Для использования sqlalchemy нужно:

1. Создать движок
2. Создать подключение

1 Создание движка:

from sqlalchemy import URL, create\_engine

from config import database\_url

engine = create\_engine(

url=database\_url, - ссылка на базу данных

echo=True, - трансляция записей в консоль

pool\_size=5, - максимальное количество основных подключений

max\_overflow=10 - максимальное количество дополнительных подключений

)

Примеры ссылок:

database\_url = 'sqlite:///sqlite3.db' – sqllite

# Подключение к серверу MySQL на localhost с помощью PyMySQL DBAPI.

engine = create\_engine("mysql+pymysql://root:pass@localhost/mydb")

# Подключение к серверу MySQL по ip 23.92.23.113 с использованием mysql-python DBAPI.

engine = create\_engine("mysql+mysqldb://root:pass@23.92.23.113/mydb")

# Подключение к серверу PostgreSQL на localhost с помощью psycopg2 DBAPI

engine = create\_engine("postgresql+psycopg2://root:pass@localhost/mydb")

# Подключение к серверу Oracle на локальном хосте с помощью cx-Oracle DBAPI.

engine = create\_engine("oracle+cx\_oracle://root:pass@localhost/mydb"))

# Подключение к MSSQL серверу на localhost с помощью PyODBC DBAPI.

engine = create\_engine("oracle+pyodbc://root:pass@localhost/mydb")

Для создания таблицы нужно:

1) Создать объект метадата

2) Создать таблицу

1 Создать объект метадата

From sqlalchemy import MetaData

metadata\_obj = MetaData()

2) Создать таблицу

from sqlalchemy import Table, Column, Integer, String

Workers\_table = Table(

‘workers’, - название таблицу

Metadata\_obj, - объект метадаты

Column(‘id’, Integer, primary\_key=True), - столбец, обязательно один должен быть первичным ключом

Column(‘username’, String),

)

metadata\_obj.create\_all(движок) – создаёт таблицу

metadata\_obj.drop\_all(движок) – удаляет таблицу

Для добавления данных в таблицу:

Есть два способа начало и конец одинаковый, отличается только запрос

From sqlalchemy import text

With engine.connect() as conn:

stmt = """INSERT INTO workers (username) VALUES

('AO Bobr'),

('OOO VOLK');"""

Conn.execut(text(stmt))

Conn.commit() – если чистый sql запрос

From sqlalchemy import select

With engine.connect() as conn:

stmt = insert(workers\_table).values([

{"username": 'AO Bobr'},

{"username": 'OOO VOLK'},

])

Conn.execut(stmt)

Conn.commit() – если sqlalchemy запрос

Второй способ

Используется другая система создания моделей

Класс base общий для всех моделей создается отдельно в фале database

class Base(DeclarativeBase):  
 pass

from sqlalchemy.orm import Mapped, mapped\_column  
from database import Base

class WorkersOrm(Base):  
 \_\_tablename\_\_ = 'workers'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 username: Mapped[str]

Создание таблиц и добавление новой информации

from database import engine, session\_factory, Base  
from models import WorkersOrm  
  
  
def create\_tables():  
 engine.echo = False  
 Base.metadata.drop\_all(engine)  
 Base.metadata.create\_all(engine)  
 engine.echo = True  
  
def insert\_data():  
 worker\_bobr = WorkersOrm(username="Bobr")  
 worker\_volk = WorkersOrm(username="Volk")  
 with session\_factory() as session:  
 session.add\_all([worker\_bobr, worker\_volk])  
 session.commit()

class ResumeOrm(Base):  
 \_\_tablename\_\_ = 'resumes'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 title: Mapped[str]  
 compensation: Mapped[Optional[int]

workload: Mapped[Workload]

Варианты, как можно сделать поле, которое может быть пустым

1) compensation: Mapped[int | None]

2) compensation: Mapped[int] = mapped\_column(nullable=True)

3) from typing import Optional

compensation: Mapped[Optional[int]]

Для создания вариантов используем enum, создаем класс, который наследуется из enum.Enum с вариантами, затем этот класс вставляем в mapped

import enum

class Workload(enum.Enum):  
 parttime = 'parttime'  
 fulltime = 'fulltime'

workload: Mapped[Workload]

Для создания внешнего ключа из sqlalchemy импортируется ForeighnKey, затем в mapped\_column вызывается ForeignKey с названием модели и нужным столбцом, для того, чтобы данные из таблицы удалялись вслед за первичным ключом, то вставляется в ForeignKey атрибут ondelete=’CASCADE’

from sqlalchemy import ForeignKey

worker\_id: Mapped[int] = mapped\_column(ForeignKey('workers.id', ondelete=True, ondelete='CASCADE'))

Для того, чтобы создать поле, в которое будет автоматом подгружаться время создания объекта нужно передать аргументу mapped\_column(server\_default=) два варианта что передавать

from sqlalchemy import text, func

1) created\_at: Mapped[datetime.datetime] = mapped\_column(server\_default=func.now())

func.now(передаёт местное время)

2) created\_at: Mapped[datetime.datetime] = mapped\_column(server\_default=text(“TIMEZONE(‘utc’, now())”))

Для того, чтобы не дублировать информацию, можно создавать общие типы с помощью Annotated, где первым указывается тип данных, а вторым условие, пример:

intpk = Annotated[int, mapped\_column(primary\_key=True)]

id: Mapped[intpk]

Для того, чтобы при изменение модели обновлять время изменений, нужно в mapped\_column, onupdate передать то, что нужно обновлять