Tema 4 - URLs, Views y QuerySets

Jorge Agustín Barón Abad

Vistas(Views)

Las vistas es dónde vamos a incluir el código que queremos que se ejecute y que indicará la página HTML que queremos que se muestre al cliente

Estas vistas son como funciones que se crean dentro del archivo **views.py.** Para acceder a ellas tenemos que hacerlo a través de URls.

```
from django.shortcuts import render

# Create your views here.
def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

Urls

Para acceder a las vistas debemos acceder a partir de las urls que indicamos en el navegador.

Para incluir URLs debemos crear el archivo **urls.py** dentro de nuestra aplicación. Y a continuación cargar dichas Urls en **mysite/urls.py**

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('blog.urls')),
]
```

mysite/urls.py

```python
from django.urls import path
from .import views

urlpatterns (module) views
    path('',views.post_list,name='post_list')
]
```

blog/urls.py

URLs

A continuación veremos diferentes ejemplos de URLs y cómo definirlas en el archivo **biblioteca/urls.py**

Lo primero que vamos a hacer es crear el archivo **urls.py** dentro del proyecto de biblioteca.

Ahora en el archivo **mysite/urls** debemos modificarlo de la siguiente forma.

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include("biblioteca.urls"))
]
```

Usamos este archivo porque es el que está definido en la variable **ROOT_URLCONF** de **mysite/settings.py**

URLs

Ahora debemos añadir las urls en el archivo **biblioteca/urls.py.** Por ejemplo vamos a añadir una url, que sirve para acceder a una página inicial donde podremos acceder a diferentes urls.

```
from django.urls import path
from .import views

urlpatterns = [
    path('',views.index,name='index')
]
```

Ahora debemos crear la vista correspondiente y la template que mostrará esa vista.

View y Template

En el archivo **biblioteca/views.py** creamos la vista correspondiente:

```python
def index(request):
    return render(request, 'index.html')
```

Ahora creamos la carpeta templates dentro de la carpeta de nuestra aplicación

Y dentro de esa carpeta creamos el archivo index.html con el contenido de la imagen

```html
<html>
    <head></head>
    <body>
    <h1>BIBLIOTECA ONLINE</h1>

    <h2>Listado de enlaces</h2>
    <ul>
        <li></li>
    </ul>
    </body>
</html>
```

URLs

Ahora vamos a crear las siguientes URLs :

- Una url que me muestre todos los libros y sus datos, incluido los relacionados
- Una url que me muestre información sobre cada libro
- Una url que me muestre los libros de un año y mes concreto
- Una url que me muestre los libros que tienen el idioma del libro o español ordenados por fecha de publicación.
- Una url que me muestre los libros de una biblioteca que contenga un texto en concreto.
- Una url que me muestre el último cliente que se llevó un libro en concreto
- Una url que muestre los libros que nunca han sido prestados.
- Una url que muestre una página con una Biblioteca y sus libros
- Una url que muestre los libros que contienen el título del libro en la descripción
- Una url que muestra la media, máximo y mínimo de puntos de todos los clientes de la Biblioteca

URL, View y QuerySet

Una url que me muestre todos los libros:

- Primero incluimos el siguiente código en **biblioteca/urls.py**

```
path('libros/listar',views.listar_libros,name='lista_libros'),
```

Vista

Ahora creamos la vista en **biblioteca/views.py:**

```
def listar_libros(request):
    libros = Libro.objects.select_related("biblioteca").prefetch_related("autores")
    libros = libros.all()
    return render(request, 'libro/lista.html',{"libros_mostrar":libros})
```

Es un poco distinto al ejemplo del tutorial, ¿verdad?¿Qué destacamos?

- **select_related:** Sirve para obtener los datos de las relaciones **OneToOne**, **ManytoOne**
- **prefetch_related:** Sirve para obtener los datos de las relaciones **ManytoMany**
- Debemos especificar el nombre del atributo en el modelo
- **Estas funciones se traducen a realizar un INNER JOIN en MYSQL, unimos varias tablas, para ejecutar sólo una SQL, en vez de usar una cada vez que la necesito. De esta forma mejoramos la eficiencia de nuestro código y aplicación**
- Pueden incluirse varios modelos separados por **“,”**

Traducción a SQL

A continuación os indicaré la traducción en SQL de las QuerySet que estamos haciendo. Para ello usaremos una función de las QuerySets que nos permiten ejecutar SQLs. **raw()**:

```
libros = Libro.objects.select_related("biblioteca").prefetch_related("autores")
libros = libros.all()
```

```
libros = (Libro.objects.raw("SELECT * FROM biblioteca_libro l "
                            + " JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id "
                            + " JOIN biblioteca_libro_autores la ON la.libro_id = l.id ")
          )
```

```
<RawQuerySet: SELECT * FROM biblioteca_libro l  JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id  JOIN biblioteca_l
ibro_autores la ON la.libro_id = l.id >
```

Dentro de esta función tendremos que llamar a las tablas, tal y como están creadas a la base de datos y especificar la tabla intermedia creada para las relaciones ManytoMany y no se específico. Para ello podemos usar un plugin para ver las base de datos sqlite3 en Vscode.

Template

Ahora creamos el directorio **libro** dentro de templates y el archivo lista.html dentro de esa carpeta.

¿Qué destacamos?

- Para obtener datos de la **Biblioteca**, solo debemos usar el atributo **biblioteca** y luego llamar al atributo de dicho modelo que queremos usar.
- Para mostrar los **autores**, recorremos la variable autores, pero usando **all**, para indicar que queremos todos los **autores**
- Para mostrar el valor del idiomas que es de tipo choices, debemos usar **_display**

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
    <h1>Django girls</h1>

    {% for libro in libros_mostrar %}
        <div>
            <h2>Nombre: {{ libro.nombre }} </h2>
            <h2>Biblioteca {{ libro.biblioteca.nombre }}</h2>
            <h2>Autores</h2>
            <ul>
                {% for autor in libro.autores.all %}
                    <li>{{ autor.nombre }}</li>
                {% endfor%}
            </ul>
            <p>Fecha Publicacion {{ libro.fecha_publicacion }}.

            </p>
            <h3>Idioma: {{ libro.get_idioma_display }}

            </h3>
            <p>Descripcion{{ libro.descripcion }}</p>
        </div>
    {% endfor %}

    </body>
</html>
```

URLs

Por último incluimos la URl en el html **index.html**, para que podamos acceder a ella, usando la siguiente etiqueta:

```
<a href="{% url 'lista_libros' %}">Libros</a>
```

Es importante que el nombre que incluyamos debe ser el mismo que pusimos en el archivo **biblioteca/urls.py**

Si ahora pinchamos en el enlace debería llevarnos a una página donde nos muestra todos los libros.

¿Y qué pasa si quiero ver la información de un Libro en concreto? Pues creamos el url.

URLs

Ahora vamos a crear una URL para acceder a un libro en concreto:

```
path("libros/<int:id_libro>/", views.dame_libro,name="dame_libro"),
```

Destacamos lo nuevo que hemos incluido en la URL: **<int:id_libro>**, esto significa que tenemos que pasarlo un número entero a esa url, y que se guardará en una variable que se llama **id_libro**.

View y QuerySet

Ahora creamos la vista:

```
def dame_libro(request,id_libro):
    libro = Libro.objects.select_related("biblioteca").prefetch_related("autores").get(id=id_libro)
    return render(request, 'libro/libro.html',{"libro_mostrar":libro})
```

¿Qué hemos incluido nuevo?

Ahora tenemos una vista, que además del parámetro request, le pasamos otro parámetro que **id_libro**. El valor de este parámetro lo obtendrá del entero que le pasemos en la URL.

Además hemos usado un método nuevo de las QuerySet: **get**

Este método recibe cómo parámetro, la columna que desea buscar y el valor de esa columna, para obtener el registro concreto.

Transformación a SQL

En la función **raw()** para pasar parámetros, debemos indicar en la consulta dónde incluir los parámetros con **%s**, y por otro lado, incluir un parámetro más con un diccionario de python, con las variables que le darán valor a esos parámetros, en el mismo orden que los hemos especificados.

También incluimos en **[0]** para indicar que vamos a obtener sólo un elemento en vez de un conjunto de datos.

```
libro = (Libro.objects.raw("SELECT * FROM biblioteca_libro l "
                           + " JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id "
                           + " JOIN biblioteca_libro_autores la ON la.libro_id = l.id "
                           + " WHERE l.id = %s",[id_libro])[0]
         )
```

```
<RawQuerySet: SELECT * FROM biblioteca_libro l  JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id  JOIN biblioteca_libro_autores la
 ON la.libro_id = l.id  WHERE l.id = 1>
```

Template

Ahora creamos la Template

correspondiente:

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
    <h1>Django girls</h1>

    <div>
            <h2>Nombre: {{ libro_mostrar.nombre }} </h2>
            <h2>Biblioteca {{ libro_mostrar.biblioteca.nombre }}</h2>
            <h2>Autores</h2>
            <ul>
                {% for autor in libro_mostrar.autores.all %}
                    <li>{{ autor.nombre }}</li>
                {% endfor%}
            </ul>
            <h3>Idioma: {{ libro_mostrar.get_idioma_display }}</h2>
            <p>Descripcion{{ libro_mostrar.descripcion }}</p>
    </div>

    </body>
</html>
```

URls: Link

Para poder crear un link a dicha página, podemos incluir la siguiente línea en el template **listar.html:**

```
<a href="{% url 'dame_libro' id_libro=libro.id %}">Pulsa para ver con más detalle</a>
```

dame_libro: Es el nombre de la URL

id_libro: El nombre del parámetro de la URL

libro.id: Es el id del libro

URLs

Una url que me muestre los libros de un año y mes concreto:

```
path("libros/listar/<int:anyo_libro>/<int:mes_libro>", views.dame_libros_fecha,name="dame_libros_fecha"),
```

Como podemos comprobar, podemos crear URLs con varios parámetros separados por “/”

View y QuerySet

Ahora creamos la vista:

```
def dame_libros_fecha(request,anyo_libro,mes_libro):
    libros = Libro.objects.select_related("biblioteca").prefetch_related("autores")
    libros = libros.filter(fecha_publicacion__year=anyo_libro,fecha_publicacion__month=mes_libro)
    return render(request, 'libro/lista.html',{"libros_mostrar":libros})
```

¿Qué destacamos de esta vista?

- Pasamos dos parámetros a la vista:anyo_libro y mes_libro
- Aplicamos el método **filter:** qué sirve para filtrar por un campo en la base de datos, como un **WHERE** en Mysql, y podemos usar **“,”** para hacer **AND.**
- Para acceder al campo año y mes de un campo de tipo fecha, debemos usar **DOS BARRAS BAJAS: __.** Esto sería como usar las funciones **YEAR** y **MONTH** de SQL

Transformación SQL

Sólo destacar que los campos de fecha hay que pasarlos a la función **raw** de tipo string:

```
libros = (Libro.objects.raw("SELECT * FROM biblioteca_libro l "
                            + " JOIN biblioteca_libro_autores la ON la.libro_id = l.id "
                            + " JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id "
                            + " WHERE strftime('%%Y', l.fecha_publicacion) = %s "
                            + " AND strftime('%%m', l.fecha_publicacion) = %s "
                            ,[str(anyo_libro),str(mes_libro)])
```

```
<RawQuerySet: SELECT * FROM biblioteca_libro l  JOIN biblioteca_libro_autores la ON la.libro_id = l.id  JOIN biblioteca_biblioteca b ON
l.biblioteca_id = b.id  WHERE strftime('%Y', l.fecha_publicacion) = 2017  AND strftime('%m', l.fecha_publicacion) = 10 >
```

Link

Para poder crear un link a dicha página, podemos incluir la siguiente línea en el template **listar.html:**

```
<p>Fecha Publicacion {{ libro.fecha_publicacion }}.
    <a href="{% url 'dame_libros_fecha' libro.fecha_publicacion.year libro.fecha_publicacion.month %}">
        Pulsa para ver libros del mismo año y mes
    </a>
</p>
```

¿Qué destacamos?

- En este caso le pasamos dos valores a la URL
- No hemos indicado el nombre de la variable en la URL, porque no es necesario, pero si aconsejable.
- Para acceder al valor del año y mes de un campo fecha tenemos que acceder como si fueran atributos del campo fecha_publicacion.

URL

Una url que me muestre los libros que tienen el idioma del libro o español ordenados por fecha de publicación:

```
path("libros/listar/<str:idioma>/", views.dame_libros_idioma,name="dame_libros_idioma"),
```

¿Qué destacamos?

- Podemos usar otros tipos en los enlaces, como las cadenas.

View y QuerySet

La vista correspondiente es:

```
from django.db.models import Q
```

```
def dame_libros_idioma(request,idioma):
    libros = Libro.objects.select_related("biblioteca").prefetch_related("autores")
    libros = libros.filter(Q(idioma=idioma) | Q(idioma="ES")).order_by("fecha_publicacion")
    return render(request, 'libro/lista.html',{"libros_mostrar":libros})
```

¿Qué destacamos?:

- la letra **Q con |** sirven para poder hacer filtros con **OR.**
- Los filtros con un campo de tipo **Choices** se filtra por la clave.
- Para ordenar de forma **ASCENDENTE** usamos la función **order_by**

Transformación SQL

```
libros = (Libro.objects.raw("SELECT * FROM biblioteca_libro l "
                            + " JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id "
                            + " JOIN biblioteca_libro_autores la ON la.libro_id = l.id "
                            + " WHERE idioma = 'ES' "
                            + " OR idioma = %s "
                            + " ORDER BY l.fecha_publicacion"
                            ,[idioma])
          )
```

```
<RawQuerySet: SELECT * FROM biblioteca_libro l  JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id  JOIN biblioteca_libro_autores la
 ON la.libro_id = l.id  WHERE idioma = 'ES'  OR idioma = ES  ORDER BY l.fecha_publicacion>
```

Link

El link para acceder a la URL anterior sería:

```
<a href="{% url 'dame_libros_idioma' libro.idioma %}">
    Pulsa para ver libros del mismo idioma o Español
</a>
```

URL

Una url que me muestre los libros de una biblioteca que contenga un texto en concreto:

```
path("biblioteca/<int:id_biblioteca>/libros/<str:texto_libro>", views.dame_libros_biblioteca,name="dame_libros_biblioteca"),
```

¿Qué destacamos?

- Hemos incluido dos parámetros: Uno de tipo entero y otro de texto
- Ambos parámetros están separados por un texto fijo que es **“libros”**

View y QuerySet

Ahora creamos la vista

```
def dame_libros_biblioteca(request,id_biblioteca,texto_libro):
    libros = Libro.objects.select_related("biblioteca").prefetch_related("autores")
    libros = libros.filter(biblioteca=id_biblioteca).filter(descripcion__contains=texto_libro).order_by("-nombre")
    return render(request, 'libro/lista.html',{"libros_mostrar":libros})
```

¿Qué destacamos?

- Accedemos a la biblioteca(Relacion OnetoOne), pasando el id de la biblioteca al atributo correspondiente
- Podemos encadenar .**filter**, para hacer un AND también.
- Para poder hacer un **LIKE '%%'** de **MYSQL** dentro de un campo concreto, hemos añadido al atributo descripcion **__contains**
- Para ordenar de forma descendente, hemos incluido un **menos(-)** delante de "**nombre**"

Transformación SQL

```
texto_buscar = "'%"+texto_libro+"%'"
libros = (Libro.objects.raw("SELECT * FROM biblioteca_libro l "
                            + " JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id "
                            + " JOIN biblioteca_libro_autores la ON la.libro_id = l.id "
                            + " WHERE b.id = %s "
                            + " AND l.descripcion LIKE %s "
                            + " ORDER BY l.nombre DESC "
                            ,[id_biblioteca,texto_buscar])
        )
```

```
<RawQuerySet: SELECT * FROM biblioteca_libro l  JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id  JOIN biblioteca_libro_autores la
 ON la.libro_id = l.id  WHERE b.id = 1  AND l.descripcion LIKE '%prueba%'  ORDER BY l.nombre DESC >
```

Link

Para crear el link que acceda a la URL anterior, tenemos el siguiente ejemplo:

```
<ul>
    <li><a href="{% url 'dame_libros_biblioteca' 1 'prueba' %}">Libros Biblioteca 1 con filtro de texto</a></li>
</ul>
```

¿Qué destacamos?

- Esta vez hemos pasado valores directamente a los parámetros en vez de usar atributos de modelos. Hemos usado el valor **“1”** para indicar la **biblioteca** con **id 1** y el valor **‘prueba’**(tiene que estar entre comillas porque es una cadena), para indicar que quiero que contenga el texto prueba.
- No es necesario especificar que entre ambos parámetros está la ruta **/libros/** porque lo único que espera la URL es dos parámetros.

Url

Una url que me muestre el último cliente que se llevó un libro en concreto:

```
path('ultimo-cliente-libro/<int:libro>',views.dame_ultimo_cliente_libro,name='ultimo_cliente_libro'),
```

View y QuerySet

Ahora creamos la Vista:

```
def dame_ultimo_cliente_libro(request,libro):
    cliente = Cliente.objects.filter(prestamo__libro=libro).order_by("-prestamo__fecha_prestamo")[:1].get()
    return  render(request, 'cliente/cliente.html',{"cliente":cliente})
```

¿Qué destacamos?:

- En este caso usamos una tabla intermedia(prestamo), de una relación ManytoMany, para saber qué clientes sacaron esos libros
- **[:1]** Significa que queremos obtener sólo un registro. Si pusiéramos **[:5],** obtendremos 5 registros, y si pusieramos **[3:6]** obtendremos del registro 3 al 6.(Esto son los **LIMIT** y **OFFSET** de MYSQL)
- **get():** A la hora de usar las QuerySet, siempre nos devuelve un conjunto de registros, en este caso sólo queremos uno, por lo tanto esta función nos devolvera el registro en sí, en vez de un conjunto.

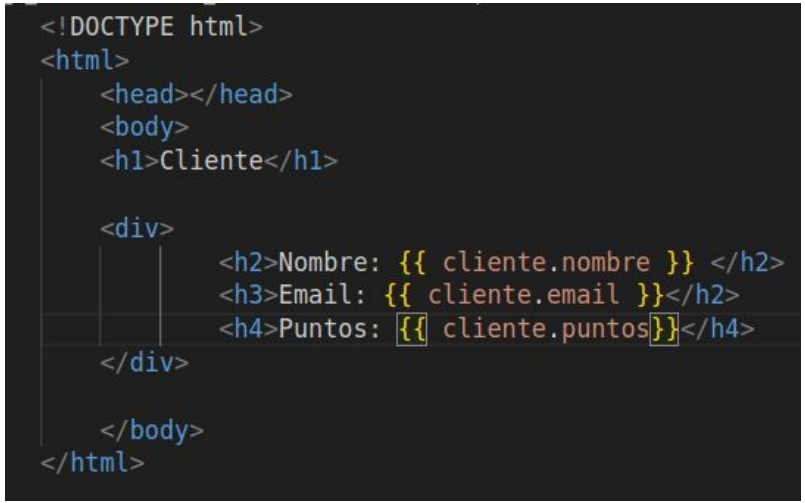
Transformación SQL

```
cliente = (Cliente.objects.raw("SELECT * FROM biblioteca_cliente c "
                               + " JOIN biblioteca_prestamo p ON p.libro_id = %s "
                               + " ORDER BY p.fecha_prestamo DESC "
                               ,[libro])[0]
```

```
<RawQuerySet: SELECT * FROM biblioteca_cliente c  JOIN biblioteca_prestamo p ON p.libro_id = 1  ORDER BY p.fecha_prestamo DESC >
```

Template

Hemos creado una carpeta llamada cliente y una plantilla llamada cliente.html con el siguiente aspecto



```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
    <h1>Cliente</h1>

    <div>
            <h2>Nombre: {{ cliente.nombre }} </h2>
            <h3>Email: {{ cliente.email }}</h2>
            <h4>Puntos: {{ cliente.puntos}}</h4>
    </div>

    </body>
</html>
```

Link

En la plantilla de **libro/lista.html** incluimos lo siguiente:

```
<h3>
    Último cliente que se llevo el libro:
    <a href="{% url 'ultimo_cliente_libro' libro.id %}">
        Pulsa para ver el último cliente
    </a>
</h3>
```

Url

URl que obtiene los libros que nunca han sido prestados:

```
re_path(r"^filtro[0-9]$", views.libros_no_prestados,name="libros_no_prestados"),
```

¿Qué destacamos?

- **re_path**: Sirve para crear expresiones regulares con las URLS.
- **r”**: Es para indicar que es una expresión regular
- **¿Qué es una expresión regular?**: Es un conjunto de caracteres especiales que identifican un conjunto de patrones de texto o numéricos, en vez de sólo una palabra en concreto.

URL: Expresiones regulares

^ denota el principio del texto

$ denota el final del texto

\d representa un dígito

+ indica que el ítem anterior debería ser repetido por lo menos una vez

() para encerrar una parte del patrón

View

Ahora creamos la vista, para obtener los libros que no han sido prestados. Para ello tenemos que obtener todos los libros que no tengan un registro asociado en la tabla **Prestamo**:

```
def libros_no_prestados(request):
    libros = Libro.objects.select_related("biblioteca").prefetch_related("autores")
    libros = libros.filter(prestamo=None)
    return render(request, 'libro/lista.html',{"libros_mostrar":libros})
```

¿Qué destacamos?

- **filter(prestamo=None):** En este caso la tabla intermedia es Prestamo, y el atributo se llama así, y queremos que el valor en el filtro sea nulo.

Transformación SQL

```
libros = (Libro.objects.raw("SELECT * FROM biblioteca_libro l "
                            + " JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id "
                            + " JOIN biblioteca_libro_autores la ON la.libro_id = l.id "
                            + " LEFT JOIN biblioteca_prestamo p ON p.libro_id = l.id "
                            + " WHERE p.id IS NULL ")
```

```
<RawQuerySet: SELECT * FROM biblioteca_libro l  JOIN biblioteca_biblioteca b ON l.biblioteca_id = b.id  JOIN biblioteca_libro_autores la
 ON la.libro_id = l.id  LEFT JOIN biblioteca_prestamo p ON p.libro_id = l.id  WHERE p.id IS NULL >
```

Url

Una Url que muestra una Biblioteca y sus libros:

```
path('biblioteca/<int:id_biblioteca>/',views.dame_biblioteca,name='dame_biblioteca'),
```

View

Ahora creamos la vista

```
def dame_biblioteca(request,id_biblioteca):
    biblioteca = Biblioteca.objects.get(id=id_biblioteca)
    return render(request, 'biblioteca/biblioteca.html',{"biblioteca":biblioteca})
```

Transformación SQL

```
biblioteca = (Biblioteca.objects.raw("SELECT * FROM biblioteca_biblioteca b "
                                     + " JOIN biblioteca_libro l ON l.biblioteca_id = b.id "
                                     )
              )[0]
```

```
RawQuerySet: SELECT * FROM biblioteca_biblioteca b  JOIN biblioteca_libro l ON l.biblioteca_id = b.id >
```

Template

Creamos la plantilla biblioteca/biblioteca.html

¿Qué destacamos?

A la hora de obtener los libros, usamos el atributo **libros_biblioteca**, pero ¿de dónde sale ese atributo?

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
    <h1>Biblioteca</h1>

    <div>
            <h2>Nombre: {{ biblioteca.nombre }} </h2>
            <h3>Email: {{ biblioteca.email }}</h2>
            <h3>Libros</h3>
            {% for libro in biblioteca.libros_biblioteca.all %}
                <li>{{ libro.nombre }}</li>
            {% endfor%}

    </div>

    </body>
</html>
```

Relación Reversa

Cuando creamos los **Modelos**, las relaciones **ManytoMany** y **OnetoOne** sólo se especifican en uno de los modelos.

En el caso de acceder a la Biblioteca de un Libro, sólo debemos poner: libro.biblioteca. ¿Pero y si quiero hacerlo al revés?¿Qué tendría que poner?

Pues para ello, debemos usar el parámetro **related_name,** no sólo nos va a servir para definir el nombre de la relación para evitar duplicidades, sino para utilizarlo como atributo en los modelos que no están definidos.

Si no usamos el **related_name** por defecto podemos usar **_set**, pero es preferible no usarlo para evitar confusiones y tener el código más ordenado y estructurado, y optimizar las querySets.

Relación Reversa

Por lo tanto nuestro modelo Libro,en la relación **ManyToOne**, debería quedar así:

```
biblioteca = models.ForeignKey(Biblioteca, on_delete = models.CASCADE,related_name="libros_biblioteca")
```

Optimizar View

En la vista anterior, los datos los libros de la biblioteca los obtenemos al pedirlo en la Template. Pero estaría mejor obtenerlos en la vista, para ello debemos hacer lo siguiente:

```
from django.db.models import Q,Prefetch
```

```
def dame_biblioteca(request,id_biblioteca):
    biblioteca = Biblioteca.objects.prefetch_related(Prefetch("libros_biblioteca")).get(id=id_biblioteca)
    return render(request, 'biblioteca/biblioteca.html',{"biblioteca":biblioteca})
```

¿Qué destacamos?

Aunque sea una relación **ManytoOne**, debemos usar **prefetch_related**, mezclado con **Prefetch. En este caso sería como una relación OnetoMany**

Prefecth:Esta función nos permite realizar los INNER JOIN con las relaciones inversas, sólo tenemos que especificar el atributo de **related_name**

URL

Una url que muestre los libros que contienen el título del libro en la descripción

```
path('dame-libros-titulo-descripcion',views.dame_libros_titulo_en_descripcion,name="dame_libros_titulo_en_descripcion"),
```

Vista

```python
#Una url que muestre los libros que contengan en la descripcion el titulo del libro
def dame_libros_titulo_en_descripcion(request):
    libros = Libro.objects.select_related("biblioteca").prefetch_related("autores")
    libros = libros.filter(descripcion__contains=F("nombre"))

    return render(request, 'libro/lista.html',{"libros_mostrar":libros})
```

¿Qué destacamos?

En este caso usamos el objeto **F** porque queremos usar el valor de un atributo del mismo modelo para usar la query. En este caso no obtenemos el valor de una variable,sino de otro atributo(columna en SQL).A continuación mostramos una SQL de ejemplo de este código.

```
"SELECT * FROM biblioteca_libro l "
"WHERE l.descripcion LIKE  CONCAT('%',l.nombre,'%')"
```

URL

```
path('dame-agrupaciones-puntos-clientes',views.dame_agrupaciones_puntos_cliente,name="dame_agrupaciones_puntos_cliente"),
```

Vista

```
from django.db.models import Avg,Max,Min
```

```
#Una url que muestra la media, máximo y  mínimo de puntos de todos los clientes de la Biblioteca
def dame_agrupaciones_puntos_cliente(request):
    resultado = Cliente.objects.aggregate(Avg("puntos"),Max("puntos"),Min("puntos"))
    media = resultado["puntos__avg"]
    maximo = resultado["puntos__max"]
    minimo = resultado["puntos__min"]
    return render(request, 'cliente/agrupaciones.html',{"media":media,"maximo":maximo,"minimo":minimo})
```

¿Qué destacamos?

En este caso para realizar las operaciones de Avg(media), Max(Máximo) y Min(Mínimo), solo debemos usar la función **aggregate** sobre el campo que vamos a realizar la operación.

El resultado es un diccionario con los valores de las operaciones realizadas, que la clave siempre es el **atributo__operacionRealizada**

Para todas las operaciones de aggregate, acceder a la documentación

Páginas de Error

En cualquier página Web es importante definir las páginas de Error, para mejorar la usabilidad de la Web.

Por ejemplo crear una página personalizada, cuando un usuario no encuentra una página concreta: **el error 404**

Vamos a seguir los siguientes pasos:

- Dentro de templates creamos la carpeta **errores**
- Dentro de la carpeta errores, creamos el archivo **404.html**
- El contenido de dicho archivo puede ser el siguiente.

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
    <h1>Página 4040</h1>

    <div>
            <h1>A la página que intenta acceder no existe</h1>
    </div>

    </body>
</html>
```

View Página de Error

Para crear la vista de la página de error debemos hacer lo siguiente:

- Importar la página de error

```
from django.views.defaults import page_not_found
```

- Crear la vista y que redireccione a mi plantilla

```
def mi_error_404(request,exception=None):
    return render(request, 'errores/404.html',None,None,404)
```

Importante indicar el 5 parámetro el estado de la respuesta:404

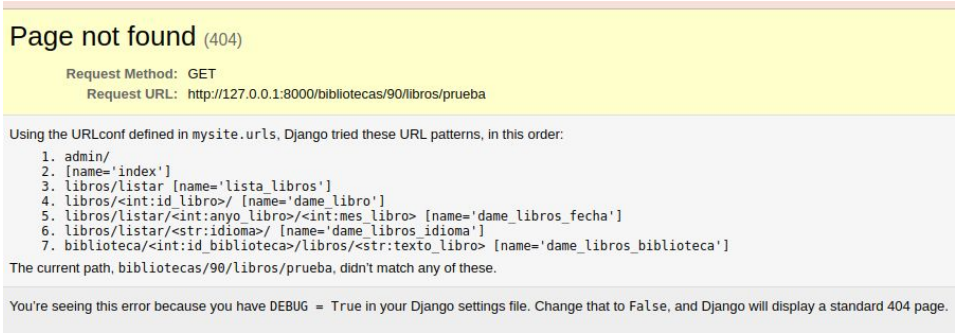
URL Página de Error

Y por último creamos la URl a la vista correspondiente en el archivo **mysite/urls.py.** Es importante crearlo ahí asociamos nuestra vista al manejador:

```
handler404 = "biblioteca.views.mi_error_404"
```

Página Error

Perro si ahora accedemos a una página que no existe en nuestro programa. Nos aparece la siguiente página de Error, que no es la que hemos configurado. ¿Por qué?



Pues la misma página nos da la solución más abajo. Estamos en un entorno de prueba, definir las páginas de errores sólo nos servirán para cuando estemos en un entorno de producción.

¿Y cómo cambiamos a un entorno de producción? Modificamos la variable **DEBUG** en **settings.py** a **False**

```
DEBUG = False
```

Página Error

A continuación mostraremos un ejemplo de como se mostraba antes y como se muestra ahora:

Not Found

The requested resource was not found on this server.

Página 404

A la página que intenta acceder no existe

Páginas de Error

Existen más páginas de Error:

- `handler400 (Petición mál realizada)`
- `handler403 (Permiso Denegado)`
- `handler500 (Error Interno en el Servidor)`