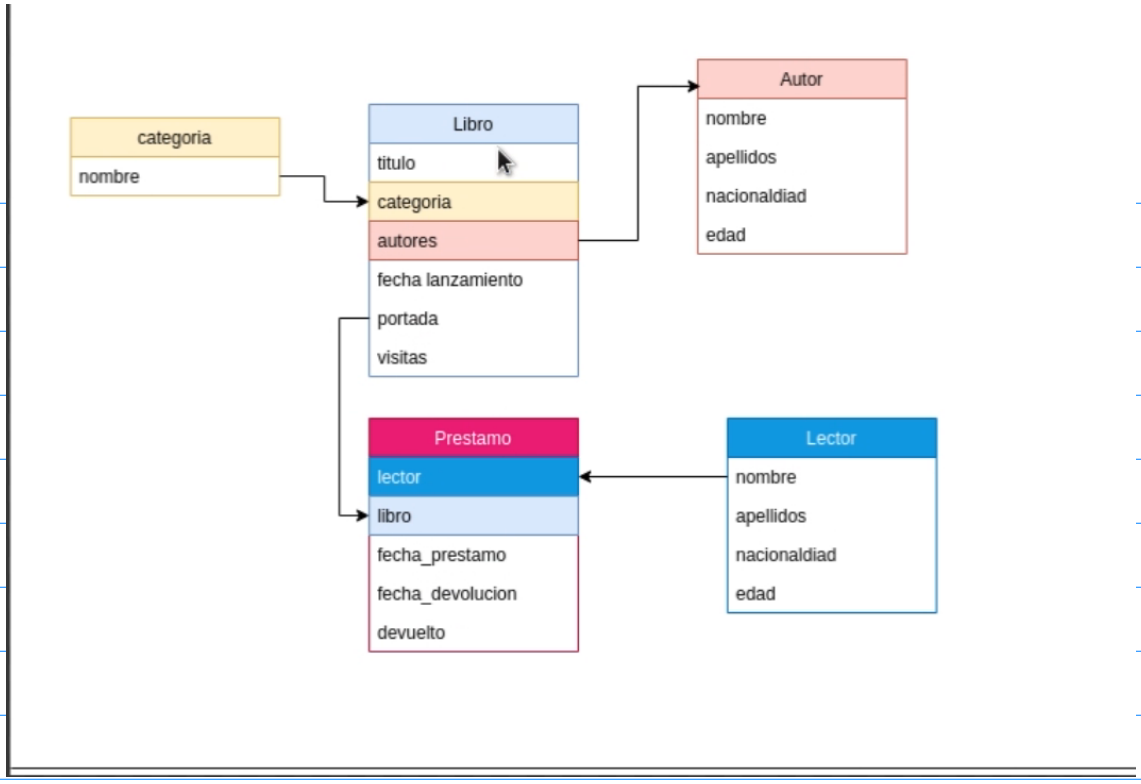
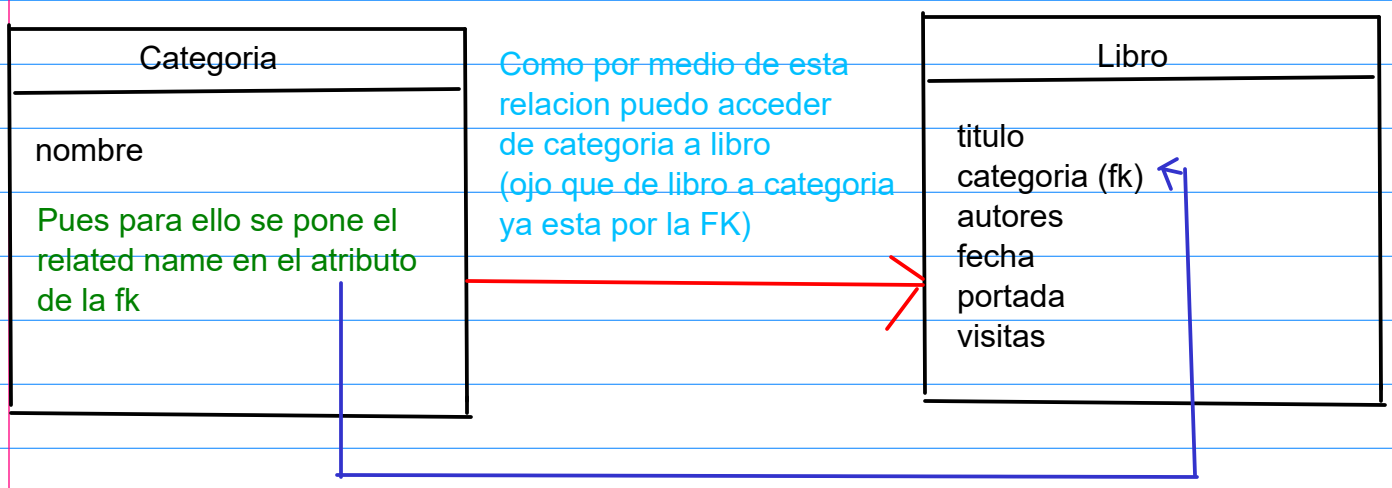


Requerimiento:
Listar todas las categorias de un autor
Los modelos de categoria y autor no estan relacionados directamente, entonces por medio de la relación que existe entre libro y categoría se pueden relacionar el autor con la categoria.

Para ello usamos el related_name que hace referencia a la relación inversa de la clave foranea, en pocas palabras nos sirve para acceder desde categoría a libro

Si se quiere acceder a libro desde categoria se puede usar este related_name



Y por medio de ese related name se puede acceder a Libro desde Categoria es decir al libro asociado a esa categoria

categoria_libro hace referencia a la relacion de libro con su fk de categoria respectiva y una vez unidos estos modelos accedo al otro atributo de la clase libro para hacer la pregunta... categoria_libro_autores_id = autor

Resumiendo el modelo libro le jala a toda la clase Categoria para al momento de hacer la pregunta retornar la categoria del libro

Clase diferencia entre Annotate y Aggregate

Aggregate: Devuelve un diccionario de valores agregados (promedios, sumas, etc.) calculados sobre el QuerySet. Cada argumento para aggregate()especifica un valor que se incluirá en el diccionario que se devuelve.

Se plantea contar cuantas veces fue prestado un libro y se intentará hacerlo usando el aggregate

De igual forma en el modelo prestamo se añade el related_name para poder acceder a la cantidad de relaciones y poder contarlas.
OjO que el Aggregate me retorna un DICCIONARIO a diferencia del Annotate que me retorna un QuerySet

¿En qué caso uso el Annotate y el Aggregate?

Para el caso del conteo que queremos en una consulta o en un queryset, obligatoriamente utilizaremos el annotate, pero si solamente necesitamos una operación aritmética para encontrar un valor, vamos a utilizar el aggregate

Se plantea que de un determinado libro del modelo libro, por ejemplo del libro A se calcule el promedio de la edad de los lectores que se prestan ese libro A

Para solucionar este problema se podria hacerlo en libro, hacer un filtro con el lado inverso y desde prestamo llegar a libro pero es mas sencillo hacer el procedimiento dentro de la tabla prestamo que ya está vinculada directamente tanto a libro como a lector, además de que es más óptimo

Entonces de igual forma se creó un manager para el modelo prestamo y ahí se hizo uso del aggregate, además al retornar un diccionario ésta funcion pues nos permite seguir añadiendo más operaciones

Values, Group_By en al ORM Django

Se plantea como requerimiento que se cuente cuantas veces se ha prestado cada libro dentro de mi tabla prestamo, por ejemplo Dracula se ha prestado 3 veces porque se repite tres veces

El annotate para que siga acumulando cada que encuentra un libro necesita algo que le indique en base a que quiere agrupar esto

En el caso del listar_categoria_libros el annotate lo agrupa automaticamente en base a un ID del mismo modelo, es por ello que aquí sí funcionaba

Pero ahora voy a listar o estoy haciendo en base a prestamos y al hacerle en base al ID pues considera diferente registros por ejemplo

Para el prestamo 1 tengo el libro 1
pero para el prestamo 2 tambien tengo el libro 1
y tambien para el prestamo 3 tengo el libro 1 entonces lo esta considerando como diferentes

*Es por ello que para que en este caso el annotate funcione pues necesita de un indicador o identificador que le indique a él en base a qué va a agrupar y contar como referencia los registros que le estamos indicando que en este caso son libros

Ojo que al usar el values ya no se devuelve un queryset sino una lista de diccionarios
¿Con qué valores?
Pues por el valor por el cual nosotros estamos agrupando y la respuesta , que serían los
(value1, valorResp) = {'libro': 9, 'num_prestados': 1} 1

Sin embargo, no siempre vamos a requerir solo de un valor para hacer el group by por ello simplemente se añade dentro del values el otro atributo

Trigam con Postgres Django

Lo que se plantea es buscar libros a partir de una palabra clave

Para ello usaremos una extensión que ya nos ofrece Django exclusiva para Postgresql que se llama Trigram similarity

* Siempre necesita de 3 caracteres
* Por defecto Postgres lo tiene desactivado entonces se debe activar, despues debemos indicar en que tabla y atributo se trabajará la triagramación
* En Windows ya viene instalado por defecto

Ya en la terminal de postgresql necesitamos indicarle a la BD que vamos a utilizar la triagramacion
CREATE EXTENSION pg_trgm;
Ahora indicaremos que se use la triagramacion en la tabla y en el atributo

CREATE INDEX (nombreAplicacion_nombreModelo_idx ON (nombreapp_nombre_modelo)
USING GIN (nombreAtributo gin_trgm_ops);

CREATE INDEX libro_titulo_idx ON libro_libro USING GIN (titulo gin_trgm_ops);

Modelos Abstract - Class Meta

Herencia <= class Empleados(Persona): la clase empleados hereda de Persona

Abstract <= Indicar a un modelo que se cree unicamente como modelo pero no en la BD

Para ello se pone el atributo abstract = True

Copia de Seguridad de Base de Datos en Postgres

OjO añadir a las variables de entorno los bin de Postgres

https://neunapp.com/contenido/copia-de-seguridad-de-una-base-de-datos-postgres-linux-ubuntu-22063

```
C:\Users\Adrian\Documents\GIIATA\Django\Biblioteca\resguardo>pg_dump -U adrian dbbiblioteca > prueba.sql
Contraseña:
```

<= Así en Windows

```
postgres=# drop database dbbiblioteca;
DROP DATABASE
postgres=# create database dbbiblioteca;
CREATE DATABASE
postgres=# ALTER USER adrian with password 'django';
ALTER ROLE
postgres=# grant all privileges on database dbbiblioteca to adrian;
GRANT
postgres=#
```

```
C:\Users\Adrian\Documents\GIIATA\Django\Biblioteca\resguardo>psql -h localhost -p 5432 -U adrian -d dbbiblioteca < prueba.sql
Contraseña para usuario adrian:
SET
```

Veremos la forma más simple de crear una copia de seguridad de nuestra base de datos **Postgres**. Si tomaste los curso de **Django** que tenemos en Udemy y en esta web, seguro en algún momento tuviste la necesidad de que crearas una copia de seguridad de tu base de datos y que también re-establecieras una base de datos en base a una copia de seguridad. Para ello veamos cómo es que lo hacemos:

Pasos a seguir:

Primero definitivamente necesitas tener acceso a tu usuario Postgres.

El nombre de tu BD o tu BD creada.

El usuario sudo o autenticado de tu terminal ubuntu.

En caso de windows ejecuta PSQshell como administrador.

Dirígete desde la terminal a una carpeta que tenga permisos de usuario, si no crea una carpeta y dale permisos con el comando:

```
chmod 777 “nombre_carpeta”
```

Accede a tu usuario Postgres desde la terminal de comandos:

Accede a tu usuario Postgres desde la terminal de comandos:

```
su postgres
```

Con el usuario postgres activo y dentro de la carpeta con permisos ejecuta:

```
pg_dump “nombre_bd” > “nombre_archivo_backup”
```

Con ello si te fijas la carpeta donde ejecutaste el comando, se habrá creado el archivo con el nombre especificado. Ese archivo ya contiene tu **backup** de base de datos.

Re-Establecer una copia de base de datos a una base de datos existente.

Ahora si ya tiene una **BD creada para tu proyecto en Django** y necesitas utilizar tu copia de seguridad para no estar ingresando datos de nuevo, puedes re-establecer la copia de seguridad. Este caso también sirve si surge algún problema en tu proyecto y necesitas re-establecer una copia de seguridad:

vuelve a ingresar con el usuario Postgres:

vuelve a ingresar con el usuario Postgres:

```
su postgres
```

Ahora ve donde tienes el archivo de backup desde la terminal, sin salir del usuario

Postgres, y ejecuta:

```
psql “nombre_db” < “nombre_archivo_copia”
```

Esto debe cargar varias líneas de comando ejecutándose, de lo contrario algo anda mal, y posiblemente sean los permisos de usuario. Así que preocúpate de ello.

Ahora si ejecutas tu proyecto **Django** seguro que te mandara un error de migraciones en la terminal. Por tanto para quitar ese error, necesitas hacer una **falsa migración.**

Para ello primero elimina todas las migraciones de tus aplicaciones, en la carpeta migrations, luego de ello ve a la terminal y con tu entorno activado y a la altura del manage.py ejecuta:

```
python manage.py makemigrations
```

Luego de ello una falsa migración

Para ello primero elimina todas las migraciones de tus aplicaciones, en la carpeta migrations, luego de ello ve a la terminal y con tu entorno activado y a la altura del manage.py ejecuta:

```
python manage.py makemigrations
```

Luego de ello una falsa migración

```
python manage.py migrate --fake
```

Con ello todo andará bien y estarás cargando los datos del **backup**. Recuerda que tenemos un video de esto en nuestro canal de youtube como **Neunapp**.

Clase Funcion get_or_create Caso Practico

Se plantea como ejercicio el caso en el que un lector hace un prestamo y aun no devuelve el libro pero a pesar de ello vuelve a pedir prestado el mismo libro pero sin haber devuelto el ejemplar

¿como evitar que se registre en la BD? TENEMOS QUE VALIDAR

Si bien la validacion se la puede hacer con el filter existe otra función dentro de la ORM de django que podría ayudarnos en este caso y ésa función se llama get_or_create

funciona así
si el registro existe nos lo devuelve y si no pues lo crea

Varios registros en un modelo - caso practico

Qué pasará si un usuario dentro de la biblioteca pide 3 libros, pues hasta ahora como está es que vaya haciendo libro por libro pero mas elegante sería poder seleccionar todos los libros que quiera por medio de un check

Para ello se ha creado otro formulario y haciendo uso de la funcion init se ha obtenido todos los libros y junto con widget se ha usado el checkbox de opcion multiple pero en la siguiente clase se hara el proceso de guardado

Bulk- Create para varios Registros

Lo que se hace es crear una lista de objetos y luego se manda la lista en el bulk-create o update

PENDIENTE!! Hacer el update_or_create

Triggers o Disparadores

Que pasaria si al momento de guardar un libro yo quiero que el libro pase por determinado proceso para saber si el libro es del interes del lector

En pocas palabras cada que se haga un save o un delete pues se necesita un disparador que registre posiblemente en otra tabla quien esta haciendo una operacion, para que cuando necesitemos verificar se tenga un registro de qué ha estado pasando

Son bien importantes los disparadores debido a las auditorias

Signal Post Save

Las imagenes no siempre se guardan con el mismo tamaño entonces el ejercicio consiste en que una vez registrado un libro pues optimice la imagen de manera que no pese mucho

Se conecta la funcion creada con el modelo sobre el cual quiero trabajar

TAREA EL PRE SAVE

Signal Post Delete

Se plantea que cuando ya se elimine un prestamo pues el stock vuelva a aumentarse

Se realizó completamente el signal, sin embargo es importante mencionar que usualmente se encuentran al final del modelo pero existen mucho signals en los modelos por ello se recomienda crear un archivo aparte donde van a estar todos nuestro signals para finalmente solo llamarlos.