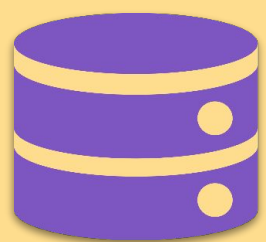
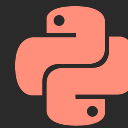


Uma visão geral sobre



# SQLModel

Live de Python # 235



## 1. SQLAlchemy

Uma introdução

## 2. Criando modelos

Trabalhando com DDL / DML

## 3. Executando consultas

Quando o SQLAlchemy aparece

## 4. Criando um CRUD

E uma aplicaçãozinha básica



[picpay.me/dunossauro](https://picpay.me/dunossauro)



[apoia.se/livedepython](https://apoia.se/livedepython)



[pix.dunossauro@gmail.com](mailto:pix.dunossauro@gmail.com)



Ajude o projeto <3



Ademar Peixoto, Adilson Herculano, Adriano Ferraz, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Andre Azevedo, Andre Mesquita, Aquiles Coutinho, Arnaldo Turque, Aslay Clevisson, Aurelio Costa, Bernardo At, Bernardo Fontes, Bruno Almeida, Bruno Barcellos, Bruno Barros, Bruno Batista, Bruno Freitas, Bruno Lopes, Bruno Ramos, Caio Nascimento, Christiano Moraes, Damianth, Daniel Freitas, Daniel Wojcickoski, Danilo Boas, Danilo Segura, Danilo Silva, David Couto, David Kwast, Davi Goivinho, Delton Porfiro, Denis Bernardo, Dgeison Peixoto, Diego Farias, Diego Guimarães, Dilenon Delfino, Diogo Paschoal, Edgar, Edinilson Bonato, Eduardo Tolmasquim, Elias Silva, Emerson Rafael, Érico Andrei, Everton Silva, Fabiano, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Felipe Rodrigues, Fernanda Prado, Fernando Celmer, Firehouse, Flávio Meira, Francisco Neto, Gabriel Barbosa, Gabriel Espindola, Gabriel Mizuno, Gabriel Moreira, Gabriel Paiva, Gabriel Simonetto, Gabriel Souza, Geandreson Costa, Gilberto Abrao, Giovanna Teodoro, Giuliano Silva, Guilherme Felitti, Guilherme Gall, Guilherme Silva, Guionardo Furlan, Gustavo Pereira, Gustavo Suto, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Hugo Cosme, Igor Riegel, Italo Silva, Janael Pinheiro, Jhonatan Martins, Joelson Sartori, Jônatas Silva, Jon Cardoso, Jorge Silva, Jose Alves, José Gomes, Joseíto Júnior, Jose Mazolini, Juan Felipe, Juan Gutierrez, Juliana Machado, Julio Franco, Júlio Gazeta, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Miranda, Leandro Silva, Leo Ivan, Leonardo Mello, Leonardo Nazareth, Leon Solon, Luancomputacao Roger, Lucas Adorno, Lucas Carderelli, Lucas Mendes, Lucas Nascimento, Lucas Schneider, Lucas Simon, Lucas Valino, Luciano Filho, Luciano Ratamero, Luciano Silva, Luciano Teixeira, Luis Alves, Luiz Duarte, Luiz Lima, Luiz Paula, Luiz Perciliano, Mackilem Laan, Marcelo Campos, Marcio Moises, Marco Mello, Marcos Gomes, Maria Clara, Marina Passos, Mateus Lisboa, Matheus Silva, Matheus Vian, Mauricio Nunes, Mirian Batista, Mlevi Lsantos, Murilo Viana, Natan Cervinski, Nathan Branco, Nicolas Teodosio, Osvaldo Neto, Otávio Carneiro, Patricia Minamizawa, Patrick Felipe, Paulo D., Paulo Tadei, Pedro Henrique, Pedro Pereira, Peterson Santos, Priscila Santos, Pydocs Pro, Pytonyc, Rafael Lopes, Rafael Romão, Rafael Veloso, Raimundo Ramos, Ramayana Menezes, Regis Santos, Renato Oliveira, Rene Bastos, Ricardo Silva, Riverfount, Rjribeiro, Robson, Robson Maciel, Rodrigo Freire, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Ribeiro, Rodrigo Vaccari, Rodrigo Vieira, Rogério Lima, Rogério Nogueira, Ronaldo Silveira, Rui Jr, Samanta Cicilia, Thaynara Pinto, Thiago Araujo, Thiago Borges, Thiago Curvelo, Tiago Minuzzi, Tony Dias, Tyrone Damasceno, Uadson Emile, Valcilon Silva, Valdir Tegon, Vcwild, Vicente Marcal, Vinicius Stein, Vladimir Lemos, Walter Reis, Willian Lopes, Wilson Duarte, Wilson Neto, Xico Silvério, Yuri Fialho, Zeca Figueiredo



Obrigado você



Uma introdução

# SQL Model

# SQLModel



- SQLModel é um projeto criado por **Sebastián Ramírez**
  - Mesmo criador do **FastAPI** e do **Typer**
- Teve sua primeira versão: **Agosto de 2021**
- Versão atual: **0.8.0**
- Licença: **MIT**
- Usa como base as bibliotecas:
  - SQLAlchemy
  - Pydantic

# ORM



**SQLModel** é um **ORM**. **Object-Relational Mapper** [Mapeador relacional de objeto].

Em outras palavras, é uma forma de representar(mapear) uma tabela de banco de dados em "objeto" de código.

```
1  from typing import Optional
2  from sqlmodel import SQLModel, Field
3
4
5  class Todo(SQLModel, table=True):
6      id: Optional[int] = Field(default=None, primary_key=True)
7      title: str
8      status: str = 'todo'
```

# ORM



**SQLModel** é um **ORM**. **Object-Relational Mapper** [Mapeador relacional de objeto].

Em outras palavras, é uma forma de representar(mapear) uma tabela de banco de dados em "objeto" de código.

| ID | TITLE           | STATUS  |
|----|-----------------|---------|
| 1  | Lavar Louça     | a fazer |
| 2  | Fazer live      | fazendo |
| 3  | Terminar Slides | pronto  |



# Json Schema



**SQLModel** é um **serealizador json**.

Isso quer dizer que ele consegue validar estruturas de objetos json ou dicionários em python

```
1  from typing import Optional
2  from sqlmodel import SQLModel, Field
3
4
5  class Todo(SQLModel, table=True):
6      id: Optional[int] = Field(default=None, primary_key=True)
7      title: str
8      status: str = 'todo'
```

# Json Schema



**SQLModel** é um **serealizador json**.

Isso quer dizer que ele consegue validar estruturas de objetos json ou dicionários em python

```
1  {  
2      "id": 1,  
3      "todo": "Lavar louça",  
4      "status": "a fazer"  
5  }
```

# O que é o SQLAlchemy?



SQLModel é a junção de duas ferramentas poderosas incríveis do ambiente Python. Caso queira entendê-las separadamente.

- SQLAlchemy: "O" ORM
  - **Lives de Python # 11, 139, 166, 211**
- Pydantic: "O" serializador
  - **Live de Python # 165**

**Basicamente nos dá o poder de usar schemas de json unificados ao banco de dados. Ele é os dois modelos ao mesmo tempo!**

# SQLModel



O SQLModel foi criado tendo em vista o **FastAPI** em mente, pois ele usa o pydantic como base e a biblioteca mais comum de usar em conjunto é o SQLModel. Porém, o **Quart** [framework web async da pallets], também pode suportar ele com o Quart-Model.

SQLModel não é a primeira tentativa de resolver esse problema, no passado Sebastian criou também o **pydantic-sqlalchemy**.

```
pip install sqlmodel
```



Instalação



# Um olá mundo dos bancos de dados



```
1  from typing import Optional
2  from sqlmodel import SQLModel, Field
3
4
5  class Todo(SQLModel, table=True):
6      id: Optional[int] = Field(default=None, primary_key=True)
7      title: str
8      status: str = 'todo'
```

Dessa forma, podemos usar essa classe



```
1  t1 = Todo(title='Lavar louça')
2  t2 = Todo(id='10', title='Fazer Live', status='doing')
3  t3 = Todo(title='Terminar slides', status='done')
4  t4 = Todo(**{'title': 'Dormir quando acabar a live'})
```

DDL /  
DML

Criação das tabelas  
e modelos



# Para criar a base de dados

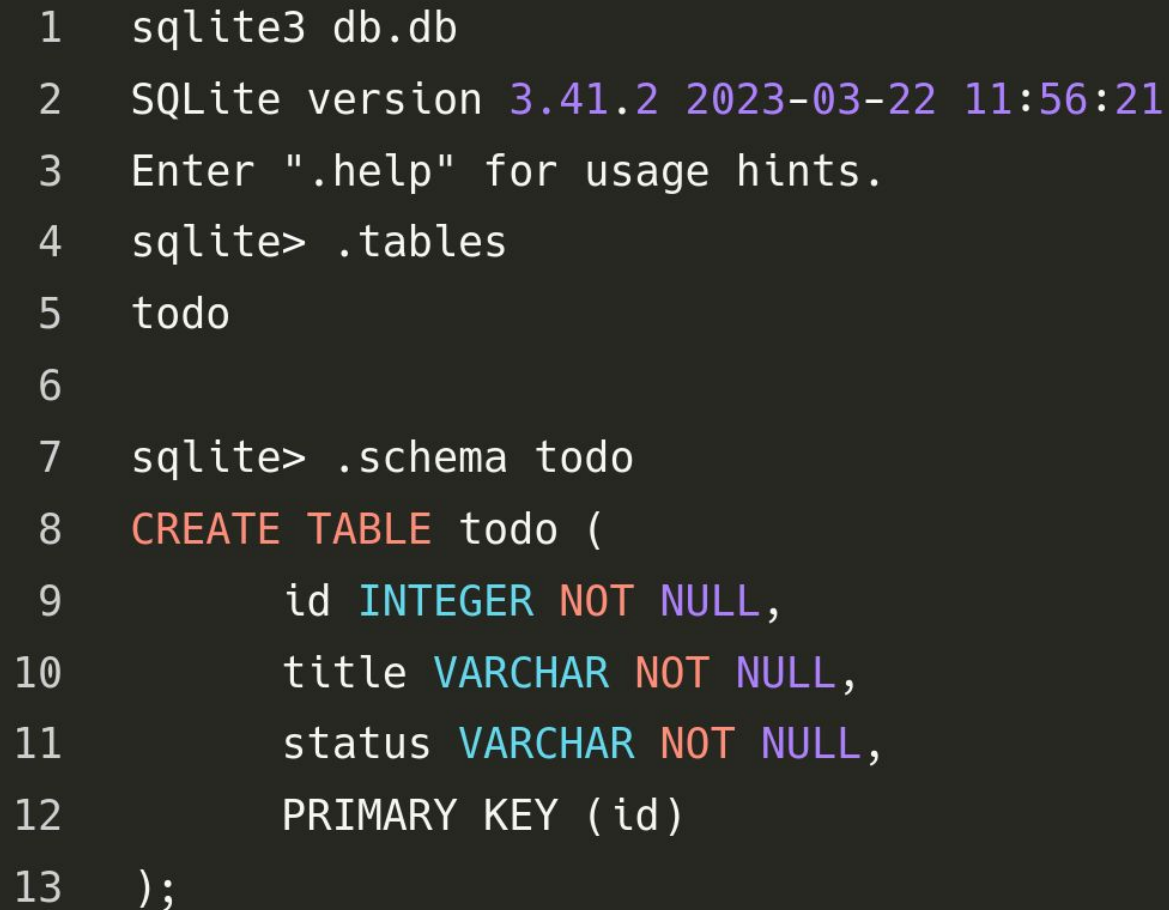


Usarei **sqlite** na live de hoje para simplificar. O **metadata.create\_all** está sendo usado para criar todas as classes que contém `table=True` que foram definidas **ANTES** do create all.

```
1 engine = create_engine('sqlite:///db.db')
2
3 SQLAlchemy.metadata.create_all(engine)
```

*\*\*o sqlalchemy ainda não tem suporte nativo a migrações, falaremos disso mais tarde\*\**

# Tabela criada



```
1  sqlite3 db.db
2  SQLite version 3.41.2 2023-03-22 11:56:21
3  Enter ".help" for usage hints.
4  sqlite> .tables
5  todo
6
7  sqlite> .schema todo
8  CREATE TABLE todo (
9      id INTEGER NOT NULL,
10     title VARCHAR NOT NULL,
11     status VARCHAR NOT NULL,
12     PRIMARY KEY (id)
13 );
```

# SQLAlchemy



Agora que temos os dados criados, o SQLAlchemy nos fornece um objeto **Session**, esse objeto é a sessão do **SQLAlchemy** e se comporta da mesma forma.

```
1  with Session(engine) as session:
2      session.add(t1)
3      session.commit()
4      session.refresh(t1)
5      print(t1)
```

```
sqlite3 db.db
sqlite> select * from todo;
1|Lavar louça|todo
```

# Relacionamentos PARTE 1



```
1 class Person(SQLModel, table=True):
2     id: Optional[int] = Field(default=None, primary_key=True)
3     name: str
4
5 class Todo(SQLModel, table=True):
6     id: Optional[int] = Field(default=None, primary_key=True)
7     title: str
8     status: str = 'todo'
9     person_id: int = Field(foreign_key='person.id')
```

# Adicionando relacionamento



O que nos traz um pouco de complexidade, mas um relacionamento existe

```
1  with Session(engine) as session:
2      p1 = Person(name='Fausto')
3      session.add(p1)
4      session.commit()
5
6      t1 = Todo(title='Lavar louça', person_id=p1.id)
7      session.add(t1)
8      session.commit()
```

# DQL

Fazendo consultas

# A session



A session do SQLAlchemy é quem é responsável por executar as queries. O método **exec** é o responsável por executá-las e **select** é quem monta as queries.

```
1  from sqlalchemy import Session, select
2
3  with Session(engine) as session:
4      query = select(Todo)
5      session.exec(query).all()
```

# Select



O select conta com vários métodos úteis e quem podem te ajudar a desenvolver as buscas de forma mais eficiente:

- **.limit(6)**: Limita o resultado em 6
- **.offset(10)**: cria paginação de 10 em 10
- **.where(Todo.id > 2, Todo.title == 'Acordar')**: Especifica filtros na busca
- **.order\_by(Todo.id.desc(), Todo.id)**

```
1  from sqlalchemy import Session, select
2
3  with Session(engine) as session:
4      query = select(Todo).where(Todo.id > 6).limit(6).offset(6)
5      session.exec(query).all()
```



# Relacionamentos 2

O SQLAlchemy nos provém um objeto chamado **Relationship**. Que nos permite **dentro da sessão**. Acessar e popular relacionamentos!

```
1  from sqlalchemy import Relationship
2
3  class Person(SQLModel, table=True):
4      id: Optional[int] = Field(default=None, primary_key=True)
5      name: str
6      todos: list['Todo'] = Relationship()
7
8
9  class Todo(SQLModel, table=True):
10     id: Optional[int] = Field(default=None, primary_key=True)
11     title: str
12     status: str = 'todo'
13     person_id: int = Field(foreign_key='person.id')
```

# Acessando tarefas pela pessoa

```
1  from sqlalchemy import Session, select
2
3  with Session(engine) as s:
4      query = select(Person).where(Person.id == 1)
5      person = s.exec(query).one()
6      for todo in person.todos:
7          print(person.name, todo)
8
9  # Marcelo Freitas title='Vizitar vovó' id=1 person_id=1 status='done'
```

# Populando um pelo outro



```
1 class Todo(SQLModel, table=True):
2     id: Optional[int] = Field(default=None, primary_key=True)
3     title: str
4     status: str = 'todo'
5     person_id: int = Field(default=None, foreign_key='person.id')
6     person: Optional['Person'] = Relationship(back_populates='todos')
7
8 class Person(SQLModel, table=True):
9     id: Optional[int] = Field(default=None, primary_key=True)
10    name: str
11    todos: Optional[list[Todo]] = Relationship(back_populates='person')
```

## Populando um pelo outro



```
1  with Session(engine) as s:  
2      p = Person(  
3          name='Dudu',  
4          todos=[Todo(title='IAUUUUUUUUU' )]  
5      )  
6      s.add(p)  
7      s.commit()
```

# Conectivos lógicos (usando where mais a fundo)



```
1  from sqlalchemy import Session, select, and_, or_
2
3  with Session(engine) as s:
4      query = select(Todo).join(Person).where(
5          and_(Person.id < 10, Todo.title == "IAUUUUUUUUU")
6      )
7      print(s.exec(query).all())
8
9      query = select(Todo).join(Person).where(
10         or_(Person.id < 10, Todo.title == "IAUUUUUUUUU")
11     )
12     print(s.exec(query).all())
```

# CRUD

Criando um  
pequeno APP

CODAAARRRR



BORA



# Pontos e observações



- **Ainda não existe suporte nativo para Async**
- Ainda não existe suporte nativo para Migrations

```
1  from sqlalchemy.ext.asyncio import AsyncSession, create_async_engine
2  from sqlalchemy.orm import sessionmaker
3
4  engine = create_async_engine(DATABASE_URL, echo=True, future=True)
5
6  async_session = sessionmaker(
7      engine, class_=AsyncSession, expire_on_commit=False
8  )
9      async with async_session() as session:
10         ...
```



# Pontos e observações



- Ainda não existe suporte nativo para Async
- **Ainda não existe suporte nativo para Migrations**

```
1  # migrations/env.py
2  ...
3  from sqlalchemy import SQLAlchemy
4  from app.models import *
5
6  ...
7
8  target_metadata = SQLAlchemy.metadata
```



[picpay.me/dunossauro](https://picpay.me/dunossauro)



[apoia.se/livedepython](https://apoia.se/livedepython)



[pix.dunossauro@gmail.com](mailto:pix.dunossauro@gmail.com)



Ajude o projeto <3

