**Урок 3. Работа с моделями (Django 2)**

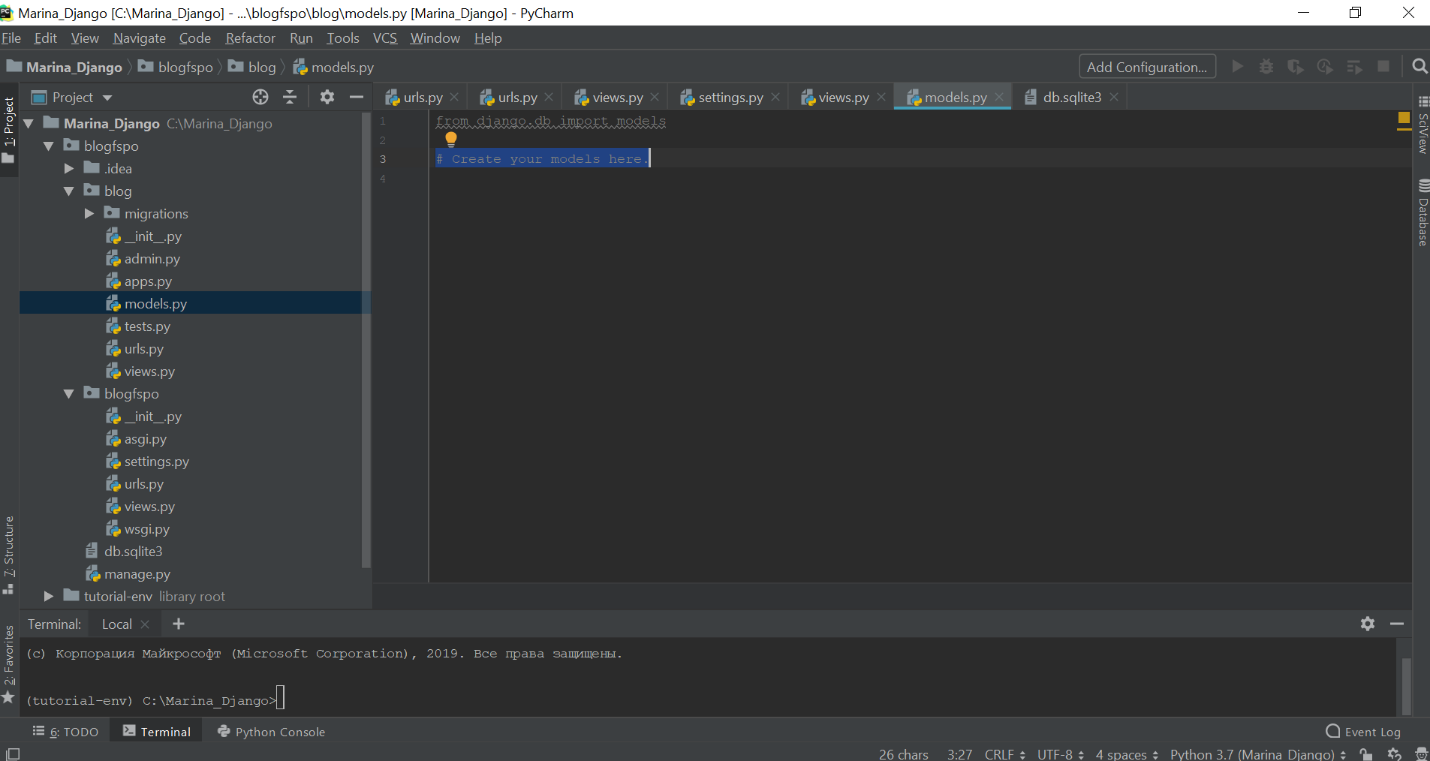
**Технология выполнения работы:**

1. **Модели** отображают информацию о данных, с которыми работает пользователь. Они содержат поля и поведение данных. Обычно одна модель представляет одну таблицу в базе данных.

Каждая модель это класс, унаследованный от [django.db.models.Model](https://djbook.ru/rel1.9/ref/models/instances.html" \l "django.db.models.Model" \o "django.db.models.Model).

Атрибут модели представляет собой поле в базе данных.

1. Для описания моделей необходимо открыть файл models.py приложения (blogfspo) и создать все модели.



Для примера рассматривается база данных, описывающая участие собак в выставках. Для собаки, участника выставки, известен владелец и данные для регистрации на выставке:



1. Модель, определяющая владельца собаки, описывается следующим образом:

class Owner(models.Model):  
 first\_name = models.CharField(max\_length=50)  
 last\_name = models.CharField(max\_length=50)

first\_name и last\_name – поля модели. Каждое поле определено как атрибут класса, и каждый атрибут соответствует полю таблицы в базе данных.

Каждое поле в модели должно быть экземпляром соответствующего [Field](https://docs.djangoproject.com/en/3.0/ref/models/fields/#django.db.models.Field)класса. (URL: <https://docs.djangoproject.com/en/3.0/topics/db/models/>, Типы полей; <https://docs.djangoproject.com/en/3.0/ref/models/fields/#model-field-types>)

Модель Owner создаст в базе данных таблицу (используется синтаксис PostgreSQL):

CREATE TABLE myapp\_owner(

"id" serial NOT NULL PRIMARY KEY,

"first\_name" varchar(30) NOT NULL,

"last\_name" varchar(30) NOT NULL

);

Название таблицы myapp\_owner автоматически создается с метаданных модели и может быть переопределено. (URL: <https://docs.djangoproject.com/en/3.0/ref/models/options/#table-names>, Название таблицы).

Поле первичного ключа id создается автоматически, но его также можно переопределить. (URL: <https://docs.djangoproject.com/en/3.0/topics/db/models/>, Автоматические поля первичного ключа).

Django использует синтаксис SQL соответственно настройкам базы данных в файле настроек. (URL: <https://docs.djangoproject.com/en/3.0/topics/settings/>, Настройки Django).

1. Модель для описания собаки следующая:

class Dog(models.Model):  
 owner\_dog =

models.ForeignKey(Owner,on\_delete=models.CASCADE)

name = models.CharField(max\_length=30)  
 age = models.IntegerField()

class\_dog = models.CharField(max\_length=30)

Для создания отношения 1:М между моделями Owner и Dog используется определение внешнего ключа [django.db.models.ForeignKey](https://docs.djangoproject.com/en/3.0/ref/models/fields/#django.db.models.ForeignKey). Оно используется так же, как и другие типы полей Field, и добавляется как атрибут в модель:

owner\_dog = models.ForeignKey(Owner,on\_delete=models.CASCADE)

Для [ForeignKey](https://djbook.ru/rel1.9/ref/models/fields.html" \l "django.db.models.ForeignKey" \o "django.db.models.ForeignKey) указывается обязательный позиционный аргумент: класс, к которому относится модель.

(URL: <https://docs.djangoproject.com/en/3.0/topics/db/models/>, Отношения).

1. После определения моделей необходимо указать Django, что необходимо их использовать.

Сначала необходимо проверить, чтобы в файле настроек проекта settings.py в [INSTALLED\_APPS](https://djbook.ru/rel1.9/ref/settings.html#std:setting-INSTALLED_APPS) было указано приложение, содержащее models.py.

Затем нужно создать миграции, выполнив [manage.py makemigrations](https://djbook.ru/rel1.9/ref/django-admin.html#django-admin-makemigrations):

(tutorial-env) C:\Marina\_Django\blogfspo>py manage.py makemigrations

Ответ:

Migrations for 'blog':

blog\migrations\0001\_initial.py

- Create model Owner

- Create model Dog

Далее выполнить команду manage.py migrate:

(tutorial-env) C:\Marina\_Django\blogfspo>py manage.py migrate

Ответ:

Operations to perform:

Apply all migrations: admin, auth, blog, contenttypes, sessions

Running migrations:

Applying blog.0001\_initial... OK

1. Модель для описания выставки следующая:

class Exhibition(models.Model):  
 TYPE\_EX = (  
 ('C1', 'CACIB'),  
 ('C2', 'CAC'),  
 ('M', 'MONO'),  
 )  
 name\_ex = models.CharField(max\_length=100)  
 date\_b = models.DateField()  
 date\_c = models.DateField()  
 type = models.CharField(max\_length=2, choices=TYPE\_EX)

Для описания поля type используется аргумент choices. Это итератор (например, список или кортеж) 2-х элементных кортежей, которые определяют варианты значений для этого поля. При определении виджет формы использует select вместо стандартного текстового поля и ограничит значение поля указанными значениями.

Первый элемент в кортеже – значение хранимое в базе данных, второй элемент отображается виджетом формы, или в ModelChoiceField.

Можно также использовать перечисляющие классы для choices-краткого определения. Тогда для модели Выставка можно использовать следующее определение:

class Exhibition(models.Model):

TYPE\_EX = models.TextChoices('CACIB', 'CAC', 'MONO')

name\_ex = models.CharField(max\_length=100)  
 date\_b = models.DateField()  
 date\_c = models.DateField()  
 type = models.CharField(blank=True,

choices= TYPE\_EX.choices, max\_length=4)

)

(URL: <https://docs.djangoproject.com/en/3.0/topics/db/models/>, Поля; <https://docs.djangoproject.com/en/3.0/ref/models/fields/#field-choices>).

1. Выполнить миграции.
2. В соответствии с моделью БД между таблицами Собака и Выставка существует отношение М:М, обладающее свойством дата\_регистрации (date\_reg). Для его определения используется [ManyToManyField](https://docs.djangoproject.com/en/3.0/ref/models/fields/#django.db.models.ManyToManyField). Чтобы его использовать, необходимо включить его в качестве атрибута класса модели.

Для [ManyToManyField](https://docs.djangoproject.com/en/3.0/ref/models/fields/" \l "django.db.models.ManyToManyField" \o "django.db.models.ManyToManyField) указывается обязательный позиционный аргумент: класс, к которому относится модель.

Для определения отношения M:M между таблицами Собака и Выставка добавить поле [ManyToManyField](https://docs.djangoproject.com/en/3.0/ref/models/fields/#django.db.models.ManyToManyField) в таблицу Выставка и создать промежуточную модель Участие (Participate):

class Exhibition(models.Model):  
 TYPE\_EX = (  
 ('C1', 'CACIB'),  
 ('C2', 'CAC'),  
 ('M', 'MONO'),  
 )  
 name\_ex = models.CharField(max\_length=100)  
 date\_b = models.DateField()  
 date\_c = models.DateField()  
 type = models.CharField(max\_length=2, choices=TYPE\_EX)

member = models.ManyToManyField(Dog, through='Participate')

class Participate(models.Model):

dog = models.ForeignKey(Dog, on\_delete=models.CASCADE)

exhibition = models.ForeignKey(Exhibition, on\_delete=models.CASCADE)

date\_reg = models.DateField()

(URL: <https://docs.djangoproject.com/en/3.0/topics/db/models/>, Отношения Многие-ко-Многим; <https://docs.djangoproject.com/en/3.0/topics/db/examples/many_to_many/>).

1. Выполнить миграции.

**Практическое задание:**

1. Создайте модели для БД по заданной предметной области в IDE PyCharm:
2. опишите поля моделей;
3. опишите отношения один-ко-многим и многие-ко-многим между моделями в соответствии со схемой БД.
4. Сохраните проект.