SQLAlchemy

Instalação

- No Terminal digite (Confirme se o **venv** está ativado):
  - pip install SQLAlchemy
  - pip install psycopg2 (Comunicar com o postgreSQL)
  - Obs: Lembre de salvar as versões das bibliotecas (pip freeze > requirements.txt)
- O **SQLAlchemy** é uma biblioteca Python que facilita a interação com bancos de dados relacionais, permitindo manipular dados usando objetos Python em vez de comandos SQL diretos.
- O **psycopg2** é um adaptador Python que permite conectar e interagir com bancos de dados PostgreSQL.

Criar um arquivo: database.py

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

user = "postgres"
password = "password"
database = "nome_database"
host = "localhost"
port = "5432"

SQLALCHEMY_DATABASE_URL = f"postgresql://{user}:{password}@{host}:{port}/{database}"

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

Criar um arquivo: database.py

Cria o motor de conexão (engine) que o SQLAlchemy usa para se comunicar com o banco de dados. Todos os comandos SQL passarão por ele.

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

user = "postgres"
password = "password"
database = "nome_database"
host = "localhost"
port = "5432"

SQLALCHEMY_DATABASE_URL = f"postgresql://{user}:{password}@{host}:{port}/{database}"

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

Cria uma classe base usada para declarar modelos de tabelas.

Cria sessões para interagir com o banco (inserir, consultar, atualizar e deletar dados).

Cria a **classe base** que será usada como **pai** para todos os **modelos de tabela**.

Exige que você chame **commit()** manualmente após as operações.

Evita que o SQLAlchemy envie mudanças ao banco automaticamente antes de certos comandos.

Vincula essa **sessão** ao **engine** criado.

Criar um arquivo: models.py

```
from sqlalchemy import Column, Integer, String, Boolean, TIMESTAMP
from sqlalchemy.sql import text
from database import Base

class Model_Mensagem(Base):
 __tablename__ = 'mensagem'

 id = Column(Integer, primary_key=True, nullable=False)
 titulo = Column(String, nullable=False)
 conteudo = Column(String, nullable=False)
 publicada = Column(Boolean, server_default='True', nullable=False)
 created_at = Column(TIMESTAMP(timezone=True), server_default=text('now()'), nullable=False)
```

Criar um arquivo: models.py

```
from sqlalchemy import Column, Integer, String, Boolean, TIMESTAMP
from sqlalchemy.sql import text
from database import Base

class Model_Mensagem(Base):
 __tablename__ = 'mensagem'

 id = Column(Integer, primary_key=True, nullable=False)
 titulo = Column(String, nullable=False)
 conteudo = Column(String, nullable=False)
 publicada = Column(Boolean, server_default='True', nullable=False)
 created_at = Column(TIMESTAMP(timezone=True), server_default=text('now()'), nullable=False)
```

Vem do arquivo **database.py** (criado com Base = declarative_base()), e serve como classe base para todos os modelos. Note que a classe **Model_Mensagem** está herdando de **Base**.

Tipo: data e hora (com fuso horário)
server_default=text('now()') → o banco preencherá automaticamente a data e hora atual (usando a função now() do PostgreSQL).
Obs: Assim, cada nova mensagem terá sua data de criação registrada automaticamente.

main.py

```
from fastapi import FastAPI
from fastapi.params import Body
from schemas import Mensagem
import models
from database import engine
models.Base.metadata.create_all(bind=engine)

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}

@app.post("/mensagens")
def criar_mensagem(nova_mensagem: Mensagem):
 print(nova_mensagem)
 return {"Mensagem": f"Título: {nova_mensagem.titulo} Conteúdo:
{nova_mensagem.conteudo} Publicada: {nova_mensagem.publicada}"}
```

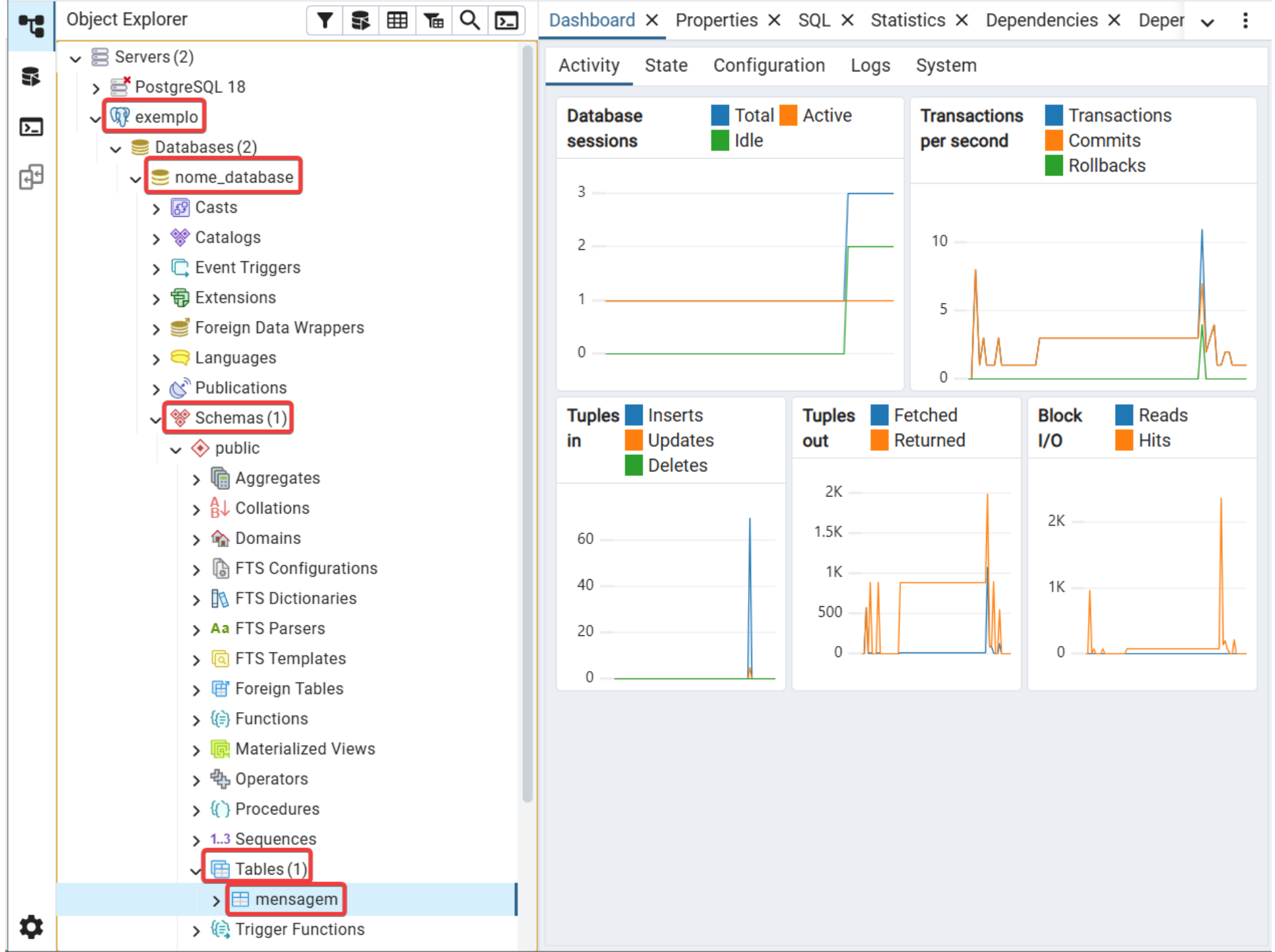
main.py

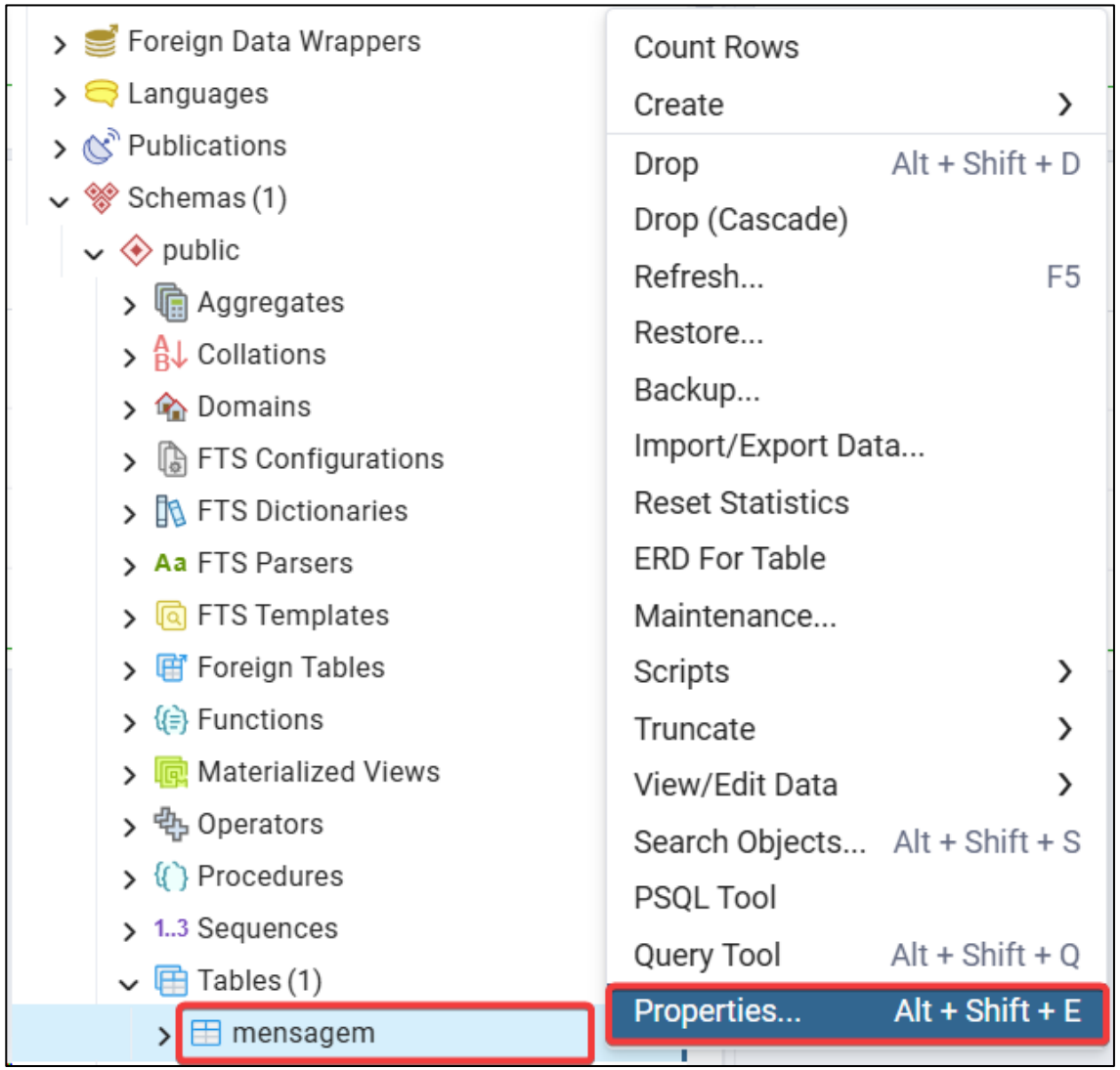
```python
from fastapi import FastAPI
from fastapi.params import Body
from schemas import Mensagem
import models
from database import engine
models.Base.metadata.create_all(bind=engine)

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}

@app.post("/mensagens")
def criar_mensagem(nova_mensagem: Mensagem):
 print(nova_mensagem)
 return {"Mensagem": f"Título: {nova_mensagem.titulo} Conteúdo:
{nova_mensagem.conteudo} Publicada: {nova_mensagem.publicada}"}
```

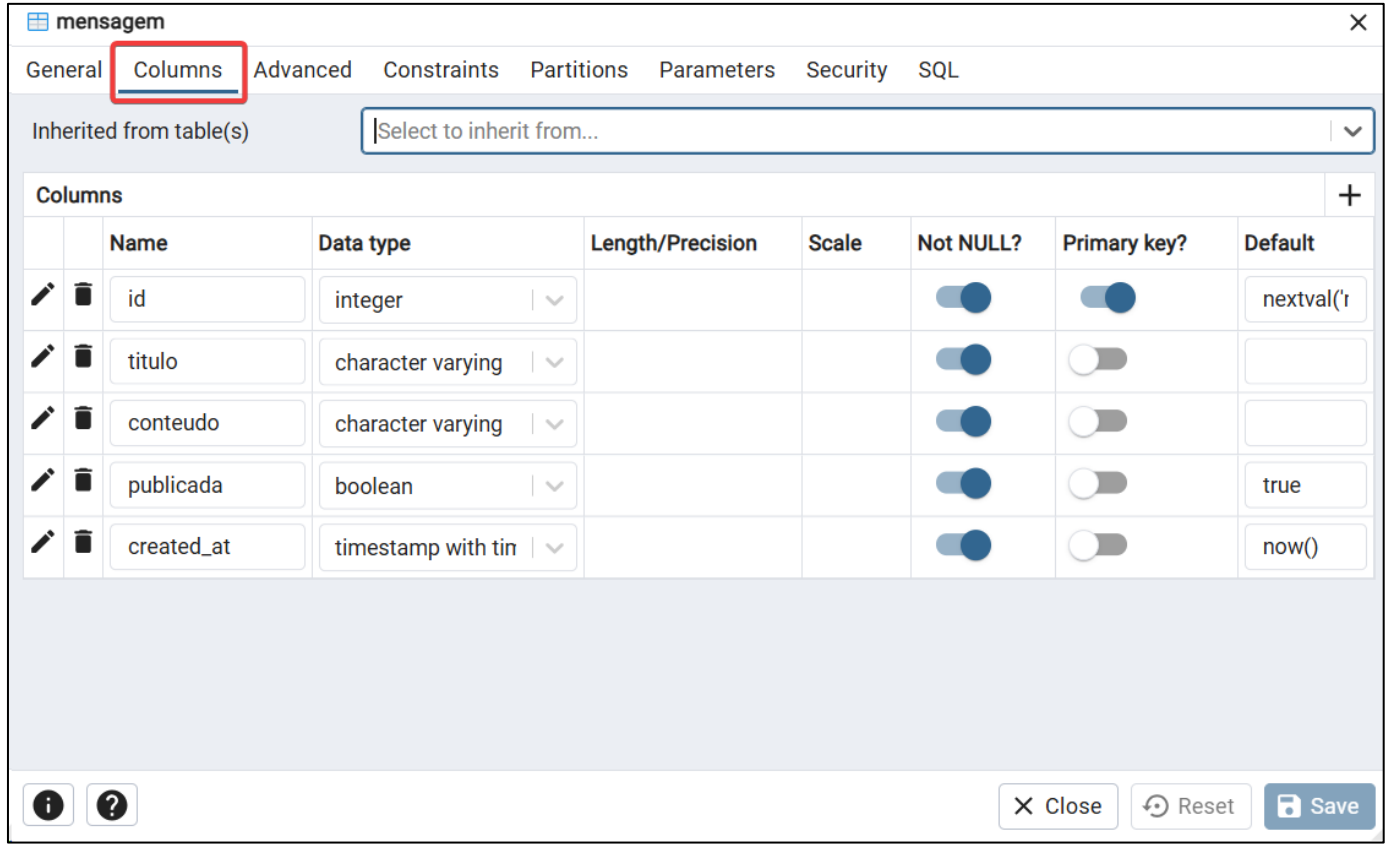
Cria automaticamente no banco todas as tabelas definidas nas classes que herdam de **Base** que estão em **models**. É o comando que transforma suas classes **Python** em tabelas reais no **PostgreSQL**.

Executar!

- **fastapi dev main.py**


Object Explorer
Servers (2)
PostgreSQL 18
exemplo
Databases (2)
nome_database
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (1)
mensagem
Trigger Functions
Dashboard
Properties
SQL
Statistics
Dependencies
Deper
Activity
State
Configuration
Logs
System
Database sessions
Total
Active
Idle
3
2
1
0
Transactions per second
Transactions
Commits
Rollbacks
10
5
0
Tuples in
Inserts
Updates
Deletes
60
40
20
0
Tuples out
Fetched
Returned
2K
1.5K
1K
500
0
Block I/O
Reads
Hits
2K
1K
0


Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (1)
mensagem
Count Rows
Create
Drop Alt + Shift + D
Drop (Cascade)
Refresh... F5
Restore...
Backup...
Import/Export Data...
Reset Statistics
ERD For Table
Maintenance...
Scripts
Truncate
View/Edit Data
Search Objects... Alt + Shift + S
PSQL Tool
Query Tool Alt + Shift + Q
Properties... Alt + Shift + E


mensagem
General
Columns
Advanced
Constraints
Partitions
Parameters
Security
SQL
Inherited from table(s)
Select to inherit from...
Columns
Name
Data type
Length/Precision
Scale
Not NULL?
Primary key?
Default
id
integer
nextval('
titulo
character varying
conteudo
character varying
publicada
boolean
true
created_at
timestamp with tin
now()
Close
Reset
Save

database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

user = "postgres"
password = "admin"
database = "nome_database"
host = "localhost"
port = "5432"

SQLALCHEMY_DATABASE_URL = f"postgresql://{user}:{password}@{host}:{port}/{database}"

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

def get_db():
 db = SessionLocal()
 try:
    yield db
 finally:
    db.close()
```

database.py

```
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Cria uma nova sessão de banco de dados. Essa sessão é o objeto que você usa para fazer operações como:

- db.query(...)
- db.add(...)
- db.commit()

SessionLocal foi criada antes. Ela sabe qual banco usar (engine) e como se comportar.

yield transforma essa função em um gerador, o que significa:

1) O objeto **db** é entregue para quem chamou a função **get_db()**;
2) Quando o uso do **db** termina, a execução continua após o **yield**, indo para o **finally**.

Na main.py:

Chama **get_db()** → cria a sessão.

Em database.py:

Passa **db** para a função na main.py que chamou **get_db()**.

Quando a requisição termina → fecha automaticamente a sessão (finally).

finally: db.close() → Depois que o uso do **db** acaba, o bloco **finally** é executado, independentemente de erro ou sucesso.

main.py

```python
from fastapi import FastAPI, status, Depends
from schemas import Mensagem
import models
from database import engine, get_db
from sqlalchemy.orm import Session

models.Base.metadata.create_all(bind=engine)

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}

@app.post("/mensagens", status_code=status.HTTP_201_CREATED)
def criar_mensagem(nova_mensagem: Mensagem, db_session: Session = Depends(get_db)):
    mensagem_criada = models.Model_Mensagem(**nova_mensagem.model_dump())
    db_session.add(mensagem_criada)
    db_session.commit()
    db_session.refresh(mensagem_criada)
    return {"Mensagem": mensagem_criada}
```

main.py

```python
from fastapi import FastAPI, status, Depends
from schemas import Mensagem
import models
from database import engine, get_db
from sqlalchemy.orm import Session

models.Base.metadata.create_all(bind=engine)

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}

@app.post("/mensagens", status_code=status.HTTP_201_CREATED)
def criar_mensagem(nova_mensagem: Mensagem, db_session: Session = Depends(get_db)):
 mensagem_criada = models.Model_Mensagem(**nova_mensagem.model_dump())
 db_session.add(mensagem_criada)
 db_session.commit()
 db_session.refresh(mensagem_criada)
 return {"Mensagem": mensagem_criada}
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status

Se o banco de dados atribuir automaticamente quaisquer atributos adicionais (como uma chave primária autoincrementada ou valores padrão para determinadas colunas), a instância na sua sessão não terá essas atualizações até que seja feito um ***refresh***.

main.py

```python
from fastapi import FastAPI, status, Depends
from schemas import Mensagem
import models
from database import engine, get_db
from sqlalchemy.orm import Session

models.Base.metadata.create_all(bind=engine)

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}

@app.post("/mensagens", status_code=status.HTTP_201_CREATED)
def criar_mensagem(nova_mensagem: Mensagem, db_session: Session = Depends(get_db)):
 mensagem_criada = models.Model_Mensagem(**nova_mensagem.model_dump())
 db_session.add(mensagem_criada)
 db_session.commit()
 db_session.refresh(mensagem_criada)
 return {"Mensagem": mensagem_criada}
```

Depends() é uma função do FastAPI que implementa injeção de dependência.
Então, quando alguém faz uma requisição para /mensagens:

1. O FastAPI chama a função **get_db()**.
2. Essa função cria uma sessão de banco (**SessionLocal()**), faz **yield** dela e depois a fecha automaticamente (no **finally**).
3. O valor retornado pelo **yield** (**db**) é injetado no parâmetro **db_session** da rota.

model_dump(): transforma o Pydantic em dicionário.
models.Model_Mensagem(...): Cria um objeto SQLAlchemy ORM que pode ser salvo no banco

POST http://127.0.0.1:8000/mensagens Send

Params Authorization Headers (8) Body Scripts Settings Cookies

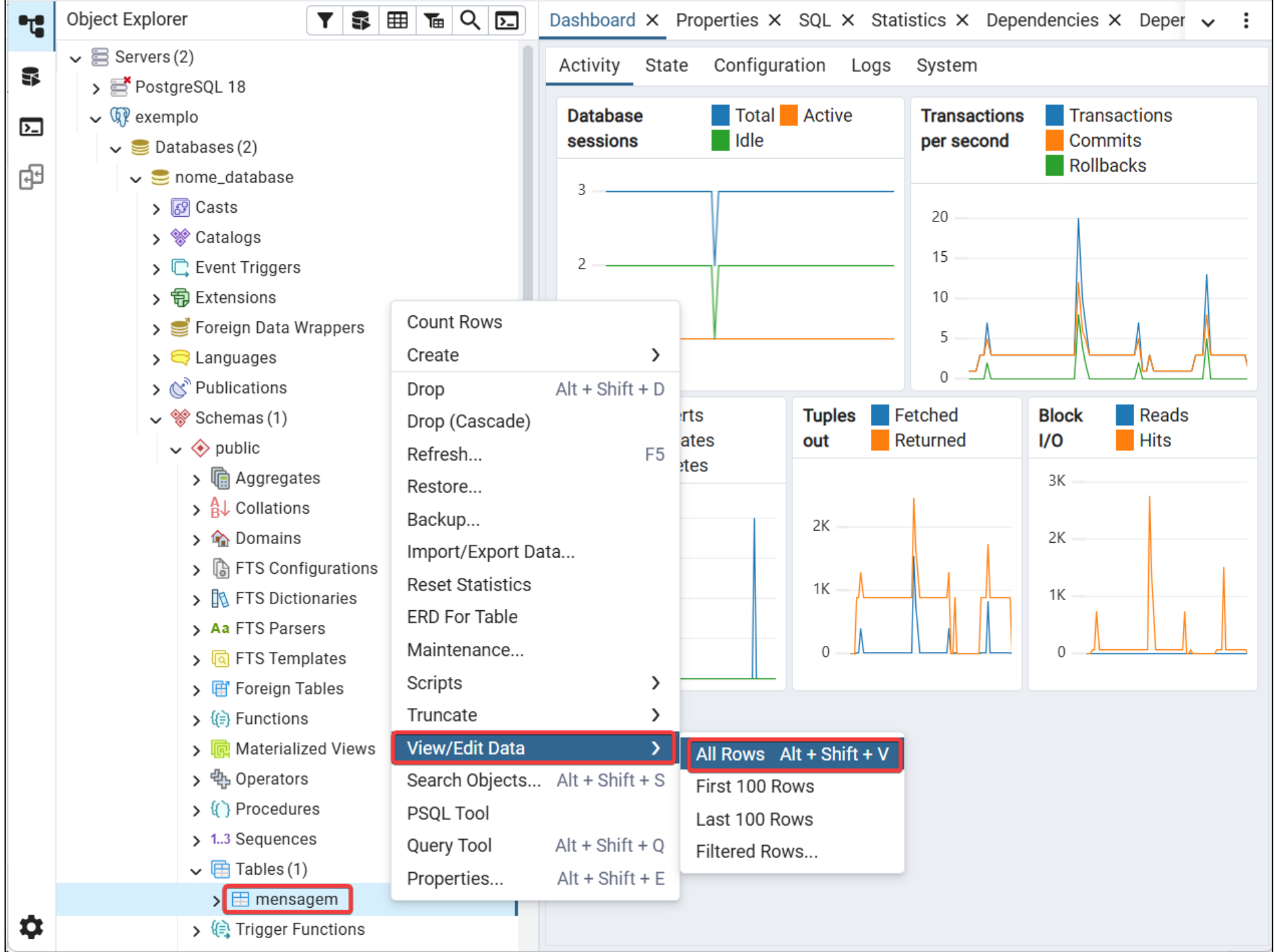
none form-data x-www-form-urlencoded raw binary GraphQL JSON Schema Beautify

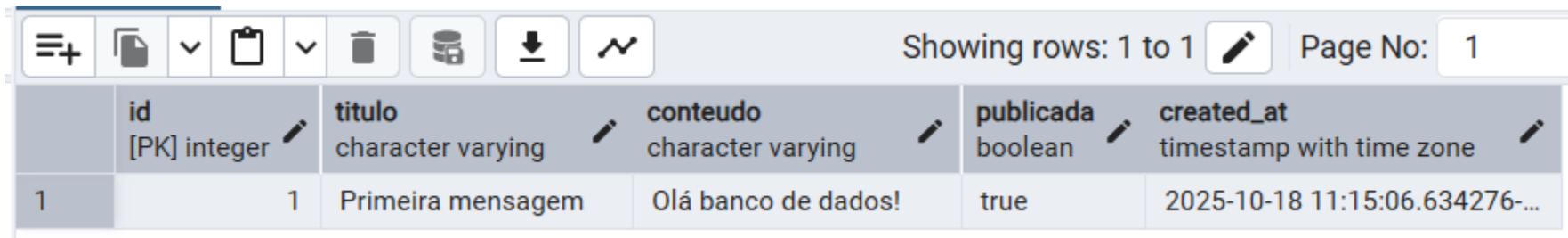
```
{
  "titulo": "Primeira mensagem",
  "conteudo": "Olá banco de dados!"
}
```

Body Cookies Headers (4) Test Results 201 Created 16 ms 280 B Save Response

{} JSON Preview Visualize


```
{
    "Mensagem": {
        "conteudo": "Olá banco de dados!",
        "publicada": true,
        "titulo": "Primeira mensagem",
        "id": 1,
        "created_at": "2025-10-18T11:15:06.634276-03:00"
    }
}
```


Object Explorer
Servers (2)
PostgreSQL 18
exemplo
Databases (2)
nome_database
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (1)
mensagem
Trigger Functions
Count Rows
Create
Drop Alt + Shift + D
Drop (Cascade)
Refresh... F5
Restore...
Backup...
Import/Export Data...
Reset Statistics
ERD For Table
Maintenance...
Scripts
Truncate
View/Edit Data
Search Objects... Alt + Shift + S
PSQL Tool
Query Tool Alt + Shift + Q
Properties... Alt + Shift + E
All Rows Alt + Shift + V
First 100 Rows
Last 100 Rows
Filtered Rows...
Dashboard
Properties
SQL
Statistics
Dependencies
Activity
State
Configuration
Logs
System
Database sessions
Total
Active
Idle
Transactions per second
Transactions
Commits
Rollbacks
Tuples out
Fetched
Returned
Block I/O
Reads
Hits



Showing rows: 1 to 1 Page No: 1

	id [PK] integer	titulo character varying	conteudo character varying	publicada boolean	created_at timestamp with time zone
1	1	Primeira mensagem	Olá banco de dados!	true	2025-10-18 11:15:06.634276-...

Desafio!

- Não deixar suas credenciais de acesso ao banco de dados no código-fonte!

```
user = "postgres"
password = "password"
database = "nome_database"
host = "localhost"
port = "5432"
```