Instrukcja 4

BAZY DANYCH A PHP

Podczas pisania serwisów WWW, szczególnie tych bardziej rozbudowanych, prędzej czy później zachodzi potrzeba zapisywania pewnych informacji po stronie serwera. W przypadku małej ilości danych do zapisu, na przykład na stronie firmowej na której co jakiś czas aktualizowane są wiadomości, istnieje możliwość zapisu danych w plikach tekstowych. Niewątpliwą zaletą plików jest szansa znalezienia tańszego hostingu – przeważnie hosting z bazami danych jest droższy (koszty licencji, koszty utrzymania). Z drugiej strony większość dostawców usług hostingowych oferują konta z przynajmniej jedną bazą danych. Tak więc stosowanie plików opłaca się jedynie w przypadku własnego serwera WWW – przykładowo w siedzibie rzeczonej firmy, gdzie użytkowanie bazy danych pociągałoby za sobą koszta licencyjne i administracyjne.
Pomijając przykład z poprzedniego akapitu większość programistów wręcz nie widzi innej możliwości gromadzenia danych jak przy użyciu baz danych. Rozwiązanie tego typu ma niemal same zalety. Najważniejszą z nich jest oczywiście brak przymusu czytania całego pliku z danymi tylko po to, by wyciągnąć z niego 1 linię tekstu. Kolejna to uporządkowanie danych – pliki tekstowe przy pewnym rozmiarze są ciężkie do odczytu, a ich zawartość, nawet przy stosowaniu pewnego rodzaju kodowania (jak np. tabulacje), staje się chaotyczna. Pozostaje też fragmentacja danych – serwery baz danych przeważnie zapobiegają temu zjawisku poprzez odpowiednie mechanizmy czyszczące. Kolejnymi profitami są:
- sprawny i wydajny mechanizm przeszukiwania
- indeksowanie zawartości jednoznacznych danych (tzw. klucze indeksowe)
- możliwość łączenia poszczególnych tabel poprzez tzw. klucze obce (podstawowy mechanizm weryfikacji danych)
- proste polecenia dodawana/edytowania/usuwania danych
- kompresowanie danych (zarówno tekstowych jak i binarnych – w tym obrazów)
- efektywny system operacji na łańcuchach znakowych (porównywanie, przeszukiwanie, łączenie oraz rozdzielanie)
- wiele innych...

Bazy danych zorganizowane są w sposób tabelaryczny. Jedna baza danych może zawierać wiele tabel z danymi. Z kolei tabela ma następujący układ – każda nowa seria danych wprowadzana do tabeli organizowana jest w jej pojedynczy wiersz; wiersz ten rozkładany jest z kolei na odpowiednie kolumny. Rozważmy następujący przykład – chcemy do bazy danych zapisywać dane z naszej klasy Osoba. Każdy obiekt tej klasy posiada następujące pola – id, imię, nazwisko oraz funkcja. Pola te będą stanowić kolumny w naszej tabeli. Przykładowy wygląd bazy danych:

id	imie	nazwisko	funkcja
1	<Imię>	<Nazwisko>	1
2	<Imię>	<Nazwisko>	2
3	<Imię>	<Nazwisko>	1

Poszczególne wartości (np. imię Bartek) zapisywane jest w komórce (skrzyżowanie wybranego wiersza z kolumną); w bazach danych komórkę nazywa się także krotką.
Indeksem w bazie danych nazywa się kolumnę, pod którą zapisuje się wartość jednoznacznie identyfikującą cały wiersz danych. To projektant bazy danych określa która kolumna ma stać się kluczem głównym danej tabeli; baza automatycznie będzie zabraniała dodania pod tą kolumną wartości, która już wcześniej została do niej wpisana (w którymkolwiek wierszu). Dlatego najlepszym rozwiązaniem jest dodawanie w tego typu kolumnie np. liczb pozycyjnych (LP), które najlepiej za każdym dodaniem zwiększać o jeden (baza danych posiada przeważnie mechanizm zwiększania wartości tego typu liczby – wystarczy go aktywować). Innym rozwiązaniem jest

stosować dane, które na pewno nie mogą się powtórzyć – chociażby numer PESEL lub NIP. Kluczem obcym staje się wskazana przez projektanta kolumna, której wartość jest ściśle związana z kolumną innej tabeli. Ustawienie takiej opcji powoduje, że gdy nastąpi próba zapisu wartości nie występującej w kolumnie drugiej tabeli to baza danych zwróci błąd i nie wykona polecenia dodania danych.
Standard SQL definiuje tzw. system transakcyjny. Każde operacje na bazie (odczyt, zapis, zmiany itd.) grupowane są w zbiory (transakcję), które powinny zostać w całości (jeżeli którakolwiek z operacji w takim zbiorze nie może zostać wykonana to cały zbiór nie powinien się wykonać). Aby transakcja mogła być transakcją musi spełnić zasadę ACID (Atomicity, Consistency, Isolation, Durability). Składa się z trzech etapów – rozpoczęcia, wykonania i zamknięcia. Ważne jest by pojedyncza transakcja wykonała się w jak najkrótszym czasie – równolegle może dokonywać się bardzo wiele innych transakcji,a część operacji w nich zawartych musi wykonywać się w określonej sekwencji. Każdy etap pojedynczej transakcji jest logowany przez co istnieje możliwość odtworzenia bazy danych bez niepożądanych informacji z uszkodzonych transakcji (niezamkniętych). Niektóre z systemów bazodanowych oferują tzw. punkty pośrednie; pozwalają one na zapis części operacji z całego zbioru przez co nie zostaną one anulowane nawet w przypadku nie wykonania reszty z nich. Bazy danych obsługują następujące polecenia transakcyjne:
- BEGIN – rozpoczęcie transakcji
- COMMIT – zatwierdzenie zmian w transakcji
- ROLLBACK – odrzucenie zmian w transakcji
- SAVEPOINT <nazwa> - punkt pośredni o określonej nazwie
- RELEASE SAVEPOINT <nazwa> - skasowanie punktu pośredniego (brak wpływu na przebieg transakcji)
- ROLLBACK TO SAVEPONT <nazwa> - przywrócenie stanu transakcji do stanu wywołanego punktu pośredniego
Oczywiście nie wszyscy programiści są chętni do stosowania baz danych. Istnieją tacy, którzy za wszelką cenę starają się wykonywać swoje aplikacje bez używania jakichkolwiek baz danych. Ich projekty bazują najczęściej na plikach XML – są to pliki tekstowe swoją budową przypominające źródło strony HTML – obydwa należą do tej samej rodziny językowej SGML. Konstrukcja tych plików jest bardzo prosta. Przykładowo zapis zawartości naszej Osoby mógłby wyglądać następująco:

```
<Osoba>
	<id>1</id>
	<imie>Darek</imie>
	<nazwisko>Iksek</nazwisko>
	<funkcja>1</funkcja>
</Osoba>
<Osoba>
	<id>2</id>
	<imie>Marek</imie>
	<nazwisko>Kółko</nazwisko>
	<funkcja>1</funkcja>
</Osoba>
<Osoba>
	<id>3</id>
	<imie>Irek</imie>
	<nazwisko>Pasek</nazwisko>
	<funkcja>3</funkcja>
</Osoba>
```

Z kolei do odczytów danych z takiej struktury istnieją wydajne klasy dzięki którym w prosty i

szybki sposób można pobrać interesujące nas dane. O ile w średnich projektach to rozwiązanie sprawdza się dobrze, o tyle w dużych może prowadzić do bardzo dużego spowolnienia odczytu danych – pomimo odpowiednio tworzonej struktury w pliku nadal mamy do czynienia z plikiem tekstowym, którego przeszukiwanie po przekroczeniu pewnego rozmiaru (przeważnie ok 300 MB) staje się bardzo nieefektywne.
Jak zostało wspomniane na początkowych zajęciach kursu, PHP potrafi współpracować niemal z każdym większym systemem bazodanowym, takim jak:
- IBM DB2
- dBase
- Firebird/InterBase
- Paradox
- SQLite
- MySQL
- Mssql
- PostgreSQL
- OracleSQL
- i wiele innych...

Ponadto PHP posiada obsługę 4 tzw. obiektów połączeniowych warstw abstrakcyjnych dla baz danych (abstrakcyjne ponieważ programista korzysta z gotowego, ujednoliconego zestawu metod połączeniowych do baz danych; to obiekt pilnuje by dane polecenie zostało poprawnie przetłumaczone dla wskazanej bazy danych):
- ODBC – Open DataBase Connectivity) – współpracuje z większością istniejących baz danych (opracowany przez SQL Access Group). Jego zaletą jest współpraca także z systemami nie bazującymi na języku SQL – w takim wypadku zapytania SQL tłumaczy on na polecenia zrozumiałe dla danego systemu bazodanowego, a jego odpowiedzi z kolei zamienia na odpowiedzi według standardu SQL
- dbx – architektura sterowników baz danych umożliwiająca obsługę m.in. OracleSQL, DB2, InterBase, MySQL, itp. Opracowana pierwotnie na potrzeby Borland Delphi/Borland C++. Obecnie standard rozwijany przez firmę Embarcadero.
- DBA – sterownik umożliwiający dostęp do baz danych według standardu Berkeley DB. Ze względu na swoją budowę jego zakres ograniczony jest jedynie do nowoczesnych baz danych, jak np. Oracle Berkeley DB.
- PDO (PHP Data Objects) – rozszerzenie języka definiujące lekki i spójny interfejs do obsługi baz danych. Jednak sam w sobie interfejs nie jest w stanie operować na jakiejkolwiek bazie – potrzebny jest mu odpowiedni sterownik. Klasa dostarczana jest od wersji PHP 5.1 wraz z parserem. Jej używanie możliwe jest z kolei od wersji 5.0. Współpraca z poprzednimi wersjami PHP nie jest możliwa przez wzgląd na brak pewnych dodatków w jądrze parsera.

1. MySQL – opis środowiska.

Wśród programistów stron WWW jednym z najbardziej popularnych systemów bazodanowych jest MySQL. Pierwotnie rozwijany był przez szwedzką firmę MySQL AB na licencji GPL. Następnie firma wraz z projektem została przejęta przez Sun Microsystems, by na koniec trafić w ręce korporacji Oracle. W międzyczasie został stworzony tzw. fork (z angielskiego rozwidlenie) kodu MySQL o nazwie MariaDB. Powołany został do życia przez jednego ze współtwórców MySQL w obawie przez zamknięciem kodu i jego pełną komercjalizacją. Projekt bazuje na dokładnie tym samym kodzie co MySQL i każda kolejna wersja dąży do zgodności z pierwowzorem.
Środowisko tworzone było z myślą głównie o szybkości działania. Pomimo nazwy ze standardem SQL miał niewiele wspólnego (poza tworzonymi zapytaniami). Nawet w chwili obecnej projekt jest tylko w większości zgodny ze standardem ANSI/ISO SQL.
Wersja 5 oferuje:
- obsługę transakcji

- tworzenie procedur składowych
- obsługę widoków
- tworzenie wyzwalaczy
- tworzenie kursorów
- wsparcie replikacji baz danych (master-slave)
- wielojęzyczność (możliwość wyboru różnych kodowań w obrębie nawet pojedynczej krotki)

Niewątpliwą zaletą jest też możliwość używania MySQL niemal na każdym systemie operacyjnym (w tym najpopularniejszych jak Unix, Linux, Windows czy Mac OS) oraz współpraca z najbardziej popularnymi językami programowania (C/C++, PHP, Java) poprzez system odpowiednich bibliotek i wtyczek.
Od wersji 4.1 istnieją dwa sposoby licencjonowania środowiska – GPL oraz komercyjne. Jeżeli aplikacja nie spełnia wymogów licencji GPL należy wykupić licencję komercyjną. Z tego też powodu w PHP wyłączono domyślnie obsługę MySQL na rzecz otwartego SQLite (licencja Public Domain).

2. Podstawowe polecenia w SQL

a) polecenie tworzenia nowej bazy danych:
CREATE DATABASE nazwabazy;

wydane polecenie utworzy nową pustą bazę danych o nazwie nazwabazy

b) polecenie usunięcia bazy danych:
DROP DATABASE nazwabazy;

wydane polecenie usuwa bazę danych o nazwie nazwabazy wraz z jej zawartością

c) polecenie tworzenia nowej tabeli danych:
CREATE TABLE tabela (nazwa_pola <typ> [AUTO_INCREMENT][UNSIGNED][ZEROFILL] [NOT NULL][UNIQUE][DEFAULT <wartosc>], [PRIMARY KEY (<nazwa_pola>)],[FOREIGN KEY (<nazwa_pola>) REFERENCES <nazwa_tabeli>(<nazwa_pola_podanej_tabeli>)]);

polecenie tworzy tabelę o nazwie tabela. W nawiasie podaje się po przecinku kolejne nazwy kolumn, pod jakimi mają być zapisywane wprowadzane dane. Każda z kolumn musi posiadać określone, jakiego typu dane będą pod nią zapisywane; dodatkowo może posiadać pewne cechy (podane na przykładzie w nawiasach []):
-AUTO_INCREMENT – wartość zapisana pod daną kolumną będzie automatycznie zwiększana przy każdorazowym dodawaniu kolejnego wiersza danych; działa na typy liczbowe
- UNSIGNED – wartości zapisywane pod kolumną nie mogą być ujemne (typy liczbowe)
- ZEROFILL – tylko dla typów liczbowych; jeżeli określimy konkretną ilość cyfr przypadającą na zapisywaną wartość liczbową, a liczba ta będzie zawierać mniejszą liczbę cyfr to baza danych automatycznie uzupełni te braki wypełniając je zerami
- NOT NULL – domyślnie dana kolumna może pozostać pusta (NULL) przy uzupełnianiu nowego wiersza danymi; dodanie wspomnianej cechy uniemożliwi dodanie wiersza jeżeli pod daną kolumną nastąpi próba zapisu wartości NULL
- UNIQUE – wartości zapisywane pod tą kolumną muszą być unikatowe; jeżeli nastąpi próba zapisu wartości już występującej pod tą kolumną w innym wierszu danych zostanie zwrócony błąd
- DEFAULT <wartosc> - jeżeli dana wartość przy dodawania nowego wiersza będzie typu NULL to we wskazanej komórce pojawi się wprowadzona przez projektanta wartość
Poza dodawaniem kolumn przy tworzeniu tabeli można także wskazać (po przecinku) która z kolumn będzie kluczem głównym tablicy, a która będzie połączona z kolumną innej tablicy (klucz obcy):

- PRIMARY KEY (<nazwa_pola>) - ustawia klucz główny w danej tabeli na kolumnę z nazwy pola; klucz główny automatycznie musi posiadać w dodawanych wierszach unikatowe wartości (nie powtarzające się wcześniej w innych wierszach danych).
- FOREIGN KEY (<nazwa_pola>) REFERENCES <nazwa_tabeli>(<nazwa_pola_podanej_tabeli>) - tworzy połączenie pola z tworzonej tabeli określonej <nazwa_pola> z polem <nazwa_pola_podanej_tabeli> wskazanej tabeli <nazwa_tabeli>; wskazana tabela MUSI istnieć już w bazie danych (tak samo wskazane pole musi w niej występować). Po połączeniu jeżeli dodawana nowa wartość do kolumny w utworzonej tabeli nie będzie istniała w powiązanym polu we wskazanej tabeli wywołany zostanie błąd spójności i dodawany wiersz nie zostanie docelowo zapiany w bazie.

Wybrane dostępne typy danych w MySQL

INT – liczba całkowita; opcjonalnie może podać maksymalną liczbę cyfr przypadającą na liczbę (np. INT(4) będzie oznaczać liczbę z przedziału +-9999, a INT(8) +-99999999)

FLOAT/DOUBLE – liczba zmiennoprzecinkowa pojedynczej/podwójnej precyzji; jak poprzednio można zdefiniować liczbę cyfr przypadającą na liczbę

CHAR – typ znakowy; można zdefiniować ilość znaków przypadających na wartość (domyślnie 1, maksymalnie 255); jeżeli np. ustalimy iż chcemy przechowywać 10 znaków, a później nastąpi zapis wartości mającej tylko 5 znaków to pozostałe 5 znaków zostanie wypełnione białymi spacjami

VARCHAR – typ znakowy o zmiennej długości; określa się tylko jego maksymalną długość (np.20), jednak gdy nastąpi zapis np. wartości 10 znakowej to dana wartość będzie miała tylko 10 znaków (w przeciwieństwie do typu CHAR)

TEXT – tekst o maksymalnej długości 65535 znaków (2 ^ 16 – 1),

LONGTEXT – tekst o maksymalnej długości 4294967295 znaków (2 ^ 32 – 1), czyli 4 GiB

BLOB – obiekt typu BLOB (Binary Large Object/Basic Large Object) umożliwiający przechowywanie np. obrazu o maksymalnej wielkości 65535 bajtów (2 ^ 16 – 1)

LARGEBLOB – obiekt typu BLOB o maksymalnej wielkości 4294967295 bajtów (2 ^ 32 – 1), czyli 4 GiB.

Oczywiście baza MySQL posiada o wiele więcej predefiniowanych typów danych – ich opis znajduje się w podręczniku na stronie http://dev.mysql.com/doc/refman/5.0/en/data-types.html

PRZYKŁAD UŻYCIA:

```
CREATE TABLE funkcja (id INT(8) AUTO_INCREMENT, nazwa VARCHAR(15), opis TEXT, PRIMARY KEY(id));
CREATE TABLE osoba (id INT(8) AUTO_INCREMENT, imie VARCHAR(20) NOT NULL, nazwisko VARCHAR(20) NOT NULL, funkcja INT, PRIMARY KEY (id), FOREIGN KEY (funkcja) REFERENCES funkcja(id));
```

Tworzy tabelę funkcja oraz osoba; kolumny id stają się głównymi kluczami, natomiast kolumna funkcja w osoba zostaje połączona z kolumną id tabeli funkcja – odtąd tylko wartości będące zapisane pod kolumną id w tabeli funkcja będą mogły być wpisywane jako wartość do kolumny funkcja w tabeli osoba.

Szczegółowy opis polecenia http://dev.mysql.com/doc/refman/5.1/en/create-table.html

d)polecenie edycji tabeli danych

1)ALTER TABLE <tabela> ADD COLUMN <nowa_kolumna> <defnicja kolumny>
dodaje nową kolumnę <nowa_kolumna> do istniejącej tabeli <tabela> jako końcową kolumnę; dodatkowo można użyć następujących opcji

ALTER TABLE <tabela> ADD COLUMN <nowa_kolumna> <defnicja kolumny> FIRST
doda kolumnę na sam początek

ALTER TABLE <tabela> ADD COLUMN <nowa_kolumna> <defnicja kolumny> AFTER <nazwa_kolumny>

dodaje nową kolumnę za kolumną <nazwa_kolumny>

2)ALTER TABLE <tabela> CHANGE COLUMN <nazwa_kolumny> <nowa_nazwa_kolumny> <definicja kolumny>
polecenie zmienia nazwę kolumny <nazwa_kolumny> na <nowa_nazwa_kolumny> oraz jej definicję (np. z typu INT na VARCHAR); opcjonalnie można dodać parametr FIRST lub AFTER <nazwa_kolumny> w celu przemieszczenia modyfikowanej kolumny; UWAGA – jeżeli zmienimy typ kolumny, a w tabeli istnieją już dane to należy liczyć się z ich utratą (bądź błędną konwersją na nowy typ danych). Dlatego należy najpierw sporządzić kopię zapasową bazy danych przed tą modyfikacją.

3) ALTER TABLE <tabela> MODIFY COLUMN <nazwa_kolumny> <definicja kolumny>
polecenie modyfikuje definicję kolumny (jej typ) i umożliwia zmianę jej położenia (parametr FIRST lub AFTER <nazwa_kolumny>)

4) ALTER TABLE <tabela> DROP COLUMN <nazwa_kolumny>
polecenie usuwa wskazaną kolumnę

Szczegółowy opis polecenia http://dev.mysql.com/doc/refman/5.1/en/alter-table.html

e)polecenie zmiany nazwy tabeli
RENAME TABLE <nazwa_tabeli> TO <nowa_nazwa_tabeli>

Szczegółowy opis polecenia http://dev.mysql.com/doc/refman/5.0/en/rename-table.html

f)polecenie usunięcia tabeli
DROP TABLE <tabela>
usuwa wskazaną tabelę z aktywnej bazy danych; można użyć następującej składni:

DROP TABLE IF EXIST <tabela>
wtedy polecenie wykona się tylko w przypadku gdy dana tabela istnieje w bazie danych

Szczegółowy opis polecenia http://dev.mysql.com/doc/refman/5.6/en/drop-table.html

g) dodanie nowego rekordu (wiersza) danych do tabeli:

INSERT INTO <tabela> VALUES (<wartosc_kolumny 1>, <wartosc_kolumny 2>, …, <wartosc_kolumny N>)
polecenie wstawia do wskazanej tabeli wartości podane w nawiasie; wartości muszą być podawane w kolejności układu tabeli (biorąc pod uwagę przykład tworzonej tabeli osoba wartosc1 odpowiada kolumnie id, wartosc2 odpowiada kolumnie imie, wartosc3 odpowiada kolumnie nazwisko itd.)
PRZYKŁAD:
INSERT INTO osoba VALUES (1, 'Marek', 'Drugi', 1)

należy jednak mieć na uwadze, że bezcelowe podawanie jest parametru pierwszego (id) – cokolwiek nie wstawimy baza i tak zmieni go na numer nadawany wedle wewnętrznej numeracji. Jak zatem dodać tylko wybrane wartości zamiast wszystkich? Odpowiedź jest prosta:

INSERT INTO <tabela> (<nazwa_kolumny>,...,<nazwa_kolumny>) VALUES (<wartosc_kolumny>, … , <wartosc_kolumny>);

dzięki tej modyfikacji możemy wstawiać wartości tylko pod określone kolumny, np.

INSERT INTO osoba (imie, nazwisko) VALUES ('Marek', 'Drugi');
w ten sposób dodamy tylko imię oraz nazwisko; id zostanie dodane automatycznie (kolejny numer z bazy danych), a w kolumnie funkcja zagości wartość NULL

Szczegółowy opis polecenia http://dev.mysql.com/doc/refman/5.6/en/insert.html
h) edycja istniejących już danych w tabeli:
UPDATE <nazwa_tabeli> SET
<nazwa_kolumny>=<nowa_wartosc>,...,<nazwa_kolumny>=<nowa_wartosc>

polecenie ustawia nowe wartości dla wskazanych kolumn we wskazanej tabeli; powyższym poleceniem zostaną zaktualizowane WSZYSTKIE rekordy (wiersze) wskazanej tabeli; aby zamienić tylko konkretne rekordy należy użyć dodatkowego parametru:

UPDATE <nazwa_tabeli> SET <nazwa_kolumny>=<nowa_wartosc> WHERE <nazwa_kolumny>=<wartosc>
dzięki tej podmianie tylko wiersze zawierające wskazaną <wartosc> pod wskazaną kolumną będą mieć zmienioną wartość wskazanej kolumny (po słowie SET)

Szczegółowy opis polecenia http://dev.mysql.com/doc/refman/5.0/en/update.html

i) usunięcie wskazanych danych w tabeli:
DELETE FROM <tabela> WHERE <kolumna>=<wartosc>
polecenie usuwa ze wskazanej tabeli wszystkie wiersze, które zawierają wskazaną <wartosc> pod wskazaną <kolumna>

Szczegółowy opis polecenia http://dev.mysql.com/doc/refman/5.0/en/delete.html

j) wybranie i wyświetlenie zawartości ze wskazanej tabeli:
SELECT * FROM <tabela>
polecenie wybierze wszystkie kolumny (odpowiada za to operator *) ze wskazanej tabeli i wyświetli na ekranie. Oczywiście takie polecenie sprawdzi się tylko dla małej tablicy z małą ilością wierszy. Można wybrać tylko określone kolumny:

SELECT <kolumna1>,<kolumna2>,...,<kolumnaN> FROM <tabela>
PRZYKŁAD:
SELECT imie,nazwisko FROM osoba

Istnieje możliwość zamiany wyświetlanej nazwy kolumny na bardziej przyjazną, czytelniejszą dla odbiorcy (PRZYKŁAD):
SELECT imię AS 'Imię osoby',nazwisko AS 'Nazwisko osoby' FROM osoba

Innym sposobem ograniczenia wyświetlania wyników jest podanie warunku WHERE
SELECT * FROM <tabela> WHERE <kolumna>=<wartosc>
wyświetlą się tylko te wiersze które zawierają wskazaną wartość

MySQL wspiera obsługę wszystkich operatorów porównywania wartości (<,>,>=,<=,=) oraz podawanie kilku warunków do spełnienia (spójniki warunków to AND oraz OR); ponadto istnieje dodatkowe polecenie BETWEEN, dzięki któremu możemy pewien akceptowalny zakres wartości (np. id BETWEEN 5 AND 10 spowoduje wybranie wierszy posiadające wartość kolumny id pomiędzy 5 a 10).

PRZYKŁAD:

SELECT nazwisko,funkcja FROM osoba WHERE (imie='Marek' AND id BETWEEN 1 AND 10) OR funkcja = 2
polecenie wyświetli zawartość kolumn nazwisko oraz funkcja tylko z wierszy, w których kolumna imie zawiera wartość Marek oraz kolumna id należy do przedziału 1-10 lub jeżeli kolumna funkcja ma wartość 2

Wyniki polecenia SELECT można porządkować względem wartości wskazanej odpowiedniej kolumny
SELECT * FROM <tabela> WHERE <kolumna>=<wartosc> ORDER BY <kolumna>

Domyślnie porządkowanie odbywa się od najmniejszej do największej wartości liczbowej (lub alfabetycznie od a do z); jeżeli polecenie wywołane zostanie z dodatkowym słowem:
SELECT * FROM <tabela> WHERE <kolumna>=<wartosc> ORDER BY <kolumna> DESC

to kolejność porządkowania zostanie odwrócona.

Dane mogą być wybierane także poprzez dopasowanie przez wzorzec. Służy do tego słowo LIKE podawane w klauzuli WHERE
SELECT * FROM <tabela> WHERE <kolumna> LIKE '%wyr%'
oznacza to, że wyświetlone zostaną tylko te wiersze, w których <kolumna> będzie zawierać fragment słowa wyr. Znak procentowy zastępuje dowolny ciąg znakowy; w powyższym przypadku wyrażenie więc oznacza, że zarówno przed jak i po fragmencie wyr mogą się znaleźć dowolne znaki. Możliwe kombinacje
wyr% - ciąg zaczynający się od znaków wyr; dalsza część dowolna
%wyr – ciąg kończący się znakami wyr; dowolny ciąg znakowy przed
%wyr% - kombinacja z przykładu; zarówno przed jak i po znakach wyr ciąg znakowy dowolny
w%yr – może wystąpić dowolny ciąg znakowy po literze w, a przed literami yr
itd...

Ostatnim z ważniejszych parametrów polecenia SELECT jest LIMIT. Określa ona ile konkretnie rekordów ma zostać pobranych ze wskazanej tabeli. Istnieją trzy sposoby użycia tej klauzuli
1)SELECT * FROM <tabela> LIMIT 5 #pobierze pierwszych 5 wierszy
2)SELECT * FROM <tabela> LIMIT 5,10 #pobierze wiersze od 6 do 15
3)SELECT * FROM <tabela> LIMIT 5 OFFSET 10 #pobierze rekord 10-15

Rozwiązanie numer 3 jest zgodne ze standardem SQL (Oracle, PostgreSQL, SQLite, MsSQL); dwa pierwsze to rozwiązania dostępne jedynie w MySQL.

Szczegółowy opis polecenia http://dev.mysql.com/doc/refman/5.0/en/select.html

2a. Wybrane funkcje dostępne w MySQL

MySQL, tak jak i inne języki, posiada szereg wbudowanych funkcji. Poniżej zaprezentowane zostaną najczęściej używane:

RAND(N) – funkcja generuje pseudolosową zmienną zmiennoprzecinkową; parametr N jest opcjonalny; jeżeli zostanie podana pod niego jakaś wartość to będzie ona służyć za tzw. sadzonkę losowości (pozwala na uzyskanie większego stopnia losowości)
PRZYŁAD:
SELECT RAND();
SELECT RAND(4);

COUNT(wyrażenie) – funkcja zlicza ilość elementów podanych jako wyrażenie; występuje także w formie COUNT (DISTINCT wyrażenie). W tym drugim przypadku zliczane są tylko unikatowe wystąpienia danego wyrażenia.
PRZYKŁADY:
SELECT COUNT(*) FROM <tabela> - zlicza wszystkie wystąpienia jakichkolwiek danych we wskazanej tabeli; ponieważ jako wyrażenie została podana * to można przyjąć iż funkcja zwróci liczbę wszystkich wierszy z danymi
SELECT COUNT(funkcja) FROM osoba- w tym wypadku funkcja zwróci liczbę równą ilości wystąpień danych w kolumnie funkcja; wiersze, w których pod tą kolumną będzie wpisana wartość NULL nie zostaną wliczone do wyniku
SELECT COUNT(DISTINCT funkcja) FROM osoba – pod kolumną funkcja wartości mogą się powtarzać (wynika to z faktu, że kilka osób może pełnić tę samą funkcję). Może jednak zajść potrzeba zliczenia ile zostało obsadzonych stanowisk (a nie ile suma ile stanowisk zostało obsadzonych i ile osób łącznie ma stanowisko). Przedstawiony przykład zliczy tylko liczbę osadzonych stanowisk – jeżeli wartości danych się powtórzą to nie zostaną wliczone do wyniku funkcji (dzięki słowu DISTINCT).

CURDATE() - funkcja zwraca aktualna datę; możliwe są dwa wywołania:
SELECT CURDATE(); - zwróci datę wedle standardu 'YYYY-MM-DD' (ciąg znaków)
SELECT CURDATE() + 0; - zwróci datę jako typ numeryczny YYYYMMDD

CURTIME() - zwraca aktualny czas
SELECT CURTIME(); - zwróci czas wedle standardu 'HH:ii:ss'
SELECT CURTIME() + 0; – zwróci datę jako float Hhiiss.000000

CONCAT(str1,str2,...,strN) – funkcja łączy podane jako parametry ciągi znakowe
SELECT CONCAT(imie, ' ', nazwisko) AS 'Imię i nazwisko' FROM osoba – polecenie połączy wyniki z tabeli osoba dostępne pod kolumnami imie oraz nazwisko; wyświetlą się one pod nazwą „Imię i nazwisko”

REPLACE(ciag_znakowy,do_zmiany,wymiana_na) funkcja zamienia w podanym ciągu znakowym znak (podciąg znakowy) do_zamiany na znak (ciąg znakowy) wymiana_na
PRZYKŁAD:
SELECT REPLACE('Czesc', 'sc', 'ść!'); – wydrukuje napis 'Cześć!'

REVERSE(ciag_znakowy) – odwraca wspak podany ciąg znakowy
SELECT REVERSE('abcd'); - zwróci 'dcba'

Wszystkie dostępne funkcje i operatory http://dev.mysql.com/doc/refman/5.0/en/functions.html

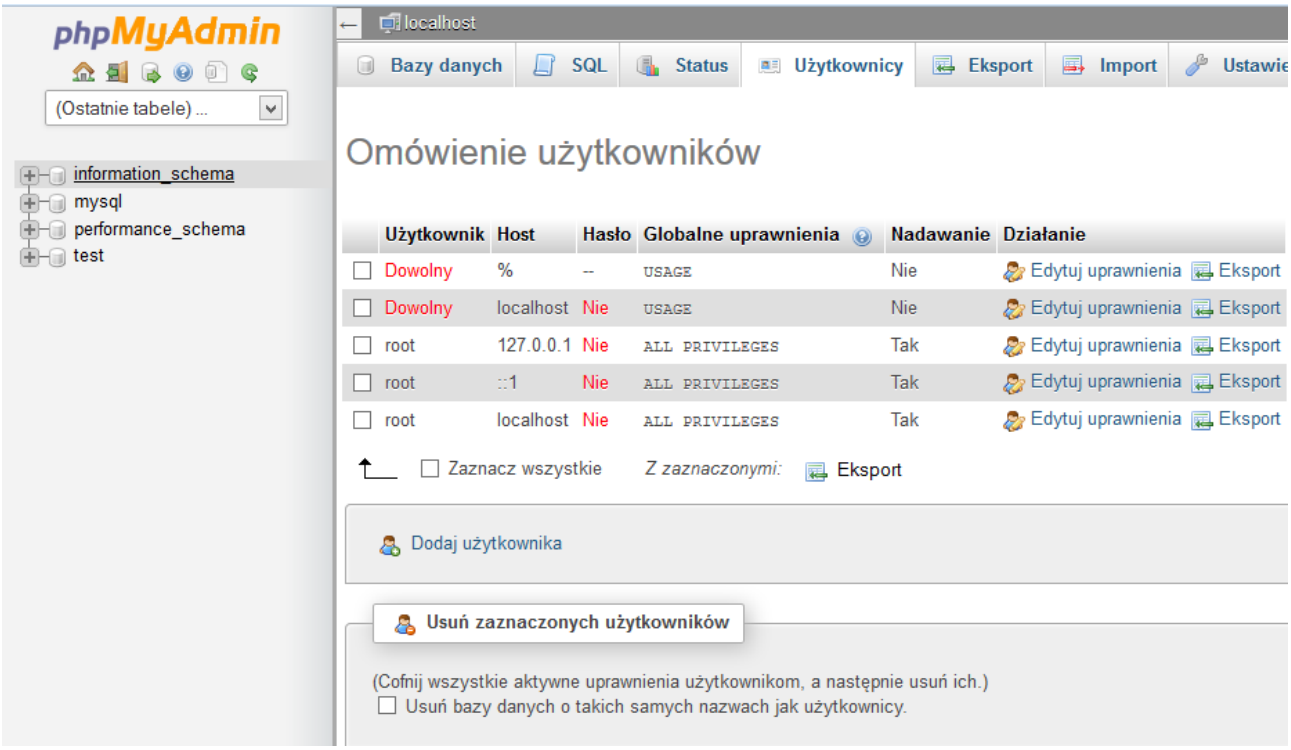
3. MySQL a phpMyAdmin

Dzięki pakietowi aplikacji dla programistów WWW WAMP/XAMPP od momentu ich instalacji posiadamy serwer baz danych MySQL. Dostarczona baza jest już wstępnie skonfigurowana (domyślnie) i od razu gotowa do użytku. Jednak by jej używać należy wykorzystywać wiersz poleceń – parser poleceń znajduje się (zakładając domyślą instalację pakietów) w C:\wamp\bin\mysql\mysql5.6.12\bin\mysql.exe (oczywiście różnica może wystąpić w dostarczonej wersji MySQL – wtedy trzeba zamienić fragment mysql5.6.12 na aktualnie dostępny) lub w C:\xampp\mysql\bin\mysql.exe. Oczywiście należy zadbać o to by baza danych była uruchomiona jako usługa – wystarczy uruchomić cały pakiet WAMP bądź samą bazę danych w przypadku XAMPP z jego panelu administracyjnego.
Taki sposób dostępu do bazy danych jest wygodny i wydajny. Jeżeli jednak nie będziemy korzystać

z własnego serwera WWW to należy pamiętać, że dostawcy przeważnie nie udostępniają możliwości zarządzania dostarczoną przez nich bazą danych przez linię poleceń. Dlatego powstał projekt phpMyAdmin. Dzięki niemu zarządzanie bazami danych dostępne jest z każdej przeglądarki WWW; projekt w całości pisany jest w PHP.
W przypadku naszych serwerów dla Windows (zarówno WAMP jak i XAMPP) phpMyAdmin dostępny jest pod adresem

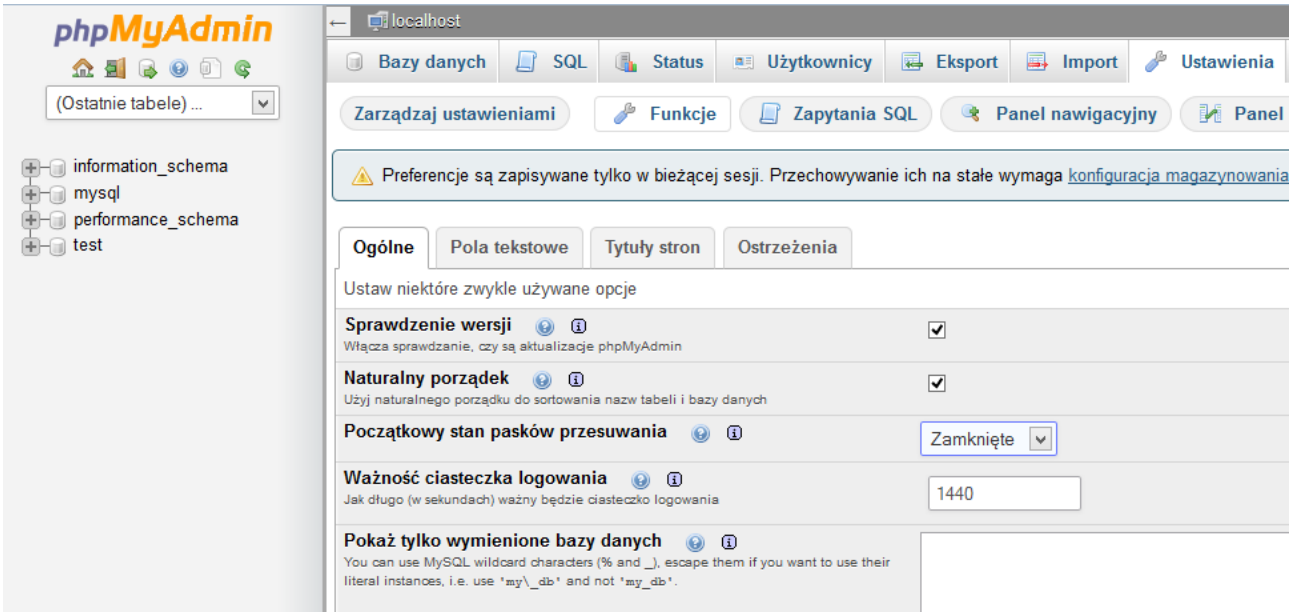
http://localhost/phpmyadmin/

Domyślnie dostęp do bazy danych ma jej główny administrator o nazwie 'root', dla którego nie jest ustawione żadne hasło. Takie ustawienie jest niebezpieczne, dlatego zaraz przy pierwszej konfiguracji należy albo zamienić je w pliku konfiguracyjnym my.ini lub poprzez phpMyAdmin poprzez zakładkę 'Użytkownicy'



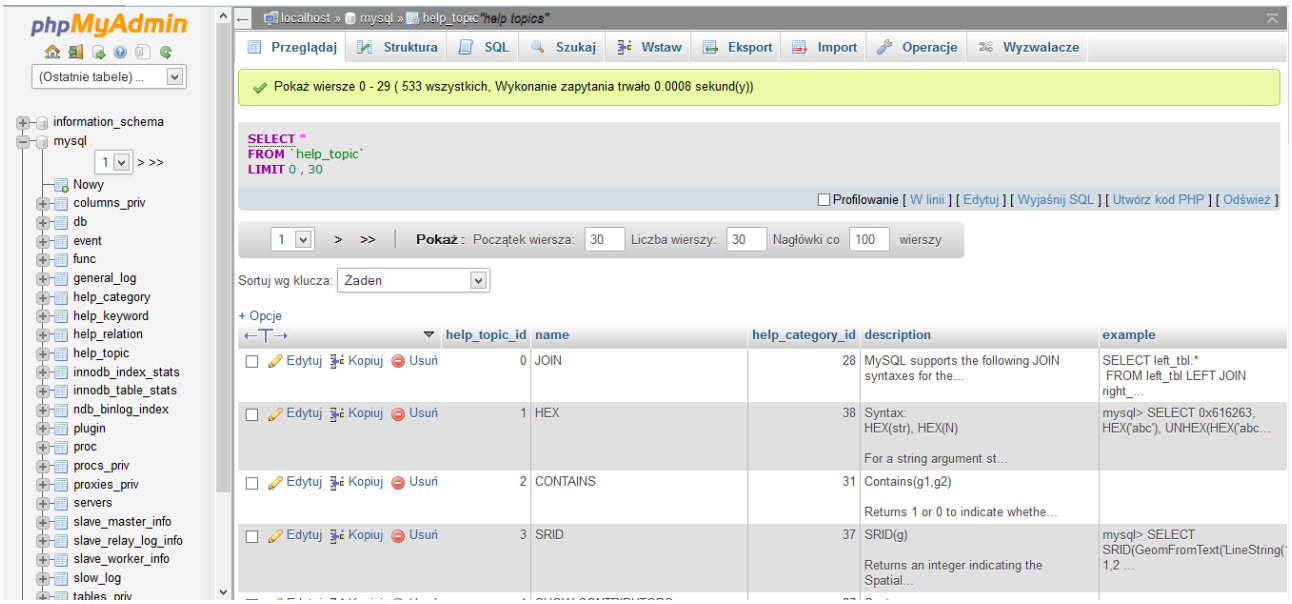
Rys 1. Zakładka użytkownicy w phpMyAdmin

Dobrym rozwiązaniem jest WYŁĄCZENIE konta root na działającym serwerze WWW i wykorzystywanie tylko stworzonych przez siebie kont użytkowników z tylko odpowiednio nadanymi uprawnieniami (przykładowo jeden użytkownik powinien mieć dostęp tylko do swoich baz danych – nie do wszystkich). Należy również pamiętać o bardzo bezpiecznym haśle dostępowym – najlepiej je wygenerować poprzez losową funkcję; jest to szczególnie zalecane w przypadku wykorzystywania baz danych w skryptach PHP – hasło to podawane jest w odpowiedniej funkcji (najczęściej jednorazowo) przez co nie musi być intuicyjne.
Dla celów naszych zajęć nie trzeba jednak czegokolwiek zmieniać w użytkownikach i ich uprawnieniach – można korzystać z konta root bez podawania hasła. Dobrze jest jednak zmienić wartość ważności ciasteczka logowania – domyślnie ustawione jest na 1440 sekund (około 24 minuty); po tym czasie, jeżeli nic nie wykonamy na stronie phpMyAdmina (ważne – chodzi o PRZEŁADOWANIE strony, nie np. pisanie polecenia w polu tekstowym) to zostaniemy automatycznie wylogowani. Ustawienie krótkiego życia sesji jest ważna w przypadku używania phpMyAdmin na zewnętrznym serwerze WWW (wtedy warto nawet zmniejszyć wartość do 180 sekund – czyli 3 minut), jednak w przypadku używania go lokalnie (np. w WAMP czy XAMPP) staje się to uciążliwe. Dlatego warto jest zwiększyć ważność logowania do np. 1 godziny (3600)



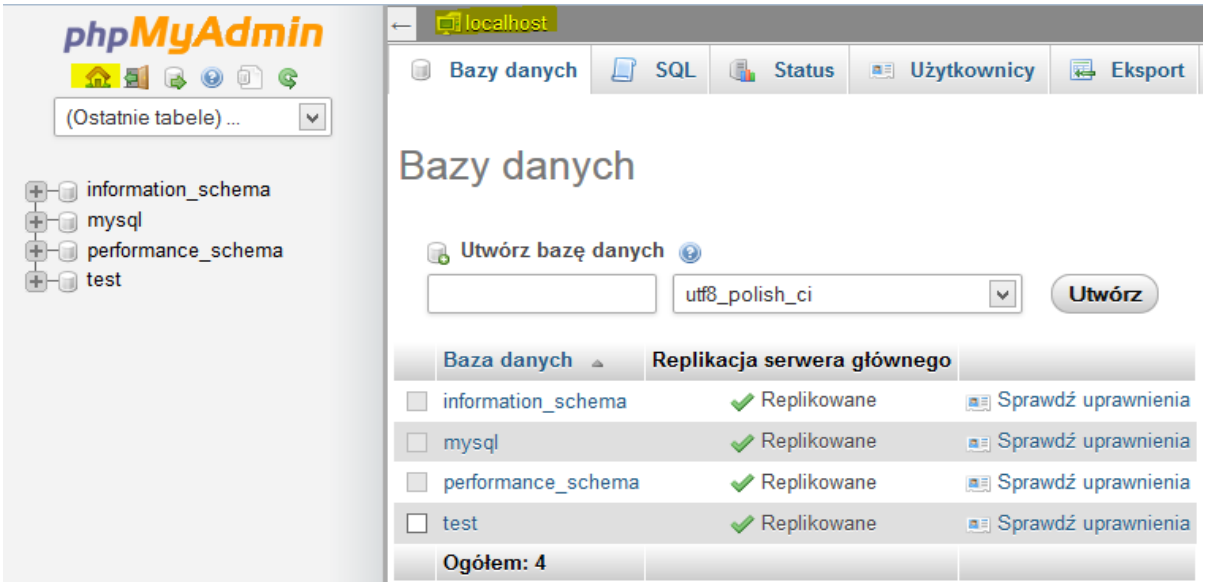
Rys 2. Dostęp do ustawień Ważności ciasteczka logowania (Ustawienia->Funkcje)

Po lewej stronie narzędzia dostępny jest panel nawigacyjny po wszystkich dostępnych tabelach dla danego użytkownika (w wypadku użytkownika root jest to dostęp do WSZYSTKICH dostępnych baz danych). Dzięki temu łatwo możemy przełączać się po kolejnych bazach danych i podglądać ich zawartość (dostępne w nich tabele). Z kolei klikając dowolną tabele wywołamy automatycznie polecenie SELECT * FROM <tabela> LIMIT 0,30. W prosty sposób (za pomocą przycisków nawigacyjnych) możemy się przełączać pomiędzy kolejnymi wierszami danych dostępnymi w tabeli; można również w prosty sposób zmieniać ilość wyświetlanych danych jak i linię od której zaczną być pobierane dane.



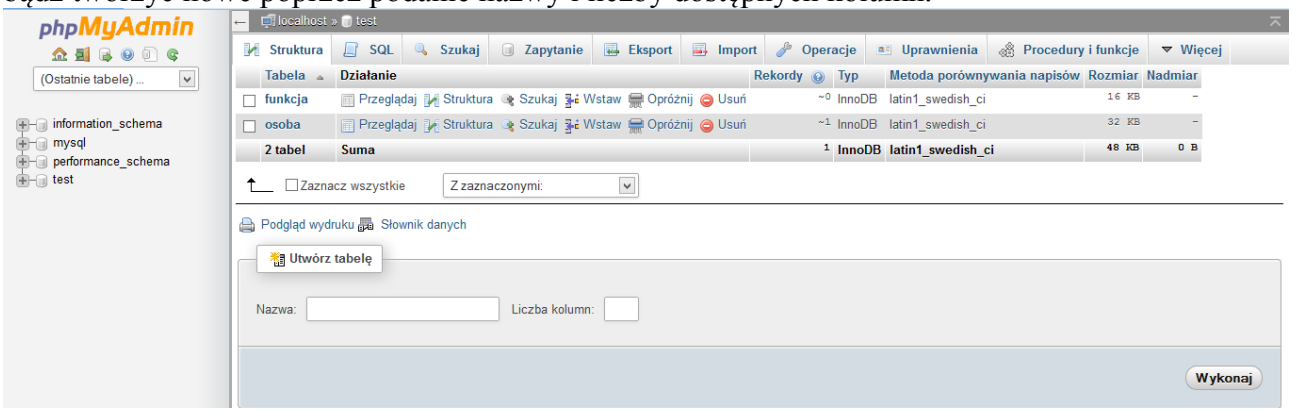
Rys 3. Przykład wyświetlenia zawartości jednej z dostępnych tabel.

Dodawanie nowej bazy danych również nie sprawia większych problemów – wystarczy podać nową nazwę bazy i kliknąć przycisk 'Utwórz'. Dobrym rozwiązaniem jest od razu wybrać domyślną metodę porównywania napisów (najlepiej utf8_polish_ci). Aby utworzyć nową bazę danych trzeba znajdować się na najwyższym poziomie bazy danych (pasek nawigacyjny musi zawierać tylko nazwę naszego serwera – w tym wypadku localhost, lub trzeba kliknąć w lewym panelu nawigacyjnym ikonę domu – Wejście).



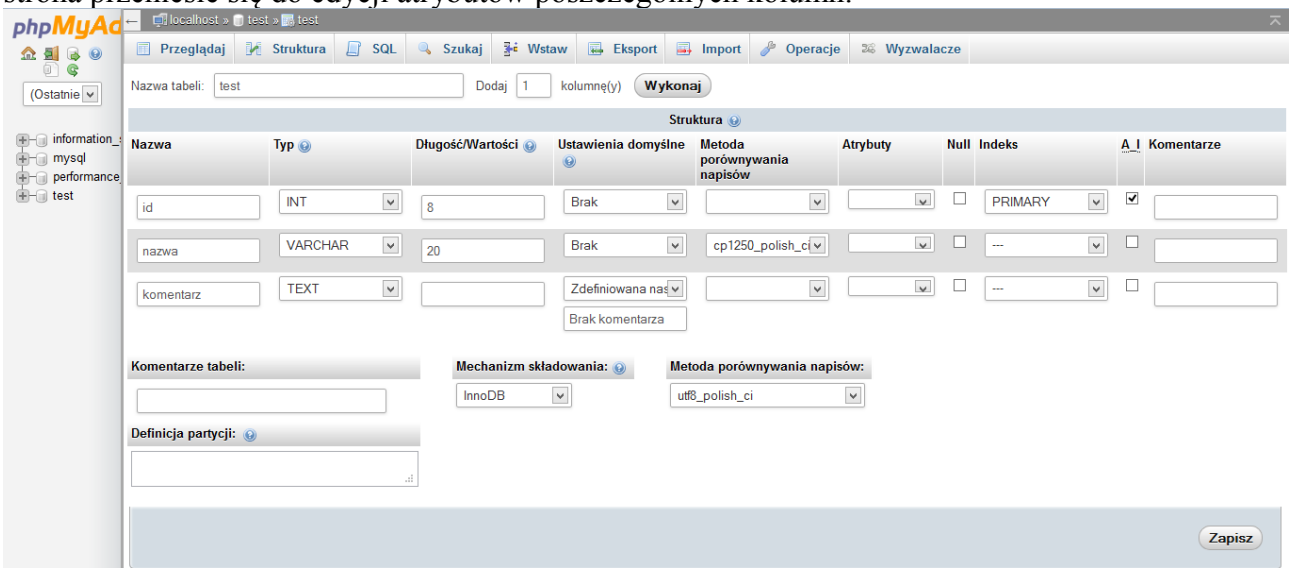
Rys 4. Zakładanie nowej bazy danych (żółtym kolorem zaznaczone zostały odnośniki, po kliknięciu których pojawia się w menu przycisk 'Bazy danych').

Po utworzeniu bazy danych można ją wybrać poprzez kliknięcie jej nazwy w panelu nawigacyjnym (o ile sama się nie wybierze). Na wyświetlonej stronie można przeglądać dostępne tabele z danymi bądź tworzyć nowe poprzez podanie nazwy i liczby dostępnych kolumn.



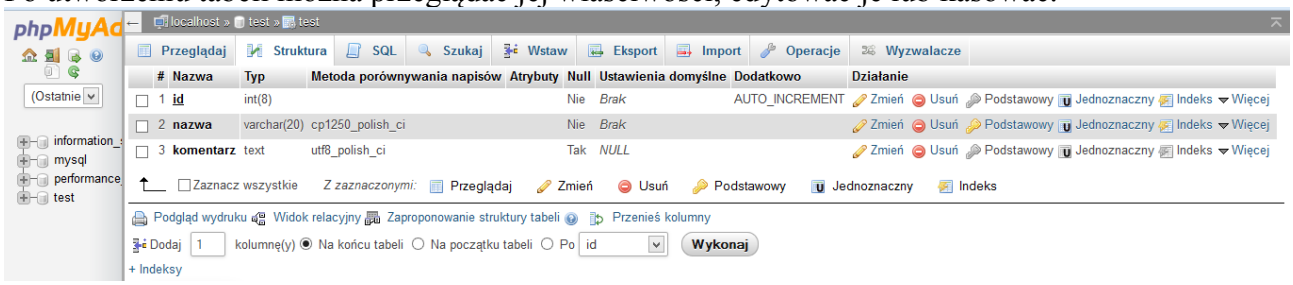
Rys 5. Widok przeglądania/edycji/dodawania nowych tabel w bazie danych.

Jeżeli zechcemy dodać nową tabelę (podamy jej nazwę + liczbę kolumn) klikając przycisk wykonaj strona przeniesie się do edycji atrybutów poszczególnych kolumn.



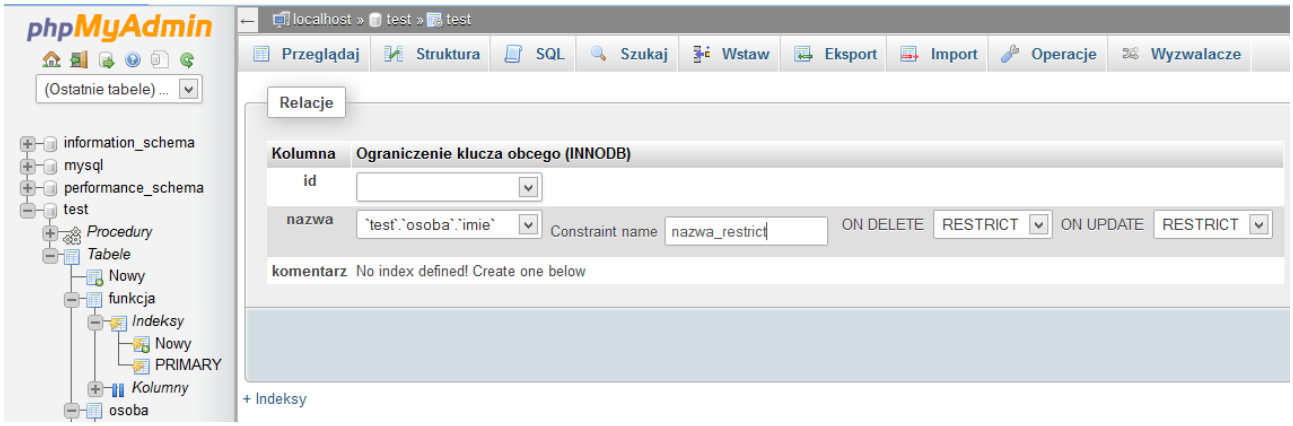
Rys 6. Widok edycji szczegółów poszczególnych kolumn nowej tabeli danych

W pierwszej kolumnie podajemy nazwy tworzonych kolumn, w drugiej określamy typ przechowywanych danych, pod Długość/Wartości podajemy maksymalny rozmiar (liczba znaków/cyfr) przechowywanej zmiennej. Pod Ustawienia domyślne możemy określić jakimi wartościami zostaną uzupełnione pola tabeli w przypadku gdy osoba wprowadzająca dane nie uzupełni danej kolumny. Opcjonalnie możemy ustawić Metodę porównywania napisów (jeżeli chcemy określić je dla danego pola indywidualnie). Pod polem Atrybuty można ustawić jakiego typu będą przechowywane dane (binarne, bez znaków, itd.). Jeżeli zaznaczymy opcję NULL dane pole w tworzonej tabeli będzie mogło przyjąć pustą wartość. Pod indeks można określić czy dane pole będzie kluczem głównym tabeli czy też typu Index/Unique/Fulltext. Następna opcja, jeżeli zostanie zaznaczona, będzie zwiększać automatycznie wartość danej kolumny o 1. Można również dodać komentarz do kolumny – będzie wtedy widoczny dla osoby wprowadzającej dane (np. wskazówki jakie dane powinny znaleźć się w danej kolumnie).
Po utworzeniu tabeli można przeglądać jej właściwości, edytować je lub kasować.



Rys 7. Widok podglądu/edycji kolumn utworzonej tabeli

phpMyAdmin pozwala na tworzenie kluczy obcych. W celu stworzenia takiego połączenia należy danej kolumnie w tabeli ustawić w polu Indeks wartości INDEX. Taką wartość musi posiadać zarówno pole,na które zostanie nałożona restrykcja jak i pole z drugiej tabeli, które będzie stanowić tą restrykcję. Kiedy ustawimy już wspomniane wartości otwieramy tabele zawierającą kolumnę, która ma posiadać restrykcję. Wybieramy opcję 'Widok relacyjny' (poniżej wypisu dostępnych kolumn).

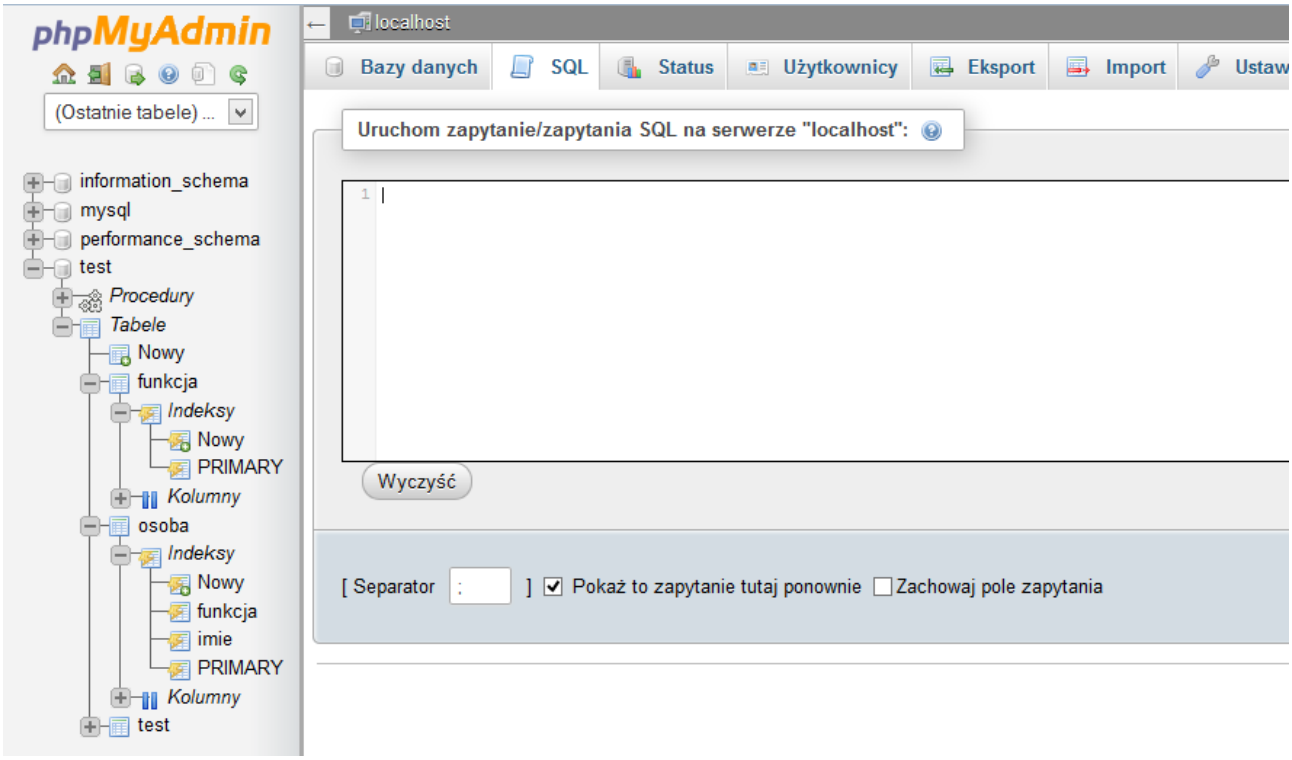


Rys 8. Dodawanie klucza obcego dla pola edytowanej tabeli (po wybraniu opcji 'Widok relacyjny').

Jak widać na Rys 8. tabela test posiada kolumnę nazwa, dla której została nałożona restrykcja wartości spod tabeli test.osoba.imie (ostatni człon to nazwa kolumny). Proszę zwrócić uwagę na wygląd apostrofów, w które ujęte są nazwy zmiennych w bazie – są to tzw. odwrócone apostrofy (` - dostępne przeważnie nad klawiszem tabulacji,wraz z tyldą ~) a nie ' - dostępne wraz z klawiszem cudzysłowu. Gdyby nie użycie odwróconego apostrofu SQL potraktował by nazwę zmiennej jako zwykły ciąg znakowy, przez co nie odwołalibyśmy się do pożądanej kolumny. W Constraint name należy podać nazwę tworzonego klucza obcego (dowolna,jednoznaczna). Domyślnie zarówno przy kasowaniu jak i aktualizacji mamy domyślną akcję wyzwalacza – restrict (można zmienić na inne).

Powyżej zostały przedstawione główne, podstawowe operacje na bazie danych przy użyciu

narzędzia phpMyAdmin. Jak można zauważyć nawet bez znajomości poleceń SQL użytkownik jest dzięki niemu w stanie zaprojektować, edytować i wykorzystywać bazy danych dla swoich potrzeb. Trzeba jednak zaznaczyć, że dla doświadczonego użytkownika/programisty tworzenie i zarządzanie bazami w ten sposób jest uciążliwy – o wiele szybsze i efektywniejsze jest bowiem pisanie poleceń/skryptów, które przy odpowiedniej modyfikacji/zastosowaniu zmiennych można wykorzystywać wielokrotnie dla różnych baz danych/tabel bez konieczności ciągłego przełączania się pomiędzy stronami i ciągłego klikania myszą. PhpMyAdmin umożliwia wykonywanie własnych poleceń/skryptów poprzez swoją stronę; w tym celu trzeba wybrać zakładkę SQL



Rys 9. Wybrana zakładka SQL.

Przykładowy skrypt SQL:

```
CREATE DATABASE test2;
USE test2;
CREATE TABLE przykład (id INT(11) AUTO_INCREMENT, nick VARCHAR(30), haslo TEXT, userLevel TINYINT(1), PRIMARY KEY (id));
INSERT INTO przykład (nick, haslo, userLevel) VALUES ('moderator', 'tajne', 1);
SELECT nick FROM przykład WHERE id = 1;
DROP DATABASE test2;
```

Powyższy skrypt utworzy bazę danych test2 i wybierze ją jako aktywną. Następnie utworzy tabelę przyklad i wstawi do niej nową wartość. Na koniec wyświetli nam nick osoby pod id = 1 i skasuje całą utworzoną bazę danych. Oczywiście skrypt będzie „lepiej”wyglądał gdy zapiszemy go w tej sposób:

```
CREATE DATABASE `test2`;
USE `test2`;
CREATE TABLE `przykład` (`id` INT(11) AUTO_INCREMENT, `nick` VARCHAR(30), `haslo` TEXT, `userLevel` TINYINT(1), PRIMARY KEY (id));
INSERT INTO `przykład` (`nick`, `haslo`, `userLevel`) VALUES ('moderator', 'tajne', 1);
SELECT `nick` FROM `przykład` WHERE `id` = 1;
DROP DATABASE `test2`;
```

jednak jest to bardziej w tym wypadku zabieg kosmetyczny – ujęcie nazw baz, tabel czy kolumn w odwrócone apostrofy wymagane jest jedynie gdy nazwa rozdzielona jest białą spacją (np. zamiast `haslo` brzmiałaby `haslo dostepu`).

Jak zostało wspomniane w SQL można tworzyć też własne procedury i funkcje. Różnica pomiędzy procedurą i funkcją, poza samym wywołaniem (procedurę wywołuję się słowem CALL <nazwa procedury>, a funkcję SELECT <nazwa_funkcja>) jest taka, że procedura NIE ZWRACA żadnej wartości podczas gdy funkcja musi zawierać wartość zwracaną.
Przykład funkcji:

```
USE test;

delimiter ;;

CREATE FUNCTION testowaF() RETURNS INT
DETERMINISTIC
BEGIN
DECLARE a INT;
SET a = 10;
RETURN a;
END;;

delimiter ;
```

W powyższym przykładzie użyliśmy bazy danych test. Kolejna linijka kodu zmienia nam domyślny separator kolejnych linii kodu na ';;' - jest to wymagane ponieważ w ciele funkcji będzie trzeba poszczególne linie kodu separować ';'. Jeżeli główny kod SQL nadal posiadałby także separator w postaci pojedynczego średnika nastąpiłby konflikt i (teoretycznie) funkcja zakończyła by się po 1 linii (oczywiście wywołałby się błąd pisanego skryptu i nic nie zostało by dodane do bazy danych). W kolejnej linii podajemy nazwę funkcji (testowaF()) pod jaką ma nasza funkcja występować; zaraz po nazwie musi znaleźć się słowo RETURNS,a po nim typ wartości jaki funkcja będzie zwracać (w tym wypadku INT; może być dowolny typ danych obsługiwany przez bazę danych). W następnej linii musi znaleźć się jedno z kluczowych słów – (NOT) DETERMINISTIC, READS SQL DATA lub NO SQL (użyte w przykładzie słowo DETERMINIST oznacza, że bez względu na parametry wejściowe wynik funkcji zawsze będzie ten sam – bez modyfikacji tychże parametrów). Słowo DECLARE pozwala na deklarację nowych zmiennych potrzebnych do działania funkcji (podajemy nazwę oraz typ jaki ma przechowywać).
Słowo SET umożliwia ustawienie wartości dla podanej zmiennej (zawsze musi wystąpić przed podaniem wartości).
Słowo RETURN kończy działanie funkcji – funkcja zwraca określoną wartość).
Zamiast nawiasów klamrowych (znanych z PHP) stosuje się słowa BEGIN oraz END (styl języka Object Pascal). Ustawiony przez nas separator kodu MUSI wystąpić po słowie END – inaczej otrzymamy błąd skryptu. Na koniec MUSIMY również zamienić z powrotem separator linii na standardowy – obligatoryjne jest to tylko w przypadku dodawania funkcji poprzez phpMyAdmin (poprzez linię poleceń możemy korzystać z dowolnych separatorów).

Dodaną funkcję można wywołać poprzez polecenie:

```
SELECT testowaF();
```

Pisanie procedur jest bardzo podobne. Przykładowa procedura wygląda następująco:

```
USE test;
```

```
delimiter ;;

CREATE PROCEDURE testP()
BEGIN
        SELECT 'HELLO! I\'M FROM MySQL PROCEDURE!';
END;;

delimiter ;
```

Tak jak poprzednio ustawiamy odpowiedni separator (i zmieniamy ponownie na końcu). Jak widać procedura nie posiada słów RETURNS oraz RETURN na zakończenie działania. Procedura wykona jedynie linie w niej zawarte – tak jak w funkcji można deklarować zmienne, ustawiać ich wartość bądź przyjmować je poprzez parametry. W testowanym przykładzie wywołanie procedury spowoduje wyświetlenie tekstu podanego po słowie SELECT. Wywołanie procedury następuje poprzez:

```
CALL testP();
```

Usuwanie funkcji/procedury

```
DROP PROCEDURE testP;
DROP FUNCTION testowaF;
```

Zarówno procedury jak i funkcje mogą zostać wzbogacone o instrukcje warunkowe, pętle sterujące:
http://dev.mysql.com/doc/refman/5.0/en/control-flow-functions.html (CASE/IF)
http://dev.mysql.com/doc/refman/5.0/en/loop.html (FOR/LOOP)
http://dev.mysql.com/doc/refman/5.0/en/while.html (WHILE)
http://dev.mysql.com/doc/refman/5.0/en/repeat.html (DO...WHILE/REPEAT...UNTIL)

Szczegółowy opis tworzenia funkcji i procedur znajduje się pod adresem http://dev.mysql.com/doc/refman/5.0/en/create-procedure.html .

Zadania do realizacji:

I. Proszę utworzyć bazę danych użytkowników portalu. Baza powinna zawierać imiona, nazwiska, login, hasło, uprawnienia użytkownika, obrazek, 'rangę' na forum oraz datę ostatniego poprawnego logowania. Dobrym rozwiązaniem jest również dodać pole z adresem IP, z którego to logowanie nastąpiło. Baza powinna zawierać także rangi użytkowników wraz z opisem kiedy użytkownik ją otrzymuje. To od projektanta bazy zależy czy rangi otrzymuje się od ilości poprawnych logowań, czasu jaki upłynął od rejestracji czy też od aktywności na stronie (np. liczba komentarzy/postów). Wynika z tego, że użytkownikowi nie można przypisać rangi spoza puli. Podobnie powinno być z uprawnieniami użytkownika – uprawnienia także powinny być przechowywane oddzielnie (co najmniej: administrator, moderator, użytkownik; może być także młodszy moderator, młodszy administrator itp.)
UWAGA – wykonanie tego ćwiczenia oraz jego wynik zależy wyłącznie od przemyśleń i realizacji wykonującego. Dotyczy to zwłaszcza rozmieszczenia danych w bazie danych (ilości tabel) jak i utworzonych w tych tabelach kolumn. Ważne jedynie by dane nie się uzupełniały i nie powtarzały – przykładowo aby nie trzeba było przy połowie użytkowników wpisywać słowa „użytkownik”, a pobierać je z odpowiedniego pola innej tabeli.
Po zaprojektowaniu bazy proszę spróbować dodać po kilka pozycji do bazy danych, spróbować je wyświetlić. Proszę zastanowić się jak za pomocą jednego polecenia wyświetlić dane z wielu tabel

(np. zamiast id uprawnień użytkownika będącego np. typem numerycznym wyświetlić faktyczną nazwę tegoż uprawnienia mieszczącego się w drugiej tabeli); do realizacji tego polecenia dobrym pomysłem jest przestudiowanie dokumentacji polecenia SELECT.
II. Zadanie polega na stworzeniu prostego sklepu internetowego. Baza danych ma zawierać tabelę (kolumny zależne od zamysłu realizatora) z użytkownikami, ich wirtualnym portfelem, dostępnymi produktami, oraz historią zakupów. Trzeba pamiętać iż tabela z produktami powinna mieć stan magazynowy, który podczas zakupów nie powinien spaść poniżej zera! Z kolei jeżeli użytkownik ma ujemny bilans konta także nie powinien mieć możliwości zakupu produktów (za to można zrobić zapis do bazy danych z historią o potencjalnej próbie zakupu). Zadanie można zrealizować za pomocą procedur lub funkcji (opartych na prostych poleceniach; jedyną instrukcją sterującą powinien być IF sprawdzający np. ilość dostępnego produktu i/lub saldo użytkownika). Przydatne informacje można znaleźć pod adresem (w celu zapisania w zmiennej określonego wyniku z polecenia SELECT):
http://stackoverflow.com/questions/3075147/select-into-variable-in-mysql-declare-causes-syntax-error
Tutaj więcej o parametrach wejściowych procedur/funkcji:
http://www.mysqltutorial.org/stored-procedures-parameters.aspx
http://stackoverflow.com/questions/5039324/creating-a-procedure-in-mysql-with-parameters

4. Używanie PHP i MySQL

Dostęp do bazy danych z poziomu PHP odbywa się poprzez odpowiedni sterownik bazy danych. Do wersji PHP 5 dostępny był oryginalny sterownik Mysql dostarczony jeszcze przez firmę Mysql AB; jego wydajność jest już znacznie obniżona – głównie poprzez niedostosowanie do MySQL w wersji wyższej niż 4.1. Dlatego na stronie dokumentacji jest silne zalecane porzucenie tego rodzaju połączenia do bazy danych; polecane jest wykorzystanie sterownika Mysqli (ulepszone rozszerzenie) lub Mysqlnd (naturalny sterownik). Pierwszy z wymienionych stanowi niejako nową, ulepszoną wersję oryginalnego sterownika Mysql;jest dostępny od 5 wersji PHP w postaci łatwej do obsługi klasy. Sterownik ten rozwijany i dostarczany jest przez firmę Oracle.
Drugi ze wspomnianych sterowników cechuje jeszcze większa wydajność i efektywność. Dostarczany od wersji PHP 5.3.0 jako dynamiczna biblioteka w całości pisana w C. Dostęp do niej możliwy jest poprzez interfejs PDO (PHP Data Object). Sterownik nie działa dla wersji starszych od PHP 5 (brak wsparcia w jądrze parsera) oraz dla bazy danych MySQL starszej niż 4.1.
Na zajęciach omówiony zostanie właśnie interfejs PDO; jeżeli ktokolwiek byłby zainteresowany używaniem sterownika Mysqli to opis jego klasy oraz dostępnych metod znajduje się pod adresem http://www.php.net/manual/en/class.mysqli.php .
Niewątpliwą zaletą PDO jest jego „niezależność" względem wybranej bazy danych. Może być użyty zarówno dla baz MySQL jak i MsSQL, PostgreSQL czy SQLite (a także baz nie wchodzących w standard SQL).

Połączenie z bazą danych rozpoczyna się od powołania do życia obiektu klasy PDO. Konstruktor klasy wygląda następująco:

public PDO::__construct() (string $dsn, string $username, string $password, array $driver_options)

poszczególne parametry:
1) $dsn – skrót od Data Source Name (nazwa źródła danych). Zawiera informacje niezbędne do połączenia się z określoną bazą danych. W przypadku MySQL zmienna $dsn może wyglądać w następujący sposób:
'mysql:host=localhost;port=3306;dbname=nazwabazy'

Pierwszy człon ciągu, 'mysql:' określa jaki sterownik ma zostać wykorzystany – w naszym

wypadku MySQL; po dwukropku podaje się szczegóły połączenia – parametr 'host' powinien zawierać adres pod którym działa nasz serwer (zarówno dla WAMP, jak i znaczącej większości serwerów hostujących strony WWW wartość dla tej zmiennej będzie miała wartość 'localhost'). Opcjonalnie można podać port, na którym działa serwer bazy (podany w przykładzie to domyślny port, pod którym baza nasłuchuje połączeń; parametr należy podać w przypadku, gdy numer protu będzie inny niż domyślny, np. 5555). Obowiązkowym parametrem jest natomiast dbname; musi on przyjąć nazwę bazy danych, na której zamierzamy przeprowadzać operacje SQL.
Ponieważ najlepszym rozwiązaniem jest przechowywanie ciągów znakowych przy użyciu kodowania UTF-8 trzeba bazę o tym fakcie poinformować. W tym wypadku do całego ciągu należy dodać jeszcze jeden parametr o nazwie charset z wartością utf8. W związku z powyższym nasz ciąg $dsn powinien wyglądać:

'mysql:host=localhost;port=3306;dbname=nazwabazy;charset=utf8'

Trzeba jednak pamiętać, że do wersji PHP 5.3.6 parametr charset, pomimo ustawienia, nie będzie brany pod uwagę; przed podaną wersją kodowanie ustawia się poprzez dodatkowy parametr opisany pod parametrem konstruktora $driver_options.

2) $username – parametr opcjonalny; powinien zawierać nazwę użytkownika bazy danych, na której chcemy operować (w naszym wypadku użytkownik nazywa się 'root').

3) $password – parametr opcjonalny; powinien zawierać hasło użytkownika; w naszym wypadku należy podać '' (pusty ciąg znakowy) jeżeli nie ustawiliśmy hasła dla użytkownika root

4) $driver_option – parametr opcjonalny; musi być podany jako tablica wedle schematu klucz=>wartość; jeżeli nasz parser PHP jest starszy od wersji 5.3.6 (np. 5.2.0, 5.3.0 itp.) to właśnie pod tym parametrem musimy podać bazie danych informacje o kodowaniu znaków jakiego chcemy używać; działa także dla najnowszych wersji PHP. Jednak takie ustawienie kodowania znaków działa tylko wtedy gdy zestaw znakowy używany w pliku skryptu współdzieli te same niższe 7 bitów jak w ASCII (np. ISO-8869-1 lub samo UTF8).

Przykład użycia parametru:
$driver_option = array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8');

Przykład inicjalizacji obiektu odpowiedzialnego za połączenie z bazą danych (dla wersji starszej niż 5.3.6):

```
$dbc = new PDO('mysql:host=localhost;port=3306;dbname=nazwabazy',
                        'root',
                        '',
                        array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8'));
```

Dla wersji 5.3.6 lub starszej:

```
$dbc = new PDO('mysql:host=localhost;port=3306;dbname=nazwabazy;charset=utf8',
                        'root',
                        '');
```

Tak utworzone połączenie istnieje do czasu życia obiektu $dbc; jest automatycznie zamykane wraz ze skasowaniem obiektu/zakończeniem działania skryptu wywołującego połączenie.

Jeżeli nie jesteśmy pewni jakie możliwe sterowniki mamy do wykorzystania dla klasy PDO można

się tego dowiedzieć dzięki funkcji

public static array PDO::getAvailableDrivers (void)

funkcja jest statyczna, a w związku z powyższym nie trzeba tworzyć obiektu w celu jej wywołania. Aby wyświetlić dostępne sterowniki dla klasy PDO można użyć następującej linii kodu:

print_r(PDO::getAvailableDrivers());

Mając ustanowione połączenie z bazą danych możemy rozpocząć na niej operować. Stworzony obiekt oferuje dwie metody przesyłania zapytań SQL (dwiema osobnymi metodami):

1) public int PDO::exec (string $statement) - metoda wykonuje przekazane polecenie SQL przekazane poprzez zmienną $statement i zwraca liczbę odpowiadającą ilości wierszy, na które wpływ miało wykonane polecenie. Metoda ta NIE ZWRACA wyników zapytania, stąd też wykonywanie poprzez nią zapytań SELECT nie da pożądanego rezultatu; przykłady wywołania metody:

$rowCount = $dbc->exec(„CREATE TABLE IF NOT EXISTS `tabela` (`id` INT(11) AUTO_INCREMENT, `nazwa` VARCHAR(50), `opis` TEXT, PRIMARY KEY (`id`));”);

$rowCount = $dbc->exec(„INSERT INTO `tabela` (`nazwa`, `opis`) VALUES ('przykładowy przedmiot', 'przykładowy opis');”);

$rowCount = $dbc->exec(„UPDATE `tabela` SET `nazwa` = 'nowy przedmiot' WHERE `id` = 1;”);

$rowCount = $dbc->exec(„DELETE FROM `tabela` WHERE `id` = 2”);

$rowCount = $dbc->exec(„DROP TABLE IF EXISTS `tabela`;”);

2) druga metoda pozwala nam zarówno przesyłać zapytania wymienione powyżej (w przykładzie użycia PDO::exec()) jak i zapytania SELECT, z których w tym wypadku otrzymamy interesujące nas wyniki. Zapisywane są w specjalnie tworzonym obiekcie przez prezentowane poniżej metody; obiekt tworzony jest na podstawie predefiniowanej klasy PDOStatement (jej opis nastąpi później).

public PDOStatement PDO::query (string $statement) - przekazuje do wykonania polecenie SQL spod zmiennej $statement; zwraca obiekt klasy PDOStatement, pod którym zapisywane są wyniki zapytania.

public PDOStatement PDO::query (string $statement , int $PDO::FETCH_COLUMN , int $colno) - przeciążenie funkcji; stała **`PDO::FETCH_COLUMN`** określa zachowanie metody fetch – ma ona w tym wypadku zwracać wartość z pojedynczej zażądanej kolumny przy pobraniu następnego wiersza wyników. Parametr $colno określa numer kolumny, której wartość ma zostać zwrócona

public PDOStatement PDO::query (string $statement , int $PDO::FETCH_CLASS , string $classname , array $ctorargs) - przeciążenie funkcji; stała **`PDO::FETCH_CLASS`** określa, że metoda fetch ma zwracać nową instancję (byt) wskazanej klasy (jej nazwa określona poprzez zmienną $classname) mapując (dopasowując) nazwy pobranych kolumn do pól pod identyczną nazwą (jeżeli dane pole nie istnieje to wywoływana jest magiczna metoda __set() ustanawiająca nowe pole). Pod parametr $ctorargs można wstawić w postaci tablicy wartości parametrów konstruktora.

public PDOStatement PDO::query (string $statement , int $PDO::FETCH_INTO , object $object)

- przeciążenie funkcji; stała **PDO::FETCH_INTO** określa, że metoda fetch powinna aktualizować aktualnie utworzony już obiekt wynikami z zapytania; nazwy kolumn są mapowane do pól klasy (nazwa kolumny <=> nazwa pola). Parametr $object określa istniejący już obiekt w skrypcie (musi być wcześniej utworzony).

Przykładowe użycie:

```
foreach($dbc->query(„SELECT `nazwa`, `opis` FROM `tabela`;”) as $row) {
        echo $row['nazwa'] . „ „ . $row['opis'] . „\n”;
}
```

Istnieje jeszcze jedna możliwość utworzenia zapytania do bazy danych przy pomocy PDO. Jednak przed zaprezentowaniem tego sposobu dobrze jest zapoznać się z klasą, która stanowi podstawę obiektu do którego będą przypisywane wyniki zapytania.
Obiekt klasy PDOStatement tworzony jest automatycznie przez metodę PDO:query() lub PDO::prepare(). W pierwszym przypadku utworzony obiekt automatycznie zawiera wynik wykonanego zapytania. W drugim wypadku obiekt otrzymuje wzorzec zapytania, który następnie może zostać wywołany poprzez odpowiednią metodę.
Najpierw warto zapoznać się z ostatnią funkcją do budowania zapytań klasy PDO:

public PDOStatement PDO::prepare (string $statement , array $driver_options = array()) -

- pod parametr $statement wstawia się wzorzec zapytania SQL; dzięki niemu nie trzeba każdorazowo pisać od nowa całego zapytania w przypadku różnych podawanych warunków

- $driver_option jest parametrem opcjonalnym (można go pominąć); głównym zastosowaniem parametru jest ustawienie parametru PDO::ATTR_CURSOR. Domyślnie ustawiony jest na PDO::CURSOR_FWDONLY (przepływ danych z zapytania od pierwszego do ostatniego rekordu danych; domyślna i najszybsza metoda czytania wyników) jednak można go zmienić na PDO::CURSOR_SCROLL (umożliwia swobodne przeglądanie zwróconych rekordów; nie każdy sterownik SQL obsługuje ten tryb).

Przykład użycia:

```
$pdostm = $dbc->prepare(„SELECT * FROM `tabela` WHERE `id` = :id OR `nazwa` = :nazwa;”);
```

Od tego momentu możemy wykorzystywać obiekt PDOStatement, zapisany pod zmienną $pdostm. Pierwszym krokiem powinno być użycie jednej z dwóch dostępnych metod przypisujących wartości do tworzonego szablonu:

1) public bool PDOStatement::bindParam (mixed $parameter , mixed &$variable, int $data_type = PDO::PARAM_STR, int $length, mixed $driver_options)

parametr $parameter stanowi identyfikator parametru. W przedstawionym wyżej przykładzie zmienna mogłaby przyjąć wartość ':id' lub ':nazwa'; oznacza to, że wartość tej zmiennej musi znajdować się w szablonie i rozpoczynać od dwukropka (może to być dowolna nazwa poprzedzona dwukropkiem); istnieje jeszcze możliwość by zamiast nazwy podstawiać wartości liczbowe (kolejny numer zmiennej w utworzonym szablonie) jednak to w przykładzie poniżej;

zmienna &$variable ma przyjąć wartość jaką chcemy podstawić w naszym szablonie za wskazaną nazwę; w naszym wypadku mogłaby to być np. wartość 1 dla ':id'

$data_type to opcjonalny parametr określający jaki typ wartości chcemy przekazać do naszego zapytania; określany jest poprzez stałe klasy PDO, wedle schematu PDO::PARAM_* (dostępne wartości to PDO::PARAM_INT, PDO::PARAM_BOOL, PDO::PARAM_STR, PDO::PARAM_ULL, PDO::PARAM_LOB, PDO::PARAM_STMT lub PDO::PARAM_INPUT_OUTPUT; dokładny opis tych stałych znajduje się pod adresem http://www.php.net/manual/en/pdo.constants.php). Proszę zwrócić szczególnie uwagę na stałą PDO::PARAM_INPUT_OUTPUT – jest ona dostępna niemal tylko dla procedur/funkcji SQL; pozwala nam na zwrócenie wartości, jaka zostanie przekazana przez procedurę/funkcję pod podany parametr tejże (przykładowo mamy procedurę zwroc_cos(parametr) w bazie danych; jeżeli podstawimy pod nią odpowiednią zmienna z szablonu, której nadamy cechy wejścia/wyjścia to pod tą naszą zmienną zostanie zwrócona wartość jaka zwrócona zostałaby w SQL).

Opcjonalny parametr $length może określić długość przekazywanej zmiennej (jej wielkość). Obligatoryjne ustawienie tego parametru wymagane jest jedynie w przypadku gdy chcemy, aby nasza zmienna wejściowa była także wyjściem (czyli chcemy odebrać dane z procedury/funkcji SQL).

$driver_option jest zupełnie opcjonalny i ma dokładnie to samo zastosowanie jak w metodzie PDOStatement::prepare()

2) public bool PDOStatement::bindValue (mixed $parameter , mixed $value, int $data_type = PDO::PARAM_STR) -

parametr $parameter ma to samo zastosowanie co w metodzie 1)

parametr $value ma to samo zastosowanie co w metodzie 1) z tą różnicą, że nie stanowi on aliasu (nie może być do niego nic zwrócone)

trzeci, opcjonalny parametr $data_type ma to samo zastosowanie co w metodzie 1) z tą różnicą, że nie można przypisać mu stałej PDO::PARAM_INPUT_OUTPUT.

Przez przykładem dobrze jest zapoznać się z definicją jeszcze jednej metody, wywołującej nasze zapytanie (do tej pory było ono tylko tworzone):

public bool PDOStatement::execute (array $input_parameters) - funkcja wykonuje wcześniej przygotowane zapytanie SQL. Jako zupełnie opcjonalny parametr wejściowy możemy podać tablicę przypisywanych parametrów do zapytania (jeżeli nie użyliśmy żadnej z powyżej opisywanych metod). Parametry te automatycznie będą tylko wejściowe i będą traktowane jako ciągi znakowe (stała PDO::PARAM_STR).

Przykłady użycia (dla wcześniej stworzonego szablonu):

```
$name = 'operator';
$pdostm->bindValue(':id', 1, PDO::PARAM_INT);
$pdostm->bindValue(':nazwa', $name, PDO::PARAM_STR);
$pdostm->execute();

$name = 'liniowy';
$pdostm->bindValue(':id', 1, PDO::PARAM_INT);
$pdostm->bindValue(':nazwa', $name, PDO::PARAM_STR);
$pdostm->execute();
```

```
$pdostm->bindValue(':id', 5, PDO::PARAM_INT);
$pdostm->bindValue(':nazwa', $name, PDO::PARAM_STR);
$pdostm->execute();
```

Przykład 2 (dla nowego szablonu)

```
$pdostm = $dbc->prepare(„SELECT * FROM `tabela` WHERE `id` = ? OR `nazwa` = ?;”);
```

Proszę zwrócić uwagę, że szablon zamiast zawierać nazwy parametrów posiada znaki zapytania. W tym wypadku używamy liczbowego określenia o który parametr będzie nam chodziło (licząc od 1).

```
$name = 'operator';
$pdostm->bindValue(1, 1, PDO::PARAM_INT);
$pdostm->bindValue(2, $name, PDO::PARAM_STR);
$pdostm->execute();

$name = 'liniowy';
$pdostm->bindValue(1, 1, PDO::PARAM_INT);
$pdostm->bindValue(2, $name, PDO::PARAM_STR);
$pdostm->execute();

$pdostm->bindValue(1, 5, PDO::PARAM_INT);
$pdostm->bindValue(2, $name, PDO::PARAM_STR);
$pdostm->execute();
```

Przykład 3 (gdy chcemy uzyskać wartość z parametru procedury):

Najpierw stwórzmy w bazie danych odpowiednią procedurę (np. wpisując ją w zapytaniu SQL poprzez phpMyAdmin):

```
DROP PROCEDURE IF EXISTS testP;
delimiter ;;

CREATE PROCEDURE testP(INOUT wartosc INT)
BEGIN
        SET wartosc = wartosc + 5;
END;;

delimiter ;
```

Proszę od razu przetestować działanie procedury takim oto kodem (okno zapytań MySQL):

```
SET @param = 10;
CALL testP(@param);

SELECT @param;
```

jak widać wyświetlona nam zostanie wartość 15 (10 + 5).
Teraz można sprawdzić jak zadziała nam procedura poprzez wywołanie w PHP:

```
$pdostm = $dbc->prepare(„CALL testP(?);”);
```

```
$retVal = 20;
$pdostm->bindParam(1, $retVal, PDO::PARAM_INT | PDO::PARAM_INPUT_OUTPUT, 12);
$pdostm->execute();

echo $retVal;
```

Proszę zauważyć, że 3 parametr zawiera dwie stałe – pierwsza określa typ, a druga cechę podanej zmiennej; tak naprawdę jest to jedna wartość bitowa, a operator | ustawia odpowiednio jej bity (wykonuje binarne lub z wartości obu zmiennych).

PDOStatement zawiera także dwie metody dostępowe do zwróconych wyników przez SQL.

1) public mixed PDOStatement::fetch (int $fetch_style, int $cursor_orientation = PDO::FETCH_ORI_NEXT, int $cursor_offset = 0) – metoda zwraca kolejny (pojedynczy) zwrócony przez SQL rekord danych spośród wszystkich dostępnych.

wszystkie parametry tej funkcji są OPCJONALNE (możemy ją wywołać po prostu jako fetch())

wartość parametru $fetch_style określa jak będzie wyglądać struktura zwróconego rekordu (zmienna przyjmuje wartość jednej z predefiniowanych stałych PDO::FETCH_*); domyślnie parametr ma ustawioną wartość PDO::FETCH_BOTH, przez co wyniki zwracane są w tablicy zawierającą zarówno indeksy liczbowe (każda kolejna kolumna z polecenia SELECT zostaje ponumerowana, zaczynając od zera) jak i indeksy z nazwami kolumn; pozostałe opcje można poznać pod adresem http://www.php.net/manual/en/pdostatement.fetch.php (możliwe kombinacje: PDO::FETCH_ASSOC, PDO::FETCH_BOUND, PDO::FETCH_CLASS, PDO::FETCH_INTO, PDO::FETCH_LAZY, PDO::FETCH_NUM, PDO::FETCH_OBJ).

Parametr $cursor_orientation określa który rekord metoda ma zwrócić (definiowane przez stałe; domyślnie PDO::FETCH_ORI_NEXT). Możliwe kombinacje: PDO::FETCH_ORI_PRIOR, PDO::FETCH_ORI_FIRST, PDO::FETCH_ORI_LAST, PDO::FETCH_ORI_ABS, PDO::FETCH_ORI_REL.

$cursor_offset działa tylko w przypadku gdy wybraliśmy w $cursor_orientation PDO::FETCH_ORI_ABS lub PDO::FETCH_ORI_REL. W pierwszym wypadku metoda zwróci nam dokładnie ten rekord, który wskażemy poprzez opisywany parametr (np. 5 lub 7); w drugim wypadku jeżeli odczytywaliśmy np. pozycję 5, a w parametrze wstawimy wartość 3 to metoda zwróci nam rekord 8 (przesunięcie nastąpi relatywnie względem aktualnej pozycji w wynikach).

2) public array PDOStatement::fetchAll (int $fetch_style, mixed $fetch_argument, array $ctor_args = array()) - funkcja, w przeciwieństwie do poprzedniej, zwraca naraz wszystkie wyniki zadanego zapytania do zmiennej tablicowej. Tak jak poprzedniczka może być wywoływana bez żadnego parametru.

$fetch_style – parametr jak w metodzie 1) określa w jaki sposób będzie zbudowana struktura (tablica) zwróconych wyników. Domyślnie tworzy się identyczna struktura jak w poprzedniej metodzie (stała **`PDO::ATTR_DEFAULT_FETCH_MODE`** będąca aliasem **`PDO::FETCH_BOTH`**). Inne możliwe kombinacje: PDO::FETCH_COLUMN (tworzy klucze tylko z nazw kolumn), PDO::FETCH_UNIQUE (musi wystąpić wraz z PDO::FETCH_COLUMN – wtedy wszystkie wyniki zostaną spakowane pod pojedyncze klucze) lub PDO::FETCH_GROUP (musi wystąpić wraz z PDO::FETCH_COLUMN – wyniki zostaną pogrupowane w tablice asocjacyjne).

$fetch_argument – będzie posiadał różne znaczenie w zależności od poprzednio ustawionego argumentu:

PDO::FETCH_COLUMN – można określić, poprzez podanie liczby całkowitej, którą kolumnę chcemy pobrać (np. 0 pobierze nam tylko pierwszą kolumnę wyniku, pomijając kolejne)
PDO::FETCH_CLASS – tworzy kolejne instancje (byty) klas i mapuje je pola z nazwami kolumn (jako wartość parametru podaje się nazwę klasy do której chcemy mapować wyniki)
PDO::FETCH_FUNC – zwraca wyniki do parametrów wskazanej przez ten parametr funkcji, mapując wedle zasady kolejna kolumna → kolejny parametr funkcji.

$ctor_args – można przez niego przekazywać (w postaci tablicowej) parametry konstruktora funkcji, o ile wybrano $fetch_style = PDO::FETCH_CLASS

Zadanie: proszę wypróbować działanie przedstawionych metod – proszę sprawdzić jak zwracają wyniki, jak zmiana poszczególnych parametrów wpływa na wyświetlenie wyników itd. Podpowiedź dla PDOStatement::fetch() - aby wyświetlić wszystkie zwrócone rekordy za pomocą tej funkcji należy spróbować takiego przykładowego kodu:

```
while ($row = $pdostmt->fetch() ) {
        echo $row[0];
}
```

Klasa PDO posiada jeszcze 3 przydatne metody:

public bool PDO::beginTransaction (void) - metoda wyłącza automatyczne akceptowanie każdej transakcji; od czasu jej wydania ŻADNE zapytanie SQL nie będzie zatwierdzane na bazie danych

public bool PDO::commit (void) - metoda akceptuje wszystkie wydane zapytania bazy SQL. Wywołanie tej metody włącza automatycznie akceptowanie każdej transakcji.

public bool PDO::rollBack (void) - metoda anuluje wszystkie wydane zapytania do bazy SQL jeżeli ich wywołanie nastąpiło po wcześniej wywołanej metodzie PDO::beginTransaction(). Wywołanie tej metody włącza automatyczne akceptowanie każdej kolejnej transakcji.

Zadanie:

Proszę wykorzystać nasz rozwijany przykład Osoba-Funkcja. Należy zapisywać dodawane informacje przez utworzony formularz do bazy danych. Następnie należy je stamtąd pobierać do wyświetlenia. Należy do klasy Osoba dodać pole hasło. Jeżeli dana osoba będzie chciała się usunąć z bazy danych musi podać hasło (proste logowanie). Proszę obsłużyć także potencjalną funkcję zmiany danych (np. nazwiska, funkcji, hasła). Hasło proszę spróbować zakodować (zahaszować) funkcją md5($str), gdzie $str to hasło do zakodowania (szczegółowy opis funkcji http://pl1.php.net/manual/en/function.md5.php).