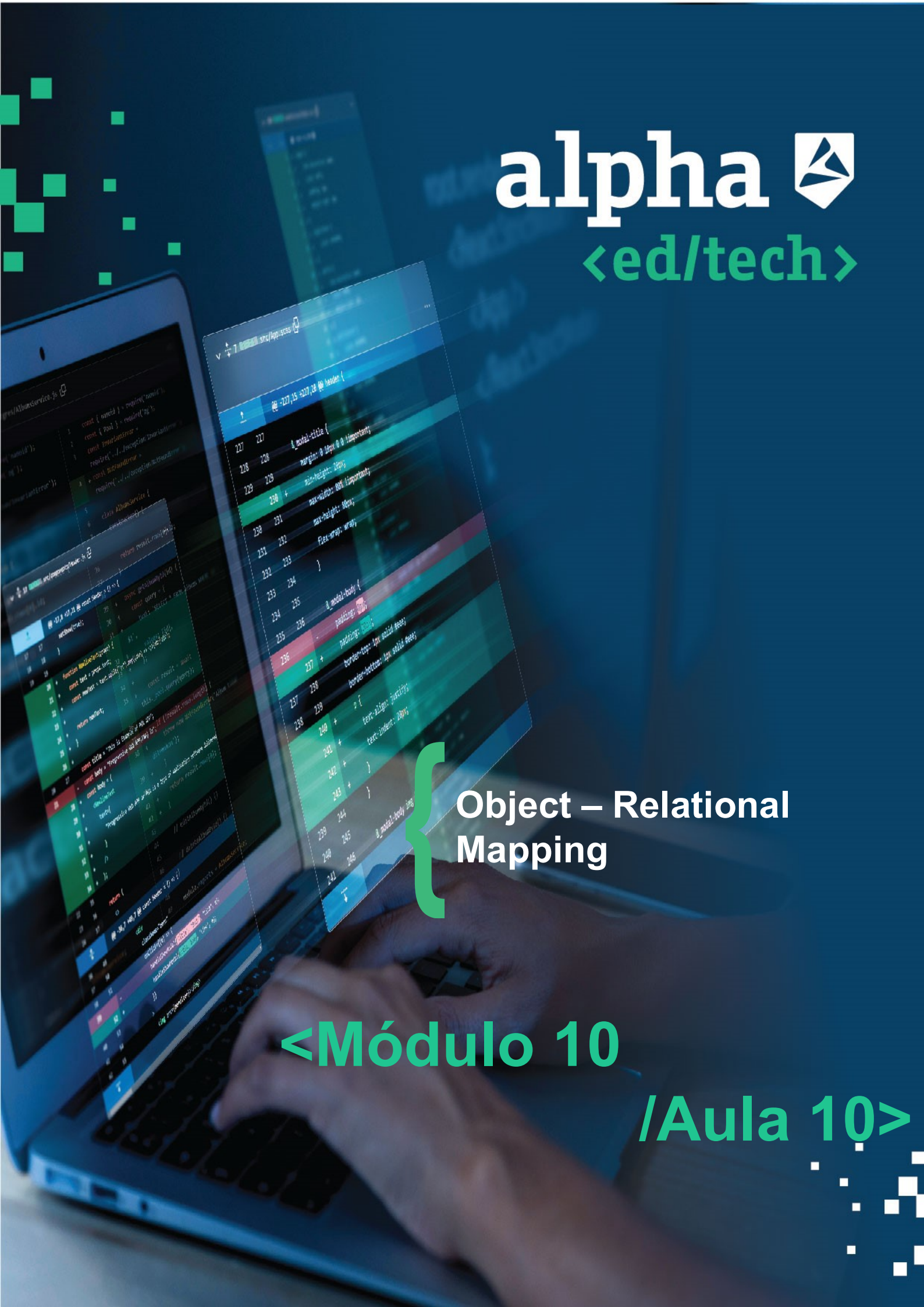
alpha
<ed/tech>
Object – Relational Mapping
<Módulo 10
/Aula 10>

ORM Object Relational Mapping

SQLAlchemy

As bibliotecas ORM fazem o mapeamento da Programação Orientada a Objetos para os comandos SQL para o Banco de Dados Relacional. Também na programação de API a biblioteca SQLAlchemy integra-se bem com muitos pacotes, incluindo muitos ORMs.

Você pode usar a maioria dos bancos de dados relacionais integrando facilmente com SQLAlchemy.

SQLAlchemy suporta PostgreSQL, MySQL, SQLite, Oracle, Microsoft SQL Server e outros.

Outras estruturas de microsserviços Python, como Flask, não se integram facilmente ao SQLAlchemy. É comum usar o Flask com um pacote chamado Flask-SQLAlchemy. Flask-SQLAlchemy não é necessário e tem seus próprios problemas.

SQLAlchemy itens importantes

1. Introdução ao SQLAlchemy:
   - Documentação oficial do SQLAlchemy como visão geral do que é o SQLAlchemy e como ele funciona

2. Conhecendo as Classes do SQLAlchemy:
   - Entender `declarative_base`, que é uma classe base para definir suas classes de modelo (tabelas) no SQLAlchemy
   - Entender `create_engine`, `Column`, `String`, `Integer`, `Date`
   - Entender como criar classes de modelo usando a sintaxe do SQLAlchemy
   - Utilizar echo=True visualiza o comando equivalente em SQL

3. Conexão com o Banco de Dados:
   - Configurar a conexão com o banco de dados usando as configurações apropriadas
   - Criar o database na memória ou arquivo.db
   - Criação de tabelas no banco de dados usando as classes do modelo definido.

Instalação da biblioteca SQLAlchemy

pip install sqlalchemy

Ambiente virtual

python - m venv env

No Windows para ativar

.\env\Scripts\activate

pip freeze > requirements.txt

SQLAlchemy

Exemplo de criação/implementação de um Modelo Físico de Base de Dados

Verificação da instalação do SQLAlchemy

```
# source code: ./modeloTeste.py

import sqlalchemy

print("sqlalchemy.__version__", sqlalchemy.__version__)

# output
#sqlalchemy.__version__ 2.0.19
```

Criação de tabela de pets (animais de estimação) na memória e em arquivo pets.db

```
# source code: ./modelo0.py

from sqlalchemy import (create_engine, Column, String, Integer, Date)
from sqlalchemy.orm import declarative_base

engine = create_engine('sqlite:///:memory:', echo=True)
# echo=True repete no console o comando em Python e depois equivalente em SQL
# engine = create_engine('sqlite:///pets.db', echo=True)

# class
Base = declarative_base()

class Pet(Base):
    # Tabela pets.db
    __tablename__ = 'pets'

    id = Column(Integer, primary_key=True)
    petNome = Column(String)
    petIdade = Column(Integer)
    petTipo = Column(String)

    def __repr__(self):
        return f'{self.id} | {self.petNome} {self.petIdade} {self.petTipo}'


if __name__ == '__main__':
    Base.metadata.create_all(engine)
```

SQLAlchemy

Inserir dados na tabela pets (animais de estimação) em arquivo pets.db

```
# source code: ./modelo0b.py
# gravando em arquivo em disco

from sqlalchemy import (create_engine, Column,
                        String, Integer)
from sqlalchemy.orm import declarative_base


# https://docs.sqlalchemy.org/en/20/orm/quickstart.html#create-an-engine
# engine = create_engine('sqlite:///:memory:', echo=True)
# echo=True repete no console o comando em Python e depois equivalente em SQL
engine = create_engine('sqlite:///pets.db', echo=True)

# class
Base = declarative_base()

class Pet(Base):
    # Tabela pets.db
    __tablename__ = 'pets'

    id = Column(Integer, primary_key=True)
    petNome = Column(String)
    petIdade = Column(Integer)
    petTipo = Column(String)

    def __repr__(self):
        return f'{self.id} | {self.petNome} {self.petIdade} {self.petTipo}`

if __name__ == '__main__':
    Base.metadata.create_all(engine)

kika = Pet(id=1, petNome="Kika", petIdade=8, petTipo="Cachorro")
print(kika)
print(kika.id)
print(kika.petNome)
print(kika.petIdade)
print(kika.petTipo)
print('================')
print()

# output
# 1 | Kika 8 Cachorro
# 1
# Kika
# 8
# Cachorro
# ===============
```

SQLAlchemy

Inserir novo pet sem o atributo id (no arquivo pets.db)

```
# source code: ./modelo0b.py
# gravando em arquivo em disco
from sqlalchemy import (create_engine, Column,
                        String, Integer)
from sqlalchemy.orm import declarative_base

# https://docs.sqlalchemy.org/en/20/orm/quickstart.html#create-an-engine
# engine = create_engine('sqlite:///:memory:', echo=True)
# echo=True repete no console o comando em Python e depois equivalente em SQL
engine = create_engine('sqlite:///pets.db', echo=True)
# class
Base = declarative_base()

class Pet(Base):
    # Tabela pets.db
    __tablename__ = 'pets`
    id = Column(Integer, primary_key=True)
    petNome = Column(String)
    petIdade = Column(Integer)
    petTipo = Column(String)

    def __repr__(self):
        return f'{self.id} | {self.petNome} {self.petIdade} {self.petTipo}`

if __name__ == '__main__':
    Base.metadata.create_all(engine)

kika = Pet(id=1, petNome="Kika", petIdade=8, petTipo="Cachorro")
Print(kika)

# output
#2023-08-18 07:27:40,992 INFO sqlalchemy.engine.Engine BEGIN (implicit)
#2023-08-18 07:27:40,992 INFO sqlalchemy.engine.Engine PRAGMA main.table_info("pets")
#2023-08-18 07:27:40,992 INFO sqlalchemy.engine.Engine [raw sql] ()
#2023-08-18 07:27:40,993 INFO sqlalchemy.engine.Engine PRAGMA temp.table_info("pets")
#2023-08-18 07:27:40,993 INFO sqlalchemy.engine.Engine [raw sql] ()
#2023-08-18 07:27:40,997 INFO sqlalchemy.engine.Engine
#CREATE TABLE pets (
#                id INTEGER NOT NULL,
#                "petNome" VARCHAR,
#                "petIdade" INTEGER,
#                "petTipo" VARCHAR,
#                PRIMARY KEY (id)
#)
#2023-08-18 07:27:40,997 INFO sqlalchemy.engine.Engine [no key 0.00026s] ()
#2023-08-18 07:27:40,997 INFO sqlalchemy.engine.Engine COMMIT

# 1 | Kika 8 Cachorro
```

SQLAlchemy

“Visualizador” de database simples

https://sqlitebrowser.org/dl/

Inserir novo pet sem o atributo id

```
# source code: ./modelo0c.py
# gravando em arquivo em disco
from sqlalchemy import (create_engine, Column, String, Integer)
from sqlalchemy.orm import declarative_base

engine = create_engine('sqlite:///pets.db', echo=True)
# class
Base = declarative_base()

class Pet(Base):
    # Tabela pets.db
    __tablename__ = 'pets`
    id = Column(Integer, primary_key=True)
    petNome = Column(String)
    petIdade = Column(Integer)
    petTipo = Column(String)

    def __repr__(self):
        return f'{self.id} | {self.petNome} {self.petIdade} {self.petTipo}`

if __name__ == '__main__':
    Base.metadata.create_all(engine)

kika = Pet(id=1, petNome="Kika", petIdade=8, petTipo="Cachorro")
print(kika)

# Inserir outro animal
nina = Pet(petNome="Nina", petIdade=3, petTipo="Gato")
print(nina)
print('================')
print()

# Inserir outro animal
joye = Pet(petNome="joye", petIdade=7, petTipo="Gato")
print(joye.id)
print('================')
print()

#1 | Kika 8 Cachorro
#
#None | Nina 3 Gato
#================
#
#None
#================
```

SQLAlchemy

Criando outra tabela de usuarios

```
# source code: ./modelo02.py

from typing import List
from typing import Optional
from sqlalchemy import ForeignKey
from sqlalchemy import String
from sqlalchemy.orm import DeclarativeBase
from sqlalchemy.orm import Mapped
from sqlalchemy.orm import mapped_column
from sqlalchemy.orm import relationship


class Base(DeclarativeBase):
    pass

class User(Base):
    __tablename__ = "user_account"
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(30))
    fullname: Mapped[Optional[str]]
    addresses: Mapped[List["Address"]] = relationship(
        back_populates="user", cascade="all, delete-orphan"
    )
    def __repr__(self) -> str:
        return f"User(id={self.id!r}, name={self.name!r}, fullname={self.fullname!r})"

class Address(Base):
    __tablename__ = "address"
    id: Mapped[int] = mapped_column(primary_key=True)
    email_address: Mapped[str]
    user_id: Mapped[int] = mapped_column(ForeignKey("user_account.id"))
    user: Mapped["User"] = relationship(back_populates="addresses")
    def __repr__(self) -> str:
        return f"Address(id={self.id!r}, email_address={self.email_address!r})"

from sqlalchemy import create_engine

engine = create_engine('sqlite:///pets02.db', echo=True)

from sqlalchemy.orm import declarative_base

Base = declarative_base()

if __name__ == '__main__':
    Base.metadata.create_all(engine)
```