Goga Patarkatsishvili Follow
Dec 13 · 6 min read


Credits—author : Jez Timms; illustrator : Tatuna Gverdtsiteli

A cheat sheet for Django ORM relationships—version 2.1

I want to start this story with saying thanks to Mahmoud Zalt, who published a really useful article "Eloquent Relationships Cheat Sheet" about a year ago and also gave me permission to use the same structure/images/examples in my article.

. . .

- **One to One Relationship**

- **One to Many Relationship**
- **Many to Many Relationship**
- **Polymorphic One to Many Relationship—Generic Relations**
- **Polymorphic One to Many Relationship—django-polymorphic package**
- **Polymorphic Many to Many Relationship—django-polymorphic package**

. . .

# One to One Relationship

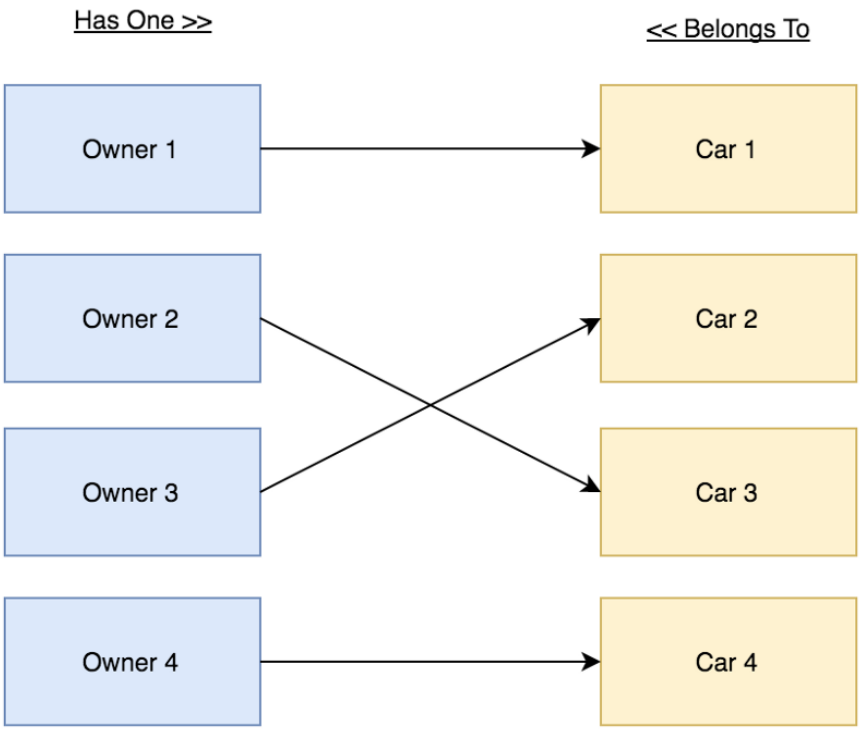
### Demo details:

In this demo we have 2 models (**Owner** and **Car**), and 2 tables (**owners** and **cars**).

### Business Rules:

The **Owner** can own one **Car**.
The **Car** can be owned by one **Owner**.

### Relations Diagram:




## Relationship Details:

The **Cars** table should store the **Owner ID**.

## Models:

```
class Owner(models.Model):

    #...
    name = models.CharField(max_length=255)


class Car(models.Model):

    #...
    name = models.CharField(max_length=255)
    owner = models.OneToOneField(
        Owner,
        on_delete=models.CASCADE,
        related_name='car'
    )
```

## Store Records:

```
car = Car.objects.get(id=1)
owner  = Owner.objects.get(id=1)

# Create relation between Owner and Car.

owner.car = car
owner.car.save()

# Create relation between Car and Owner.

car.owner = owner
car.save()
```

### Retrieve Records:

```
# Get Owner Car

owner.car


# Get Car Owner

car.owner;
```

. . .

## One to Many Relationship

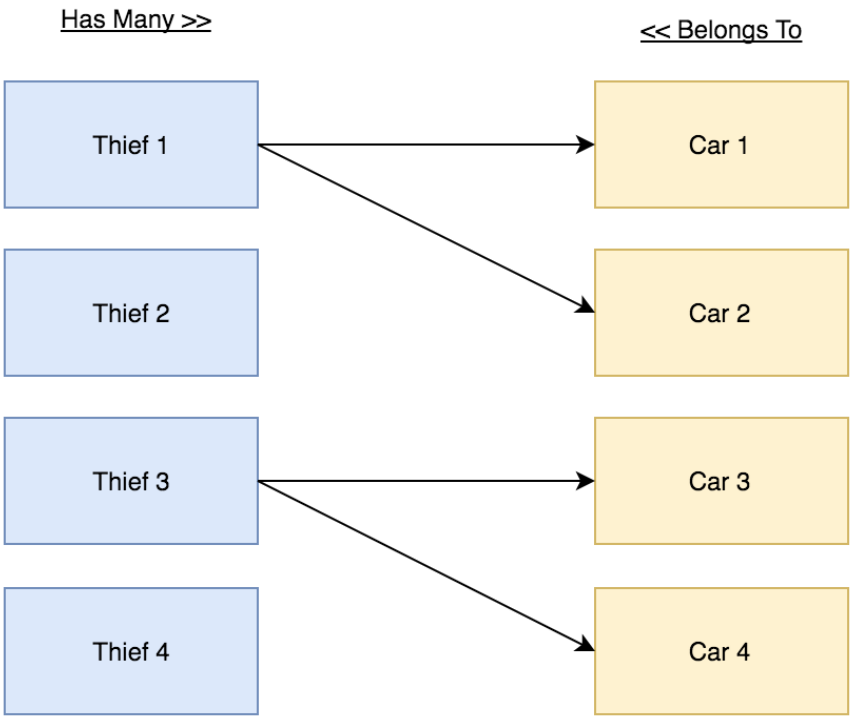
### Demo details:

In this demo we have 2 models (**Thief** and **Car**), and 2 tables (**thieves** and **cars**).

### Business Rules:

The **Thief** can steal many **Cars**.
The **Car** can be stolen by one **Thief**.

### Relations Diagram:




## Relationship Details:

The **Cars** table should store the **Thief ID**.

## Models:

```
class Thief(models.Model):

    # ...
    name = models.CharField(max_length=255)


class Car(models.Model):

    # ...
    name = models.CharField(max_length=255)
    thief = models.ForeignKey(
        Thief,
        on_delete=models.CASCADE,
        related_name='cars'
    )
```

## Store Records:

```
thief  = Thief.objects.get(id=1)
car1 = Car.objects.get(id=1)
...

# Create relation between Thief and Car.

thief.cars.add(car1,car2, car3)

# Create relation between Car and Thief.

car.thief = thief
car.save()

# When we creating new car :
car = Car(name = 'test name', thief=thief)
car.save()
```

### Retrieve Records:

```
# Get Thief Car

thief.cars.all()

# Get Car Thief

car.thief
```

. . .

## Many to Many Relationship

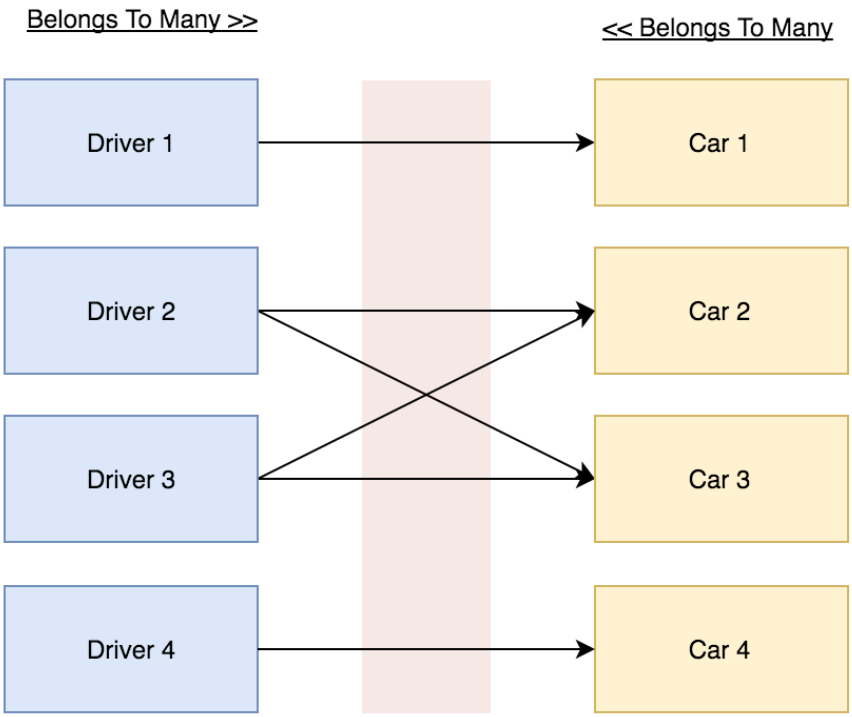
### Demo details:

In this demo we have 2 models (**Driver** and **Car),** and 3 tables (**drivers**, **cars** and a pivot table named **car_drivers**).

### Business Rules:

The **Driver** can drive many **Cars**.
The **Car** can be driven by many **Drivers**.

## Relations Diagram:




## Relationship Details:

The **Pivot** table “car_driver” should store the **Driver ID** and the **Car ID**.

## Models:

```
class Driver(models.Model):

    # ...
    name = models.CharField(max_length=255)


class Car(models.Model):

    # ...
    name = models.CharField(max_length=255)
    drivers = models.ManyToManyField(
        Driver,
        related_name='cars'
    )
```

## Store Records:

```
# Create relation between Driver and Car.


driver  = Driver.objects.get(id=1)
car1 = Car.objects.get(id=1)
car2 = Car.objects.get(id=2)


driver.cars.add(car1,car2)


# Create relation between Car and Driver.
car = Car.objects.get(id=1)
driver1  = Driver.objects.get(id=2)
driver2  = Driver.objects.get(id=3)

car.drivers.add(driver1, driver2)
```

### Retrieve Records:

```
# Get Driver Car


driver.cars.all()


# Get Car Drivers


car.drivers.all()
```

. . .

## Polymorphic One to Many Relationship

**With Django's Generic Relations**

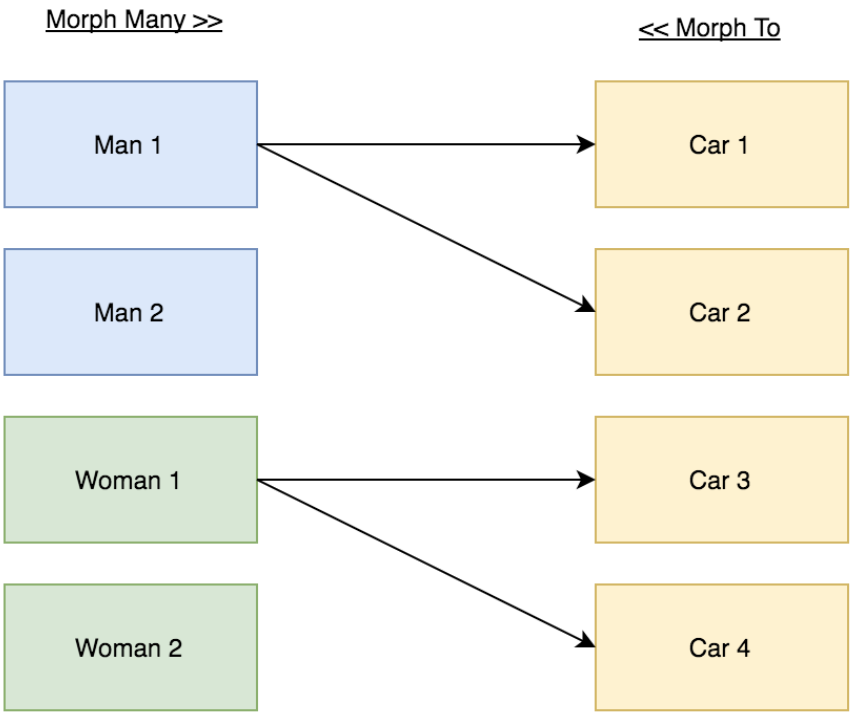
### Demo details:

In this demo we have 3 models (**Man, Woman** and **Car),** and 3 tables (**men**, **women** and **cars**).

### Business Rules:

The **Man** (buyer) can buy many **Cars**.
The **Woman** (buyer) can buy many **Cars**.
The **Car** can be bought by one buyer (**Man** or **Woman**).

## Relations Diagram:




## Relationship Details:

The **Car** table should store the **Buyer ID** and the Buyer table should store the relation between **ID** and **Type**.

## Models:

```
from django.contrib.contenttypes.fields import
GenericForeignKey
from django.contrib.contenttypes.fields import
GenericRelation
from django.contrib.contenttypes.models import ContentType


class Car(models.Model):


    # ...
    name = models.CharField(max_length=255)


    content_type = models.ForeignKey(ContentType,
on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    content_object = GenericForeignKey()
```

```
class Woman(models.Model):

    # ...
    name = models.CharField(max_length=255)
    cars = GenericRelation(Car)

class Man(models.Model):

    # ...
    name = models.CharField(max_length=255)
    cars = GenericRelation(Car)
```

## Store Records:

```
man = Man.objects.get(id=1)
woman = Woman.objects.get(id=1)

# Create relation between buyer (Man/Woman) and Car.

car = Car.objects.get(id=1)
woman.cars.add(car)

# Create relation between Car and buyer (Men/Women).

man = Man.objects.get(id=1)
woman = Woman.objects.get(id=1)

c = Car(name = 'test name',content_object=man)
c.save()

c = Car(name = 'test name',content_object=woman)
c.save()
```

## Retrieve Records:

```
# Get buyer (Man/Woman) Cars

man.cars.all()
woman.cars.all()

# Get Car buyer (Man and Woman)
```

```
car.content_object
```

. . .

# Polymorphic One to Many Relationship

**With package—django-polymorphic.**

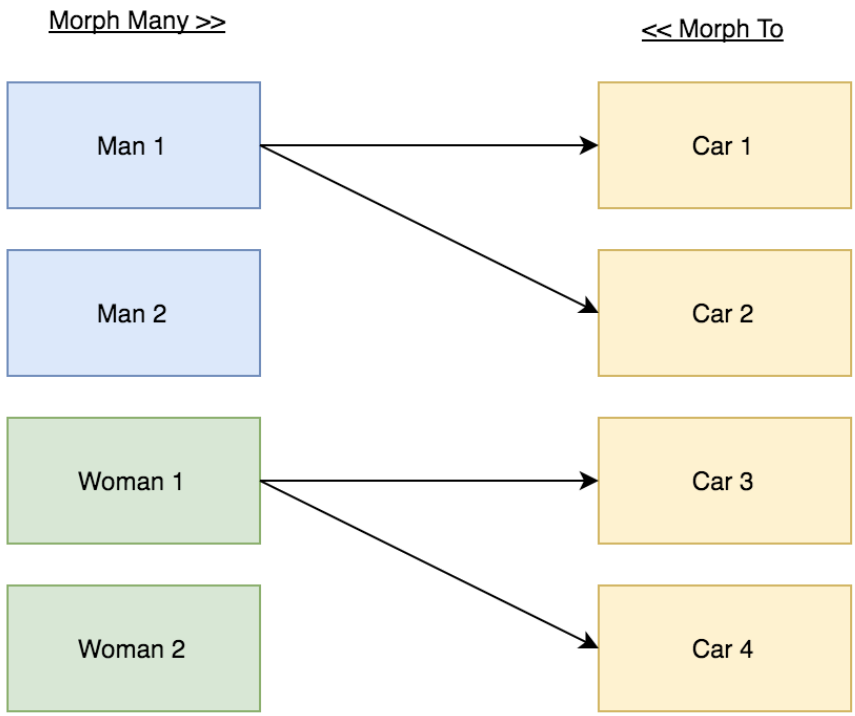
## Demo details:

In this demo we have 4 models (**Buyer**, **Man, Woman** and **Car),** and 4 tables (**buyer**, **men**, **women** and **cars**).

## Business Rules:

The **Man** (buyer) can buy many **Cars**.
The **Woman** (buyer) can buy many **Cars**.
The **Car** can be bought by one buyer (**Man** or **Woman**).

## Relations Diagram:




## Relationship Details:

The **Car** table should store the **Buyer ID** and the Buyer table should store the relation between **ID** and **Type**.

## Models:

```
from polymorphic.models import PolymorphicModel


class Buyer(PolymorphicModel):
    pass

class Woman(Buyer):

    # ...
    name = models.CharField(max_length=255)



class Man(Buyer):

    # ...
    name = models.CharField(max_length=255)



class Car(models.Model):

    # Fields

    name = models.CharField(max_length=255)
    buyer = models.ForeignKey(
        Buyer,
        on_delete=models.CASCADE,
        related_name='cars'
    )
```

## Store Records:

```
man = Buyer.objects.get(id=1) # or Man.objecets.get(id=1)
woman = Buyer.objects.get(id=2) # or
Woman.objecets.get(id=2)


# Create relation between buyer (Man/Woman) and Car.


man.cars.add(car1, car2)


woman.cars.add(car1, car2)
```

```
# Create relation between Car and buyer (Men/Women).

c = Car(name = 'test name', buyer = man)
c.save()

c = Car(name = 'test name', buyer = woman)
c.save()
```

### Retrieve Records:

```
# Get buyer (Man/Woman) Cars

man.cars.all()
woman.cars.all()

# Get Car buyer (Man and Woman)

car.buyer
```

. . .

## Polymorphic Many to Many Relationship

**With package—django-polymorphic.**

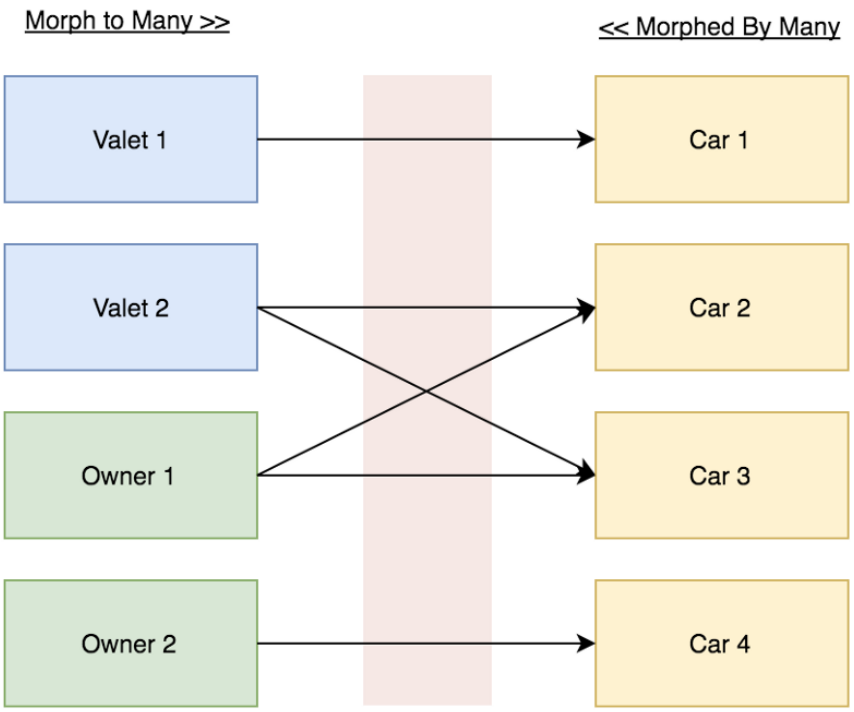
### Demo details:

In this demo we have 3 models (**Valet, Owner** and **Car),** and 4 tables (**valets**, **owners, cars** and **drivers**).

### Business Rules:

The **Valet** (driver) can drive many **Cars**.
The **Owner** (driver) can drive many **Cars**.
The **Car** can be driven by many drivers (**Valet** or/and **Owner**).

### Relations Diagram:




## Relationship Details:

The **Pivot** table "drivers" should store the **Driver ID**, **Driver Type** and the **CarID**.
*"driver" is a name given to a group of models (Valet and Owner). And it's not limited to two. The driver type is the real name of the model.*

## Models:

```
from polymorphic.models import PolymorphicModel

class Driver(PolymorphicModel):
    pass


class Owner(Driver):

    # ...
    name = models.CharField(max_length=255)


class Valet(Driver):

    # ...
    name = models.CharField(max_length=255)
```

```
class Car(models.Model):

    # ...

    name = models.CharField(max_length=255)
    drivers = models.ManyToManyField(
        Driver,
        related_name='cars'
    )
```

## Store Records:

```
# Create relation between driver (Valet/Owner) and Car.

owner.cars.add(car1, car2)

# Create relation between Car and driver (Valet/Owner).

car.drivers.add(owner, valet)
```

## Retrieve Records:

```
# Get driver (Valet/Owner) Cars

valet.cars.all()
owner.cars.all()

# Get Car drivers (Valet and Owner)

car.drivers.all()
car1.drivers.instance_of(Valet)
car1.drivers.instance_of(Owner)
```

I also highly recommend to read the article, that describes polymorphic many to many relationships with another django package (django-gm2m).

. . .

Thanks for reading and stay in touch.