МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

Урок №5

Продолжение работы с ORM

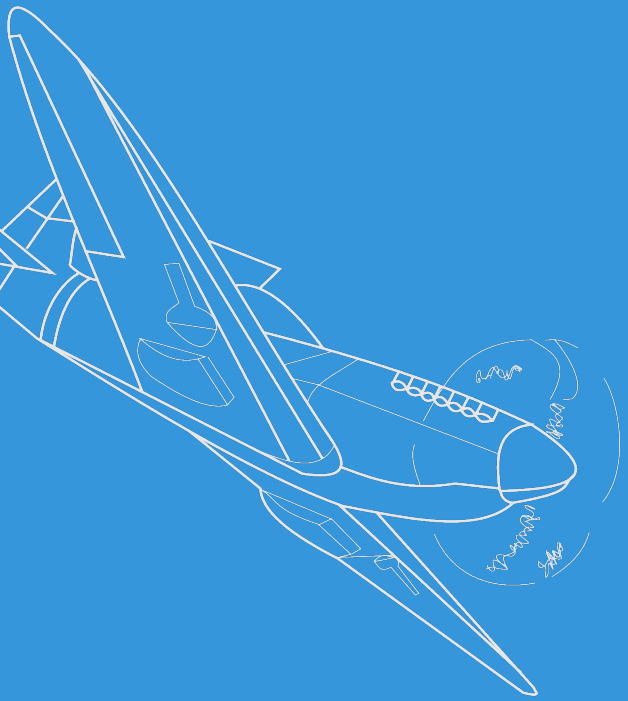
на котором расскажут ещё немного про миграции, админку Джанго, методы модели, ModelManager.

3 апреля 2023 года

Антон Кухтичев

Содержание занятия

1. Технический долг;
2. Ещё немного про миграции;
3. Админка Джанго;
4. Методы модели;
5. Продолжаем ORM;
6. ModelManager.



Технический долг

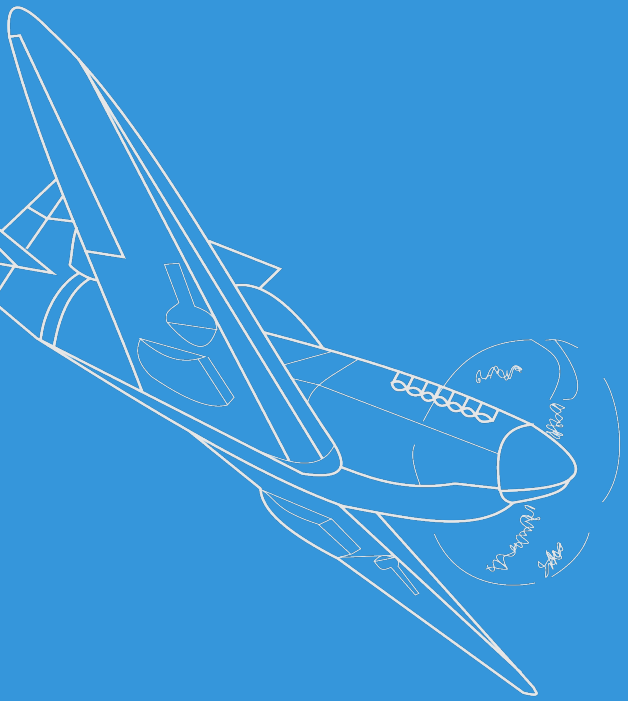
Что надо сделать, если не хотите, чтобы создавалось у модели поле id, и создать своё поле.

P.S. Когда-нибудь я перемещу этот слайд в прошлую лекцию.

Как сделать своё поле с первичным ключём

Если по каким-либо причинам Вы не хотите, чтобы создавалось поле id, то надо создать своё с `primary_key=True`!

```
class Blog(models.Model):
    my_super_pk = models.AutoField(primary_key=True)
    ...
```



Ещё немного про миграции

Откат до предыдущей миграции, создание пустой и программирование своей логики миграции.

Откат до предыдущей миграции


```
# В общем случае команда выглядит так:
$ ./manage.py migrate <ваше приложение> <название предыдущей миграции>
# Откат всех миграций!
$ ./manage.py migrate <ваше приложение> zero
# Примерчик:
$ ./manage.py showmigrations movies
movies
 [X] 0001_initial
 [X] 0002_movie_genre
 [X] 0003_auto_20201005_1456
$ ./manage.py migrate movies 0002_movie_genre
```

Создание пустой миграции


```
$ ./manage.py makemigrations --empty <ваше приложение>
```

```
from django.db import migrations


class Migration(migrations.Migration):
    dependencies = [
        ('yourappname', '0001_initial'),
]
    operations = [
        migrations.RunPython(forwards_func, reverse_func),
]
```

Еще немного про миграции

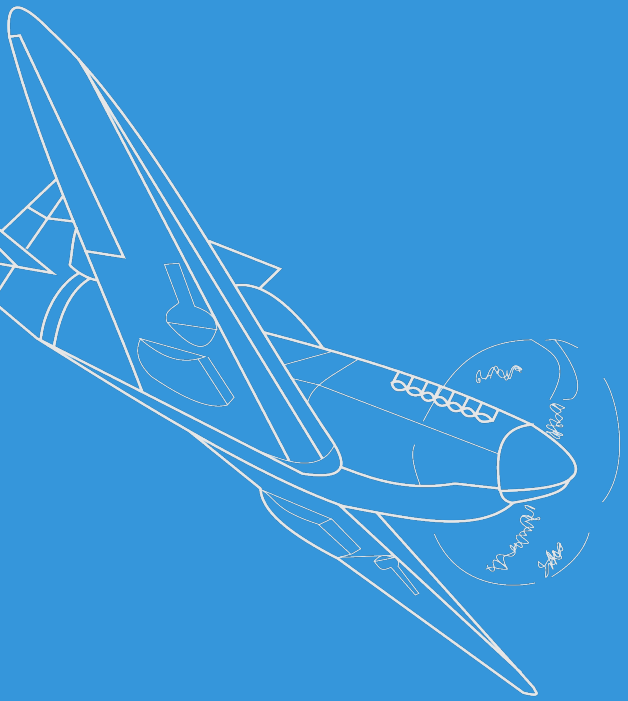

```
def forwards_func(apps, schema_editor):
    SomeModel = apps.get_model('some_app', 'SomeModel')
    SomeModel.objects.all().update(field=True)

def reverse_func(appsm schema_editor):
    SomeModel = apps.get_model('some_app', 'SomeModel')
    SomeModel.objects.all().update(field=None)
```

Ещё полезная команда sqlmigrate


```
$ ./manage.py sqlmigrate movies 0002

BEGIN;
--
-- Rename field category_id on movie to category
--
ALTER TABLE "movies_movie" RENAME COLUMN "category_id_id" TO
"category_id";
--
-- Add field year to movie
--
ALTER TABLE "movies_movie" ADD COLUMN "year" integer NULL;
--
-- Alter field added_at on movie
--
COMMIT;
```



Админка Джанго

Админка Джанго

Надо создать для начала суперпользователя!

```
./manage.py createsuperuser
```

После этого надо запустить приложение и зайти на `http://127.0.0.1:8000/admin/`

Параметры у поля в модели

`verbose_name` — человекочитаемое название поля;

`null` — может ли быть поле NULL в БД, по умолчанию - False

`blank` — может ли быть поле пустым в форме, по умолчанию False

`choices` — выбор значения из списка;

`default` — значение по умолчанию;

`db_index` — нужно ли создать индекс в БД;

`help_text` — подробное описание поля;

`primary_key` — первичный ключ;

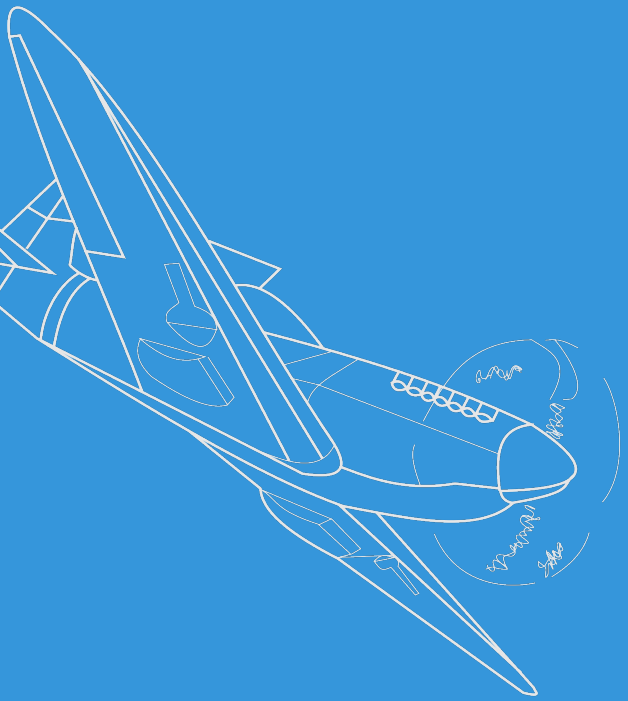
`unique` — уникальное ли поле;

`editable` — можно ли поле изменять;

Админка Джанго


```
# blog.admin.py
from django.contrib import admin
from blogs.models import Blog
class BlogAdmin(admin.ModelAdmin):
    list_display = ('id', 'title')
    list_filter = ('is_active',)
admin.site.register(Blog, BlogAdmin)

@admin.register(Blog)
class BlogAdmin(admin.ModelAdmin):
    pass
# https://docs.djangoproject.com/en/3.2/ref/contrib/admin/
```



Методы модели

Методы модели (1)

`def __str__(self) -> str` — магический метод, строковое представление любого объекта

`def get_absolute_url(self) -> str` — как сделать URL для объекта;

`def save(self, *args, **kwargs)` — как сохранить объект;

```
class Meta:
    verbose_name = “Название модели”
    verbose_name_plural = “Название модели во множественном числе”
    abstract = True
    ordering = ['name']
```

Методы модели (2)


```
def get_absolute_url(self):
    from django.urls import reverse
    return reverse('people.views.details', args=[str(self.id)])

def get_absolute_url(self):
    return f"/people/{self.id}/"

def __str__(self):
    return f'{self.first_name} {self.last_name}'
```

Методы модели (3)

```
def save(self, *args, **kwargs):
    if self.image:
        small = rescale_image(self.image,width=100,height=100)
        self.image_small = SimpleUploadedFile(name,small_pic)
    super(Model, self).save(*args, **kwargs)
```

Расширение AbstractUser (1)

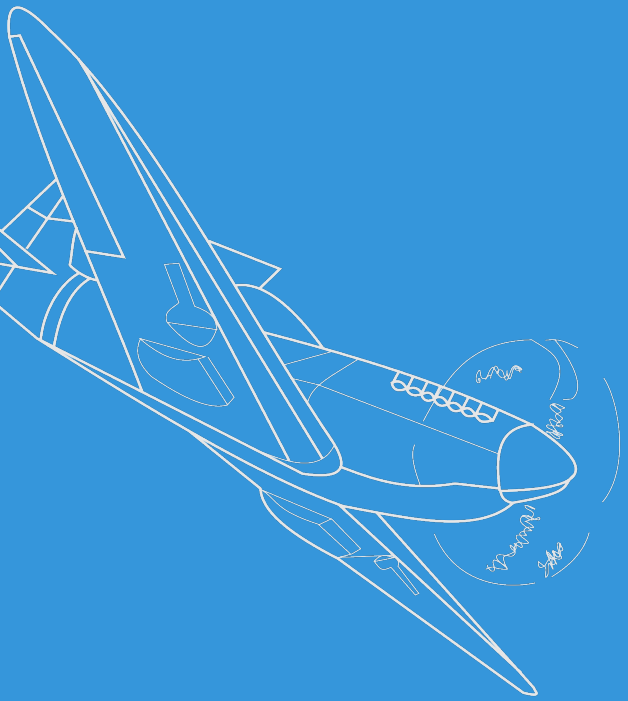
По умолчанию модель `User` в Django использует username для уникальной идентификации во время аутентификации. Можно создать собственную пользовательскую модель `User`, используя для этого подклассы `AbstractUser` или `AbstractBaseUser`.

- `AbstractUser`: Используйте этот подкласс, если вас устраивают существующими поля в модели `User` и Вы просто хотите удалить поле username.
- `AbstractBaseUser`: Используйте этот подкласс, если Вы хотите создать с нуля собственную, совершенно новую модель `User`.

Расширение AbstractUser (2)


```
from django.db import models
from django.contrib.auth.models import AbstractUser


class User(AbstractUser):
    username = None
    bio = models.TextField('Биография', max_length=500, blank=True)
    location = models.CharField('Город', max_length=30, blank=True)
    birthday = models.DateField('Дата рождения', null=True,
blank=True)
# в settings.py добавить AUTH_USER_MODEL = "users.YourUserModel"
```



Продолжаем ORM

Продолжаем ORM


```
# Создание объекта без связей
post = Post()
post.title = ‘Test’
post.save()
# Второй способ
post2 =
    Post.objects.create(title=’Test2’)
# Создание объекта со связями
post3 =
    Post.objects.create(title=’Test3’)
```

```
# Руками проставляем id категории.
post3.category_id = 2
post3.save()
# А можем присвоить объект внутри create
category =
    Category.objects.create(title=’Test
category’)
post4 = Post.objects.create(title=’Test
4’, category=category)
```

Связи ManyToMany


```
post = Post.objects.create(title='Test2')
tag = Tag.objects.create(slug='some_tag')
post.tags.add(tag)
```

Загрузка объектов из БД


```
# По ключу:
try:
    post = Post.objects.get(id=5)
except Post.DoesNotExist:
    post = None
# По другому полю:
try:
    post = Post.objects.get(title='Python')
except MultipleObjectsReturned:
    post = None
```

Выборка нескольких объектов из БД (1)

```
all_posts = Post.objects.all()
first_three = Post.objects.all()[:3]

some_category = Categoty.objects.get(id=1)

category_posts = Post.objects.filter(category=some_category)
category_posts = Post.objects.filter(category_id=1)
```

Выборка нескольких объектов из БД (2)


```
css_posts = Post.objects.filter(title__contains='css')
css_posts = css_posts.order_by('-rating')
css_posts = css_posts[10:20]

# Выборка с условием с ИЛИ
from django.db.models import Q

Post.objects.filter(Q(title__contains='css') |
                    Q(title__contains='html'))
```

QuerySet

- `QuerySet` — объекты, представляющие собой запрос к базе данных (не результаты).
- Именно запрос, а не его результаты.
- `QuerySet` — ленивые (lazy) объекты.

Цепочки запросов (chaining)


```
posts = Post.objects                          # ModelManager
posts = posts.filter(title__contains='CSS')   # QuerySet
posts = posts.exclude(id=21)                  # QuerySet
posts = posts.order_by('-rating')             # QuerySet
posts = posts.reverse()                       # QuerySet
posts = posts[:3]                             # [ Post ]
```

Методы QuerySet (chaining)

- `filter`, `exclude` — фильтрация, `WHERE` в SQL;
- `order_by` — сортировка;
- `annotate` — выбор агрегатов, в SQL - `JOIN` и `GROUP_BY`;
- `values` — выбор отдельных колонок, а не объектов;
- `values_list` — то же, только без названия колонок;
- `distinct` — выбор уникальных значений;
- `select_related`, `prefetch_related` — выборка из нескольких таблиц.

Методы QuerySet (результат)

- `create` — создание нового объекта
- `update` — обновление всех подходящих объектов
- `delete` — удаление одного или нескольких объектов
- `get_or_create` — выборка объекта или его создание
- `count` — выборка количества `COUNT(*)`

Синтаксис условий в `filter` и `exclude`

`field = value` — точное совпадение

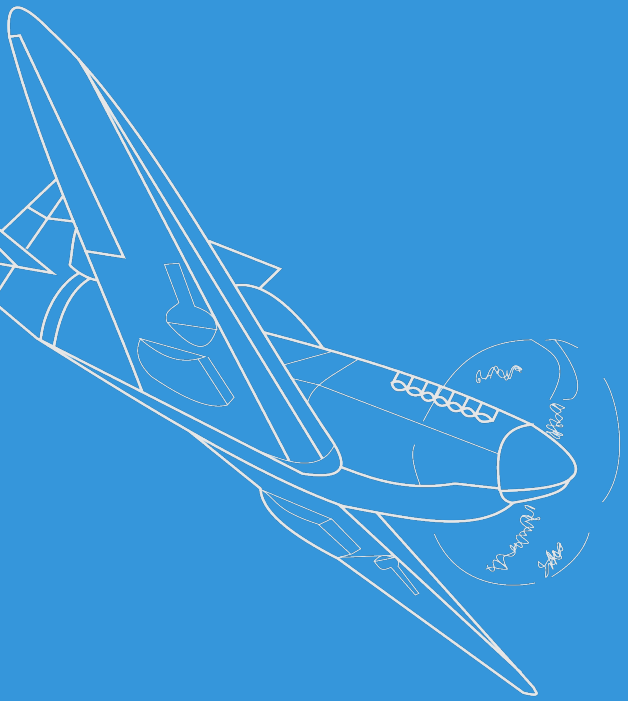
`field__contains = value` — суффикс оператора LIKE

`field__isnull, field__gt, field__lte`

`relation__field = value` — условие по связанной таблице

```
category__title__contains = 'Python'

# Названия полей и таблиц не могут содержать __
```



ModelManager

ModelManager

В модели содержатся методы для работы с одним объектом (одной строкой).

В `ModelManager` содержатся объекты для работы с множеством объектов.

`ModelManager` по умолчанию содержит все те же методы, что и `QuerySet` и используется для создания `QuerySet` объектов, связанных с моделью.

ModelManager «по-умолчанию»


```
class Post(models.Model):
    title = models.CharField()
    ...

posts = Post.objects                    # ModelManager
posts = Post.objects.all()              # QuerySet
posts = Post.objects.filter(id__gt=10) # QuerySet
```

Свой ModelManager


```
class PostManager(models.Manager):
    def best_posts(self):
       return self.filter(rating__gte=50)
    def published(self):
        return self.filter(status=Post.IS_PUBLISHED)
class Post(models.Model):
    title = ...
    ...
    objects = PostManager()
)
```

RelatedManager

```
class Post(models.Model):
    #...
    tags = models.ManyToMany(Tag)

p1 = Post.objects.get(pk=3)
tags = p1.tags # RelatedManager
```

RelatedManager связан с конкретным объектом Post и во все выборки будет добавлять условие post=p1.

Методы RelatedManager

- `create(**kwargs)` — создание новой категории, связанной с постом
- `add(category)` — привязка существующей категории к посту
- `remove(category)` — отвязка
- `clear()` — очистка списка категорий у текущего поста

Что еще?


```
Post.objects.get()
Post.objects.first() # обычно после filter
Post.objects.last()  # обычно после filter
Post.objects.none()
Post.objects.filter(id=2).exists()
```

Усложняем запросы (1)


```
from django.db.models import Count
from django.db.models import Max
from django.db.models.functions import Length

posts = Post.objects.annotate(Count('tags')) # tags__count
posts = Post.objects.all().annotate(tag_count=Count('tags'))
posts = posts.filter(tag_count__gte=3)
posts = Post.objects.all().annotate(length=Length('title'))
users = User.objects.all().aggregate(age_max=Max(age))
```

Усложняем запросы (2)

https://docs.djangoproject.com/en/2.2/misc/design-philosophies/
https://docs.djangoproject.com/en/2.2/topics/db/models/
https://docs.djangoproject.com/en/2.2/ref/models/querysets/

Домашнее задание № 5

1. Реализовать методы для:
    a. Поиска пользователей
    b. Создания персонального чата
    c. Получения списка чатов

Рекомендуемая литература

[Документация Django]

Для саморазвития (опционально)

[Чтобы не набирать двумя пальчиками]

Спасибо за внимание!

Антон Кухтичев

a.kukhtichev@mail.ru

@toshunster