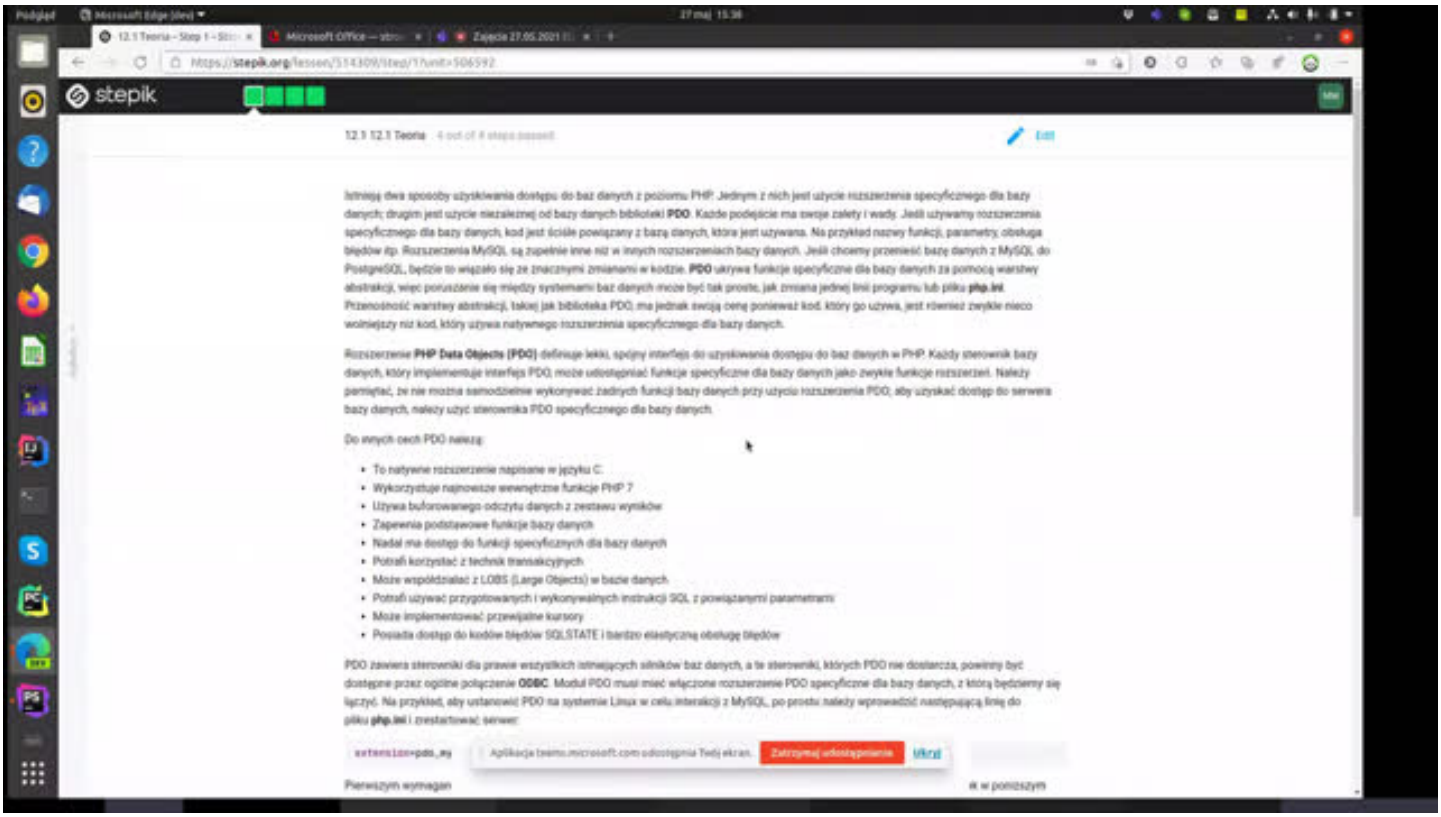
13.2 13.2 Teoria - PDO

Step 1



To watch this video please visit https://stepik.org/lesson/514309/step/1

Step 2

Istnieją dwa sposoby uzyskiwania dostępu do baz danych z poziomu PHP. Jednym z nich jest użycie rozszerzenia specyficznego dla bazy danych; drugim jest użycie niezależnej od bazy danych biblioteki **PDO**. Każde podejście ma swoje zalety i wady. Jeśli używamy rozszerzenia specyficznego dla bazy danych, kod jest ściśle powiązany z bazą danych, która jest używana. Na przykład nazwy funkcji, parametry, obsługa błędów itp. Rozszerzenia MySQL są zupełnie inne niż w innych rozszerzeniach bazy danych. Jeśli chcemy przenieść bazę danych z MySQL do PostgreSQL, będzie to wiązało się ze znacznymi zmianami w kodzie. **PDO** ukrywa funkcje specyficzne dla bazy danych za pomocą warstwy abstrakcji, więc poruszanie się między systemami baz danych może być tak proste, jak zmiana jednej linii programu lub pliku **php.ini**. Przenośność warstwy abstrakcji, takiej jak biblioteka PDO, ma jednak swoją cenę ponieważ kod, który go używa, jest również zwykle nieco wolniejszy niż kod, który używa natywnego rozszerzenia specyficznego dla bazy danych.

Rozszerzenie **PHP Data Objects (PDO)** definiuje lekki, spójny interfejs do uzyskiwania dostępu do baz danych w PHP. Każdy sterownik bazy danych, który implementuje interfejs PDO, może udostępniać funkcje specyficzne dla bazy danych jako zwykłe funkcje rozszerzeń. Należy pamiętać, że nie można samodzielnie wykonywać żadnych funkcji bazy danych przy użyciu rozszerzenia PDO; aby uzyskać dostęp do serwera bazy danych, należy użyć sterownika PDO specyficznego dla bazy danych.

Do innych cech PDO należą:

- To natywne rozszerzenie napisane w języku C.
- Wykorzystuje najnowsze wewnętrzne funkcje PHP 7
- Używa buforowanego odczytu danych z zestawu wyników
- Zapewnia podstawowe funkcje bazy danych
- Nadal ma dostęp do funkcji specyficznych dla bazy danych

- Potrafi korzystać z technik transakcyjnych
- Może współdziałać z LOBS (Large Objects) w bazie danych
- Potrafi używać przygotowanych i wykonywalnych instrukcji SQL z powiązanymi parametrami
- Może implementować przewijalne kursory
- Posiada dostęp do kodów błędów SQLSTATE i bardzo elastyczną obsługę błędów

PDO zawiera sterowniki dla prawie wszystkich istniejących silników baz danych, a te sterowniki, których PDO nie dostarcza, powinny być dostępne przez ogólne połączenie **ODBC**. Moduł PDO musi mieć włączone rozszerzenie PDO specyficzne dla bazy danych, z którą będziemy się łączyć. Na przykład, aby ustanowić PDO na systemie Linux w celu interakcji z MySQL, po prostu należy wprowadzić następującą linię do pliku **php.ini** i zrestartować serwer:

```
extension=pdo_mysql
```

Pierwszym wymaganiem dla PDO jest nawiązanie połączenia z daną bazą danych i utrzymanie tego połączenia w zmiennej, jak w poniższym kodzie:

```
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    $db = new PDO("mysql:host=localhost;dbname=TUTORIALS", $dbuser,$dbpass)
?>
```

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.

Step 3

Wróćmy do bazy `classics`, którego zawartość tabeli wygląda następująco:

```
+--------------------+-----------------------------+-----------------+------+---------------+
| author             | title                       | category        | year | isbn          |
+--------------------+-----------------------------+-----------------+------+---------------+
| Charles Dickens    | The Old Curiosity Shop      | Classic Fiction | 1841 | 9780099533474 |
| William Shakespeare | Romeo and Juliet           | Play            | 1594 | 9780192814968 |
| Charles Darwin     | The Origin of Species       | Non-Fiction     | 1856 | 9780517123201 |
| Jane Austen        | Pride and Prejudice         | Classic Fiction | 1811 | 9780582506206 |
| Mark Twain         | The Adventures of Tom Sawyer | Classic Fiction | 1876 | 9781598184891 |
+--------------------+-----------------------------+-----------------+------+---------------+
```

Po nawiązaniu połączenia z silnikiem bazy danych i bazą danych, z którą chcesz współdziałać, możemy użyć tego połączenia do wysyłania poleceń SQL do serwera. Prosta instrukcja `UPDATE` wyglądałaby następująco:

```
<html>
<head>
    <title>Getting MySQL Database Info with PDO</title>
</head>
<body>
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    $db = new PDO("mysql:host=localhost;dbname=publications", $dbuser,$dbpass);
    $db->query("Update classics SET author='New Mark Twain' WHERE year=1876");

?>
</body>
</html>
```

```
+--------------------+-----------------------------+-----------------+------+---------------+
| author             | title                       | category        | year | isbn          |
+--------------------+-----------------------------+-----------------+------+---------------+
| Charles Dickens    | The Old Curiosity Shop      | Classic Fiction | 1841 | 9780099533474 |
| William Shakespeare| Romeo and Juliet            | Play            | 1594 | 9780192814968 |
| Charles Darwin     | The Origin of Species       | Non-Fiction     | 1856 | 9780517123201 |
| Jane Austen        | Pride and Prejudice         | Classic Fiction | 1811 | 9780582506206 |
| New Mark Twain     | The Adventures of Tom Sawyer| Classic Fiction | 1876 | 9781598184891 |
+--------------------+-----------------------------+-----------------+------+---------------+
```

Ten kod po prostu aktualizuje tabelę książek i zwalnia zapytanie. Pozwala to na wysyłanie prostych poleceń SQL (np. `UPDATE`, `DELETE`, `INSERT`) bezpośrednio do bazy danych.

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);
```

```
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    $db = new PDO("mysql:host=localhost;dbname=publications", $dbuser,$dbpass);
    $query = "SELECT * FROM classics";
    $result = $db->query($query);
    if (!$result){
        echo "BŁĄD w zapytaniu";
        exit;
    }
    while($row = $result->fetch(PDO::FETCH_ASSOC)){
        print_r($row);
    }
?>
```

```
Array
(
    [author] => Charles Dickens
    [title] => The Old Curiosity Shop
    [category] => Classic Fiction
    [year] => 1841
    [isbn] => 9780099533474
)
Array
(
    [author] => William Shakespeare
    [title] => Romeo and Juliet
    [category] => Play
    [year] => 1594
    [isbn] => 9780192814968
)
Array
(
    [author] => Charles Darwin
    [title] => The Origin of Species
    [category] => Non-Fiction
    [year] => 1856
    [isbn] => 9780517123201
)
Array
(
    [author] => Jane Austen
    [title] => Pride and Prejudice
    [category] => Classic Fiction
    [year] => 1811
    [isbn] => 9780582506206
)
Array
(
    [author] => New Mark Twain
    [title] => The Adventures of Tom Sawyer
    [category] => Classic Fiction
    [year] => 1876
    [isbn] => 9781598184891
)
```

```php
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    $db = new PDO("mysql:host=localhost;dbname=publications", $dbuser,$dbpass);
    $query = "SELECT * FROM classics";
    $result = $db->query($query);
    if (!$result){
        echo "BŁĄD w zapytaniu";
        exit;
    }
    $result->setFetchMode(PDO::FETCH_OBJ);
    while($row = $result->fetch()){
        print_r($row);
    }
?>
```

```
stdClass Object
(
    [author] => Charles Dickens
    [title] => The Old Curiosity Shop
    [category] => Classic Fiction
    [year] => 1841
    [isbn] => 9780099533474
)
stdClass Object
(
    [author] => William Shakespeare
    [title] => Romeo and Juliet
    [category] => Play
    [year] => 1594
    [isbn] => 9780192814968
)
stdClass Object
(
    [author] => Charles Darwin
    [title] => The Origin of Species
    [category] => Non-Fiction
    [year] => 1856
    [isbn] => 9780517123201
)
stdClass Object
(
    [author] => Jane Austen
    [title] => Pride and Prejudice
    [category] => Classic Fiction
    [year] => 1811
    [isbn] => 9780582506206
)
stdClass Object
(
    [author] => New Mark Twain
    [title] => The Adventures of Tom Sawyer
    [category] => Classic Fiction
    [year] => 1876
    [isbn] => 9781598184891
)
```

Zwykle będziemy używać tzw. **prepared statements** w PDO. Rozważmy następujący kod:

```
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    $db = new PDO("mysql:host=localhost;dbname=publications", $dbuser,$dbpass);
    $statement = $db->prepare("SELECT * FROM classics");
    $statement->execute();
    while($row = $statement->fetch()){
        print_r($row);
    }
    $statement = null;

?>
```

```
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    $db = new PDO("mysql:host=localhost;dbname=publications", $dbuser,$dbpass);
    $statement = $db->prepare("SELECT * FROM classics");
    $statement->execute();
    var_dump($statement->fetchAll());
    $statement = null;
?>
```

W tym kodzie kod SQL jest przygotowywany, a następnie wykonywany. Następnie przechodzimy przez wynik z kodem while i na koniec zwalniamy obiekt, przypisując mu wartość `null`. To może nie wyglądać tak potężnie w tym prostym przykładzie, ale istnieją inne funkcje, których można użyć z przygotowanymi instrukcjami. Rozważmy teraz ten kod:

```php
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    $db = new PDO("mysql:host=localhost;dbname=publications", $dbuser,$dbpass);
    $statement = $db->prepare("INSERT INTO classics(author, title, category, year, isbn)" .
    "VALUES (:author, :title, :category, :year, :isbn)");
    $statement->execute(array(
        "author" => "Mateusz Miotk",
        "title" => "Introduce to Programming in PHP",
        "category" => "IT",
        "year" => 2021,
        "isbn" => "9781598184895",
    ));
?>
```

```
+--------------------+---------------------------------+-----------------+------+---------------+
| author             | title                           | category        | year | isbn          |
+--------------------+---------------------------------+-----------------+------+---------------+
| Charles Dickens    | The Old Curiosity Shop          | Classic Fiction | 1841 | 9780099533474 |
| William Shakespeare | Romeo and Juliet               | Play            | 1594 | 9780192814968 |
| Charles Darwin     | The Origin of Species           | Non-Fiction     | 1856 | 9780517123201 |
| Jane Austen        | Pride and Prejudice             | Classic Fiction | 1811 | 9780582506206 |
| New Mark Twain     | The Adventures of Tom Sawyer    | Classic Fiction | 1876 | 9781598184891 |
| Mateusz Miotk      | Introduce to Programming in PHP | IT              | 2021 | 9781598184895 |
+--------------------+---------------------------------+-----------------+------+---------------+
```

Tutaj przygotowujemy instrukcję SQL z nazwanymi symbolami zastępczymi: `author`, `title`, `category`, `year` oraz `isbn`. W tym przypadku są to te same nazwy, co kolumny w bazie danych, ale jest to zrobione tylko dla przejrzystości - nazwy symboli zastępczych mogą być dowolne. W momencie wywołania zapytania zamieniamy te symbole zastępcze na rzeczywiste dane, które chcemy wykorzystać w tym konkretnym przypadku. Jedną z zalet prepared statemets jest to, że można wykonać to samo polecenie SQL i za każdym razem przekazać różne wartości przez tablicę. Możemy również wykonać tego typu przygotowanie instrukcji z pozycyjnymi symbolami zastępczymi (nie nazywając ich w rzeczywistości), oznaczanymi przez symbol `?`, który jest pozycją do zastąpienia, co pokazuje następny kod:

```php
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    $db = new PDO("mysql:host=localhost;dbname=publications", $dbuser,$dbpass);
    $statement = $db->prepare("INSERT INTO classics(author, title, category, year, isbn)" .
    "VALUES (?, ?, ?, ?, ?)");
    $statement->execute(array(
        "Mateusz Miotk",
        "Introduce to Programming in PHP",
        "IT",
        2022,
        "9781598184845",
    ));
?>
```

```
+--------------------+---------------------------------+-----------------+------+---------------+
| author             | title                           | category        | year | isbn          |
+--------------------+---------------------------------+-----------------+------+---------------+
| Charles Dickens    | The Old Curiosity Shop          | Classic Fiction | 1841 | 9780099533474 |
| William Shakespeare | Romeo and Juliet               | Play            | 1594 | 9780192814968 |
| Charles Darwin     | The Origin of Species           | Non-Fiction     | 1856 | 9780517123201 |
| Jane Austen        | Pride and Prejudice             | Classic Fiction | 1811 | 9780582506206 |
| Mateusz Miotk      | Introduce to Programming in PHP | IT              | 2022 | 9781598184845 |
| New Mark Twain     | The Adventures of Tom Sawyer    | Classic Fiction | 1876 | 9781598184891 |
| Mateusz Miotk      | Introduce to Programming in PHP | IT              | 2021 | 9781598184895 |
+--------------------+---------------------------------+-----------------+------+---------------+
```

Rezultat powyższego kodu jest taki sam jak poprzednio, ale przy mniejszej ilości kodu, ponieważ obszar wartości instrukcji SQL nie określa nazw elementów do zastąpienia, a zatem tablica w instrukcji `execute` musi przesyłać tylko surowe dane bez nazw.

Step 4

Niektóre systemy bazodanowe obsługują **transakcje**, w których można zatwierdzić serię zmian w bazie danych (wszystkie zastosowane jednocześnie) lub wycofać (odrzucone, bez żadnych zmian wprowadzonych do bazy danych). Na przykład, gdy bank obsługuje przelew pieniężny, wypłata z jednego konta i wpłata na inne musi nastąpić razem - ani jedno, ani drugie powinno się odbywać bez drugiego i nie powinno być opóźnienia między tymi dwoma działaniami. PDO elegancko obsługuje transakcje dzięki strukturom `try ... catch`, tak jak poniżej:

```
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    try {
        $db = new PDO("mysql:host=localhost;dbname=publications", $dbuser,$dbpass);
    } catch (Exception $error){
        die("Connection failed: " . $error->getMessage());
    }
    try{
        $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); //Do raportowania błędów
        $db->beginTransaction();
        $db->exec("DELETE FROM classics WHERE author='Mateusz Miotk'");
        $db->commit();
    } catch (Exception $error) {
        $db->rollBack();
        echo "Transaction not completed: " . $error->getMessage();
    }
?>
```

Jeśli całość transakcji nie może zostać zakończona, żadna z nich nie zostanie zakończona i zostanie zgłoszony wyjątek. Jeśli wywołamy `commit()` lub `rollback()` w bazie danych, która nie obsługuje transakcje, metody zwracać będą `DB_ERROR`.

Interfejs PDO udostępnia metodę wyświetlania szczegółów instrukcji PDO, co może być przydatne do debugowania, jeśli coś pójdzie nie tak.

```
<?php
    $dbuser = 'root';
    $dbpass = 'silneHaslo123@';
    try {
        $db = new PDO("mysql:host=localhost;dbname=publications", $dbuser,$dbpass);
    } catch (Exception $error){
        die("Connection failed: " . $error->getMessage());
    }
    $author = "Mateusz Miotk";
    $statement = $db->prepare("SELECT title FROM classics WHERE author=?");
    $statement->bindParam(1,$author,PDO::PARAM_STR);
    $statement->execute();
    $statement->debugDumpParams();
?>
```

Wywołanie metody `debugDumpParams()` wyświetla następujące rezultaty:

```
SQL: [41] SELECT title FROM classics WHERE author=?
Sent SQL: [55] SELECT title FROM classics WHERE author='Mateusz Miotk'
Params:  1
Key: Position #0:
paramno=0
name=[0] ""
is_param=1
param_type=2
```

Step 5

Teraz użyjmy innego systemu bazodanowego jakim jest **PostgreSQL** (zainstalowany między innymi na serwerze szuflandia). Odpalenie konsoli **PostgreSQL** odbywa się za pomocą komendy `psql`. Najpierw utworzymy sobie tabelę o nazwie `customers`.

```
CREATE TABLE customers (firstname VARCHAR(128), secondname VARCHAR(128));
```

Następnie dodamy parę rekordów danych:

```
INSERT INTO customers(firstname, secondname) VALUES ('Mateusz', 'Miotk');
INSERT INTO customers(firstname, secondname) VALUES ('Jan', 'Kowalski');
INSERT INTO customers(firstname, secondname) VALUES ('Anna', 'Nowak');
INSERT INTO customers(firstname, secondname) VALUES ('Misio', 'Pysio');
```

Wyświetlenie danych (polecenie SELECT * FROM customers) wyświetli wówczas następujące dane:

```
 firstname | secondname
-----------+------------
 Mateusz   | Miotk
 Mateusz   | Miotk
 Jan       | Kowalski
 Anna      | Nowak
 Misio     | Pysio
```

Teraz utwórzmy formularz w PHP, który będzie dodawał do bazy danych imię i nazwisko. Do podłączenia się z bazą danych skorzystamy z PDO.

```
<?php
    //Ten kod musi być umieszczony na serwerze szuflandia
    $dbuser = 'mmiotk';
    $dbpass = '*********';
    $db = new PDO("pgsql:host=localhost;port=5434", $dbuser, $dbpass);
    $query = "SELECT * FROM customers";
    $result = $db->query($query);
    if (!$result){
        echo "BŁĄD w zapytaniu";
        exit;
    }
    $result->setFetchMode(PDO::FETCH_OBJ);
    while($row = $result->fetch()){
        print_r($row);
    }
?>
```

Uwaga. Aby korzystać na lokalnym komputerze z `pgsql` należy mieć zainstalowane (w systemie Linux) pakiet `php-pgsql`. Aby dowiedzieć się na jakim porcie uruchomiona jest baza danych należy w konsoli użyć polecenia `\conninfo`.

```
<html>
<head><title>Creating MySQL Database</title></head>
<body>
<form id="formualarz" method="post">
    <label>Imię: </label> <input type="text" name="firstname"> <br/>
    <label>Nazwisko: </label> <input type="text" name="secondname"> <br/>
    <input type="submit">
</form>
<?php
    if ((isset($_POST['firstname'])) && (isset($_POST['secondname']))){
        //Ten kod musi być umieszczony na serwerze szuflandia
        $dbuser = 'mmiotk';
        $dbpass = '*********';
        $db = new PDO("pgsql:host=localhost;port=5434", $dbuser, $dbpass);
        $query = "INSERT INTO customers(firstname, secondname) VALUES (?,?)";
        $statement = $db->prepare($query);
        $statement->execute(array(
                $_POST['firstname'],
                $_POST['secondname'],
        ));
    }
    else{
        die("Error!!!");
    }
?>
</body>
</html>
```

Uwaga. W razie problemów z autoryzacją z pgsql, najprościej po prostu zmienić hasło za pomocą komendy (w konsoli psql):

```
ALTER USER user_name WITH PASSWORD 'new_password';
```

Step 6

Jednym z zalet korzystania z `PDO` jest natywne wsparcie przy odpieraniu ataków typu `SQL Injection`. Rozważmy następujący kod:

```
<?php
  $connection = new mysqli("localhost", "root", "", "test");
  $login = "'Lukasz' OR 1=1";
  $query = "SELECT * FROM Persons WHERE name LIKE ${login}";
  echo $query . PHP_EOL;
  $result = $connection->query($query);
  $row = $result->fetch_assoc();
  print_r($row);
  if ($connection->errno) {
      printf("Error in database: %s<br />", $connection->error);
  }
  $connection->close();
```

W zmiennej o nazwie `$login` umieściliśmy kod `OR 1=1` co spowoduje wyświetlenie zapytania `SELECT * FROM Persons WHERE name LIKE 'Lukasz' OR 1=1`, w wyniku czego zostanie wykonane całe zapytania, ponieważ przeszedł warunek `1 = 1`.

Aby uniknąć takich sytuacji należy umieszczać zapytania w tzw. prepared statements lub korzystając z funkcji `bind_param`, które później będą kopiowane do zapytania `SQL`.