Tworzenie i usuwanie bazy

Najpierw zaprojektujemy prostą bazę danych ***ksiegarnia_internetowa***, aby potem ją utworzyć, wypełnić danymi i wykonywać operacje na danych. Posłużymy się przykładem zaprezentowanym w książce *PHP i MySQL tworzenie stron WWW* autorstwa Luke Welling oraz Laury Thomson.

Identyfikatory MySQL

Podsumujmy zasady nazewnictwa, które będziemy stosować podczas pracy z bazą MySQL.

Nazwa obiektu bazy danych to jego identyfikator. W MySQL mamy takie identyfikatory jak **bazy danych, tabele, kolumny i indeksy oraz aliasy**. O indeksach i aliasach dowiesz się w dalszym toku nauki. Tworząc nazwy będziemy stosować następujące zasady:

- Używamy liter a-z oraz A-Z, bez "polskich ogonków", a także cyfr 0-9. Co prawda od wersji 3.23.6 serwera MySQL, można stosować standard Unicode zawierający znaki różnych języków, ale będziemy w tym zakresie zgodni ze standardami obowiązującymi w językach programowania.
- Pierwszym znakiem nie może być cyfra.
- Zamiast spacji będziemy uzywaćpodkreślnika np. `nazwa_tabeli`
- Należy stosować nazwy czytelne, oddające rzeczywistość np. `recenzje_ksiazek`, a nie np. `ksiazki_2`.
- Serwer baz danych rozróżnia wielkie i małe litery, jeżeli robi to system operacyjny. Unix jest system który rozróżnia wielkość liter. Będziemy jednak stosować nazewnictwo uniwersalne, które sprawdzi się w każdej sytuacji - wystarczy stosować nazwy które są unikatowe bez względu na wielkość liter.
- Na koniec, musimy wziąć pod uwagę fakt, że w PHP przyjęto stosowanie małych liter. W dalszym toku nauki będziemy administrować MySQL z poziomu PHP, więc powinniśmy dla wygody ujednolicić stosowane przez nas nazewnictwo. Dlatego umówmy się, że podczas pracy z MySQL, w pisowni identyfikatorów jak również w nazwach plików i folderów będziemy stosować małe litery, zgodnie z omówionymi wcześniej zasadami.

Polecenia SQL będziemy pisać wielkimi literami.

Schemat bazy *ksiegarnia_internetowa*

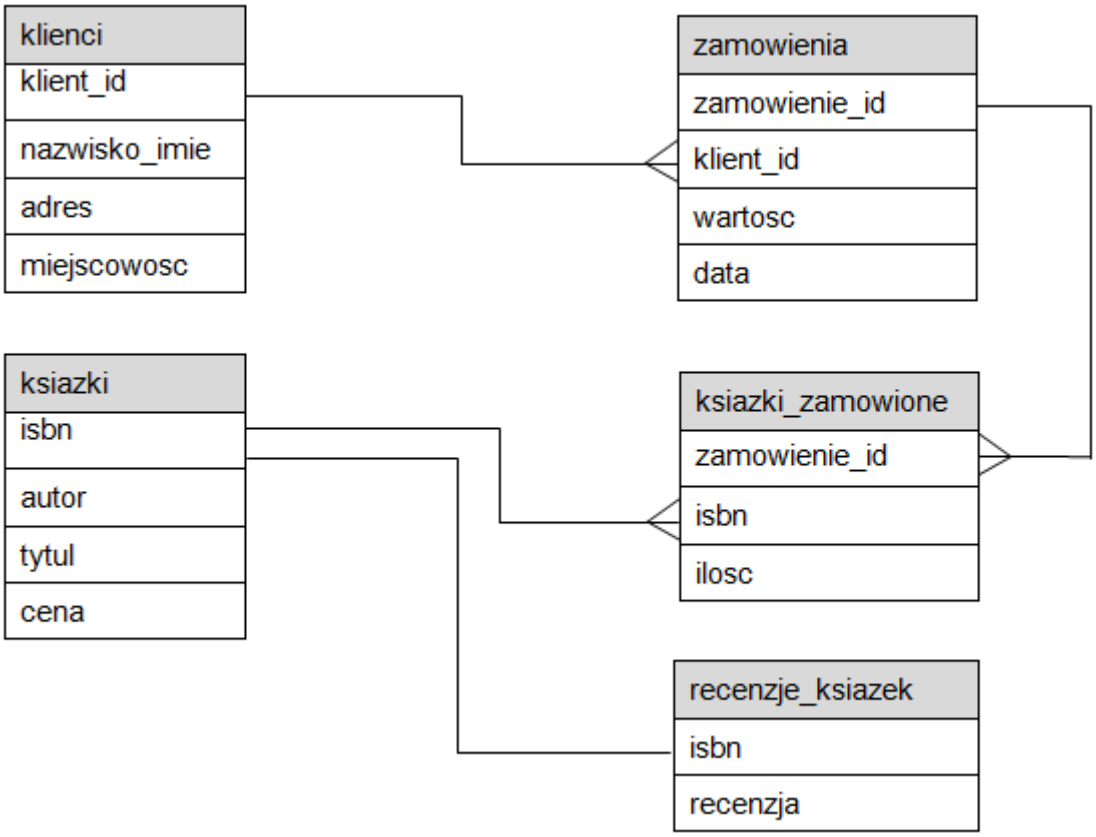
Nasza baza będzie rejestrować sprzedaż książek w księgarni internetowej. W tym celu będą utworzone następujące tabele:

- klienci(klient_id, nazwisko_imie, adres, miejscowosc)
- zamowienia(zamowienie_id, klient_id, wartosc, data)
- ksiazki(isbn, autor, tytul, cena)
- ksiazki_zamowione(zamowienie_id, isbn, ilosc)
- recenzje_ksiazek(isbn, recenzja)

Wyjaśnijmy, że `isbn` to unikatowy numer książki ISBN, możesz go znaleźć na początku książki - w miejscu gdzie podawane są inne informacje, jak wydawnictwo, rok wydania, informacja o prawach autorskich, itp. Tak więc mamy gotowy klucz główny w tabeli `ksiazki`. Tabela `ksiazki_zamowione` nie posiada własnego pola identyfikującego, ponieważ `zamowienia_id` oraz `isbn`, brane pod uwagę łącznie - zapewniają unikatowość każdego rekordu, pod warunkiem, że każda zamówiona książka będzie wpisywana w jednym wierszu, niezależnie od tego ile egzemplarzy zamówiono. Dlatego tabela ta posiada kolumnę `ilosc`. Zakładamy, że tylko do części książek została napisana recenzja (tylko jedna recenzja danej książki), dlatego została utworzona oddzielna tabela `recenzje_ksiazek`, zawierająca istniejące recezje. Dzięki tej tabeli unikamy pustych pól, które wystąpiłyby, gdyby kolumna `recenzje` znajdowała się w tabeli `ksiazki`.

Diagram tabel i relacji bazy *ksiegarnia_internetowa*

Poniższy diagram pokazuje układ tabel i relacji. Na tej podstawie możemy przeanalizować przepływ danych między poszczególnymi tabelami bazy.



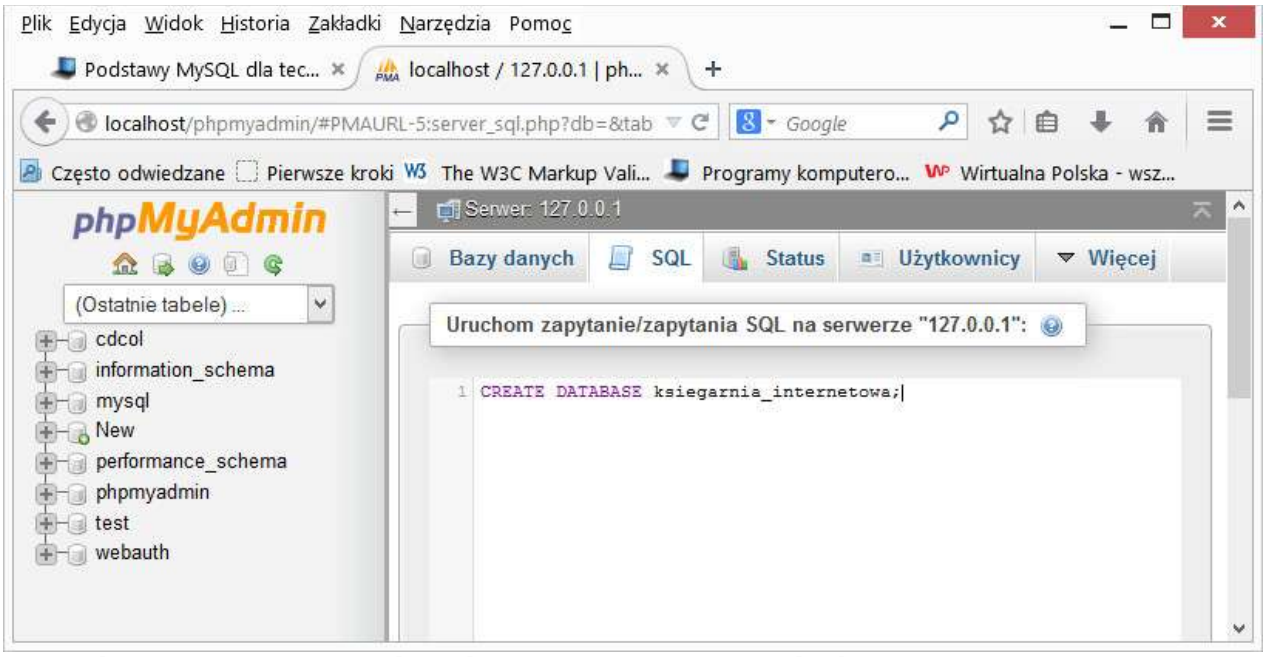
Rysunek 3_1_0_1. Diagram tabel i relacji bazy danych *ksiegarnia_internetowa*

Utworzenie bazy danych *ksiegarnia_internetowa*

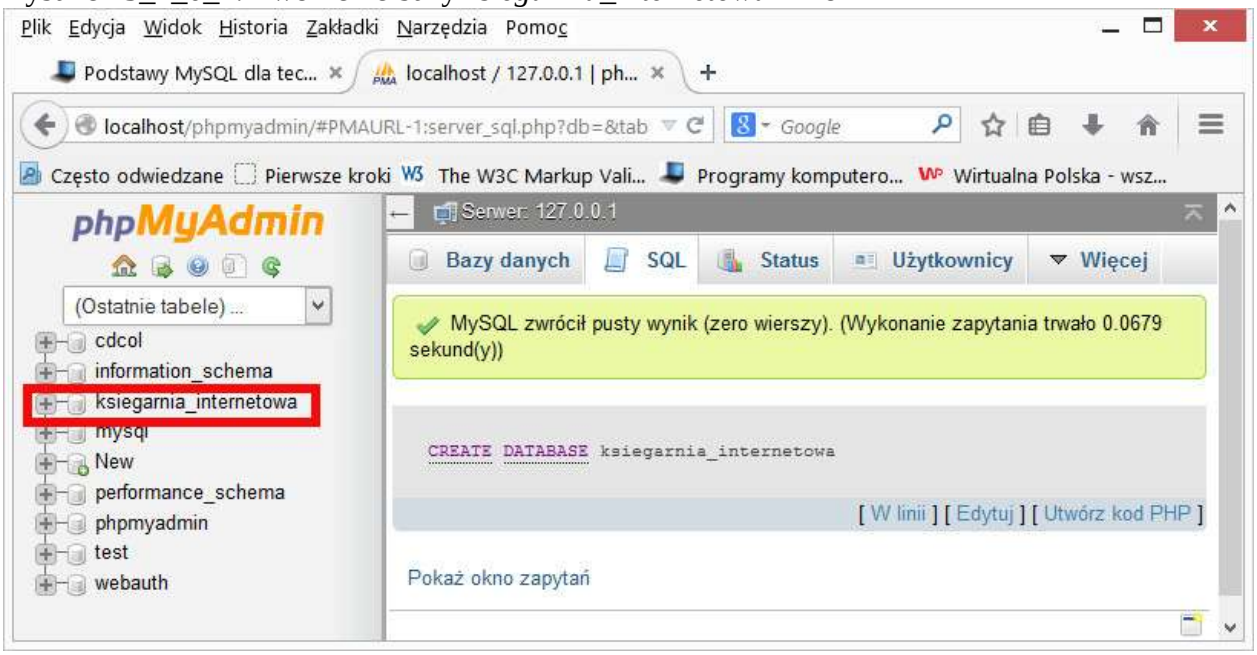
Na tych zajęciach uczynimy pierwszy, bardzo prosty krok w procesie tworzenia bazy. Utworzymy naszą bazę danych o nazwie `ksiegarnia_internetowa`. W tym celu należy otworzyć phpMyAdmina - zakładka SQL, a następnie:

1. Wpisać `CREATE DATABASE ksiegarnia_internetowa;` i kliknąć *Wykonaj*, przy czym zastosujemy bardziej rozbudowane polecenie `CREATE DATABASE ksiegarnia_internetowa DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;`, które zagwaranuje nam poprawne wyświetlanie polskich liter.
2. Powinien pojawić się komunikat o wykonaniu polecenia, a nasza baza powinna zostać dołączona do listy baz.

`CREATE DATABASE ksiegarnia_internetowa;` to instrukcja napisana w języku **SQL**. Proszę zwrócić uwagę na składnię `CREATE DATABASE` - jest pisana dużymi literami, taką właśnie konwencję będziemy stosować. Co prawda MySQL nie rozróżnia wielkich i małych liter w poleceniach SQL, ale taką pisownię będziemy stosować w "ręcznym" administrowaniu bazą.



Rysunek 3_1_0_2. Tworzenie bazy *ksiegarnia_internetowa* - krok 1



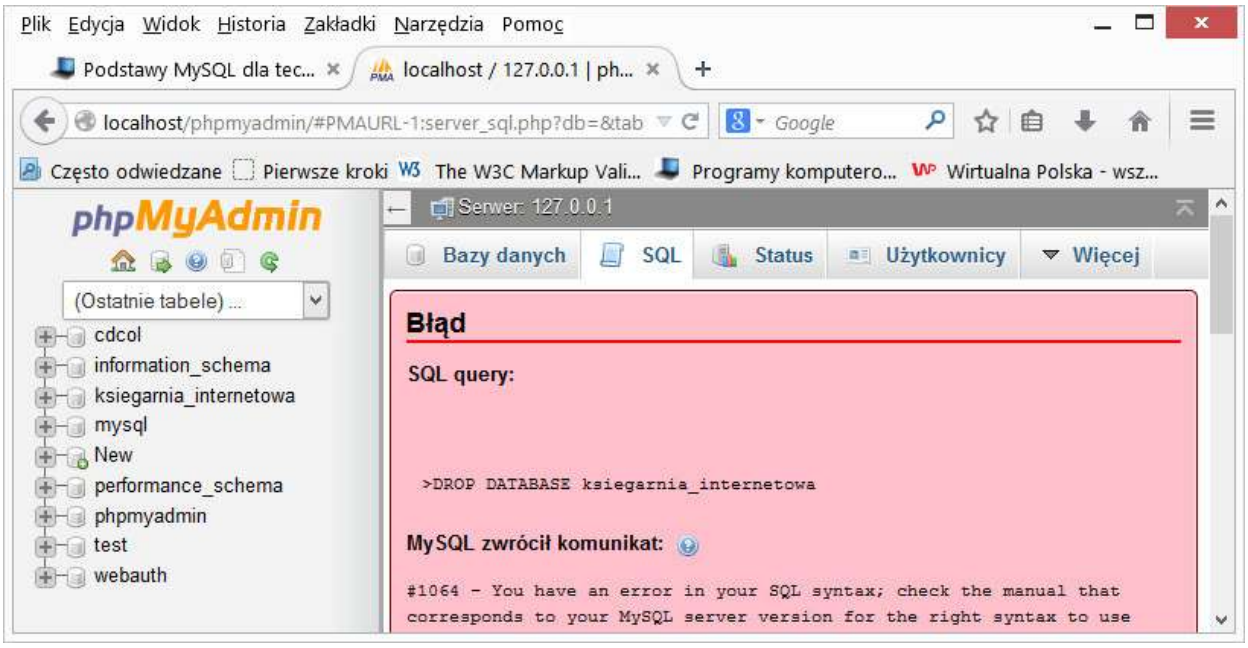
Rysunek 3_1_0_3. Tworzenie bazy *ksiegarnia_internetowa* - krok 2

Usunięcie bazy danych *ksiegarnia_internetowa*

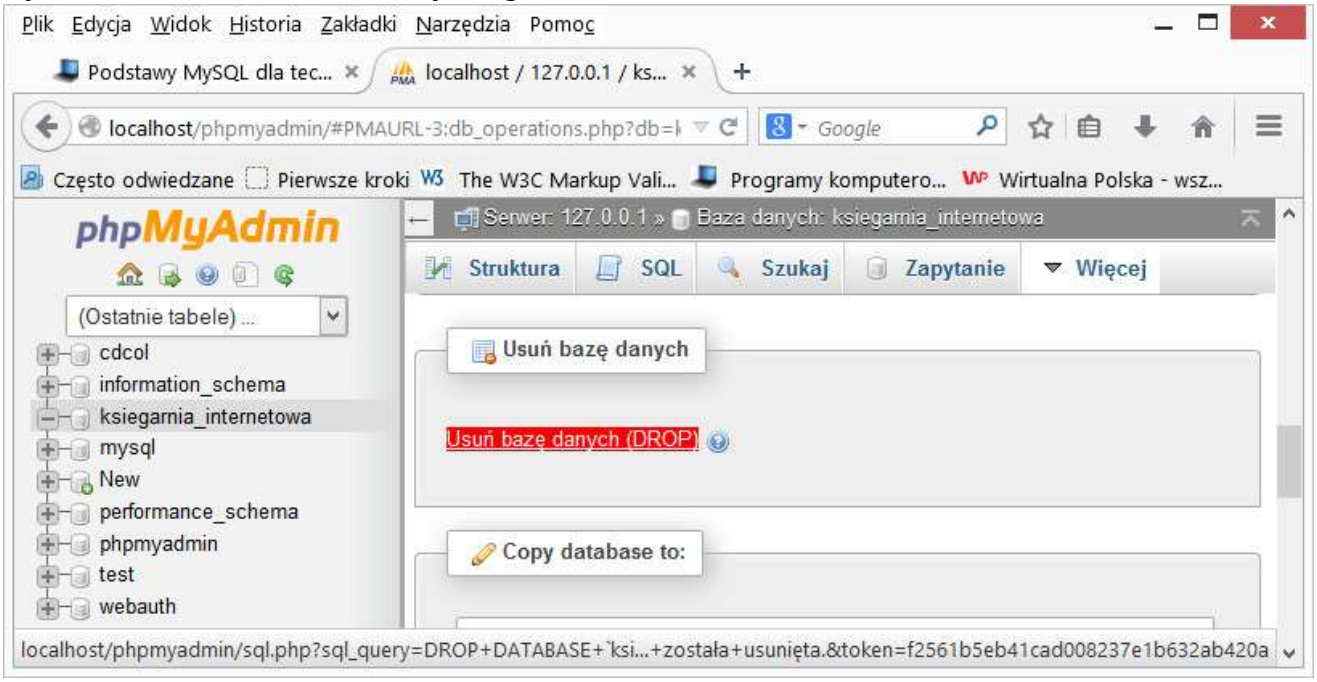
Teraz możemy spokojnie przećwiczyć usuwanie bazy danych, ponieważ nic ona jeszcze nie zawiera. **Po usunięciu bazy, należy ją ponownie utworzyć**, aby na następnych zajęciach z niej korzystać.

1. Polecenie `DROP DATABASE ksiegarnia_internetowa;` powinno bazę usunąć.
2. Jeżeli pojawi się komunikat, że jest to niemożliwe, wtedy klikamy nazwę naszej bazy na liście baz (po lewej).
3. Otwieramy zakładkę *Operacje*, a następnie *Usuń bazę danych*.

Opisane czynności usuwania bazy pokazują zrzuty ekranu:



Rysunek 3_1_0_4. Usuwanie bazy *ksiegarnia_internetowa* - krok 1 i 2



Rysunek 3_1_0_5. Usuwanie bazy *ksiegarnia_internetowa* - krok 3

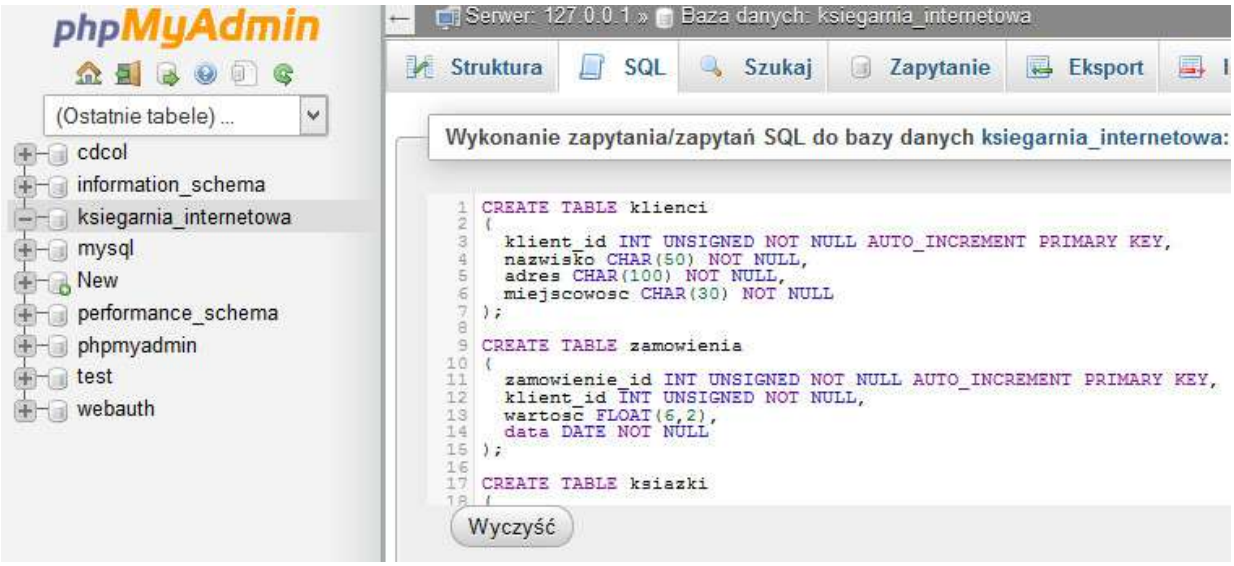
Tworzenie tabel

Podczas tych zajęć utworzymy tabele naszej przykładowej bazy danych **ksiegarnia_internetowa**. Ponieważ nasza baza zaczyna przyjmować realny kształt, nauczymy się także eksportować ją do pliku, aby nie utracić efektów naszej pracy.

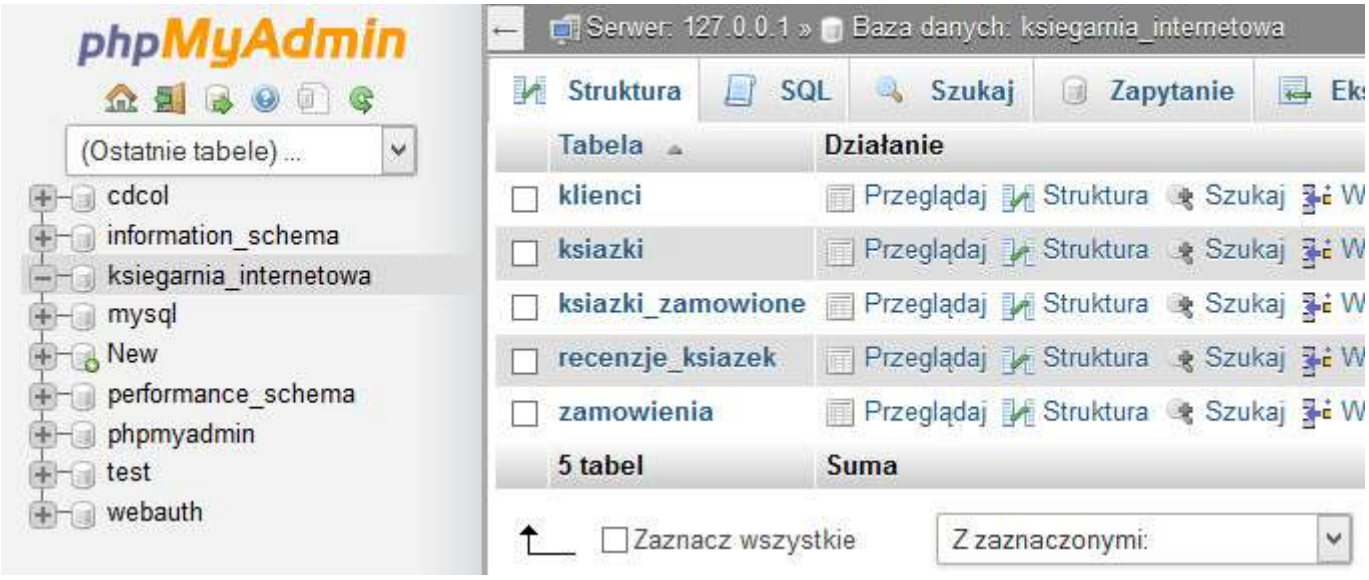
Ćwiczenie 3_4_0_1. Tworzenie tabel bazy *ksiegarnia_internetowa*

Uruchom *phpMyAdmina*, otwórz bazę *ksiegarnia_internetowa*, napisz poniższy kod SQL tworzący tabele i wciśnij *Wykonaj*.

```
CREATE TABLE klienci
(
klient_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  nazwisko CHAR(50) NOT NULL,
  adres CHAR(100) NOT NULL,
miejscowosc CHAR(30) NOT NULL
);

CREATE TABLE zamowienia
(
zamowienie_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
klient_id INT UNSIGNED NOT NULL,
wartosc FLOAT(6,2),
  data DATE NOT NULL
);

CREATE TABLE ksiazki
(
isbn CHAR(13) NOT NULL PRIMARY KEY,
  autor  CHAR(50),
tytul CHAR(100),
  cena FLOAT(4,2)
);


CREATE TABLE ksiazki_zamowione
(
zamowienie_id INT UNSIGNED NOT NULL,
isbn CHAR(13) NOT NULL,
ilosc TINYINT UNSIGNED,
  PRIMARY KEY(zamowienie_id, isbn)
);

CREATE TABLE recenzje_ksiazek
(
isbn CHAR(13) NOT NULL PRIMARY KEY,
  recenzja TEXT
);
```



Rysunek 3_4_0_1a. Tworzenie tabel bazy *ksiegarnia_internetowa*

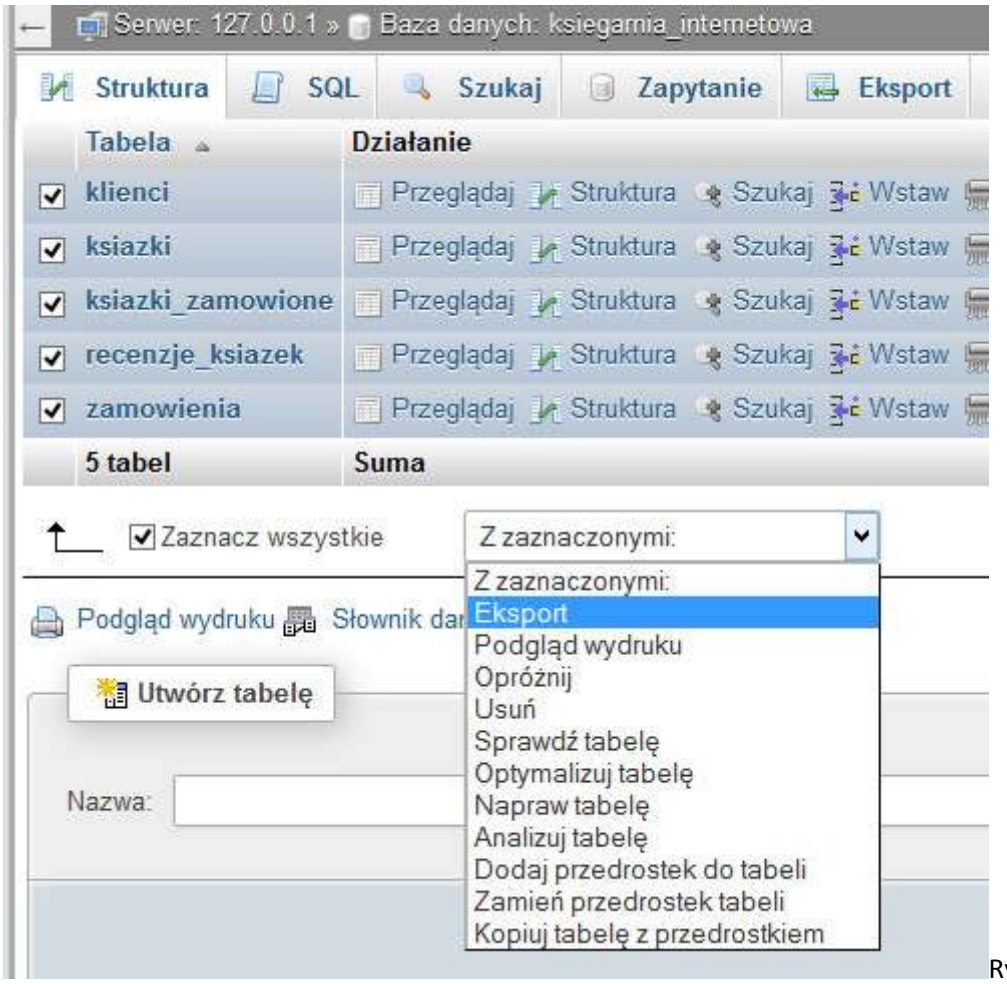


Rysunek 3_4_0_1b. Sprawdzenie utworzenia tabel bazy *ksiegarnia_internetowa*

Przyjrzyj się kodowi SQL. Jest w nim kilka fragmentów, które widzisz po raz pierwszy. Na kolejnych zajęciach wszystko będzie wytłumaczone.

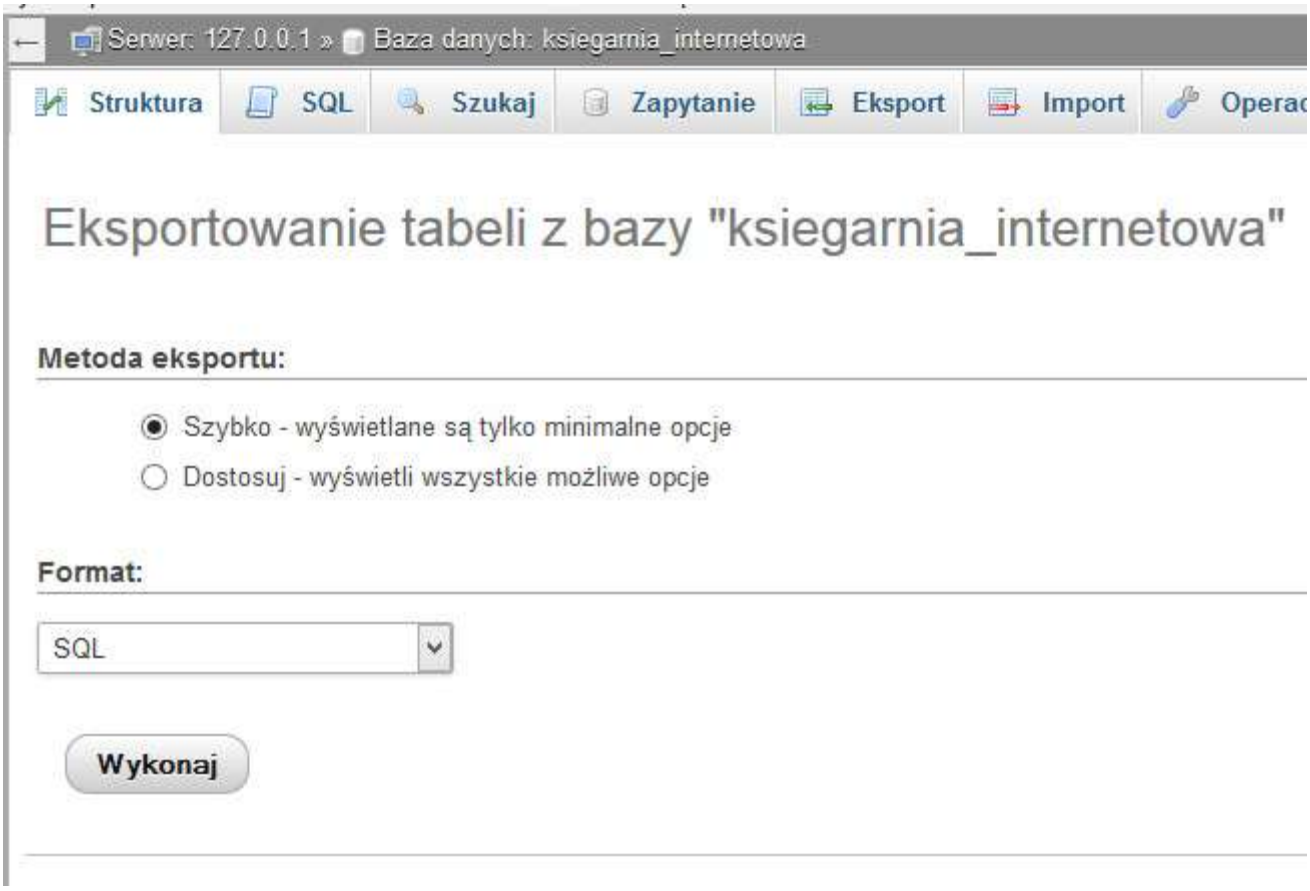
Ćwiczenie 3_4_0_2. Eksport tabel bazy *ksiegarnia_internetowa*

Nauczymy się eksportować tabele bazy danych do pliku, który możemy przechowywać na nośniku zewnętrznym. Otwórz bazę *ksiegarnia_internetowa*, zakładka *Struktura*, zaznacz wszystkie tabele i z listy rozwijanej wybierz *Export*.



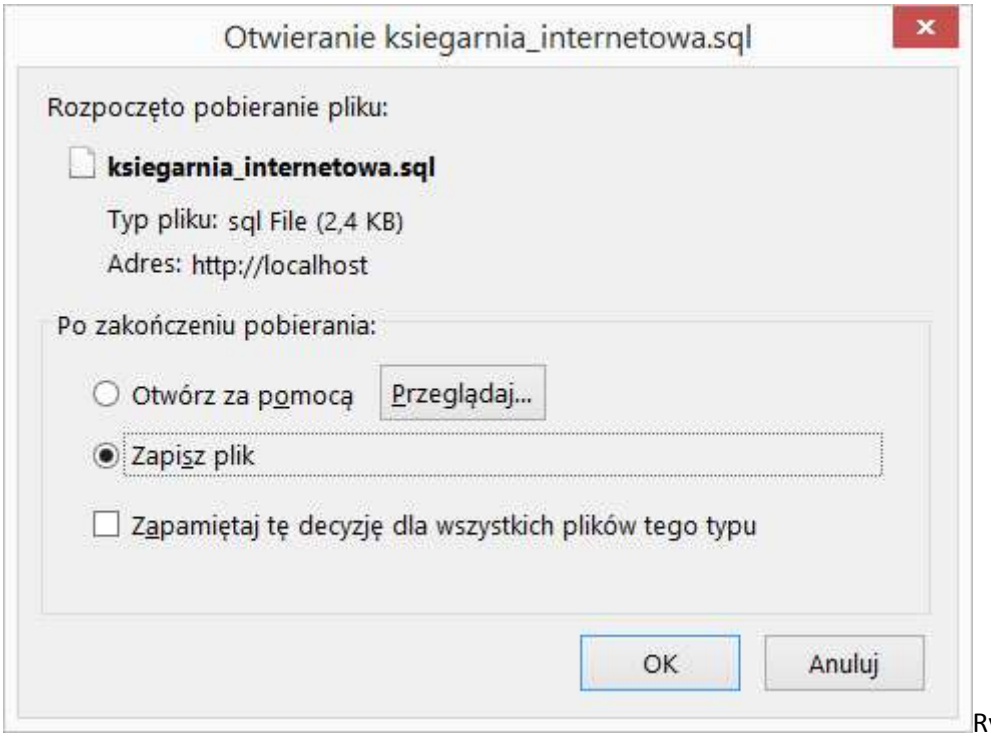
Rysunek 3_4_0_2a. Eksport tabel bazy *ksiegarnia_internetowa* - krok 1

W kroku drugim pozostawiamy ustawienia domyślne i klikamy *Wykonaj*



Rysunek 3_4_0_2b. Eksport tabel bazy *ksiegarnia_internetowa* - krok 2

W trzecim kroku, zapisujemy plik bazy danych.



Rysunek 3_4_0_2c. Eksport tabel bazy *ksiegarnia_internetowa* - krok 3

Ćwiczenie 3_4_0_3. Import tabel bazy *ksiegarnia_internetowa*

Nauczymy się importować tabele bazy danych z pliku, otrzymanego wcześniej poprzez eksport tabel bazy danych. Otwórz bazę *ksiegarnia_internetowa*, zakładka *Import*, wybierz plik i kliknij *Wykonaj*.

Rysunek 3_4_0_3. Import tabel bazy *ksiegarnia_internetowa*

Atrybuty kolumn, indeksowanie, typy kolumn

Podczas tych zajęć omówimy nowe elementy SQL, które wystąpiły **podczas tworzenia tabel**:

Atrybuty kolumn

Analizując polecenie SQL, za pomocą którego tworzymy nową tabelę, możemy zauważyć, że **niektóre kolumny posiadają atrybuty wyróżniające je spośród innych kolumn**:

- `NOT NULL` - w każdym rekordzie kolumny oznaczonej tym atrybutem, musi znajdować się jakaś wartość. Jeżeli ten atrybut został pominięty, mamy sytuację przeciwną - w danej kolumnie, mogą znajdować się pola puste `NULL`.
- `AUTO_INCREMENT` - dosłownie *automatyczny przyrost*. Stosuje się dla liczb całkowitych. Jeżeli utworzymy nowy rekord, to MYSQL automatycznie doda w tej kolumnie unikalny identyfikator - liczbę o jeden większą od największej z liczb znajdujących się w kolumnie. Jak łatwo się domyślić, taki atrybut stosujemy w kolumnie, która jest kluczem głównym tabeli. Kolumny z tym atrybutem muszą być indeksowane. **Indeksy**, to struktury danych umożliwiające szybkie wyszukiwanie danych w bazie danych. Mówiąc krótko - bez indeksów, baza danych działałaby bardzo powoli. Dobra wiadomość - w kolumnach, które zostały wskazane jako klucze, indeksy są tworzone automatyczne. Jest co prawda polecenie SQL `CREATE INDEX`, ale na naszym poziomie nie będziemy go stosować.
- `PRIMARY KEY` - kolumna jest kluczem głównym tabeli. Wynika z tego, że wartości tam wpisywane myszą być unikalne. Jak już powiedzieliśmy w poprzednim punkcie, MySQL automatycznie indeksuje takie kolumny. Dodając atrybut `AUTO_INCREMENT`, powierzamy MySQL generowanie klucza - zapewnienie unikatowości oraz indeksowania.

Atrybut `PRIMARY KEY`, może być umieszczany za nazwą kolumny tylko w przypadku, gdy kluczem podstawowym tabeli jest tylko jedna kolumna. Jednak w tabeli **ksiazki_zamowione**, w skład klucza głównego wchodzą dwie kolumny, dlatego w tym przypadku klucz główny zdefiniowaliśmy następująco: `PRIMARY KEY(zamowienie_id, isbn)`.

Typy kolumn

Podczas tworzenia nowej tabeli, należy każdej kolumnie przypisać odpowiedni typ danych. Podstawowe typy danych zostaną dokładnie omówione podczas następnych zajęć, teraz przyjrzyjmy się definiowanym przez nas tabelom bazy `ksiegarnia_internetowa`. Zacznijmy od tabeli `klienci`:

```
CREATE TABLE klienci
(
klient_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  nazwisko CHAR(50) NOT NULL,
  adres CHAR(100) NOT NULL,
miejscowosc CHAR(30) NOT NULL
);
```

- W pierwszej kolumnie `klient_id` mogą znajdować się tylko **liczby całkowite** nieujemne. Wynika to z zastosowania opisanych wcześniej atrybutów `AUTO_INCREMENT` oraz `PRIMARY KEY`. Liczby całkowite określa się mianem `INT`, natomiast `UNSIGNED` wyklucza zastosowanie liczb ujemnych.
- W pozostałych kolumnach, mamy tak zwane **łańcuchy znaków (zwane także ciągami znaków, często mówi się krótko - łańcuchy lub stringi)**. Ważna uwaga - w ujęciu programistycznym, pojęcie **znaki** jest inne niż potocznie kojarzone. Często myślimy o znakach interpunkcyjnych (kropka, przecinek, itp.), znakach specjalnych (©, czy ®) itp.), natomiast stringi, to pisane za pomocą klawiatury teksty. `'Ala ma kota'` to jest właśnie przykład łańcucha znaków. Typ `CHAR(50)`, definiuje łańcuchy zawierające nie więcej jak 50 znaków - jest to tak zwana **długość łańcucha**. Weźmy dla przykładu kolumnę `nazwisko` - nawet jeżeli wpisane nazwisko zajmie mniej jak 50 znaków, to i tak zajmie ono tyle samo pamięci, jak nazwisko składajace się z 50 znaków - MySQL po prostu uzupełni spacjami pozostałe wolne miejsca. Tworząc bazę danych musimy oszczędnie gospodarować pamięcią - pamiętajmy, że tworzymy je dla zarządzania wielkimi zasobami danych.

Kolejne **typy danych** zauważymy analizując kod tworzący tabelę `zamowienia`:

```
CREATE TABLE zamowienia
(
zamowienie_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
klient_id INT UNSIGNED NOT NULL,
wartosc FLOAT(6,2),
  data DATE NOT NULL
);
```

- Kolumna `wartosc` jest deklarowana, jako `FLOAT(6,2)`. Są to **liczby zmiennoprzecinkowe** znane Ci z matematyki jako **liczby rzeczywiste**. Na przykład liczby *6.23*, *24.455* są przykładami liczb zmiennoprzecinkowych. Przy czym **piszemy kropki,**

nie przecinki. Pierwsza liczba w nawiasach oznacza maksymalną liczbę cyfr jaka może być wyświetlona, natomiast liczba druga - ilość cyfr pokazywanych po przecinku. Złotówka ma 100 groszy, więc kwoty będą pokazywane z dokładnością do 2 miejsc po przecinku.
- W kolumnie `data` mamy dane typu `DATE` wyświetlaną w formacie RRRR-MM-DD, np. 2014-10-29.
- Zwróć uwagę, że kolumna `wartosc` nie posiada atrybutu `NOT NULL`. Nowe zamówienie, wpisujemy bowiem w tabeli `zamowienia`, następnie powinniśmy dodać nowe rekordy w tabeli `ksiazki_zamowione` i dopiero wtedy obliczyć wartość zamowienia. W momencie wpisywania nowego zamówienia, jego wartość nie zawsze jest znana, dlatego wartości pola `wartosc` mogą być nieokreślone.

Podobnie jak `zamowienia`, jest tworzona tabela `ksiazki`:

```
CREATE TABLE ksiazki
(
isbn CHAR(13) NOT NULL PRIMARY KEY,
  autor  CHAR(50),
tytul CHAR(100),
  cena FLOAT(4,2)
);
```

- Jak już wspominaliśmy numer `isbn` jest unikatowy dla każdej książki, więc ta kolumna będzie kluczem głównym tej tabeli.
- We wszystkich kolumnach poza kluczem głównym, mogą znajdować się wartości `NULL`, ponieważ dopuszczamy możliwość stopniowego uzupełniania danych o nazwisko autora, tytuł czy cenę.

W tabeli `ksiazki_zamowione`:

```
CREATE TABLE ksiazki_zamowione
(
zamowienie_id INT UNSIGNED NOT NULL,
isbn CHAR(13) NOT NULL,
ilosc TINYINT UNSIGNED,
  PRIMARY KEY(zamowienie_id, isbn)
);
```

- Pokazany jest sposób deklarowania klucza podstawowego, składającego się z dwóch kolumn: `zamowienie_id` oraz `isbn`.
- Typ `TINYINT UNSIGNED` oznacza liczby całkowite z zakresu 0..255.

W ostatniej z tabel `recenzje_ksiazek`:

```
CREATE TABLE recenzje_ksiazek
(
isbn CHAR(13) NOT NULL PRIMARY KEY,
  recenzja TEXT
);
```

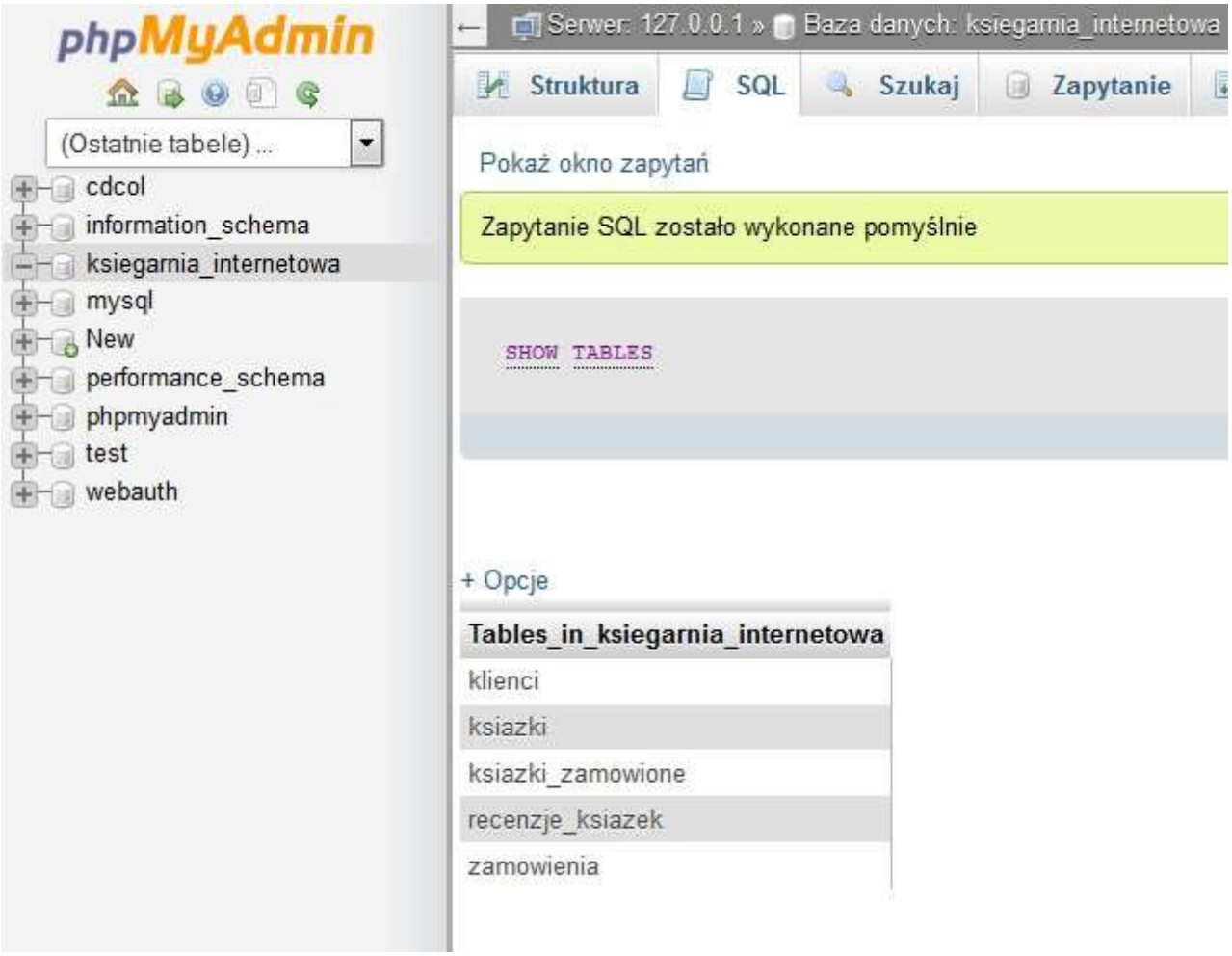
- Typ `TEXT` jest stosowany w przypadku długich tekstów.

Polecenia *SHOW* i *DESCRIBE*

Polecenia *SHOW* i *DESCRIBE* powodują wyświetlenie informacji o bazie danych oraz poszczególnych tabelach. Działanie przetestujemy wykonując ćwiczenia.

Ćwiczenie 3_4_1_1. Pokazanie listy tabel bazy *ksiegarnia_internetowa*

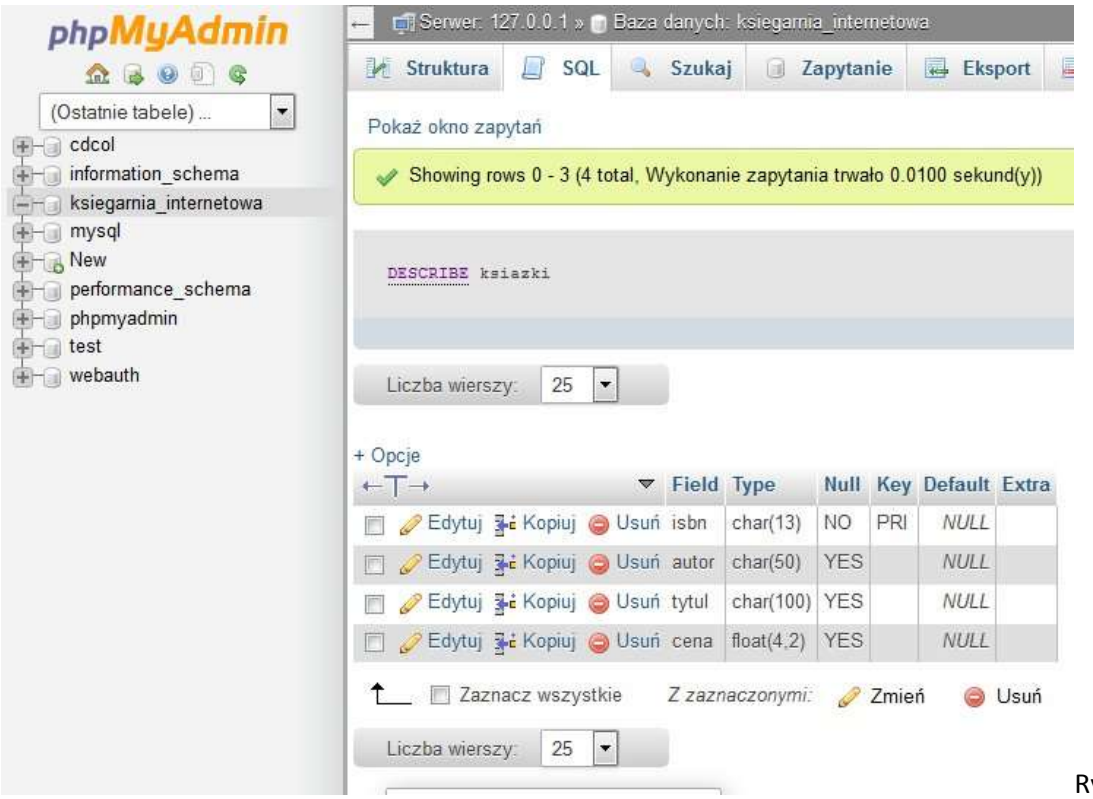
Otwórz bazę *ksiegarnia_internetowa*, zakładam że utworzyłeś w niej na poprzednicj zajęciach wszystkie tabele. Wpisz polecenie SQL `SHOW TABLES` i wciśnij *Wykonaj*.



Rysunek 3_4_1_1. Lista tabel bazy *ksiegarnia_internetowa*

Ćwiczenie 3_4_1_2. Pokazanie informacji na temat wskazanej tabeli

Otwórz bazę *ksiegarnia_internetowa*, zakładam że utworzyłeś w niej na poprzednich zajęciach wszystkie tabele. Wpisz polecenie SQL `DESCRIBE ksiazki` i wciśnij *Wykonaj*.



Rysunek 3_4_1_2. Pokazanie informacji na temat tabeli *książki*

Łączenie tabel

Podczas poprzednich zajęć wyszukiwaliśmy dane w jednej tabeli. Najczęściej jednak, występują bardziej skomplikowane sytuacje. Jak wiemy, zastosowanie relacji skutkuje "rozrzuceniem informacji" po wielu tabelach. W naszej bazie `ksiegarnia_internetowa`, jeżeli chcemy się dowiedzieć, jacy klieci złożyli zamówienia w danym miesiącu, to musimy **przejrzeć tabele** `klienci` i `zamowienia`. Jeżeli dodatkowo chcielibyśmy wiedzieć jakie książki zostały wtedy zamówione, musielibyśmy dodatkowo przejrzeć `ksiazki_zamowione`.

Aby za pomocą języka SQL uzyskać informacje znajdujące się w kilku tabelach, należy wykonać operację nazywaną łączeniem - ang. join. Chociaż sama zasada nie jest trudna do zrozumienia, to jednak łączenie tabel należy do bardziej złożonych operacji `SQL`. Należy tutaj powiedzieć, że jest kilka typów łączenia, z których każdy służy do innych celów.

Łączenie dwóch tabel

Omówimy wykonując ćwiczenie

Ćwiczenie 4_2_2_1. Łączenie dwóch tabel

Zastosujmy następujące zapytanie `SQL`:

```
SELECT zamowienia.zamowienie_id, zamowienia.wartosc,
zamowienia.data
FROM klienci, zamowienia
WHERE klienci.nazwisko='Józef Koń'
AND klienci.klient_id=zamowienia.klient_id;
```

Rysunek 4_2_2_1. Wyszukiwanie danych poprzez łączenie dwóch tabel

Omówimy teraz elementy, które wystąpiły w tym ćwiczeniu. Najpierw **przejrzyj się tabelom** `klienci` oraz `zamowienia` i porównaj wyniki wyszukiwania z danymi znajdującymi się w tych tabelach. Dla `Józef Koń`, `klient_id=5`. W tabeli `zamowienia`, `klient_id` o wartości równej 5 występuje 2 razy i właśnie te rekordy zostały pokazane w wyniku wyszukiwania. Uwagi do tego wyszukiwania:

- W pierwszym wierszu zapytania `SELECT zamowienia.zamowienie_id, zamowienia.wartosc, zamowienia.data` mamy polecenie pokazania kolumn `zamowienie_id`, `wartosc` oraz `data` z tabeli `zamowienia`. Zwróć uwagę na zastosowanie kropki, która:
  - Jest niezbędna dla określenia, z której tabeli pochodzi dana kolumna, mamy na przykład `klienci.klient_id`, ale mamy także `zamówienia.klient_id`.
  - Zwiększa czytelność polecenia, ponieważ widzimy (przed kropką) tabelę, do której dana kolumna należy. Co prawda można nazwę tabeli pominąć w sytuacji, kiedy nazwy kolumn są unikatowe, niemniej właśnie z powodu czytelności powinno się stosować zapis z kropką.

- W drugim wierszu mamy `FROM klienci, zamowienia`, a więc wymienione nazwy dwóch tabel z których będą brane dane. Nastąpiło tutaj łączenie tabel, ponieważ zamiast przecinka, który zastosowaliśmy, mogliśmy użyć takich słów kluczowych jak:
    - `INNER JOIN` - łączenie tabel po wspólnych rekordach, wyszukiwane są po prostu wiersze które do siebie pasują, w naszym przykładzie mają jednakowe wartości pól `klient_id`. Jest to najbardziej popularna operacja używana w wyszukiwaniach, zastosowana również w naszym przykładzie.
    - `CROSS JOIN` - łączenie rekordów obu tabel na zasadzie "każdy z każdym". Taki typ połączenia tabel nosi nazwę **fulljoin** albo **iloczyn kartezjański**. Łączy on każdy wiersz pierwszej tabeli z każdym wierszem tabeli drugiej. W naszym przykładzie w wyniku takiego łączenia otrzymalibyśmy jedną dużą tabelę, w której każdy wiersz tabeli `klienci` byłby połączony z każdym wierszem tabeli `zamowienia`, bez względu na to, czy dany klient złożył to zamówienie czy nie. Jest to mało wyrafinowana operacja, zajmująca dużo pamięci i w większości przypadków nic nam nie dająca. W naszym przykładzie rozpatrywane są rekordy z zamówieniami złożonymi przez wskazanego klienta w połączeniu z rekordem zawierającym jego dane.

  Pasujące do siebie rekordy są odnajdowane dzięki zastosowaniu **warunku połączenia (joincondition)** w klauzuli `WHERE`. W naszym przykładzie, wyrażenie warunkowe `klienci.klient_id=zamowienia.klient_id` jednoznacznie wskazuje pola które ustanawiają zwiazek miedzy tabelami. Dzięki niemu `MYSQL` pokazał w tabeli wynikowej tylko te wiersze, w których wartość pola `klient_id` z tabeli `klienci` odpowiada wartości pola `klient_id` z tabeli `zamowienia`.

- Dodanie do zapytania warunku połączenia, spowodowało zmianę tytu połączenia na typ określany jako **equi-join**.

Łączenie trzech lub więcej tabel

Zasadą jest zestawianie tabel parami łączonymi poprzez określony warunek łączenia. W przypadku trzech tabel, można powiedzieć, że należy zdefiniować związki między danymi w jednej tabeli z danymi w drugiej tabeli, które to dane należy powiązać z danymi tabeli trzeciej.

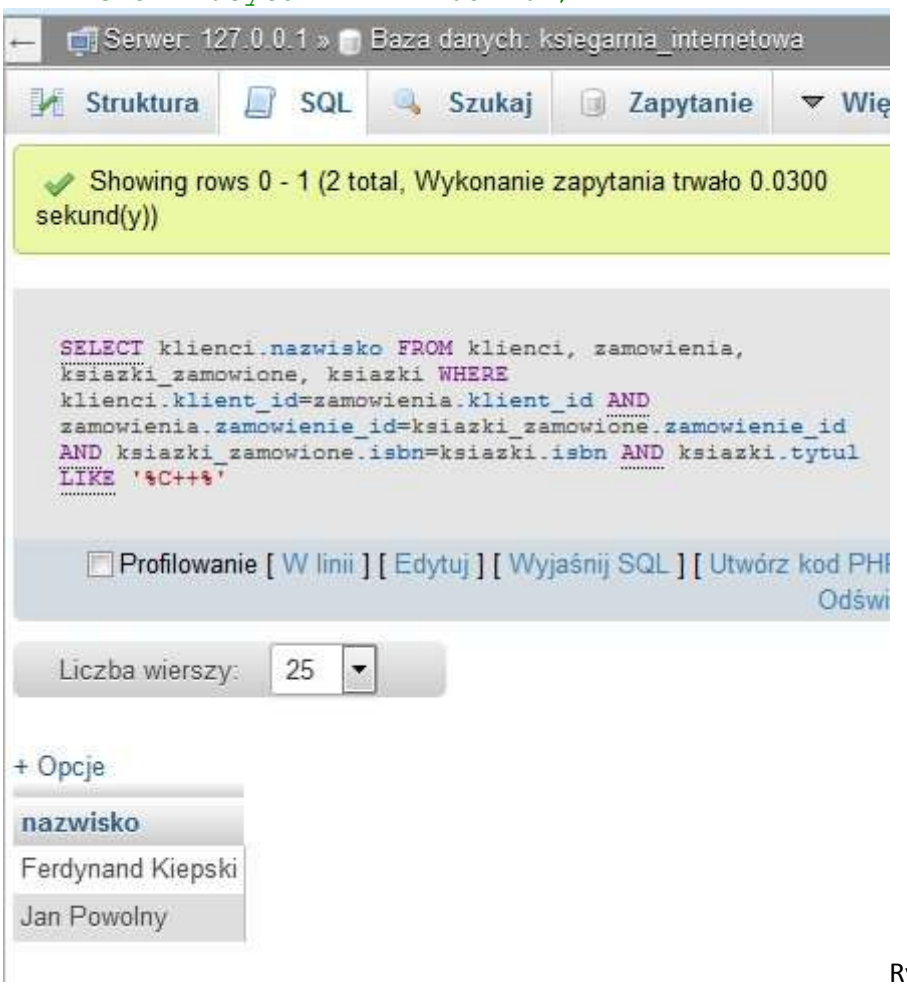
W naszej bazie danych `ksiegarnia_internetowa`, aby znaleźć klientów, którzy nabyli książki dotyczące języka programowania *C++* należy wziąć pod uwagę 4 tabele: `klienci`, `zamowienia`, `ksiazki` oraz `ksiazki_zamowione`.

1. Aby powiązać dane z tabeli `klienci`, z danymi z tabeli `zamowienia`, trzeba przeanalizować wartości pól `klient_id`, tak jak to robiliśmy w ostatnim ćwiczeniu.
2. Tabela `zamowienia` jest powiązana z tabelą `ksiazki_zamowione`, za pomocą pola `zamowienie_id`, natomiast tabela `ksiazki_zamowione` powiązana jest z tabelą `ksiazki` za pomocą pola `isbn`.

Ćwiczenie 4_2_2_2. Łączenie kilku tabel

Aby wykonać zadanie opisane powyżej, zastosujmy następujące zapytanie `SQL`:

```
SELECT klienci.nazwisko
FROM klienci, zamowienia, ksiazki_zamowione, ksiazki
WHERE klienci.klient_id=zamowienia.klient_id
AND zamowienia.zamowienie_id=ksiazki_zamowione.zamowienie_id
AND ksiazki_zamowione.isbn=ksiazki.isbn
AND ksiazki.tytul LIKE '%C++%';
```



Rysunek 4_2_2_2.
Wyszukiwanie danych poprzez łączenie kliku tabel

Jak więc widzimy, panowie Ferdynand Kiepski oraz Jan Powolny, są miłośnikami programowania w C++.

Powiązaliśmy dane z 4 tabel. Aby je powiązać za pomocą połączenia typu *equi-join*, należało zastosować 3 różne warunki połączenia. W przypadku łączenia tabel, prawidłowością jest, że na każdą ich parę przypada jeden warunek. W rezultacie, **suma wszystkich nałożonych warunków połączenia, jest o jeden mniejsza, niż liczba łączonych tabel**.

Typy połączeń

Na tych zajęciach **podsumujemy i utrwalimy stosowanie różnych typów połączeń tabel**. W tym celu, utworzymy bardzo prostą bazę `szkola`, składającą się z dwóch bardzo prostych tabel `uczniowie` oraz `klasy`, a następnie przetestujemy podstawowe typy połączeń.

uczniowie	
id_uczen	nazwisko
1	Mądry
2	Zdolny
3	Pracowity
4	Zaradny
5	Ambitny

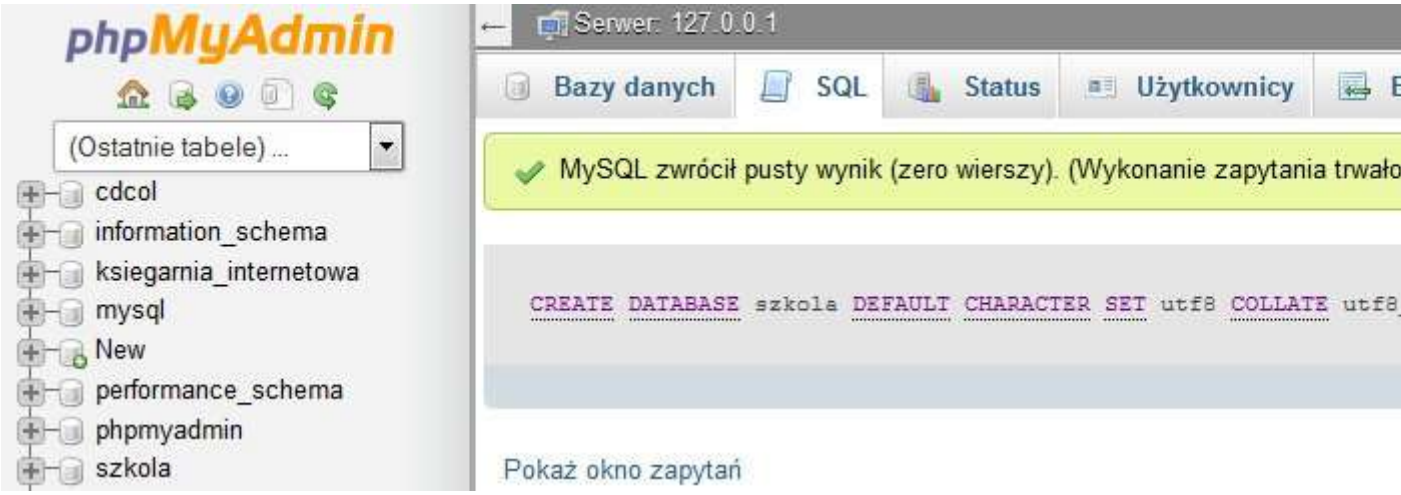
klasy	
id_uczen	klasa
2	1 A
4	1 A
5	2 B

Tabele bazy danych `szkola`

Ćwiczenie 4_2_5_1. Utworzenie bazy danych *szkola*

Zastosuj polecenie SQL:

```
CREATE DATABASE szkola DEFAULT CHARACTER SET utf8 COLLATE
utf8_unicode_ci;
```



Rysunek 4_2_5_1. Utworzenie bazy danych `szkola`

Ćwiczenie 4_2_5_2. Utworzenie tabel bazy danych *szkola*

Zastosuj polecenie SQL:

```
CREATE TABLE uczniowie
(
id_uczen INT UNSIGNED NOT NULL PRIMARY KEY,
  nazwisko CHAR(50) NOT NULL
);
CREATE TABLE klasy
(
id_uczen INT UNSIGNED,
  klasa CHAR(20)
);
```

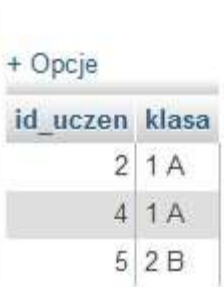
Rysunek 4_2_5_2. Utworzenie tabel bazy danych `szkola`

Ćwiczenie 4_2_5_3. Wypełnienie tabel danymi

Zastosuj polecenie SQL:

```
INSERT INTO uczniowie VALUES
  (1, 'Mądry'),
  (2, 'Zdolny'),
  (3, 'Pracowity'),
  (4, 'Zaradny'),
  (5, 'Ambitny');

INSERT INTO klasy VALUES
  (2, '1 A'),
  (4, '1 A'),
  (5, '2 B');
```

Rysunek 4_2_5_4. Tabela `uczniowie`



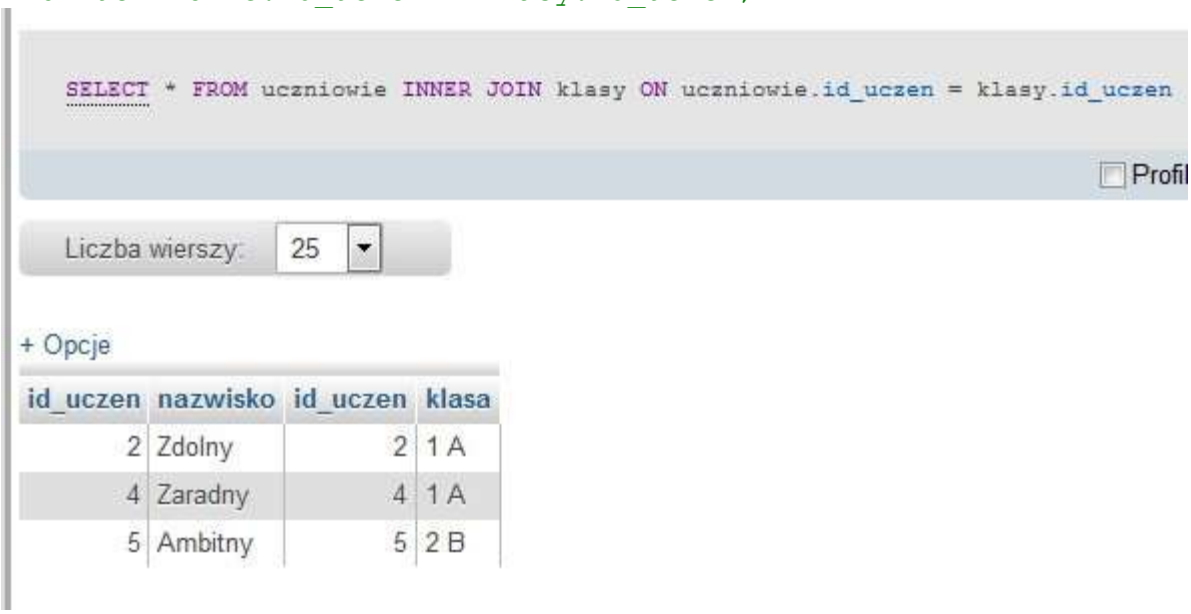
Rysunek 4_2_5_5. Tabela `klasy`

Ćwiczenie 4_2_5_4. Połączenie *innerjoin*

Jest to **najprostsze połączenie**. Umożliwia pobrać wszystkie rekordy z tabeli A mające odpowiedniki w tabeli B. Mówiąc inaczej, pobierana jest część wspólna tych tabel. Klauzula

pozwala na łączenie 2 lub więcej tabel. W naszym przykładzie elementem wspólnym tabel, na podstawie którego wykonywane jest porównywanie jest "id_uczen". Zastosuj następujące zapytanie SQL:

```
SELECT * FROM uczniowie
  INNER JOIN klasy
  ON uczniowie.id_uczen = klasy.id_uczen;
```



Rysunek 4_2_5_6. Połączenie *innerjoin*

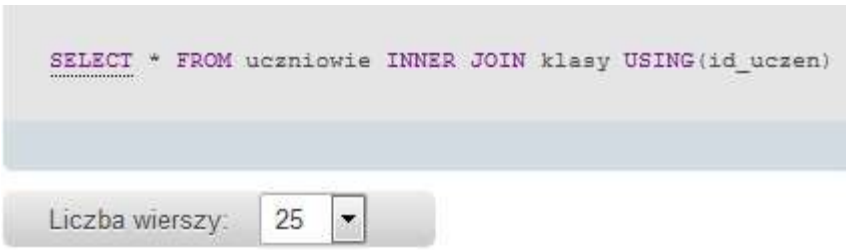
Identyczny efekt uzyskamy, stosując zapytanie:

```
SELECT * FROM uczniowie, klasy
  WHERE uczniowie.id_uczen = klasy.id_uczen;
```

Ćwiczenie 4_2_5_5. Połączenie *equi-join*

Jest to szczególny przypadek *innerjoin*, w którym do porównań używa się tylko znaków równości. Tak więc poprzedni przykład jest połączeniem typu *equi-join*. MySQL umożliwia dla *equi-join*, zastosowanie wersji skróconej:

```
SELECT * FROM uczniowie
  INNER JOIN klasy
  USING(id_uczen);
```

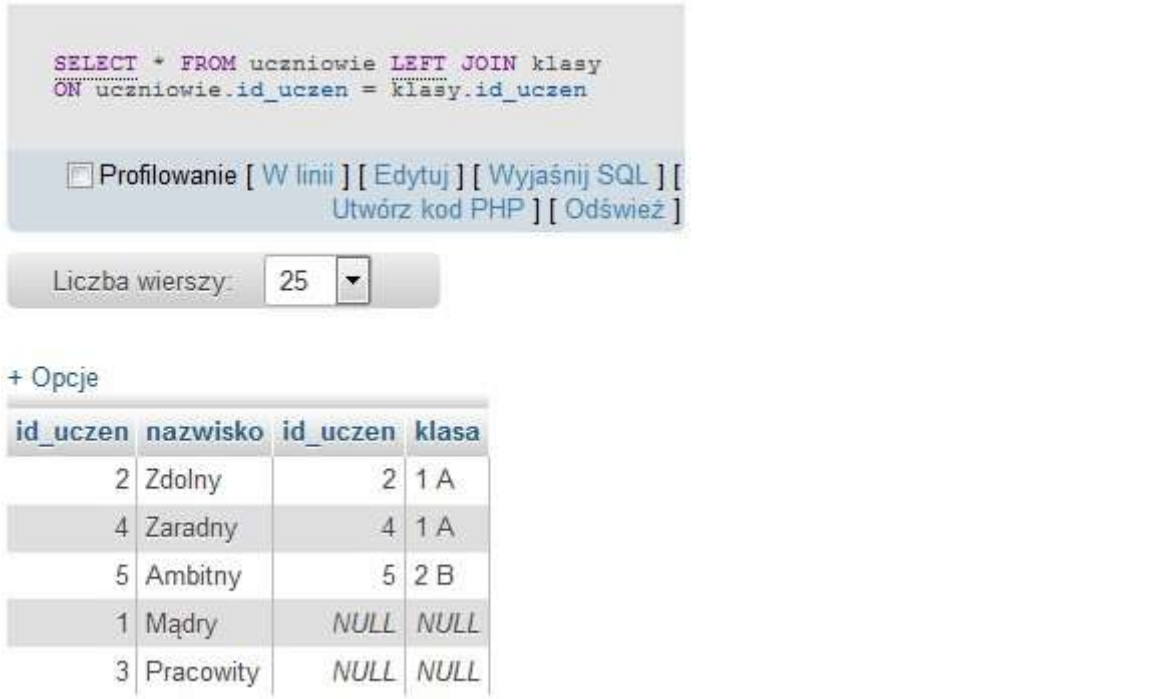


Rysunek 4_2_5_7. Połączenie *equi-join*

Ćwiczenie 4_2_5_6. Połączenie *leftjoin*

Jest to **najczęstsze z używanych połączeń**. Pozwala pobrać wszystkie rekordy z tabeli A wraz z odpowiadającymi im rekordami z tabeli B nawet wtedy, gdy w tabeli B nie ma odpowiedników dla rekordów z tabeli A.

```
SELECT * FROM uczniowie
LEFT JOIN klasy ON uczniowie.id_uczen = klasy.id_uczen;
```



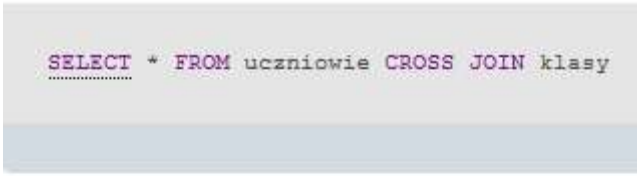
Rysunek 4_2_5_8. Połączenie *leftjoin*

Wiersze w których nie ma odpowiedników, zostały wypełnione wartościami `NULL`.

Ćwiczenie 4_2_5_7. Połączenie *cross join*

Jest to **najciekawsze z połączeń**. W wyniku daje wszystkie możliwe kombinacje rekordów z tabeli A z rekordami z tabeli B (iloczyn kartezjański). Należy o tym pamiętać, jeżeli chcielibyśmy wykonać takie połączenie dwóch dużych tabel. Liczba wyników jest iloczynem liczby rekordów tabeli A i liczby rekordów tabeli B. Równoznaczym typem połączenia, jest *fulljoin*.

```
SELECT * FROM uczniowie
CROSS JOIN klasy;
```



Liczba wierszy: 25

+ Opcje

id_uczen	nazwisko	id_uczen	klasa
1	Mądry	2	1 A
1	Mądry	4	1 A
1	Mądry	5	2 B
2	Zdolny	2	1 A
2	Zdolny	4	1 A
2	Zdolny	5	2 B
3	Pracowity	2	1 A
3	Pracowity	4	1 A
3	Pracowity	5	2 B
4	Zaradny	2	1 A
4	Zaradny	4	1 A
4	Zaradny	5	2 B
5	Ambitny	2	1 A
5	Ambitny	4	1 A
5	Ambitny	5	2 B

Rysunek 4_2_5_9. Połączenie *cross join*