Панель администратора. Базы данных

Панель администратора

Создание админского пользователя через терминал:

```
python manage.py createsuperuser
```

Далее следуем инструкциям в терминале.

По окончании создания пользователя можно зайти в панель администратора:

http://127.0.0.1:8000/admin/

По умолчанию в панели видны всего две таблицы базы данных сайта: пользователи и группы.

Миграция

Миграция - это способ Django распространять изменения, которые вы вносите в свои модели, в схему вашей базы данных.

Когда выполняется миграция?

ПРИ ЛЮБОМ ИЗМЕНЕНИИ В БАЗЕ!

Миграция – система контроля версий для вашей схемы базы данных

Пример работы с миграциями

Была таблица с одним полем

models.py

```
from django.db
        import models

class New(models.Model):
    text =
       models.TextField()
```

Добавили ещё одно

models.py

```
from django.db
        import models

class New(models.Model):
  text =
    models.TextField()
  num =
    models.IntegerField()
```

Помненять models.py недостаточно! После изменения файла models.py в терминале обязательно надо выполнить две команды:

```
python manage.py makemigrations

python manage.py migrate
# а ещё зарегистрировать таблицу в
админ.панели (см. след. слайд)
```

Алгоритм добавления таблицы

1. В выбранном приложении открыть файл models.py
2. В нём создать класс, являющийся моделью таблицы. Поля класса = поля таблицы
3. python manage.py makemigrations
4. python manage.py migrate
5. Зарегистрировать таблицу в админке. Открываем admin.py в соответствующем приложении и пишем:

```
from django.contrib import admin
from .models import New

admin.site.register(New)  # New – имя класса таблицы.
```

6. Теперь можно проверить наличие этой таблицы в админ.панели сайта.

Команды терминала

```
migrate                 # применение и отмена миграции.
makemigrations          # создание новых миграций на основе изменений, которые вы внесли в свои модели.
sqlmigrate  #отображает операторы SQL для миграции.
showmigrations          # отображает миграции проекта и их статус.
```

Например:

```
python manage.py makemigrations
```

Виды полей моделей

BooleanField(): хранит значение True или False
(0 или 1)

DateField(): хранит дату

TimeField(): хранит время

DateTimeField(): хранит дату и время

AutoField(): хранит целочисленное значение, которое автоматически инкрементируется, обычно применяется для первичных ключей

FloatField(): хранит, значение типа Number, которое представляет число с плавающей точкой

IntegerField(): хранит значение типа Number, которое представляет целочисленное значение

CharField(max_length=N): хранит строку длиной не более N символов

TextField(): хранит строку неопределенной длины

EmailField(): хранит строку, которая представляет email-адрес. Значение автоматически валидируется встроенным валидатором EmailValidator

FileField(): хранит строку, которая представляет имя файла

URLField(): хранит строку, которая представляет валидный URL-адрес

Пример создания класса таблицы

```
from django.db import models
class Articles(models.Model):
    title = models.CharField('Название', max_length=50)
# 1 поле - текстовое, одна строка 255 символов. В аргументе - подпись
    anons = models.CharField('Анонс', max_length=250)
# эти названия будут отображаться в админке
    full_text = models.TextField('Статья')
 # textField - нормальный полноценный текст
    date = models.DateTimeField('Дата публикации')
# DateTime - дата и время, Date - только дата
    def __str__(self):
        return self.title
```

Manager

Класс Manager – это интерфейс для запросов к базе данных. По умолчанию Django добавляет экземпляр Manager с именем **objects** в каждый класс модели Django. Поэтому мы можем использовать методы этого класса для получения данных.

Методы класса Manager

Метод	Пример	Суть
all()	Blog.objects.all()	Возвращает все объекты класса в базе данных
count()	Blog.objects.count()	Возвращает количество записей в QuerySet
filter()	Blog.objects.filter(year = 2022)	Возвращает все объекты класса в базе данных, соответствующие значению аргумента
exists	Blog.objects.filter(year = 2022).exists()	возвращает True, если записи за 2022 год есть, иначе False

Вывод данных из БД

request

urls.py

```
urlpatterns = [
…
path('newPage/',
include('newPage.urls')),
]
```

newPage.urls.py

```
urlpatterns = [path('',
views.new_home,
name='new_home')]
```

БАЗА ДАННЫХ с таблицей **New**

response

views.py

```
from .models import New
def newOpen(request):
    newPage = New.objects.all()
    return render(request,
'newPage/new_home.html',
{'new': newPage })
```

models.py

```
from django.db import models

class New(models.Model):
    text =
        models.TextField()
```

Ключ

Объект. Теперь в new_home есть {{newobject}}

QuerySet

Слова **query** и **request**
могут быть переведены как «запрос».
Различие:
query – вопрос (предоставьте, пожалуйста, сведения);
request – запрос (предоставьте сведения в данном формате).

QuerySet – набор запросов.

Это итерируемая коллекция.

Коллекцию данного типа возвращает метод **`objects.all()`.**

В примере на предыдущем слайде **newobject** – как раз такая коллекция.

Так как она итерируемая, доступен перебор её элементов в цикле:

```
{% for elem in newobject %}
<p>{{elem}}</p>
{% endfor %}
```

Полезно проверить существование объекта, прежде чем пытаться из него что-то вывести:

```
{% if newobject %}
…манипуляции с объектом…
{% endif %}
```

Запросы на выборку из БД

На предыдущих слайдах мы получили информацию из базы данных в виде QuerySet, не используя SQL. Отсюда вывод:

В терминах SQL
QuerySet приравнивается к оператору SELECT,
а фильтр является ограничивающим предложением,
таким как WHERE

Что почитать

Поля моделей – полный список:
https://metanit.com/python/django/5.2.php

Документация по миграциям:
https://django.fun/ru/docs/django/4.1/topics/migrations/

Документация по запросам:
https://django.fun/ru/docs/django/4.1/topics/db/queries/