

Bazy danych: MySQL w PHP

v 1.1

Plan

- [Wprowadzenie do baz danych](#)
- [Przygotowanie do pracy z MySQL](#)
- [Trochę teorii o MySQL](#)
- [MySQL i PHP](#)

- [Łączenie tabel](#)
- [Relacje między tabelami](#)
- [Zaawansowany SQL](#)

Wprowadzenie do baz danych

Co to są bazy danych?

Bazy danych są to aplikacje, których jedynym celem jest przetrzymywanie, analizowanie i zwracanie danych.

Aplikacja bazodanowa musi implementować możliwość:

- definicji danych,
- dodawania, usuwania i modyfikacji danych,
- zarządzania dostępami do danych.

Dlaczego stosujemy bazy danych?

Bazy danych stosujemy, gdy mamy zmienną ilość danych w naszym projekcie.

Trzymanie takich danych w przeznaczonej do tego bazie pozwoli nam na szybkie zarządzanie taką kolekcją i łatwe współdzielenie jej z innymi programami.

Bazy danych pomagają też w następujących zagadnieniach:

- trzymaniu bardzo dużych zbiorów danych (są bardzo dobrze zoptymalizowane pamięciowo),
- szybkim przeszukiwaniu i sortowaniu zbiorów danych,
- łączeniu danych w relacje.

Typy baz danych

Hierarchiczne

Mówi się o hierarchicznych bazach danych, jeżeli pomiędzy danymi zachowanymi w takim systemie następuje relacja rodzic – dziecko. Ten typ baz danych został stworzony przez IBM w 1968 roku i jest już niestosowany.

Relacyjne

Bazy danych, które skupiają się na relacjach między danymi. Dane w takich bazach są przedstawiane jako dwuwymiarowe tabele, gdzie każda kolumna to atrybut, a rząd to dane.

Obiektowe

Bazy danych stworzone w oparciu na idei programowania obiektowego. W pamięci przetrzymywane są obiekty danych klas. Przydatne przy przetrzymywaniu plików multimedialnych. Nie są bardzo popularne, gdyż są drogie w utrzymaniu.

Nierelacyjne

Najnowsze podejście do baz danych. Przechowujemy dane jako pary klucz – wartość, gdzie wartości nie mają ujednoliconej struktury. Łatwo skalowalne i szybkie przy dużych zestawach danych.

Relacyjne bazy danych

W relacyjnych bazach danych występują trzy relacje między tabelami:

- jeden do jednego,
- jeden do wielu,
- wielu do wielu.



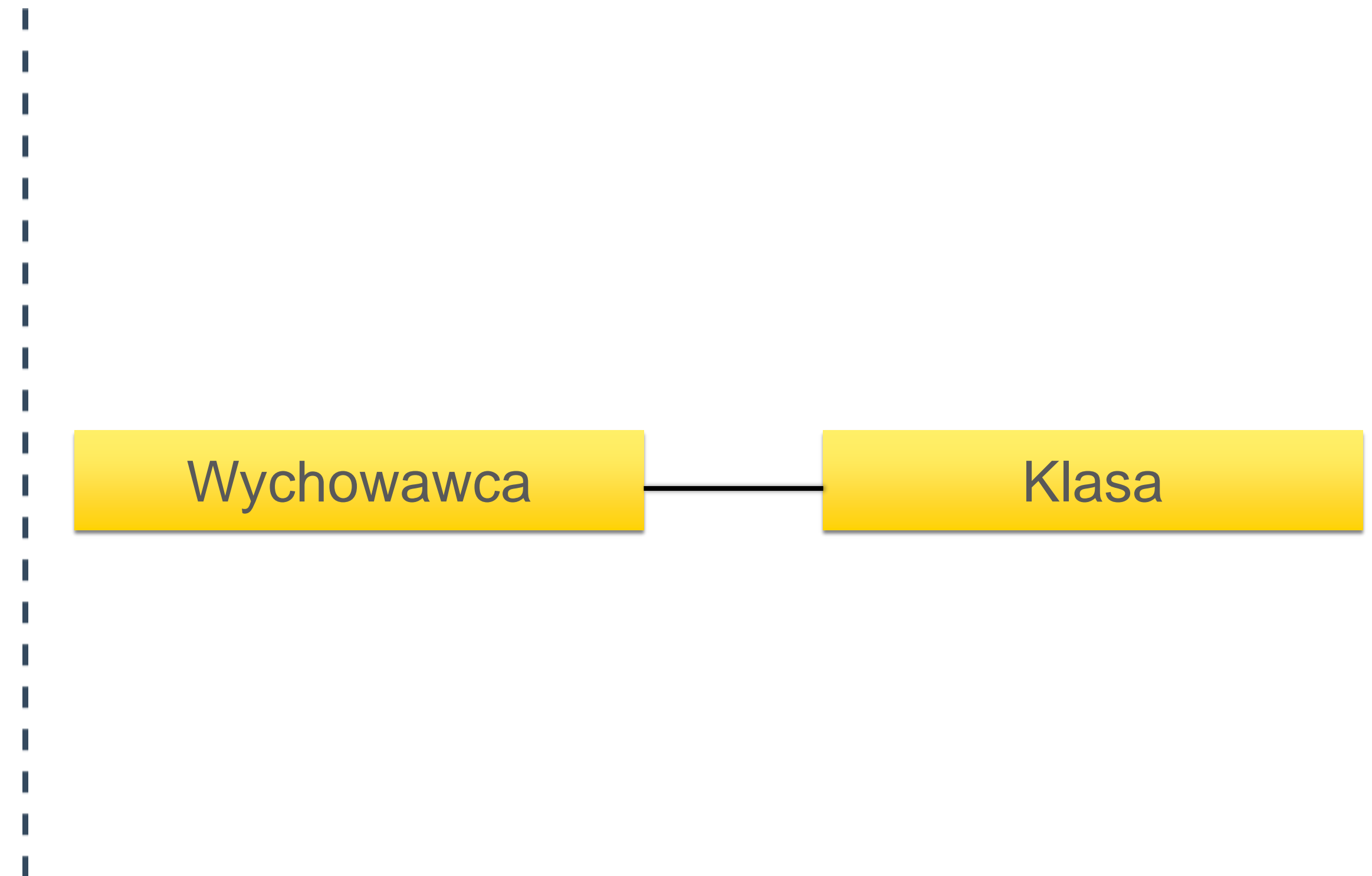
Relacja jeden do jednego

Relacja, w której jeden element z danej tabeli może być połączony tylko z jednym elementem z innej tabeli.

Przykład

Nauczyciel może być wychowawcą tylko jednej klasy.

Klasa może mieć tylko jednego wychowawcę.

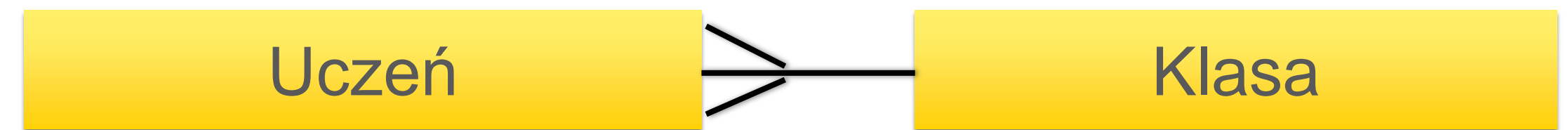


Relacja jeden do wielu

Relacja, w której jeden element z danej tabeli, może być połączony z wieloma elementami z innej tabeli.

Przykład

Uczeń może należeć tylko do jednej klasy, klasa może mieć wielu uczniów.

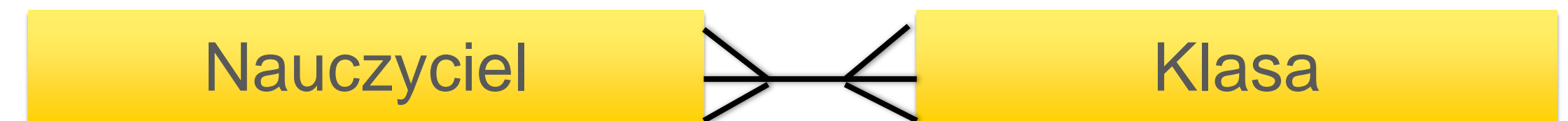


Relacja wiele do wielu

Relacja, w której wiele elementów z danej tabeli może być połączona z wieloma elementami z innej tabeli.

Przykład

Klasa może być uczona przez wielu nauczycieli, jeden nauczyciel może uczyć wiele klas.



Najpopularniejsze bazy danych

Oracle

Relacyjna baza danych oparta na języku SQL (trochę rozbudowanym). Jest raczej używana przez duże firmy, istnieją tylko jej płatne wersje.

MySQL

Relacyjna baza danych oparta na języku SQL. Darmowa do zastosowań niekomercyjnych. Popularna w małych i średnich firmach.

MongoDB

Nierelacyjna baza danych oparta na przetrzymywaniu całych dokumentów. Staje się bardzo popularna w zastosowaniach webowych.

PostgreSQL

Relacyjna baza danych, oparta na języku SQL. Wydana na licencji open source. Wygodna do stawiania prostych stron.

Dlaczego uczymy się MySQL?

Najczęściej stosowana w mniejszych firmach.
Implementuje standard języka SQL.

Zalety

- dobrze skalowalna i bardzo szybka,
- łatwa w zarządzaniu,
- bezpieczna.

Przygotowanie do pracy z MySQL

Instalacja i konfiguracja w systemie Windows

Wystarczy, że ściągniemy instalator ze strony:

<http://dev.mysql.com/downloads/windows/installer>

Instalator przeprowadzi nas przez wszystkie kroki potrzebne do zainstalowania i skonfigurowania bazy danych.

Instalacja i konfiguracja – Linux

Wystarczy nam komenda:

```
sudo apt-get install mysql-server
```

Instalator przeprowadzi nas przez wszystkie kroki potrzebne do zainstalowania i skonfigurowania bazy danych.

Instalacja i konfiguracja – OS X

Wystarczy, że ściągniemy instalator z strony:

<http://dev.mysql.com/downloads/mysql>

Instalator przeprowadzi nas przez wszystkie kroki potrzebne do zainstalowania i skonfigurowania bazy danych.

Sposoby dostępu do bazy danych

Istnieje wiele sposobów dostępu do bazy danych.
Oto najpopularniejsze z nich:

- konsola **mysql**,
- klient **mysqladmin**,
- panel **phpMyAdmin**,
- wrapery dla poszczególnych języków.

Dostęp do MySQL – konsola

Choć konsola dostępu **mysql** jest najprostszym graficznie systemem dostępu do naszej bazy danych, to daje największe możliwości.

Komenda uruchamiająca konsolę

```
mysql -h hostname -u username -p -D databasename
```

Odpowiedź konsoli

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 32

...

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _

Dostęp do MySQL – uruchamianie konsoli

Najważniejsze opcje przy uruchamianiu konsoli

- **-h hostname** – podajemy adres hosta, na którym znajduje się nasza baza danych (domyślnie localhost).
- **-u username** – podajemy nazwę użytkownika.
- **-p** – podczas logowania zostaniemy poproszeni o hasło.
- **-D database** – po zalogowaniu od razu zostanie załadowana podana baza danych (domyślnie nie zostanie załadowana żadna baza).

Dostęp do MySQL – klient mysqladmin

Klient **mysqladmin** jest programem, który ma zaimplementowane komendy do najczęściej powtarzanych zadań administracyjnych.

Wpisanie poniższej komendy zwróci nam wszystkie opcje:

mysqladmin

Dostęp do MySQL – klient mysqladmin

Najczęściej używane opcje:

- **-u username**
 - wykonuje dane polecenie jako podany użytkownik,
- **-p**
 - pyta o hasło przed wykonaniem czynności,
- **create nazwaBazy**
 - tworzy nową bazę danych o podanej nazwie,
- **drop nazwaBazy**
 - niszczy bazę o podanej nazwie,
- **password noweHaslo**
 - zmienia hasło podanego użytkownika,
- **ping**
 - sprawdza działanie podanego hosta MySQL,
- **status**
 - wyświetla statystyki naszej bazy danych.

Dostęp do MySQL – panel phpMyAdmin

- Jest to w pełni funkcjonalny, napisany w PHP, klient graficzny do zarządzania naszą bazą danych.
- Często jedyny sposób dostępu do baz danych na wykupionych serwerach.
- Program jest darmowy
- Można go do ściągnąć ze strony: <http://www.phpmyadmin.net>

Dostęp do MySQL: Panel phpMyAdmin

Tworzenie backupu bazy danych

Do tworzenia backupu bazy danych służy narzędzie **mysqldump**.

Podstawowe użycie tego narzędzia wygląda tak:

```
mysqldump -u [user_name] -p [database_name] > [file].sql
```

Powyższa komenda utworzy plik o podanej nazwie, w którym będzie pełny zapis naszej bazy danych.

Wczytywanie backupu

Żeby wczytać backup należy po prostu przesłać wszystkie znajdujące się w nim informacje do silnika **MySQL**.

Robimy to przy pomocy komendy:

```
mysql -u[user_name] -p < [file].sql
```

Baza danych o odpowiedniej nazwie musi być wcześniej stworzona i musi być pusta!

Trochę teorii o MySQL

Silniki baz danych

Bazy danych z czasem rozrosły się, wspomagając dużo różnych rozwiązań i technologii. Powstało zatem wiele silników wspierających MySQL.

Silniki możemy zrozumieć jako moduły odpowiedzialne za tworzenie i zarządzanie tabelami z danymi.

Najpopularniejsze silniki dla MySQL

- MyISAM
- IBMDB2I
- InnoDB
- MEMORY
- MERGE
- FEDERATED
- ARCHIVE
- CSV
- EXAMPLE
- BLACKHOLE

Opis silników dla MySQL

- **MyISAM** – Podstawowy silnik dla MySQL. Niezależny od systemu operacyjnego (tabele są przenaszalne). Szybki i wydajny, jednak nie wspiera transakcji. Powinien być wybierany przy częstych operacjach SELECT i INSERT.
- **InnoDB** – Silnik wydany na licencji open source. Jego główną zaletą jest pełne wsparcie dla transakcji. Powinien być wybierany, jeżeli w bazie będzie dużo operacji typu UPDATE lub jeżeli są potrzebne transakcje.
- **MEMORY** – Silnik trzymający wszystkie dane w pamięci. Silnik bardzo szybki, ale tracący wszystkie dane, jeżeli proces MySQL nie zostanie poprawnie zamknięty. Nie wspomaga wszystkich typów danych.
- **CSV** – Silnik zapamiętujący wszystkie dane jako pliki .csv. Nie wspomaga indeksów.
- **EXAMPLE** – Template pozwalające tworzyć nowe silniki.
- **BLACKHOLE** – Silnik identyczny w działaniu do MyISAM, jednak nie przetrzymuje żadnych danych. Używany do przeprowadzania testów.

MySQL i PHP

Interfejsy MySQL w PHP

Dla PHP mamy kilka interfejsów MySQL:

- **MySQLi procedural**
– proceduralna implementacja interfejsu MySQL.
- **MySQLi object**
– obiektowa implementacja interfejsu MySQL.
- **PDO (PHP Data Objects)**
– bardziej skomplikowany interfejs,
pozwalający na komunikację
z 12 typami baz danych.

Praca z bazą danych

Łączenie się z bazą

Pierwszym krokiem do używania bazy danych jest podłączenie się do niej.

W obiektowym **MySQLi** stworzenie nowego obiektu klasy **mysqli** oznacza nawiązanie połączenia:

```
$conn = new mysqli(servername, username,  
                    password, [baseName]);
```

Jeżeli podamy nazwę bazy,
od razu podłączymy się do tej bazy.

Kończenie pracy z bazą

- Musimy też pamiętać o poprawnym zamknięciu połączenia.
- Jeżeli obiekt zostanie zniszczony, zanim zdąży zamknąć połączenie, niektóre dane mogą nie zostać poprawnie zapamiętane.

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

Sprawdzanie błędów

- Po każdej operacji wykonanej na połączeniu z bazą danych powinniśmy sprawdzić, czy ta operacja zakończyła się powodzeniem.
- Atrybut **connect_error** jest ustawiony na NULL, gdy błąd nie wystąpił.
- Gdy błąd wystąpi, wyczytamy z niego komunikat błędu.

```
if ($conn->connect_error) {  
    die("Połączenie nieudane. Błąd: " . $conn->connect_error);  
}
```


Przykład połączenia

```
$servername = "localhost";  
$username = "username";  
$password = "password";  
$baseName = "sql_cwiczenia";
```

Tworzenie nowego połączenia:

```
$conn = new mysqli($servername,  
                  $username,  
                  $password,  
                  $baseName);
```

Sprawdzamy, czy połączenie się udało:

```
if ($conn->connect_error) {  
    die("Połączenie nieudane. Błąd: " .  
        $conn->connect_error);  
}  
echo("Połączenie udane.");
```

Niszczymy połączenie:

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

Tworzenie nowej bazy danych

Pierwszym krokiem jest stworzenie bazy danych.

Często jeden projekt może mieć więcej niż jedną bazę danych np.:

- do testów,
- dla developmentu,
- produkcyjną.

Bazę tworzymy poprzez zapytanie SQL:

CREATE DATABASE <nazwa_nowej_bazy>;

Musimy pamiętać, że każde zapytanie SQL musi kończyć się średnikiem, ale tylko jeżeli wywołujemy je w konsoli.

W kodzie PHP nie musimy kończyć zapytań SQL średnikiem.

Praca z połączeniem mysqli

W PHP komunikacja z MySQL polega na wysyłaniu zapytań (**queries**) przez obiekt klasy **mysqli**.

```
$result = $conn->query("Tekst zapytania SQL");
```

Metoda **query** zwraca:

- **FALSE** w przypadku w których zapytanie się nie udało,
- Obiekt klasy **mysqli_result** jeżeli zapytanie **SELECT**, **SHOW**, **DESCRIBE** albo **EXPLAIN** się udało. Obiekt ten będzie rzutowany na wartość logiczną **TRUE** jeżeli będzie taka potrzeba,
- **TRUE** w przypadku w którym inne, nie wymienione wyżej, zapytanie się udało.

Tworzenie nowej bazy danych przez PHP

Obiekt `$conn` to poprawnie stworzony obiekt klasy `mysqli`.

Zawsze pamiętaj o poprawnym zamknięciu i zniszczeniu połączenia.

```
$sql = "CREATE DATABASE sql_cwiczenia";  
$result = $conn->query($sql);  
if ($result != FALSE) {  
    echo("Baza stworzona poprawnie");  
} else {  
    echo("Błąd podczas tworzenia nowej bazy: "  
        . $conn->error);  
}
```

Tworzenie nowej tabeli

Nową tabelę tworzymy przez następujące zapytanie SQL:

CREATE TABLE **table_name**

(
nazwa_kolumny_1 typ_danych(size) [atrybuty],
nazwa_kolumny_2 typ_danych(size) [atrybuty],
nazwa_kolumny_3 typ_danych(size) [atrybuty],
....
);

Przyjęte jest,
że pierwszy element takiej tabeli to ID.

Typy danych w MySQL

Typy danych w **MySQL**
dzielimy na trzy główne grupy:

- zmienne liczbowe,
- daty i czas,
- napisy.

Typy danych w MySQL – liczby

- **INT** – podstawowa zmienna liczbowa.
Jeżeli używana jest ze znakiem,
to pomieści liczby w następującym przedziale:
od **-2 147 483 648** do **2 147 483 647**.
Jeżeli nie podamy znaku,
to przedział ten wynosi od **0** do **4 294 967 295**.
- **TINYINT** – najmniejsza zmienna liczbowa.
Mieści przedział liczb od **-128** do **127**
(lub **0** do **255**).

- **SMALLINT** – mała zmienna liczbowa.
Mieści przedział liczb:
od **-32 768** do **32 767**
(lub **0** do **65 535**).
- **MEDIUMINT** – średnia zmienna liczbowa.
Mieści przedział liczb:
od **-8 388 608** do **8 388 607**
(lub od **0** do **16 777 215**).

Typy danych w MySQL – liczby

- **BIGINT** – duża zmienna liczbowa.
Mieści od **–9 223 372 036 854 775 808**
do **9 223 372 036 854 775 807**
(lub **0** do **18 446 744 073 709 551 615**).
- **FLOAT(M,D)** – zmienna reprezentująca liczbę zmiennoprzecinkową.
M – liczba wyświetlanych cyfr,
D – liczba cyfr po przecinku,
może trzymać do 24 miejsc po przecinku.
- **DOUBLE(M,D)** – liczba zmiennoprzecinkowa o większej dokładności. Może trzymać do 53 miejsc po przecinku. **REAL** jest synonimem zmiennej typu **DOUBLE**.
- **DECIMAL(M,D)** – liczba zmiennoprzecinkowa, do której nie używamy kompresji.
NUMERIC jest synonimem **DECIMAL**.

Typy danych w MySQL – data i czas

- **DATE** – data w formacie RRRR-MM-DD.
Może trzymać daty
od **1000-01-01** do **9999-12-31**.
- **DATETIME** – data w formacie RRRR-MM-DD
GG:MM:SS.
Może trzymać daty od **1000-01-01 00:00:00**
do **9999-12-31 23:59:59**.

- **TIMESTAMP** – data trzymiana w formacie
RRRRMMDDGGMMSS. Może trzymać daty
między 1 stycznia 1970 roku a 2038 rokiem.
- **TIME** – trzyma czas w formacie GG:MM:SS.
- **YEAR(M)** – trzyma rok w formacie dwu-
lub czterocyfrowym.

Typy danych w MySQL – napisy

- **CHAR(M)** – napis mający z góry określoną liczbę znaków, parametr **M** przyjmuje wartość między **0** a **255**. Wypełniany spacjami, jeżeli napis będzie krótszy.
- **VARCHAR(M)** – napis o zmiennej liczbie znaków, nie większej jednak niż podany parametr **M** (o wartości od 0 do 255).
- **BLOB (Binary Large Object) lub TEXT** – pole zawierające maksymalnie 65535 znaków. BLOB od zmiennej **TEXT** różni się tym, że porównanie zmiennej typu **BLOB** nie jest wrażliwe na wielkość znaków.
- **TINYBLOB lub TINYTEXT** – zmienna **BLOB** lub **TEXT**, ale o maksymalnej długości 255 znaków.
- **MEDIUMBLOB lub MEDIUMTEXT** – zmienna **BLOB** lub **TEXT**, ale o maksymalnej długości **16 777 215** znaków.
- **LONGBLOB lub LONGTEXT** – zmienna **BLOB** lub **TEXT**, ale o maksymalnej długości **4 294 967 295** znaków.

Atrybuty danych w MySQL

Najczęściej używane atrybuty

- Atrybuty dodają dodatkowych założeń do typów danych.
- Nałożenie odpowiednich atrybutów może całkowicie zmienić zastosowanie danej kolumny w tabeli naszych danych.
- **PRIMARY KEY** – czyli klucz główny. Atrybut stosowany do wskazania, że ta kolumna będzie jednoznacznie identyfikowała każdy wpis. Zazwyczaj do stworzenia klucza głównego, używamy zmiennej typu **INT** z włączoną opcją **AUTO_INCREMENT**.

Klucz główny nie powinien być zmieniany po utworzeniu. Może nam to zniszczyć układ całej bazy.

Najczęściej używane atrybuty

- **UNSIGNED** – stosowany przy zmiennych liczbowych.
- **ZEROFILL** – stosowany przy zmiennych liczbowych. Powoduje dopełnienie liczby zerami poprzedzającymi.
- **CHARACTER SET** – stosowany przy napisach. Powoduje używanie odpowiedniego kodowania do napisów.
- **BINARY** – Używany przy zmiennych typu CHAR lub VARCHAR. Powoduje, że sortowanie będzie **case-sensitive**.

Najczęściej używane atrybuty

- **DEFAULT** – ustawia domyślną wartość wpisywaną w tę kolumnę.
- **NULL / NOT NULL** – pozwala (lub nie pozwala) na wprowadzanie pustych danych w tę kolumnę.
- **AUTO_INCREMENT** – stosowany przy zmiennych liczbowych. Powoduje, że wartość w tej kolumnie zwiększa się o jeden przy każdym wpisie.

Tworzenie nowej tabeli

CREATE TABLE **users**

(
user_id int **AUTO_INCREMENT**,
user_name **varchar**(**255**),
user_email **varchar**(**255**) **UNIQUE**,
PRIMARY KEY(user_id)
);

user_id będzie kluczem
głównym w naszej tabeli.
Ten numer będzie
identyfikował każdy wpis.

User_email ma flagę **UNIQUE**
– chcemy żeby w naszym
systemie mógł być tylko jeden
użytkownik z danym mailem

Tworzenie nowej tabeli przez PHP

```
$sql = "CREATE TABLE users (user_id int AUTO_INCREMENT,  
                                user_name varchar(255),  
                                user_email varchar(255) UNIQUE,  
                                PRIMARY KEY(user_id));"
```

```
$result = $conn->query($sql);  
if ($result === TRUE) {  
    echo("Tabela Users została stworzona poprawnie");  
} else {  
    echo("Błąd podczas tworzenia tabeli: " . $conn->error);  
}
```

Czas na zadania

- Przeróbcie ćwiczenia z części A
Pierwsze dwa ćwiczenia zróbcie z wykładowcą.

Dodawanie elementów do tabeli

Dane do tabeli dodajemy za pomocą zapytania
INSERT INTO:

Jeżeli po nazwie tabeli nie podamy nazw kolumn,
dane będą wkładane w kolejne kolumny tabeli
(zgodnie z jej definicją).

```
INSERT INTO table_name(columnName1, columnName2, columnName3, ...)  
VALUES (value1, value2, value3, ...);
```

Dodawanie elementów do tabeli

```
INSERT INTO users VALUES (0, "Jacek", "jacek@gmail.com");  
Query OK, 1 row affected (0.06 sec)
```

Błąd spowodowany tym że
ilość kolumn nie jest równa
z ilością przekazanych
danych.

```
INSERT INTO users VALUES("Wojtek", "wojtek@gmail.com");  
ERROR 1136 (21S01): Column count doesn't match value count at row 1
```



```
INSERT INTO users(user_name, user_email) VALUES("Wojtek", "wojtek@gmail.com");  
Query OK, 1 row affected (0.06 sec)
```

Dodawanie elementów do tabeli przez PHP

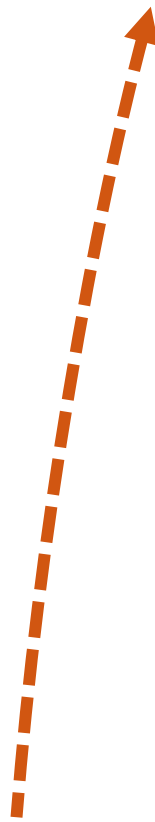
```
$sql = "INSERT INTO users(user_name, user_email) VALUES('Wojtek', 'wojtek@gmail.com')";
```

```
if ($conn->query($sql) === TRUE) {  
    echo("Nowy user został dodany");  
} else {  
    echo("Błąd: " . $sql . "<br>" . $conn->error);  
}
```

Ostatni wstawiony/edytowany element (w PHP)

- Po każdej operacji wstawienia lub edycji możemy otrzymać ID (wartość klucza głównego) elementu, na którym pracowaliśmy.
- Jest on zapisany w atrybucie **insert_id** naszego połączenia.

```
$last_id = $conn->insert_id;  
echo "Nowy rekord o id " . $last_id . " został  
wstawiony.";
```



Jest to jedyny sposób
żeby móc dowiedzieć się
o wartości klucza
głównego dla właśnie
wpisanego elementu

Wczytywanie elementów z tabeli

Dane z tabeli wczytujemy za pomocą zapytania
SELECT:

```
SELECT column_name, column_name  
FROM table_name;
```

Żeby wybrać wszystkie kolumny możemy użyć *

```
SELECT * FROM table_name;
```

Wczytywanie elementów z tabeli

Elementy zwracane są w następującej postaci:

```
SELECT * FROM users;
```

user_id	user_name
1	Wojtek
2	Wojtek2
3	Paweł
4	Janusz

4 rows in set (0.00 sec)

Obiekt typu mysqli_result

Metoda **query** w przypadku udanych zapytań **SELECT**, **SHOW**, **DESCRIBE** albo **EXPLAIN** zwraca nam obiekt klasy **mysqli_result**.

Obiekt ten trzyma w sobie wiele interesujących nas danych i posiada wiele przydatnych metod.

Najważniejsze atrybuty:

- **result->num_rows** - zwraca nam ilość wierszy zwrócone przez zapytanie,
- **result->field_count** – zwraca nam ilość kolumn zwróconych przez zapytanie.

Najważniejsze metody:

- **result->fetch_assoc()** - zwraca nam jeden rząd z odpowiedzi jako tablice asocjacyjną lub NULL (jeżeli nie ma więcej rzędów). Tablica jako klucze ma nazwy kolumn.
- **result->fetch_row()** - zwraca nam jeden rząd z odpowiedzi jako tablice lub NULL (jeżeli nie ma więcej rzędów).

Obiekt typu mysqli_result

Obiekt klasy **mysqli_result** działa w ten sposób że po wczytaniu jednego rzędu przesuwając swój wewnętrzny wskaźnik na następny rząd. Dzięki temu każdorazowe wywołanie powyższych metod zwraca kolejne rzędy.

Obiekt klasy **mysqli_result** implementuje też interfejs Traversable. Oznacza to że można taki obiekt używać wraz z pętlą foreach. Więcej o interfejsach dowiedziecie się na zajęciach z zaawansowanego PHP.

Wczytywanie elementów z tabeli (PHP)

- Po wywołaniu klauzury SELECT w PHP liczba wczytanych wierszy z bazy danych zostanie zapisana w atrybucie **num_rows** obiektu zwróconego przez metodę query.
- Wiersze te możemy wczytać (pojedynczo!) za pomocą metody **fetch_assoc()**, która zwróci nam tabelę asocjacyjną.
- Jeżeli nie ma więcej wczytanych rzędów, **fetch_assoc()** zwróci nam NULL.

```
$sql = "SELECT user_id, user_name FROM users";
$result = $conn->query($sql);

if ($result->num_rows > 0) {

    // Wypisz na ekran dane
    while($row = $result->fetch_assoc()) {
        echo("id " . $row["user_id"]) .
            "imie" . $row["user_name"];
    }
}
else {
    echo("Brak wyników");
}
```

Pętla jest powtarzana tak długo aż metoda nie zwróci NULL

Wczytywanie elementów z tabeli (PHP)

- To samo możemy uzyskać za pomocą pętli foreach.

```
$sql = "SELECT user_id, user_name FROM users";  
$result = $conn->query($sql);  
  
foreach($row as $result) {  
  
    // Wypisz na ekran dane  
    echo("id " . $row["user_id"]) .  
        "imie" . $row["user_name"];  
}
```

Klauzula WHERE

Możemy zawężyć wyniki wyszukiwania przez dodanie klauzuli WHERE do naszego zapytania SELECT.

```
SELECT column_name, column_name  
FROM table_name  
WHERE column_name = <szukana wartość>;
```

Np.:

```
SELECT * FROM users  
WHERE user_name = "Wojtek";
```

Klauzula WHERE

Elementy zwracane są w postaci tabelki asocjacyjnej gdzie kluczem jest nazwa kolumny:

```
SELECT * FROM users WHERE user_name LIKE "W%";
```

user_id	user_name
1	Wojtek
2	Wojtek2

2 rows in set (0.00 sec)

Operacje porównania w MySQL

=	Równe
< >	Nierówne (w nowszych wersjach można użyć !=)
> (>=)	Większe niż (większe równe niż)
< (<=)	Mniejsze niż (mniejsze równe niż)
BETWEEN a AND b	Pomiędzy podanym zakresem (wliczając podany zakres)
LIKE	Szuka podanego wzorca (tylko napisy)
IN(a,b,c)	Znajduje się w zmiennych podanych w nawiasach
NOT	Może poprzedzać inne operacje
OR / AND	Operatory logiczne łączące poszczególne wyrażenia

Klauzula AS

Jeżeli z jakiegoś powodu w wynikach wyszukiwania mamy 2 kolumny o takiej samej nazwie to w PHP będziemy mieli dostęp tylko do jednej z nich (jeżeli korzystamy z funkcji zwracających tablice asocjacyjną).

Możemy zawsze nadać kolumnie nową nazwę (alias) na czas tego wyszukania.

Robimy to za pomocą klauzuli AS:

```
SELECT column_name AS column_alias  
FROM table_name;
```

Np.:

```
SELECT user_id AS id FROM Users;
```

Klauzula ORDER BY

Możemy sortować znalezione wyniki względem jednej kolumny (lub więcej). Służy do tego klauzula ORDER BY

```
SELECT column_name, column_name  
FROM table_name  
ORDER BY column_name ASC|DESC,  
column_name ASC|DESC;
```

Przykład

```
SELECT * FROM users  
ORDER BY name;
```

Wybieramy jedną z możliwości:
ASC – rosnąco (ascending),
DESC – malejąco (descending).

```
SELECT * FROM users ORDER BY name;
```

user_id	user_name
1	Antek
3	Beata
2	Wojtek2

3 rows in set (0.00 sec)

Czas na zadania

- Przeróbcie ćwiczenia z części B
Pierwsze dwa ćwiczenia zróbcie z wykładowcą.

Zmiana wartości danych

Dane z tabeli wczytujemy za pomocą zapytania
UPDATE:

UPDATE table_name

SET column1=value1, column2=value2, ...

WHERE some_column=some_value;

UPDATE stosujemy zawsze z klauzulą WHERE
(inaczej wszystkie dane z tabelki zostaną
zmienione).

Zmiana wartości danych

```
UPDATE users  
SET user_name="Grzesiek"  
WHERE user_id=2;
```

```
SELECT * FROM users;
```

user_id	user_name
1	Wojtek
2	Grzesiek
...	

4 rows in set (0.00 sec)

PHP

```
$sql = "UPDATE users SET user_name='Grzesiek'  
WHERE user_id=2";  
  
if ($conn->query($sql) === TRUE) {  
    echo "Wpis został poprawiany."  
} else {  
    echo "Bład: " . $conn->error;  
}
```

Usuwanie danych z tabeli

Dane z tabeli usuwamy za pomocą zapytania DELETE:

DELETE FROM table_name

WHERE some_column=some_value;

DELETE stosujemy z klauzulą WHERE (inaczej wszystkie dane z tabeli zostaną usunięte).

Usuwanie danych z tabeli

```
DELETE FROM users
WHERE user_name="Grzesiek";
```

```
SELECT * FROM users;
```

user_id	user_name
1	Wojtek
3	Paweł
4	Janusz

3 rows in set (0.00 sec)

```
$sql = "DELETE FROM users
WHERE user_name="Grzesiek";

if ($conn->query($sql) === TRUE) {
    echo "Wpis został usunięty.";
} else {
    echo "Bład: " . $conn->error;
}
```

Czas na zadania

- Przeróbcie ćwiczenia z części C
Pierwsze dwa ćwiczenia zróbcie z wykładowcą.

Modyfikacja tabeli

Wygląd tabeli (liczba kolumn, dane w nich przechowywane) możemy zmienić za pomocą ALTER TABLE:

Dodanie nowej kolumny

ALTER TABLE table_name **ADD** column_name datatype;

Usunięcie kolumny

ALTER TABLE table_name **DROP COLUMN** column_name;

Zmiana danych trzymanych w kolumnie

ALTER TABLE table_name **MODIFY COLUMN** column_name new_datatype;

Modyfikacja tabeli

```
mysql> DESCRIBE users;
```

Field	Type	Null	Key	Default	Extra
user_id	int(11)	NO	PRI	NULL	auto_increment
user_name	varchar(255)	YES		NULL	

```
2 rows in set (0.00 sec)
```

Modyfikacja tabeli

```
mysql> ALTER TABLE users MODIFY COLUMN user_name varchar(30);  
mysql> ALTER TABLE users ADD email varchar(30);
```

```
mysql> DESCRIBE users;
```

Field	Type	Null	Key	Default	Extra
user_id	int(11)	NO	PRI	NULL	auto_increment
user_name	varchar(30)	YES		NULL	
email	varchar(30)	YES		NULL	

```
3 rows in set (0.00 sec)
```


Usunięcie tabeli

Tabelę możemy usunąć z naszej bazy danych przez użycie zapytania DROP TABLE:

```
DROP TABLE table_name;
```

Całą bazę danych możemy zniszczyć, używając zapytania DROP DATABASE;

```
DROP DATABASE db_name;
```

Czas na zadania

- Przeróbcie ćwiczenia z części D
Pierwsze dwa ćwiczenia zróbcie z wykładowcą.