1.Consultas cuando tenemos relaciones onetoone

Recordamos esta consulta que devuelve un tipo resulset

>>> from core.models import Book

>>> Book.objects.all()

<QuerySet [<Book: 222 La gran princesa>, <Book: 636 El principito>, <Book: 98652 El gran misterio>]>

>>> Book.objects.all().first()

<Book: 222 La gran princesa>

>>> book_1=Book.objects.all().first()

>>> book_1

<Book: 222 La gran princesa>

>>> book_1.title

'La gran princesa'

>>> book_1.cod

'222'

Si queremos acceder al valor del autor correspondiente a ese libro:

>>> book_1.author

<Author: Pepito Perez 25>

Esta consulta me devuelve un objeto, no una cadena de texto como me ha devuelto el title y el cod. De hecho me devuelve también el 25 y el apellido porque tenemos puesta dentro de la clase Author lo la siguiente definición:

```
def __str__(self):
      return f'{self.name} {self.last_name} {self.age}'
```

La cuestión es que para obtener directamente los valores de los campos, como la relación es onetone, con el operador . podemos acceder al resto de campos de la tabla autor.

>>> book_1.author.age

25

```
>>> book_1.author.name
'Pepito'
>>> book_1.author.id
1
```

Si quisiéramos obtener todos los libros de ese autor con id=1 (sólo hay un libro, pues tenemos la relación uno a uno)

```
>>> Book.objects.filter(author_id=1)
<QuerySet [<Book: 222 La gran princesa>]>
```

También puede encontrarse esta misma consulta de esta forma:

```
>>> Book.objects.filter(author__id=1)
<QuerySet [<Book: 222 La gran princesa>]>
```

Amabas formas realizan la misma consulta. Vamos a igualar el resultado de cada forma a una variable para ver la consulta que realizan:

```
>>> book_1=Book.objects.filter(author__id=1)
>>> book_2=Book.objects.filter(author_id=1)
>>> print(book_2.query)
SELECT "core_book"."id", "core_book"."title", "core_book"."cod", "core_book"."author_id" FROM "core_book" WHERE "core_book"."author_id" = 1
>>> print(book_1.query)
SELECT "core_book"."id", "core_book"."title", "core_book"."cod", "core_book"."author_id" FROM "core_book" WHERE "core_book"."author_id" = 1
```

En esta caso ocurre así porque la relación es onetone.

2.Consulta cuando tenemos relaciones uno a muchos

Vamos a cambiar ahora la relación y ponerla uno a muchos.

```
author=models.ForeignKey(Author,null=True,blank=True,on_delete=models.CASCADE)
```

Recordar que después de modificar el modelo hay que ejecutar:

python manage.py makemigrations
python manage.py mígrate

Ahora, tras cambiar la relación uno a muchos, me deja insertar más de un libro a un mismo autor.
Volvemos a consultar lo mismo que en la relación anterior para ver si ha cambiado algo:

```
>>> book_3=Book.objects.all().first()
>>> book_3
<Book: 222 La gran princesa>
>>> book_3.author
<Author: Pepito Perez 25>
>>> book_3.id
1
>>> Book.objects.filter(author_id=1)
<QuerySet [<Book: 222 La gran princesa>, <Book: 3444 El pastor alemán>]>
```

Realmente la estructura de las consultas no van cambiando.

3.Consultas cuando tenemos relaciones muchos a muchos

Cambiamos en el model.py la relación entra las tablas. En el modelo Book quitamos la relación uno a muchos para sustituirla por:

```
author=models.ManyToManyField(Author)
```

Igualmente ejecutamos las sentencias necesarias para llevarlas a la BD

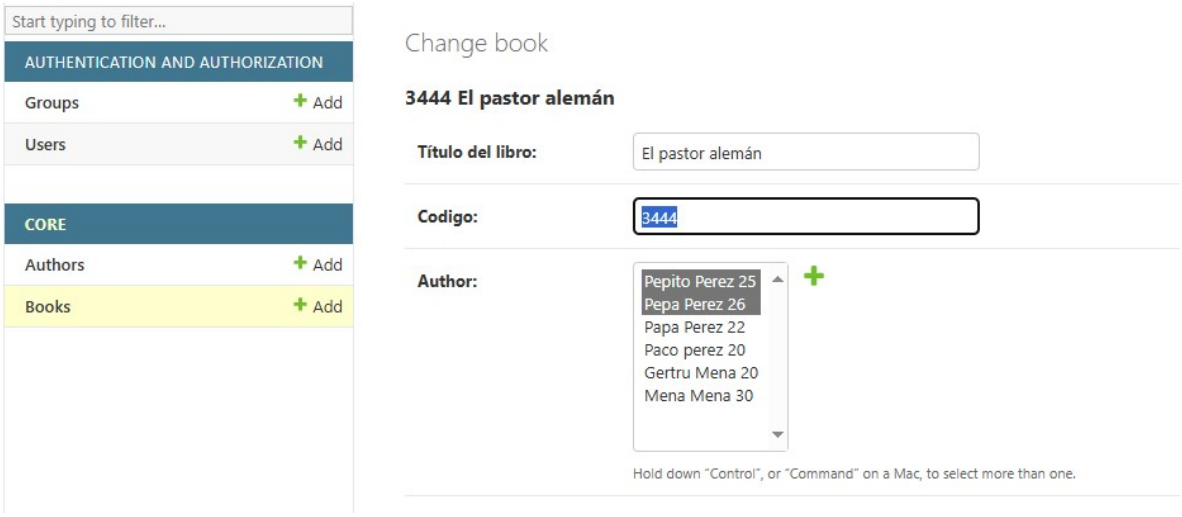
python manage.py makemigrations
python manage.py mígrate

Ahora cuando agregamos un libro, nos va a aparece la posibilidad de agregar más de un autor para ese libro. Visualizarlo desde /admin.
A nivel visual ha cambiado así como a nivel estructural.

Vamos a comprobar cómo consultamos :

```
>>> book_4=Book.objects.filter(cod=3444)
>>> book_4
<QuerySet [<Book: 3444 El pastor alemán>]>
>>> book_4=Book.objects.filter(cod=3444).first()
>>> book_4.author
<django.db.models.fields.related_descriptors.create_forward_many_to_many_manager.<locals>.ManyRelatedManager object at 0x000001CE5D477EF0>
>>> book_4
<Book: 3444 El pastor alemán>
```

Cuando tenemos este tipo de relaciones, se crea internamente otra nueva tabla que nos obliga a tener otra estructura para realizar las consultas.
Por ejemplo para poder acceder a los autores del libro 3444



La forma de consultar que habíamos visto antes, book_4.author, no es posible, con lo que hay que añadirle el all(), pues en sí, el atributo author es otro objeto el cual tiene el método all(). Esto ocurre porque el tipo de relación, como he comentado antes, es de muchos a muchos, y la estructura interna de la BD es algo diferente.

```
>>> book_4.author.all()
<QuerySet [<Author: Pepito Perez 25>, <Author: Pepa Perez 26>]>
>>>
```

Además del método all(), hay otros más que vamos a ir utilizando más adelante.