

Python średnio zaawansowany

Dzień 9



Blok nr 3:

Przetwarzanie danych

AGENDA

- Object Relational Mapping
- SQLAlchemy

Object Relational Mapping

ORM - definicja

Mapowanie obiektowo - relacyjne polega na **mapowaniu** danych w postaci **tabelarycznej** (relacji w bazie danych) na **obiekty** i odwrotnie.

To nowoczesne podejście do zagadnienia współpracy z bazą danych, wykorzystujące filozofię programowania obiektowego.

Wady SQL

- Zależność od silnika bazodanowego
- Podatny na ataki typu SQL injection
- Utrudnione aktualizowanie kodu podczas zmiany struktury tabel
- Boilerplate code
- Kod SQL wymieszany z kodem aplikacji

* więcej nt boilerplate code: <https://stormit.pl/boilerplate-code/>

Zalety ORM

- Niezależność od silnika bazodanowego
- Większa produktywność programisty
- Migracje ułatwiają zmianę struktury tabel
- Możliwość wstrzyknięcia kodu SQL jeżeli ten generowany przez ORM jest zbyt skomplikowany/wolny

Wady ORM

- Skomplikowane zapytania SQL będą wymagać zapisania ich wprost, bez wykorzystania ORM
- Mechanizm ORM może być powolny

SQLAlchemy

SQLAlchemy



SQLAlchemy – moduł dla języka Python służący do pracy z bazami danych oraz mapowania obiektowo-relacyjnego.

Składa się z dwóch odrębnych komponentów: Core oraz ORM.

Wspiera takie bazy danych jak: Firebird, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SQLite.

SQLAlchemy



Z SQLAlchemy korzystają m.in.: Fedora Project, Freshbooks, Fundacja Mozilla, OpenStack, reddit, SurveyMonkey oraz Yelp.



Dobry wstęp do SQLAlchemy:

<https://docs.sqlalchemy.org/en/latest/orm/tutorial.html>

SQLAlchemy - analogia do SQL

Postać SQL

```
SELECT * FROM klienci;
```

Postać ORM

```
Klient.query.all()
```



```
SELECT * FROM klienci
```

```
WHERE age > 40;
```



```
Klient.query.filter(age > 40)
```

SQLAlchemy - analogia do SQL

Postać SQL

```
SELECT COUNT(*) FROM  
klienci;
```

Postać ORM

```
Klient.query.count()
```



Więcej informacji nt filtrowania:

<https://docs.sqlalchemy.org/en/latest/orm/tutorial.html#common-filter-operators>

SQLAlchemy - przykład

```
class Adres(Base):
```

```
    __tablename__ = 'adres'
```

```
    idx = Column(Integer, primary_key=True)
```

```
    nazwa = Column(String(250))
```

```
    adres = Column(String(250))
```

```
    klient_id = Column(
```

```
        Integer,
```

```
        ForeignKey('klienci.idx')
```

```
)
```

```
    klient = relationship(Klient)
```

Klucz obcy

```
class Klient(Base):
```

```
    __tablename__ = 'klienci'
```

```
    idx = Column(Integer, primary_key=True)
```

```
    imie = Column(String(250), nullable=False)
```

```
    nazwisko = Column(String(250), nullable=False)
```

* przykład na relację „wielu:1”, bez relacji dwukierunkowej

SQLAlchemy - klucz obcy

```
adres = Adres.query.first()
```

```
adres.klient
```

← obiekt powiązany relacją

```
adres.klient_id
```

← wartość w kolumnie
klient_id, klucz obcy

```
adres.klient.imie
```

← wartość w kolumnie *imie* w
tabeli powiązanej relacją

Dzięki!

