy jakiego typu zmienną przesyłamy, więc zwalnia nas to jednocześnie z ujmowania odpowiednich wartości w cudzysłowy.
4. Daje nam to z automatu prosty system walidacji, gdyż mechanizm nie przyjmie w parametrze wartości innej, niż określona - nie wyślemy np. stringa do pola zdefiniowanego jako integer.
5. I chyba najważniesze - mechanizm ten całkowicie zabezpiecza przed atakami typu SQL Injection, polegającymi na wstrzykiwaniu do zapytania spreparowanych wartości. Tutaj to nie zadziała, gdyż zmienne nie są wysyłane razem z zapytaniem - są przetwarzane oddzielnie.

Dodawanie nowego rekordu

a) Wykonanie zapytania- exec()

Dodawanie nowych rekordów za pomocą PDO jest bardzo proste i ogranicza się do wykonania zapytania INSERT INTO. Można to zrobić na dwa sposoby - za pomocą sparametryzowanych zmiennych jak i bez. Oczywiście zawsze wykorzystywać będziemy metodę pierwszą, jednak przyjrzyjmy się też metodzie drugiej ze względów czysto formalnych. Przeanalizujemy zatem teraz najprostrzy przykład, niewykorzystujący bindowania zmiennych, czyli metodę exec(). Metoda ta szybkie wykonanie zapytania typu INSERT, UPDATE lub DELETE. W tym przypadku wykonamy zapytanie INSERT INTO.

Użycie metody exec() wyglada następująco:

```
$result = $dbh->exec($zapytanie);
```

Przygotować musimy prosty formularz, w którym podamy wszystkie wymagane dane, aby dodać użytkownika do bazy. Formularz wyglądać będzie tak:

```
<form method="POST" action="add_user.php">
```

```
<input type="hidden" name="action" value="add" />

<input type="text" name="login" value="" />

<input type="password" name="password" value= ""/>

<input type="text" name="first_name" value="" />

<input type="text" name="last_name" value="" />

<input type="text" name="email" value="" />

<input type="text" name="country" value="" />

<input type="text" name="city" value="" />

<input type="text" name="post_code" value="" />

<input type="text" name="address" value="" />

<input type="text" name="phone" value="" />

<input type="submit" name="add" value="add" />

</form>
```

Formularz nie będzie zawierał żadnej walidacji, robimy to tylko dla przykladu. Załóżmy jednak, że wszytkie pola w formularzu są wymagane i ich wypełnienie jest konieczne dla dodania nowego klienta do bazy.

```
$created_at = time();

// przygotowanie szkieletu zapytania INSERT INTO

$stmt = $dbh->prepare('INSERT INTO customers(

login,
password,
first_name,
last_name,
email,
country,
city,
post_code,
address,
phone,
created_at,
updated_at
)
```

```
VALUES(
:login,
:password,
:first_name,
:last_name,
:email,
:country,
:city,
:post_code,
:address,
:phone,
:created_at,
:updated_at
)');

// przypisujemy zmienne do placeholderów

bindValue(':login', $login, PDO::PARAM_STR);
bindValue(':password', $password, PDO::PARAM_STR);
bindValue(':first_name', $first_name, PDO::PARAM_STR);
bindValue(':last_name', $last_name, PDO::PARAM_STR);
bindValue(':email', $email, PDO::PARAM_STR);
bindValue(':country', $country , PDO::PARAM_STR);
bindValue(':city', $city, PDO::PARAM_STR);
bindValue(':post_code', $post_code, PDO::PARAM_STR);
bindValue(':address', $address, PDO::PARAM_STR);
bindValue(':phone', $phone, PDO::PARAM_STR);
bindValue(':created_at', $created_at, PDO::PARAM_INT);
bindValue(':updated_at', $created_at, PDO::PARAM_INT);

// wykonujemy zapytanie
$result = $stmt->execute();
if($result !== false)
{
    echo 'Dodano użytkownika o ID = ' . $dbh->lastInsertId();
} else {
    echo 'Wystąpił błąd';
}
$stmt->closeCursor();
unset($stmt);
$dbh = null;
}
```

Pobranie id dodawanego rekordu - lastInsertId()

Przy okazji zapytań INSERT warto wspomnieć o pobieraniu wartości id ostatniego dodanego rekordu. Służy do tego metoda:

```
$inserted_id = $dbh->lastInsertId()
```

Aktualizacja rekordu

Aktualizowanie rekordów w bazie za pomocą PDO nie różni się niczym od poznanych do tej pory metod. Podobnie jak przy zapytaniu INSERT INTO, do zapytania UPDATE również wykorzystujemy metodę prepare() do przygotowania szkieletu zapytania, metodę bindValue() do przypisania zmiennych, a na koniec metodę execute() do wykonania zapytania. Przykład poniżej:

```
if($_POST['action'] == 'update')

{
   $id = $_POST['id'];

  $password = $_POST['password'];

  $first_name = $_POST['first_name'];

  $last_name = $_POST['last_name'];

  $email = $_POST['email'];

  $country = $_POST['country '];

  $city = $_POST['city'];

  $post_code = $_POST['post_code'];

  $address = $_POST['address'];

  $phone = $_POST['phone'];

  $updated_at = time();
```

```
 // przygotowanie szkieletu zapytania
$stmt = $dbh->prepare('UPDATE customers SET
                      password = :password,
                      first_name = :first_name,
                      last_name =  :last_name,
                      email =  :email,
                      country = :country,
                      city = :city,
                      post_code = :post_code,
                      address = :address,
                      phone = :phone,
                      updated_at = :updated_at
                     WHERE customer_id= :id
  ');
// przypisujemy zmienne do placeholderów
bindValue(':login', $login, PDO::PARAM_STR);
bindValue(':password', $password, PDO::PARAM_STR);
bindValue(':first_name', $first_name, PDO::PARAM_STR);
bindValue(':last_name', $last_name, PDO::PARAM_STR);
bindValue(':email', $email, PDO::PARAM_STR);
bindValue(':country', $country , PDO::PARAM_STR);
bindValue(':city', $city, PDO::PARAM_STR);
bindValue(':post_code', $post_code, PDO::PARAM_STR);
bindValue(':address', $address, PDO::PARAM_STR);
bindValue(':phone', $phone, PDO::PARAM_STR);
bindValue(':updated_at', $updated_at, PDO::PARAM_INT);
bindValue(':id', $id, PDO::PARAM_INT);
```

```
   // wykonujemy zapytanie

    $result = $stmt->execute();

    if($result !== false)
     {
       echo 'Zaktualizowano użytkownika o ID = ' . $id;
     } else {
       echo 'Wystąpił błąd';
     }

     $stmt->closeCursor();
     unset($stmt);
     $dbh = null;
   }
```

Usuwanie rekordów

I analogicznie ma się sprawa z zapytaniem DELETE, które wykonujemy w identyczny sposób jak zapytania INSERT INTO i UPDATE. Przykład poniżej usuwa z tabeli klienta o id pobieranym ze zmiennej $del_id:

```
   // przygotowanie szkieletu zapytania INSERT INTO

  $stmt = $dbh->prepare('DELETE FROM customers WHERE customer_id =
:id');

  // przypisujemy zmienną do placeholdera

  bindValue(':id', $del_id, PDO::PARAM_INT);

  // wykonujemy zapytanie

  $result = $stmt->execute();

  if($result !== false)
  {
    echo 'Usunięto użytkownika o ID = ' . $del_id;
  } else {
     echo 'Wystąpił błąd';
  }

  $stmt->closeCursor();
   unset($stmt);
   $dbh = null;
```