



Python

Programowanie baz danych
Zadania



Silnik, napis połączeniowy

Zadanie 1

Połączenie z bazą

Twoja kolej na stworzenie pierwszego silnika!

Silnik to po prostu interfejs do bazy. Informacje, które silnik potrzebuje do nawiązania z bazą zawarte są w napisie połączeniowym. Dla baz sqlite ma on postać np. `sqlite:///example.sqlite`.

"sqlite" w napisie połączeniowym to sterownik (ang. driver), nazywany również konektorem. Konektor określa typ naszej bazy, system zarządzania bazą danych (DBMS- DataBase Management System). "example.sqlite" w napisie połączeniowym to ścieżka do pliku bazodanowego.

Więcej o napisie połączeniowym możesz przeczytać na:

<https://docs.sqlalchemy.org/en/14/core/engines.html#database-urls>

Zadanie 1

Połączenie z bazą

Zainicjalizuj silnik, który podłączysz do bazy sqlite – census.sqlite.
Po utworzeniu połączenia wypisz nazwy tabel znajdujących się w tej bazie używając metody `.table_names()`.

1. Zaimportuj `create_engine` z modułu `sqlalchemy`
2. Używając funkcji `create_engine()`, stwórz silnik do lokalnego pliku `census.sqlite` używając sterownika `sqlite`. Upewnij się, że poprawnie wprowadziłeś napis połączeniowy
3. Wyświetl wyjście metody `table_names()` obiektu `engine`



Odbicie

Zadanie 2

Odbicie tabeli

SQLAlchemy może być używane do automatycznego wczytywania tabeli z bazy danych za pomocą tzw. odbicia (ang. reflection).

Odbicie to proces odczytania bazy i wygenerowania metadanych na podstawie odczytanych informacji. Jest przeciwieństwem ręcznego tworzenia tabel i przydaje się bardzo przy pracy z istniejącymi bazami.

W celu wykonania odbicia należy zaimportować klasę `MetaData` i zainicjalizować obiekt tej klasy. Zainicjalizowany obiekt klasy `MetaData` jest pusty. W trakcie odbijania obiekt klasy `MetaData` zostanie automatycznie wypełniony informacjami o tabelach bazy, więc jedyne co musimy zrobić przed odbiciem to zainicjalizować ten obiektwołając `MetaData()`.

Ponadto potrzebujemy zaimportować klasę `Table`. Używamy jej do odczytania tabeli za pomocą silnika, załadowania kolumn tabeli i wypełnienia metadanych. Realizujemy to poprzez zainicjalizowanie obiektu klasy `Table()`. W celu automatycznego załadowania kolumn za pomocą silnika, przy inicjalizacji obiektu ustawiamy parametry `autoload=True` i `autoload_with=engine`.

W celu wyświetlenia informacji o stworzonym w ten sposób obiekcie używamy funkcji `repr`, ponieważ funkcja `str` zwraca nazwę tabeli. Funkcja `repr` zwróci domyślną reprezentację obiektu. W przypadku obiektu klasy `Table`, będą to informacje o tabeli.

Zadanie 2

Odbicie tabeli

W zadaniu należy odciec tabelę 'census' do zmiennej o nazwie census.

1. Zaimportuj Table i MetaData z sqlalchemy
2. Stwórz obiekt klasy MetaData i przypisz go do zmiennej metadata
3. Odbij tabelę 'census' przy pomocy obiektu Table z parametrami:
 - nazwa tabeli - 'census'
 - obiekt klasy MetaData, który zainicjalizowaliśmy
 - autoload = True
 - parametr autoloaded_with to będzie zainicjalizowany silnik (engine)
4. Wyświetl szczegóły dotyczące tabeli census używając funkcji repr

Zadanie 3

Szczegóły tabeli

Kiedy już odbiliśmy tabelę możemy zacząć sprawdzać informacji o jej strukturze. Do poszczególnych kolumn tabeli odwołujemy się za pomocą atrybutu `.columns` i metody `.keys()`, np. `census.columns.keys()` zwróci listę nazw kolumn tabeli `census`.

Więcej informacji o odbitej tabeli (takich jak typy kolumn) znajdziemy w kontenerze `metadata`. Na przykład informacje o obiektach tabeli są przechowywane w atrybucie `metadata.tables` (`metadata.tables['census']`). Takie odwołanie przyniesie podobny efekt co wywołania funkcji `repr` w poprzednim zadaniu.

W ramach zadania:

1. Odbij tabelę `census` za pomocą obiektu klasy `Table` (identycznie jak zrobiłeś to w poprzednim zadaniu).
2. Wyświetl listę nazw kolumn tabeli `census`, poprzez odwołanie się do metody `keys()` obiektu `census.columns`
3. Wyświetl szczegóły tabeli `census` używając słownika `metadata.tables` i funkcji `repr`. W tym celu odwołaj się do klucza `'census'` słownika `metadata.tables`, a następnie umieść wynik w funkcji `repr`.

Zapytania SQL



Klausura SELECT

Zadanie 4

Klauzula SELECT

Aby mieć dostęp do danych i nimi manipulować musimy najpierw nawiązać połączenie z bazą. Do utworzenia obiektu reprezentującego takie połączenie używamy metody **.connect()** obiektu klasy **Engine**. Funkcja `create_engine()` zwraca instancję silnika, ale sam silnik nie nawiązuje fizycznego połączenia z bazą dopóki nie zostanie wywołana akcja wymagająca połączenia, taka jak np. zapytanie sql.

Na zainicjalizowanym obiekcie połączenia wywołujemy metodę **.execute()** przekazując do niej jako parametr zapytanie sql. Metoda **.execute** zwraca obiekt klasy **ResultProxy**. Ostatecznie to tego obiektu używamy do wykonania zapytania na bazie. Zapytanie zostanie wykonane na bazie po wywołaniu metody **.fetchall()** obiektu klasy **ResultProxy**. Metoda **.fetchall()** powoduje nawiązanie fizycznego połączenia z bazą, wykonanie na bazie odpowiedniego zapytania i zwraca obiekt klasy **ResultSet**, który zawiera odpowiedź bazy na nasze zapytanie.

Zadanie 4

Klauzula SELECT

W tym zadaniu użyj tradycyjnego zapytania sql. Zauważ, że kiedy wykonujesz zapytanie używając surowego sql bezpośrednio odpytujesz bazę danych. W szczególności nie jest wymagane żadne odciecie.

Twoim zadaniem jest bezpośrednie odpytanie bazy.

1. Użyj metody `.connect` silnika, żeby utworzyć obiekt reprezentujący połączenie z bazą.
2. Zbuduj zapytanie sql o wszystkie wiersze tabeli census i przypisz je do zmiennej `stmt`. Pamiętaj, że twoje zapytanie musi być napisem.
3. Użyj metod `.execute()` i `fetchall()`. Wynik zapytania przypisz do zmiennej `results`. Pamiętaj, że `.execute()` musi zostać wykonana przed `.fetchall()` i że zapytanie (`stmt`) musi zostać przekazane do metody `.execute()`
- 4 .Wyświetl wynik.



Funkcja select

Zadanie 5

Funkcja select

SQLAlchemy dostarcza obiektowej składni komunikacji z bazą danych. W poprzednim zadaniu używając surowego sql wypytywałeś bazę bezpośrednio. Ale w SQLAlchemy istnieją inne narzędzia dzięki, którym nie musimy posługiwać się składnią relacyjną.

W SQLAlchemy przechodzimy przez obiekt Table i to SQLAlchemy odpowiada za przetłumaczenie naszego zapytania (kodu) na odpowiedni sql. W ten sposób zamiast martwić się szczegółami różnych dialektami sql (np. MySQL vs. PostgreSQL) możemy użyć Pythonowego frameworku i jemu pozostawić zawłośc dialektów sql.

W tym zadaniu ponownie zbudujesz zapytanie select o wszystkie wiersze z tabeli census. Tym razem użyjesz jednak funkcji select biblioteki SQLAlchemy. Ta metoda przyjmuje listę tabel lub kolumn, np. `select([nazwa_tabeli])`. Z ResultProxy wyciągniemy tylko kilka pierwszych wpisów używając metody `.fetchmany()` z argumentem `size` wskazującym liczbę rekordów, które chcemy pobrać z bazy.

Zadanie 5

Funkcja select

1. Zaimportuj select z SQLAlchemy
2. Odbij tabelę census (dwa pierwsze etapy zostały już zaimplementowane)
3. Używając funkcji select() stwórz zapytanie o wszystkie wpisy w tabeli census. W tym celu przekaz do metody select jednoelementową listę z tabelą census.
4. Wyświetl stmt. Popatrz na sql, który został wygenerowany przez SQLAlchemy.
5. Pobierz 10 wpisów i przypisz je do zmiennej results. W tym celu użyj metody execute() na połączeniu z argumentem stmt oraz użyj metody fetchmany() obiektu klasy ResultProxy z odpowiednią wartością parametru size w celu otrzymania obiektu klasy ResultSet.



Dodatki