SQLAlchemy - ORM
ORM - *Object-Relational Mapping* (Объектно-Реляционное представление). Технология связывающая реляционное представление информации в РСУБД с концепциями ООП

В алхимии, как и в любой другой ORM, все таблицы описываются на основании классов и называются моделями.
Каждая модель должна наследоваться от базовой модели, для ее создании в алхимии есть класс **DeclarativeBase** (в алхимии версии 2+, до 2 версии использовалась функция declarative_base возвращающая базовую модель)

```
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass
```

Это пример самой простой базовой модели, все таблицы мы будем наследовать от класса **Base**

```
from sqlalchemy import Column, INT, VARCHAR

class Category(Base):
    # магический атрибут указывающий на название таблицы в СУБД
    __tablename__ = 'category'

    # Первичный ключ таблицы типа integer
    # id INTEGER PRUMARY KEY
    id = Column(INT, primary_key=True)

    # Колонка name, строчная с ограничением 64 символа, не пустая, уникальная
    # name VARCHAR (64) NOT NULL UNIQUE
    name = Column(VARCHAR(64), nullable=False, unique=True)
```

Основные атрибуты класса **Column**

1. **name** - позиционный не обязательный атрибут, указывает на название атрибута в СУБД, если он не передан, то атрибут в СУБД будет назван аналогично атрибуту в модели
2. **type** - позиционный атрибут указывающий на его тип, для указания типов используются собственные типы алхимии импортируемые напрямую из sqlalchemy
3. **autoincrement**: bool - атрибут присущий только sqlite базе данных, используется для указания автоматического инкремента целочисленного атрибута (как правило для первичных ключей)
4. **default** - значение по умолчанию выдаваемое самой алхимией (не влияет на DDL) может являться как обычным значением так и может возвращаться какой-либо функцией
5. **doc**: str - необязательный строчный атрибут для описания атрибута модели, не влияет на структуру БД, при его указании, у атрибута модели появляется атрибут comment возвращающий значение атрибута doc
6. **index**: bool - опциональный атрибут указывающий на создание индекса БД (CREATE INDEX) используется как правило с уникальными атрибутами таблицы для быстрого поиска данных по таблицы
7. **nullable**: bool - указывает может ли быть атрибут таблицы пустым (False - не может, True - может), влияет на DDL (NOT NULL)
8. **primary_key**: bool - указывает на то, будет ли данный атрибут модели являться первичным ключом, влияет на DDL (PRIMARY KEY)
9. **server_default**: str - значение по умолчанию, указывается в виде строки sql, влияет на DDL (DEFAULT), не рекомендуется использовать default и server_default одновременно
10. **unique**: bool - указывает на уникальность значений в данном атрибуте таблицы, влияет на DDL (UNIQUE)

11. **comment**: str - аналогичен doc, но уже влияет на DDL (добавляет комментарий в таблицу)

Для описания внешнего ключа используется класс **ForeignKey**

category_id = Column(INT, ForeignKey('category.id', ondelete='CASCADE), nullable=False)

Основные атрибуты **ForeignKey**

1. **column** - колонка на которую ссылается внешний ключ, может быть указан несколькими способами (Category.id или 'category.id') второй способ более удобный так как нам не обязательно описывать таблицу, на которую мы ссылаемся, в этом же файле выше или импортировать в этот же файл, позволяет сослаться на таблицу которая описана в другом месте или после текущей модели
2. **name**: str - аналогичен name в классе Column
3. **ondelete** и **onupdate** - ссылочная спецификация (поведения связаной записи при удалении/обновлении главной записи), указывается в виде строк, влияет на структуру БД

```
class Post(Base):
        __tablename__ = 'post'

         id = Column(INT, primary_key=True)
        title = Column(VARCHAR(128) nullable=False)
        body = Column(TEXT, nullable=False)
        category_id = Column(INT, ForeignKey('category.id', ondelete='CASCADE'),
nullable=False)
```

Для удобства можно выносить одинаковые атрибуты моделей в базовый класс, например id, таким образом его не надо будет описывать в каждой модели (наследованные), кроме ситуаций где необходимо сменить его тип данных

Так же можно описывать генератор атрибута __tablename__ на основании имени класса, для этого в базовой модели необходимо описать метод класса, обернутый в декоратор declared_attr импортируемый из sqlalchemy.orm

Пример генератора имени таблицы на основании имени модели:

```
@declared_attr
def __tablename__(cls):
        return ''.join(f'_{i.lower()} if i.isupper() else i for i in cls.__name__).strip('_')
```

Данный декоратор позволяет генерировать любые атрибуты класса при необходимости

Для удобства дальнейшего использования, можно описывать отношения (атрибуты не влияющие на структуру БД, но позволяющие обращаться к связанным записям через дополнительный атрибут модели)

```
from sqlalchemy.orm import relationship

class Category(Base):
        ...
        posts = relationship('Post', back_populates='category')
class Post(Base):
        ...
        category = relationship(Category, back_populates='posts')
```

Отношения это своего рода встроенный JOIN запрос в атрибут модели,
Category.posts вернет все посты выбранной категории в виде списка
Post.category вернет категорию поста