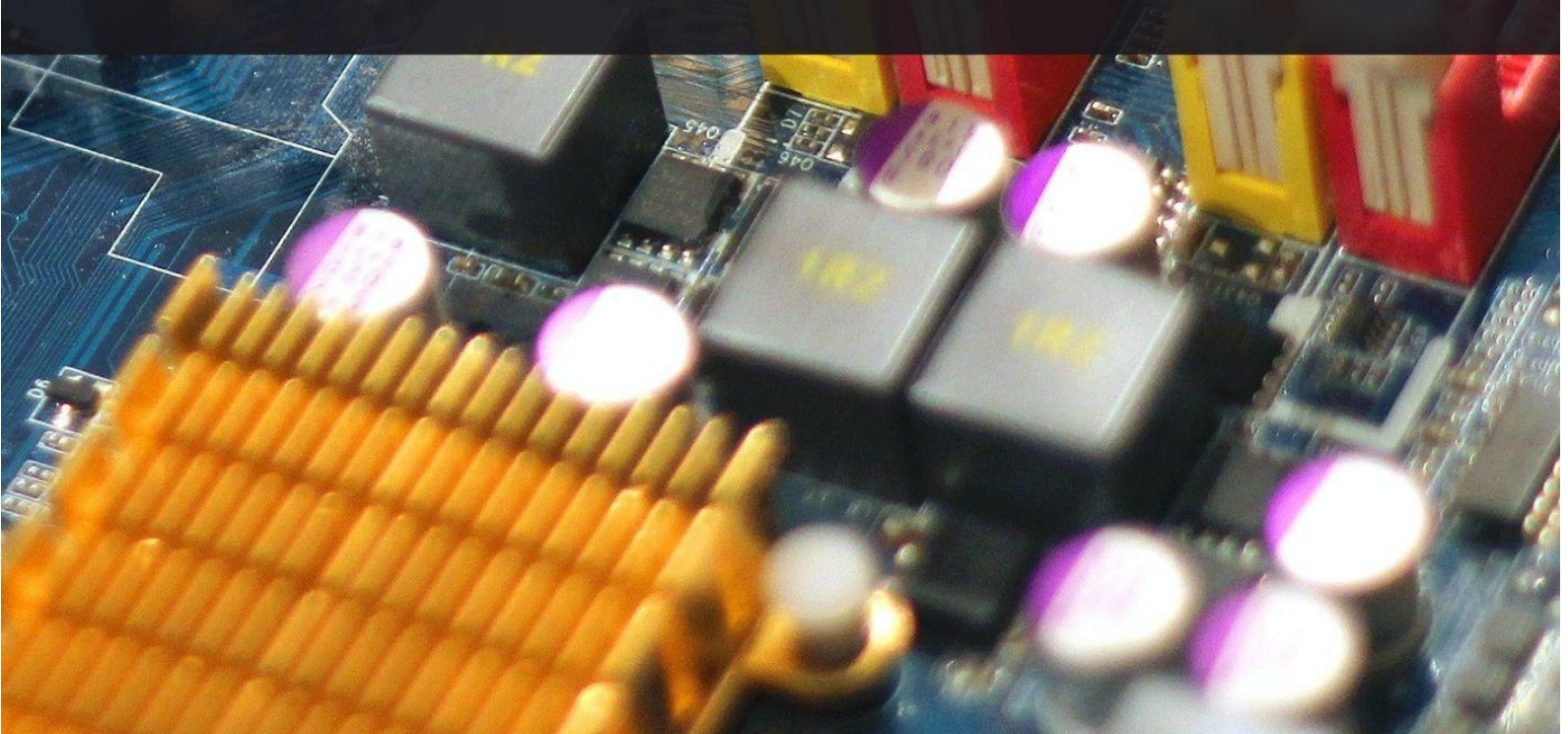


# DESENVOLVIMENTO DE APIs E MICROSSERVIÇOS





# 2

## SQL no python com SQLAlchemy

Lucas Mendes Marques Gonçalves

### *Resumo*

*Você consegue integrar seus conhecimentos de SQL com seus conhecimentos de python? Nessa aula, esse é o objetivo: revisar SQL se necessário, e integrar ele no python usando a biblioteca SQLAlchemy.*

*Especificamente, vamos aprender a instalar o SQLAlchemy, conectar com banco de dados sqlite, abrir um banco de dados sqlite usando o site [www.sqliteonline.com](http://www.sqliteonline.com), e revisar os comandos SQL select, update, create e join.*

### 2.1. SQLAlchemy

O SQLAlchemy é uma biblioteca para conectar o python em bancos de dados. Podemos nos conectar a bancos como Posgres, MySQL e MariaDB. O SQLAlchemy facilita usar sistemas de bancos de dados relacionais de forma a escrever código igual ou muito similar, e trocar o banco de dados quando quiser.

A instalação do SQLAlchemy é muito simples, basta executar **pip install --user sqlalchemy** no terminal integrado do VSCode (ou no CMD do windows). Se esse procedimento falhar, faça referência ao final desse documento, onde explicamos algumas resoluções de problemas com a instalação.

### 2.2. sqlite

O sqlite é um sistema de bancos de dados relacionais muito simplificado. Guarda todas as informações do banco de dados em um único arquivo, e não precisa de nenhuma configuração adicional. Usaremos o sqlite nessa aula.

Do ponto de vista de desenvolvimento, a conversão de um código usando sqlite, para outro banco de dados relacional é muito simples. Os códigos fornecidos com essa aula também funcionam, com alterações mínimas, no banco de dados Postgres.

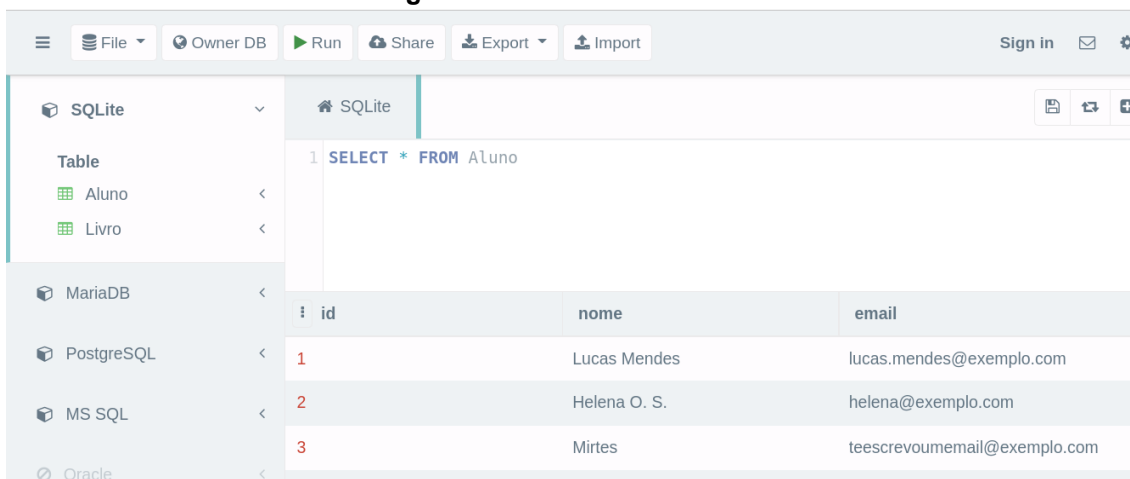
Para podermos desenvolver mais confortavelmente, usaremos o site [www.sqliteonline.com](http://www.sqliteonline.com) para ler nosso arquivo de banco de dados e experimentar queries

sql. Feito isso, teremos os comandos SQL desejados. Os implementaremos em python, usando o sqlalchemy.

### 2.3. Banco de dados da biblioteca

O exemplo que seguiremos nesta aula é um banco de dados de uma biblioteca. Envie o arquivo biblioteca.db para o site [www.sqliteonline.com](http://www.sqliteonline.com), clicando file > open db e selecionando o arquivo.

**Figura 2.1. Tabela alunos**



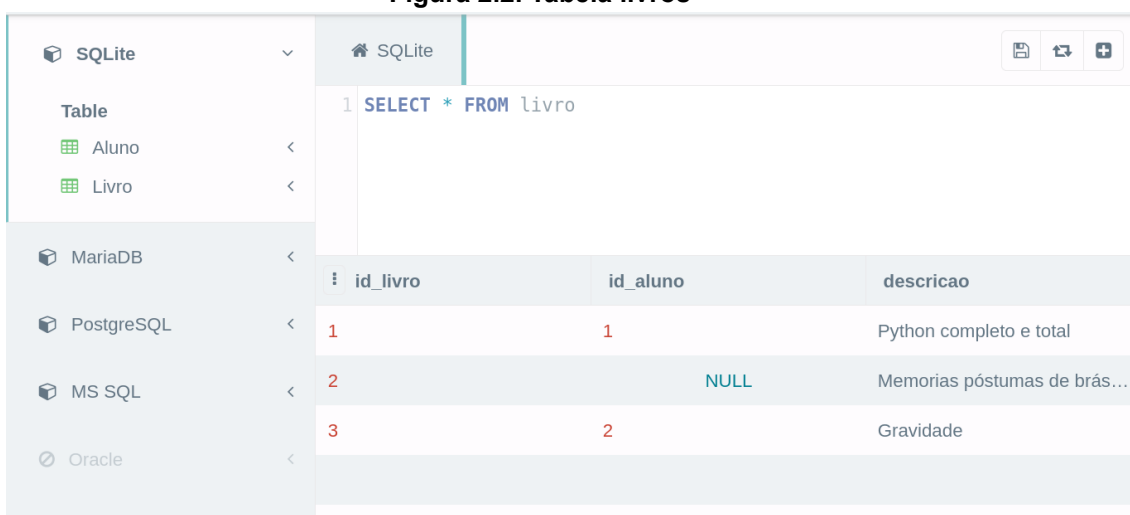
id	nome	email
1	Lucas Mendes	lucas.mendes@exemplo.com
2	Helena O. S.	helenas@exemplo.com
3	Mirtes	teescrevouemail@exemplo.com

Fonte: [sqliteonline.com](http://sqliteonline.com), 2021

Repare, na figura 2.1, que podemos ver as tabelas disponíveis (aluno e livro). Executamos um comando sql (**select \* from aluno**) e podemos ver a resposta, com 3 alunos.

A tabela alunos tem as colunas **id**, **nome** e **email**.

**Figura 2.2. Tabela livros**



id_livro	id_aluno	descricao
1	1	Python completo e total
2	NULL	Memorias póstumas de brás...
3	2	Gravidade

Fonte: [sqliteonline.com](http://sqliteonline.com), 2021

De acordo com a figura 2.2, a tabela livros tem as colunas **id\_livro**, **id\_aluno** e **descricao**.

A parte mais interessante dessa tabela é a coluna **id\_aluno**, que representa uma relação entre alunos e livros. Se **id\_aluno** for 1, o livro está emprestado para o aluno de id 1. Se for **NULL**, o livro está na biblioteca, aguardando alguém que queira pegá-lo emprestado.

## 2.4. Código de inicialização

Não discutiremos o código de inicialização (importar bibliotecas, criar tabelas) nessa aula. Discutiremos brevemente em outra ocasião.

### Codificação 2.1. Código de inicialização

```
from sqlalchemy import create_engine
from sqlalchemy.sql import text

#quero usar o banco de dados nesse arquivo, usando o formato sqlite
engine = create_engine('sqlite:///biblioteca.db')

#mas se quisesse uma solução mais robusta, poderia
#usar mudando o código muito pouco

#engine =
create_engine('postgresql://usuario:senha@localhost:5432/imobiliaria')

#criar a tabela
def criar_tabelas():
    with engine.connect() as con:
        create_tabela_aluno = """
        CREATE TABLE IF NOT EXISTS Aluno (
            id INTEGER PRIMARY KEY,
            nome TEXT NOT NULL,
            email TEXT NOT NULL UNIQUE
        )
```

Núcleo de Educação a Distância | Faculdade Impacta

```
add_livro = "INSERT INTO Livro (id_livro, descricao)
VALUES (2,'Memórias póstumas de brás cubas')"

rs = con.execute(add_livro)

add_livro = "INSERT INTO Livro (id_livro, id_aluno,
descricao) VALUES (3,2,'Gravidade') "

rs = con.execute(add_livro)
```

Fonte: do autor, 2021

## 2.5. Consulta no sqlalchemy

Não é necessário aprender muitas funcionalidades para usar o SQLAlchemy, basta saber criar uma query, reservando espaços para os inputs de usuário, como:

### Codificação 2.2. Consulta sqlalchemy

```
sql_consulta = text ("SELECT * FROM aluno WHERE id = :id_do_aluno")
```

Fonte: do autor, 2021

Repare no :id\_do\_aluno, um espaço reservado para inputs externos.

Executar uma query, preenchendo tais espaços reservados:

### Codificação 2.3. Consulta sqlalchemy - 2

```
rs = con.execute(sql_consulta, id_do_aluno = id_aluno)
```

Fonte: do autor, 2021

Pegar uma linha da resposta.

### Codificação 2.4. Consulta sqlalchemy - 3

```
result = rs.fetchone()
```

Fonte: do autor, 2021

E verificar se já consumimos todas as linhas da resposta.

### Codificação 2.5. Consulta sqlalchemy - 4

```
if result == None
```

Fonte: do autor, 2021

Isso é tudo que precisamos saber de sintaxe.

Veja abaixo um exemplo completo.

### Codificação 2.6. Consulta sqlalchemy - 5

```
# 1) Crie uma função consulta_aluno que recebe a id de um aluno e
devolve
# um dicionário com os dados desse aluno

#idéia da query no sql
"SELECT * FROM aluno WHERE id = 2"

def consulta_aluno(id_aluno):
    with engine.connect() as con: #conectar ao banco de dados
        sql_consulta = text ("SELECT * FROM aluno WHERE id =
:id_do_aluno")
        # id          : nome da coluna da tabela
        # id_aluno     : nome da variavel python que a funcao recebeu
        # id_do_aluno  : nome do "buraco" definido na query
        sql_consulta, nome passado como argumento do con.execute

        rs = con.execute(sql_consulta, id_do_aluno = id_aluno)

        #executamos a query definida em sql_consulta, preenchendo o
        "buraco" :id_do_aluno

        result = rs.fetchone()

        # fetchone: pega uma linha do resultado

        # se tiver mais linhas, podemos pegar a próxima com outro
        fetchone

        # se não tiver mais linhas, receberemos um None

        if result == None: #se a query retornou 0 linhas, o primeiro
        fetchone já resulta None

            raise AlunoNaoExisteException

        # As linhas que o sqlalchemy devolve são parecidas com
        dicionários,
```

```
# mas não é exatamente a mesma coisa. Por isso, fazemos uma
conversão

return dict(result)
```

Fonte: do autor, 2021

## 2.6. Exercícios

A resolução comentada se encontra no vídeo da aula. O conteúdo da sessão anterior, mais seus conhecimentos prévios de SQL, serão suficientes para fazer esses exercícios, mas o vídeo revisa o SQL quando necessário. O exercício 6 faz uso de **join**.

1b) Crie uma função `todos_alunos` que retorna uma lista com um dicionário para cada aluno.

1c) Crie uma função `todos_livros` que retorna uma lista com um dicionário para cada livro.

2) Crie uma função `cria_livro` que recebe os dados de um livro (id e descrição) e o adiciona no banco de dados.

3) Crie uma função `empresta_livro`, que recebe a id de um livro, a id de um aluno e marca o livro como emprestado pelo aluno.

4) Crie uma função `devolve_livro`, que recebe a id de um livro, e marca o livro como disponível.

5) Crie uma função `livros_parados` que devolve a lista de todos os livros que não estão emprestados por ninguém (uma lista de dicionários, um para cada livro).

6) Crie uma função `livros_do_aluno`, recebe o nome do aluno e devolve a lista de todos os livros que estão com o aluno no momento.

### Você sabia?

O SQLAlchemy permite o uso de diversos sistemas de bancos de dados, sem mudança de código. E você pode obter acesso a um banco de dados pequeno em postgres usando o serviço heroku (ao criar um aplicativo nesse site, gratuitamente). Com isso, poderá alterar a linha de conexão ao banco de dados no código fornecido e tentar rodar com Postgres. Deve funcionar sem muita dificuldade!

## 2.7. Problemas com a instalação

O comando `pip` não é um programa válido - '`pip`' não é reconhecido como um comando interno ou externo.

- 1) Se você está usando linux ou mac, rode o comando de instalação usando `pip3` no lugar do `pip`.
- 2) Se você está usando windows, experimente o comando `python` no `cmd`. Se funcionar (ou seja, o `python` funciona e o `pip` não), sua situação não é usual. Peça ajuda no fórum ou ao professor.

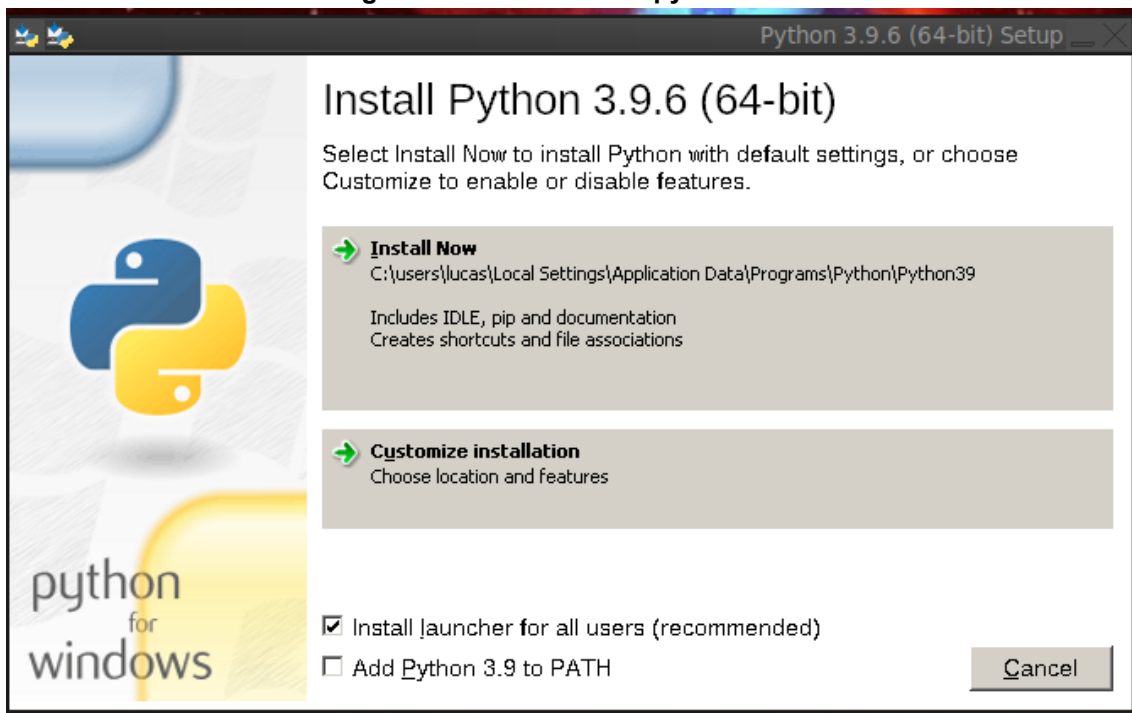


- 3) Se ambos os comandos (pip e python) não funcionarem no cmd, reinstalar o python deve resolver.

Ao reinstalar, marcar a opção “adicionar o python no path” ou “adicionar o python nas variáveis de ambiente” isso faz com que os comandos “python” e “pip” passem a ser comandos válidos no cmd.

Depois de desinstalar e reinstalar, feche o cmd e abra um novo, pra ele carregar os novos comandos.

**Figura 2.3. Instalador do python**



**Fonte: Python 3.9.6 (64bits), 2021**

Na figura 2.3, vemos que a caixa **Add Python 3.9 to PATH**, está desmarcada. Ache essa opção na parte de baixo da imagem.

Essa é a opção que faz com que os comandos **python** e **pip** estejam disponíveis no **cmd**, e deve ser marcada.

## Referências

SQLALCHEMY. **A high level view and getting set up**. SQLAlchemy, 21 jul. 2021. Disponível em: <<https://docs.sqlalchemy.org/en/14/>>. Acesso em: 29 jul. 2021.