<codoacodo/>

Agencia de Aprendizaje a lo largo de la vida

DJANGO
Reunión 22

Models - 2

BA

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 21

Django: Models - 1

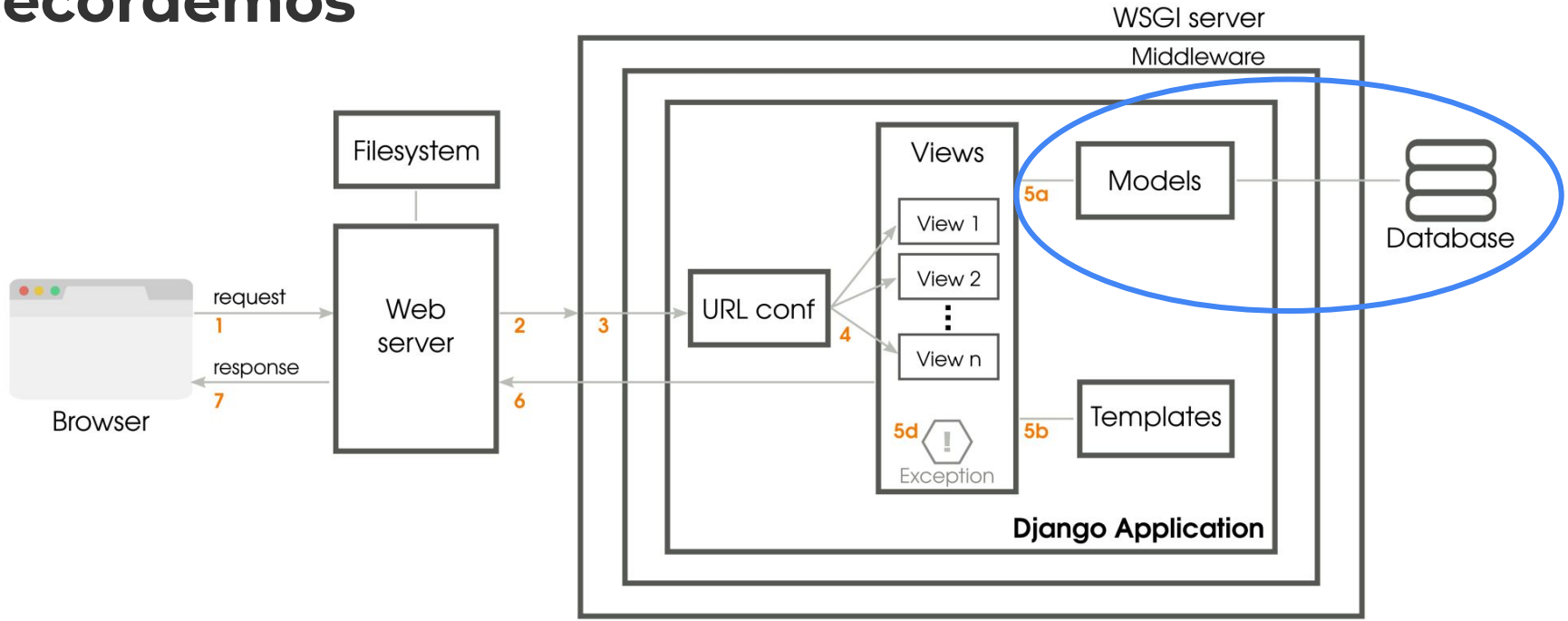
- Crear modelos
- Campos y atributos (DjangoFields)
- Migraciones de modelos (makemigrations, migrate)
- Django Shell

Clase 22

Django: Models - 2

- Relaciones entre modelos
- Herencia de modelos
- QuerySets (lazyloading)

Recordemos


WSGI server
Middleware
Filesystem
Views
Models
Database
View 1
View 2
View n
5a
request
1
response
7
Web
server
2
3
URL conf
4
6
5d
Exception
5b
Templates
Browser
Django Application

¿Qué tipo de relaciones hay?

Django ofrece formas de definir los tres tipos más comunes de relaciones de base de datos: muchos a uno, muchos a muchos y uno a uno. En lo que respecta a las relaciones de uno a uno, no solo podremos definirlas como una relación, sino también utilizando la técnica de herencia.

Relaciones de muchos a uno

Una categoría puede tener muchos cursos

```python
class Categoria(models.Model):
    nombre = models.CharField(max_length=50, verbose_name='Nombre')
    baja = models.BooleanField(default=False)


class Curso(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    descripcion = models.TextField(null=True, verbose_name='Descripcion')
    fecha_inicio = models.DateField(verbose_name='Fecha de inicio')
    portada = models.ImageField(upload_to='imagenes/', null=True, verbose_name='Portada')
    categoria = models.ForeignKey(Categoria, on_delete=models.CASCADE)
```

Puede haber relaciones que referencien a la misma clase

Relaciones de muchos a muchos

Un curso tiene muchos estudiantes y un estudiante puede estar en varios cursos

```python
class Estudiante(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    apellido = models.CharField(max_length=150, verbose_name='Apellido')
    email = models.EmailField(max_length=150, verbose_name='Email')
    dni = models.IntegerField(verbose_name='DNI')

class Curso(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    descripcion = models.TextField(null=True, verbose_name='Descripcion')
    fecha_inicio = models.DateField(verbose_name='Fecha de inicio')
    portada = models.ImageField(upload_to='imagenes/', null=True, verbose_name='Portada')
    estudiantes = models.ManyToManyField(Estudiante)
```

No importa que modelo tenga el ManyToManyField, pero solo debe estar en uno.

Relaciones de muchos a muchos

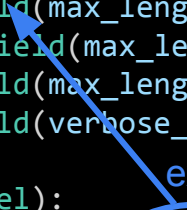
Un curso tiene muchos estudiantes y un estudiante puede estar en varios cursos y se desea registrar información sobre esa relación (la inscripción)

```python
class Estudiante(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    apellido = models.CharField(max_length=150, verbose_name='Apellido')
    email = models.EmailField(max_length=150, verbose_name='Email')
    dni = models.IntegerField(verbose_name='DNI')

class Curso(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    descripcion = models.TextField(null=True, verbose_name='Descripcion')
    fecha_inicio = models.DateField(verbose_name='Fecha de inicio')
    portada = models.ImageField(upload_to='imagenes/', null=True, verbose_name='Portada')
    estudiantes = models.ManyToManyField(Estudiante, through='Inscripcion')

class Inscripcion(models.Model):
    estudiante = models.ForeignKey(Estudiante, on_delete=models.CASCADE)
    curso = models.ForeignKey(Curso, on_delete=models.CASCADE)
    fecha = models.DateField()
```

Relaciones de uno a uno

Un estudiante extiende a una persona

```
class Persona(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    apellido = models.CharField(max_length=150, verbose_name='Apellido')
    email = models.EmailField(max_length=150, verbose_name='Email')
    dni = models.IntegerField(verbose_name='DNI')

class Estudiante(models.Model):
    persona = models.OneToOneField(Persona, on_delete=models.CASCADE, primary_key=True)
    legajo = models.CharField(max_length=100, verbose_name='Legajo')
```



Relaciones de uno a uno

Un estudiante extiende a una persona

```
python .\manage.py shell
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from cac.models import *
>>> p1 = Persona(nombre="Juan", apellido="Perez", email="mail@mail.com", dni=123)
>>> p1.save()
>>> estudiante = Estudiante(persona=p1, legajo="LN001")
>>> estudiante.save()
```

Relaciones de uno a uno (herencia)

Estilo de herencia	Pros	Cons
Sin herencia: si los modelos tienen un campo en común, poner ese campo en ambos modelos	Hace mas simple entender de un vistazo como los modelos asignan a las tablas.	Si hay muchos campos duplicados en el modelo puede ser difícil de mantener.
Clases base abstractas: las tablas son solo creadas para las clases derivadas.	Tener los campos en común en una clase base nos ahorra de escribir los campos mas de una vez. No tenemos la sobrecarga de tablas extra y joins que suceden con la herencia de múltiples tablas.	No podemos usar la clase base de manera independiente.
Herencia de múltiples tablas: se crean tablas tanto para la superclase como las subclases. Un campo implícito OneToOneField asocia la base al hijo.	Nos crea una tabla por cada clase por lo que podemos realizar consultas tanto de la superclase como las subclases. Nos da la posibilidad de obtener la subclase desde la superclase usando: parent.child	Agrega una sustancial sobrecarga ya que cada query en una subtabla requiere un join con la tabla base. Se recomienda enormemente tratar de evitar la herencia de múltiples tablas.
Modelos proxy: es creada una sola tabla por el modelo original.	No permite tener un alias de un modelo con diferentes comportamientos en Python..	No podemos cambiar los campos del modelo.

Clase base abstracta

```python
class Persona(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    apellido = models.CharField(max_length=150, verbose_name='Apellido')
    email = models.EmailField(max_length=150, verbose_name='Email')
    dni = models.IntegerField(verbose_name='DNI')

    class Meta:
        abstract = True

class Estudiante(Persona):
    legajo = models.CharField(max_length=100, verbose_name='Legajo')

class Docente(Persona):
    cuit = models.CharField(max_length=100, verbose_name='Cuit')
```

No es la clase abstracta de python (ABC)

Herencia de múltiples tablas

```python
class Persona(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    apellido = models.CharField(max_length=150, verbose_name='Apellido')
    email = models.EmailField(max_length=150, verbose_name='Email')
    dni = models.IntegerField(verbose_name='DNI')

class Estudiante(Persona):
    legajo = models.CharField(max_length=100, verbose_name='Legajo')

class Docente(Persona):
    cuit = models.CharField(max_length=100, verbose_name='Cuit')
```

Modelos proxy

```python
class Persona(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    apellido = models.CharField(max_length=150, verbose_name='Apellido')
    email = models.EmailField(max_length=150, verbose_name='Email')
    dni = models.IntegerField(verbose_name='DNI')

class SuperPersona(Persona):
    class Meta:
        proxy = True

    def volar(self):
        print("Vuelooooo")
```

¿Cómo interactúo con la BD?

Django brinda automáticamente una API de abstracción de base de datos que permite crear, recuperar, actualizar y eliminar objetos. Para la recuperación de objetos de la base de datos, se utiliza un QuerySet vía un Manager de la clase de Modelo. Un QuerySet representa una colección de objetos de la base de datos.

CRUD objetos

```
class Persona(models.Model):
    nombre = models.CharField(max_length=100, verbose_name='Nombre')
    apellido = models.CharField(max_length=150, verbose_name='Apellido')
```

Crear

```
>>> lucas = Persona(nombre="Lucas", apellido="Pratto")
>>> lucas.save()
```

Actualizar

```
>>> lucas.nombre = "Luquitas"
>>> lucas.save()
```

Eliminar

```
>>> lucas.delete()
(1, {'cac.Persona': 1})
```

CRUD objetos

```
>>> lucas = Persona(nombre="Lucas", apellido="Pratto")
>>> lucas.save()
>>> luquitas = Persona(nombre="Lucas", apellido="Janson")
>>> luquitas.save()
```

QuerySets son LAZY

Obtener

```
>>> Persona.objects.all()
<QuerySet [<Persona: Persona object (2)>, <Persona: Persona object (3)>]>
>>> Persona.objects.get(pk=2)
<Persona: Persona object (2)>
>>> Persona.objects.filter(apellido="Pratto")
<QuerySet [<Persona: Persona object (2)>]>
>>> Persona.objects.filter(apellido="Janson")
<QuerySet [<Persona: Persona object (3)>]>
>>> Persona.objects.filter(apellido="Janson").first()
<Persona: Persona object (3)>
```

No te olvides de completar la asistencia y consultar dudas

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.

TODO EN EL AULA VIRTUAL