Programowanie i projektowanie obiektowe

Przechowywanie obiektów

Paweł Daniluk

Wydział Fizyki

Jesień 2014

Przechowywanie danych

Żaden poważny system informatyczny nie może działać bez składowania danych na trwałych nośnikach.

Dane można zapisywać:

- w plikach (tekstowych lub binarnych)
- w relacyjnej bazie danych
- na zdalnym serwerze (przy użyciu mniej lub bardziej standardowego protokołu)

Bardzo pożądane są metody pozwalające na zapisywanie o odczytywanie całych obiektów, które występują w systemie. Najlepiej razem z powiązaniami pomiędzy nimi.

Python pickle

Serializacja obiektów - za Wikipedią

W programowaniu komputerów proces przekształcania obiektów, tj. instancji określonych klas, do postaci szeregowej, czyli w strumień bajtów, z zachowaniem aktualnego stanu obiektu. Serializowany obiekt może zostać utrwalony w pliku dyskowym, przesłany do innego procesu lub innego komputera poprzez sieć. Procesem odwrotnym do serializacji jest deserializacja. Proces ten polega na odczytaniu wcześniej zapisanego strumienia danych i odtworzeniu na tej podstawie obiektu klasy wraz z jego stanem bezpośrednio sprzed serializacji.

W Pythonie służy do tego (m. in.) moduł `pickle`.

pickle przykład

Serializacja

```
In [1]: class A:
   ...:     pass
   ...:

In [2]: a = A()

In [3]: a.a = 1

In [4]: a.b = 2

In [5]: import pickle

In [6]: f = open('plik', 'w')

In [7]: pickle.dump(a, f)

In [8]: f.close()
```

pickle przykład

plik

```
(i__main__
A
p0
(dp1
S'a'
p2
I1
sS'b'
p3
I2
sb.
```

Deserializacja

```
In [9]: f = open('plik', 'r')

In [10]: b = pickle.load(f)

In [11]: b.__dict__
Out[11]: {'a': 1, 'b': 2}
```

pickle c.d.

Uwagi

- Sieci powiązań pomiędzy obiektami są poprawnie serializowane.
- Reprezentacja obiektu nie zawiera opisu klasy.
- Python gwarantuje kompatybilność wsteczną z danymi serializowanymi przy pomocy poprzednich wersji.
- Nie obejmuje to kwestii zmiany definicji klas w aplikacji.
- Działa tylko z Pythonem.

Zastosowania

- Zapisywanie danych ad-hoc.
- Przesyłanie obiektów przez sieć.

XML

eXtensible Markup Language

Znaczniki

- określają znaczenie podciągów znaków w dokumencie
- teksty ujęte w nawiasy kątowe <...>
- występują w parach – otwierający <...> i zamykający </...>

Zastosowania

- Przechowywanie ustrukturyzowanych danych w plikach tekstowych
- Wymiana danych pomiędzy aplikacjami

Tryby dokumentów XML

Dobrze sformowany XML

- Podejście semistrukturalne
- Dowolne znaczniki
- Brak ustalonego schematu

Ustalony typ dokumentu

- Podejście pośrednie pomiędzy schematem semistrukturalnym, a ścisłymi (np. relacyjnym)
- **D**ocument **T**ype **D**efinition
- Specyfikacja dopuszczalnych znaczników
- Gramatyka zagnieżdżania

Dobrze sformowany XML

Przykład

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<etaty>
    <etat>
        <nazwa>Profesor</nazwa>
        <placa_od>3000</placa_od>
        <placa_do>6000</placa_do>
    </etat>
    <etat>
        ...
    </etat>
</etaty>
```

Dokument ustalony

Dokument bez specyfikacji typu

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
```

Nagłówek

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE typDokumentu SYSTEM "specyfikacja.dtd">
```

Dokument ustalony c.d.

Przykład

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Etaty SYSTEM "etaty.dtd">
<etaty>
    <etat>
        <nazwa>Profesor</nazwa>
        <placa_od>3000</placa_od>
        <placa_do>6000</placa_do>
    </etat>
    <etat>
        ...
    </etat>
</etaty>
```

Dokument ustalony c.d.

- Deklaracja elementu głównego (korzenia)
- Deklaracje elementów
- Deklaracje reguł zagnieżdżania

etaty.dtd

```
<!DOCTYPE etaty [
    <!ELEMENT etaty (etat*)>
    <!ELEMENT etat (nazwa, placa_od, placa_do)>
    <!ELEMENT nazwa (#PCDATA)>
    <!ELEMENT placa_od (#PCDATA)>
    <!ELEMENT placa_do (#PCDATA)>
]>
```

Dokument ustalony c.d.

- Nazwa typu dokumentu: `<!DOCTYPE [...]>`
- Nazwy dopuszczalnych elementów `<!ELEMENT (...)>`
- Reguły zagnieżdżania dopuszczalnych składowych elementów
- element (podelement1*, podelement2+, podelement3?,...)
- Operatory:
  - *: 0 lub więcej
  - +: 1 lub więcej
  - ?: co najwyżej raz
- element (`#PCDATA`)
- Typ `#PCDATA` oznacza dowolny tekst
- Nie występują typy elementów

Przykład

Model relacyjny

Gwiazdy(nazwisko, adres)
Filmy(tytul, rok, dlugosc)
GwiazdyW(tytul, rok, nazwiskoGwiazdy)

Dokument DTD

```
<!DOCTYPE Gwiazdy [
    <!ELEMENT gwiazdy (gwiazda*)>
    <!ELEMENT gwiazda (nazwisko, adres+, filmy)>
    <!ELEMENT nazwisko (#PCDATA)>
    <!ELEMENT adres (\#PCDATA)>
    <!ELEMENT filmy (film*)>
    <!ELEMENT film (tytul, rok, dlugosc)>
    <!ELEMENT tytul (\#PCDATA)>
    <!ELEMENT rok (\#PCDATA)>
    <!ELEMENT dlugosc (#PCDATA)>
]>
```

Przykładowe dane

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Gwiazdy SYSTEM "gwiazdy.dtd">
<gwiazdy>
    <gwiazda>
        <nazwisko>Carrie Fischer</nazwisko>
        <adres>123 Maple St.</adres>
        <filmy>
            <film>
                <tytul>Gwiezdne Wojny</tytul>
                <rok>1977</rok>
                <dlugosc>93</dlugosc>
            </film>
            <film>
                <tytul>Imperium Kontratakuje</tytul>
                <rok>1980</rok>
                <dlugosc>96</dlugosc>
            </film>
            <film>
                <tytul>Powrot Jedi</tytul>
                <rok>1983</rok>
                <dlugosc>94</dlugosc>
```

Przykładowe dane

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Gwiazdy SYSTEM "gwiazdy.dtd">
<gwiazdy>
    <gwiazda>
        <nazwisko>Carrie Fischer</nazwisko>
        <adres>123 Maple St.</adres>
        <filmy>
            <film>
                <tytul>Gwiezdne Wojny</tytul>
                <rok>1977</rok>
                <dlugosc>93</dlugosc>
            </film>
            <film>
                <tytul>Imperium Kontratakuje</tytul>
                <rok>1980</rok>
                <dlugosc>96</dlugosc>
            </film>
            <film>
                <tytul>Powrot Jedi</tytul>
                <rok>1983</rok>
                <dlugosc>94</dlugosc>
            </film>
        </filmy>
    </gwiazda>
    <gwiazda>
        <nazwisko>Mark Hamill</nazwisko>
        <adres>456 Oak Rd.</adres>
        <adres>789 Pine Av.</adres>
        <filmy>
            <film>
                <tytul>Imperium Kontratakuje</tytul>
                <rok>1980</rok>
                <dlugosc>96</dlugosc>
            </film>
```

Dokument ustalony c.d.

- Zagnieżdżanie znaczników nie pozwala przedstawić wszystkich informacji
- Atrybuty pozwalają na dodatkowy opis danych
- Zmniejszenie redundancji
- Mogą służyć do powiązania pojedynczej wartości ze znacznikiem
- Alternatywa dla podznaczników, które są zwykłymi tekstami (`#PCDATA`)

Składnia

- Po deklaracji elementu `<!ELEMENT (...)>`
- `<!ATTLIST element ...>`
- lista atrybutów
  - atrybut1
  - atrybut2 `ID`
  - atrybut3 `IDREF` lub `IDREFS`

Atrybuty – przykład

Dokument DTD

```
<!DOCTYPE Gwiazdy-Filmy [
    <!ELEMENT gwiazdy-filmy (gwiazda*, film*)>
    <!ELEMENT gwiazda (nazwisko, adres+)>
        <!ATTLIST gwiazda
        gwiazdaId ID
        wystepujeW IDREFS>
    <!ELEMENT nazwisko (\#PCDATA)>
    <!ELEMENT adres (ulica, miasto)>
    <!ELEMENT ulica (\#PCDATA)>
    <!ELEMENT miasto (\#PCDATA)>
    <!ELEMENT film (tytul, rok, dlugosc)>
        <!ATTLIST film
        filmId ID
        gwiazdyW IDREFS>
    <!ELEMENT tytul (\#PCDATA)>
    <!ELEMENT rok (\#PCDATA)>
    <!ELEMENT dlugosc (\#PCDATA)>
]>
```

Atrybuty – przykład

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Gwiazdy-Filmy SYSTEM "gwiazdy-filmy.dtd">
<gwiazdy-filmy>
    <gwiazda gwiazdaId="cf" wystepujeW="gw, ik, pj">
        <nazwisko>Carrie Fischer</nazwisko>
        <adres>
            <ulica>123 Maple St.</ulica>
            <miasto>Hollywood</miasto>
        </adres>
    </gwiazda>
    <gwiazda gwiazdaId="mh" wystepujeW="ik, pj">
        <nazwisko>Mark Hamill</nazwa>
        <adres>
            <ulica>456 Oak Rd.</ulica>
            <miasto>Malibu</miasto>
        </adres>
        <adres>
            <ulica>123 Pine Av.</ulica>
            <miasto>Brentwood</miasto>
        </adres>
    </gwiazda>
```

Atrybuty – przykład

```
    <film  filmId="gw"  gwiazdyW="cf">
        <tytul>Gwiezdne Wojny</tytul>
        <rok>1977</rok>
        <dlugosc>93</dlugosc>
    </film>
    <film  filmId="ik"  gwiazdyW="cf, mh">
        <tytul>Gwiezdne Wojny</tytul>
        <rok>1980</rok>
        <dlugosc>96</dlugosc>
    </film>
    <film  filmId="pj"  gwiazdyW="cf, mh">
        <tytul>Powrot Jedi</tytul>
        <rok>1983</rok>
        <dlugosc>94</dlugosc>
    </film>
</gwiazdy-filmy>
```

Document Object Model – DOM

Konwencja opisu dokumentu XML (HTML, XHTML) pozwalająca na dostęp do poszczególnych elementów oraz wprowadzanie zmian w dokumencie.

Implementacje w przeglądarkach webowych (JavaScript) oraz bibliotekach obsługujących pliki XML.

W Pythonie moduł `xml.dom`.

`xml.dom` – podstawowe API

Klasy

Document	Cały dokument
Node	Klasa, z której dziedziczą składniki dokumentu
Element	Elementy XML
Attr	Atrybuty elementów
Comment	Komentarze
Text	Napisy (tekstowa zawartość elementów)

xml.dom.minidom – minimalistyczna implementacja DOM

Funkcje

xml.dom.minidom.parse(filename_or_file) Wczytuje dokument z pliku lub obiektu plikowego

xml.dom.minidom.parseString(string) Wczytuje dokument z napisu

Przykład

```
>>> d=xml.dom.minidom.parseString("<node>Text</node>")
>>> d
<xml.dom.minidom.Document instance at 0x105bed638>
>>> d.documentElement
<DOM Element: node at 0x105bed680>
>>> d.documentElement.tagName
u'node'
>>> d.documentElement.childNodes
[<DOM Text node "u'Text'">]
```

Modyfikacje DOM

```
>>> newel=d.createElement('extranode')
>>> nodetext=d.documentElement.childNodes[0]
>>> d.documentElement.insertBefore(newel, nodetext)
<DOM Element: extranode at 0x105bed3f8>
>>> newel=d.createElement('extranode')
>>> d.documentElement.appendChild(newel)
<DOM Element: extranode at 0x105ac7320>
>>> d.documentElement.childNodes
[<DOM Element: extranode at 0x105bed3f8>, <DOM Text node "u'Text'">, <
>>> d.documentElement.childNodes[0].appendChild(d.createTextNode('Extr
<DOM Text node "'Extra1'">
>>> d.documentElement.childNodes[2].appendChild(d.createTextNode('Extr
<DOM Text node "'Extra2'">
>>> d.documentElement.toxml()
u'<node><extranode>Extra1</extranode>Text<extranode>Extra2</extranode>
>>> d.documentElement.removeChild(nodetext)
<DOM Text node "u'Text'">
>>> d.documentElement.toxml()
u'<node><extranode>Extra1</extranode><extranode>Extra2</extranode></no
>>>
```

Wyszukiwanie

tree.xml

```
<!DOCTYPE tree [<!ATTLIST node id ID #IMPLIED>]>
<tree>
    <node type="domain" id="Bacteria"></node>
    <node type="domain" id="Archaea"></node>
    <node type="domain" id="Eukaryota">
        <node type="supergroup" id="Archaeplastida"></node>
        <node type="supergroup" id="Opisthokonta">
            <node type="kingdom" id="Fungi">
                <node type="genus" id="Muchomorek"></node>
            </node>
            <node type="kingdom" id="Animalia">
                <node type="type" id="Chordata">
                    <node type="class" id="Amphibia">
                        <node type="genus" id="Zielona␣zabka"></node>
                    </node>
                    <node type="class" id="Sauropsida"></node>
                    <node type="class" id="Aves"></node>
                    <node type="class" id="Synapsida"></node>
                    <node type="class" id="Mammalia">
                        <node type="genus" id="Myszka␣mala"></node>
                    </node>
                </node>
                <node type="type" id="Arthropoda"></node>
```

Wyszukiwanie c.d.

```
>>> d=xml.dom.minidom.parse('tree.xml')
>>> myszka=d.getElementById('Myszka mala')
>>> myszka
<DOM Element: node at 0x105be4a28>
>>> x=myszka
>>> while x.tagName=='node':
...     res.append(x)
...     x=x.parentNode
...
>>> res
[<DOM Element: node at 0x105be4a28>, <DOM Element: node at 0x105bed128
<DOM Element: node at 0x105bf1248>, <DOM Element: node at 0x105bf0f80
<DOM Element: node at 0x105bf07a0>, <DOM Element: node at 0x105bf0290
>>> [(x.getAttribute('type'), x.getAttribute('id')) for x in res]
[(u'genus', u'Myszka mala'), (u'class', u'Mammalia'), (u'type', u'Cho
(u'kingdom', u'Animalia'), (u'supergroup', u'Opisthokonta'),
(u'domain', u'Eukaryota')]
>>>
```

gnosis.xml

http://www.gnosis.cx/download/Gnosis_Utils.More/Gnosis_Utils-1.2.2.tar.gz

Moduł służący do konwersji XML na hierarchie zwykłych obiektów.

gnosis.xml c.d.

Przykład

```
>>> d=objectify.make_instance("tree1.xml")
>>> d
<tree id="107256890">
>>> d.__dict__
{'_seq': [u'\textbackslash{}n     ', <node id="1072568d0">, u'\textbac
', <node id="107256850">,
u'\textbackslash{}n     ', <node id="107256950">, u'\textbackslash{}n']
<node id="107256850">, <node id="107256950">], 'PCDATA': '', '__parent
>>> d.node
[<node id="1072568d0">, <node id="107256850">, <node id="107256950">]
>>> d.node[0].__dict__
{u'type': u'domain', u'id': u'Bacteria', '__parent__': <tree id="1072
>>> [x.id for x in d.node]
[u'Bacteria', u'Archaea', u'Eukaryota']
```

gnosis.xml c.d.

Wypisywanie

```
>>> objectify.utils.write_xml(d)
<tree>
    <node type=domain id=Bacteria></node>
    <node type=domain id=Archaea></node>
    <node type=domain id=Eukaryota>
        <node type=supergroup id=Archaeplastida></node>
        <node type=supergroup id=Opisthokonta>
            <node type=kingdom id=Fungi>
                <node type=genus id=Muchomorek></node>
            </node>
            <node type=kingdom id=Animalia>

...

            </node>
        </node>
    </node>
</tree>>>>>
>>> d.label="Tree of life"
>>> objectify.utils.write_xml(d)
<tree label=Tree of life>
    <node type=domain id=Bacteria></node>
    ...
```

gnosis.xml c.d.

Obchodzenie drzewa

```
>>> objectify.utils.walk_xo(d)
<generator object walk_xo at 0x107315b90>
>>> list(objectify.utils.walk_xo(d))
[<tree id="107256890">, <node id="1072568d0">, <node id="107256850">,
<node id="107256950">, <node id="1072569d0">, <node id="107256b10">,
<node id="107256b50">, <node id="107256b90">, <node id="107256bd0">,
<node id="107256c10">, <node id="107256c50">, <node id="107256c90">,
<node id="107256cd0">, <node id="107256d10">, <node id="107256d50">,
<node id="107256d90">, <node id="107256dd0">, <node id="107256e10">]
>>>
```

Czym są elementy dokumentu w gnosis.xml?

```
>>> d.__class__
<class 'gnosis.xml.objectify._objectify._XO_tree'>
>>> d.node[0].__class__
<class 'gnosis.xml.objectify._objectify._XO_node'>
```

Czym są elementy dokumentu w gnosis.xml?

```
>>> d.__class__
<class 'gnosis.xml.objectify._objectify._XO_tree'>
>>> d.node[0].__class__
<class 'gnosis.xml.objectify._objectify._XO_node'>
```

Obiekty mogą mieć metody.

Czym są elementy dokumentu w gnosis.xml?

```
>>> d.__class__
<class 'gnosis.xml.objectify._objectify._XO_tree'>
>>> d.node[0].__class__
<class 'gnosis.xml.objectify._objectify._XO_node'>
```

Obiekty mogą mieć metody.

```
>>> def node_repr(self):
...     return "'"+self.type+": "+self.id+"'"
...
>>> gnosis.xml.objectify._XO_node.__repr__=node_repr
```

Czym są elementy dokumentu w gnosis.xml?

```
>>> d.__class__
<class 'gnosis.xml.objectify._objectify._XO_tree'>
>>> d.node[0].__class__
<class 'gnosis.xml.objectify._objectify._XO_node'>
```

Obiekty mogą mieć metody.

```
>>> def node_repr(self):
...     return "'"+self.type+": "+self.id+"'"
...
>>> gnosis.xml.objectify._XO_node.__repr__=node_repr
```

W00t

```
>>> list(objectify.utils.walk_xo(d))
[<tree id="107256890">, 'domain: Bacteria', 'domain: Archaea',
'domain: Eukaryota', 'supergroup: Archaeplastida', 'supergroup: Opisth
'kingdom: Fungi', 'genus: Muchomorek', 'kingdom: Animalia', 'type: Ch
'class: Amphibia', 'genus: Zielona zabka', 'class: Sauropsida', 'class
```

Czym są elementy dokumentu w gnosis.xml?

Można *a priori* zdefiniować klasy odpowiadające elementom.

```
class Node(gnosis.xml.objectify._XO_):
    def repr(self):
        return "'"+self.type+": "+self.id+"'"

gnosis.xml.objectify._XO_node=Node
```

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* – $\{\text{kolor}, \text{czarno-biały}\}$

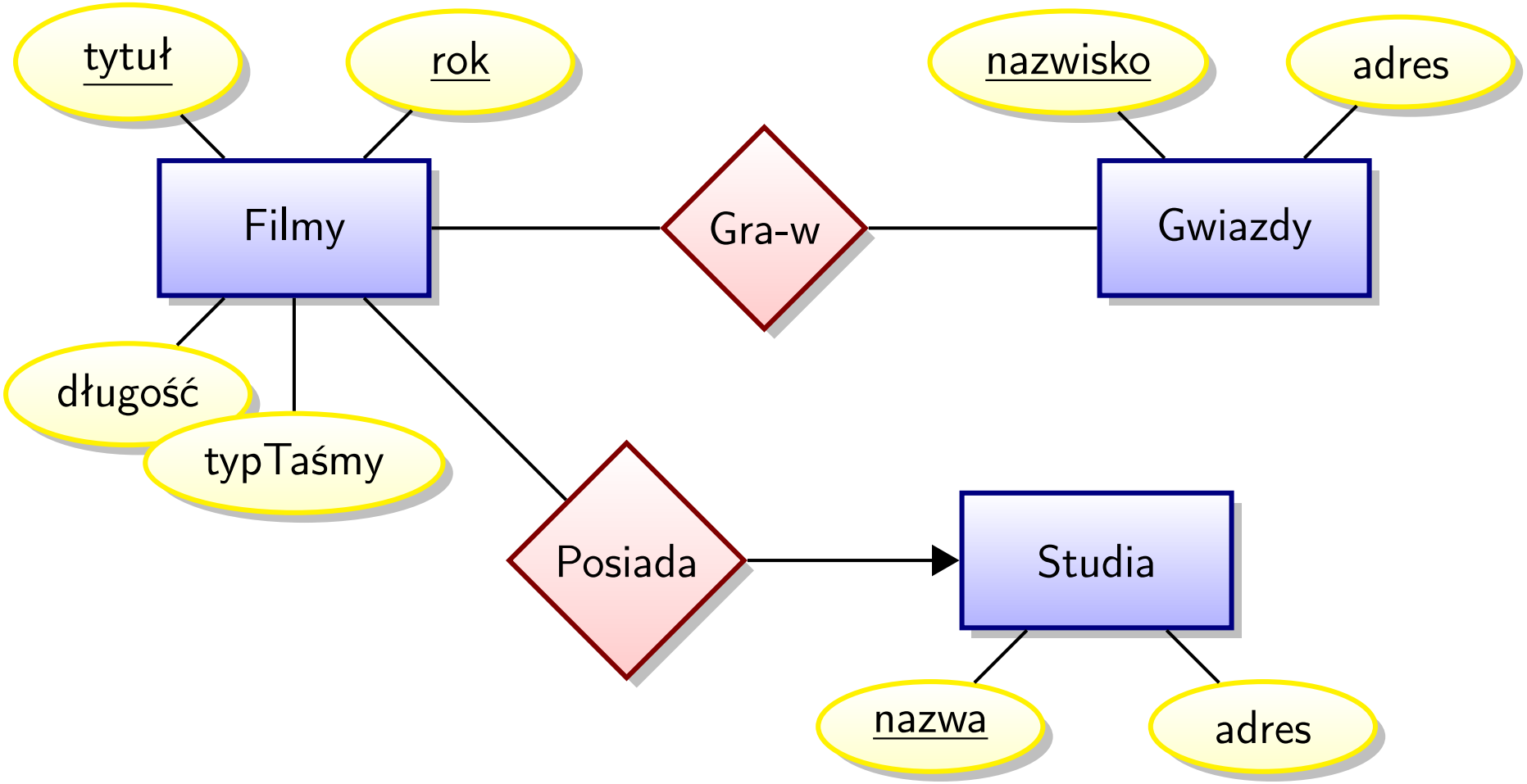
Schemat

- 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 → relacje

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

Związki → relacje

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

Relacje odpowiadające związkom

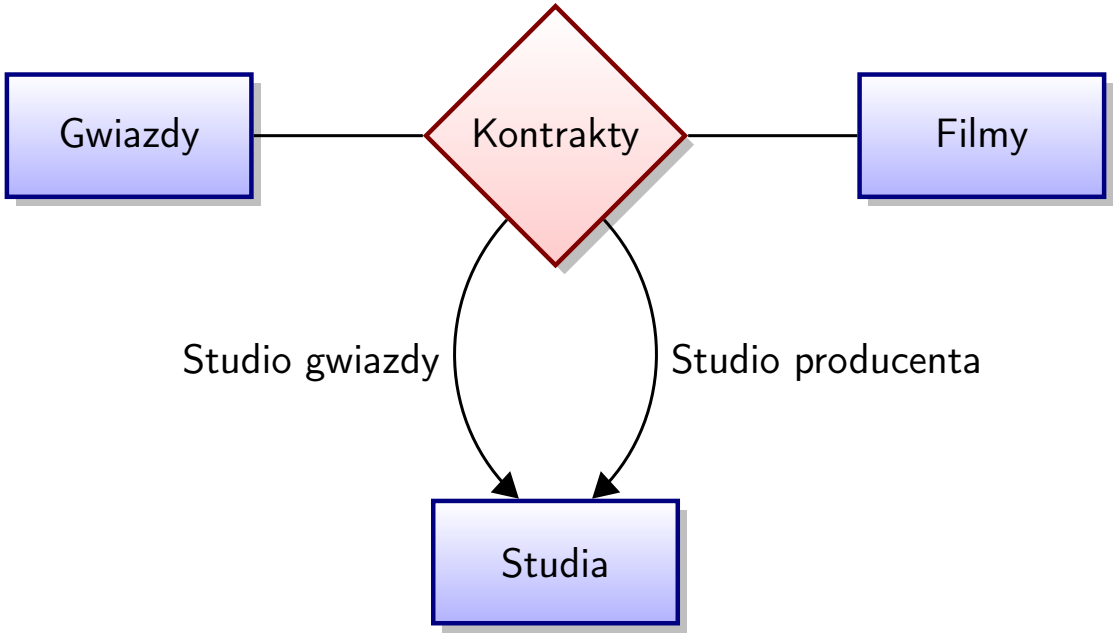
Posiada – wiele do jeden

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 PRAGM
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 COMM
>>>
```

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', 'fakeus
[<User('Edwardo','Ed Jones', 'f8s7ccs')>, <User('fakeuser','Inva
```

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

Nie ma

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

```
>>> for name, in session.query(User.name).\
... filter(User.fullname=='Ed Jones'):
...     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.id)
>>> query.all()
[<User('ed','Ed Jones', 'f8s7ccs')>, <User('fred','Fred Flinston
```

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
```

Związki c.d.

Alternatywnie

```
class User(Base):
    # ....
    addresses = relationship("Address",
                order_by="Address.id", backref="user")
```

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, 2 – Edycja dokumentu

Zadanie

Dopisz kilka ulubionych taksonów do `tree.xml`. Wykonaj to przy pomocy `xml.dom.minidom` i `gnosis.xml`.

Zadanie 3 – Tutorial

Zadanie

Zapoznaj się z tutorialem do SQLAlchemy.

Zadanie 4 – Pracownicy/etaty

Zadanie

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