# Antes de tudo...

Vamos ver os passos iniciais para instalar, fazer a biblioteca funcionar e outras coisas iniciais para então partirmos para a criação de tabelas

## Instalar a biblioteca

Antes de criar as tabelas, primeiro precisamos configurar o SQLAlchemy e conectar ao banco de dados, isso levando em conta que já se tem outras bibliotecas instalada.

* Instalar as dependências:

|  |
| --- |
| pip install SQLAlchemy |

Se caso esteja utilizando o MySQL, recomenda-se também instalar o drive do mesmo:

|  |
| --- |
| pip install pymysql |

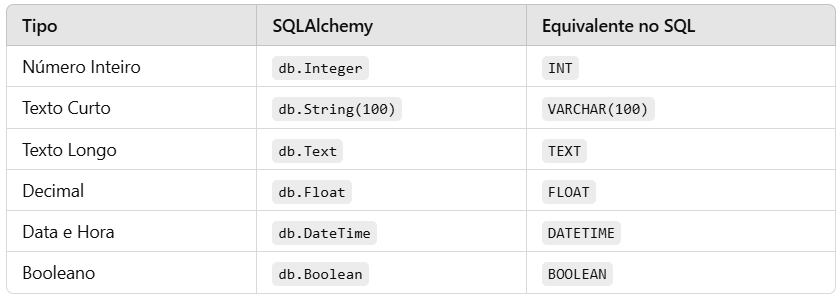
Se por acaso esteja utilizando também o Flask para utilização do SQLAlchemy, pode-se também instalar:

|  |
| --- |
| pip install flask flask-sqlalchemy |

* Criando a aplicação (com Flask) e conectando ao banco:

|  |
| --- |
| from flask import Flask  from flask\_sqlalchemy import SQLAlchemy  app = Flask(\_\_name\_\_)  *# Configuração do banco de dados (MySQL com PyMySQL)*  app.config["SQLALCHEMY\_DATABASE\_URI"] = "mysql+pymysql://usuario:senha@localhost/nome\_do\_banco"  app.config["SQLALCHEMY\_TRACK\_MODIFICATIONS"] = False  db = SQLAlchemy(app) |

# Tipos de dados no SQLAlchemy



# Como criar tabelas?

No SQLAlchemy, cada classe Python representa uma tabela do banco de dados.

Uma observação a ser feita, é que se precisa criar uma base de dados antes de rodar o código para criação de tabelas no SQLAlchemy.

Criando uma tabela chamada “Usuario”:

|  |
| --- |
| class Usuario(db.Model):  id = db.Column(db.Integer, primary\_key=True) # Chave primária  nome = db.Column(db.String(100), nullable=False) # Nome do usuário  email = db.Column(db.String(100), unique=True, nullable=False) # Email único  idade = db.Column(db.Integer, nullable=True) # Idade (pode ser nulo)  data\_criacao = db.Column(db.DateTime, default=db.func.current\_timestamp()) # Data automática |

Mesmo que seja feita uma criação de tabelas através de SQL puro, é altamente recomendado criar as tabelas usando o SQLAlchemy. Os motivos para isso são:

* **Controle total com modelos Python:**
  + Usar SQLAlchemy permite definir toda a estrutura do banco diretamente em código Python.
  + Qualquer alteração futura na estrutura das tabelas pode ser feita facilmente sem precisar escrever SQL manualmente, bastando apenas utilizar o **flask-migrate**.
* **Migrações automáticas com Flask-Migrate**
  + Se você criar tabelas direto no MySQL e depois precisar alterá-las, terá que rodar comandos SQL manualmente. Com o Flask-Migrate, você faz isso de forma automática e segura.
  + Com SQLAlchemy + Flask-Migrate:



* **Facilidade na manutenção e compartilhamento**
* **Portabilidade para outros bancos:**
  + Se no futuro você quiser trocar o MySQL por SQLite ou PostgreSQL, SQLAlchemy faz isso automaticamente, pois abstrai os comandos SQL.