### El caparazón de Django

Ahora que hemos introducido algunos datos, podemos examinarlos programáticamente a través de una sesión de terminal interactiva. Este entorno interactivo se llama Django *shell*, y es un entorno estupendo para probar y solucionar problemas de tu proyecto. Aquí tienes un ejemplo de sesión de terminal interactiva:

(ll\_env)learning\_log$ python manage.py shell  
❶ >>> from learning\_logs.models import Topic  
>>> Topic.objects.all()  
<QuerySet [<Topic: Chess>, <Topic: Rock Climbing>]>

El comando python manage.py shell, ejecutado en un entorno virtual activo, lanza un intérprete de Python que puedes utilizar para explorar los datos almacenados en la base de datos de tu proyecto. Aquí, importamos el modelo Topic del módulo learning\_logs.models ❶. A continuación, utilizamos el método Topic.objects.all() para obtener todas las instancias del modelo Topic; la lista que se devuelve se denomina *queryset*.

Podemos hacer un bucle sobre un conjunto de consultas igual que lo haríamos sobre una lista. Así puedes ver el ID que se ha asignado a cada objeto tema:

>>> topics = Topic.objects.all()  
>>> for topic in topics:  
... print(topic.id, topic)  
...  
1 Chess  
2 Rock Climbing

Asignamos el conjunto de consultas a topics y, a continuación, imprimimos el atributo id de cada tema y la representación de cadena de cada tema. Podemos ver que Chess tiene un ID de 1 y Rock Climbing tiene un ID de 2.

Si conoces el ID de un objeto concreto, puedes utilizar el método Topic.objects.get() para recuperar ese objeto y examinar cualquier atributo que tenga. Veamos los valores text y date\_added de Chess:

>>> t = Topic.objects.get(id=1)  
>>> t.text  
'Chess'  
>>> t.date\_added  
datetime.datetime(2022, 5, 20, 3, 33, 36, 928759,  
 tzinfo=datetime.timezone.utc)

También podemos mirar las entradas relacionadas con un tema determinado. Anteriormente, definimos el atributo topic para el modelo Entry. Se trataba de un ForeignKey, una conexión entre cada entrada y un tema. Django puede utilizar esta conexión para obtener cada entrada relacionada con un tema determinado, de la siguiente manera:

❶ >>> t.entry\_set.all()  
<QuerySet [<Entry: The opening is the first part of the game, roughly...>, <Entry:  
In the opening phase of the game, it's important t...>]>

Para obtener datos a través de una relación de clave foránea, utiliza el nombre en minúsculas del modelo relacionado seguido de un guión bajo y la palabra set ❶. Por ejemplo, supongamos que tienes los modelos Pizza y Topping, y que Topping está relacionado con Pizza a través de una clave ajena. Si tu objeto se llama my\_pizza, y representa una única pizza, puedes obtener todos los ingredientes de la pizza utilizando el código my\_pizza.topping\_set.all().

Utilizaremos esta sintaxis cuando empecemos a codificar las páginas que pueden solicitar los usuarios. El intérprete de comandos es realmente útil para asegurarte de que tu código recupera los datos que deseas. Si tu código funciona como esperas en el intérprete de comandos, también debería funcionar correctamente en los archivos de tu proyecto. Si tu código genera errores o no recupera los datos que esperas, es mucho más fácil solucionar los problemas de tu código en el sencillo entorno del shell que dentro de los archivos que generan las páginas web. No nos referiremos mucho al shell, pero deberías seguir utilizándolo para practicar el trabajo con la sintaxis de Django para acceder a los datos almacenados en el proyecto.

Cada vez que modifiques tus modelos, tendrás que reiniciar la shell para ver los efectos de esos cambios. Para salir de una sesión de shell, pulsa CTRL-D; en Windows, pulsa CTRL-Z y luego ENTER.

[anterior](c18_24.html)[Subtema 25 de 41: (Ver todo)](c18.html)[siguiente](c18_26.html)