Python i bazy danych

Paweł Gliwny

ORM - Mapowanie Obiektowo-Relacyjne

- Mapowanie obiektowo-relacyjne.
- Technika łączenia programowania obiektowego z bazami danych relacyjnych.
- Pozwala pracować z danymi w sposób naturalny, korzystając z obiektów i ich relacji, zamiast pisać zawiłe zapytania SQL.

Zasada działania ORM

- **Definicja modelu:** tworzymy klasy Pythona reprezentujące encje w bazie danych (np. klasa *Użytkownik* reprezentuje tabelę *użytkownicy*).
- **Mapowanie:** ORM automatycznie mapuje klasy na tabele i kolumny w bazie danych.
- **Interakcja z danymi:** Operujemy na obiektach, a ORM tłumaczy te operacje na odpowiednie zapytania SQL i aktualizuje bazę danych.

Zalety ORM

- **Zwiększona produktywność:** Programowanie staje się prostsze i szybsze, ponieważ nie trzeba pisać ręcznie zapytań SQL.
- **Poprawa czytelności kodu:** Kod jest bardziej zrozumiały i łatwiejszy w utrzymaniu, ponieważ operacje na danych są wyrażane w języku obiektowym.

Przykładowe bibliotek ORM w Pythonie

- **SQLAlchemy** - popularna i rozbudowana biblioteka, oferująca wiele funkcji i dużą elastyczność.
- **Peewee** - prostsza i lżejsza biblioteka, łatwa do nauczenia i użycia.

SQLAlchemy

- Zestaw narzędzi Python, który pozwala uzyskiwać dostęp i zarządzać bazami danych SQL za pomocą Pythona.
- Możesz pisać zapytania w formie łańcucha znaków lub łączyć obiekty Python dla podobnych zapytań.
- Praca z obiektami zapewnia elastyczność i pozwala tworzyć wysoce wydajne aplikacje oparte na SQL.

Engine w SQLAlchemy

- **Engine** w SQLAlchemy jest kluczowym elementem, który odpowiada za **łączenie z bazą danych** i **wykonywanie operacji SQL**. Można go postrzegać jako **interfejs** pomiędzy kodem Python a bazą danych.

Funkcje Engine:

- **Tworzenie połączenia z bazą danych:** Engine przyjmuje parametry połączenia, takie jak typ bazy danych, adres serwera, nazwę użytkownika i hasło.
- **Wykonywanie zapytań SQL:** Engine może uruchamiać dowolne zapytanie SQL, takie jak `SELECT`, `INSERT`, `UPDATE` i `DELETE`.
- **Przetwarzanie wyników zapytań:** Engine zwraca wyniki zapytań w postaci obiektów Python, które można łatwo manipulować i analizować.

Zalety używania Engine:

- **Uniwersalność:** Engine obsługuje wiele różnych typów baz danych, co czyni go narzędziem uniwersalnym.
- **Wydajność:** Engine jest zoptymalizowany pod kątem wydajności i zapewnia szybkie wykonywanie operacji SQL.
- **Elastyczność:** Engine można używać w różnych kontekstach, np. w aplikacjach webowych, skryptach i frameworkach Python.

- Można wyobrazić sobie Engine jako **sterownika** do samochodu. Sterownik umożliwia **interakcję** z silnikiem i innymi elementami samochodu, ale nie jest on sam w sobie silnikiem. Silnik to **mechanizm**, który wykonuje pracę.

Połączenia z bazą danych

- **Engine:** wspólny interfejs do bazy danych z SQLAlchemy
- **Connection string:** Wszystkie szczegóły wymagane do znalezienia bazy danych (oraz logowania, jeśli jest to konieczne)

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///census_nyc.sqlite')
connection = engine.connect()
```

Co jest w Twojej bazie danych?

- Zanim zaczniesz ją przeszukiwać, będziesz chciał wiedzieć, co się w niej znajduje: na przykład jakie są tabele.

```
from sqlalchemy import create _ engine, inspect
engine = create _ engine('sqlite:///census _ nyc.sqlite')

inspector = inspect(engine)
table_names = inspector.get_table_names()
print(table_names)
```

Reflection

- Odczytuje bazę danych i buduje obiekt **Tabel**

```
metadata = MetaData()
census = Table('census', metadata, autoload_with=engine)
print(repr(census))
```

Instrukcje SQL

- Wybieranie, wstawianie, aktualizacja i usuwanie danych
- Tworzenie i modyfikowanie danych

SELECT column _ name FROM table _ name

- SELECT pop2008 FROM People
- SELECT * FROM People

Tworzenie zapytań SQL w SQLAlchemy

```
stmt = text('SELECT state, sex FROM census')
result_proxy = connection.execute(stmt)
results = result_proxy.fetchall()
```

Tworzenie zapytań w sposób pythonowy

Tworzenie obiektu MetaData

```
>> metadata = MetaData()
```

Tworzenie silnika (engine) do połączenia z bazą danych

```
>> engine = create_engine('sqlite:///census.sqlite')
```

Powiązanie tabeli z metadanymi

```
>> census = Table('census', metadata, autoload_with=engine)
```

Polecenie func

- **`func`** jest narzędziem w SQLAlchemy do używania funkcji bazodanowych, takich jak **`sum`**, **`avg`**, **`count`**, itp.
- Umożliwia stosowanie funkcji bazodanowych w zapytaniach SQLAlchemy.

```
stmt = select(
    func.sum(census.c.pop2008)
)
results = connection.execute(stmt).fetchall()
print(results)
```

Wyrażenie **`where`**

- Wyrażenie **where** służy do filtrowania danych w zapytaniach.
- Pozwala na ustalenie warunków, które muszą zostać spełnione, aby wziąć pod uwagę rekordy w wyniku zapytania.

```
stmt = select(
    func.sum(census.c.pop2008)
).where(census.c.sex == 'F')
```

CRUD (Create, Read, Update, Delete)

- **CRUD** to akronim oznaczający **Create, Read, Update, Delete** (tworzenie, odczytywanie, aktualizowanie, usuwanie).
- Te cztery operacje stanowią podstawę interakcji z bazami danych i są stosowane w niemal każdym systemie opartym na danych.
- **SQLAlchemy 2.0** wprowadza nowe podejście do konstruowania zapytań, które jest bardziej deklaratywne i bardziej zgodne z *modern Python coding practices*.

SQLAlchemy

- Źródło: https://www.datacamp.com/tutorial/sqlalchemy-tutorial-examples

```
engine = create_engine('sqlite+pysqlite:///:memory:',
echo=True, future=True)
```

- **Silnik bazy danych**: `sqlite+pysqlite` oznacza użycie bazy danych SQLite przy pomocy biblioteki pysqlite (która jest domyślnym sterownikiem SQLite w Pythonie).
- **Adres URL bazy danych**: `'sqlite+pysqlite:///:memory:'` to ścieżka URL. Fragment `:memory:` oznacza, że baza danych zostanie stworzona w pamięci RAM komputera, co oznacza, że jest tymczasowa i nie zostanie zapisana na dysku twardym. Po zakończeniu sesji Pythona, dane przechowywane w tej bazie danych zostaną utracone.
- **echo**: Parametr `echo=True` w funkcji `create_engine` mówi SQLAlchemy, aby logowało wszystkie generowane instrukcje SQL. Jest to przydatne do debugowania, gdyż pozwala zobaczyć dokładnie, jakie zapytania SQL są wykonywane.
- **future**: Parametr `future=True`, który jest stosunkowo nowym dodatkiem w SQLAlchemy, włącza nowe, bardziej przyszłościowo zorientowane zachowania i API, które będą preferowane w przyszłych wersjach SQLAlchemy. Przykładowo, wprowadza zmiany w API sesji oraz obsługi transakcji.
- Ten kod jest bardzo przydatny do celów testowych lub demonstracyjnych, gdyż można szybko stworzyć bazę danych w pamięci, pracować na niej, a potem po prostu zostawić ją do automatycznego usunięcia po zamknięciu aplikacji.