Podejście obiektowe.

1. Korzystanie z PDO

PDO z ang. ***PHP Data Objects***, to rozszerzenie pozwalające na ujednoliconą obsługę różnych baz danych w technice obiektowej. Jest częścią ***PHP*** począwszy od wersji 5.1. Jego zaletą jest ujednolicona obsługa połączeń z bazami, niezależnie od tego, czy będzie to ***SQLite***, ***MySQL,*** ***PostgreSQL***, czy dowolna inna.

1.1 Nawiązanie połączenia

Nawiązanie połączenia z bazą wygląda tu zupełnie inaczej niż w klasycznym podejściu. Należy w tym celu utworzyć nowy obiekt klasy ***PDO***, korzystając z konstruktora w postaci:

PDO(źródło_danych[, użytkownik[, hasło[, opcje]]]);

Gdzie ***źródło_danych*** to ciąg znaków określający rodzaj bazy danych i sposób połączenia, użytkownik to nazwa użytkownika, hasło to hasło, a opcje to tablica zawierająca dodatkowe opcje związane z połączeniem. Jedynym wymaganym argumentem jest ***źródło_danych***, pozostałe są opcjonalne.

Uproszczona postać ciągu ***źródło_danych*** dla bazy ***MySQL***, z której będziemy korzystać, ma postać:

mysql:host=nazwa_serwera;port=numer_portu;dbname=nazwa_bazy

Poszczególne składowe mają następujące znaczenie:

- ***nazwa_serwera*** – określenie nazwy lub adresu serwera baz danych (np. localhost)
- ***numer_portu*** – port na którym ma nastąpić połączenie z bazą. Może być pominięty, zostanie wtedy zastosowana wartość standardowa.
- ***nazwa_bazy*** – nazwa bazy danych, z którą ma nastąpić połączenie.

Tak więc ciąg ***źródło_danych***, który pozwala na połączenie z bazą ***testphp*** znajdującą się na serwerze ***MySQL*** pracującym na komputerze lokalnym na porcie ***3306***, będzie miał postać:

mysql:host=localhost;port=3306;dbname=testphp

lub, jeśli określenie portu zostanie pominięte

mysql:host=localhost;dbname=testphp

Jeżeli wywołanie konstruktora zakończy się sukcesem, zostanie zwrócony obiekt, który będzie służył do dalszej komunikacji, należy więc przypisać go jakiejś zmiennej. W przypadku gdy połączenia nie uda się nawiązać, zostanie wygenerowany wyjątek typu ***PDOException***, który będzie zawierał opis przyczyny powstania błędu. A zatem fragment kodu ustalający, czy udało się nawiązać połączenie z bazą mógłby mieć postać:

```php
<?php
$dsn = "mysql:host=localhost;dbname=testphp";
$uzytkownik = "php";
$haslo = "test";
try{
  $dbo = new PDO($dsn, $uzytkownik, $haslo);
}
catch (PDOException $e){
  echo 'Błąd połączenia: ' . $e->getMessage();
  exit;
}
//dalsze instrukcje skryptu
?>
```

1.2 Kończenie połączenia

Połączenie nawiązane za pomocą obiektu ***PDO*** pozostaje aktywne przez cały czas życia obiektu i jest kończone przy usuwaniu obiektu z pamięci. Zostanie to wykonane automatycznie po zakończeniu pracy skryptu lub po przypisaniu wartości ***null*** zmiennej obiektowej przechowującej odwołanie do obiektu.

1.3 Zapytania pobierające dane

Zapytania są wykonywane za pomocą metody ***query***, której należy przekazać treść zapytania w postaci argumentu. Zakładając zatem, że mamy do dyspozycji obiekt ***$dbo*** powstały przez wywołanie konstruktora klasy ***PDO***, zapytanie ***SQL*** można wykonać przez zastosowanie konstrukcji:

$dbo->query(„treść zapytania”);

Oczywiści treść zapytania może być również przekazywana w postaci zmiennej, np.:

$query= „treść zapytania”;

$dbo->query($query);

Metoda ***query*** zwraca obiekt typu ***PDOStatement*** pozwalający na odczyt danych po wykonaniu zapytania, gdy zakończyło się sukcesem, lub też wartość ***false*** w przeciwnym razie.

Wspomniany obiekt zawiera metodę ***fetch***, która udostępnia pobrane dane. Jej ogólne wywołanie ma postać:

fetch([typ_wyniku])

Zwracaną wartością jest kolejny wiersz z wyników zapytania lub wartość ***false***, jeżeli kolejnego wiersza nie uda się pobrać (np. zostały już odczytane wszystkie dane). Postać zwróconych danych zależy od stanu argumentu ***typ_wyniku***, który może przyjmować następujące wartości:

- ***PDO::FETCH_ASSOC*** – zwraca tablicę asocjacyjną, w której nazwy kolumn wynikowych są kluczami,
- ***PDO::FETCH_BOTH*** – zwraca tablicę indeksowaną zarówno numerycznie, jak i asocjacyjnie (jest to wartość domyślna),
- ***PDO::FETCH_BOUND*** – zwraca wartość ***true*** oraz przypisuje wartość z kolumn wyniku do zmiennych ***PHP*** ustalonych wcześniej za pomocą wywołania metody ***bindColumn(),***
- ***PDO::FETCH_CLASS*** – zwraca nową instancja klasy, dokonując mapowania kolumn wynikowych na właściwości klasy,
- ***PDO::FETCH_INTO*** – uaktualnia istniejącą instancje klasy, dokonując mapowania kolumn wynikowych na właściwości klasy,
- ***PDO::FETCH_LAZY*** – kombinacja ***PDO::FETCH_BOTH i PDO::FETCH_OBJ***,
- ***PDO::FETCH_NUM*** – zwraca tablicę indeksowaną numerycznie,
- ***PDO::FETCH_OBJ*** – zwraca obiekt z właściwościami o nazwach i wartościach odpowiadającym kolumnom wynikowym zapytania.

Aby zmienić domyślny tryb obowiązujący dla wszystkich zapytań (czyli standardowe ***PDO::FETCH_BOTH***), należy wykorzystać metodę ***setFetchMode***, której wywołanie ma postać:

setFetchMode([domyślny_typ_wyniku])

gdzie ***domyślny_typ_wyniku*** to jedna z wartości wymienionych wyżej. Jeśli chcemy na przykład, aby domyślnym typem wyniku była tablica indeksowana numerycznie, zastosujemy wywołanie:

$result->setFetchMode(PDO::FETCH_NUM);

Po którym wszystkie wywołania metody ***fetch*** będą zwracały właśnie tablice numeryczne.

Ponieważ każde wywołanie ***fetch*** powoduje zwrócenie kolejnego wiersza wyniku lub wartości ***false***, jeśli zostały odczytane wszystkie wiersze metodę tę można wywoływać w pętli ***while*** o schematycznej postaci:

while($row=$result->fetch()){

//instrukcje przetwarzające wiersz tabeli

}

Czas zatem zebrać przedstawione dotychczas w tej lekcji informacje i zapisać skrypt odczytujący za pomocą ***PDO*** dane z wybranej tabeli bazy danych.

```php
<?php
$dsn = "mysql:host=localhost;dbname=testphp";
$uzytkownik = "php";
$haslo = "test";
try{
   $dbo = new PDO($dsn, $uzytkownik, $haslo);
}
catch (PDOException $e){
   echo 'Błąd połączenia: ' . $e->getMessage();
   echo '</div></body></html>';
   exit;
}

$query = "SELECT * FROM osoba";
$result = $dbo->query($query);

if(!$result){
   echo "Nie mogę wykonać zapytania.<br />";
   echo '</div></body></html>';
   exit;
}
?>
<table>
<tr>
   <td>Id</td><td>Imie</td><td>Nazwisko</td>
   <td>Rok urodzenia</td><td>Miejsce urodzenia</t
</tr>
<?php
while($row = $result->fetch(PDO::FETCH_NUM)){
   echo "<tr>\n";
   echo "<td>{$row[0]}</td>\n";
   echo "<td>{$row[1]}</td>\n";
   echo "<td>{$row[2]}</td>\n";
   echo "<td>{$row[3]}</td>\n";
   echo "<td>{$row[4]}</td>\n";
   echo "</tr>\n";
}
$dbo = null;
?>
```

Warto w tym miejscu pokazać, jak wyglądałby taki skrypt, gdyby wyniki zapytania były by pobierane jako obiekty – taka technika jest często spotykana. W takiej sytuacji wartością zwracaną przez metodę ***fetch*** jest obiekt, którego właściwości odpowiadają kolumnom

wynikowym zapytania ***SQL***. To znaczy nazwami właściwości są nazwy kolumn, a ich wartościami – wartości zapisane w tych kolumnach. A zatem pętla ***while*** generująca kolejne komórki tabeli ***HTML*** o zawartości pobieranej z wyników zapytania miałyby w tej technice następującą postać:

```
<?php
while($row = $result->fetch(PDO::FETCH_OBJ)){
   echo "<tr>\n";
   echo "<td>{$row->Id}</td>\n";
   echo "<td>{$row->Imie}</td>\n";
   echo "<td>{$row->Nazwisko}</td>\n";
   echo "<td>{$row->Rok_urodzenia}</td>\n";
   echo "<td>{$row->Miejsce_urodzenia}</td>\n";
   echo "</tr>\n";
}
$dbo = null;
?>
```

1.4 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.

Przykładowo, metody ***fetchAll()*** użyć możemy tak:

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

foreach($result as $row)

 {

  echo 'id: ' . $row['ID']   .

  ', imię:' . $row['Imie']   .

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

  '<br />';

 }
```

1.5 **Zwolnienie kursora - closeCursor()**

Różnica Teraz bardzo ważna sprawa. Jak już napisałem 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 możemy 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:

$result->closeCursor();

którą wywołujemy ZAWSZE po skończonej pracy na danym zbiorze wyników.

1.6 Bindowanie zmiennych i zapytania sparametryzowane

Parametryzowanie zapytań to niezwykle ciekawe rozwiązanie. Jest to o wiele lepsze i bezpieczniejsze rozwiązanie, niż wysyłanie zapytań do bazy wraz z zagnieżdżonymi w nim danymi pobranymi ze zmiennych. W tej części zobaczymy ile korzyści może płynąć z takiego właśnie sposobu. Parametryzowanie zapytań przede wszystkim pozwala na oddzielenie wstrzykiwanych w zapytania danych od treści samego zapytania i wprowadza kilka aspektów bezpieczeństwa.

Jak to działa?

Przyjrzyjmy się zapytaniu:

```
$stmt = $dbo->query('SELECT Imie, Nazwisko FROM osoba WHERE Id =' . $id);
```

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

```
$stmt = $dbo->prepare('SELECT Imie, Nazwisko FROM osoba WHERE Id = :id');  //1
```

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 = $dbo->prepare('SELECT Imie, Nazwisko FROM osoba WHERE Id = :id');  //1

$stmt->bindValue(':id', $id, PDO::PARAM_INT);  //2

$result = $stmt->execute();  //3

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

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:

```
$stmt->bindValue(':id', $id, PDO::PARAM_INT);  //2
```

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

1.7 **Szybkie bindowanie - znak zapytania (?)**

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

```
$rok = 1977;

$miejsce = 'Sopot';

$stmt = $dbh->prepare('SELECT * FROM osoba WHERE Rok_urodzenia=? AND Miejsce_urodzenia=?');

$result = $stmt->execute(array($rok, $miejsce));

// wykona się zapytanie: 'SELECT * FROM osoba WHERE Rok_urodzenia=1977 AND
Miejsce_urodzenia="Sopot"'
```

W przypadku takim jak powyżej wszystkie ***placeholdery*** podajemy jako znak zapytania, następnie 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.

1.8 Zapytania modyfikujące dane

Do wykonywania zapytań modyfikujących używa sią metody ***exec***. Wysyła ona zapytanie do serwera oraz zwraca wartość całkowitą określającą liczbę rekordów, na które to zapytanie miało wpływ (czyli np. liczbę zmodyfikowanych, usuniętych lub dodanych wierszy tabeli). Jeżeli wykonanie zapytania nie powiodło się , metoda zwraca wartość ***false***. Zapytanie należy przekazać w postaci argumentu, zatem schematyczne wywołanie wygląda następująco:

$dbo->exec(„Treść zapytania”);

Zobaczmy więc, jak w praktyce dodać wiersz do tabeli osoba.

```php
$imie = $_POST['imie'];
$nazwisko = $_POST['nazwisko'];
$rok_ur = $_POST['rok_ur'];
$miejsce_ur = $_POST['miejsce_ur'];
 // wykonanie polecenia INSERT
$result = $dbo->exec('INSERT INTO osoba(
                      Imie,
                      Nazwisko,
                      Rok_urodzenia,
                      Miejsce_urodzenia
                      )
                      VALUES(
                      "'.$imie.'",
                      "'.$nazwisko.'",
                      ".$rok_ur.",
                      "'.$miejsce_ur.'"
                      )');
if($result !== false)
{
  echo 'Dodano użytkownika o ID = ' . $dbo->lastInsertId();

} else {
  echo 'Wystąpił błąd';
}
```

Wykonanie zapytania za pomocą ***prepare()*** i ***execute():***

```
$imie = $_POST['imie'];
$nazwisko = $_POST['nazwisko'];
$rok_ur = $_POST['rok_ur'];
$miejsce_ur = $_POST['miejsce_ur'];
 // wykonanie polecenia INSERT
$stmt = $dbo->prepare('INSERT INTO osoba(
                        Imie,
                        Nazwisko,
                        Rok_urodzenia,
                        Miejsce_urodzenia
                        )
                        VALUES(
                        :imie,
                        :nazwisko,
                        :rok_ur,
                        :miejsce_ur
                        )');
                        // przypisujemy zmienne do placeholderów
bindValue(':imie', $imie, PDO::PARAM_STR);
bindValue(':nazwisko', $nazwisko, PDO::PARAM_STR);
bindValue(':rok_ur', $rok_ur, PDO::PARAM_INT);
bindValue(':miejsce_ur', $miejsce_ur, PDO::PARAM_STR);
 // wykonujemy zapytanie
$result = $stmt->execute();
if($result !== false)
{
  echo 'Dodano użytkownika o ID = ' . $dbo->lastInsertId();

} else {
  echo 'Wystąpił błąd';
}
```