بسم الله

django

DJANGO MODELS

HOSSEIN FORGHANI

MAKTAB SHARIF

Contents

Model Fields

Some of Field options

- null: If True, Django will store empty values as NULL in the database
- blank: If True, the field is allowed to be blank
- choices: choices for this field
- db_column: The name of the database column to use for this field
- db_index: If True, a database index will be created for this field
- default: The default value for the field
- editable: If False, the field will not be displayed in the admin or any other ModelForm
- help_text: Extra “help” text to be displayed with the form widget

Some of Field options – cont.

- primary_key: If True, this field is the primary key for the model
- unique: If True, this field must be unique throughout the table
- verbose_name: A human-readable name for the field
- validators: A list of validators to run for this field

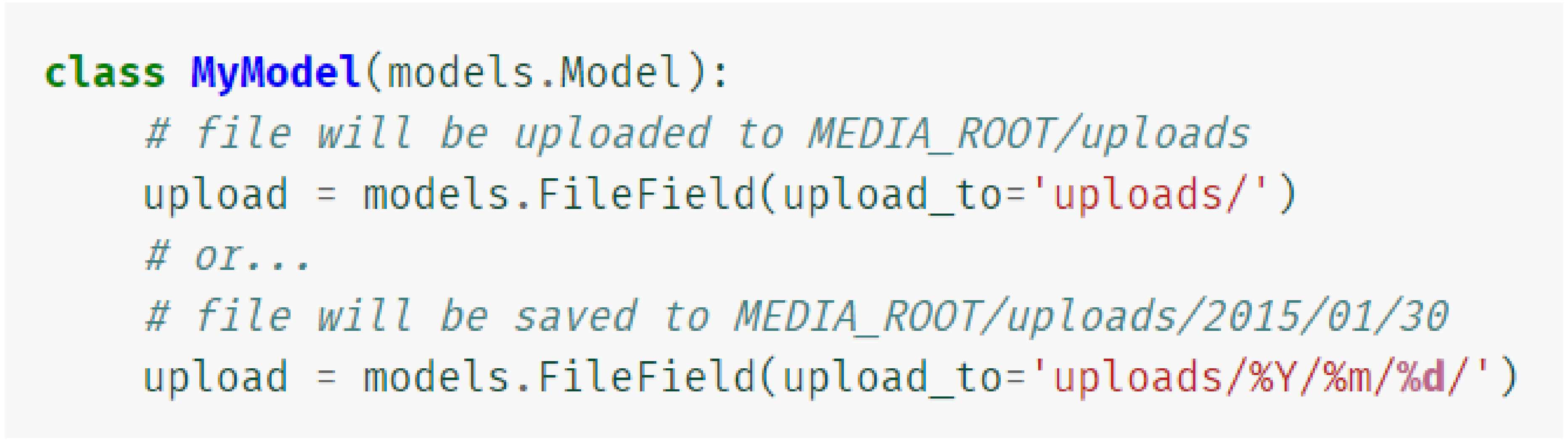
Choices Example

```
from django.db import models

class Student(models.Model):
    FRESHMAN = 'FR'
    SOPHOMORE = 'SO'
    JUNIOR = 'JR'
    SENIOR = 'SR'
    GRADUATE = 'GR'
    YEAR_IN_SCHOOL_CHOICES = [
        (FRESHMAN, 'Freshman'),
        (SOPHOMORE, 'Sophomore'),
        (JUNIOR, 'Junior'),
        (SENIOR, 'Senior'),
        (GRADUATE, 'Graduate'),
    ]
    year_in_school = models.CharField(
        max_length=2,
        choices=YEAR_IN_SCHOOL_CHOICES,
        default=FRESHMAN,
    )

    def is_upperclass(self):
        return self.year_in_school in {self.JUNIOR, self.SENIOR}
```

Some of Field types

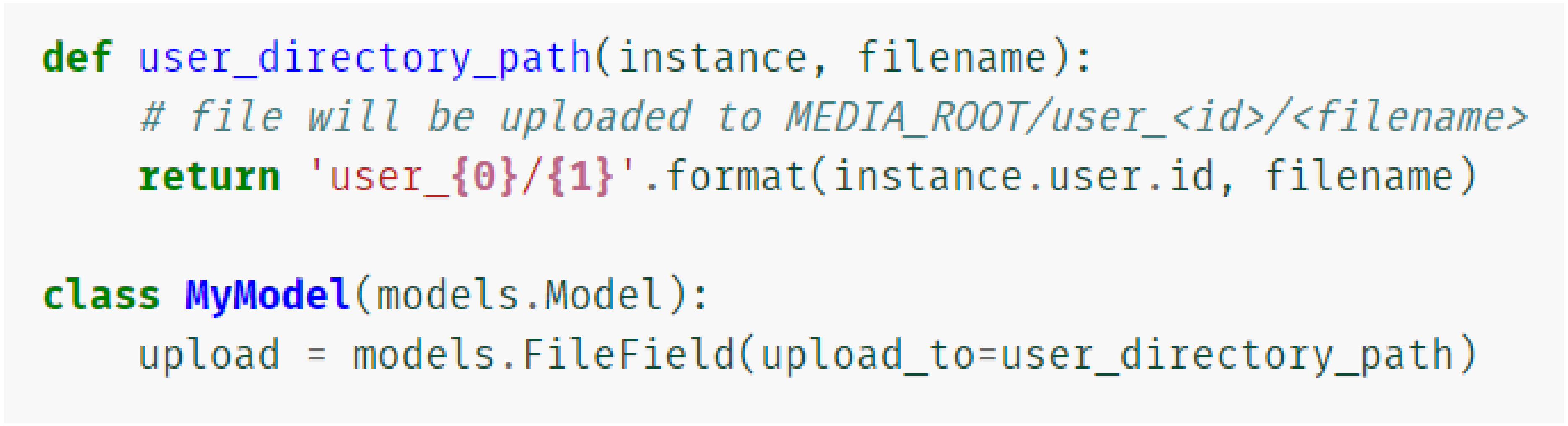
- IntegerField
- FloatField
- BooleanField
- NullBooleanField
- CharField
- TextField
- DateField
- TimeField
- DateTimeField
- DecimalField
- DurationField
- EmailField
- URLField
- FileField
- ImageField
- ForeignKey
- ManyToManyField
- OneToOneField

FileField Example



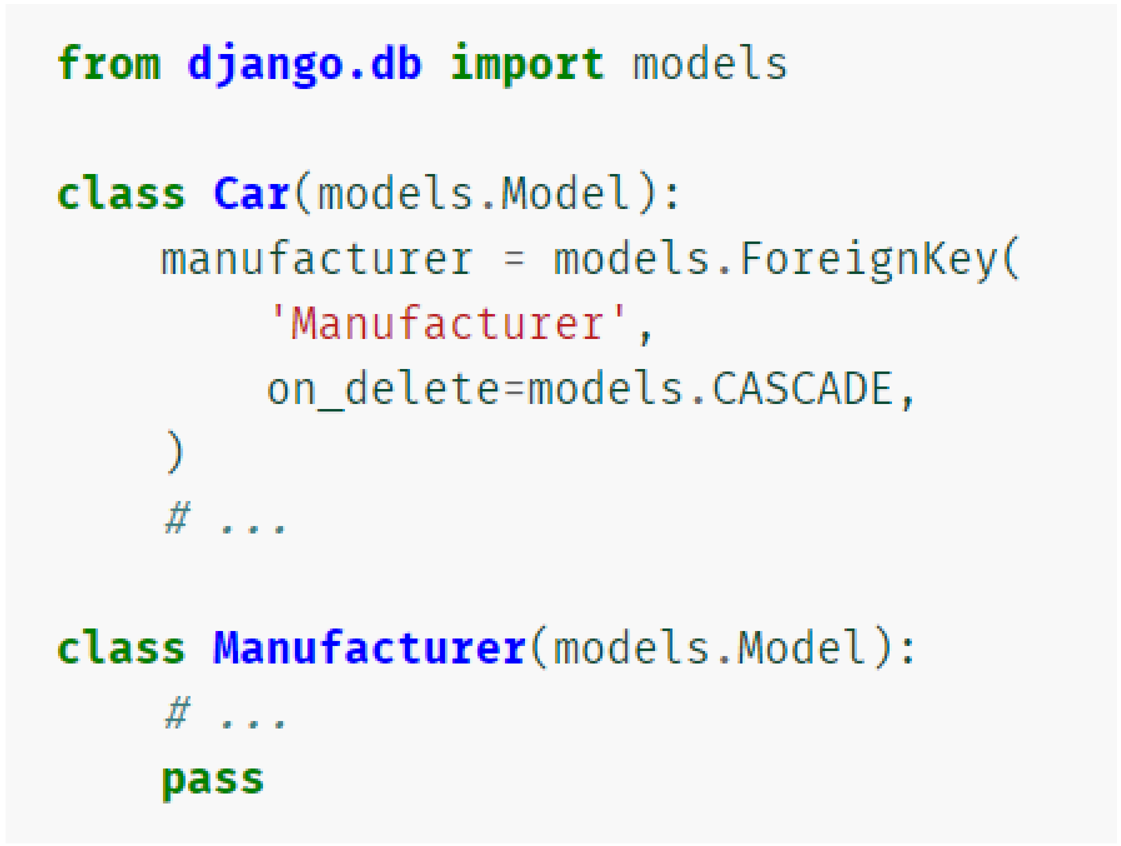
```python
class MyModel(models.Model):
    # file will be uploaded to MEDIA_ROOT/uploads
    upload = models.FileField(upload_to='uploads/')
    # or...
    # file will be saved to MEDIA_ROOT/uploads/2015/01/30
    upload = models.FileField(upload_to='uploads/%Y/%m/%d/')
```

FileField Example 2



```python
def user_directory_path(instance, filename):
    # file will be uploaded to MEDIA_ROOT/user_<id>/<filename>
    return 'user_{0}/{1}'.format(instance.user.id, filename)

class MyModel(models.Model):
    upload = models.FileField(upload_to=user_directory_path)
```

ForeignKey



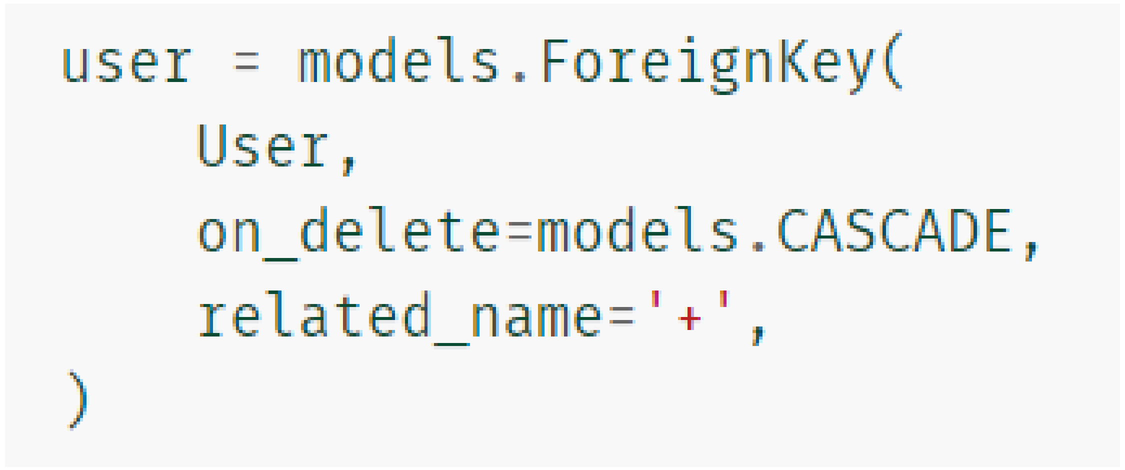
```
from django.db import models

class Car(models.Model):
    manufacturer = models.ForeignKey(
        'Manufacturer',
        on_delete=models.CASCADE,
    )
    # ...

class Manufacturer(models.Model):
    # ...
    pass
```

ForeignKey on_delete

- CASCADE
- PROTECT
- RESTRICT
- SET_NULL
- SET_DEFAULT
- SET()
- DO_NOTHING

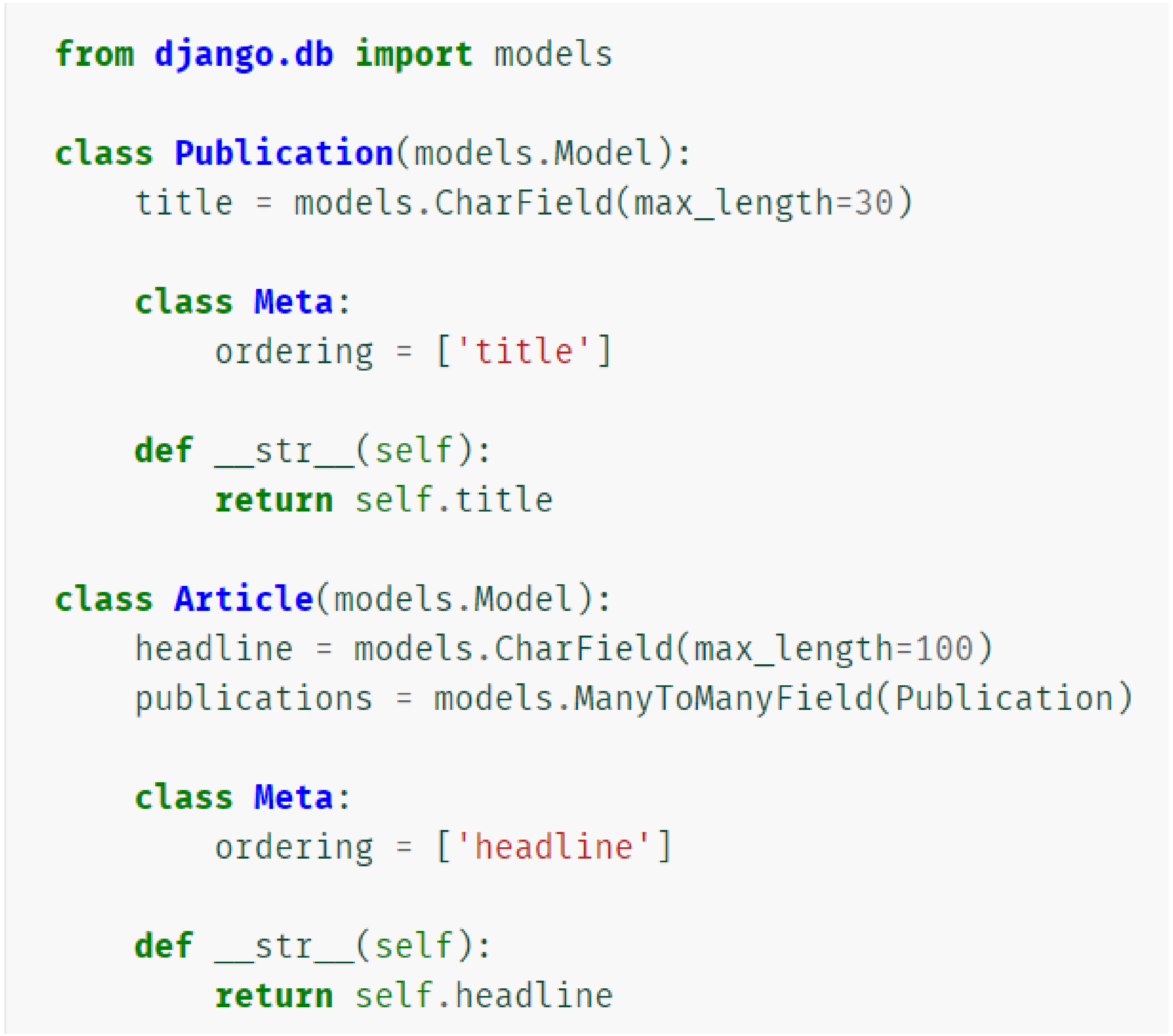
ForeignKey related_name

- The default related_name is ‘model_set’
- You can specify it as you want (for example if you have 2 foreign keys to the same model)
- If you’d prefer Django not to create a backwards relation, set related_name to ‘+’ :



```
user = models.ForeignKey(
    User,
    on_delete=models.CASCADE,
    related_name='+',
)
```

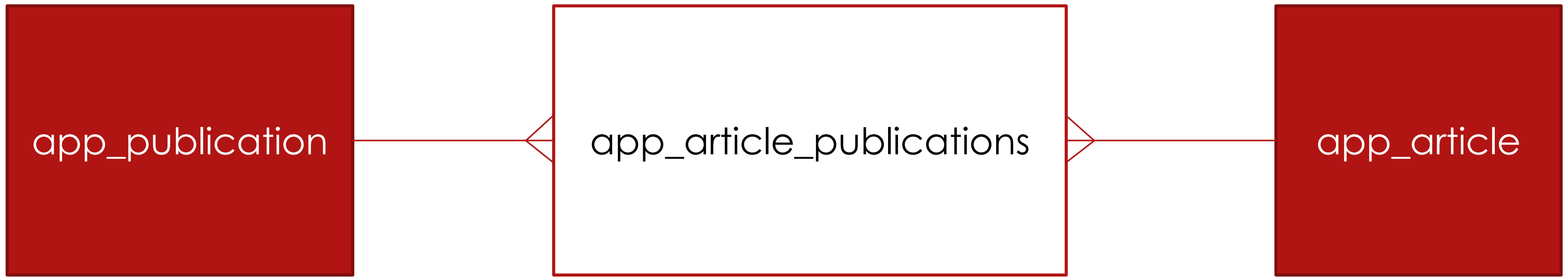
ManyToManyField



```python
from django.db import models

class Publication(models.Model):
    title = models.CharField(max_length=30)

    class Meta:
        ordering = ['title']

    def __str__(self):
        return self.title

class Article(models.Model):
    headline = models.CharField(max_length=100)
    publications = models.ManyToManyField(Publication)

    class Meta:
        ordering = ['headline']

    def __str__(self):
        return self.headline
```

ERD

- Creates a relationship table behind the scenes



ManyToManyField

- Suppose a1 is a saved Article and p1 is a saved Publication
- To add p1 to a1's publications do:

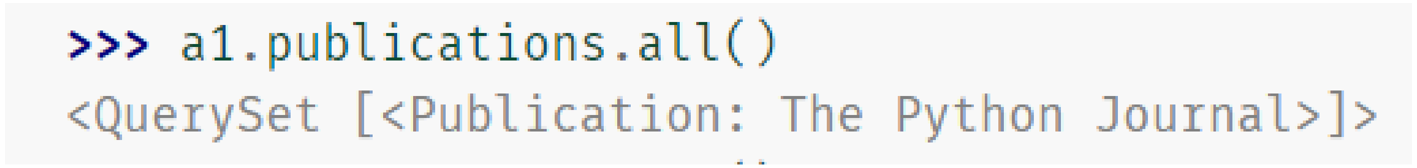
```
>>> a1.publications.add(p1)
```

- Or create and add:

```
>>> new_publication = a2.publications.create(title='Highlights
for Children')
```

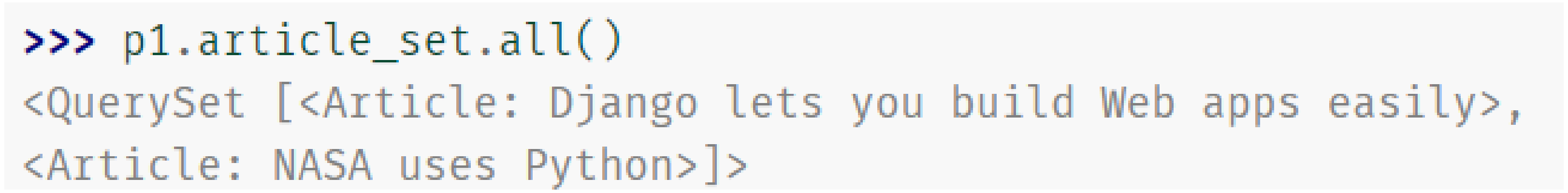
ManyToManyField – cont.

- Article objects have access to their related Publication objects:



```
>>> a1.publications.all()
<QuerySet [<Publication: The Python Journal>]>
```

- Publication objects have access to their related Article objects:



```
>>> p1.article_set.all()
<QuerySet [<Article: Django lets you build Web apps easily>,
<Article: NASA uses Python>]>
```

ManyToManyField – cont.

- Many-to-many relationships can be queried using lookups across relationships:

```
>>>
Article.objects.filter(publications__title__startswith="Science")
<QuerySet [<Article: NASA uses Python>, <Article: NASA uses
Python>]>
```

- Removing Publication from an Article and vice versa:

```
>>> a4.publications.remove(p2)
```

```
>>> p2.article_set.remove(a5)
```

OneToOneField

- Conceptually, this is similar to a ForeignKey with unique=True
- But the “reverse” side of the relation will directly return a single object
- Default value of related_name is lowercase name of the current model

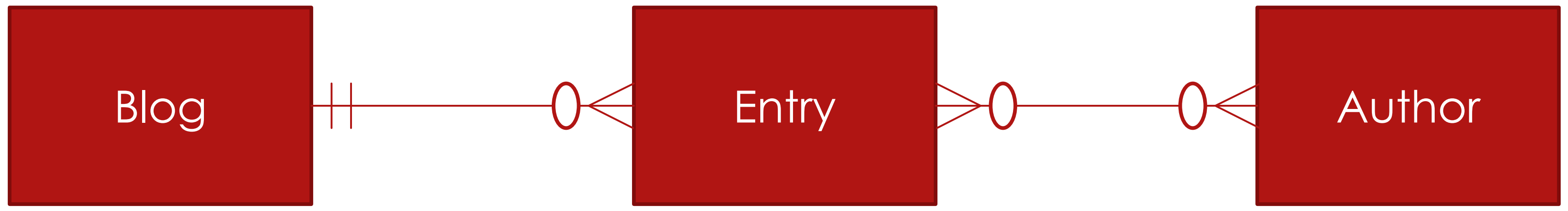
```python
from django.conf import settings
from django.db import models

class MySpecialUser(models.Model):
    user = models.OneToOneField(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
    )
    supervisor = models.OneToOneField(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='supervisor_of',
    )
```

Making Queries

We use these models throughout this lesson

```python
from django.db import models

class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

    def __str__(self):
        return self.name

class Author(models.Model):
    name = models.CharField(max_length=200)
    email = models.EmailField()

    def __str__(self):
        return self.name

class Entry(models.Model):
    blog = models.ForeignKey(Blog, on_delete=models.CASCADE)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
    mod_date = models.DateField()
    authors = models.ManyToManyField(Author)
    number_of_comments = models.IntegerField()
    number_of_pingbacks = models.IntegerField()
    rating = models.IntegerField()

    def __str__(self):
        return self.headline
```

ERD



Creating objects

- Does INSERT SQL statement

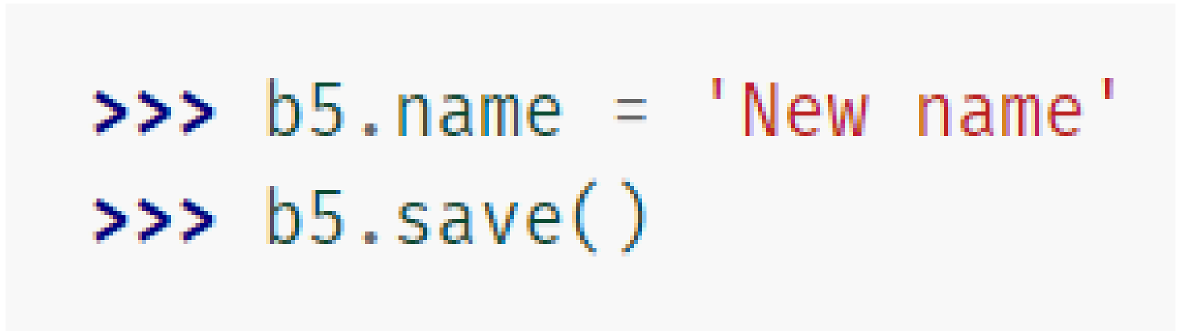
```
>>> from blog.models import Blog
>>> b = Blog(name='Beatles Blog', tagline='All the latest Beatles
news.')
>>> b.save()
```

- Or:

```
p = Person.objects.create(first_name="Bruce",
last_name="Springsteen")
```

Saving changes to objects

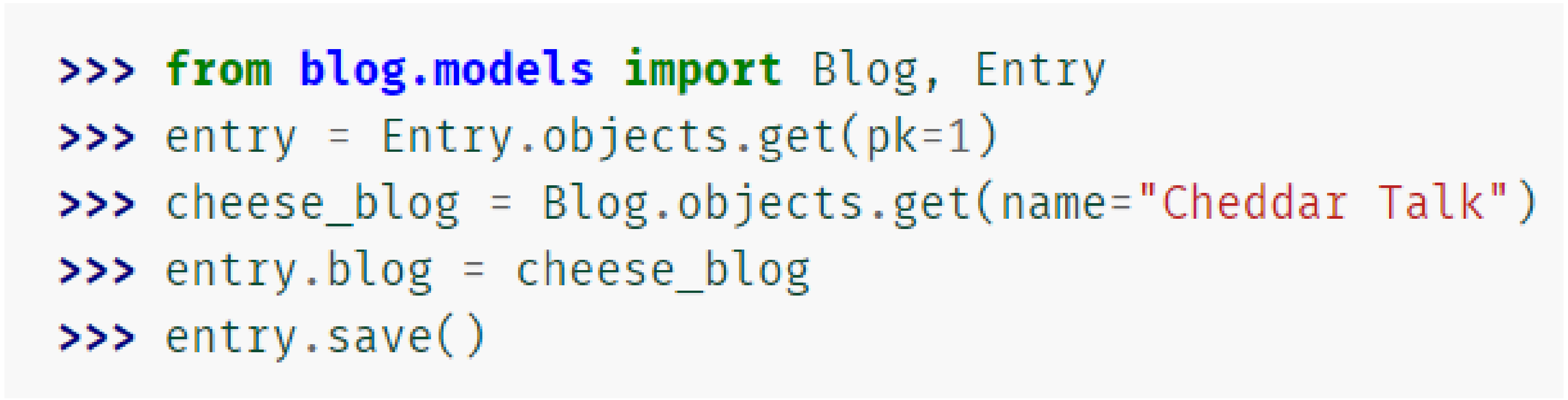
- Does UPDATE SQL statement



```
>>> b5.name = 'New name'
>>> b5.save()
```

Saving ForeignKey and ManyToManyField fields

- exactly the same way as saving a normal field



```
>>> from blog.models import Blog, Entry
>>> entry = Entry.objects.get(pk=1)
>>> cheese_blog = Blog.objects.get(name="Cheddar Talk")
>>> entry.blog = cheese_blog
>>> entry.save()
```

Retrieving objects

- Retrieving all objects:

```
>>> all_entries = Entry.objects.all()
```

Manager

Returns a Queryset

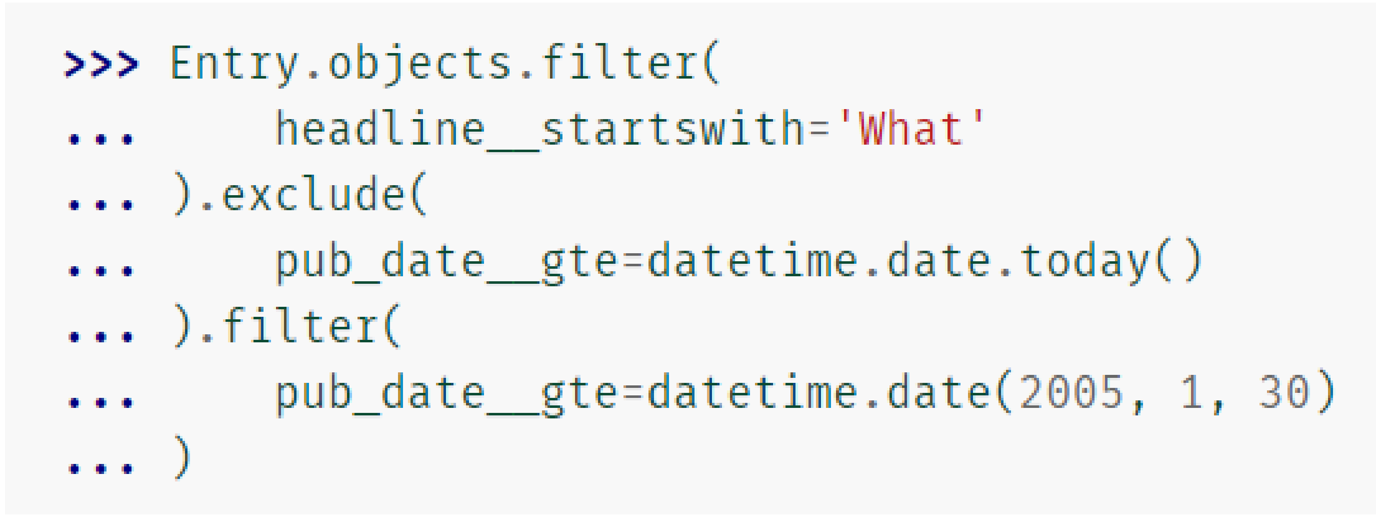
- Retrieving specific objects filter(**kwargs) and exclude(**kwargs)

```
Entry.objects.filter(pub_date__year=2006)
```

Returns a Queryset

Queryset

- The result of refining a QuerySet is itself a QuerySet

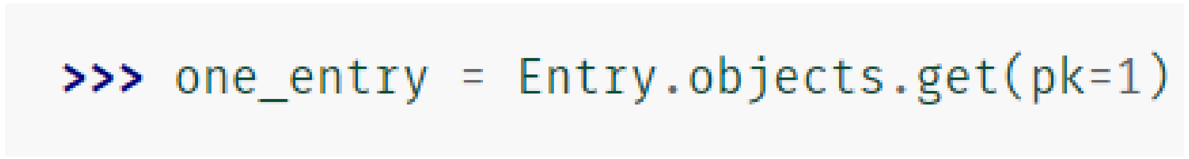


```
>>> Entry.objects.filter(
...     headline__startswith='What'
... ).exclude(
...     pub_date__gte=datetime.date.today()
... ).filter(
...     pub_date__gte=datetime.date(2005, 1, 30)
... )
```

- QuerySets are lazy

Retrieving a single object

- Retrieving a single object with get()

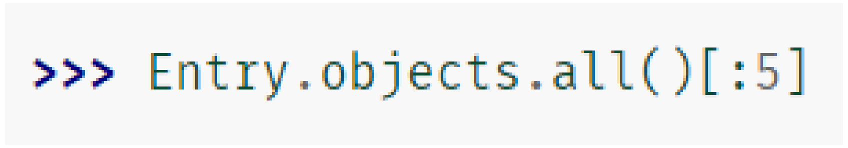


```
>>> one_entry = Entry.objects.get(pk=1)
```

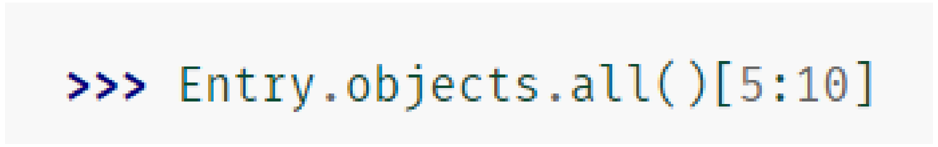
- If there are no results, will raise a DoesNotExist exception
- If more than one item matches, it will raise MultipleObjectsReturned

Limiting QuerySets

- equivalent of SQL's LIMIT and OFFSET



```
>>> Entry.objects.all()[:5]
```



```
>>> Entry.objects.all()[5:10]
```

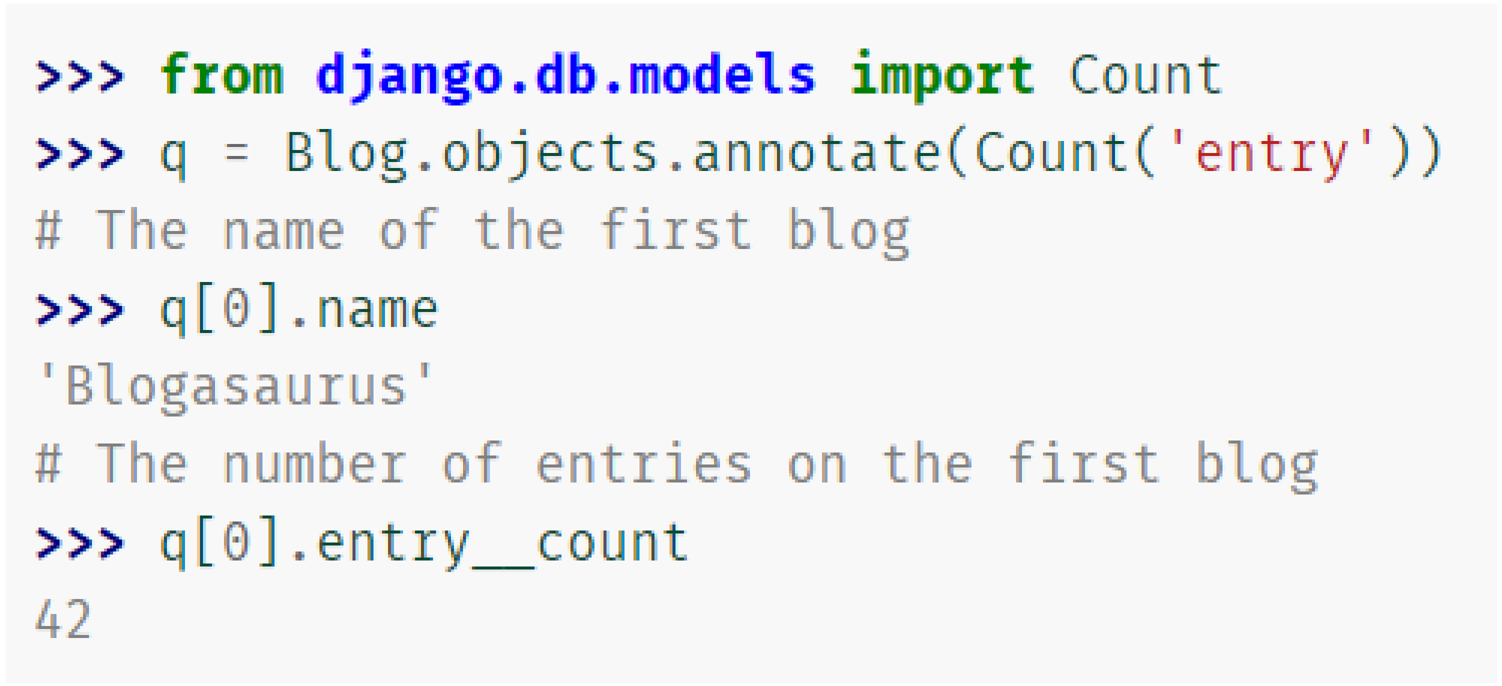
- Negative indexing (i.e. Entry.objects.all()[-1]) is not supported

Some of Other QuerySet methods

- order_by()
- reverse()
- distinct()
- values()
- values_list()
- dates()
- datetimes()
- none()
- union()
- intersection()
- difference()
- select_related()
- prefetch_related()
- defer()
- only()
- raw()
- get_or_create()
- update_or_create()
- bulk_create()
- bulk_update()
- count()
- latest()
- earliest()
- aggregate()
- annotate()
- exists()
- update()
- delete()

annotate(*args, **kwargs)

- Annotates each object in the QuerySet with the provided list of query expressions



```
>>> from django.db.models import Count
>>> q = Blog.objects.annotate(Count('entry'))
# The name of the first blog
>>> q[0].name
'Blogasaurus'
# The number of entries on the first blog
>>> q[0].entry__count
42
```

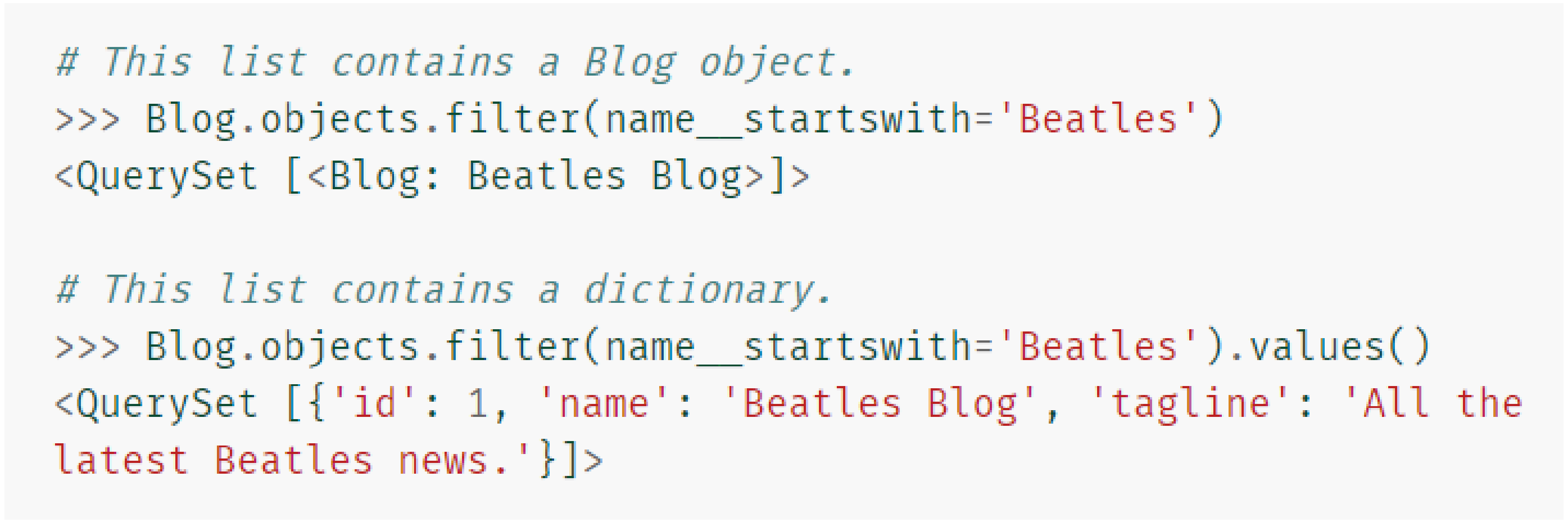
order_by(*fields)

```
Entry.objects.filter(pub_date__year=2005).order_by('-pub_date',
'headline')
```

```
Entry.objects.order_by('blog__name')
```

values(*fields, **expressions)

- Returns a QuerySet that returns dictionaries, rather than model instances



```
# This list contains a Blog object.
>>> Blog.objects.filter(name__startswith='Beatles')
<QuerySet [<Blog: Beatles Blog>]>

# This list contains a dictionary.
>>> Blog.objects.filter(name__startswith='Beatles').values()
<QuerySet [{'id': 1, 'name': 'Beatles Blog', 'tagline': 'All the
latest Beatles news.'}]>
```

values_list(*fields, flat=False, named=False)

```
>>> Entry.objects.values_list('id', 'headline')
<QuerySet [(1, 'First entry'), ...]>
>>> from django.db.models.functions import Lower
>>> Entry.objects.values_list('id', Lower('headline'))
<QuerySet [(1, 'first entry'), ...]>
```

```
>>> Entry.objects.values_list('id', flat=True).order_by('id')
<QuerySet [1, 2, 3, ...]>
```

dates(field, kind, order='ASC')

```
>>> Entry.objects.dates('pub_date', 'year')
[datetime.date(2005, 1, 1)]
>>> Entry.objects.dates('pub_date', 'month')
[datetime.date(2005, 2, 1), datetime.date(2005, 3, 1)]
>>> Entry.objects.dates('pub_date', 'week')
[datetime.date(2005, 2, 14), datetime.date(2005, 3, 14)]
>>> Entry.objects.dates('pub_date', 'day')
[datetime.date(2005, 2, 20), datetime.date(2005, 3, 20)]
>>> Entry.objects.dates('pub_date', 'day', order='DESC')
[datetime.date(2005, 3, 20), datetime.date(2005, 2, 20)]
>>>
Entry.objects.filter(headline__contains='Lennon').dates('pub_date
', 'day')
[datetime.date(2005, 3, 20)]
```

Union, Intersection, Difference

- Uses SQL's UNION operator to combine the results of two or more QuerySets:

```
>>> qs1.union(qs2, qs3)
```

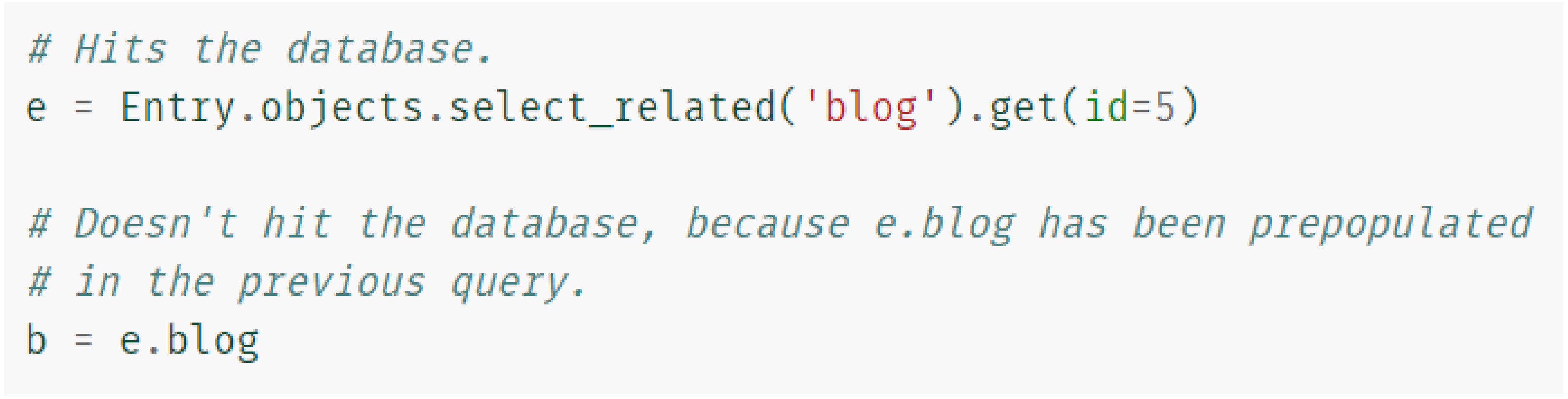
- Also intersection() and difference()

```
>>> qs1.intersection(qs2, qs3)
```

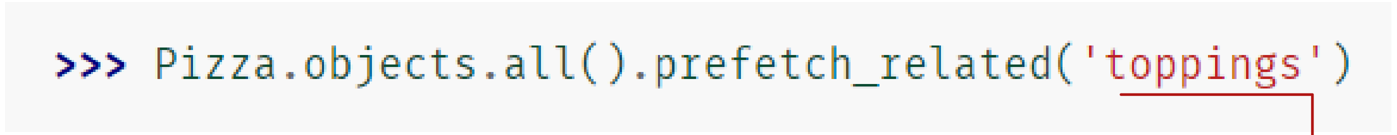
```
>>> qs1.difference(qs2, qs3)
```

select_related() & prefetch_related()

- select_related for ForeignKey and OneToOneField
- prefetch_related for ManyToManyFields



```
# Hits the database.
e = Entry.objects.select_related('blog').get(id=5)

# Doesn't hit the database, because e.blog has been prepopulated
# in the previous query.
b = e.blog
```

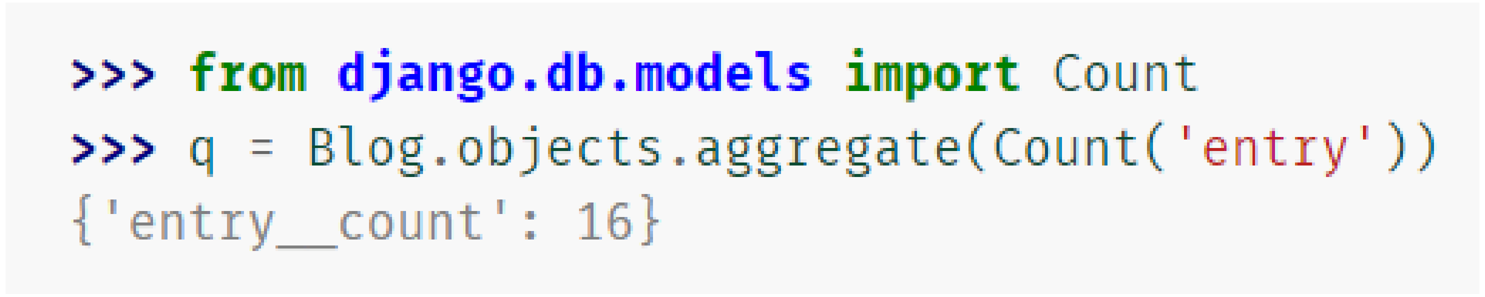


```
>>> Pizza.objects.all().prefetch_related('toppings')
```

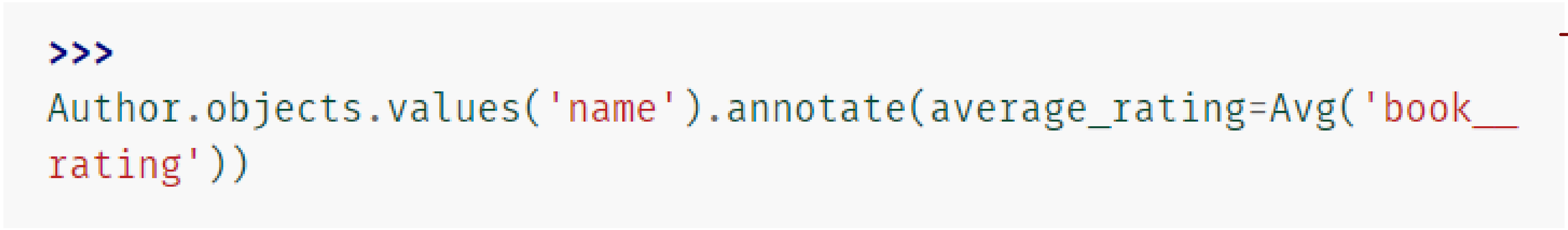
Suppose this is a ManyToManyField

aggregate()

- Returns a dictionary of aggregate values (averages, sums, etc.) calculated over the QuerySet



```
>>> from django.db.models import Count
>>> q = Blog.objects.aggregate(Count('entry'))
{'entry__count': 16}
```



```
>>>
Author.objects.values('name').annotate(average_rating=Avg('book__
rating'))
```

Group by

Update() and delete()

```
>>>
Entry.objects.filter(pub_date__year=2010).update(comments_on=Fals
e)
```

```
# Delete all the entries belonging to this Blog.
>>> Entry.objects.filter(blog=b).delete()
```

Some of Field lookups

- contains
- in
- gt
- gte
- lt
- lte
- range
- startswith
- endswith
- date
- year
- month
- day
- time
- isnull
- regex

References

- https://docs.djangoproject.com/en/3.1/topics/db/queries/
- https://docs.djangoproject.com/en/3.1/topics/db/examples/many_to_many/
- https://docs.djangoproject.com/en/3.1/ref/models/fields/
- https://docs.djangoproject.com/en/3.1/topics/db/examples/many_to_many/
- https://docs.djangoproject.com/en/3.1/ref/models/querysets

Any Question?