One-to-one relationships

To define a one-to-one relationship, use `OneToOneField`.

In this example, a `Place` optionally can be a `Restaurant`:

```
from django.db import models


class Place(models.Model):
    name = models.CharField(max_length=50)
    address = models.CharField(max_length=80)

    def __str__(self):
        return f"{self.name} the place"


class Restaurant(models.Model):
    place = models.OneToOneField(
        Place,
        on_delete=models.CASCADE,
        primary_key=True,
    )
    serves_hot_dogs = models.BooleanField(default=False)
    serves_pizza = models.BooleanField(default=False)

    def __str__(self):
        return "%s the restaurant" % self.place.name


class Waiter(models.Model):
    restaurant = models.ForeignKey(Restaurant, on_delete=models.CASCADE)
    name = models.CharField(max_length=50)

    def __str__(self):
        return "%s the waiter at %s" % (self.name, self.restaurant)
```

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

Create a couple of Places:

```
>>> p1 = Place(name="Demon Dogs", address="944 W. Fullerton")
>>> p1.save()
>>> p2 = Place(name="Ace Hardware", address="1013 N. Ashland")
>>> p2.save()
```

Create a Restaurant. Pass the “parent” object as this object’s primary key:

```
>>> r = Restaurant(place=p1, serves_hot_dogs=True, serves_pizza=False)
>>> r.save()
```

A Restaurant can access its place:

```
>>> r.place
<Place: Demon Dogs the place>
```

A Place can access its restaurant, if available:

```
>>> p1.restaurant
<Restaurant: Demon Dogs the restaurant>
```

p2 doesn’t have an associated restaurant:

```
>>> from django.core.exceptions import ObjectDoesNotExist
>>> try:
...     p2.restaurant
... except ObjectDoesNotExist:
...     print("There is no restaurant here.")
...
There is no restaurant here.
```

You can also use `hasattr` to avoid the need for exception catching:

```
>>> hasattr(p2, "restaurant")
False
```

Set the place using assignment notation. Because place is the primary key on Restaurant, the save will create a new restaurant:

```
>>> r.place = p2
>>> r.save()
>>> p2.restaurant
<Restaurant: Ace Hardware the restaurant>
>>> r.place
<Place: Ace Hardware the place>
```

Set the place back again, using assignment in the reverse direction:

```
>>> p1.restaurant = r
>>> p1.restaurant
<Restaurant: Demon Dogs the restaurant>
```

Note that you must save an object before it can be assigned to a one-to-one relationship. For example, creating a `Restaurant` with unsaved `Place` raises `ValueError`:

```
>>> p3 = Place(name="Demon Dogs", address="944 W. Fullerton")
>>> Restaurant.objects.create(place=p3, serves_hot_dogs=True, serves_pizza=False)
Traceback (most recent call last):
...
ValueError: save() prohibited to prevent data loss due to unsaved related object 'place'.
```

Restaurant.objects.all() returns the Restaurants, not the Places. Note that there are two restaurants - Ace Hardware the Restaurant was created in the call to r.place = p2:

```
>>> Restaurant.objects.all()
<QuerySet [<Restaurant: Demon Dogs the restaurant>, <Restaurant: Ace Hardware the restaurant>]>
```

Place.objects.all() returns all Places, regardless of whether they have Restaurants:

```
>>> Place.objects.order_by("name")
<QuerySet [<Place: Ace Hardware the place>, <Place: Demon Dogs the place>]>
```

You can query the models using lookups across relationships:

```
>>> Restaurant.objects.get(place=p1)
<Restaurant: Demon Dogs the restaurant>
>>> Restaurant.objects.get(place__pk=1)
<Restaurant: Demon Dogs the restaurant>
>>> Restaurant.objects.filter(place__name__startswith="Demon")
<QuerySet [<Restaurant: Demon Dogs the restaurant>]>
>>> Restaurant.objects.exclude(place__address__contains="Ashland")
<QuerySet [<Restaurant: Demon Dogs the restaurant>]>
```

This also works in reverse:

```
>>> Place.objects.get(pk=1)
<Place: Demon Dogs the place>
>>> Place.objects.get(restaurant__place=p1)
<Place: Demon Dogs the place>
>>> Place.objects.get(restaurant=r)
<Place: Demon Dogs the place>
>>> Place.objects.get(restaurant__place__name__startswith="Demon")
<Place: Demon Dogs the place>
```

If you delete a place, its restaurant will be deleted (assuming that the `OneToOneField` was defined with `on_delete` set to `CASCADE`, which is the default):

```
>>> p2.delete()
(2, {'one_to_one.Restaurant': 1, 'one_to_one.Place': 1})
>>> Restaurant.objects.all()
<QuerySet [<Restaurant: Demon Dogs the restaurant>]>
```

Add a Waiter to the Restaurant:

```
>>> w = r.waiter_set.create(name="Joe")
>>> w
<Waiter: Joe the waiter at Demon Dogs the restaurant>
```

Query the waiters:

```
>>> Waiter.objects.filter(restaurant__place=p1)
<QuerySet [<Waiter: Joe the waiter at Demon Dogs the restaurant>]>
>>> Waiter.objects.filter(restaurant__place__name__startswith="Demon")
<QuerySet [<Waiter: Joe the waiter at Demon Dogs the restaurant>]>
```