Programowanie i projektowanie obiektowe

SQLAlchemy

Paweł Daniluk

Wydział Fizyki

Jesień 2013

Relacje

- Relacja to podzbiór iloczynu kartezjańskiego
- Dwuargumentowa $R \subset A \times B$
- n-argumentowa $R \subset A_1 \times A_2 \times \ldots A_n$
- Reprezentacja jako dwuwymiarowa tabela

Relacje

- Relacja to podzbiór iloczynu kartezjańskiego
- Dwuargumentowa $R \subset A \times B$
- n-argumentowa $R \subset A_1 \times A_2 \times \ldots A_n$
- Reprezentacja jako dwuwymiarowa tabela

Relacja *Filmy*

tytuł	rok	długość	typFilmu
Gwiezdne Wojny	1977	124	kolor
Potężne Kaczory	1991	104	kolor
Świat Wayne'a	1992	95	kolor

Relacje c.d.

Atrybuty

tytuł, *rok*, *długość*, *typFilmu*

Schemat

Filmy(*tytuł*, *rok*, *długość*, *typFilmu*)

Dziedziny

- *tytuł* – tekst
- *rok*, *długość* – liczby całkowite
- *typFilmu* – $\{\text{kolor}, \text{czarno-biały}\}$

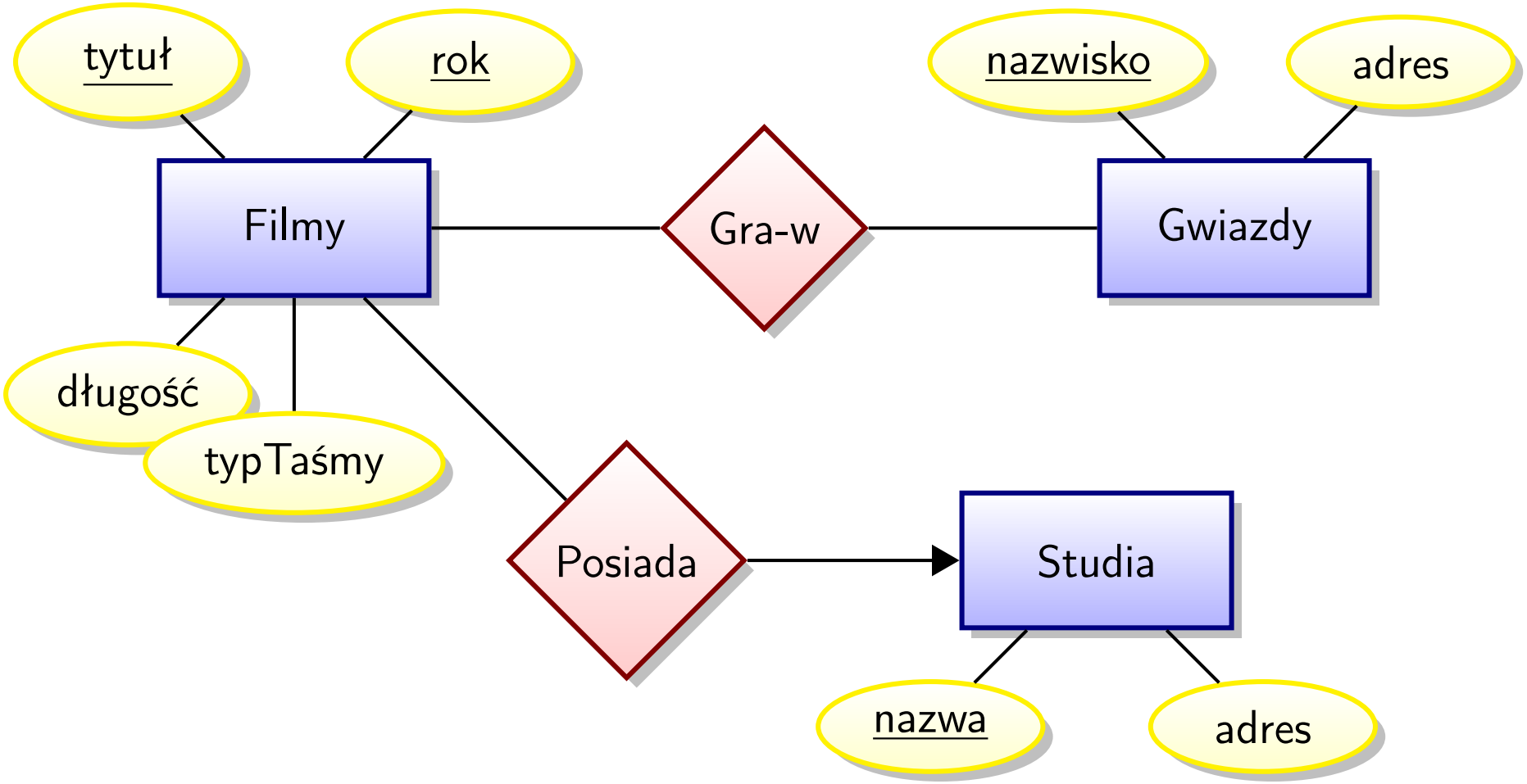
Schemat

- Nazwy zbiorów w iloczynie kartezjańskim to *atrybuty*
- Nazwa relacji i zbiór uporządkowany jej atrybutów tworzą *schemat relacji*
- Projekt relacyjnej bazy danych zawiera zazwyczaj kilka relacji
- Zbiór schematów relacji projektu nazywamy *schematem bazy danych*
- Wiersze tabeli (poza nagłówkowym) – elementy relacji – nazywane są *krotkami*

Dziedziny (typy danych)

- Każda składowa relacji ma swój typ danych
- Dostępne typy danych zależą od implementacji DBMS
- Najczęściej używane to
  - liczba całkowita
  - liczba rzeczywista
  - znak
  - łańcuch znaków
  - data
  - boolean

Schemat związków encji



Schemat relacyjny

Zbiory encji $\rightarrow$ relacje

- *Filmy*(*tytuł*, *rok*, *długość*, *typFilmu*)
- *Gwiazdy*(*nazwisko*, *adres*)
- *Studia*(*nazwa*, *adres*)

Związki $\rightarrow$ relacje

- *Posiada*(*tytuł*, *rok*, *nazwaStudia*)
- *Gra-w*(*tytuł*, *rok*, *nazwiskoGwiazdy*)

Relacje odpowiadające związkom

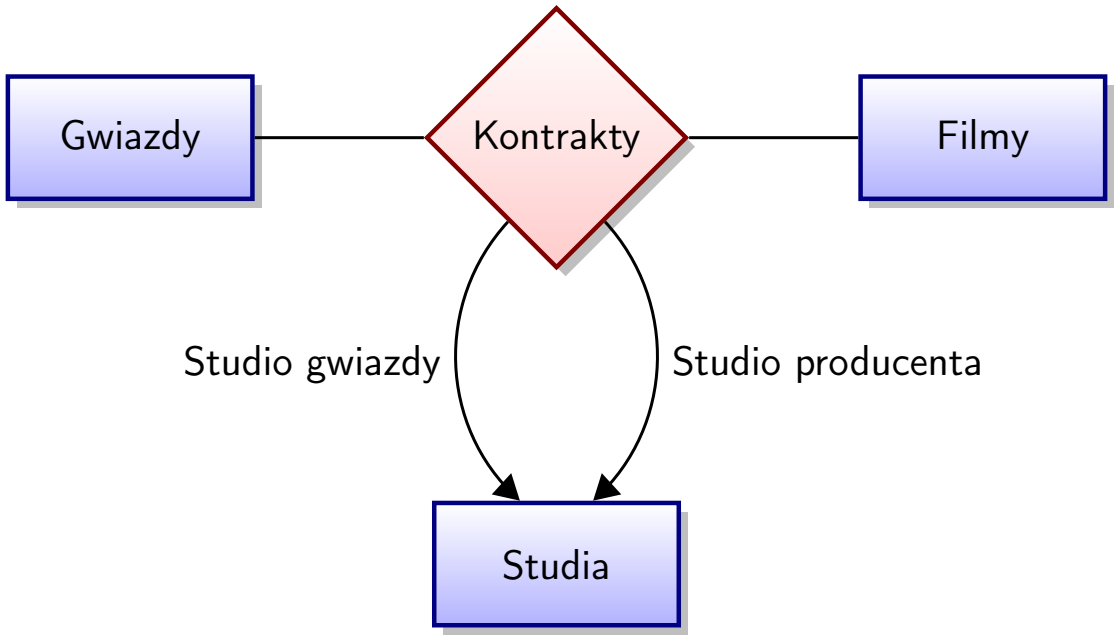
Posiada – wiele do jeden

tytuł	rok	nazwaStudia
Gwiezdne Wojny	1977	Fox
Potężne Kaczory	1991	Disney
Świat Wayne'a	1992	Paramount

Gra-w – wiele do wiele

tytuł	rok	nazwiskoGwiazdy
Gwiezdne Wojny	1977	Carrie Fisher
Gwiezdne Wojny	1977	Mark Hamill
Gwiezdne Wojny	1977	Harrison Ford
Potężne Kaczory	1991	Emilio Estevez
Świat Wayne'a	1992	Dana Carvey
Świat Wayne'a	1992	Mike Meyers

Relacje odpowiadające związkom c.d.



Kontrakty(*nazwiskoGwiazdy*, *tytuł*, *rok*, *studioGwiazdy*, *studioProducenta*)

Podejście obiektowe

Problem

Korzystanie z SQL może być kłopotliwe. Formułowanie zapytań i konwersje odpowiedzi do strawnego formatu jest bardzo żmudne.

Pomysł

Zdefiniujmy mapowanie encji z bazy danych na obiekty w Pythonie.

Podejście obiektowe

Problem

Korzystanie z SQL może być kłopotliwe. Formułowanie zapytań i konwersje odpowiedzi do strawnego formatu jest bardzo żmudne.

Pomysł

Zdefiniujmy mapowanie encji z bazy danych na obiekty w Pythonie.

SQLAlchemy

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.
It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

SQLAlchemy

1. relacje $\longleftrightarrow$ klasy
2. encje(krotki) $\longleftrightarrow$ obiekty
3. atrybuty relacji $\longleftrightarrow$ atrybuty obiektów
4. związki $\longleftrightarrow$ atrybuty obiektów
5. definicja relacji (CREATE TABLE) $\longleftrightarrow$ atrybuty klasowe
6. zapytania SQL $\longleftrightarrow$ metoda query
7. wstawianie krotek (INSERT) $\longleftrightarrow$ tworzenie obiektów i dodawanie do sesji

Klasy odpowiadające relacjom mogą mieć dowolne metody.

Baza danych działa trochę jak repozytorium obiektów.

Przykłady

Tutorial

http://docs.sqlalchemy.org/en/rel_0_8/orm/tutorial.html

Start

```
>>> import sqlalchemy
>>> sqlalchemy.__version__
0.8.0
```

Połączenie z bazą danych

```
>>> engine=sqlalchemy.create_engine('sqlite:///:memory:', echo=True)
>>> engine.execute("SELECT 1").scalar()
2013-12-18 23:43:47,672 INFO sqlalchemy.engine.base.Engine SELECT 1
2013-12-18 23:43:47,672 INFO sqlalchemy.engine.base.Engine ()
1
>>>
```

W przykładach używamy SQLite.

MySQL

`mysql://login:haslo@serwer/baza`

Mapowanie

```
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()
```

```
from sqlalchemy import Column, Integer, String
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    password = Column(String)

    def __init__(self, name, fullname, password):
        self.name = name
        self.fullname = fullname
        self.password = password

    def __repr__(self):
        return "<User('\%s','\%s', '\%s')>" \% (self.name, self.
```

Tworzenie tabel

```
>>> Base.metadata.create_all(engine)
2013-12-19 00:02:39,729 INFO sqlalchemy.engine.base.Engine PRAGMA table_inf
2013-12-19 00:02:39,730 INFO sqlalchemy.engine.base.Engine ()
2013-12-19 00:02:39,730 INFO sqlalchemy.engine.base.Engine
CREATE TABLE users (
    id INTEGER NOT NULL,
    name VARCHAR,
    fullname VARCHAR,
    password VARCHAR,
    PRIMARY KEY (id)
)


2013-12-19 00:02:39,730 INFO sqlalchemy.engine.base.Engine ()
2013-12-19 00:02:39,730 INFO sqlalchemy.engine.base.Engine COMMIT
>>>
```

Tworzenie krotek/obiektów

```
>>> ed_user = User('ed', 'Ed Jones', 'edspassword')
>>> ed_user.name
'ed'
>>> ed_user.password
'edspassword'
>>> str(ed_user.id)
'None'
```

Sesje

- Operacje na bazie danych odbywają się za pośrednictwem sesji.
- Sesja przechowuje informacje o obiektach, które były przywołane z bazy oraz stworzone (i dodane do sesji).
- Zapytania wydane podczas sesji zwracają dane zgodne ze stanem bazy danych, a nie sesji.
- Zamknięcie sesji albo wykonanie zapytania zazwyczaj wiąże się zapisaniem zmian w BD.

Generator sesji

```
>>> from sqlalchemy.orm import sessionmaker
>>> Session = sessionmaker(bind=engine)
```

Tworzenie sesji

```
>>> session = Session()
```

Dodawanie obiektu do bazy

Dodawanie do sesji

```
>>> ed_user = User('ed', 'Ed Jones', 'edspassword')
>>> session.add(ed_user)
```

Zapytanie

```
>>> our_user = session.query(User).filter_by(name='ed').first()
>>> our_user
<User('ed','Ed Jones', 'edspassword')>
>>> ed_user is our_user
True
```

Więcej obiektów

Kilka obiektów na raz

```
>>> session.add_all([
...     User('wendy', 'Wendy Williams', 'foobar'),
...     User('mary', 'Mary Contrary', 'xxg527'),
...     User('fred', 'Fred Flinstone', 'blah')])
```

Zmiana atrybutu

```
>>> ed_user.password = 'f8s7ccs'
```

Krotki zmienione

```
>>> session.dirty
IdentitySet([<User('ed','Ed Jones', 'f8s7ccs')>])
```

Krotki dodane

```
>>> session.new
IdentitySet([<User('wendy','Wendy Williams', 'foobar')>,
<User('mary','Mary Contrary', 'xxg527')>,
<User('fred','Fred Flinstone', 'blah')>])
```

Sesję można wycofać

Głupie zmiany

```
>>> ed_user.name = 'Edwardo'
>>> fake_user = User('fakeuser', 'Invalid', '12345')
>>> session.add(fake_user)
```

Są

```
>>> session.query(User).filter(User.name.in_(['Edwardo', 'fakeuser'])).all(
[<User('Edwardo','Ed Jones', 'f8s7ccs')>, <User('fakeuser','Invalid', '1234
```

```
>>> session.rollback()
```

Nie ma

```
>>> ed_user.name
u'ed'
>>> fake_user in session
False
>>> session.query(User).filter(User.name.in_(['ed', 'fakeuser'])).all()
[<User('ed','Ed Jones', 'f8s7ccs')>]
```

Proste zapytania

Wyjmowanie obiektów

```
>>> for instance in session.query(User).order_by(User.id):
...     print instance.name, instance.fullname
ed Ed Jones
wendy Wendy Williams
mary Mary Contrary
fred Fred Flinstone
```

Wyjmowanie krotek

```
>>> for name, fullname in session.query(User.name, User.fullname):
...     print name, fullname
ed Ed Jones
wendy Wendy Williams
mary Mary Contrary
fred Fred Flinstone
```

Filtrowanie

```
>>> for name, in session.query(User.name). ...                filter_by(fullna
...     print name
ed
```

```
>>> for name, in session.query(User.name). ...                filter(User.full
...     print name
ed
```

Wielokrotne

```
>>> for user in session.query(User). ...            filter(User.name=='ed').
...           filter(User.fullname=='Ed Jones'):
...     print user
<User('ed','Ed Jones', 'f8s7ccs')>
```

Standardowe operatory

- `query.filter(User.name == 'ed')`
- `query.filter(User.name != 'ed')`
- `query.filter(User.name.like('\%ed\%'))`
- `query.filter(User.name.in_(['ed', 'wendy', 'jack']))`
- `query.filter(User.name.in_(session.query(User.name).filter(User.name.li`
- `query.filter(~User.name.in_(['ed', 'wendy', 'jack']))`
- `query.filter(User.name == None)`
- `query.filter(User.name != None)`
- `query.filter(sqlalchemy.and_(User.name == 'ed', User.fullname == 'Ed␣Jo`
- `query.filter(User.name == 'ed').filter(User.fullname == 'Ed␣Jones')`
- `query.filter(sqlalchemy.or_(User.name == 'ed', User.name == 'wendy'))`

all(), first(), one()

Wszystkie krotki

```
>>> query = session.query(User).filter(User.name.like('%ed')).order_by(User
>>> query.all()
[<User('ed','Ed Jones', 'f8s7ccs')>, <User('fred','Fred Flinstone', 'blah')
```

Tylko pierwsza

```
>>> query.first()
<User('ed','Ed Jones', 'f8s7ccs')>
```

all(), first(), one() c.d.

Dokładnie jedna

```
>>> from sqlalchemy.orm.exc import MultipleResultsFound
>>> try:
...     user = query.one()
... except MultipleResultsFound, e:
...     print e
Multiple rows were found for one()
```

```
>>> from sqlalchemy.orm.exc import NoResultFound
>>> try:
...     user = query.filter(User.id == 99).one()
... except NoResultFound, e:
...     print e
No row was found for one()
```

Związki

```
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship, backref

class Address(Base):
    __tablename__ = 'addresses'
    id = Column(Integer, primary_key=True)
    email_address = Column(String, nullable=False)
    user_id = Column(Integer, ForeignKey('users.id'))

    user = relationship("User",
                backref=backref('addresses', order_by=id))

    def __init__(self, email_address):
        self.email_address = email_address

    def __repr__(self):
        return "<Address('\%s')>" \% self.email_address
```

Alternatywnie

```
class User(Base):
    #
```

Operacje na związkach

```
>>> jack = User('jack', 'Jack Bean', 'gjffdd')
>>> jack.addresses
[]
```

```
>>> jack.addresses = [
...                 Address(email_address='jack@google.com'),
...                 Address(email_address='j25@yahoo.com')]
```

Związki działają bez SQLa (obiekty nie są jeszcze dodane do sesji)

```
>>> jack.addresses[1]
<Address('j25@yahoo.com')>

>>> jack.addresses[1].user
<User('jack','Jack Bean', 'gjffdd')>
```

Operacje na związkach

Całość dodaje się automatycznie

```
>>> session.add(jack)
SQL>>> session.commit()
```

Dostęp

```
>>> jack = session.query(User). ... filter_by(name='jack').one()
>>> jack
<User('jack','Jack Bean', 'gjffdd')>
```

Dopiero teraz załadują się adresy

```
>>> jack.addresses
[<Address('jack@google.com')>, <Address('j25@yahoo.com')>]
```

Więcej możliwości

- zapytania ze złączeniami
- podzapytania
- związki "wiele do wielu"
- transakcje

Zadanie 1 – Tutorial

Zadanie

Zapoznać się z tutorialem do SQLAlchemy.

Zadanie 2 – Pracownicy/etaty

Zadanie

Odtwórz bazę "Pracownicy" podaną jako przykład na przedmiocie Bazy danych. Przetestuj wykonywanie zapytań.