ORM - Object Relational Mapper

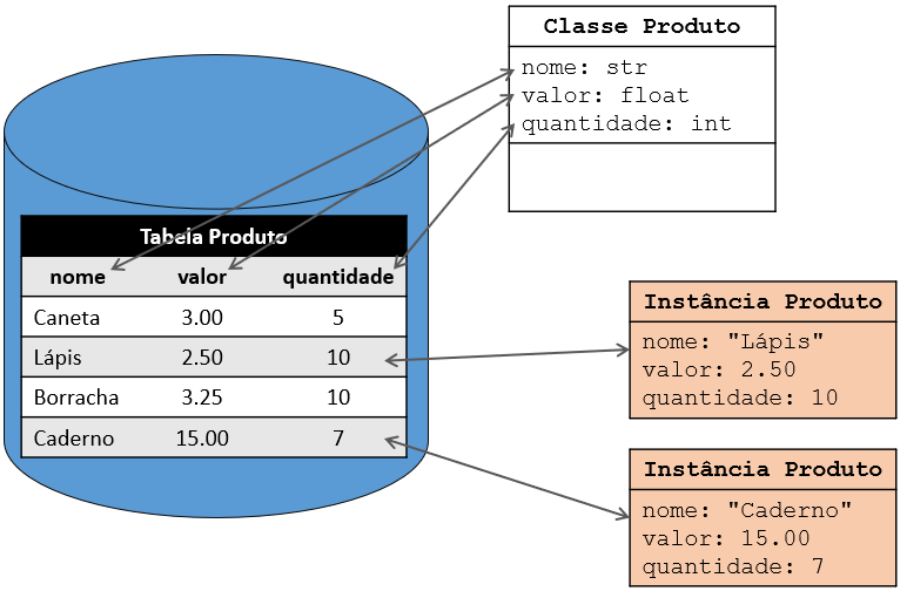
Object-Relational Mapping (ORM), em português, mapeamento objeto-relacional, é uma técnica para aproximar o paradigma de programação orientada a objetos ao paradigma do banco de dados relacional.

Ultimamente tem sido muito utilizada e vem crescendo bastante nos últimos anos. Este crescimento tem se dado principalmente pelo fato de muitos desenvolvedores não se sentirem a vontade em escrever código SQL e pela produtividade que esta técnica proporciona.

O uso da técnica de mapeamento objeto-relacional é realizado através de um mapeador objeto-relacional, que geralmente é uma biblioteca ou framework que ajuda no mapeamento e uso do banco de dados (neste tutorial será utilizado o framework **SQLAlchemy**).

O mapeamento objeto-relacional consiste em representar as tabelas de um banco de dados em classes de um programa orientado a objetos, onde cada campo da tabela é representado por um atributo da classe.

Essa técnica facilita a manipulação dos dados contidos na tabela, assim como a inserção e atualização desses dados.



Conexão com Banco de Dados SQLITE

Instalar pacote sqlalchemy:

Executar no terminal o comando a seguir:

`pip install sqlalchemy`

Conectar com o banco de dados SQLITE

```
# Importar os módulos que serão utilizados
import sqlalchemy
from sqlalchemy import Column, Integer, String, Float
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import Session


# Criar Conexão com Banco SQLITE. Caso o arquivo não exista, ele será criado
engine = sqlalchemy.create_engine("sqlite:///server.db")
connection = engine.connect()

# Instancia da classe Base
Base = declarative_base(engine)

# Criação da sessão
session = Session()
```

Criação de Tabela

```
# Exemplo de script para criar tabela no banco de dados SQLite
connection.execute("""CREATE TABLE IF NOT EXISTS FUNCIONARIO (
                        ID INTEGER PRIMARY KEY,
                        NOME VARCHAR(255) NOT NULL,
                        IDADE INT NOT NULL,
                        SALARIO FLOAT NOT NULL)
                    """)
```

Mapeamento de Tabela

Realiza o mapeamento de um tabela do banco de dados em um classe.

```
# Mapeamento da tabela
class Funcionario(Base):
    __tablename__ = 'FUNCIONARIO'
    id = Column('ID', Integer, primary_key=True, autoincrement=True)
    nome = Column('NOME', String(255))
    idade = Column('IDADE', Integer)
    salario = Column('SALARIO', Float)

    def __init__(self, nome, idade, salario):           # Construtor
        self.nome = nome
        self.idade = idade
        self.salario = salario
```

Inserir registro na tabela

.add(): adiciona um objeto na tabela

.add_all(): adiciona uma lista de objetos na tabela

```

# Inserir dados (um objeto)
func = Funcionario('Zezinho', 20, 1700)                 # criar o objeto
session.add(func)
session.commit()                                        # necessario fazer o commit()


# Inserir dados (lista de objetos)
func1 = Funcionario('Luizinho', 22, 1250)
func2 = Funcionario('Huguinho', 22, 2000)
lista = [func1, func2]
session.add_all(lista)
session.commit()         # NECESSÁRIO fazer o commit()
```

Consultar dados da tabela

Funções Principais:

.query(nome_da_classe): executa uma query na tabela mapeada pela classe informada no parâmetro.

.all(): retorna todos os resultados da consulta realizada

.first(): retorna o primeiro resultado da consulta realizada

.get(valor): realiza a busca por um registro específico, de acordo com o valor informado (primary_key)

.filter(): define o filtro a ser aplicado na consulta

.order_by(): ordena a consulta relaizada de acordo com o valor informado

```
# Busca todos os dados da tabela
print('-'*30)
result = session.query(Funcionario).all()           # retorna lista de objetos
for i in result:
    print(i.id, i.nome, i.idade, i.salario)

# Busca um dado específico (pela primary key)
print('-'*30)
func = session.query(Funcionario).get(2)          # busca pela chave primária e retorna o objeto
print(func.id, func.nome, func.idade, func.salario)

# Busca utilizando filtros
# salario maior que 1500 (.all() para retornar todos)
print('-'*30)
d = session.query(Funcionario).filter(Funcionario.salario>1500).order_by(Funcionario.nome).all()
for i in d:
    print(i.id, i.nome, i.idade, i.salario)

# Busca utilizando vários filtros (.all() para retornar todos)
print('-'*30)
d = session.query(Funcionario).filter(Funcionario.idade == 22,
Funcionario.nome.like('%inho%')).all()
for i in d:
    print(i.id, i.nome, i.idade, i.salario)

# Busca utilizando filtros (.first() retorna apenas o primeiro)
print('-'*30)
d = session.query(Funcionario).filter(Funcionario.idade == 22,
Funcionario.nome.like('%inho%')).first()
print(d.id, d.nome, d.idade, d.salario)
```

Alterar dados

As alterações realizadas no objeto automaticamente serão replicadas no banco de dados.

```
# Alterar um registro
func = session.query(Funcionario).get(1)          # busca um objeto
func.nome = 'Zezinho da Silva'                    # altera os atributos do objeto
func.idade = 25
session.commit()                                  # realiza o commit
```

Excluir dados

```
# Excluir um registro
func = session.query(Funcionario).get(2)          # busca um objeto
session.delete(func)                              # deleta o objeto
session.commit()
```

Fechando a conexão

Após realizar as operações no banco, a conexão deve ser fechada.

```
# Fechar conexão com banco de dados
connection.close()
```