Dostęp do baz danych przy wykorzystaniu interfejsu PDO

1 Wprowadzenie

W opisie zadań wykorzystano środowisko programistyczne *NetBeans* (wersja 7.2.1). Celem ćwiczenia jest zapoznanie się z mechanizmem dostępu do bazy danych z poziomu języka PHP z wykorzystaniem interfejsu PDO (PHP Data Object). Interfejs PDO jest obiektowo-zorientowanym interfejsem programistycznym dostępu do SQL-owych baz danych z wykorzystaniem języka PHP. Główną ideą oraz zaletą interfejsu jest niezależność od źródła danych oraz zapewnienie współpracy aplikacji z różnymi bazami danych np.: FreeTDS / Microsoft SQL Server / Sybase, Firebird/Interbase 6, IBM DB2, IBM Informix Dynamic Server, MySQL, Oracle Call Interface, ODBC v3 (IBM DB2, unixODBC and win32 ODBC), PostgreSQL, SQLite itd. Poprawna aplikacja składa się z następujących podstawowych kroków:

- Połączenie ze źródłem danych
- Określenie parametrów połączenia
- Przygotowanie i wykonanie polecenia
- Pobranie wyników
- Odłączenie od źródła danych (opcjonalnie)

Interfejs PDO składa się z trzech klas: PDO, PDOStatement oraz PDOException. Klasa PDO jest odpowiedzialna za utrzymanie połączenia z bazą danych. Klasa PDOStatement jest odpowiedzialna za operacje wykonywane w języku SQL oraz przetwarzanie ich wyników. Klasę PDOException wykorzystujemy do obsługi błędów.
Niniejsze ćwiczenie przybliży pracę z interfejsem PDO z wykorzystaniem powyższych klas. Przedstawione zostaną takie elementy jak nawiązywanie połączenia, wykonywanie zapytań prostych i parametryzowanych, obsługa błędów, obsługa transakcji itp.

2 Uruchomienie serwera aplikacji

Aby wykorzystywać skrypty PHP, potrzebny jest serwer aplikacji, który będzie je wykonywał. W naszym ćwiczeniu wykorzystamy serwer Apache, który wchodzi w skład zintegrowanego narzędzia XAMPP.

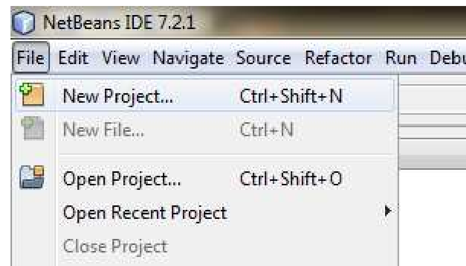
2.1 Uruchom program XAMPP Control Panel.

2.2 Sprawdź czy usługa *Apache* została uruchomiona, jeśli nie naciśnij przycisk *start.*

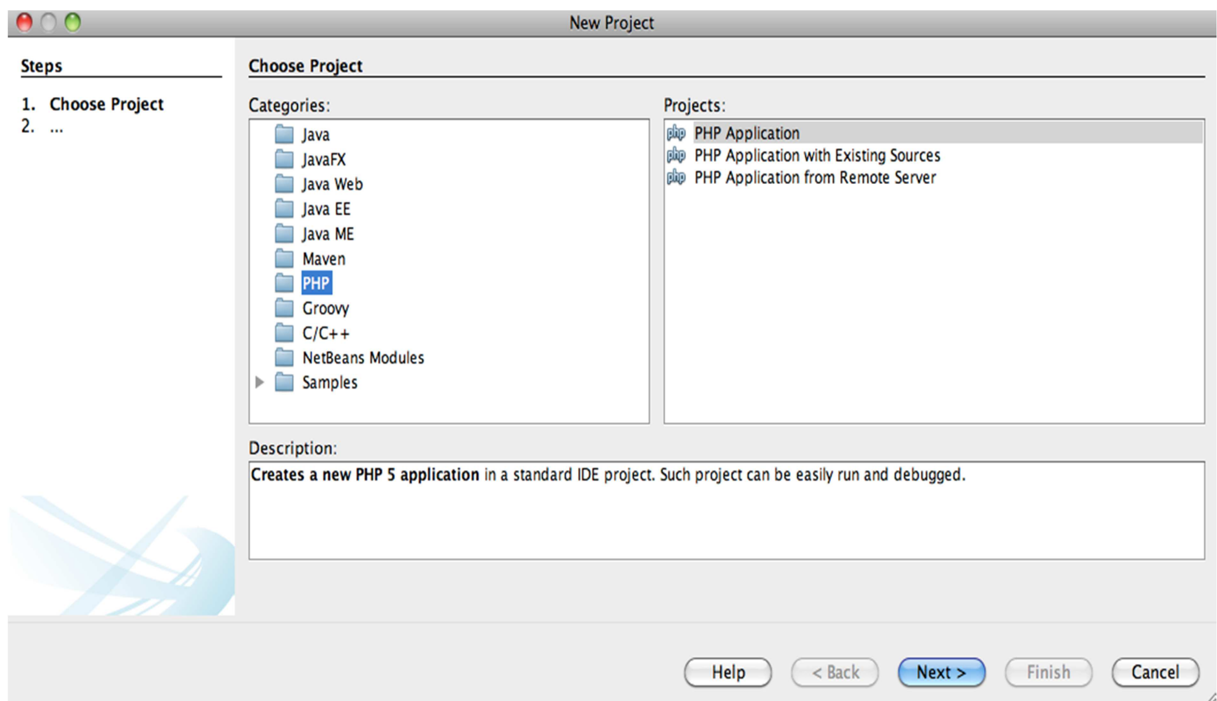
3 Utworzenie projektu w środowisku NetBeans

3.1 Uruchom środowisko *NetBeans*.

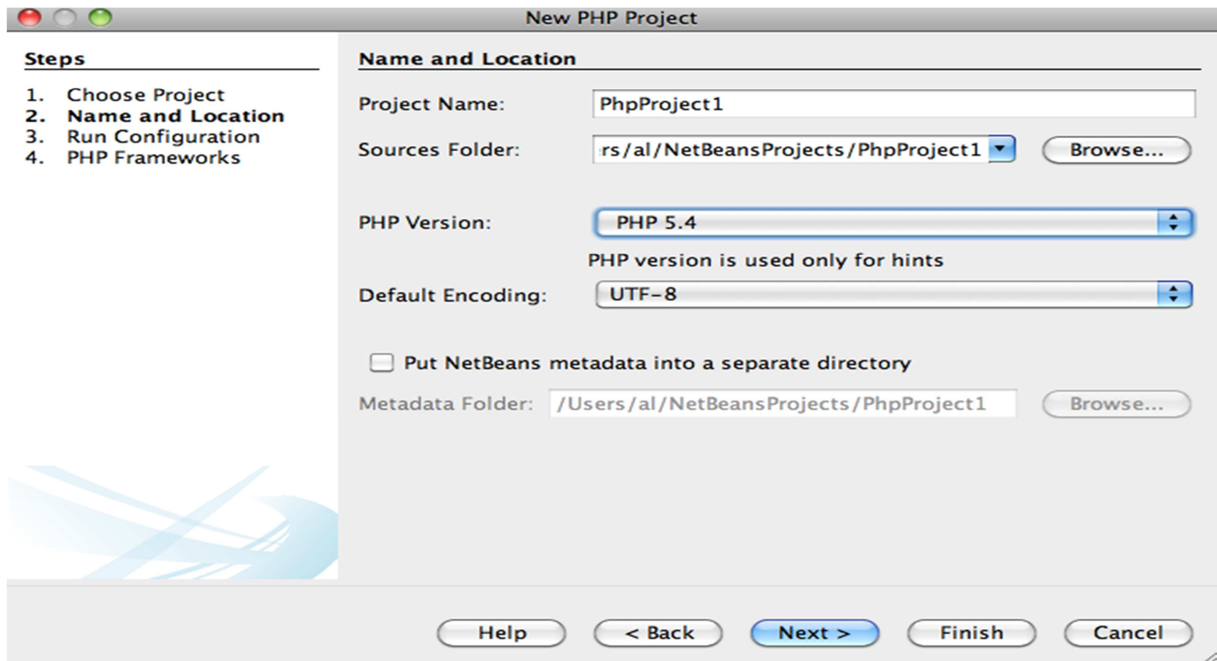
3.2 Utwórz nowy projekt wybierając z górnego menu ***File->New Project***.



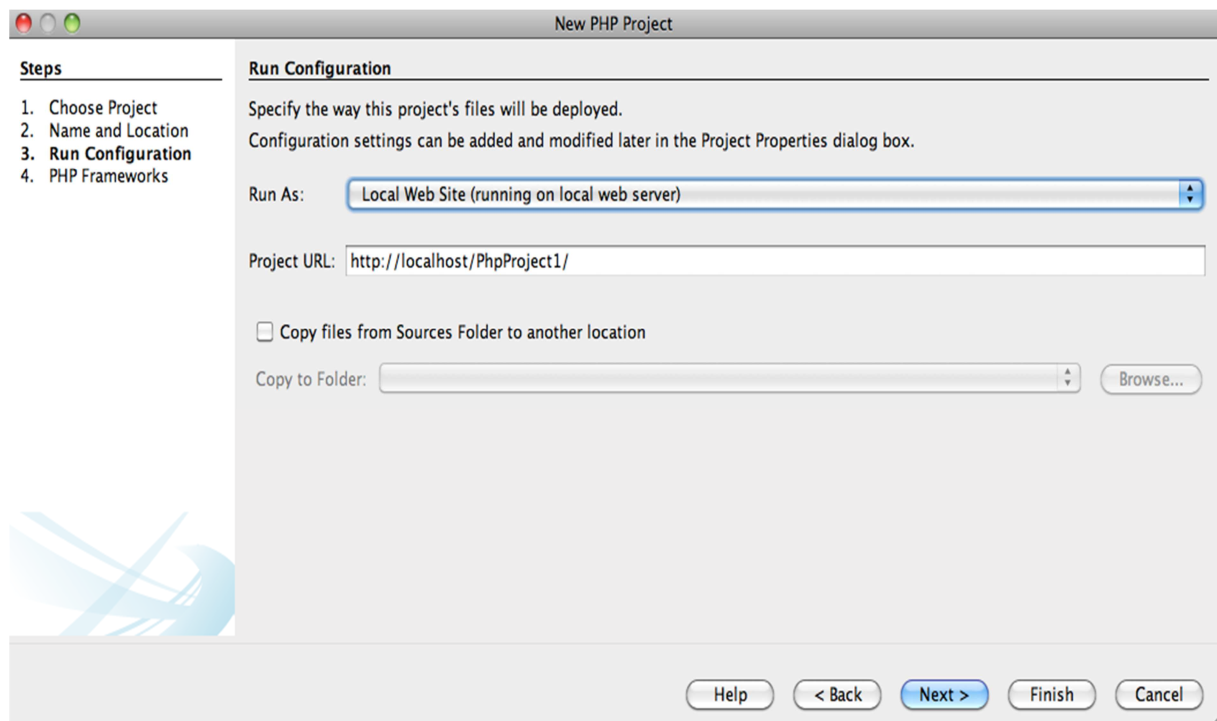
3.3 Wybierz kategorię *PHP* i wskaż jako typ projektu *PHP Application*. Naciśnij przycisk **Next >**.



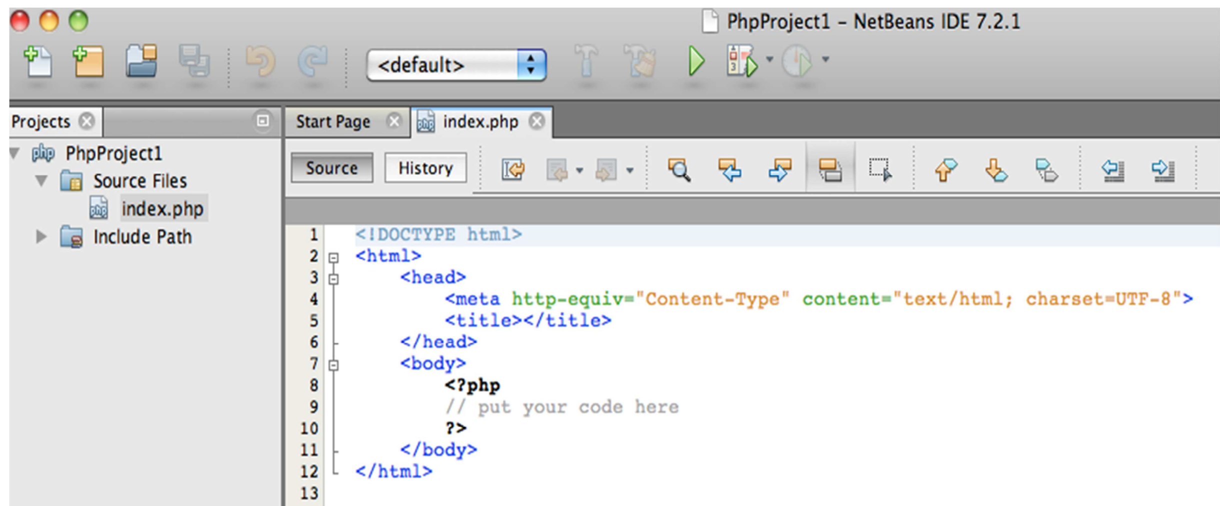
3.4 Wprowadź nazwę projektu np. *PhpProject1*. Pozostałe opcje pozostaw bez zmian. Naciśnij przycisk **Next**.



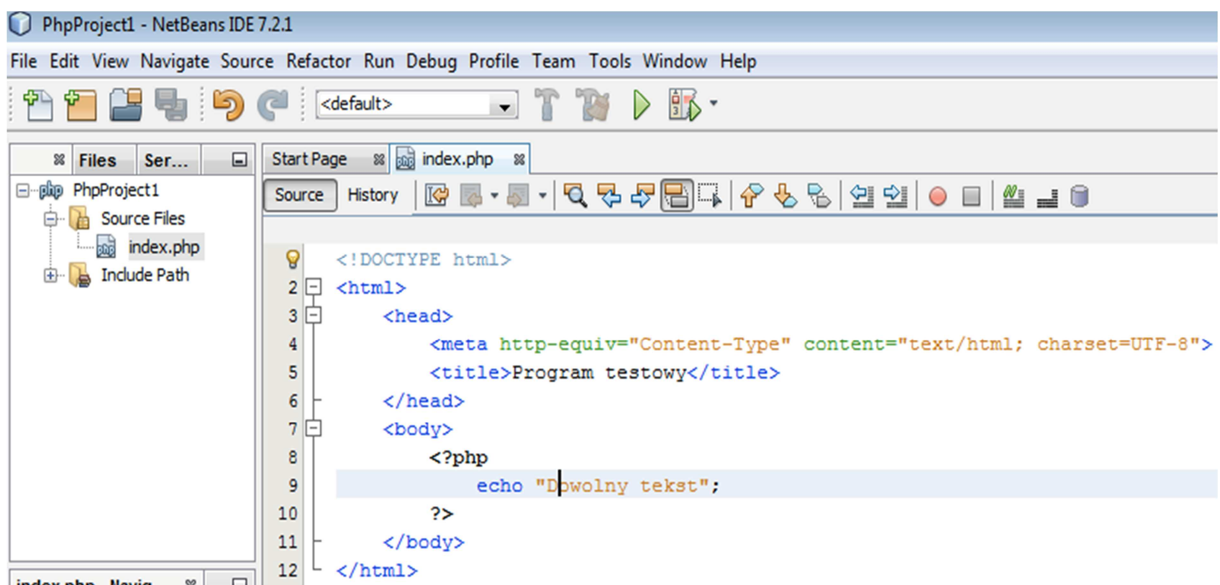
3.5 Wybierz opcję Run As: Local Web Site a następnie naciśnij przycisk **Finish** aby zakończyć działanie kreatora.



3.6 Efektem działania kreatora jest utworzenie projektu o wskazanej nazwie. W projekcie automatycznie zostanie dodany skrypt .php, który stanowi punkt startowy wykonania programu.



3.7 Zaczniemy od uruchomienia testowego skryptu. W sekcji **php** dodajemy kod, który będzie wykonany przez serwer aplikacji np.: (*echo " dowolny tekst ";)*.



Jednym z najczęściej pojawiających się błędów jest błąd wynikający z braku zainstalowanego sterownika (plik php_pdo_oci.dll). Aby sprawdzić czy dysponujemy odpowiednim sterownikiem do naszej bazy danych uruchamiamy skrypt http://localhost/xampp/phpinfo.php. Wynikiem skryptu jest zestawienie informacji na temat aktualnego stanu systemu z uwzględnieniem włączonych sterowników. W naszym przypadku jest to sterownik oci (oracle call interface), o którym informacje znajdują się w sekcji PDO -> PDO drivers -> oci (enabled).

PDO

PDO support	enabled
PDO drivers	mysql, oci, sqlite

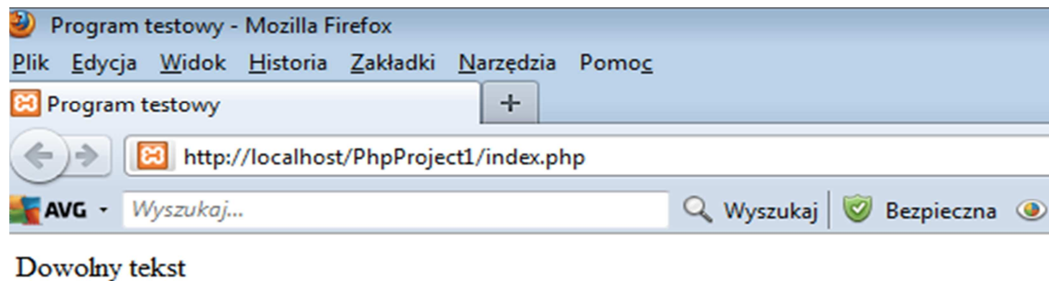
Innym sposobem weryfikacji sterownika jest wykorzystanie funkcji PDO::getAvailableDrivers().

```
<?php

foreach(PDO::getAvailableDrivers() as $driver) {
  echo "$driver.<br/>";}

?>
```

3.8 Po sprawdzeniu zainstalowanego sterownika możemy następnie, uruchomić program testowy, wybierając z górnego menu opcję ***Run->Run Project*** lub używając skrótu klawiszowego ***F6***. Przed uruchomieniem nastąpi weryfikacja poprawności składniowej po czym gotowy skrypt .php zostaje przesłany do przeglądarki, gdzie będzie widoczny efekt działania kodu wygenerowany przez serwer aplikacji.



4 Nawiązanie połączenia z bazą danych.

4.1 Połączenie z bazą danych

W pierwszym kroku wykorzystujemy konstruktor klasy PDO do ustanowienia połączenia ze źródłem danych (***PDO->_construct()***). Parametry przekazywane do konstruktora są następujące: źródło danych DSN (Data Source Name), gdzie podajemy adres hosta, port oraz identyfikator bazy danych, nazwa użytkownika (opcjonalnie), hasło użytkownika (opcjonalnie) oraz tablica z parametrami (opcjonalnie). Na wyjściu otrzymujemy obiekt klasy PDO reprezentujący połączenie z bazą danych. Aby móc przechwycić ewentualny komunikat o możliwym wyjątku w połączeniu możemy użyć konstrukcji try-catch. Poniższy przykład pokazuje proste nawiązanie połączenia z użyciem minimalnego zestawu parametrów.

```
<?php

try
{
 $conn = new PDO("oci:dbname=".
          "admlab2-main.cs.put.poznan.pl:1521/dblab01.cs.put.poznan.pl",
          "scott","tiger");
      echo "Połączono z bazą danych!";
}
catch(PDOException $e)
{
    echo ($e->getMessage());
}
?>
```

W sekcji catch korzystamy z klasy PDOException aby przechwycić ewentualny komunikat o błędzie. Szczegółowe informacje na temat obsługi błędów przedstawione zostaną w dalszej części ćwiczenia (patrz rozdział Detekcja błędów).

Czwartym możliwym parametrem jest opcjonalna tablica z parametrami, w której można podać informacje takie jak tryb obsługi błędu, rodzaj kodowania czy format zwracanego wyniku. Przykładowo ustawienie atrybutu ***PDO::ATTR_CASE*** wartości ***PDO:: ATTR_UPPER*** spowoduje, że nazwy kolumn w zbiorze wynikowym będą zapisane wielkimi literami.

```
...
$dsn = "admlab2-main.cs.put.poznan.pl:1521/dblab01.cs.put.poznan.pl";
$username = "scott";
$passwd = "tiger";
$opt = array(PDO::ATTR_CASE=>PDO::CASE_UPPER)
```

```
try
{
 $conn = new PDO("oci:dbname=".$dsn, $username, $passwd, $opt);
      echo ($e->getMessage());
}
...
```

Alternatywnym sposobem ustawienia atrybutów połączenia jest użycie metody ***PDO->setAttribute()***. Adresatem jest tu obiekt klasy PDO reprezentujący połączenie z bazą danych. Metodzie przekazujemy parametry w postaci *atrybut->wartość.* Na wyjściu zwracana jest wartość TRUE w przypadku sukcesu, FALSE w przeciwnym przypadku. Poniżej ten sam przykład z użyciem metody ***setAttribute().***

```
$conn->setAttribute(PDO::ATTR_CASE,PDO::CASE_UPPER);
```

Przykładowe atrybuty połączenia:

PDO::ATTR_ORACLE_NULLS	**konwersja pustych ciągów znaków na SQL'owe wartości NULL w zbiorze przetwarzanych danych**
PDO::ATTR_STRINGIFY_FETCHES	**konwersja wartości numerycznych na ciągi znaków**
PDO :: ATTR_TIMEOUT	**określa limit czasu połączenia**

Pełna lista możliwych atrybutów wraz z opisem dostępna na stronie: http://php.net/manual/en/pdo.constants.php

4.2 Zamknięcie połączenia z bazą danych następuje w wyniku utraty zakresu zmiennej reprezentującej połączenie bądź poprzez przypisanie zmiennej odpowiedzialnej za połączenie wartości null.

```
$conn = null;
```

Zadanie 1	**Połączenie z bazą danych**
Przygotuj skrypt, który nawiąże połączenie z bazą danych i wyświetli komunikat o pomyślnym wykonaniu zadania.	

5 Odczyt danych

5.1 Polecenia odczytujące dane możemy implementować na różne sposoby w zależności od potrzeb aplikacji. Jeśli polecenie jest wykonywane w aplikacji jednokrotnie możemy skorzystać z tzw. bezpośredniego wykonania polecenia. Polecenie jest wówczas parsowane przy każdym uruchomieniu skryptu. W przypadku poleceń modyfikujących (poleceń, w których nie ma potrzeby dostępu do zbioru wynikowego) wykorzystujemy metodę ***PDO->exec()***. Adresatem jest obiekt klasy PDO reprezentujący połączenie z bazą danych. Parametrem jest tekst polecenia SQL, natomiast wyjściem jest liczba przetworzonych wierszy.

```
$conn->exec("UPDATE pracownicy SET placa_pod=placa_pod*1.1");
```

```
```

5.2 W przypadku poleceń, w których oczekujemy zbioru wynikowego zapytania korzystamy z metody ***PDO->query().*** Adresatem jest obiekt klasy PDO reprezentujący połączenie z bazą danych. Parametrem jest tekst polecenia SQL, natomiast na wyjściu otrzymujemy obiekt klasy PDOStatement reprezentujący wynik zapytania.

```
$conn->query("SELECT * FROM pracownicy");
```

5.3 Innym sposobem wykonywania poleceń jest wykonanie przegotowanego polecenia. Polecenie realizuje się w dwóch krokach. W pierwszym następuje przygotowanie polecenia. Wykorzystujemy do tego metodę ***PDO->prepare().*** Adresatem jest obiekt klasy PDO reprezentujący połączenie z bazą danych. Jako parametr przekazujemy tekst polecenia SQL. Na wyjściu otrzymujemy obiekt klasy PDOstatement reprezentujący przygotowane polecenie. W drugim kroku wykorzystujemy metodę ***PDOStatement->execute()*** do wykonania przygotowanego polecenia. Adresatem jest obiekt klasy PDOStatement reprezentujący przygotowane polecenie. Jako parametr przekazujemy tablicę ze zmiennymi wiązania dla parametrów wejściowych.

```
$stmt = $conn->prepare("SELECT * FROM pracownicy WHERE id_prac = ?");
$stmt->execute(array("100"));
```

Przy wykorzystaniu tego sposobu wykonania poleceń należy pamiętać o następujących ograniczeniach:
Metoda ***execute()*** oczekuje zawsze tablicy, nawet jeśli jest to pojedyncza wartość.
Typ danych wartości parametru musi być znakowy (patrz powyższy przykład).

5.4 Zmienne wiązania wykorzystywane w momencie przygotowania polecenia możemy oznaczyć znakiem '**?**' lub poprzez podanie nazwy zmiennej **:nazwa_zmiennej.** Aby związać parametr ze zmienną wiązania wykorzystujemy metodę ***PDOStatement->bindParam().*** Obiektowi klasy PDOStatement reprezentującemu przygotowane polecenie przekazujemy w postaci parametrów:
pozycję parametru bądź jego nazwę
zmienną wiązania
typ przetwarzanych danych dla parametru + tryb IN/OUT (opcjonalnie)
długość parametru typu OUT

Tabelka przedstawia dostępne wartości trzeciego parametru w zależności od przetwarzanego typu danych:

PDO::PARAM_BOOL	**reprezentuje typ logiczny**
PDO::PARAM_NULL	**reprezentuje wartość pustą**
PDO::PARAM_INT	**reprezentuje typ SQL INTEGER**
PDO::PARAM_STR	**reprezentuje typy SQL CHAR, SQL VARCHAR**
PDO::PARAM_LOB	**reprezentuje duży obiekt (LOB)**
PDO::PARAM_STMT	**reprezentuje zbiór wynikowy, aktualnie nie wspierany przez żaden driver**
PDO::PARAM_INPUT_OUTPUT	**określa jako parametr rodzaju INOUT dla składowanych procedur. Należy użyć bitwise-OR tej wartości i typu danych PDO::PARAM_***

Poniższy przykład ilustruje użycie polecenia przygotowanego z wykorzystaniem zmiennych wiązania wykonanym na różne sposoby.

```
$stmt = $conn->prepare("SELECT * FROM pracownicy WHERE id_zesp=:id_zesp
and etat=:etat ");

$id_zesp=10;
$etat='ASYSTENT';
$stmt->bindParam(':id_zesp',$id_zesp)
$stmt->bindParam(':etat',$etat)

$stmt->execute();
```

```
$stmt = $conn->prepare("SELECT * FROM pracownicy WHERE id_zesp=? and
etat=? ");

$id_zesp=10;
$etat='ASYSTENT';

$stmt->bindParam(1,$id_zesp)
$stmt->bindParam(2,$etat)

$stmt->execute();
```

5.5 Po wykonaniu polecenia otrzymujemy zbiór wynikowy zawierający przetworzone dane. Aby odczytać liczbę kolumn w zbiorze wynikowym korzystamy z metody ***PDOStatement->ColumnCount()***. Adresatem jest obiekt klasy PDOStatement reprezentujący wykonane polecenie. Wyjściem jest liczba kolumn zbioru wynikowego, w przypadku braku zbioru wynikowego pojawia się 0. Oprócz liczby kolumn zbioru wynikowego możemy za pomocą metody ***PDOStatement->getColumnMeta()*** otrzymać informacje na temat metadanych kolumn. Adresatem metody jest obiekt klasy PDOStatement reprezentujący wykonane polecenie. Jako parametr podajemy numer kolumny (numerowany od zera). Na wyjściu otrzymujemy tablicę asocjacyjną zawierającą następujące informacje: natywny (PHP) typ danych kolumny wykorzystany do reprezentacji danych z kolumny, typ SQL, flagi ustawione dla kolumny, nazwa kolumny, długość, precyzja, typ PDO.

```
...

$stmt = $conn->query(SELECT * FROM pracownicy);
echo "<br/>"."Liczba kolumn w tabeli PRACOWNICY".$stmt->columnCount();
...
```

5.6 Aby móc wyświetlić dane ze zbioru wynikowego wykorzystujemy metodę ***PDOStatement->fetch()*** do przeglądania zbioru wynikowego. Metoda ta pobiera kolejny wiersz zbioru wynikowego. PDO zwraca kolumny zbioru wynikowego jako wartości typu string. Adresatem metody jest obiekt klasy PDOStatement reprezentujący polecenie. Jako parametr określamy typ wartości zwrotnej. Istnieje wiele możliwych wartości parametrów metody fetch(), poniżej zostały przedstawione najczęściej używane wartości parametrów.

PDO::FETCH_NUM	**tablica indeksowana numerem pozycji kolumny w zbiorze wynikowym (od 0), w każdym elemencie tablicy znajduje się kolejna wartość kolumny pobranego wiersza**
PDO::FETCH_NAMED	**tablica asocjacyjna indeksowana nazwami kolumn w zbiorze wynikowym. Jeśli w zbiorze znajdują się kolumny o powtarzających**

	się nazwach zwraca zwraca tablicę wartości dla kolumn o powtarzających się nazwach
PDO::FETCH_ASSOC	**tablica asocjacyjna indeksowana nazwami kolumn w zbiorze wynikowym. Jeśli w zbiorze znajdują się kolumny o powtarzających się nazwach zwraca tylko jedną kolumnę o powtarzającej się nazwie**
PDO::BOTH	**tablica indeksowana zarówno pozycjami jak i nazwami kolumn**
PDO::FETCH_OBJ	**obiekt anonimowy, którego nazwy własności odpowiadają nazwom kolumn**
PDO::FETCH_LAZY	**kombinacja PDO::BOTH oraz PDO::FETCH_OBJ, własności obiektu są tworzone w momencie odwołania się do nich**
PDO::FETCH_BOUND	**zwraca TRUE, jeżeli powiodło się pobranie kolejnego wiersza do zmiennych związanych z kolumnami zbioru wynikowego (patrz: PDOStatement->bindColumn)**
PDO::FETCH_COLUMN	**zwraca tylko pojedynczą kolumnę ze zbioru wynikowego**
PDO::FETCH_CLASS	**zwraca nową instancję klasy mapując kolumny z ich właściwościami**

Pełna lista dostępna pod adresem http://php.net/manual/en/pdo.constants.php

Wartością wyjściową jest wartość zwrotna o typie zgodnym z podaną wartością parametru. Poniżej przykład użycia funkcji ***fetch()*** z parametrem ***PDO::FETCH_ASSOC***.

```
...
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        echo "Nazwisko $row[NAZWISKO], etat $row[ETAT]<br>"; }
...
```

Przykład użycia funkcji ***fetch()*** z parametrem ***PDO::FETCH_OBJ.***

```
...
while ($row = $stmt->fetch(PDO::FETCH_OBJ)) {
        echo "Nazwisko {$row->NAZWISKO}, etat {$row->ETAT}<br>"; }
...
```

Inną metodą, którą wykorzystujemy do przeglądania zbioru wynikowego jest metoda ***PDOStatement->bindColumn()***. Metoda ta wiąże zmienne z kolumnami zbioru wynikowego. Jako parametr podajemy numer pozycji (od 1) lub nazwa kolumny zbioru wynikowego, zmienną wiązaną z kolumną zbioru wynikowego, bądź typ danych zmiennej wiązania (opcja). Na wyjściu otrzymujemy wartość TRUE w przypadku powodzenia, FALSE w przeciwnym przypadku.

```
...
 $stmt->bindColumn('NAZWISKO', $nazwisko);
 $stmt->bindColumn('ETAT', $etat);
 while ($stmt->fetch(PDO::FETCH_BOUND)) {
      echo "Nazwisko $nazwisko, etat $etat\n";
 }
...
```

5.7 Mamy już pełen zestaw metod do stworzenia skryptu przetwarzającego zapytanie SQL i wyświetlenia ich w oknie przeglądarki. Wykonajmy przykład, ilustrujący pobieranie informacji z bazy danych o nazwiskach i etatach pracowników zatrudnionych w zespole 20. Do przeglądania zbioru wynikowego wykorzystaliśmy metodę ***fetch()*** z parametrem ***PDO::FETCH_NUM*** (dostęp po numerze kolumny w zbiorze wynikowym).

```
...
//przygotowanie polecenia
$stmt = conn->prepare("SELECT nazwisko,zatrudniony FROM pracownicy WHERE
```

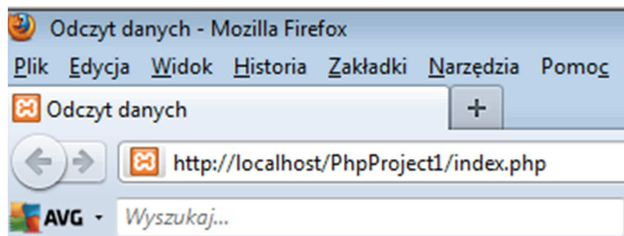
```
id_zesp=? ");

$id_zesp=20;
$stmt->bindParam(1,$id_zesp)

//wykonanie przygotowanego polecenia
$stmt->execute();

//pobranie danych ze zbioru wynikowego i wyświetlenie ich w przeglądarce
while ($row = $stmt->fetch(PDO::FETCH_NUM)) {
      echo "Pracownik $row[0] zatrudniony $row[1]<br/>";
 }
...
```

Po uruchomieniu skryptu powinniśmy otrzymać poniższy wynik:



Połączono z bazą danych!
Pracownik BRZEZINSKI zatrudniony 68/07/01
Pracownik MORZY zatrudniony 75/09/15
Pracownik KROLIKOWSKI zatrudniony 77/09/01
Pracownik KOSZLAJDA zatrudniony 85/03/01
Pracownik JEZIERSKI zatrudniony 92/10/01
Pracownik MATYSIAK zatrudniony 93/09/01
Pracownik KONOPKA zatrudniony 93/10/01

Zadanie 2 Odczyt danych

Przygotuj skrypt wyświetlający nazwiska pracowników zatrudnionych na stanowisku, którego nazwa zostanie przekazana jako parametr wywołania metody GET protokołu HTTP (np. http:/host/skrypt.php?parametr1=wartosc¶metr2=wartosc). Wynik uporządkuj wg daty zatrudnienia. Wykorzystaj zmienne wiązania. Wskazówka: do odczytu wartości parametrów metody GET wykorzystaj zmienną $_GET, np. $_GET['parametr1']

6 Detekcja błędów

6.1 PDO umożliwia trzy alternatywne tryby obsługi błędów. Tryb błędu ustalamy poprzez ustawienie atrybutów połączenia ***PDO->setAttribute().***

```
$conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_SILENT);
```

Rozważmy przykład ilustrujący działanie wszystkich trzech trybów obsługi błędów. Zakładamy, że tabela PRAC nie istnieje w schemacie użytkownika scott:

6.2 ***PDO::ERRMODE_SILENT*** jest trybem domyślnym. PDO ustawia kod błędu, który odczytujemy za pomocą metody ***PDO->errorCode()***. Na poziomie poleceń ***PDOStatement->errorCode()*** zwraca SQLCODE w postaci pięcioznakowego, alfanumerycznego ciągu według standardu ANSI SQL-92.

Oprócz kodu błędu informację o jego treści można uzyskać wykorzystując metodę ***PDO->errorInfo()***. Na poziomie poleceń metoda ***PDOStatement->errorInfo()*** zwraca tablicę trójelementową, która zawiera następujące informacje: SQLCODE, numer błędu oraz tekst błędu specyficzny dla źródła danych.

```
<?php
...
$conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_SILENT);
$conn->query("SELECT * FROM PRAC");
echo "errorCode:";
print $conn->errorCode();
echo "errorInfo:";
print_r($conn->errorInfo());
...
?>
```

6.3 ***PDO::ERRMODE_WARNING*** jest trybem w którym oprócz kodu błędu, PDO emituje tradycyjny komunikat **E_WARNING,** który jest przydatny przy testowaniu aplikacji.

```
<?php
...
$conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_WARNING);
$conn->query("SELECT * FROM PRAC");
...
?>
```

6.4 ***PDO::ERRMODE_EXCEPTION*** oprócz ustawienia kodu błędu, PDO zgłasza wyjątek PDOException, ułatwia czytelną organizację kodu obsługującego błędy.

```
<?php
try{
...
$conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
$conn->query("SELECT * FROM PRAC");
}
catch (PDOException $e){
echo "Błąd wykonania: ".$e->getMessage();
}
...
?>
```

Zadanie 3 Odczyt danych z detekcją błędów

Przygotuj skrypt, który zwróci podstawowe informacje o pracownikach (nazwisko, etat, data zatrudnienia) zespołu podanego jako parametr (nazwa zespołu). Zasymuluj błąd – podaj w zapytaniu błędną nazwę atrybutu. Przechwyć błąd i wypisz odpowiedni komunikat. Dodatkowo, wyświetl odpowiedni komunikat w przypadku, gdy zespół nie będzie zatrudniał żadnych pracowników.

7 Kursory przewijane

7.1 Zbiór wynikowy przetwarzamy za pomocą kursora korzystając z metody ***fetch().*** Oprócz kursora zwykłego, w którym odczytujemy wiersz po wierszu (w jednym kierunku), w PDO jest możliwość korzystania z kursora przewijanego, który umożliwia poruszanie się po zbiorze wynikowym w różnych kierunkach. Domyślnie używamy kursor zwykły (jednokierunkowy), możemy to zmienić ustawiając atrybut kursora na kursor przewijany (***PDO::CURSOR_SCROLL***)

```
...
$conn->setAttribute(PDO::ATTR_CURSOR,PDO::CURSOR_SCROLL);
```

```
...
```

lub bezpośrednio przy przetwarzanym poleceniu, w którym ma być przetwarzany kursor przewijany:

```
...
$stmt = $conn->prepare($sql, array(PDO::ATTR_CURSOR =>
PDO::CURSOR_SCROLL));
...
```

W metodzie ***fetch(),*** która pobiera wiersz zbioru wynikowego podajemy jako pierwszy parametr stałą określającą typ wartości zwrotnej, drugi parametr nadaje kierunek pobierania oraz opcjonalny trzeci parametr, w którym podajemy przesunięcie pobierania.

Dopuszczalne wartości nadawania kierunku (drugi parametr):

PDO::FETCH_ORI_NEXT	**pobiera następny wiersz**
PDO::FETCH_ORI_PRIOR	**pobiera poprzedni wiersz**
PDO::FETCH_ORI_FIRST	**pobiera pierwszy wiersz**
PDO::FETCH_ORI_LAST	**pobiera ostatni wiersz**
PDO::FETCH_ORI_ABS	**pobiera wiersz wskazany za pomocą bezwzględnego przesunięcia pobierania**
PDO::FETCH_ORI_REL	**pobiera wiersz wskazany za pomocą przesunięcia pobierania względem bieżącej pozycji kursora**

7.2 Poniższy przykład ilustruje odczyt danych z wykorzystaniem kursora przewijanego.

```
...
$sql = 'SELECT nazwisko, etat FROM pracownicy ORDER BY placa_pod';
  try {
    $stmt = $conn->prepare($sql, array(PDO::ATTR_CURSOR =>
PDO::CURSOR_SCROLL));
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_LAST);
    do {
      $data = $row[0] . "\t" . $row[1] . "\n";
      print $data;
    } while ($row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_PRIOR));
    $stmt = null;
  }
  catch (PDOException $e) {
    print $e->getMessage();
  }
}
...
```

Zadanie 4 Kursory przewijane

Napisz program, który wykonuje zapytanie odnajdujące wszystkich pracowników zatrudnionych na etacie ASYSTENT i sortuje ich malejąco według pensji, a następnie wyświetla asystenta, który zarabia najmniej, trzeciego najmniej zarabiającego asystenta i przedostatniego asystenta w rankingu najmniej zarabiających asystentów (użyj do tego celu kursorów przewijanych).

8 Modyfikacja danych

8.1 Modyfikacji danych w źródle dokonuje się zarówno przez polecenie wykonywane bezpośrednio ***PDO->exec()*** lub polecenia przygotowane ***PDOStatement->execute().*** Polecenia modyfikujące dane nie tworzą zbioru wynikowego, aby sprawdzić powodzenie operacji możemy odczytać liczbę wierszy, które zostały zmodyfikowane. Do tego celu używamy metody ***PDOStatement->rowCount().***

```
...
$rowCount = $conn->exec("update pracownicy set placa_pod=placa_pod+1");
echo $rowCount;
...
```

```
...
$stmt = $conn->prepare("update pracownicy set placa_pod=placa_pod+1");
$stmt->execute();
echo $stmt->rowCount();
...
```

Zadanie 5	**Modyfikacja danych**
Napisz skrypt, który wszystkim asystentom zwiększy pensję o 20%.	

9 Obsługa transakcji

9.1 W PDO mamy dwa tryby zatwierdzania transakcji. Domyślnym jest tryb automatyczny, w którym każde polecenie jest osobną transakcją, zatwierdzaną zaraz po zakończeniu jego wykonywania. Drugi tryb to tryb ręczny, w którym jawnie wskazujemy moment rozpoczęcia transakcji poprzez wywołanie metody ***PDO::beginTransaction().*** W tym trybie koniec transakcji następuje poprzez wywołanie metody ***PDO::commit()*** (zatwierdzenie transakcji) bądź ***PDO::rollBack()*** (anulowanie transakcji).

```
...
$conn->beginTransaction();

//rozpoczęcie transakcji, wyłącznie automatycznego trybu zatwierdzania
transakcji

$stmt = $conn->prepare("UPDATE pracownicy SET placa_pod=placa_pod*1.1");
$stmt->execute()
echo $stmt->rowCount();

$conn->commit();
//zatwierdzenie transakcji, powrót do trybu autocommit
...
```

9.2 Należy również pamiętać o tym, że niektóre bazy danych (np. Oracle, MySQL) operacje DDL wykonują w osobnej transakcji. Dlatego poniższy przykład ilustruje sytuację, w której zmiany nie będą wycofane ponieważ zostają zatwierdzone w wyniku działania operacji DDL.

```
...
$conn->beginTransaction();

$stmt = $conn->exec("UPDATE pracownicy SET placa_pod=placa_pod*1.1");
$stmt->exec("DROP TABLE zespoly");

$conn->rollBack();
...
```

Zadanie 6 Przetwarzanie transakcyjne

Przygotuj skrypt, który wykona następujące czynności:

- Wyłącz automatyczne zatwierdzanie transakcji.
- Wyświetl wszystkie etaty.
- Wstaw nowy etat do tabeli ETATY.
- Ponownie wyświetl wszystkie etaty.
- Wycofaj transakcję.
- Ponownie wyświetl wszystkie etaty.
- Wstaw nowy etat do tabeli ETATY.
- Zatwierdź transakcję.

Ponownie wyświetl wszystkie etaty

10 Wywoływanie procedur i funkcji

10.1 Procedury i funkcje w PDO wykonujemy przy pomocy poleceń odczytujących dane. Parametry procedury bądź funkcji związujemy ze zmienną bądź nadajemy wartość parametrom wejściowym poprzez użycie funkcji ***bindParam().***

10.2 Rozważmy następujący przykład ilustrujący wywołanie procedury składowanej z poziomu skryptu php. Użytkownik scott posiada w swoim schemacie procedurę o nazwie AVG_ZESP, która dla podanego zespołu wylicza średnie w nim zarobki. Procedura posiada parametr wejściowy (ID_Z) – identyfikator zespołu, w którym liczone są średnie zarobki oraz parametr wyjściowy (WYNIK). W pierwszym kroku przygotowujemy polecenie, które wywoła naszą procedurę składowaną z odpowiednimi parametrami. Wykorzystujemy zmienne wiązania oraz metodę **bindParam()** do związania odpowiednio wartości wejściowej (identyfikator zespołu), oraz wyjściowej średniej.

```
<?php
$stmt = $conn->prepare("CALL AVG_ZESP(?,?)");
$id_z=10;
$stmt->bindParam(1, $id_z, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT,10);
$stmt->bindParam(2, $wynik, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT, 10);

$stmt->execute();

print "Wynik: $wynik\n";
?>
```

10.3 Analogicznie postępujemy z funkcją. W schemacie użytkownika scott istnieje funkcja PODATEK, która dla podanego identyfikatora pracownika (parametr wejściowy) wyliczy jego roczny podatek.

```
<?php
$stmt = $conn->prepare("CALL PODATEK(?) INTO :wynik");
$stmt->bindParam(":wynik", $value, PDO::PARAM_INT, 10);

$stmt->execute();

print "Wynik: $value\n";
?>
```

Zadanie 7 Wywołanie procedur i funkcji

Napisz procedurę PODWYZKA, która dla podanego etatu (parametr wejściowy) dokona podwyżki o określony procent (parametr wejściowy). Napisz skrypt który ją wywoła oraz

wyświetli wyniki przed modyfikacją oraz po modyfikacji.

Zadanie podsumowujące

Zadanie 8

Przygotuj skrypt, który wyświetli raport matrycowy dotyczący zatrudnienia pracowników. Wynikiem ma być tabelka, pierwszy wiersz ma zawierać nazwy zespołów uporządkowane alfabetycznie od lewej do prawej strony, pierwsza kolumna ma zawierać nazwy etatów uporządkowane alfabetycznie z góry na dół, na przecięciu odpowiedniej kolumny i wiersza powinna się znajdować informacja o liczbie zatrudnionych pracowników w danym zespole na danym etacie.