SQL Alchemy

Haz ORO tus datos

Contenido

Qué es y porqué usarlo

Modelado de una tabla

Ingreso y actualización de datos

Consultas

Modelado de relaciones

SQL Alchemy

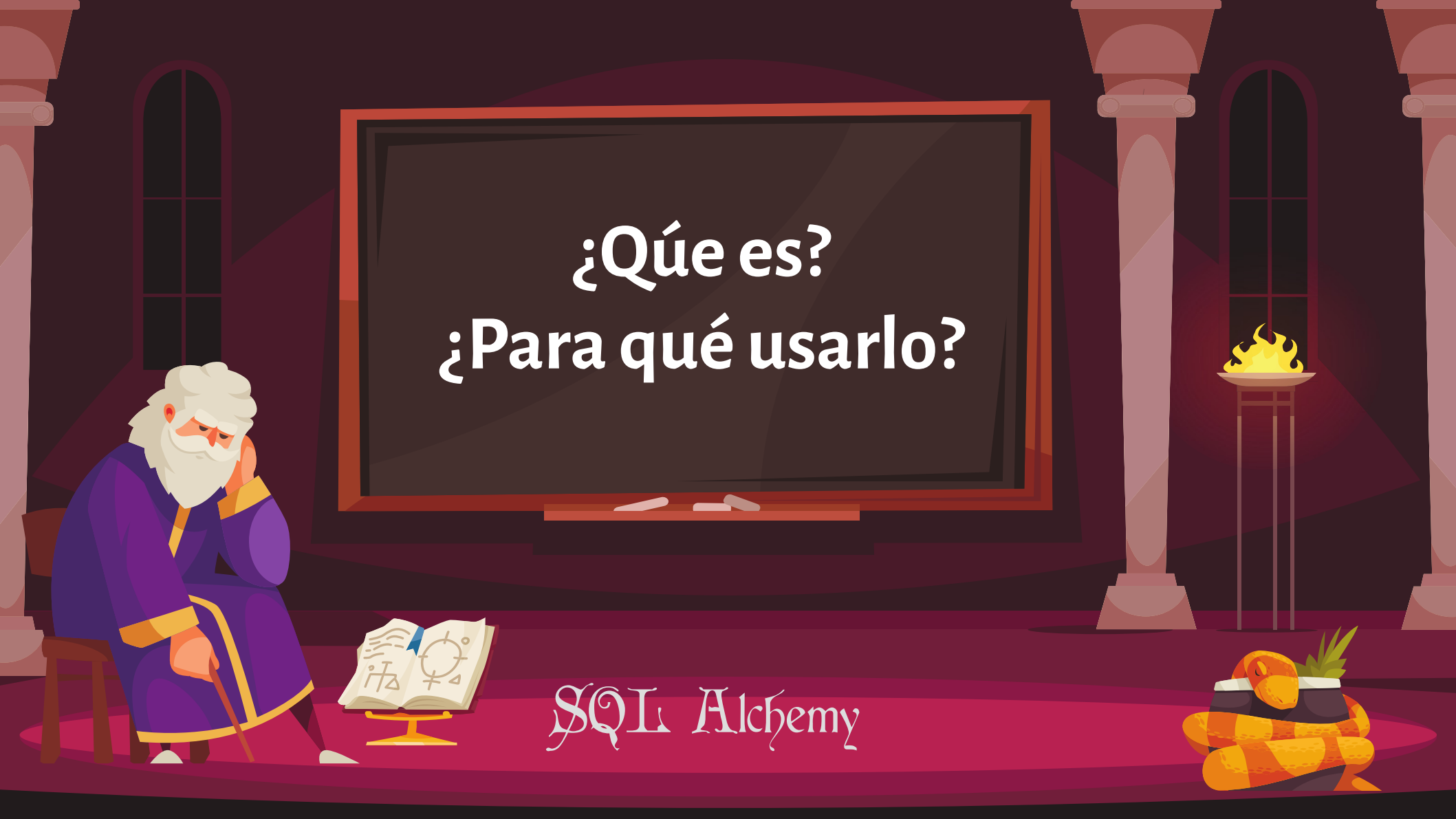
Contenido

Legacy

Extensiones: Migraciones, Flask

SQL Alchemy

¿Qúe es?
¿Para qué usarlo?



SQL Alchemy

Qué es

Relacionarnos con bases de datos

Ayuda a navegar por los datos

SQL Alchemy

Ventajas

Portabilidad entre BBDD.

Sencillez de uso.

Prácticas más seguras.

SQL Alchemy

Inconvenientes

Consultas menos óptimas

Algo más de código para correr.

Curva de aprendizaje

SQL Alchemy


Modelando
SQL Alchemy

Definiendo una tabla

```python
class Base(DeclarativeBase):
  pass

class Client(Base):
  __tablename__ = "client"
  id:Mapped[int] = mapped_column(primary_key=True)
  name: Mapped[str] = mapped_column(String(60))
  balance: Mapped[int]
```

SQL Alchemy

Tipos de datos

String, Unicode
Cadenas.
Text, UnicodeText
Textos sin tamaño máx.

Boolean
Booleano, verdadero o falso

SmallInteger, Integer, BigInteger Enteros

Float, Double Punto flotante

Enum

Pertenece a un conjunto enumerado.

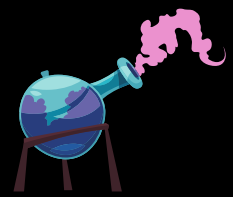
Date, DateTime, Time Fecha, hora

Interval Período

LargeBinary

Datos binarios

PickleType formato pickle.



SQL Alchemy

Argumentos en `mapped_column`

Primary_key: si es clave primaria

Default: valor por defecto

Nullable: Boleano. Define si puede o no ser nulo

Index: Campo indexado.

SQL Alchemy

Conectar con el motor

```
from sqlalchemy import create_engine

engine = create_engine("sqlite://",
                       echo=True)
```

SQL Alchemy

Diferentes motores

SQLite

```
sqlite:///file_path                    sqlite://   (memoria)
```

MySQL

```
mysql+pymysql://user:pass@host/dbname
```

PostGreSQL

```
postgresql+psycopg2://user:pass@host:port/dbname
```

SQL Alchemy

Modificaciones

Sesiones

```
with Session(engine) as session:
    [cambios en las instancias]

    session.flush()
    session.commit()
```

SQL Alchemy

Inserción de datos

```
with Session(engine) as session:
    bob = Client(name="bob", balance=90)
    mary = Client(name="mary", balance=100)
    session.add_all([mary, bob])
    session.commit()
```

SQL Alchemy

Actualizaciones

```
stmt = select(Client).where(Client.name ==
"bob")
bob = session.scalars(stmt).first()


bob.balance +=10
session.commit()
```

SQL Alchemy

Borrado

```
mary = session.get(Client, 1)
session.delete(mary)
session.commit()
```

SQL Alchemy

Estados de la instancia en sesión

Transient: Recién creada, no agregada

Pending: Agregada, sin grabar

Persisted: Grabada, transacción abierta.

Deleted: Borrada, transacción abierta

Detached: Ya confirmada la transacción, y sacada de la sesión

SQL Alchemy

Consultas

SQL Alchemy

Consulta de datos

```
from sqlalchemy import select
session = Session(engine)
stmt = select(Client).where(
    Client.name == "mary")

session.execute(stmt).all()
```

SQL Alchemy

Ordenar y limitar las filas

```
stmt = (select(Client)
          .where(Client.balance > 0)
          .order_by(Client.balance)
          .limit(10))

session.scalars(stmt).all()
```

SQL Alchemy

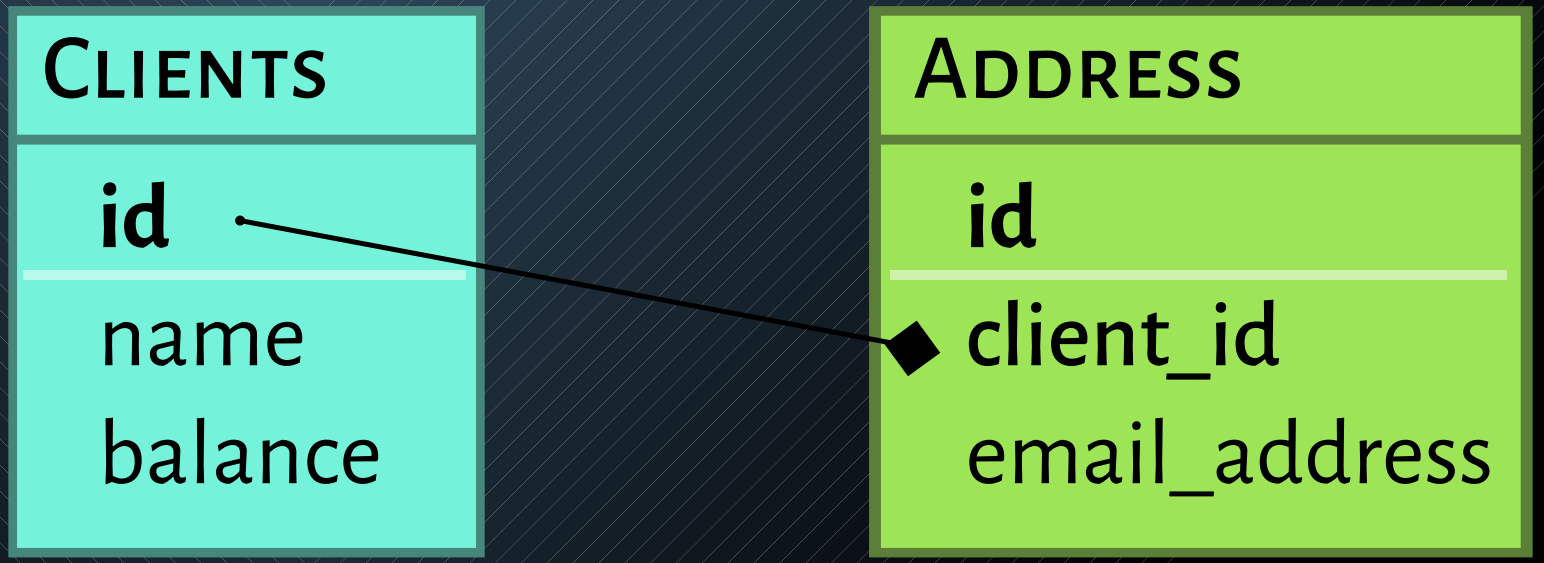
Conectores

```
from sqlalchemy import or_
stmt = select(Client).where(
                    or_(
                        Client.name == 'bob',
                        Client.name == 'mary'
                    ))
res = session.execute(stmt).all()
```

SQL Alchemy

Tablas relacionadas

Tablas relacionadas



Tablas relacionadas

```python
class Client(Base):
    __tablename__ = "clients"
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(60))
    balance: Mapped[float]

class Address(Base):
    __tablename__ = "address"
    id: Mapped[int] = mapped_column(primary_key=True)
    email_address: Mapped[str]
```

SQL Alchemy

Tablas relacionadas

```python
class Address(Base):
    __tablename__ = "address"
    ...
    client_id: Mapped[int] =
        mapped_column(ForeignKey("clients.id"))
    client: Mapped["Client"] =
        relationship(back_populates="addresses")
```

SQL Alchemy

Tablas relacionadas

```
class Client(Base):
    __tablename__ = "clients"
    ...
    addresses: Mapped[list["Address"]] =
        relationship(
                back_populates="client")
```

SQL Alchemy

Grabar usando la relación

```
with Session(engine) as session:
    sandy = Client(name="sandy", balance=120,
        addresses=[
            Address(email_address="sandy@any.org"),
            Address(email_address="sandy@sqpower.org"),
        ])
    session.add(sandy); session.commit()
```

SQL Alchemy

Agregar o borrar

```
sandy = session.get(Client, 3)
new_address=Address(email_address="sandy@new.org")
sandy.addresses.append(new_address)

old_address = sandy.addresses[0]
sandy.addresses.remove(old_address)
```

SQL Alchemy

Consultas con join

```
stmt = (
    select(Address)
    .join(Address.client)
    .where(Client.name == "sandy")
    .where(Address.email_address ==
                    "sandy@new.org")
)
sandy_address = session.execute(stmt).one()
```

SQL Alchemy

Elegir columnas

```
stmt = (select(Client.name,
               Address.email_address)
    .join(Client.addresses)
    .order_by(Client.id, Address.id)
)
result = session.execute(stmt)
```

SQL Alchemy

Seleccionar por relación

```
stmt = select(Client.name).where(
    Client.addresses.any(
        Address.email_address== "sandy@new.org")
)

session.execute(stmt).all()
```

SQL Alchemy

Seleccionar por relación

```
stmt = (select(Client.name)
          .where(~Client.addresses.any()))

session.execute(stmt).all()
```

SQL Alchemy

Agrupando

```
with Session(engine) as session:
    stmt = (
        select(Client.name, func.count(Address.id).label("count"))
        .join(Address)
        .group_by(Client.name)
        .having(func.count(Address.id) > 1)
    )
    result = session.execute(stmt).all()
```

SQL Alchemy

Inspeccionar `inspect()`

Introspección

- Objetos
- Atributos
- Tipos ... y más

```
from sqlalchemy import inspect
```

SQL Alchemy


Código
legado
SQL Alchemy

Pasar consultas SQL crudas

```
engine = create_engine(conn_str)

with engine.connect() as conn:
    stmt = "select x, y from table"
    result = conn.execute(text(stmt))

    row = result.first()
```

SQL Alchemy

Reflexión

```
metadata_obj = MetaData()
metadata_obj.reflect(bind=someengine)
users_table = metadata_obj.tables["users"]
addresses_table = (
        metadata_obj.tables["addresses"])
```

A Code Gen: Generador de código

```
> pip install sqlacodegen


> sqlacodegen --generator dataclasses
sqlite:///database.db
```

SQL Alchemy

Extensiones

SQL Alchemy

Migraciones con Alembic

```
> alembic init alembic
> alembic upgrade head
> alembic history
> alembic downgrade -1
```

SQL Alchemy

Migraciones con Alembic

```
revision = 'ae1027a6acf'
down_revision = '1975ea83b712'
branch_labels = None

def upgrade():
    op.create_table(
        'account',
        sa.Column('id', sa.Integer, primary_key=True),
        sa.Column('name', sa.String(50), nullable=False),
        sa.Column('description', sa.Unicode(200)))

def downgrade():
    op.drop_table('account')
```

SQL Alchemy

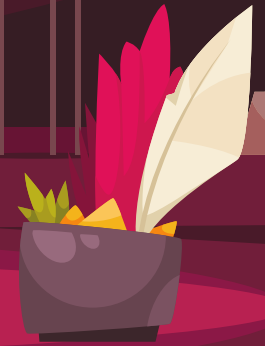
Integración con Flask

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy


db = SQLAlchemy()
app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] =
"sqlite:///project.db"
db.init_app(app)
```

SQL Alchemy

Conclusión



SQL Alchemy

Conclusión

Ampliamente probado

Gran versatilidad

En constante desarrollo.

SQL Alchemy

marian-vignau

mavignau

mavignau.gitlab.io

SQL Alchemy