



# pasja-informatyki.pl

## Programowanie webowe

Mirosław Zelent

### Odcinek #6

ten, w którym zrozumiemy mechanikę  
współpracy skryptu PHP z bazą danych MySQL

# Spis treści

Tytułem wstępu .....	3
Instalacja pakietu XAMPP .....	3
Rozszerzenia PHP do komunikacji z bazą danych: mysql, mysqli, PDO.....	5
Dane dostępowe do bazy danych MySQL.....	7
Współpraca mysqli z bazą (wariant proceduralny) .....	9
Współpraca mysqli z bazą (wariant obiektowy).....	12
Zadania do samodzielnego wykonania.....	16
Test wiedzy z odcinka.....	18

## Tytułem wstępu

Najczęstszy kontekst użycia PHP, który spotkamy na egzaminie, to wykorzystać ten język do połączenia się z bazą danych, następnie przy pomocy zapytania SQL coś do tej bazy włożyć (albo wyjąć z niej) oraz w jakiś sposób przetworzyć te dane i po wszystkim pokazać wyniki na ekranie. W dzisiejszym epizodzie serii nauczymy się jak okodować współpracę PHP z bazą danych MySQL przy użyciu rozszerzenia MySQLi (w wariacie proceduralnym jak i obiektowym). Ten odcinek dedykuję każdemu uczniowi technikum, niezależnie od umiejętności i myślę że pomoże on napisać PHP wielu osobom, które nie podejrzewałyby siebie o to, że mogą taki egzamin z programowania zaliczyć. Tego Wam życzę! Bierzmy się do pracy!

## Instalacja pakietu XAMPP

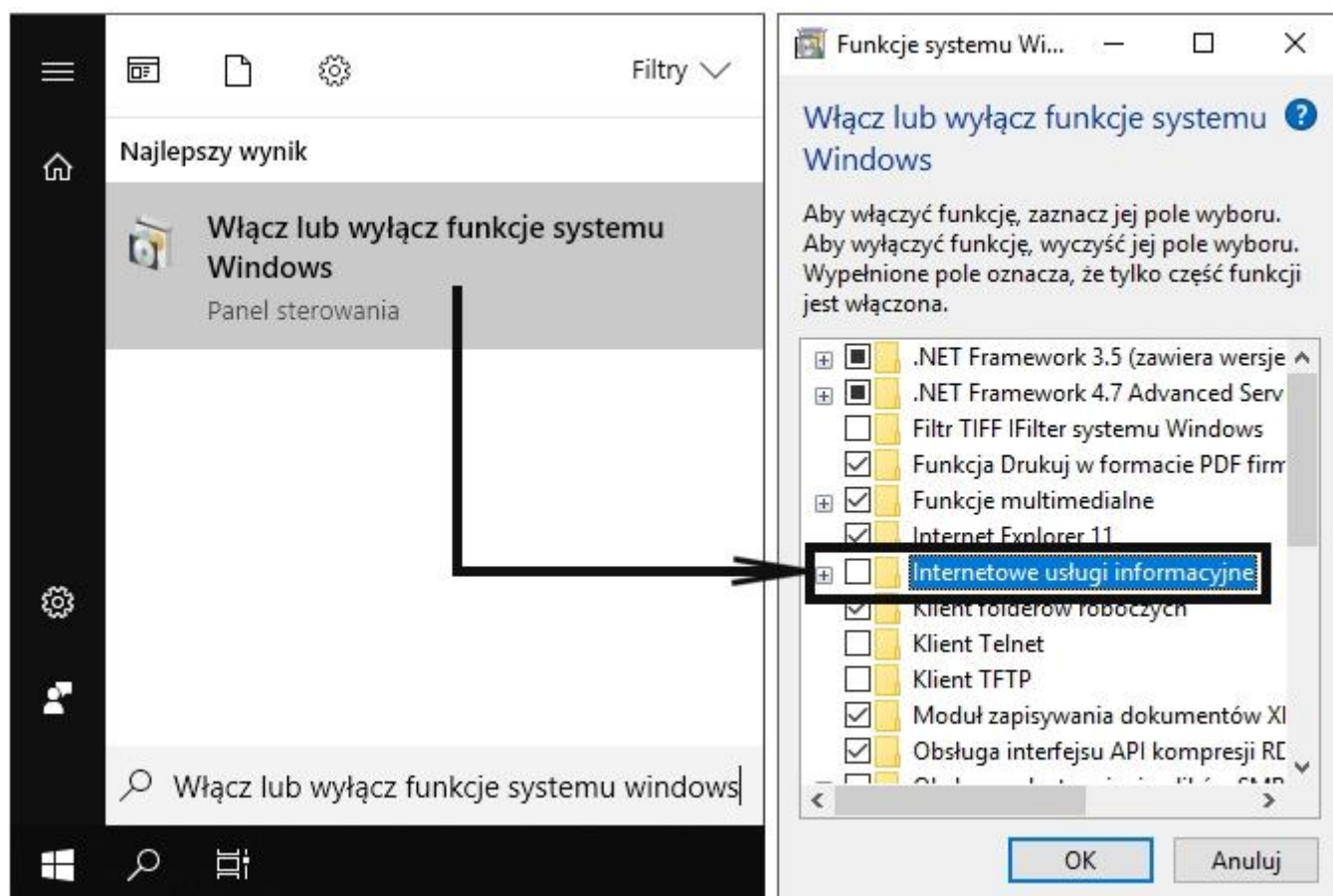
W tym epizodzie używamy języka PHP, który jest back-endowy, co znaczy: interpretowany po stronie serwera. Potrzebujemy więc dostępu do działającego serwera (hosta). Oczywiście będzie to serwer lokalny - tzw. localhost, symulowany na naszym własnym komputerze, z użyciem oprogramowania o nazwie XAMPP. Zapewne znasz już ten program z zajęć szkolnych, ale jeśli nie, to nie problem - wystarczy zajrzeć do [tego tutoriala](#), w którym wyjaśniliśmy czym ten pakiet jest oraz jak go zainstalować.

Pakiet XAMPP można pobrać tutaj: [apachefriends.org](http://apachefriends.org)

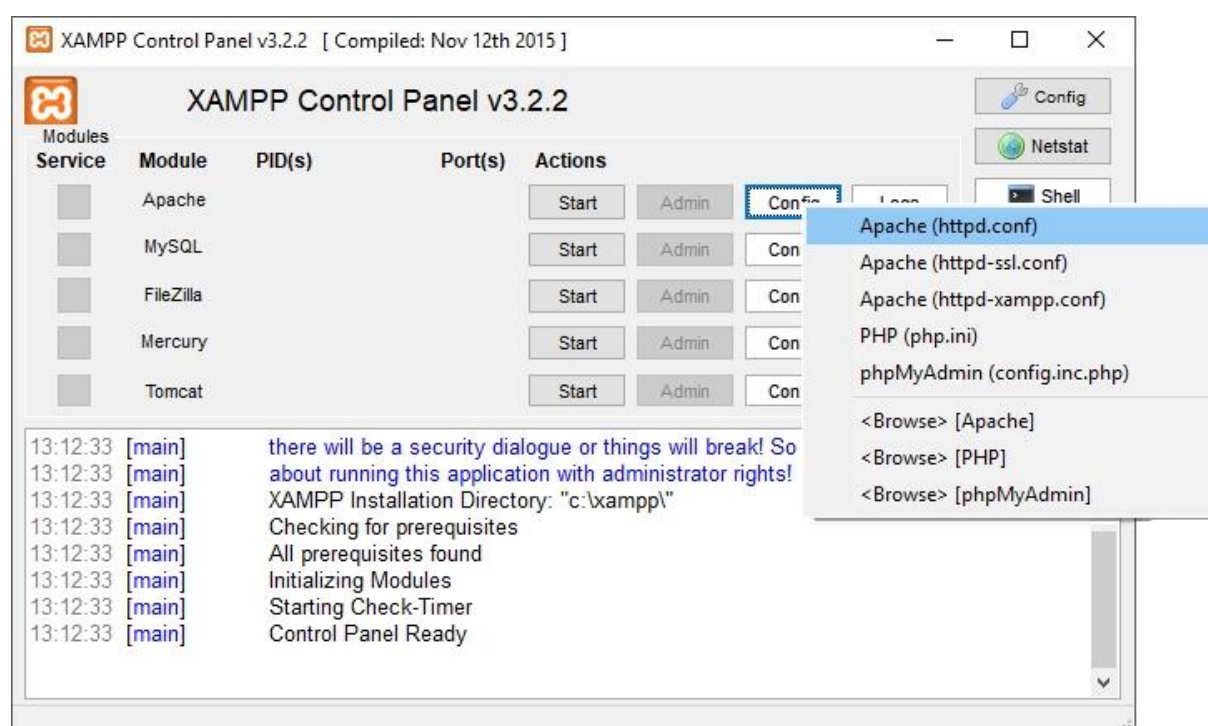
### Problemy z działaniem XAMPP?


Jeżeli pomimo przeprowadzenia instalacji w sposób pokazany na filmie XAMPP sprawia problemy, to zajrzyj do poniższej listy możliwych przyczyn. Problemy z działaniem Apache'a najczęściej wynikają z faktu, iż port 80 jest używany przez inną aplikację bądź usługę.

1. Program Skype często blokuje port 80 i bywa preinstalowany w Windows 10. Sprawdź czy na komputerze znajduje się Skype i czy odinstalowanie tej aplikacji rozwiązało problem.
2. Port 80 może być także używany w Windows 10 przez usługi IIS (Internet Information Services). Aby je wyłączyć wystarczy w menu Start wpisać: *Włącz lub wyłącz funkcje systemu Windows* po czym odznaczyć checkboxa przy *Internetowe usługi informacyjne*:



Możesz także używać Apache'a na innym porcie niż 80. Wystarczy, że w *Panelu Sterowania XAMPP Control Panel* wybierzesz kolejno: *Config*, *Apache (httpd.conf)* po czym w otwartym pliku tekstowym zmienisz numer portu na którym prowadzony jest nasłuch na inny niż 80, na przykład 85.





```
#  
# Mutex default:logs  
  
#  
# Listen: Allows you to bind Apache to specific IP addresses and/or  
# ports, instead of the default. See also the <VirtualHost>  
# directive.  
#  
# Change this to Listen on specific IP addresses as shown below to  
# prevent Apache from glomming onto all bound IP addresses.  
#  
#Listen 12.34.56.78:80  
Listen 85
```

Jednak uwaga! Po zmianie posługuj się już nie adresem *localhost*, lecz *localhost:85*

## Rozszerzenia PHP do komunikacji z bazą danych: mysql, mysqli, PDO

W celu połączenia się z bazą danych z użyciem PHP możemy teoretycznie skorzystać z trzech możliwych tzw. rozszerzeń (czyli bibliotek) komunikacyjnych. Pierwszą z nich jest *mysql* (biblioteka jednak wycofana od PHP w wersji 7.0.0), *mysqli* (gdzie *i* oznacza *improved*, czyli wersję poprawioną, aktualną) albo możemy użyć rozszerzenia o nazwie *PDO* - *PHP Data Objects* (w pełni obiektowego).

Poznamy teraz szczegóły dotyczące wszystkich rozszerzeń, jednak zanim to zrobimy zapoznajmy się z definicją akronimu *API*.

**API** (*ang. Application Programming Interface*) - interfejs klas, metod, funkcji, zmiennych, parametrów, których aplikacja PHP używa w celu zrealizowania zaplanowanych przez programistę zadań (w naszym kontekście chodzi o komunikację z bazą danych).

Interfejs *API* może być albo *proceduralny* albo *obiektowy*. Jeżeli jest *proceduralny*, to oznacza to, że operacje bazodanowe realizowane są przez odpowiednio przygotowane i wywołane funkcje. Każda funkcja wykonuje unikalne, dające się jednoznacznie wyróżnić działanie związane z bazą danych. Parametry przekazujemy do funkcji jako argumenty w nawiasie, w momencie ich wywołania.

Jeśli zaś interfejs API jest obiektowy, to oznacza to, iż w aplikacji są tworzone obiekty (na podstawie "przepisu" klas), a operacje bazodanowe realizowane są poprzez metody (czyli funkcje wewnątrz klas) wywoływane na rzecz tychże stworzonych obiektów. Oprócz metod, obiekty mogą posiadać przypisane atrybuty (parametry, właściwości). Więcej informacji na temat podejścia obiektowego (klasy, obiekty, metody, atrybuty, konstruktory, destruktory itd.) znajdziesz w [tym tutorialu](#).

**Rozszerzenie *mysql*** – dodatek wprowadzony w wersji 2.0 specyfikacji języka PHP, został zdeprecjonowany od PHP 5.5.0 oraz całkowicie usunięty od PHP 7.0.0. Nie zaleca się stosowania *mysql* we współczesnych projektach - zamiast tej biblioteki powinniśmy użyć *mysqli* lub PDO. Biblioteka posiada interfejs jedynie proceduralny, brak interfejsu obiektowego.

**Rozszerzenie *mysqli* (i = ang. improved)** - dodatek wprowadzony w wersji 5.0 specyfikacji języka PHP, usprawnił i zaktualizował komunikację do tej pory realizowaną biblioteką *mysql*. To rozszerzenie oferuje zarówno API proceduralne jak i obiektowe. W pełni aktualne, współczesne, zalecane do użycia w nowych projektach. Aczkolwiek w odróżnieniu od dodatku PDO (który potrafi obsłużyć różne silniki bazodanowe), *mysqli* współpracuje tylko bazami MySQL.

**Rozszerzenie PDO (ang. PHP Data Objects)** - dodatek wprowadzony również w wersji 5.0 specyfikacji języka PHP, zrealizowany w pełni obiektowo. W pełni aktualne, współczesne, zalecane do użycia w nowych projektach. Jako jedyne rozszerzenie potrafi oprócz MySQL obsłużyć różne silniki bazodanowe, w tym m.in.: PostgreSQL, Oracle, MS SQL, SQLite, IBM DB2, Firebird i wiele innych.

Przyjrzyjmy się teraz porównaniu rozszerzeń w postaci tabelarycznej:

Rozszerzenie PHP	<i>mysql</i>	<i>mysqli</i>	PDO
Wprowadzono w wersji PHP	2.0	5.0	5.0
Wsparcie dla rozszerzenia	zdeprecjonowane od PHP 5.5.0, usunięte od PHP 7.0.0	aktywnie wspierane	aktywnie wspierane
Zalecane dla nowych projektów	nie	tak	tak
Interfejs proceduralny	tak	tak	nie
Interfejs obiektowy	nie	tak	tak

Użycie różnych silników (driverów) SQL	nie	nie	tak
Wsparcie funkcji MySQL 5.1+	nie	tak	większość
Prepared Statements po stronie serwera	nie	tak	tak
Prepared Statements po stronie klienta	nie	nie	tak

W technikum nauczymy się korzystać z bazy danych z użyciem biblioteki mysqli (zarówno w metodzie proceduralnej, jak i obiektowej). A co do biblioteki PDO, która jest już tylko obiektowa, to zainteresowanych odsyłam do [odcinka piątego serii dedykowanej PHP](#), gdzie wyjaśniamy i pokazujemy w akcji szczegółowe różnice pomiędzy użyciem mysqli i PDO.

## Dane dostępne do bazy danych MySQL

Pierwszym krokiem realizowania skryptu współpracującego z bazą danych MySQL jest zawsze podanie informacji dostępowych, potrzebnych do połączenia się z bazą. Zazwyczaj są to następujące dane:

- host (komputer przechowujący bazę),
- login użytkownika MySQL (który posiada uprawnienia do wykonywania wybranych zapytań SQL),
- hasło dostępne tegoż użytkownika,
- nazwa bazy danych, z której skorzystamy.

Te informacje umieszczamy najczęściej w jednym, zewnętrznym skrypcie (tutaj *dbconnect.php*), co pozwala nam w razie konieczności ich zmiany (z powodu np. przeprowadzki serwera) dokonać poprawki tylko w tym jednym miejscu, zamiast w wielu skryptach PHP łączących się z bazą danych. Wiele frameworków przestrzega tej zasady, umieszczając kluczowe informacje konfiguracyjne w specjalnym pliku, na przykład: *wp-config.php* w WordPress.

Przykładowa zawartość naszego *dbconnect.php* (w razie wątpliwości patrz [tutorial video](#)):

```
<?php
$host = "localhost";
$user = "root";
$pass = "";
$db = "szkola";
?>
```

## Podpięcie danych dostępowych

Oczywiście plik należy teraz dołączyć do kodu - w naszym docelowym skrypcie łączącym się z bazą danych. Możemy to zrealizować na cztery sposoby:

1. Zainkluduj dane, a w razie gdyby pliku nie udało się poprawnie otworzyć wygeneruj tylko ostrzeżenie (warning) i kontynuuj działanie skryptu:

```
include "dbconnect.php";
```

2. Wymagaj danych, czyli w razie gdyby pliku nie udało się poprawnie otworzyć przerwij działanie skryptu (fatal error):

```
require "dbconnect.php";
```

3. Inkluduj dane, ale jednorazowo (ang. *once*). A zatem gdyby kiedyś zdarzyło nam się w jednym skrypcie podpiąć drugi raz ten plik (a zdarzyć się teoretycznie może, bo przegapić taki fakt w pojęczynie połączeń pomiędzy plikami łatwo), to i tak skrypt skorzysta tylko z pierwszego podpięcia, przez co unikniemy ponownej redefinicji wartości:

```
include_once "dbconnect.php";
```

4. Oczywiście istnieje także wymaganie pliku w wersji bez powtarzania:

```
require_once "dbconnect.php";
```

Dołączenie przygotowanych namiarów, pozwala nam użyć zmiennych zdefiniowanych w *dbconnect.php* do połączenia się z bazą danych:

```
require_once "dbconnect.php";  
$conn = mysqli_connect($host, $user, $pass, $db);
```

Wydzielenie danych dostępowych do zewnętrznego, osobnego pliku nie jest oczywiście koniecznością. W najprostszej wersji, zamiast w ogóle tworzyć zmienne, wartości potrzebne do połączenia wpisujemy wprost w funkcję `mysqli_connect()`:



```
$conn = mysqli_connect("localhost", "root", "", "szkola");
```

Co prawda nie przestrzegamy tutaj zasady wydzielenia danych dostępowych jeden raz dla całej witryny, ale nie jest to w żaden sposób zapis błędny (jest nieoptymalny, ale jak najbardziej akceptowalny). Jeśli więc interesuje Cię wersja najprostsza połączenia (chcesz po prostu zdać egzamin), to możesz użyć od razu zapisu bez tworzenia zmiennych ani dodatkowego pliku.

## Współpraca mysqli z bazą (wariant proceduralny)

Opiszmy teraz kolejne kroki implementacji połączenia z bazą danych przy użyciu rozszerzenia mysqli w wariantcie proceduralnym (czyli paradygmacie opartym na funkcjach). Najprościej mówiąc: każdej charakterystycznej czynności odpowiada specjalnie przygotowana funkcja, której użycie sprowadza się najczęściej jedynie do podania właściwych parametrów wejściowych.

### Nawiązanie połączenia

Połączenie z bazą danych nawiązujemy z użyciem funkcji `mysqli_connect()` podając dane dostępowe wprost w funkcji:

```
$conn = mysqli_connect("localhost", "root", "", "szkola");
```

Lub umieszczamy te dane w zmiennych zdefiniowanych w zewnętrznym pliku, osadzając go przed połączeniem instrukcją `require_once` (patrz poprzedni akapit). Nazwy zmiennych znajdują się w wywołaniu i ich nazwy muszą być takie same jak w pliku `dbconnect.php`

```
require_once "dbconnect.php";  
$conn = mysqli_connect($host, $user, $pass, $db);
```

### Obsługa polskich znaków

Już po nawiązaniu połączenia, włączamy obsługę standardu utf8 dla przesyłanych danych z użyciem funkcji `mysqli_set_charset()`. Argumentami funkcji są kolejno: identyfikator połączenia

(tutaj `$conn`) oraz nazwa zestawu znaków (tutaj `"utf8"`). Zwróćmy uwagę: w MySQL zapisujemy nazwę standardu bez myślnika (inaczej niż w HTML, gdzie myślnik występuje).

```
mysqli_set_charset($conn, "utf8");
```

## Wykonanie zapytania SQL

Kolejnym krokiem jest zdefiniowanie treści zapytania SQL, które mamy zamiar wykonać na naszej bazie danych. Treść kwerendy zostaje umieszczona w pomocniczej zmiennej (tutaj noszącej nazwę `$q` nawiązującej do angielskiego słowa "query"). Jednak rzeczywiste wykonanie kwerendy odbywa się dopiero w momencie wywołania funkcji `mysqli_query()`. Argumentami funkcji są kolejno: identyfikator połączenia (tutaj `$conn`) oraz treść zapytania (tutaj przechowywana w pomocniczej zmiennej `$q`):

```
$q = "SELECT id, nazwa FROM klasa";  
$result = mysqli_query($conn, $q) or die("Problemy z odczytem danych!");
```

Ponieważ wykonywanym w przykładzie zapytaniem jest `SELECT`, to zwrócone przez bazę rekordy zostają włożone do zmiennej `$result` (zwróćmy uwagę, że dla np. kwerendy `INSERT` nie ma takiej potrzeby, bo wówczas baza nie "odpowiada" rekordami). Ponadto, jako że wykonanie zapytania może się zwyczajnie nie udać (choćby dlatego, że jest niepoprawne składniowo albo nie mamy uprawnień do wykonania danego rodzaju kwerendy), to pojawia się zapis `or die()`, co tłumaczymy prosto jako "lub umrzyj" :) Stąd gdy kwerendy nie uda się wykonać, dzięki funkcji `die()` skrypt zostanie "zabity", a na stronie pojawi się komunikat zapisany w przesłanym w nawiasie argumencie.

Uwaga! Czasami zdarzy się, iż częścią zapytania będzie ciąg znaków podany przez użytkownika w formularzu - jako przykład niech posłuży takie zapytanie:

```
$klasa = $_POST["klasa"]; // wartość pobrana z formularza  
$q = "SELECT id FROM klasa WHERE nazwa='$klasa'";
```

W takim przypadku istnieje spore ryzyko wykonania takiego zapytania, gdyż użytkownik o złych intencjach może spróbować "wstrzyknąć" nam SQL w naszą predefiniowaną kwerendę poprzez tę zmienną, której treść sam przecież definiuje w formularzu. I jakkolwiek podczas egzaminu w technikum zazwyczaj nie wymaga się obecnie zabezpieczenia skryptu przed tego

typu atakiem (nazywanym *SQL injection*), to warto być świadomym natury tego zagrożenia. Po dokładne wyjaśnienie jak poradzić sobie ze wstrzykiwaniem SQL udaj się do filmu zrealizowanego w [drugim odcinku serii PHP](#).

**Wstrzykiwanie SQL (ang. SQL injection)** – metoda ataku witryny internetowej, polegająca na przemyśleniu fragmentu zapytania SQL poprzez odpowiednio spreparowaną wartość wprowadzaną w formularzu. Atak zadziała tylko w przypadku niedostatecznej tzw. sanityzacji (oczyszczania) pobranej od usera wartości

### Pobranie ilości zwróconych rekordów

Dzięki funkcji `mysqli_num_rows()` istnieje możliwość odczytania liczby zwróconych przez zapytanie *SELECT* rekordów (co w niektórych zastosowaniach może okazać się przydatne). W analizowanym w tym artykule przykładzie, wynik kwerendy został zapisany pod nazwą `$result`, stąd taki jest argument funkcji przesłany do `mysqli_num_rows()`. W pomocniczej zmiennej `$ile` znajdzie się liczba zwróconych rekordów, natomiast bezwzględnie należy pamiętać, aby tę linię umiejscowić w kodzie pod wywołaniem `mysqli_query()`, nigdy powyżej - to logiczne.

```
$result = mysqli_query($conn, $q) or die("Problemy z odczytem danych!");  
$ile = mysqli_num_rows($result);
```

### Fetchowanie pojedynczych rekordów

Angielskie słowo *fetch* oznacza: *sprawdzić, przynieść, wydobyć*. Doskonale oddaje ono proces wyjmowania pojedynczych rekordów z całej chmury zwróconych zapytaniem danych, umieszczonych w `$result`. Pomocnicza zmienna `$row` zawiera zawsze jeden, pojedynczy rekord (tablicę jednowymiarową, wiersz) wyjęty ze wszystkich rezultatów kwerendy. Oczywiście to wyjmowanie do `$row` musi być realizowane w pętli (bo rekordów może być dużo). W poniższym przykładzie jest to `while()` wykonujący się dopóki istnieją rekordy do wyjęcia (pętla sama się przerwie, gdy wszystkie rekordy zostaną obsłużone).

```
$q = "SELECT id, nazwa FROM klasa";  
$result = mysqli_query($conn, $q) or die("Problemy z odczytem danych!");  
while($row = mysqli_fetch_row($result))  
{  
    echo $row[1]."<br>";  
    // wartość $row[1] wypisze nazwę klasy,  
    // gdyby było $row[0] to wypisalibyśmy jej id  
}
```

Wyróżniamy trzy podstawowe rodzaje fetchowania rekordów do tablicy jednowymiarowej, tutaj mającej nazwę \$row:

- `mysqli_fetch_row()` – wyjmowanie liczbowe - ponumeruj (od zera) szufladki tablicy jednowymiarowej;
- `mysqli_fetch_assoc()` – wyjmowanie asocjacyjne (czyli skojarzeniowe) - ponazywaj słownie szufladki tablicy jednowymiarowej, nadając im zawsze taką samą nazwę, jaką noszą kolumny w bazie danych;
- `mysqli_fetch_array()` – zarówno ponumeruj jak i ponazywaj szufladki tablicy jednowymiarowej (wada: redundancja danych - każda szufladka jest "podwójna", gdyż istnieje zarówno komórka zaindeksowana numerycznie jak i słownie).

Przykład fetchowania asocjacyjnego:

```
$q = "SELECT id, nazwa FROM klasa";
$result = mysqli_query($conn, $q) or die("Problemy z odczytem danych!");
while($row = mysqli_fetch_assoc($result))
{
    echo $row['id']."<br>";
    // wartość $row['nazwa'] wypisze nazwę klasy
}
```

### Zamknięcie połączenia z bazą

Brak linii zamykającej połączenie z bazą danych może spowodować utratę punktów na egzaminie – nie zapomnijmy dokonać tego w odpowiednim miejscu kodu, po wszystkich wykonanych bazodanowych czynnościach! Argumentem funkcji `mysqli_close()` jest identyfikator zamykanego połączenia:

```
mysqli_close($conn);
```

## Współpraca mysqli z bazą (wariant obiektowy)

W tym artykule przedstawimy porównanie skryptów łączących się z bazą danych w wariantach proceduralnym i obiektowym, ze szczegółowym omówieniem tej drugiej metody. Za każdym razem najpierw pokażemy wariant proceduralny, po czym nastąpi alternatywny kod obiektowy, wraz z krótkimi wyjaśnieniami. Pamiętajmy, że cały kod skryptu obiektowego znajdziemy w [paczce ZIP](#) dołączonej do tego odcinka (katalog "koniec pracy", a w nim plik

"index2.php"). Przed nauką podejścia obiektowego w mysqli, warto najpierw dobrze poznać podejście proceduralne (patrz poprzedni akapit). Ponadto, osobom chcącym zrozumieć paradygmat abstrakcyjnego myślenia opartego o klasy, obiekty i metody polecam zobaczenie [tego tutoriala](#). A teraz przejdźmy już do poszczególnych zadań skryptu współpracującego z bazą danych.

## Nawiązanie połączenia

Wariant proceduralny (z użyciem funkcji):

```
require_once "dbconnect.php";  
$conn = mysqli_connect($host, $user, $pass, $db);
```

Wariant obiektowy:

```
require_once "dbconnect.php";  
$conn = new mysqli($host, $user, $pass, $db);
```

Nasze połączenie `$conn` to nowy obiekt, tworzony z użyciem operatora *new* (ang. nowy). Obiekt będzie posiadał tzw. *metody* (czyli funkcje przewidziane w przepisie klasy, wywoływane na rzecz naszego obiektu) oraz *atrybuty* (właściwości, zmienne charakteryzujące obiekt). Natomiast zapis prezentowany powyżej nazywamy *konstruktorem* (gdyż rzeczywiście następuje tu konstrukcja, czyli stworzenie obiektu).

## Obsługa polskich znaków

Wariant proceduralny:

```
mysqli_set_charset($conn, "utf8");
```

Wariant obiektowy:

```
$conn->set_charset("utf8");
```

W wersji obiektowej ustawienie zestawu znaków utf8 realizowane jest z użyciem metody `set_charset()` wywołanej na rzecz naszego obiektu `$conn`. Koniecznie zwróćmy uwagę na operator `->`, który jest jak to mówimy dwuargumentowy, to znaczy posiada coś po lewej

swojej stronie, jak i po prawej. I w przypadku metod obiektowych mysqli, po lewej stronie operatora -> znajdzie się nasz obiekt `$conn` reprezentujący połączenie z bazą, zaś po prawej stronie znajdzie się odpowiednia *metoda* (tutaj `set_charset()`, ustawiająca zestaw znaków `utf8`). Metoda tym różni się od zwykłej funkcji, iż jej definicja jest częścią klasy, która opisuje obiekt, przez co może zawierać lepsze mechanizmy *hermetyzacji* (chronienia, ukrywania) danych oraz obsługi błędów.

## Wykonanie zapytania SQL

Wariant proceduralny:

```
$q = "SELECT id, nazwa FROM klasa";  
$result = mysqli_query($conn, $q);
```

Wariant obiektowy:

```
$q = "SELECT id, nazwa FROM klasa";  
$result = $conn->query($q);
```

W wersji obiektowej nie potrzebujemy dwóch argumentów (identyfikatora połączenia oraz treści zapytania), gdyż metoda `query()` wywołana jest przecież na rzecz naszego obiektu `$conn` (stąd wystarczy jedynie treść zapytania, umieszczona w zmiennej `$q`).

## Pobranie ilości zwróconych rekordów

Wariant proceduralny:

```
$result = mysqli_query($conn, $q);  
$ile = mysqli_num_rows($result);
```

Wariant obiektowy:

```
$result = mysqli_query($conn, $q);  
$ile = $result->num_rows;
```

W tym przypadku mamy do czynienia nie z metodą obiektu, tylko z jego atrybutem `num_rows` (zwróćmy uwagę na brak nawiasów okrągłych właściwych dla funkcji). Liczba rekordów

zostaje umieszczona w atrybucie `num_rows` (czyli zmiennej przypisanej do obiektu `$conn` w klasie).

### Fetchowanie pojedynczych rekordów

W akapicie dotyczącym wariantu proceduralnego biblioteki `mysql`, poznaliśmy trzy sposoby fetchowania rekordów do tablicy jednowymiarowej:

- `mysql_fetch_row()` – wyjmowanie liczbowe – ponumeruj (od zera) szufladki tablicy jednowymiarowej;
- `mysql_fetch_assoc()` – wyjmowanie asocjacyjne (czyli skojarzeniowe) – ponazywaj słownie szufladki tablicy jednowymiarowej, nadając im zawsze taką samą nazwę, jaką noszą kolumny w bazie danych;
- `mysql_fetch_array()` – zarówno ponumeruj jak i ponazywaj szufladki tablicy jednowymiarowej (wada: redundancja danych – każda szufladka jest "podwójna", gdyż istnieje zarówno komórka zaindeksowana numerycznie jak i słownie).

Przykład użycia `mysql_fetch_row()`:

```
$q = "SELECT id, nazwa FROM klasa";
$result = mysql_query($conn, $q);
while($row = mysql_fetch_row($result))
{
    echo $row[0]."<br>"; // $row[0] wypisze id klasy
}
```

W zapisie obiektowym możemy pozbyć się (dla każdej z tych trzech metod) przedrostka `mysql_`, a ponadto zamiast zapisania `$result` jako argumentu funkcji, umieszczamy go po lewej stronie operatora `->` (jako obiekt, na rzecz którego wywołujemy metodę fetchującą pojedynczy rekord):

```
$q = "SELECT id, nazwa FROM klasa";
$result = mysql_query($conn, $q);
while($row = $result->fetch_row())
{
    echo $row[0]."<br>"; // $row[0] wypisze id klasy
}
```

Dodatkowo, poznajmy także czwarty sposób dostępu do pojedynczych rekordów. Będzie to metoda `fetch_object()`, która umieszcza wyjęty rekord w obiekcie `$obj` posiadającym atrybuty o takich samych nazwach jak kolumny w bazie:

```
$q = "SELECT id, nazwa FROM klasa";  
$result = mysqli_query($conn, $q);  
while($obj = $result->fetch_object())  
{  
    echo $obj->id."<br>";  
    echo $obj->nazwa."<br>";  
}
```

### Zamknięcie połączenia z bazą

Brak linii zamykającej połączenie z bazą danych może spowodować utratę punktów na egzaminie - nie zapomnijmy dokonać tego w odpowiednim miejscu kodu, po wszystkich wykonanych bazodanowych czynnościach!

Wariant proceduralny:

```
mysqli_close($conn);
```

Wariant obiektowy:

```
$conn->close();
```

Jak widzimy, ponownie mamy tu do czynienia z zapisem *obiekt->metoda()*, który po uprzednim przepracowaniu poprzednich przykładów nie powinien nas już dziwić.

### Obsługa wyjątków zachodzących w aplikacji

W skrypcie umieszczonym w pliku "index2.php" dołączonym do paczki z odcinka, zastosowaliśmy dodatkowo instrukcję *try..catch* zamiast klasycznego podejścia *or die()*. To tak zwane "łapanie wyjątków" pozwala nam dużo bardziej elastycznie reagować na poszczególne problemy w aplikacji webowej, zamiast po prostu "zabijać" skrypt. Oczywiście w tym odcinku jedynie zarysowaliśmy tę ideę, po więcej informacji na temat użycia *try..catch* odsyłam do [trzeciego epizodu](#) kursu PHP.

## Zadania do samodzielnego wykonania

Poniżej znajdziesz listę kilku zadań do samodzielnego zrealizowania – ich przepracowanie pozwoli utrwalić mechanikę współpracy PHP z bazą danych:



## Zadanie 1

Skorzystaj z bazy danych "szkola" przedstawionej w odcinku. Dodaj do tabeli z ocenami klasy pierwszą kolumnę zawierającą "liczbą porządkową" (nazwa kolumny: L.p.). Rekordy mają być numerowane od wartości 1 - zobacz przykład poniżej dla klasy 2a:

L.p.	Imię	Nazwisko	Średnia ocen
1	Sebastian	Kowalski	4
2	Paweł	Michalak	4
3	Damian	Markiewicz	3.5
4	Magda	Rysik	4.8
5	Łukasz	Bury	4.5
6	Stanisława	Kowalik	4
7	Krzysztof	Konarski	4.5

Średnia klasy: 4.19

## Zadanie 2

Zmień interfejs formularza witryny zrealizowanej w odcinku w taki sposób, aby nazwę klasy można było wybrać z rozwijanej listy (znacznik `<select></select>`), zamiast wpisywać ją ręcznie w polu tekstowym. Oczywiście lista ma być uzupełniona nazwami klas wyjętymi z bazy danych - gdyby w bazie "szkola" pojawiła się nowa klasa, to ma ona pojawić się także na liście wyboru klas w naszej witrynie.

Wybierz klasę:

1a ▼

1a

1b

2a

2b

## Zadanie 3

Ponad tabelą z uzyskanymi przez daną klasę ocenami powinna się pojawić także informacja (wyjęta z bazy danych) kto jest jej wychowawcą w szkole. Uwaga: ta informacja ma się pojawić tylko jeden raz, wypisana ponad tabelą, w akapicie `<p></p>`. Przykład wykonania skryptu dla klasy 2a:

Wychowawca: Bożena Michalska

L.p.	Imię	Nazwisko	Średnia ocen
1	Sebastian	Kowalski	4
2	Paweł	Michalak	4
3	Damian	Markiewicz	3.5
4	Magda	Rysik	4.8
5	Łukasz	Bury	4.5
6	Stanisława	Kowalik	4
7	Krzysztof	Konarski	4.5

Średnia klasy: 4.19

#### Zadanie 4

Stwórz nowy skrypt o nazwie *insert.php*, którego zadaniem będzie dodawanie nowego ucznia do tabeli "uczen". Interfejs formularza powinien składać się z inputów do wprowadzenia imienia, nazwiska i średniej ocen oraz z listy wyboru, która określi do której klasy przypisano ucznia. Oczywiście także tym razem lista ma być uzupełniona nazwami klas wyjętymi z bazy danych - gdyby w bazie "szkola" pojawiła się nowa klasa, to ma ona pojawić się także na tej liście wyboru.

Skrypt dodawania ucznia do bazy danych powinien także sprawdzać, czy poprawnie uzupełniono pola formularza - imię i nazwisko powinno mieć przynajmniej po 2 znaki, zaś średnia powinna być liczbą z zakresu 0-6. W razie podania błędnych danych formularz nie musi pamiętać wprowadzonych do niego wartości (nie trzeba używać mechanizmu sesji do ich zapamiętania).

## Test wiedzy z odcinka

Sprawdź się! :) Zapraszamy do testu wiedzy udostępnionego tutaj:

<http://pasja-informatyki.pl/programowanie-webowe/test/php-mysqli-baza-danych/>