SQLAlchemy | Я.Шпора

Начало работы с SQLAlchemy

1. Установить библиотеку:

```
pip install sqlalchemy==1.4
```

2. Создать движок. Движок создаётся один раз при запуске программы при помощи функции `create_engine()`. В эту функцию нужно передать строку с параметрами подключения к БД:

```
dialect+driver://db_username:db_password@db_host:db_port/database_
```

- `dialect` — тип базы данных.
- `driver` — драйвер для соединения с базой данных. Если драйвер не указан — для заданного `dialect` будет выбран драйвер по умолчанию.
- `username` и `password` — имя пользователя и пароль для подключения к базе данных.
- `host` — местоположение сервера базы данных.
- `port` — порт сервера базы данных.
- `database` — имя базы данных.

Если проект работает с SQLite, при создании движка достаточно указать тип базы данных и адрес файла: `'sqlite:///sqlite.db'`.

```
from sqlalchemy import create_engine

if __name__ == '__main__':
    # Создать движок.
    engine = create_engine('sqlite:///sqlite.db')
```

Как описать модели в декларативном стиле в SQLAlchemy

1. Создать родительский класс `Base`, который предоставит функциональность и интерфейс для взаимодействия модели с базой данных.

```
from sqlalchemy.orm import declarative_base

Base = declarative_base()
```

> `declarative_base` — это функция для создания базового класса, который используется для определения моделей в декларативном стиле.

2. Описать модель:
   - Через приватный атрибут `__tablename__` задать имя таблицы, с которым будет связана модель.
   - Описать атрибуты класса, которые будут соответствовать столбцам в таблице БД. Для каждого столбца нужно указать тип данных.

```
from sqlalchemy import Column, Integer, String, create_engine
from sqlalchemy.orm import declarative_base

Base = declarative_base()


class Pep(Base):
    __tablename__ = 'pep'  # Имя таблицы в БД.

    # Свойства модели/колонки таблицы.
    id = Column(Integer, primary_key=True)
    pep_number = Column(Integer, unique=True)
    name = Column(String(200))
    status = Column(String(20))


if __name__ == '__main__':
    # echo=True — выводить в консоль все SQL-запросы,
```

```
    # которые выполняются на этом движке. Параметр можно отключить
    engine = create_engine('sqlite:///sqlite.db', echo=True)
```

3. Описать класс `Base` так, чтобы:
   - имена всех таблиц автоматически создавались из имени модели, но в нижнем регистре;
   - в каждой модели был столбец `id`.

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import declared_attr, declarative_base


class PreBase:

    # Декоратор для динамического определения атрибутов класса мод
    @declared_attr
    def __tablename__(cls):
        # Создать имя таблицы в нижнем регистре из имени модели
        # и вернуть это значение.
        return cls.__name__.lower()

    # Создать колонку id типа Integer.
    id = Column(Integer, primary_key=True)


# Декларативная база включит в себя атрибуты,
# описанные в классе PreBase.
Base = declarative_base(cls=PreBase)


# Наследники класса Base автоматически получат
# приватный атрибут __tablename__ и атрибут id.
class Pep(Base):
    pep_number = Column(Integer, unique=True)
    name = Column(String(200))
    status = Column(String(20))
```

Работа с таблицами в SQLAlchemy

Чтобы создать таблицы, необходимо вызвать метод `Base.metadata.create_all(engine)`:

```
...

if __name__ == '__main__':
    engine = create_engine('sqlite:///sqlite.db', echo=True)
        Base.metadata.create_all(engine)
```

Если добавить или изменить столбец в модели, то вызов `create_all()` не приведёт к изменению таблицы в базе данных.

Чтобы изменить структуру таблицы, можно:

- Создать файлы с миграциями при помощи отдельной библиотеки `alembic`.
- Внести изменения в БД вручную.
- Выполнить «сырой» SQL-запрос для разового изменения таблицы, например, так:

  ```
  ALTER TABLE pep
  ADD COLUMN publication_year INT;
  ```

- Удалить все таблицы:

  ```
  Base.metadata.drop_all(engine)
  ```

- Добавить все таблицы:

  ```
  Base.metadata.create_all(engine)
  ```

Управление данными: CRUD

С записями в базе данных SQLAlchemy взаимодействует через сессию — объект класса `Session`.

Сессия создаётся один раз в начале работы программы:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine('sqlite:///sqlite.db', echo=True)
# Сессия создаётся на основе движка.
session = Session(engine)
```

Любые изменения в базе данных нужно подтверждать командой `session.commit()`.

Чтобы отменить изменения, сделанные в пределах текущей сессии, используется команда `session.rollback()`. Она не требует подтверждения.

Добавить объект

```
...

if __name__ == '__main__':
        ...
        pep8 = Pep(
        pep_number=8,
        name='Style Guide for Python Code',
        status='Active'
    )
    # Добавить в базу объект pep8.
        session.add(pep8)
    # Применить изменения к БД.
        session.commit()
```

Создать объект

```
...
# Импортировать функцию insert().
from sqlalchemy import insert

# Здесь - создание движка, сессии и модели Pep.
...
```

```
session.execute(
    insert(Pep).values(
        pep_number='1000',
        name='Pep from Future',
        status='Proposal'
    )
)
session.commit()
```

Получить данные

```
...
# Импортировать функцию select().
from sqlalchemy import select

...

result = session.execute(
    select(Pep).where(Pep.status == 'Active')
)
print(result.all())
```

Обновить объект

```
...
# Импортировать функцию update().
from sqlalchemy import update

...

session.execute(
    update(Pep).where(Pep.pep_number == 8).values(status='Active')
)
session.commit()
```

Удалить объект

```
...
# Импортировать функцию delete().
from sqlalchemy import delete

...

session.execute(
    delete(Pep).where(Pep.status == 'Active')
)
session.commit()
```

Полезные ресурсы

- О приватных атрибутах в документации Python
- О типах данных в документации SQLAlchemy
- О расширении класса декларативной базы в документации SQL
- О сессиях в документации SQLAlchemy

Практикум