

# Django ORM: выборочное получение данных из БД | Я.Шпора

## Метод `.values()`

При запросе к БД можно получить не полную запись, а только её необходимые поля. В Django ORM эти поля нужно перечислить в аргументах метода

`.values()` :

```
<Модель>.objects.values('<поле1>', '<поле2>')
```

## Расширенный `.filter()`

Метод `.filter()` принимает на вход именованные (keyword) аргументы:

- название поля,
- модификатор поиска,
- значение для фильтрации.

Метод `.filter()` возвращает QuerySet с объектами, которые соответствуют заданным условиям.

```
<Модель>.objects.filter(<свойство>__<модификатор>=<значение для фильтрации>)
```



Самый распространённый модификатор, применяемый в методе

`filter()`

—

`__exact`, «точное совпадение». Если в выражении сравнения модификатор не указан явно — то по умолчанию применяется

`__exact`.

Перечень и описание доступных модификаторов [есть в документации](#).

## Метод .exclude()

Для **исключения** объектов, соответствующих определённому условию, применяют метод `.exclude()`

```
<Модель>.objects.exclude(<свойство>__<модификатор>=<значение для фильтрации>)
```

## Фильтрация по датам

Для фильтрации по датам в Django ORM применяют дополнительные суффиксы `__date`, `__year`, `__month`, `__day`, `__week`, `__week_day`, `__quarter`:

```
# Условия для конкретной даты:
Post.objects.filter(pub_date__date=datetime.date(1890, 1, 1))
# Ранее первого января 1895 года:
Post.objects.filter(pub_date__date__lt=datetime.date(1895, 1, 1))
# Конкретный год:
Post.objects.filter(pub_date__year=1890)
# Любой год с января по июнь включительно:
Post.objects.filter(pub_date__month__lte=6)
# Первый квартал любого года:
Post.objects.filter(pub_date__quarter=1)
```

Остальные примеры будут приведены на двух связанных моделях — `IceCream` и `Category`:

```
from django.db import models

class Category(PublishedModel):
    title = models.CharField(max_length=256)
    slug = models.SlugField(max_length=64, unique=True)
    output_order = models.PositiveSmallIntegerField(default=100)
    is_published = models.BooleanField(default=True)

class IceCream(PublishedModel):
    title = models.CharField(max_length=256)
```

```
description = models.TextField()
category = models.ForeignKey(
    Category,
    on_delete=models.CASCADE,
    related_name='ice_creams',
)
```

## Объединение условий

Объединить несколько условий в методе `.filter()` можно через запятую. В SQL-запросе условия будут объединены через `AND`.

```
ice_cream_list = IceCream.objects.values(
    'id', 'title', 'description'
).filter(
    is_published=True, is_on_main=True # Два в одном!
)
```

Иногда требуется составить более сложный комбинированный запрос. В Django ORM для этого применяют **Q-объекты**.

В **Q-объект** передаётся название поля, модификатор и значение для фильтрации, а сами объекты объединяются в запрос логическими операторами `~` (NOT), `&` (AND) и `|` (OR):

SQL	ORM	Что делает
<условие 1> NOT <условие 2>	Q(<условие 1>) & ~Q(<условие 2>)	Выполняется <b>условие 1</b> и одновременно НЕ выполняется <b>условие 2</b>
<условие 1> AND <условие 2>	Q(<условие 1>) & Q(<условие 2>)	Выполняется <b>условие 1</b> и одновременно выполняется <b>условие 2</b>
<условие 1> OR <условие 2>	Q(<условие 1>)   Q(<условие 2>)	Выполняется <b>условие 1</b> или выполняется <b>условие 2</b>

## Примеры запросов с логическими операторами

**SQL:** из таблицы `ice_cream_icecream` получаем записи, у которых значения полей `is_on_main` и `is_published` равны `TRUE`:

```
SELECT "ice_cream_icecream"."id"  
FROM "ice_cream_icecream"  
WHERE ("ice_cream_icecream"."is_on_main" AND "ice_cream_icecream"."is_published")
```

В Django ORM такой запрос можно написать по-разному:

```
# Вариант 1, через запятую в аргументах метода .filter():  
IceCream.objects  
.values('id')  
.filter(is_published=True, is_on_main=True)  
  
# Вариант 2, через Q-объекты:  
IceCream.objects  
.values('id')  
.filter(Q(is_published=True) & Q(is_on_main=True))  
  
# Вариант 3, дважды вызываем метод .filter();  
# так обычно не пишут, но иногда этот вариант тоже встречается:  
IceCream.objects  
.values('id')  
.filter(is_published=True).filter(is_on_main=True)
```

## Логический оператор OR

**SQL:** получаем записи, у которых поле `is_on_main` ИЛИ поле `is_published` равно `True`:

```
SELECT "ice_cream_icecream"."id"  
FROM "ice_cream_icecream"  
WHERE ("ice_cream_icecream"."is_on_main" OR "ice_cream_icecream"."is_published")
```

## Django ORM:

```
# Можно так, через Q-объекты:
IceCream.objects
.values('id')
.filter(Q(is_published=True) | Q(is_on_main=True))

# А можно и так - более многословно, без Q-объектов:
IceCream.objects.values('id').filter(is_published=True)
| IceCream.objects.values('id').filter(is_on_main=True)
```

## Логический оператор NOT

**SQL:** получаем записи, у которых поле `is_published` равно `True` и одновременно поле `is_on_main` не равно `False`:

```
SELECT "ice_cream_icecream"."id",
FROM "ice_cream_icecream"
WHERE ("ice_cream_icecream"."is_published"
      AND NOT (NOT "ice_cream_icecream"."is_on_main") -- HE (HE True)
```

## Django ORM:

```
# Лучше так:
IceCream.objects
.values('id')
.filter(Q(is_published=True) & ~Q(is_on_main=False))

# Но сработает и так:
IceCream.objects
.values('id')
.filter(is_published=True)
.exclude(is_on_main=False)
```

Приоритет выполнения логических операторов: оператор `NOT` имеет самый высокий приоритет (выполняется первым), следующий по приоритету — оператор `AND`, а самый последний — оператор `OR`. Для объединения условий в группы применяются скобки `()`.

## Сортировка через класс Meta в модели

Порядок сортировки по умолчанию можно объявить прямо в модели:

```
# <название_приложения>/models.py

class <Модель>(models.Model):
    ...

    class Meta:
        ordering = ('<название_поля>',)
```

Чтобы поменять порядок сортировки и упорядочить объекты от больших значений к меньшим — достаточно поставить символ «минус» `-` перед названием поля, по которому проводится сортировка:

```
ordering = ('-<название_поля>',)
```

Порядок сортировки можно указать и в конкретном запросе; для этого у `objects` есть метод `.order_by()`:

```
<Модель>.objects.order_by('<название_поля>')
```

Если правила сортировки указаны одновременно в `Meta` и в `objects.order_by()` — будут применены правила из `objects.order_by()`.

Метод `.order_by()` может сортировать и по нескольким полям:

```
<Модель>.objects.order_by('<название_поля_1>', '<название_поля_2>')
```

## LIMIT и OFFSET

Ограничить количество объектов в QuerySet можно с помощью **срезов**. QuerySet хранит **список** словарей или объектов модели, и к нему можно применять все инструменты, которыми в Python обрабатываются списки.

```
ice_cream_list = IceCream.objects.values(
    'id', 'title', 'description'
).filter(
```

```
is_published=True, is_on_main=True
).order_by('title')[1:4]
```

## Метод .get(): получение одного объекта

```
ice_cream = IceCream.objects.get(pk=1)
```

## Функция get\_object\_or\_404()

Если методом `.get()` запросить из базы несуществующий объект — Django выбросит исключение `DoesNotExist: IceCream matching query does not exist`.

Исключения можно избежать, применив функцию `get_object_or_404()`. На вход она ожидает

- аргумент `klass`: имя модели или QuerySet, из которого нужно получить запрошенный объект;
- аргументы `*args, **kwargs` — параметры для фильтрации.

Перед применением функции `get_object_or_404()` необходимо её импортировать.

```
from django.shortcuts import get_object_or_404, render

from ice_cream.models import IceCream

def ice_cream_detail(request, pk):
    template_name = 'ice_cream/detail.html'
    # Отфильтруй объект модели IceCream,
    # у которого pk равен значению переменной из пути.
    # Если такого объекта не существует - верни 404 ошибку:
    ice_cream = get_object_or_404(IceCream, pk=pk)
    context = {
        'ice_cream': ice_cream,
    }
    return render(request, template_name, context)
```

## Получить первый или последний объект из QuerySet: методы `.first()` и `.last()`

Отдельный объект можно получить и таким способом:

- получить QuerySet,
- методом `.first()` или `.last()` получить из него первый или последний объект.

```
# Такой запрос вернёт первый элемент из QuerySet:  
IceCream.objects.filter(is_published=True).order_by('pk').first()
```

## JOIN с помощью метода `.values()`

Метод `.values()` может вернуть не только поля объектов запрошенной модели, но и значения полей той модели, которая связана с запрошенной:

```
ice_cream_list = IceCream.objects.values('id', 'title', 'category__title')  
# values(..., '<поле fk>__<поле в модели, связанной по fk>')
```

В аргументе метода `.values()` передаётся имя атрибута, где хранится внешний ключ ( `category` в приведённом примере), и через двойное нижнее подчёркивание — название того поля связанной модели, значение которого нужно получить.

Подобный синтаксис с двойным подчёркиванием применяется и в Django-шаблоне для получения и вывода значения поля из связанной модели.

```
{% for ice_cream in ice_cream_list %}  
  <h3>{{ ice_cream.title }} ID: {{ ice_cream.id }}</h3>  
  <p>Категория: {{ ice_cream.category__title }}</p>  
{% endfor %}
```

## JOIN с помощью `.select_related()`

Другой способ создать JOIN-запрос в Django ORM — вызвать метод `.select_related()`.



```
ice_cream_list = IceCream.objects.select_related('category')
```

В результате выполнения команды будет сформирована выборка, выполнение которой сформирует один сложный SQL запрос:

```
SELECT
    "icecream"."id",
    "icecream"."title",
    "icecream"."description",
    "icecream"."category_id",
    "category"."id" AS "category_id",
    "category"."title" AS "category_title",
    "category"."slug" AS "category_slug",
    "category"."output_order" AS "category_output_order",
    "category"."is_published" AS "category_is_published"
FROM
    "icecream_icecream" AS "icecream"
INNER JOIN
    "icecream_category" AS "category" ON ("icecream"."category_id" = "cat
```

В результате вернётся информация о мороженом и связанной категории в одном запросе.

## Фильтры в связанных моделях

Итоговую выборку, получившуюся при запросе к связанным моделям, можно отфильтровать по полям связанной модели.

Фильтрация при работе с методом `.values()` :

```
ice_cream_list = IceCream.objects.values(
    'id', 'title', 'category__title'
).filter(
    # Вернуть только те объекты IceCream, у которых
    # в связанном объекте Category в поле is_published хранится значение
    True:
    category__is_published=True
)
```

Фильтрация при работе с методом `.select_related()`:

```
ice_cream_list = IceCream.objects.select_related(
    'category'
).filter(
    # В точности то же самое:
    category__is_published=True
)
```

 Практикум