

Модели

Создание моделей и миграции базы данных

Последнее обновление: 25.02.2018



Модели в Django описывают структуру используемых данных. Используемые в программе данные хранятся в базах данных, и с помощью моделей как раз осуществляется взаимодействие с базой данных.

По умолчанию Django в качестве базы данных использует SQLite. Она очень проста в использовании и не требует запущенного сервера. Все файлы базы данных могут легко переноситься с одного компьютера на другой. Однако при необходимости мы можем использовать в Django большинство распространенных СУБД.

Для работы с базами данных в проекте Django в файле **settings.py** определен параметр **DATABASES**, который по умолчанию выглядит следующим образом:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Конфигурация используемой базы данных в данном случае складывается из двух параметров. Параметр **ENGINE** указывает на используемый движок для доступа к БД. В данном случае это встроенный пакет `django.db.backends.sqlite3`. Второй параметр - **NAME** указывает на путь к базе данных. После первого запуска проекта в нем по умолчанию будет создан файл `db.sqlite3`, который собственно и будет использоваться в качестве базы данных.

Чтобы использовать другие системы управления базами данных, необходимо будет установить соответствующий пакет.

| СУБД | Пакет | Команда установки |
|------|-------|-------------------|
|      |       |                   |

|            |              |                          |
|------------|--------------|--------------------------|
| PostgreSQL | psycopg2     | pip install psycopg2     |
| MySQL      | mysql-python | pip install mysql-python |
| Oracle     | cx_Oracle    | pip install cx_Oracle    |

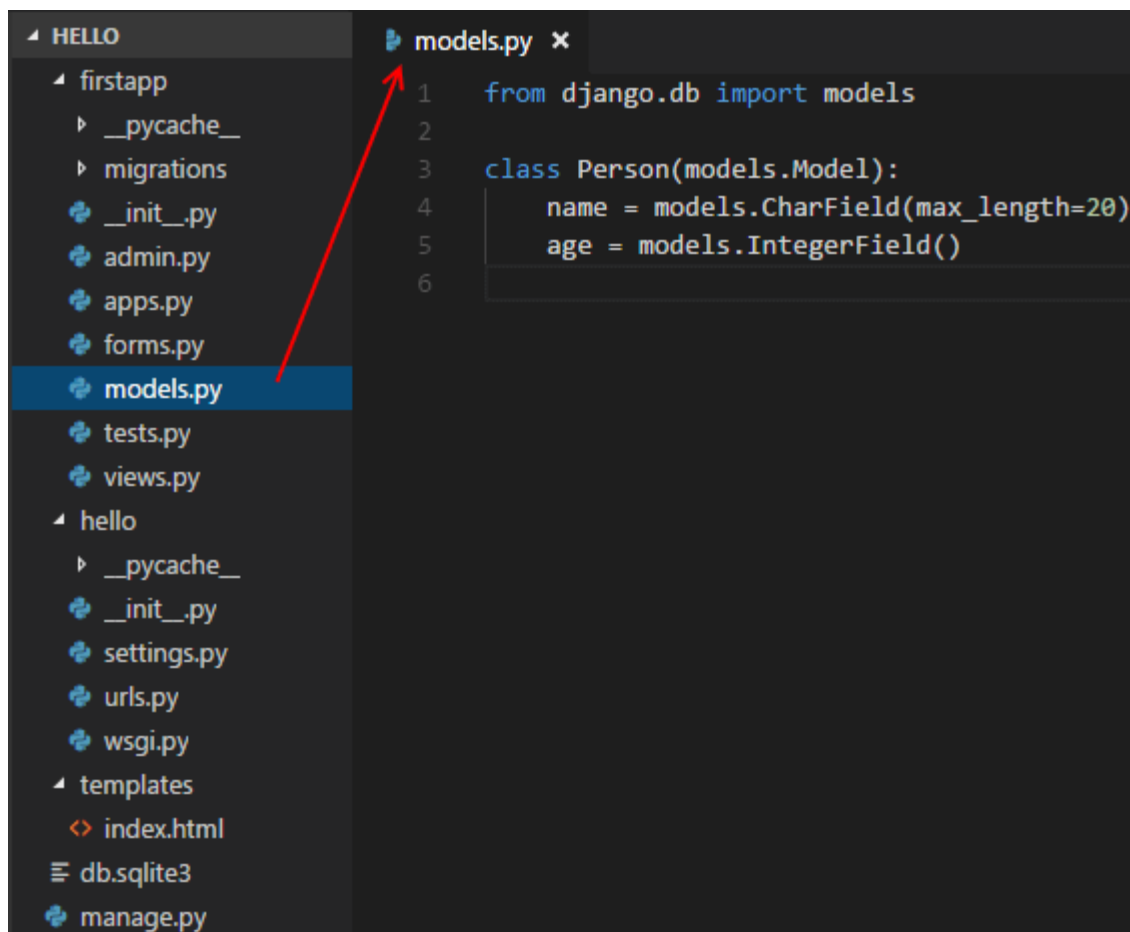
## Создание моделей

При создании приложения по умолчанию в его каталог добавляется файл **models.py**, который применяется для определения моделей. Модель представляет класс, унаследованный от **django.db.models.Model**.

Так, изменим файл **models.py** следующим образом:

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=20)
    age = models.IntegerField()
```



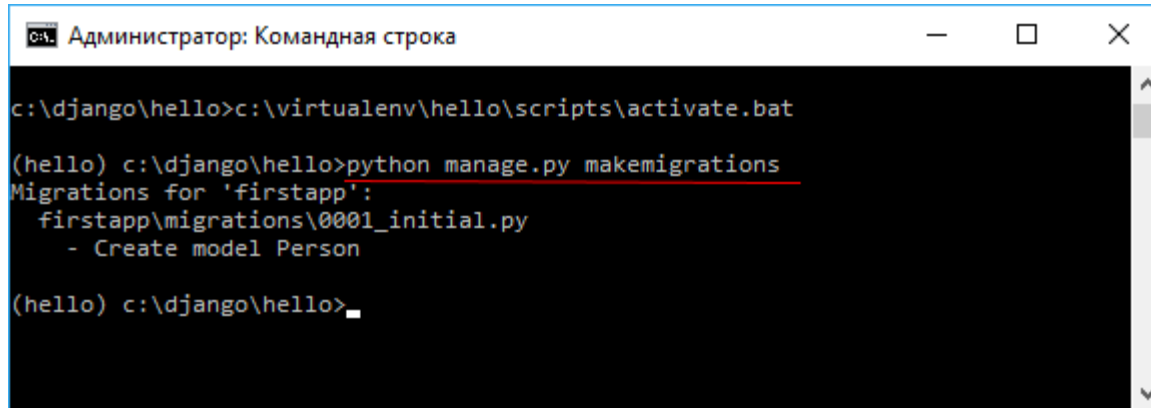
Здесь определена простейшая модель, которая называется Person и которая представляет человека. В модели определены два поля. Поле name представляет тип CharField - текстовое поле, которое хранит последовательность символов. Оно будет хранить имя человека. Для CharField обязательно надо указать параметр max\_length, который задает максимальную длину хранящейся строки. И поле age представляет тип IntegerField - числовое поле, которое хранит целые числа. Оно предназначено для хранения возраста

человека.

Каждая модель сопоставляется с определенной таблицей в базе данных. Однако пока у нас нет в бд таблицы, которая хранит объекты модели Person. И в этом случае нам надо создать и выполнить миграцию. Миграция преобразует базу данных в соответствии с определением моделей.

Вначале необходимо создать миграцию с помощью команды

```
python manage.py makemigrations
```



```

c:\django\hello>c:\virtualenv\hello\scripts\activate.bat

(hello) c:\django\hello>python manage.py makemigrations
Migrations for 'firstapp':
  firstapp\migrations\0001_initial.py
    - Create model Person

(hello) c:\django\hello>_

```

После этого в приложении в папке **migrations** мы обнаружим новый файл, который будет иметь примерно следующее содержимое:

```

from django.db import migrations, models

class Migration(migrations.Migration):

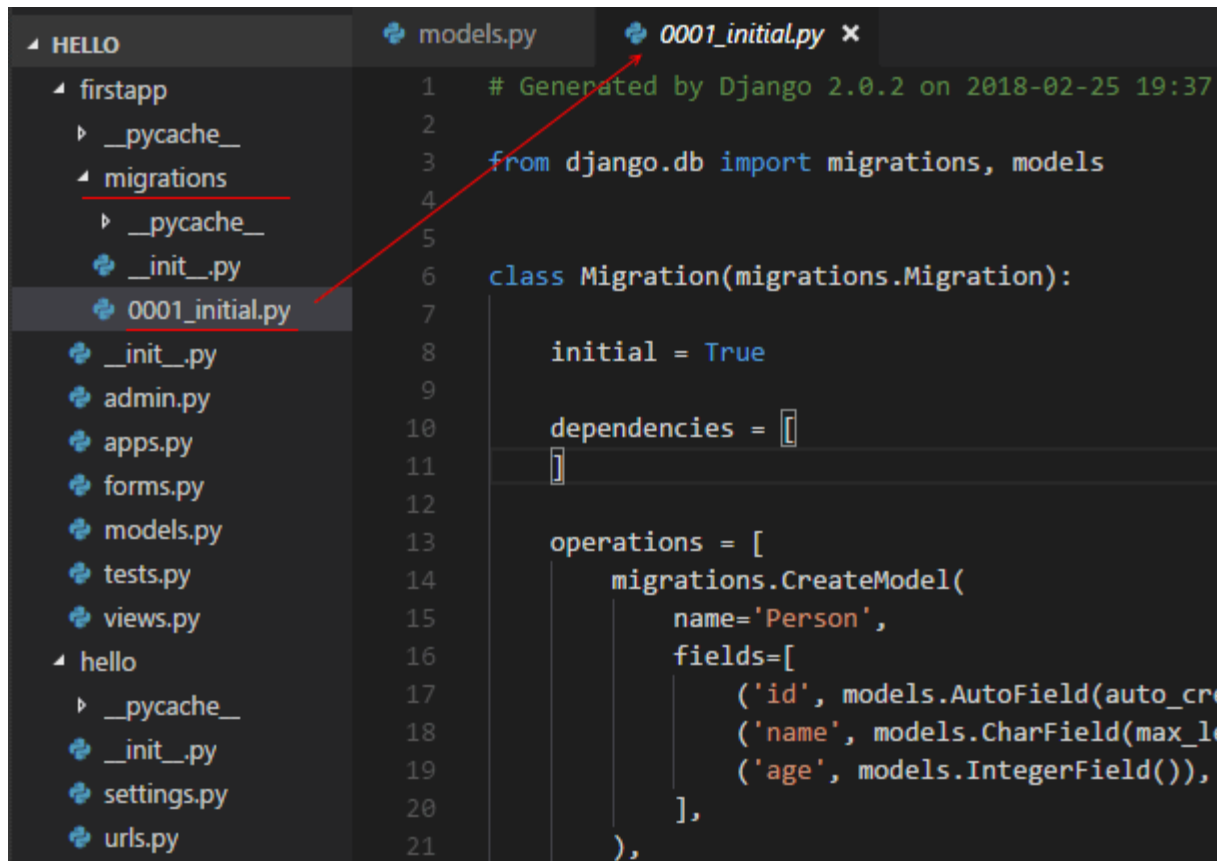
    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Person',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=20)),
                ('age', models.IntegerField()),
            ],
        ),
    ]

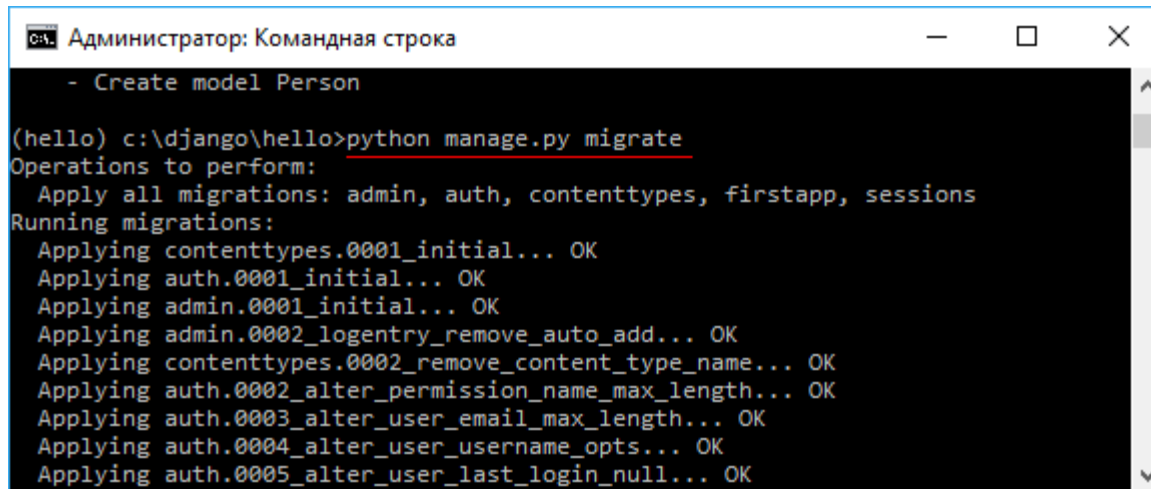
```

Это и есть миграция. Здесь можно заметить, что создается не два, а три поля - поле id, которое будет представлять первичный ключ, добавляется по умолчанию. Поэтому в принципе в самой модели нам не нужно явным образом определять какой-либо идентификатор.

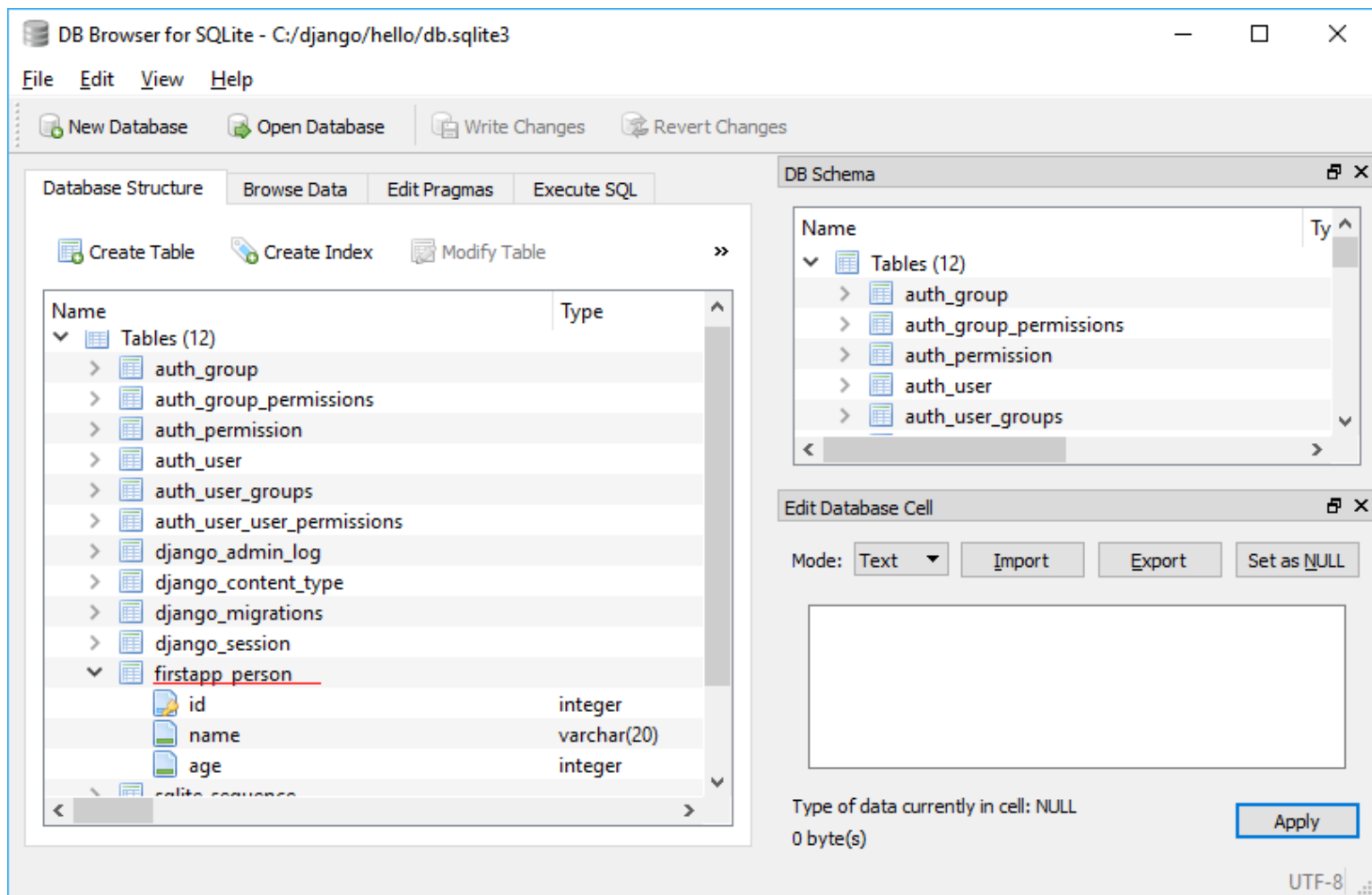


Теперь надо выполнить данную миграцию. Для этого выполняется команда

```
python manage.py migrate
```



После этого, если мы откроем базу данных **db.sqlite3**, которая есть в проекте, в какой-нибудь специальной программе для просмотра БД SQLite, то мы увидим, что она содержит ряд таблиц:



В основном это будут служебные таблицы. А нас прежде всего будет интересовать таблица, которая называется по имени приложения и модели. В моем случае это таблица `firstapp_person`, которая и будет хранить данные модели `Person`.

[Назад](#) [Содержание](#) [Вперед](#)



### **Помощь сайту**

#### **WebMoney**

- R378451176208
- Z280152397659
- U210796482817

#### **Yandex-деньги**

- 410011174743222

#### **PayPal**

- metanit22@mail.ru

---

[Вконтакте](#) | [Twitter](#) | [Канал сайта на youtube](#) | [Помощь сайту](#)

Контакты для связи: metanit22@mail.ru

Copyright © metanit.com, 2012-2019. Все права защищены.