**CONECTANDO DJANGO CON POSTGRESQL**

Para realizar la conexión desde django es necesario modificar el archivo **settings.py** en la carpeta del proyecto

Dentro del diccionario databases se encuentra la config por default, que se reemplaza por las características de la db de postgres instalada en nuestro equipo

DATABASES = {

    # 'default': {

    #     'ENGINE': 'django.db.backends.sqlite3',

    #     'NAME': BASE\_DIR / 'db.sqlite3',

    # }

    'default': {

        'ENGINE': 'django.db.backends.postgresql\_psycopg2',

        'NAME': 'sist-adm-personas',

        'USER': 'postgres',

        'PASSWORD': 'admin',

        'HOST': 'localhost',

        'PORT': '5432'

    }

}

Por ultimo es necesario instalar el paquete **psycopg2** en la carpeta principal del proyecto (cd al mismo nivel de la carpeta que contiene el entorno virtual)

**MIGRACIONES**

Para continuar trabajando con la db es necesario que django realice ciertas actualizaciones en la db.

Para ver las migraciones necesarias:

**Python** manage.py showmigrations

Para migrar:

Python manage.py migrate

Ahora en nuestro pgadmin en la db podemos ver que en public->tables se crearon algunas

Todos estos datos están relaconadas con las configuraciones de la app, pero no datos de la misma.

**CREANDO UNA CLASE MODELO**

Este concepto nos permite manejar un modelo para realizar inserciones en la db. Para esto es necesario crear una clase con especificaciones necesarias que se relacionará directamente con la DB.

Lo primero es crear una nueva app, en este caso con el nombre **personas**.

Python manage.py startapp personas

Una vez creada, es necesario registrarla en “el proyecto”, para esto hay que acceder a la carpeta que contiene las configuraciones del proyecto y en el archivo settings agregar la nueva app en el dicc INSTALLED\_APPS:

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'webapp',

    'personas',

]

Ahora podemos trabajar con el archivo models dentro de la nueva app “personas”

Aquí se define una nueva clase, que extiende de **models.Model**

Los atributos de nuestra “tabla modelo” deben ser attr estáticos y definen el tipo de dato para que luego se cree correctamente en la db

class Persona(models.Model):

    nombre = models.CharField(max\_length=255)

    apellido = models.CharField(max\_length=255)

    email = models.CharField(max\_length=255)

ahora, desde la terminal se ejecuta:

python manage.py makemigrations 🡪 genera los archivos para la migración

Crea en la carpeta migrations de la nueva app un archivo, en este caso 0001\_initial.py

python manage.py migrate 🡪 se aplican las migraciones pendientes

en caso de que se quiera ver el sql a ejecutar:

python manage.py sqlmigrate personas **0001(este num es el del archivo en la carpeta migrations)**

Argumentos comunes de los campos

Los siguientes argumentos son comunes a la mayoría de los tipos de campo y pueden usarse al declararlos:

help\_text: Proporciona una etiqueta de texto para formularios HTML (ej. en el sitio de Administración), tal como se describe arriba.

verbose\_name: Nombre de fácil lectura que se usa en etiquetas para el campo. Si no se especifica, Django inferirá el valor por defecto del verbose name a partir del nombre del campo.

default: Valor por defecto para el campo. Puede ser un valor o un callable object (objeto que puede ser llamado como una función), en cuyo caso el objeto será llamado cada vez que se cree un nuevo registro.

null: Si es True, Django guardará valores en blanco o vacíos como NULL en la base de datos para campos donde sea apropiado (un CharField guardará una cadena vacía en su lugar). Por defecto es False.

blank: Si es True, se permite que el campo quede en blanco en tus formularios. El valor por defecto es False, lo que significa que la validación de formularios de Django te forzará a introducir un valor. Con frecuencia se usa con null=True, porque si vas a permitir valores en blanco, también querrás que la base de datos sea capaz de representarlos de forma apropiada.

choices: Un grupo de valores de selección para este campo. Si se proporciona, el widget correspondiente por defecto del formulario será una caja de selección con estos valores de selección en vez del campo de texto estándar.

primary\_key: Si es True, establece el campo actual como clave primaria para el modelo (Una clave primaria es una columna especial de la base de datos, diseñada para identificar de forma única todos los diferentes registros de una tabla). Si no se especifica ningún campo como clave primaria, Django añadirá automáticamente un campo para este propósito.

**CREANDO UN SUPER ADMIN**

En todas las aplicaciones web Python existe una consola de administración.

En la ruta /admin/ se accede a la misma y se presenta un campo de inicio de sesión. Esta consola está destinada principalmente al usuario que administre el contenido de la web.

Para crear un usuario para acceder al mismo se hace:

python .\manage.py createsuperuser

aquí se ingresa usuario, email (cualquiera), password y se confirma.

Una vez realizado se puede acceder al panel y revisar varias opciones.

Para integrar el modelo de clase creada en el panel admin es necesario modificar el archivo **admin.py** en la carpeta de la nueva **app personas**

Se agrega la línea:

admin.site.register(Persona)

importada de

from .models import Persona

y ahora cada vez que se abra el panel de administración se puede instanciar esta clase y crear registros en la db.

**CREANDO OTRA CLASE MODELO RELACIONADA**

Para el ejemplo se crea otra clase modelo que se relaciona con Persona. La clase es Domicilio y tiene una relación de clave foránea con la primera.

Para esto, se crea otra clase en el archivo models con las siguientes especificaciones:

class Domicilio(models.Model):

    calle = CharField(max\_length=255, help\_text='ej: san martin', null=1)

    nro\_calle = IntegerField(help\_text='ej: 1757', null=1)

    pais = CharField(max\_length=255, help\_text='ej: arg', null=1)

    def \_\_str\_\_(self) -> str:

        return f'{self.id} | {self.calle} {self.nro\_calle} {self.pais}'

Y es necesario agregar la columna en la clase Persona

    domicilio = ForeignKey(Domicilio, on\_delete=models.SET\_NULL, null=1)

Ahora, cada vez que vayamos a agregar un registro de persona es necesario relacionarle con la clase modelo Domicilio.

Una vez creada la clase y registrada

admin.site.register(Domicilio)

Es necesario hacer:

Python manage.py makemigrations

Y luego Python manage.py migrate