TEMA IV

DESARROLLO DE APLICACIONES WEB EN BACKEND

"Siempre parece imposible,
hasta que se hace"
Nelson Mandela

7.7 BASES DE DATOS

Django puede trabajar con varios motores de bbdd:

1. Soportados oficialmente
   - SQLite3: gestor por defecto
   - PostgreSQL: gestor recomendado
   - MySQL
   - Oracle

2. Con drivers de terceros
   - SQLServer
   - DB2
   - SAB SQL….

Empezaremos con SQLite3 porque es muy sencillo el manejo de tablas y no tendremos que crear la bbdd

MODELOS EN DJANGO

Para trabajar con SQLite3 no vamos a necesitar crear la bbdd pero sí las tablas. Django tiene un sistema de mapeado **ORM** (modelo-objeto-relacional), esto significa que podremos trabajar con objeto mapeados en la bbdd de manera que al crear instancias de una clase específica (clase Model) éstas quedan guardadas como registros de forma automática. Estas clases persistentes en django se llaman modelos.

Los modelos en django no funcionan si no he creado una aplicación en mi proyecto.

El archivo models.py será el que utilicemos para trabajar con bbdd, también nos genera automáticamente el archivo views.

Como voy a trabajar con bbdd, de momento no ejecuto el comando migrate, los pasos a seguir:

1. En el archivo models creo una clase por cada tabla que queremos que tenga nuestra bbdd. No utilizaremos nosotros SQL, lo hace por debajo Django

   Las clases deber heredar de models.Model y tendrán como atributos de clases todos los campos que llevará la tabla.

   nombrecampo=tipocampo(parámetros que lo definen en la tabla)

   Tipos de campos: CharField, TextField (para campos de texto más grande), EmailField , IntegerField, DecimalField, DateField, ImageField,BooleanField...

Curiosidades:

fechaCreacion=DateTimeFied(auto_now_add=true)

fechaActualizacion=DateTimeField(auto_now=true)

2. Una vez que están definidos los modelos creamos la bbdd

   python3 manage.py makemigrations

Cada migración lleva asociado un número que nos servirá para identificarla, en la figura se ve que es 0001. Si después de crear la bbdd y añadir las tablas, hiciéramos alguna modificación tendríamos que migrar de nuevo y nos daría otro número. Esto es útil para el control de versiones.

3. Para añadir en esa bbdd las tablas:

python3 manage.py migrate

Obviamente, además de nuestras tablas genera "otras cosas"

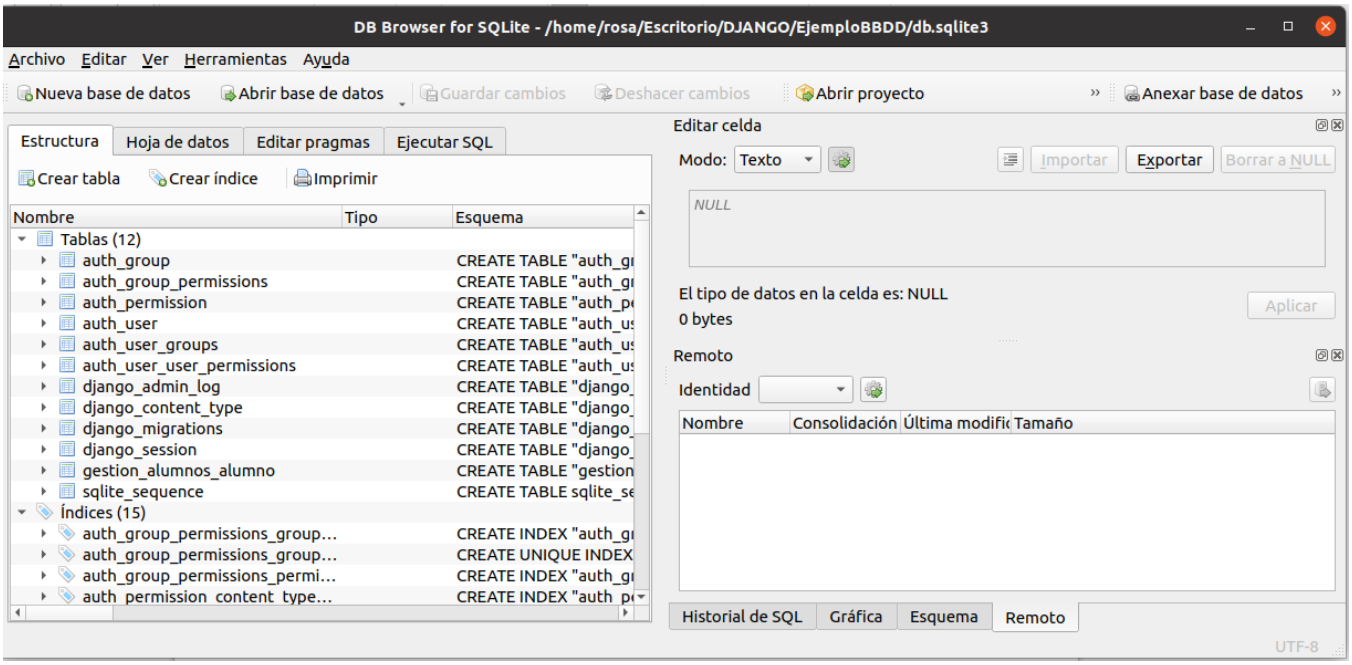
Si no añades a la clase ninguna primaryKey, Django añade un campo llamado id con esa función. No será nuestro caso porque diseñamos bien las tablas.

No olvidar añadir la aplicación al archivo de configuraciones.

Una ventaja que tiene Django es que genera el código SQL para el motor de base de datos que utilicemos, sin tener que preocuparnos nosotros de las "pequeñas diferencias" que hay entre las distintas bbdd.

Hay programas gráficos que podéis instalar para visualizar la bbdd, instalaremos DB Browser Sqlite

sudo apt-get install sqlitebrowser



Además de nuestra tabla, django genera otras muchas que necesita para trabajar.

Veamos como trabajar con nuestra bbdd

Trabajando con la BBDD

Veamos como insertar, borrar o actualizar registros.

Si pulsamos sobre la tabla, botón derecho vemos que evidentemente está vacía

1. Para añadir un registro:
   - Instanciamos un objeto del tipo que hemos definido en el modelo

```
# Create your models here.
class Alumno(models.Model):
    nombre=CharField(max_length=80)
    dni=CharField(max_length=9,primary_key=True)
    email=EmailField(max_length=50)
    nota=DecimalField(max_digits=4,decimal_places=2)
```

```
@csrf_exempt
def anyadir(request):
    if request.method=='POST':
        my_frm=FrmAlumno(request.POST)
        if my_frm.is_valid():
            nombre=my_frm.cleaned_data['nombre']
            email=my_frm.cleaned_data['email']
            dni=my_frm.cleaned_data['dni']
            nota=my_frm.cleaned_data['nota']
            alumno=Alumno(nombre=nombre,email=email,dni=dni,nota=nota)
            alumno.save()
            mensaje='Alumno añadido'
            return render(request,'base.html',{'mensaje':mensaje})
    else:
        my_frm=FrmAlumno()
    return render(request,'anyadir.html',{'form':my_frm})
```

Ejecutamos su método save(), este será el que ejecute la instrucción INSERT

Si el proceso provoca un error la bbdd podría bloquearse.

Los decimales se introducen con punto

Otra forma de añadir

alumno=Alumno.objects. create(nombre=nombre,email=email,d…)

2. Para modificar un registro
   - Cambiamos la propiedad en la variable del objeto
   - Ejecutamos save

     alumno.nota=6

     alumno.save()

3. Borrar registros
   - Utilizamos el método get de objects para buscar el registro
   - Ejecutamos el método delete sobre el objeto que nos retorna el método anterior

4. Obtener todos los registros
   - Ejecutamos Objeto.objects.all()

5. Obtener un registro
   - Ejecutamos Objeto.objects.get(campo clave='valor')

6. Filtrar registros
   - Objeto.objects.filter(campo=valor)

Se pueden encadenar filtros para búsquedas por más de un parámetro

- Objeto.objects.filter(campo=valor).filter(campo=valor)

  Objeto.objects.filter(campo=valor,campo=valor)

  Ambas opciones funcionan igual

Para comparaciones aritméticas (<,<=,>,>=) se usa It,Ite,gt, gte con la siguiente sintaxis: campo__gte, campo__it,...

- Objeto.objects.filter(nota__get=5)

En esta url se pueden consultar operaciones frecuentes:

https://programmerclick.com/article/87131207225/

7. Ordenar

- order_by(campo1, [campo2],...)
- Si anteponemos – al nombre del campo tenemos orden inverso

Se pueden consultar todas las operaciones posibles en la documentación oficial

Veamos todo esto con un ejemplo: EjemploBBDD