***BIBLIOTEKA***

Projekt bazy danych

MSSQL SERVER 2017 / C#

MySQL / Python

Autorzy: Kamil Prokopiuk

Sylwester Kwiatkowski

Spis treści

[I. Opis 3](#_Toc9445771)

[II. Tabele bazy: 3](#_Toc9445772)

[III. Relacje 5](#_Toc9445773)

[III. Diagram bazy 6](#_Toc9445774)

[IV. Dostępne operacje: 7](#_Toc9445775)

[a. Dodawanie rekordów do bazy 7](#_Toc9445776)

[b. Przeglądanie tabel wg podanego kryterium 7](#_Toc9445777)

[c. Dodanie wszystkich dostępnych książek do listy, na której można prowadzić operacje 7](#_Toc9445778)

[d. Zamówienie książki/Dodanie wypożyczenia na podstawie modelu 7](#_Toc9445779)

[V. Zarządzanie bazą danych i operacje na bazie przy pomocy aplikacji w języku Python 9](#_Toc9445780)

[1. Wstęp 9](#_Toc9445781)

[2. Ogólny zarys programu 10](#_Toc9445782)

[3. Przykłady składni i poleceń 12](#_Toc9445783)

[a. Wyświetlanie czyli opcja „SELECT” 12](#_Toc9445784)

[b. Wyszukiwanie autora 13](#_Toc9445785)

[c. Dodawanie autora do bazy 13](#_Toc9445786)

[d. Dodawanie wypożyczenia 13](#_Toc9445787)

[4. Podsumowanie 14](#_Toc9445788)

# Opis

Celem projektu było utworzenie projektu bazy danych do obsługi biblioteki, tj. dodawania, usuwania i edycji wpisów jak np. książka, autor książki, czytelnik oraz obsługi wypożyczeń. Sam projekt bazy danych dla użytkownika, który nie zna się na obsłudze bazy będzie dla niego nieprzydatny, dlatego stworzyliśmy aplikacje do obsługi danej bazy przez użytkownika w łatwy i przystępny dla niego sposób. Pierwsza wersja to aplikacja konsolowa napisana w języku Python, druga to aplikacja webowa napisana w technologii ASP.NET w języku C#.

# Tabele bazy:

**Autorzy**

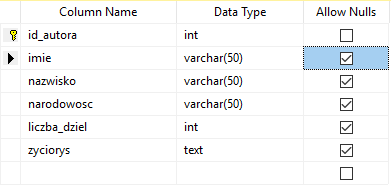
MySQL: SELECT \* FROM [biblioteka].autorzy

MSSQL: wykorzystanie obiektowości C#

Utworzenie połączenia z bazą używając Entity Framework  
 private bibliotekaEntities1 db = new bibliotekaEntities1();

db.autorzy to obiekt zawierający wszystkie kolumny z tabeli autorzy, można z niego wybrać dowolną kolumnę i wypisać ją w aplikacji. Analogicznie będzie z kolejnymi tabelami.

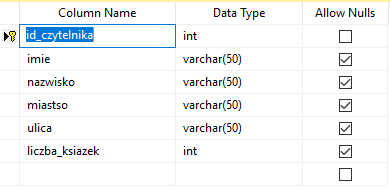
db.autorzy; //Zawiera wszystkie wyniki z tabeli autorzy



**Czytelnicy**

MySQL: SELECT \* FROM [biblioteka].czytelnicy

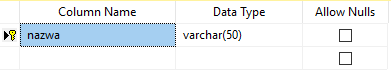
MSSQL: db.czytelnicy;



**Gatunki**

MySQL: SELECT \* FROM [biblioteka].gatunki

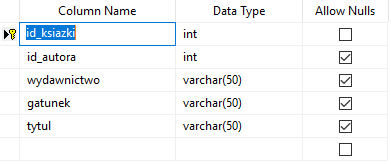
MSSQL: db.gatunki;



**Książki**

MySQL: SELECT \* FROM [biblioteka].ksiazki

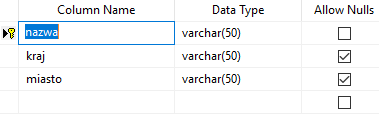
MSSQL: db.ksiazki;



**Wydawnictwa**

MySQL: SELECT \* FROM [biblioteka].wydawnictwa

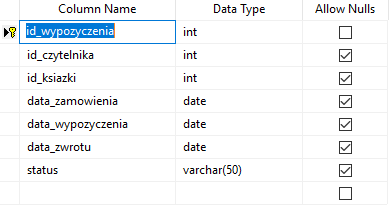
MSSQL: db.wydawnictwa;



**Wypożyczenia**

MySQL: SELECT \* FROM [biblioteka].wypozyczenia

MSSQL: db.wypozyczenia;



# Relacje

SELECT TABLE\_NAME, COLUMN\_NAME, CONSTRAINT\_NAME

FROM information\_schema.KEY\_COLUMN\_USAGE

ORDER BY TABLE\_NAME

|  |  |  |
| --- | --- | --- |
| TABLE\_NAME | COLUMN\_NAME | CONSTRAINT\_NAME |
| autorzy | id\_autora | PK\_autorzy |
| czytelnicy | id\_czytelnika | PK\_czytelnicy |
| gatunki | nazwa | PK\_gatunki |
| ksiazki | id\_autora | FK\_ksiazki\_autorzy |
| ksiazki | gatunek | FK\_ksiazki\_gatunki |
| ksiazki | wydawnictwo | FK\_ksiazki\_wydawnictwa |
| ksiazki | id\_ksiazki | PK\_ksiazki |
| wydawnictwa | nazwa | PK\_wydawnictwa |
| wypozyczenia | id\_czytelnika | FK\_wypozyczenia\_czytelnicy |
| wypozyczenia | id\_ksiazki | FK\_wypozyczenia\_ksiazki |
| wypozyczenia | id\_wypozyczenia | PK\_wypozyczenia |

# Diagram bazy



# Dostępne operacje:

* 1. Należy zakodować w SQL następujące operacje czytelnika: rejestracja w bazie, przeglądanie całego księgozbioru według wybranych przez siebie kryteriów (gatunek, autor, tytuł, zakres lat…), sprawdzanie dostępności wybranych książek, zamawianie książek.
  2. Bibliotekarz powinien dodawać i usuwać gatunki, autorów, wydawnictwa, książki, egzemplarze, realizować zamówienia i potwierdzać zwrot (zmieniać status wypożyczenia). Nie powinno być możliwe dodanie książki bez wydawnictwa i autora.
  3. Dodatkowo należy pokazać przykłady przeglądania statystyk wypożyczeń danego czytelnika, danego gatunku, autora, tytułu, egzemplarza w zadanym okresie czasu.

## Dodawanie rekordów do bazy

MSSQL + C#: public ActionResult Create([Bind(Include = "id\_autora,imie,nazwisko,narodowosc,liczba\_dziel,zyciorys")] autorzy autorzy)

        {

            if (ModelState.IsValid)

            {

                autorzy.liczba\_dziel = 0;

                db.autorzy.Add(autorzy);

                db.SaveChanges();

                return RedirectToAction("Index");

            }

            return View(autorzy);

        }

## Przeglądanie tabel wg podanego kryterium

MSSQL + C#:

db.nazwa\_tabeli.Where(x => x.nazwa\_kolumny == kryterium)

## Dodanie wszystkich dostępnych książek do listy, na której można prowadzić operacje

List<ksiazki> notReserved = new List<ksiazki>();

            notReserved = db.ksiazki.Where(x => x.status == "Dostępna" & x.status != "Zarezerwowana" || x.status == null).ToList();

## Zamówienie książki/Dodanie wypożyczenia na podstawie modelu

public ActionResult Create([Bind(Include = "id\_wypozyczenia,id\_czytelnika,id\_ksiazki,data\_zamowienia,data\_wypozyczenia,data\_zwrotu,status")] wypozyczenia wypozyczenia)

        {

                db.wypozyczenia.Add(wypozyczenia);

}

**Pkt 2. Dodawanie pól do tabel również działa na podstawie danych z formularzy**

**Przykład na podstawie tabeli autorzy. Nowy autor automatycznie ma wpisywaną wartość liczba\_dziel = 0;**

**Zabezpieczenie formularzy sprawia, że nie ma możliwości dodania książki bez autora i wydawnictwa.**

public ActionResult Create([Bind(Include = "id\_autora,imie,nazwisko,narodowosc,liczba\_dziel,zyciorys")] autorzy autorzy)

        {

            if (ModelState.IsValid)

            {

                autorzy.liczba\_dziel = 0;

                db.autorzy.Add(autorzy);

                db.SaveChanges();

                return RedirectToAction("Index");

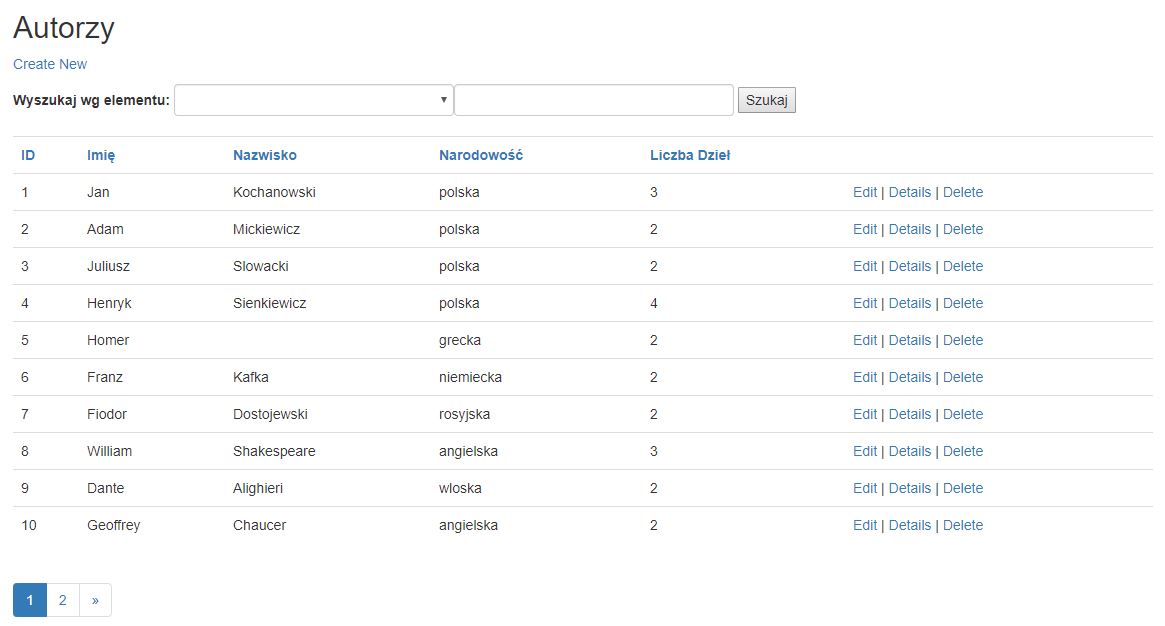
            }

            return View(autorzy);

        }

**Pkt 3. Przeglądanie statystyk**

W aplikacji w wersji Webowej zaimplementowana została możliwość filtracji oraz sortowania danych wg wybranego kryterium. Przykład:



# Zarządzanie bazą danych i operacje na bazie przy pomocy aplikacji w języku Python

## Wstęp

Python umożliwia prosty dostęp do bazy danych przy wykorzystaniu natywnych poleceń i składni języka SQL. Aby móc uzyskać do nich dostęp niezbędne jest wykorzystanie modułu mysql.connector.



Następnie należy utworzyć obiekt klasy **connect** oraz nadać parametry dla hosta, użytkownika i bazy danych, z którą chcemy nawiązać połączenie.



Dalej tworzymy obiekt klasy **cursor**, który pozwoli nam na wykonywanie operacji na bazie.

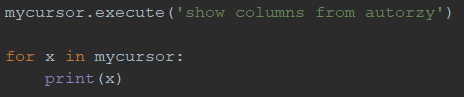


Wtedy przy pomocy polecenia **execute** wykonujemy zapytania SQL w argumencie podając wprost składnię SQL.

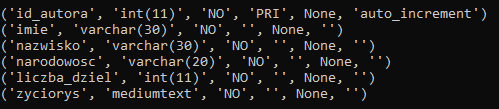


W obiekcie kursor przechowywane będą wszystkie wartości kolumn z tabeli **autorzy**.

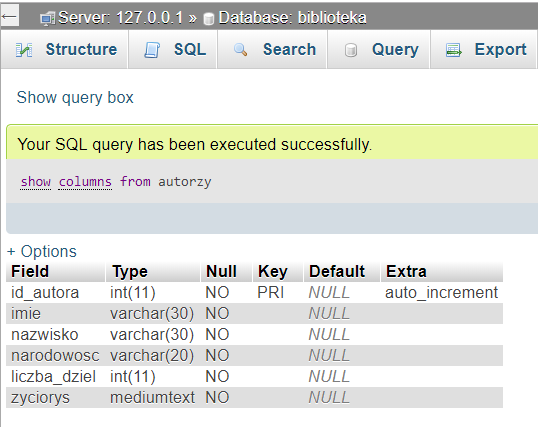
Możemy teraz przy pomocy pętli **for** wyświetlić w terminalu wszystkie te wartości:



Co daje następujący rezultat:



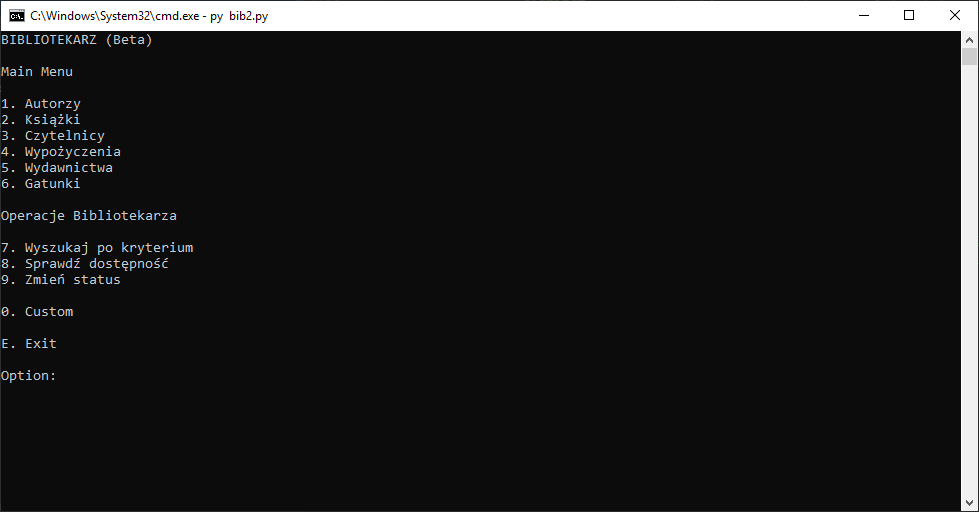
Dla porównania to samo zapytanie wykonane w phpMyAdmin wygląda tak:



Składnia zapytania oraz otrzymany rezultat są praktycznie identyczne, nie licząc środowiska i GUI, które z wiadomych względów znacząco się róznią, ale nie wpływają w żaden sposób na efekt końcowy jeśli chodzi prezentację danych.

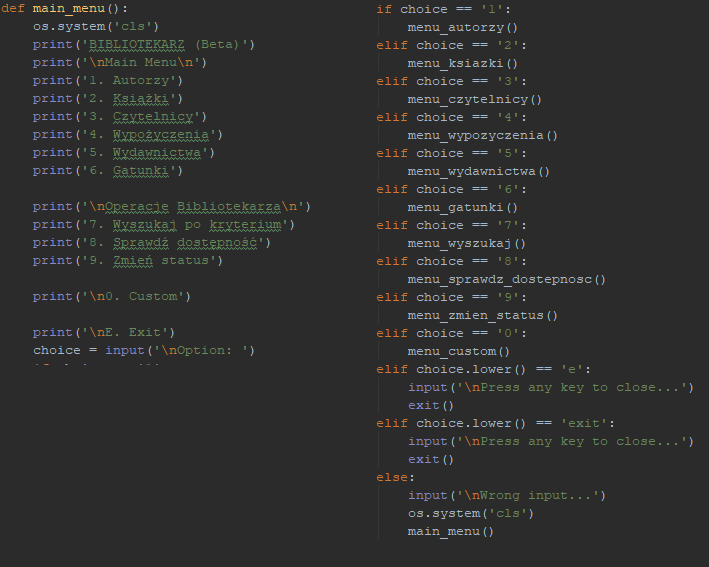
## Ogólny zarys programu

Program nie korzysta z GUI, wszystkie polecenia i wyniki wyświetlane są na ekranie terminala. Charakteryzuje go prostota i przejrzystość, już po kilku minutach użytkowania nabiera się wprawy i poruszanie się po różnych częściach menu wydaje się wręcz intuicyjne. Po uruchomieniu programu ukazuje się nam menu główne:

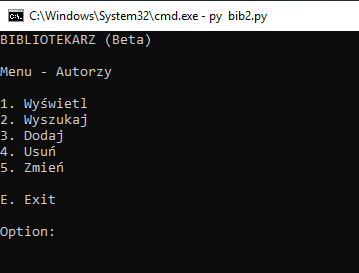


Wystarczy wprowadzić numer 0 – 9 odpowiadający róznym opcjom menu i zatwierdzając nasz wybór Enterem przechodzimy do kolejnej sekcji. Wprowadzenie znaku „E” lub „e” powoduje wyjście z programu lub powrót do poprzednego menu.

Kod menu jest funkcją, która skłąda się z serii poleceń **print,** które drukują w terminalu dostępne polecenia oraz pętli **if,** która pozwala przejściu do kolejnych części programu:



Po wpisaniu z klawiatury znaku „1” oraz zatwierdzeniu Enterem ukaże nam się menu Autorzy:



## Przykłady składni i poleceń

Jak widać we wstępie wykorzystanie Pythona w nawiązaniu połączenia z bazą danych jak i jej obsługa jest bardzo prosta i czytelna. Zalety te wynikają wprost ze składni języka, która jest spójna i przejrzysta oraz możliwości wykorzystania zewnętrznych modułów, takich jak mysql.connector, które w znacznym stopniu przyspieszają pisanie kodu.

### Wyświetlanie czyli opcja „SELECT”

Aby wyświetlić wszystkich autorów z tabeli **autorzy** używamy klauzuli SELECT \* FROM `AUTORZY`.

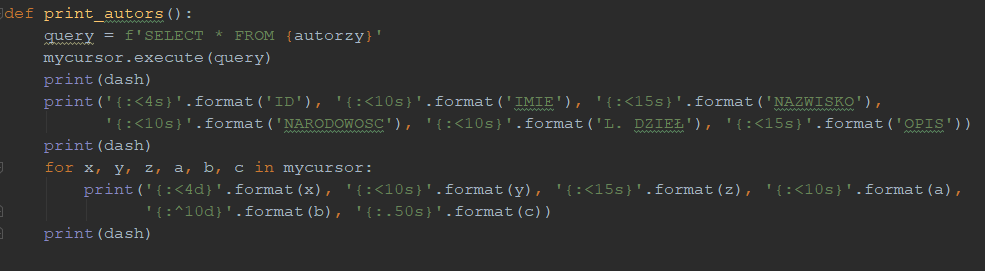


Po odpowiednim formatowaniu wyświetlonych danych otrzymamy:



Całe zapytanie zostało ujęte w funkcji, co pozwala na wykorzystanie go w innych częściach programu w prosty sposób, po prostu wywołując ową funkcję.

Kod całej funkcji\*:

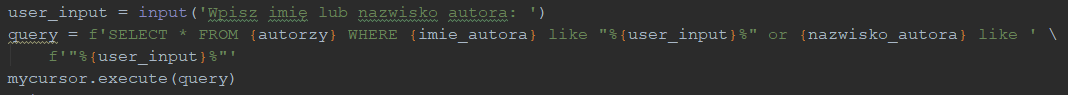


\*W poleceniu **execute** jako argumentu użyto zmiennej **query,** która została zdefiniowana wcześniej, i nadano jej wartość, którą jest składnia zapytania SQL. To rozwiązanie pozwala na użycie tego samego polecenia wiele razy i jest to zabieg mający na celu zmniejszenie objętości kodu, jednocześnie zwiększając jego czytelność. To rozwiązanie jest również sugerowaną drogą pisania programów w Pythonie (czyli jest bardziej po pythonowsku).

### Wyszukiwanie autora

Aby wyszukać autora program poprosi nas o podanie jego imienia i/lub nazwiska, następnie przypisywane jest ono do zmiennej, która jest użyta w zapytaniu SQL: SELECT \* FROM `autorzy` WHERE `imie` like `zmienna` OR `nazwisko` like `zmienna`.

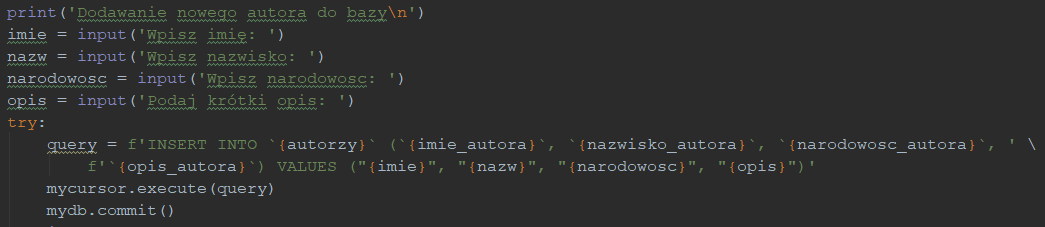
Część programu odpowiedzialna za wykonanie polecenia:



### Dodawanie autora do bazy

W przypadku dodawania autora do bazy wykorzystano polecenie INSERT INTO ... VALUES ...

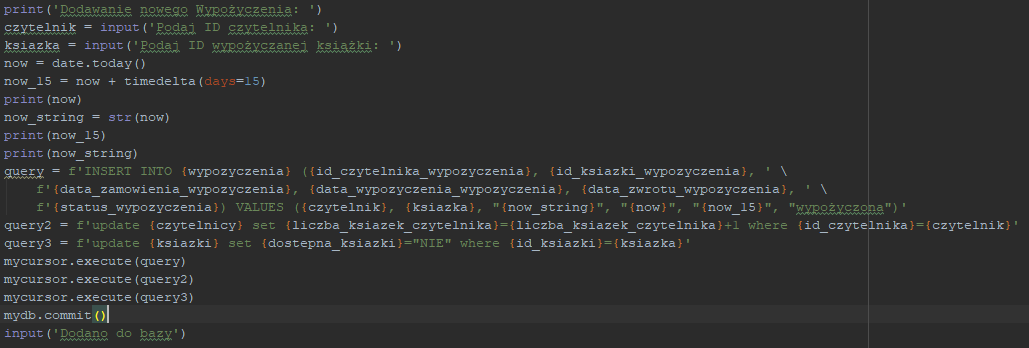
Kod tego polecenia w programie wygląda następująco:



Po wybraniu opcji z menu program zapyta użytkownika o podanie wartości dla argumentów **imię, nazwisko, narodowość** oraz **opis.** Następnie dane te są przypisywanie do zmiennych, a zmienne umieszczane w zapytaniu SQL. Aby efekt był widoczny w bazie danych należy po wykonaniu zapytania przesłać jego efekty do bazy poprzez polecenie **mydb.commit()**.

### Dodawanie wypożyczenia

Dodawanie wypożyczenia polega na utworzeniu nowego rekordu w tabeli wypożyczenia, uaktualnieniu licznika książek dla użytkownika, który wypożyczył książkę oraz zmiany statusu. Program prosi o podanie ID czytelnika oraz ID wypożyczonej książki. Następnie zmienne te są podstawiane do zapytania SQL. Zapytanie korzysta z funkcji INSERT INTO ... VALUES oraz UPDATE ... SET ... WHERE. Mimo, że kod może wydawać się na pierwszy rzut oka onieśmielający, po dokładniejszym przyjrzeniu się można zauważyć, że dalej korzystamy z tych samuch podstawowych poleceń SQL, mimo że są trochę inaczej sformatowane aby sprostać wymaganiom programu.



Zmienna **query** przechowująca wartość zapytania została rozbita na 3 zmienne, aby ułatwić modyfikację programu w przyszłości oraz ułatwić poszukiwanie ewentualnych błędów.

## Podsumowanie

Kod programu liczy 1432 wiersze. Oprócz modułu **mysql.connector** użyto również modułu **os,** który pozwala na wykorzystywanie natywnych poleceń konsoli windows, takich jak np. **Cls** do czyszczenia konsoli, oraz moduł **date** do używania i formatowania obiektów związanych z datą i czasem. Program możnaby usprawnić na kilka sposobów, na przykład zmieniając rózne czesto używane zwroty na zmienne, które można parametryzować i używać wielokrotnie w różnych częściach programu. Również wykrywanie błędów wprowadzania przez użykownika ogranicza się do wyświetlenia generycznej odpowiedzi. Błędy powinny być wyświetlane w taki sposób, aby użytkownik od razu wiedział, co zrobił źle, czy był to zwykły input error (błąd wprowadzania danych) czy operacja niedozwolona w bazie (błędy składni SQL).

Projekt wymagał znajomości podstaw Pythona, znajomość składni SQL, jak budować poprawne zapytania, jak wykorzystywać zależności między tabelami oraz umiejętność skutecznego połączenia tej wiedzy i zaprezentowania możliwości zarówno Pythona jak i SQL w możliwie przejrzysty i intuicyjny w obsłudze sposób.