



Tecnológico Nacional de México

Instituto Tecnológico de Nuevo
Laredo



Ingeniería en Sistemas Computacionales

PROGRAMACION MULTIPARADIGMA

Unidad 2

Practica #2.-PostgreSQL, Django y Flask.

Docente: Ing. Luis Daniel Castillo García

Alumnos:

Jesus Antonio Villanueva Hidalgo #18100245

Fernando Angel Lopez Soto #18100194

José Eduardo Gómez Zúñiga #18100184

Nuevo Laredo, Tam.

30 de octubre de 2022

Practica #1

Realizar una aplicación que se conecte a postgresql y contenga lo siguiente

1. Al menos 3 entidades (ejemplo clase de entidad Persona)

Primera entidad, Videojuego

```
TablasPostgres.py X
TablasPostgres.py > Compañía > __str__
1  #DAO
2  class Videojuego:
3      def __init__(self, idJuego=None, titulo=None, genero=None, lanzamiento=None) -> None:
4          self.IDj = idJuego
5          self.titulo = titulo
6          self.genero = genero
7          self.lanzamiento = lanzamiento
8
9      def __str__(self) -> str:
10         return (f'{self.IDj} {self.titulo} {self.genero} {self.lanzamiento}')
11
12     @property
13     def idJuego(self):
14         return self.IDj
15
16     def titulo(self):
17         return self.titulo
18
19     def genero(self):
20         return self.genero
21
22     def lanzamiento(self):
23         return self.lanzamiento
```

Segunda entidad, consola:

```
TablasPostgres.py X
TablasPostgres.py > Compañía > __str__
24  class Consola:
25      def __init__(self, idConsola=None, consola=None, modelo=None, año=None) -> None:
26          self.IDcl = idConsola
27          self.consola = consola
28          self.modelo = modelo
29          self.año = año
30
31      def __str__(self) -> str:
32         return (f'{self.IDcl} {self.consola} {self.modelo} {self.año}')
33
34     @property
35     def idConsola(self):
36         return self.IDcl
37
38     def consola(self):
39         return self.consola
40
41     def modelo(self):
42         return self.modelo
43
44     def año(self):
45         return self.año
```

Y tercera entidad, compañía:

```
TablasPostgres.py X
TablasPostgres.py > Consola
45
46 class Compañia:
47     def __init__(self, idCompañia=None, compañía=None, añoCreacion=None) -> None:
48         self.IDcm = idCompañia
49         self.compañia = compañía
50         self.añoCreacion = añoCreacion
51
52     def __str__(self) -> str:
53         return (f'{self.IDcm} {self.compañia} {self.añoCreacion}')
```

Las tres entidades se encuentran en un mismo archivo con el nombre Tablas Postgres en donde se toma las separaciones por cada tabla

2. Realizar CRUD de las 3 entidades

```
VideojuegosDAO.py X
VideojuegosDAO.py > ...
5
6 class VideojuegosDAO:
7     _SELECCIONAR = "SELECT * FROM videojuego ORDER BY idjuego"
8     _INSERT = "INSERT INTO videojuego(titulo,genero,lanzamiento) VALUES (%s,%s,%s)"
9     _ACTUALIZAR = "UPDATE videojuego SET titulo=%s, genero=%s, lanzamiento=%s WHERE idjuego=%s"
10    _ELIMINAR = "DELETE FROM videojuego WHERE idjuego=%s"
11
12    @classmethod
13    def seleccionar(cls):
14        with CursorDelPool() as cursor:
15            cursor.execute(cls._SELECCIONAR)
16            registros = cursor.fetchall()
17            juegos = []
18            for r in registros:
19                juego = Videojuego(r[0],r[1],r[2],r[3])
20                juegos.append(juego)
21            return juegos
22
23    @classmethod
24    def insertar(cls,juego):
25        with CursorDelPool() as cursor:
26            valores = (juego.titulo, juego.genero, juego.lanzamiento)
27            cursor.execute(cls._INSERT, valores)
28            return cursor.rowcount
29
30    @classmethod
31    def actualizar(cls,juego):
32        with CursorDelPool() as cursor:
33            valores = (juego.titulo, juego.genero, juego.lanzamiento, juego.idJuego)
34            cursor.execute(cls._ACTUALIZAR, valores)
35            return cursor.rowcount
36
37    @classmethod
38    def eliminar(cls,juego):
39        with CursorDelPool() as cursor:
40            valores = (juego.idJuego, )
41            cursor.execute(cls._ELIMINAR, valores)
42            return cursor.rowcount
43
44
```

```

CompañíasDAO.py X
CompañíasDAO.py > CompañíasDAO
5
6 class CompañíasDAO:
7     _SELECCIONAR = "SELECT * FROM compañía ORDER BY idcompañia"
8     _INSERT = "INSERT INTO compañía(compañia, añoCreacion) VALUES (%s,%s)"
9     _ACTUALIZAR = "UPDATE compañía SET compañía=%s, añoCreacion=%s WHERE idcompañia=%s"
10    _ELIMINAR = "DELETE FROM compañía WHERE idcompañia=%s"
11
12    @classmethod
13    def seleccionar(cls):
14        with CursorDelPool() as cursor:
15            cursor.execute(cls._SELECCIONAR)
16            registros = cursor.fetchall()
17            compañías = []
18            for r in registros:
19                compañía = Compañía(r[0], r[1], r[2])
20                compañías.append(compañía)
21            return compañías
22
23    @classmethod
24    def insertar(cls, comp):
25        with CursorDelPool() as cursor:
26            valores = (comp.compañia, comp.añoCreacion)
27            cursor.execute(cls._INSERT, valores)
28            return cursor.rowcount
29
30    @classmethod
31    def actualizar(cls, compañía):
32        with CursorDelPool() as cursor:
33            valores = (compañía.compañia, compañía.añoCreacion, compañía.idCompañía)
34            cursor.execute(cls._ACTUALIZAR, valores)
35            return cursor.rowcount
36
37    @classmethod
38    def eliminar(cls, compañía):
39        with CursorDelPool() as cursor:
40            valores = (compañía.idCompañía, )
41            cursor.execute(cls._ELIMINAR, valores)
42            return cursor.rowcount

```

```

6 class ConsolasDAO:
7     _SELECCIONAR = "SELECT * FROM consola ORDER BY idconsola"
8     _INSERT = "INSERT INTO consola(consola, modelo, año) VALUES (%s,%s,%s)"
9     _ACTUALIZAR = "UPDATE consola SET consola=%s, modelo=%s, año=%s WHERE idcompañia=%s"
10    _ELIMINAR = "DELETE FROM consola WHERE idconsola=%s"
11
12    @classmethod
13    def seleccionar(cls):
14        with CursorDelPool() as cursor:
15            cursor.execute(cls._SELECCIONAR)
16            registros = cursor.fetchall()
17            consolas = []
18            for r in registros:
19                consola = Consola(r[0], r[1], r[2], r[3])
20                consolas.append(consola)
21            return consolas
22
23    @classmethod
24    def insertar(cls, con):
25        with CursorDelPool() as cursor:
26            valores = (con.consola, con.modelo, con.año)
27            cursor.execute(cls._INSERT, valores)
28            return cursor.rowcount
29
30    @classmethod
31    def actualizar(cls, con):
32        with CursorDelPool() as cursor:
33            valores = (con.consola, con.modelo, con.año, con.idconsola)
34            cursor.execute(cls._ACTUALIZAR, valores)
35            return cursor.rowcount
36
37    @classmethod
38    def eliminar(cls, con):
39        with CursorDelPool() as cursor:
40            valores = (con.idconsola, )
41            cursor.execute(cls._ELIMINAR, valores)
42            return cursor.rowcount

```

3. Usar archivo de logs

Archivo creado

```

1 import logging as log
2 log.basicConfig(level=log.DEBUG, #El level son 5, debug, information, , , critico. En otros es mejor usar warning
3                 format="%(asctime)s: %(levelname)s [%(filename)s]: %(lineno)s %(message)s",
4                 datefmt="%I:%M:%S %p",
5                 handlers = [
6                     log.FileHandler('capa_datos.log'),
7                     log.StreamHandler()
8                 ])
9
10
11 if __name__ == '__main__':
12     log.debug("Mensaje Debug")

```

Uso de logs

```

if __name__ == '__main__':

    #Leer
    log.debug(f'Videojuegos en la base de datos: {VideojuegosDAO.seleccionar()}')

    #Insertar
    j1 = Videojuego(titulo = "Castlevania", genero="Plataforma", lanzamiento=1994)
    juegosIn = VideojuegosDAO.insertar(j1)
    log.debug(f'Videojuegos agregados a la base de datos {juegosIn}')

    #Actualizar
    j2 = Videojuego(idJuego=1, titulo="Castlevania", genero="Plataforma", lanzamiento=1994)
    juegosIn = VideojuegosDAO.actualizar(j2)
    log.debug(f'Videojuegos actualizados {juegosIn}')

    #Eliminar
    j3 = Videojuego(idJuego= 3)
    JuegosEl = VideojuegosDAO.eliminar(j3)
    log.debug(f'Juegos eliminados {JuegosEl}')

```

Conexion.py X

Conexion.py > Conexion > obtenerConexion

```

33
34     @classmethod
35     def obtenerConexion(cls):
36         conexion = cls.obtenerPool().getconn()
37         log.debug(f'Conexion obtenida: {conexion}')
38         return conexion
39
40     @classmethod
41     def liberarConexion(cls,conexion):
42         cls.obtenerPool().putconn(conexion)
43         log.debug(f'Conexion regresada: {conexion}')
44

```

4. Utilizar pool de conexiones

Archivo de pool creado

```

cursorPool.py X
cursorPool.py > ...
1  from logger_base import log
2  from Conexion import Conexion
3
4  class CursorDelPool:
5      def __init__(self) -> None:
6          self._conexion = None
7          self._cursor = None
8
9      def __enter__(self):
10         log.debug("Inicio metodo with")
11         self._conexion = Conexion.obtenerConexion()
12         self._cursor = self._conexion.cursor()
13         return self._cursor
14
15     def __exit__(self, tipo_excepcion, valor_excepcion, detalle_excepcion):
16         log.debug("Se ejecuta exit")
17         if valor_excepcion:
18             self._conexion.rollback()
19         else:
20             self._conexion.commit()
21         self._cursor.close()
22         Conexion.liberarConexion(self._conexion)
23
24     if __name__ == '__main__':
25         with CursorDelPool() as cursor:
26             log.debug("Dentro del bloque with")
27             cursor.execute("SELECT * FROM videojuego")
28             log.debug(cursor.fetchall())

```

Y su uso

En cada parte del CRUD de los objetos como se vio antes

La realización de la practica fue sencilla y llevada de la mano mediante el código anteriormente otorgado por el maestro por lo que se pudo concretar sin problemas, solamente de crear nuevas instancias y ArchivosDAO de respectivas tablas, el mayor inconveniente es al momento de crear los documentos de las entidades ya que si no poseen el mismo nombre para cada columna dará error de adaptar método.

Practica #2

Realizar una aplicación utilizando el Framework DJANGO y que contenga lo siguiente:

1. Conexión a base de datos postgresql.

```
urls.py  settings.py X
webproject > settings.py > ...
74
75 # Database
76 # https://docs.djangoproject.com/en/4.1/ref/settings/#databases
77
78 DATABASES = {
79     'default': {
80         'ENGINE': 'django.db.backends.postgresql_psycopg2',
81         'NAME': 'practica2', #Database Name
82         'USER': 'postgres',
83         'PASSWORD': 'admin',
84         'HOST': 'localhost',
85         'PORT': '5432'
86     }
87 }
```

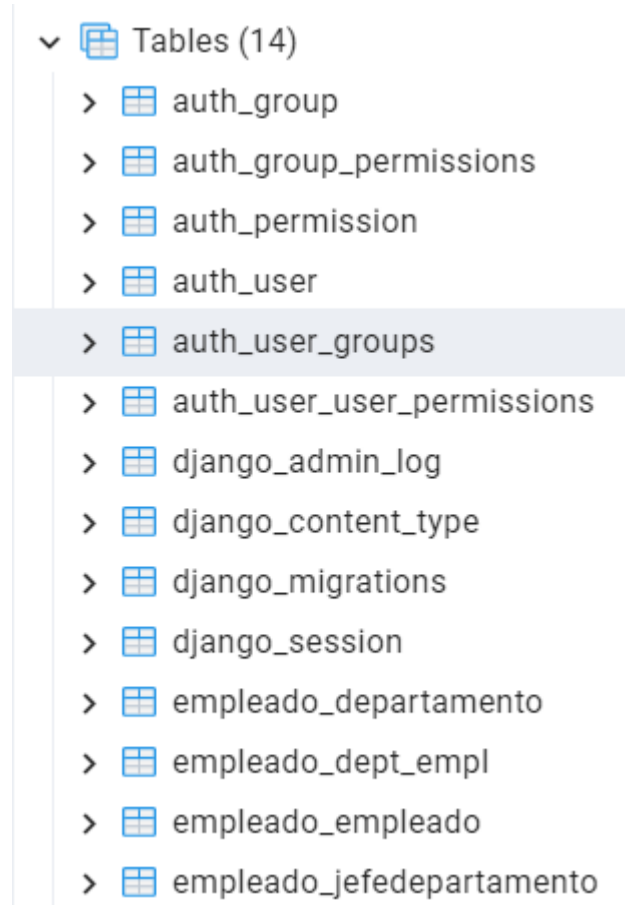
2. Utilizar al menos 4 entidades.

```
urls.py  models.py M X
empleado > models.py > Empleado > __str__
5 class Empleado(models.Model):
6     fecha_nacimiento = models.DateField()
7     nombre = models.CharField(max_length=255)
8     apellido = models.CharField(max_length=255)
9     generos = [('F', 'Femenino'), ('M', 'Masculino')]
10    genero = models.CharField(max_length=1, choices=generos, default='F')
11    fecha_contratacion = models.DateField()
12
13    def __str__(self) -> str:
14        return f'Empleado {self.id}: {self.nombre} {self.apellido}'
15
16 class Departamento(models.Model):
17     nombre = models.CharField(max_length=255)
18
19     def __str__(self) -> str:
20         return f'Departamento {self.id}: {self.nombre}'
21
22 class JefeDepartamento(models.Model):
23     idDepartamento = models.ForeignKey(Departamento, on_delete = models.CASCADE, null=True)
24     idEmpleado = models.ForeignKey(Empleado, on_delete = models.SET_NULL, null=True)
25     fecha_inicio = models.DateField()
26     fecha_fin = models.DateField()
27
28     def __str__(self) -> str:
29         return f'JefeDepartamento {self.id}: {self.idEmpleado.nombre} {self.idEmpleado.apellido}'
30
31 class Dept_empl(models.Model):
32     idEmpleado = models.ForeignKey(Empleado, on_delete = models.SET_NULL, null=True)
33     idDepartamento = models.ForeignKey(Departamento, on_delete = models.SET_NULL, null=True)
34     fecha_inicio = models.DateField()
35     fecha_fin = models.DateField()
36     def __str__(self) -> str:
37         return f'{self.idEmpleado} en {self.idDepartamento}'
```

3. Una de las entidades debe relacionarse con otra.


```
class JefeDepartamento(models.Model):
    idDepartamento = models.ForeignKey(Departamento,on_delete = models.CASCADE,null=True)
    idEmpleado = models.ForeignKey(Empleado,on_delete = models.SET_NULL,null=True)
    fecha_inicio = models.DateField()
    fecha_fin = models.DateField()
```

4. Realizar migraciones.



5. CRUD de las 4 entidades utilizando templates.

A) EMPLEADO

DEPARTAMENTOS
JEFES DE DEPARTAMENTO
RELACION EMPLEADO-DEPARTAMENTO

EMPLEADOS

[Agregar Empleado](#)

Id	Nombre	Genero	Fecha Nacimiento	Fecha Contratacion			
1	Jesus Antonio Villanueva	M	Dec. 31, 1999	Aug. 24, 2022	Ver Detalle	Editar Empleado	Eliminar Empleado
2	Yessica Gutierrez	F	Feb. 21, 2000	Oct. 29, 2022	Ver Detalle	Editar Empleado	Eliminar Empleado
3	Juan Perez	M	Nov. 26, 1986	Oct. 29, 2022	Ver Detalle	Editar Empleado	Eliminar Empleado
4	Maria Otoñel	F	Sept. 3, 1996	Jan. 1, 2022	Ver Detalle	Editar Empleado	Eliminar Empleado

B) DEPARTAMENTOS

Editar Departamento

Nombre:

[Regresar al inicio](#)

C) JEFES DE DEPARTAMENTO

2 Empleado 4: Maria Otoñel

[VOLVER](#)

D) RELACION EMPLEADO-DEPARTAMENTO

Agregar Relacion Empleado Departamento

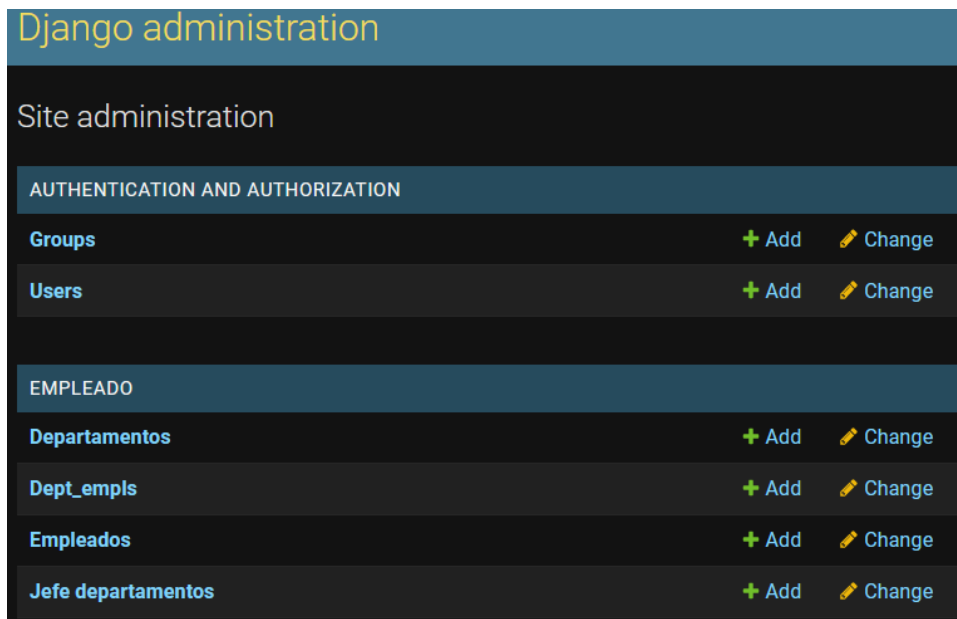
IdEmpleado:

IdDepartamento:

Fecha inicio:

Fecha fin:

6. Agregar las 4 entidades a la página de administración.



Explicación del resultado

El funcionamiento del framework Django consiste en que desde el navegador hacemos una petición HTTP va hacia las URLS del proyecto, este revisa dentro de las VISTAS creadas, luego consulta los MODELOS, por ejemplo, el modelo Empleado.

Lo siguiente es consultar la BASE de DATOS y devuelve la información al MODELO, este a la VISTA y por último devuelve o renderiza un TEMPLATE hacia el navegador del usuario.

Para este ejemplo podemos observar 4 capturas de 4 templates distintos, uno por cada entidad creada y a su vez 3 TEMPLATES por cada uno los cuales son de: agregar, editar y detalle.

Practica #3

Realizar una aplicación utilizando Framework Flask

1. Conexión a base de datos postgresql con SQLAlchemy

Create - Table

General

Columns

Advanced

Constraints

Partitions

Parameters

Security

SQL

Inherited from table(s)

Select to inherit from...

Columns

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	ID	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
	Nombre	character varying			<input type="checkbox"/>	<input type="checkbox"/>	
	Ubicacion	character varying			<input type="checkbox"/>	<input type="checkbox"/>	
	Empleados	integer			<input type="checkbox"/>	<input type="checkbox"/>	

X Close

Reset

Save

Tabla de Libro:

Create - Table

General

Columns

Advanced

Constraints

Partitions

Parameters

Security

SQL

Inherited from table(s)

Select to inherit from...

Columns

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	ID	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
	Titulo	character varying			<input type="checkbox"/>	<input type="checkbox"/>	
	Autor	character varying			<input type="checkbox"/>	<input type="checkbox"/>	
	Paginas	integer			<input type="checkbox"/>	<input type="checkbox"/>	

X Close

Reset

Save







Tabla de Autor:



Create - Table

General Columns Advanced Constraints Partitions Parameters Security SQL

Inherited from table(s)

Columns +

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
 	<input type="text" value="ID"/>	<input type="text" value="integer"/> ▼			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text"/>
 	<input type="text" value="Nombre"/>	<input type="text" value="character varying"/> ▼	<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>
 	<input type="text" value="Apellido"/>	<input type="text" value="character varying"/> ▼	<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>

Conexión a la BD

```

database.py X
database.py > ...
1  from flask_sqlalchemy import SQLAlchemy
2
3  db = SQLAlchemy()
```

Configuración para la conexión a la BD

```

#Configuracion de la base de datos
USER_DB = 'postgres'
PASS_DB = 'admin'
URL_DB = 'localhost'
NAME_DB = 'Unidad2'
FULL_URL_DB = f'postgresql://{USER_DB}:{PASS_DB}@{URL_DB}/{NAME_DB}'

app.config['SQLALCHEMY_DATABASE_URI'] = FULL_URL_DB
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db.init_app(app)
```

2. Al menos un formulario con estilo css

3. Utilizar app logging

```
app.py > ...
1  import logging
2  from flask import Flask,request,url_for,render_template,redirect
3  from database import db
4  from flask_migrate import Migrate
5  from models import Libreria, Libro, Autor
6  from forms import LibreriaForm, LibroForm, AutorForm
7
8  app = Flask(__name__)
9
10 logging.basicConfig(filename='error.log',level=logging.DEBUG)
```

```
app.py  error.log  Libreria.html  index.html
error.log
1  INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
2  * Running on http://127.0.0.1:5000
3  INFO:werkzeug:Press CTRL+C to quit
4  INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
5  * Running on http://127.0.0.1:5000
6  INFO:werkzeug:Press CTRL+C to quit
7  INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
8  * Running on http://127.0.0.1:5000
9  INFO:werkzeug:Press CTRL+C to quit
10 ERROR:flask_migrate:Error: Directory migrations already exists and is not empty
11 INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
12 * Running on http://127.0.0.1:5000
13 INFO:werkzeug:Press CTRL+C to quit
14 INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
15 * Running on http://127.0.0.1:5000
16 INFO:werkzeug:Press CTRL+C to quit
17 INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
18 * Running on http://127.0.0.1:5000
19 INFO:werkzeug:Press CTRL+C to quit
20
```

se le dio el nombre de “error” al archivo de log.

4. Utilizar al menos 3 entidades

Se utilizaron 3 entidades, Librería, Libro y Autor, para cada una se creó el Modelo correspondiente.

```
class Libreria(db.Model):
    ID = db.Column(db.Integer,primary_key = True)
    Nombre = db.Column(db.String(250))
    Ubicacion = db.Column(db.String(250))
    Empleados = db.Column(db.String(250))

    def __str__(self) -> str:
        return (f'ID: {self.id}, '
                f'Nombre: {self.nombre}, '
                f'Ubicacion: {self.apellido}, '
                f'Empleados: {self.telefono}')
```

```
class Libro(db.Model):
    ID=db.Column(db.Integer,primary_key=True)
    Titulo=db.Column(db.String(250))
    Autor=db.Column(db.String(250))
    Paginas=db.Column(db.String(250))

    def __str__(self) -> str:
        return (f'ID: {self.id}, '
                f'Titulo: {self.numero}, '
                f'Autor: {self.placas}, '
                f'Paginas: {self.marca}')
```

```
class Autor(db.Model):
    ID=db.Column(db.Integer,primary_key=True)
    Nombre=db.Column(db.String(250))
    Apellido=db.Column(db.String(250))

    def __str__(self) -> str:
        return (f'ID: {self.id}, '
                f'Numero: {self.numero}, '
                f'Placas: {self.placas}')
```

5. CRUD con pantallas de 2 entidades

6. CRUD de al menos una entidad utilizando peticiones HTTP

7. Utilizar migraciones

Se hicieron las migraciones y esto se demuestra ya que se generó la carpeta migrations:

migrations > versions > 5a0fce7aae7a_py > ...

```
19 def upgrade():
20     # ### commands auto generated by Alembic - please adjust! ###
21     op.create_table('autor',
22         sa.Column('ID', sa.Integer(), nullable=False),
23         sa.Column('Nombre', sa.String(length=250), nullable=True),
24         sa.Column('Apellido', sa.String(length=250), nullable=True),
25         sa.PrimaryKeyConstraint('ID')
26     )
27     op.create_table('libreria',
28         sa.Column('ID', sa.Integer(), nullable=False),
29         sa.Column('Nombre', sa.String(length=250), nullable=True),
30         sa.Column('Ubicacion', sa.String(length=250), nullable=True),
31         sa.Column('Empleados', sa.String(length=250), nullable=True),
32         sa.PrimaryKeyConstraint('ID')
33     )
34     op.create_table('libro',
35         sa.Column('ID', sa.Integer(), nullable=False),
36         sa.Column('Titulo', sa.String(length=250), nullable=True),
37         sa.Column('Autor', sa.String(length=250), nullable=True),
38         sa.Column('Paginas', sa.String(length=250), nullable=True),
39         sa.PrimaryKeyConstraint('ID')
40     )
41     op.drop_table('Libro')
42     op.drop_table('Autor')
43     op.drop_table('Libreria')
44     # ### end Alembic commands ###
45
46
47 def downgrade():
48     # ### commands auto generated by Alembic - please adjust! ###
49     op.create_table('Libreria',
50         sa.Column('ID', sa.INTEGER(), autoincrement=False, nullable=False),
51         sa.Column('Nombre', sa.VARCHAR(), autoincrement=False, nullable=True),
52         sa.Column('Ubicacion', sa.VARCHAR(), autoincrement=False, nullable=True),
53         sa.Column('Empleados', sa.INTEGER(), autoincrement=False, nullable=True),
54         sa.PrimaryKeyConstraint('ID', name='Libreria_pkey')
55     )
56     op.create_table('Autor',
57         sa.Column('ID', sa.INTEGER(), autoincrement=False, nullable=False),
```

Comprobación.

Explicación del resultado.

Enlace a repositorio por equipo (GitHub)

https://github.com/18100194-FernandoALS/Practicas_Multiparadigmas.git

Conclusiones y comentarios

Un Framework es un entorno de trabajo que incluye ciertas características y herramientas que nos permiten desarrollar un proyecto de una manera más rápida. Podemos ir integrando paquetes conforme los vayamos necesitando.

Lo anterior mencionado lo pudimos observar durante el desarrollo de las prácticas, ya que el crear las clases y verificar las consultas con la BD de la práctica 1, fue más tardado y complicado que realizar los modelos y vistas utilizando el framework Django.

A su vez, utilizar el framework Flask nos facilitó el desarrollo del proyecto ya fue más rápido que utilizar Django.

Para ir desarrollando las prácticas necesitamos conocimiento de la practica anterior, por ejemplo, en cada una de las practicas utilizamos bases de datos PostgreSQL. Y el conocer el cómo se hace la conexión en la práctica 1, facilitó un poco las siguientes.