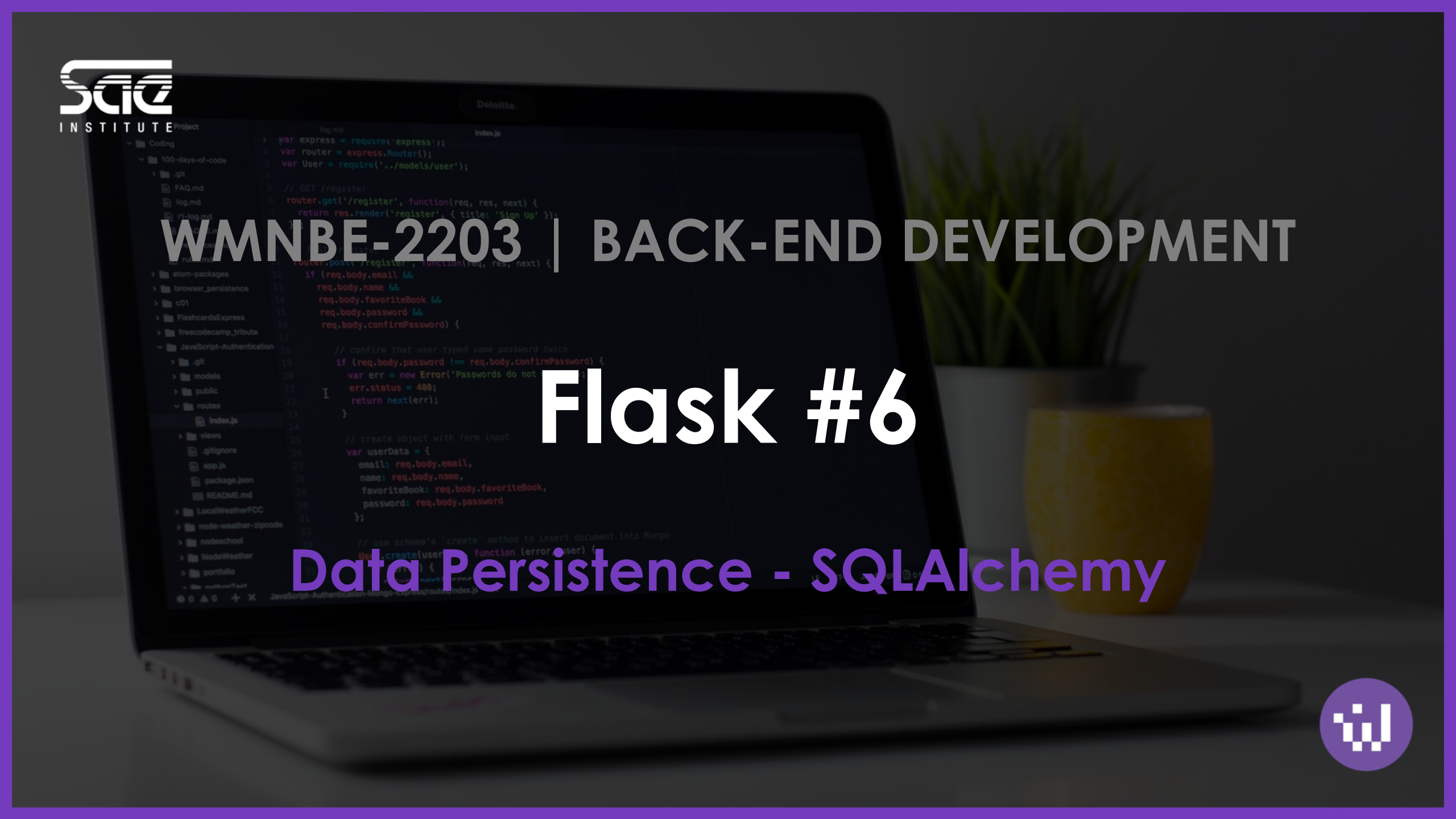

WMNBE-2203 | BACK-END DEVELOPMENT

Flask #6

Data Persistence - SQLAlchemy

ΠΕΡΙΕΧΟΜΕΝΑ

Περιεχόμενα

- Object-Oriented Programming (OOP) basics
  - Class
  - Attributes
  - Constructor / Methods
  - Inheritance
  - Properties
- SQLAlchemy
  - Session
  - Model / Column
  - Relationships

OOP BASICS

OOP basics

Python & OOP

Η **Python** υποστηρίζει αρκετές διαφορετικές προσεγγίσεις προγραμματισμού. Ο *Αντικειμενοστρεφής Προγραμματισμός* είναι μία από αυτές.

Στον *Αντικειμενοστρεφή Προγραμματισμό* (**OOP**) τα πρωτεύοντα δομικά στοιχεία του προγράμματος είναι τα δεδομένα, από τα οποία δημιουργούνται, με κατάλληλη μορφοποίηση, τα αντικείμενα (**objects**).

Στόχος του είναι η δημιουργία ευέλικτου και επαναχρησιμοποιούμενου κώδικα.

OOP basics

class

Στην καρδιά του **OOP** είναι η κλάση (**class**). Η κλάση είναι το "προσχέδιο", βάσει του οποίου φτιάχνονται τα αντικείμενα.

Συχνά, μία κλάση έχει το ρόλο μιας *user-defined* δομής δεδομένων (*data-structure*), που βασίζεται στους απλού τύπους δεδομένων της γλώσσας.

Αλλά μια κλάση, εκτός από τα χαρακτηριστικά (*attributes*), ορίζει και τη συμπεριφορά (*behavior*) των αντικειμένων.

```
class Person:
    pass
```

OOP basics

attributes

Τα βασικότερα στοιχεία μίας κλάσης είναι τα χαρακτηριστικά της.

Στην **Python** τα χαρακτηριστικά μιας κλάσης μπορεί να είναι, είτε στο επίπεδο της κλάσης (*class attributes*), είτε στο επίπεδο των αντικειμένων / στιγμιότυπων (*instance attributes*).

Τα *class attributes* ορίζονται ως μεταβλητές στο σώμα της κλάσης, ενώ τα *instance attributes* ορίζονται μέσα στη μέθοδο κατασκευαστή (*constructor*).

OOP basics

class vs instance attributes

- *instance attributes*: Χαρακτηριστικά στα οποία κάθε αντικείμενο / στιγμιότυπο κρατά τις δικές του τιμές.
- *class attributes*: Χαρακτηριστικά με κοινές τιμές για όλα τα αντικείμενα / στιγμιότυπα της κλάσης, **εκτός αν** κάποιο αντικείμενο ορίσει δικές του τιμές για αυτά.

  Χρήσιμα για σταθερές ή για προκαθορισμένες (*default*) τιμές σε **attributes**.

OOP basics

constructor

Το ρόλο του **constructor** μιας κλάσης εξυπηρετεί η *dunder* μέθοδος **__init__**.

```
class Person:
    def __init__(self):
        pass
```

Πρώτο όρισμα της μεθόδου αυτής είναι μία αναφορά στο ίδιο το αντικείμενο.

Το όρισμα αυτό μπορεί να έχει οποιοδήποτε όνομα, αλλά συνηθίζεται το όνομα **self**.

OOP basics

Παράδειγμα

```
class Person:
    # class attribute
    origin = 'Earth'

    def __init__(self, fn, ln):
        # instance attributes
        self.firstname = fn
        self.lastname = ln
```

OOP basics

Instantiation

Ένα αντικείμενο (*object*) είναι ένα στιγμιότυπο (*instance*) της κλάσης.

Έχει τις δικές του τιμές για κάθε ένα από τα χαρακτηριστικά που ορίζονται στην κλάση στην οποία ανήκει.

Τα αντικείμενα δημιουργούνται, κυρίως, καλώντας τη μέθοδο κατασκευαστή (*constructor*).

```
p1 = Person('Guido', 'Van Rossum')
p2 = Person('Armin', 'Ronacher')
```

OOP basics

methods

Όπως ειπώθηκε και νωρίτερα, δεν είναι μόνο τα *attributes* που χαρακτηρίζουν μία κλάση, αλλά και οι μέθοδοι της (**methods**).

Οι μέθοδοι είναι συναρτήσεις που ανήκουν στην κλάση και έχουν πρόσβαση στα χαρακτηριστικά ενός αντικειμένου, μέσω του ειδικού ορίσματος **self**, όπως και ο **constructor**.

```
class Person:
    …
    def say_hello(self):
        print(f"Hi, I'm {self.firstname}!")
```

OOP basics

class methods

Μιας και η *dunder* μέθοδος **__init__** είναι μοναδική ανά κλάση, σε περίπτωση που χρειάζονται παραπάνω από ένας τρόποι δημιουργίας ενός αντικειμένου, μπορεί να χρησιμοποιηθεί ο *decorator* **@classmethod**.

Μία μέθοδος "διακοσμημένη" με το **@classmethod**, δέχεται ως πρώτο όρισμα μια αναφορά στην κλάση.

```
class Person:
    …
    @classmethod
    def from_full_name(cls, name):
        fn, ln = name.split(' ')
        return cls(fn, ln)      # cls(*name.split(' '))
```

OOP basics

static methods

Όπως και οι **class methods**, μία **static** μέθοδος δεν έχει αναφορά σε συγκεκριμένο αντικείμενο, για αυτό και απουσιάζει και εδώ η παράμετρος **self**.

Χρησιμοποιείται για *utility* μεθόδους και γενικά για μεθόδους που αφορούν στην ίδια την κλάση, χωρίς όμως να μπορούν να τροποποιήσουν το *state* της.

```
class Person:
    …
    @staticmethod
    def get_definition():
        return 'A person (plural people or persons) is …'
```

OOP basics

Inheritance

Η *κληρονομικότητα* είναι ένα από τα πιο δυνατά χαρακτηριστικά του **OOP**, που βοηθά, μεταξύ άλλων, στο να μειωθεί ο επαναλαμβανόμενος κώδικας.

Μέσω της κληρονομικότητας μία κλάση "κληρονομεί" όλα τα χαρακτηριστικά της κλάσης γονέα.

Στην **Python** αυτό γίνεται ως εξής:

```
class BaseClass:
    <base class code>

class DerivedClass(BaseClass):
    <derived class code>
```

OOP basics

Παράδειγμα

```
class Person:
    def __init__(self, fn, ln):
        self.first_name = fn          # instance attribute
        self.last_name = ln           # instance attribute

    def say_hello(self):
        print(f"Hi, I'm {self.first_name} {self.last_name}!")

    def __repr__(self):
        return f'<Person {self.first_name=}, {self.last_name=}>'

class Student(Person):
    def __init__(self, fn, ln, rn):
        super().__init__(fn, ln)     # Calling Person's constructor

        self.registration_number = rn

    def __repr__(self):
        return f'<Student {self.registration_number=}>'
```

Properties

Αν χρειαζόμαστε περισσότερο έλεγχο στον τρόπο που προσπελαύνονται τα **attributes** μίας κλάσης, μπορούσε να χρησιμοποιήσουμε τον *decorator* **@property**, μαζί με τους:

@<property-name>.setter ή/και **@<property-name>.deleter**.

Με τον **@property**, ορίζουμε μία μέθοδο η οποία θα καλείται όταν ζητηθεί η τιμή ενός **property**. Το **property** αυτό θα έχει το ίδιο όνομα με τη μέθοδο.

Με τους **@<property-name>.setter** και **@<property-name>.deleter** ορίζουμε τις μεθόδους που θα κληθούν όταν προσπαθήσουμε να αλλάξουμε τιμή ή να διαγράψουμε το **property**, αντίστοιχα.

Η τιμή του **property** αποθηκεύεται σε ένα **instance attribute** το οποίο, από παραδοχή, έχει ως πρόθεμα ένα ή δύο _.

OOP basics

Παράδειγμα

```
class Student(Person):
    def __init__(self, rn):
        self.registration_number = rn
        self.__grades = []

    @property
    def grades(self):    # Getter
        return self.__grades

    @grades.setter       # Setter
    def grades(self, value):
        if len(self.__grades) == 0:
            self.__grades = value
        else:
            raise AttributeError('You cannot re-set the grades')

s = Student('12345')
s.grades = [1, 2, 3]     # Setter
print(s.grades)          # Getter
s.grades = [4, 5, 6]     # Error
```

SQLALCHEMY

SQLAlchemy

SQL Toolkit and ORM

Το **SQLAlchemy** είναι μία σουίτα εργαλείων για τη σύνδεση και επικοινωνία μιας εφαρμογής **Python** με μία ή και περισσότερες βάσεις δεδομένων.

Το πιο γνωστό κομμάτι της σουίτας αυτής είναι το κομμάτι του **Object-Relational Mapper** (**ORM**), το οποίο υλοποιεί το **data mapper** μοτίβο.

Με αυτό γίνεται αντιστοίχιση κλάσεων της εφαρμογής με τα στοιχεία της βάσης.

SQLAlchemy

session

Στην καρδιά του **SQLAlchemy** βρίσκεται το **session**. Μέσω αυτού γίνεται όλη η διαχείριση της επικοινωνίας με τη βάση με τη βάση.

Προσοχή να μη συγχέεται με το **session** μιας web εφαρμογής. Το **session** εδώ παίζει, λίγο ή πολύ, το ρόλο του **connection**.

```
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker
from sqlalchemy.ext.declarative import declarative_base

engine = create_engine('sqlite:///data/app.db')
db_session = scoped_session(sessionmaker(autocommit=False,
                                         autoflush=False,
                                         bind=engine))
@app.teardown_appcontext
def shutdown_session(exception=None):
    db_session.remove()
```

SQLAlchemy

Flask-SQLAlchemy

Μιας και ο τρόπος διαχείρισης ενός **session** είναι, συχνά, συγκεκριμένος σε μια web εφαρμογή, συνιστάται η χρήση του **Flask-SQLAlchemy** module.

Το **Flask-SQLAlchemy** μπορεί να αναλάβει τη διαχείρισή του **session** αυτόματα, με ελάχιστες ρυθμίσεις.

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///data/app.db'
db = SQLAlchemy(app)
```

SQLAlchemy

Model

Για να γίνει η σύνδεση με τη βάση, χρειάζεται να ορίσουμε κλάσεις που θα αντιπροσωπεύουν τα στοιχεία της και συνήθως αντιστοιχούν 1:1 με τους πίνακές της.

Οι κλάσεις αυτές πρέπει να κληρονομούν από την κλάση **Model** του **SQLAlchemy**.

```
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self):
        return f'<User {self.username=}>'
```

SQLAlchemy

Column

Τα *attributes*, των κλάσεων αυτών, αντιστοιχούν με τα πεδία των πινάκων. Η αντιστοίχιση γίνεται μέσω της κλάσης **Column**.

Το όνομα της στήλης θα είναι ίδιο με το όνομα του *attribute*. Αν θέλουμε να διαφέρει πρέπει να το περάσουμε ως πρώτο όρισμα στον κατασκευαστή της **Column**.

Μία στήλη μπορεί να οριστεί ως **primary_key**, **unique**, **nullable**, **autoincrement** με αντίστοιχα *optional* ορίσματα.

Παραπάνω από μία στήλες μπορούν να οριστούν ως **primary_key** στον ίδιο πίνακα.

SQLAlchemy

Column types

Ο τύπος κάθε στήλης ορίζεται μέσω του πρώτου ορίσματος της **Column** (ή του δεύτερου, αν έχει δοθεί και όνομα).

Οι πιο συνηθισμένοι τύποι είναι οι εξής:

Τύπος	Περιγραφή
Integer	an integer
String(size)	a string with a maximum length (optional in some databases)
Text	some longer unicode text
DateTime	date and time expressed as Python datetime object
Float	stores floating point values
Boolean	stores a boolean value

SQLAlchemy

ForeignKey

Οι απλές σχέσεις (**1:1**, **1:N**) ορίζονται, κυρίως, μέσω της κλάσης **ForeignKey**.

Στη δήλωση του **FK**, κατά την κλήση του κατασκευαστή της **Column**, περνάμε ως όρισμα το **db.ForeignKey('<table>.<pk>')**.

Όπου **<table>.<pk>** τα ονόματα του πίνακα, από την "άλλη πλευρά" της σχέσης, και του πεδίου (συνήθως το **PK**).

```
class Article(db.Model):
    …
    category_id = db.Column(db.Integer,
        db.ForeignKey('category.id'),
        nullable=False)
```

SQLAlchemy

relationship

Πέρα από τη δήλωση του **ForeignKey**, μπορούμε με τη μέθοδο **relationship** να δημιουργήσουμε *attributes* που αφορούν στα συσχετιζόμενα αντικείμενα.

Η κλήση της **relationship** μπορεί να βρίσκεται από οποιαδήποτε "πλευρά" της σχέσης και μπορεί να δημιουργήσει *attributes* και στα δύο "άκρα" ταυτόχρονα, με τη βοήθεια του ορίσματος **backref**.

Σε σχέσεις **1:1** αρκεί να προστεθεί το όρισμα **uselist=False**.

```
class Post(db.Model):
    …
    author_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    author = db.relationship('User', backref='posts', lazy=False)
    comments = db.relationship('Comment', backref='post', lazy=True)
```

SQLAlchemy

lazy

Η παράμετρος **lazy** καθορίζει τον τρόπο και τη στιγμή που το **SQLAlchemy** θα φορτώσει από τη βάση τα σχετιζόμενα δεδομένα.

Τιμή	Περιγραφή
select/**True**	(*default*) will load the data as necessary in one go using a standard **select** statement.
joined/**False**	load the relationship in the same query as the parent using a **join** statement.
subquery	works like **joined** but will use a **subquery** statement.
dynamic	will return another query object which you can further refine before loading the items.

SQLAlchemy

table

Στις **1:N** και **1:1** σχέσεις, σε κάθε άκρο τους υπάρχει μία οντότητα. Στις **N-N** σχέσεις, από την άλλη, υπάρχει ένα ενδιάμεσος, βοηθητικός, πίνακας.

Για να οριστούν τέτοιου είδους σχέσεις και να περιγραφούν οι ενδιάμεσοι αυτοί πίνακες, πρέπει να χρησιμοποιηθεί η μέθοδος **table**.

Με τη μέθοδο **table** θα μπορούσαν να οριστούν όλοι οι πίνακες της βάσεις, ώστε οι κλάσεις / μοντέλα να μην περιέχουν (όσο είναι δυνατό) **database-related** κώδικα.

Αυτό, όμως, αποτελεί μια διαφορετική προσέγγιση, που δεν θα καλυφθεί εδώ.

SAE INSTITUTE

SQLAlchemy

Παράδειγμα

```
book_authors = db.Table('book_authors',
    db.Column('book_id', db.Integer, db.ForeignKey('book.id')),
    db.Column('author_id', db.Integer, db.ForeignKey('author.id'))
)

class Book(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String)
    authors = db.relationship('Author',
                              secondary=book_authors,
                              backref='books')

class Author(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String)
```

Χρήσιμα links

Object-Oriented Programming (OOP) in Python 3 – Real Python
https://realpython.com/python3-object-oriented-pro…

Python Object Oriented Programming - Programiz
https://www.programiz.com/python-programming/o…

SQLAlchemy in Flask — Flask Documentation (2.0.x)
https://flask.palletsprojects.com/en/2.0.x/patterns/sql…

Under the Hood of Flask-SQLAlchemy | by Kelly Foulk | Analytics Vidhya | Medium
https://medium.com/analytics-vidhya/under-the-hoo…

Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (2.x)
https://flask-sqlalchemy.palletsprojects.com/en/2.x/

SQLAlchemy ORM — SQLAlchemy 1.4 Documentation
https://docs.sqlalchemy.org/en/14/orm/index.html

Data Management With Python, SQLite, and SQLAlchemy – Real Python
https://realpython.com/python-sqlite-sqlalchemy/#cr…

Extra info

class method vs static method in Python - GeeksforGeeks
https://www.geeksforgeeks.org/class-method-vs-stati…

python - How do I know if I can disable SQLALCHEMY_TRACK_MODIFICATIONS? - Stack Overflow
https://stackoverflow.com/questions/33738467/how-d…

python - How to define two relationships to the same table in SQLAlchemy - Stack Overflow
https://stackoverflow.com/questions/7548033/how-to…

python - SQLAlchemy default DateTime - Stack Overflow
https://stackoverflow.com/questions/13370317/sqlalc…

python - SQLAlchemy: cascade delete - Stack Overflow
https://stackoverflow.com/questions/5033547/sqlalch…

python - Sqlite / SQLAlchemy: how to enforce Foreign Keys? - Stack Overflow
https://stackoverflow.com/questions/2614984/sqlite-s…

THANK YOU!