***Examples of model relationship API usage***

**Many-to-many relationships**

To define a many-to-many relationship, use ManyToManyField.

In this example, an Article can be published in multiple Publication objects, and a Publication has multiple Article objects:

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

What follows are examples of operations that can be performed using the Python API facilities.

Create a few Publications:

>>> p1 = Publication(title="The Python Journal")

>>> p1.save()

>>> p2 = Publication(title="Science News")

>>> p2.save()

>>> p3 = Publication(title="Science Weekly")

>>> p3.save()

Create an Article:

>>> a1 = Article(headline="Django lets you build web apps easily")

You can’t associate it with a Publication until it’s been saved:

>>> a1.publications.add(p1)

Traceback (most recent call last):

...

ValueError: "<Article: Django lets you build web apps easily>" needs to have a value for field "id

,→" before this many-to-many relationship can be used.

Save it!

>>> a1.save()

Associate the Article with a Publication:

>>> a1.publications.add(p1)

Create another Article, and set it to appear in the Publications:

>>> a2 = Article(headline="NASA uses Python")

>>> a2.save()

>>> a2.publications.add(p1, p2)

>>> a2.publications.add(p3)

Adding a second time is OK, it will not duplicate the relation:>>> a2.publications.add(p3)

Adding an object of the wrong type raises TypeError:

>>> a2.publications.add(a1)

Traceback (most recent call last):

...

TypeError: 'Publication' instance expected

Create and add a Publication to an Article in one step using create():

>>> new\_publication = a2.publications.create(title="Highlights for Children")

Article objects have access to their related Publication objects:

>>> a1.publications.all()

<QuerySet [<Publication: The Python Journal>]>

>>> a2.publications.all()

<QuerySet [<Publication: Highlights for Children>, <Publication: Science News>, <Publication:␣

,→Science Weekly>, <Publication: The Python Journal>]>

Publication objects have access to their related Article objects:

>>> p2.article\_set.all()

<QuerySet [<Article: NASA uses Python>]>

>>> p1.article\_set.all()

<QuerySet [<Article: Django lets you build web apps easily>, <Article: NASA uses Python>]>

>>> Publication.objects.get(id=4).article\_set.all()

<QuerySet [<Article: NASA uses Python>]>

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

>>> Article.objects.filter(publications\_\_id=1)

<QuerySet [<Article: Django lets you build web apps easily>, <Article: NASA uses Python>]>

>>> Article.objects.filter(publications\_\_pk=1)

<QuerySet [<Article: Django lets you build web apps easily>, <Article: NASA uses Python>]>

>>> Article.objects.filter(publications=1)

<QuerySet [<Article: Django lets you build web apps easily>, <Article: NASA uses Python>]>

>>> Article.objects.filter(publications=p1)

<QuerySet [<Article: Django lets you build web apps easily>, <Article: NASA uses Python>]>

>>> Article.objects.filter(publications\_\_title\_\_startswith="Science")

<QuerySet [<Article: NASA uses Python>, <Article: NASA uses Python>]>

>>> Article.objects.filter(publications\_\_title\_\_startswith="Science").distinct()

<QuerySet [<Article: NASA uses Python>]>

The count() function respects distinct() as well:

>>> Article.objects.filter(publications\_\_title\_\_startswith="Science").count()

2

>>> Article.objects.filter(publications\_\_title\_\_startswith="Science").distinct().count()

1

>>> Article.objects.filter(publications\_\_in=[1, 2]).distinct()

<QuerySet [<Article: Django lets you build web apps easily>, <Article: NASA uses Python>]>

>>> Article.objects.filter(publications\_\_in=[p1, p2]).distinct()

<QuerySet [<Article: Django lets you build web apps easily>, <Article: NASA uses Python>]>

Reverse m2m queries are supported (i.e., starting at the table that doesn’t have a ManyToManyField):

>>> Publication.objects.filter(id=1)

<QuerySet [<Publication: The Python Journal>]>

>>> Publication.objects.filter(pk=1)

<QuerySet [<Publication: The Python Journal>]>

>>> Publication.objects.filter(article\_\_headline\_\_startswith="NASA")

<QuerySet [<Publication: Highlights for Children>, <Publication: Science News>, <Publication:␣

,→Science Weekly>, <Publication: The Python Journal>]>

>>> Publication.objects.filter(article\_\_id=1)

<QuerySet [<Publication: The Python Journal>]>

>>> Publication.objects.filter(article\_\_pk=1)

<QuerySet [<Publication: The Python Journal>]>

>>> Publication.objects.filter(article=1)

<QuerySet [<Publication: The Python Journal>]>

>>> Publication.objects.filter(article=a1)

<QuerySet [<Publication: The Python Journal>]>

>>> Publication.objects.filter(article\_\_in=[1, 2]).distinct()

<QuerySet [<Publication: Highlights for Children>, <Publication: Science News>, <Publication:␣

,→Science Weekly>, <Publication: The Python Journal>]>

>>> Publication.objects.filter(article\_\_in=[a1, a2]).distinct()

<QuerySet [<Publication: Highlights for Children>, <Publication: Science News>, <Publication:␣

,→Science Weekly>, <Publication: The Python Journal>]>

Excluding a related item works as you would expect, too (although the SQL involved is a little complex)

>>> Article.objects.exclude(publications=p2)

<QuerySet [<Article: Django lets you build web apps easily>]>

If we delete a Publication, its Articles won’t be able to access it:

>>> p1.delete()

>>> Publication.objects.all()

<QuerySet [<Publication: Highlights for Children>, <Publication: Science News>, <Publication:␣

,→Science Weekly>]>

>>> a1 = Article.objects.get(pk=1)

>>> a1.publications.all()

<QuerySet []>

If we delete an Article, its Publications won’t be able to access it:

>>> a2.delete()

>>> Article.objects.all()

<QuerySet [<Article: Django lets you build web apps easily>]>

>>> p2.article\_set.all()

<QuerySet []>

Adding via the ‘other’ end of an m2m:

>>> a4 = Article(headline="NASA finds intelligent life on Earth")

>>> a4.save()

>>> p2.article\_set.add(a4)

>>> p2.article\_set.all()

<QuerySet [<Article: NASA finds intelligent life on Earth>]>

>>> a4.publications.all()

<QuerySet [<Publication: Science News>]>

Adding via the other end using keywords:

>>> new\_article = p2.article\_set.create(headline="Oxygen-free diet works wonders")

>>> p2.article\_set.all()

<QuerySet [<Article: NASA finds intelligent life on Earth>, <Article: Oxygen-free diet works␣

,→wonders>]>

>>> a5 = p2.article\_set.all()[1]

>>> a5.publications.all()

<QuerySet [<Publication: Science News>]>

Removing Publication from an Article:

>>> a4.publications.remove(p2)

>>> p2.article\_set.all()

<QuerySet [<Article: Oxygen-free diet works wonders>]>

>>> a4.publications.all()

<QuerySet []>

And from the other end:

>>> p2.article\_set.remove(a5)

>>> p2.article\_set.all()

<QuerySet []>

>>> a5.publications.all()

<QuerySet []>

Relation sets can be set:

>>> a4.publications.all()

<QuerySet [<Publication: Science News>]>

>>> a4.publications.set([p3])

>>> a4.publications.all()

<QuerySet [<Publication: Science Weekly>]>

Relation sets can be cleared:

>>> p2.article\_set.clear()

>>> p2.article\_set.all()

<QuerySet []>

And you can clear from the other end:

>>> p2.article\_set.add(a4, a5)

>>> p2.article\_set.all()

<QuerySet [<Article: NASA finds intelligent life on Earth>, <Article: Oxygen-free diet works␣

,→wonders>]>

>>> a4.publications.all()

<QuerySet [<Publication: Science News>, <Publication: Science Weekly>]>

>>> a4.publications.clear()

>>> a4.publications.all()

<QuerySet []>

>>> p2.article\_set.all()

<QuerySet [<Article: Oxygen-free diet works wonders>]>

Recreate the Article and Publication we have deleted:

>>> p1 = Publication(title="The Python Journal")

>>> p1.save()

>>> a2 = Article(headline="NASA uses Python")

>>> a2.save()

>>> a2.publications.add(p1, p2, p3)

Bulk delete some Publications - references to deleted publications should go:

>>> Publication.objects.filter(title\_\_startswith="Science").delete()

>>> Publication.objects.all()

<QuerySet [<Publication: Highlights for Children>, <Publication: The Python Journal>]>

>>> Article.objects.all()

<QuerySet [<Article: Django lets you build web apps easily>, <Article: NASA finds intelligent life␣

,→on Earth>, <Article: NASA uses Python>, <Article: Oxygen-free diet works wonders>]>

>>> a2.publications.all()

<QuerySet [<Publication: The Python Journal>]>

Bulk delete some articles - references to deleted objects should go:

>>> q = Article.objects.filter(headline\_\_startswith="Django")

>>> print(q)

<QuerySet [<Article: Django lets you build web apps easily>]>

>>> q.delete()

After the delete(), the QuerySet cache needs to be cleared, and the referenced objects should be gone:

>>> print(q)

<QuerySet []>

>>> p1.article\_set.all()

<QuerySet [<Article: NASA uses Python>]>