Uso de TextChoices y relaciones entre modelos en Django

Este tutorial está diseñado para explicar con claridad y profundidad dos aspectos importantes de Django:

1. El uso de `TextChoices` para manejar elecciones en campos de modelo.
2. Las diferentes relaciones entre modelos: `ForeignKey`, `OneToOneField` y `ManyToManyField`.

Parte 1: Uso de TextChoices en Django

¿Qué es `TextChoices`?

`TextChoices` es una subclase de `models.TextChoices` que permite definir opciones de manera legible y organizada para campos tipo `CharField` en modelos de Django.

Ventajas:

- Evita valores "mágicos" (hardcoded strings).
- Mejora la legibilidad y mantenibilidad.
- Permite acceder al nombre legible con `get_<field>_display()`.

Ejemplo detallado:

```
from django.db import models

class TipoPapeleta(models.TextChoices):
    Nazareno = 'Nazareno', 'Nazareno'
    Acolito = 'Acolito', 'Acólito'
    Musico = 'Musico', 'Musico'
    Insignia = 'Insignia', 'Insignia'
```

```
    Generica = 'Generica', 'Generica'

class PapeletaSitio(models.Model):
    codigo = models.CharField(max_length=15)
    tipo = models.CharField(
        max_length=20,
        choices=TipoPapeleta.choices,
        default=TipoPapeleta.Generica
    )
```

Creación y uso en vistas o shell:

```
# Crear una papeleta
p = PapeletaSitio.objects.create(codigo="001",
tipo=TipoPapeleta.Nazareno)

# Comparar el valor
if p.tipo == TipoPapeleta.Nazareno:
    print("Es un nazareno")

# Mostrar el nombre legible
print(p.get_tipo_display())  # Muestra: Nazareno
```

En templates:

```
<p>Tipo: {{ papeleta.get_tipo_display }}</p>
```

Parte 2: Relaciones entre modelos en Django

1. Relación Muchos a Uno: `ForeignKey`

Un ejemplo clásico: muchos libros pueden estar escritos por un mismo autor.

```
class Autor(models.Model):
    nombre = models.CharField(max_length=100)

class Libro(models.Model):
    titulo = models.CharField(max_length=100)
    autor = models.ForeignKey(Autor, on_delete=models.CASCADE)
```

Uso:

```
autor = Autor.objects.create(nombre="Cervantes")
libro = Libro.objects.create(titulo="Don Quijote", autor=autor)

# Acceso directo
print(libro.autor.nombre)  # Cervantes

# Acceso inverso
print(autor.libro_set.all())  # Lista de libros escritos por Cervantes
```

`on_delete` opciones comunes:

- `CASCADE`: borra los hijos si se borra el padre.
- `SET_NULL`: pone a NULL el campo si se borra el padre (requiere `null=True`).

---

2. Relación Uno a Uno: `OneToOneField`

Común para extender modelos como `User` de Django:

```
from django.contrib.auth.models import User

class Perfil(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    telefono = models.CharField(max_length=20)
```

Uso:

```
usuario = User.objects.first()
perfil = Perfil.objects.create(user=usuario, telefono="123456789")

print(perfil.user.username)
print(usuario.perfil.telefono)
```

---

3. Relación Muchos a Muchos: `ManyToManyField`

Ejemplo: estudiantes y cursos. Un estudiante puede estar en varios cursos, y un curso tener varios estudiantes.

```
class Curso(models.Model):
    nombre = models.CharField(max_length=100)

class Estudiante(models.Model):
    nombre = models.CharField(max_length=100)
    cursos = models.ManyToManyField(Curso)
```

Uso:

```
curso1 = Curso.objects.create(nombre="Historia")
curso2 = Curso.objects.create(nombre="Física")
estudiante = Estudiante.objects.create(nombre="Juan")

# Asignar cursos
estudiante.cursos.add(curso1, curso2)

# Ver cursos del estudiante
print(estudiante.cursos.all())

# Ver estudiantes del curso
print(curso1.estudiante_set.all())
```

Recomendaciones adicionales:

- Usar `related_name` para personalizar el acceso inverso.

```
class Libro(models.Model):
    autor = models.ForeignKey(Autor, on_delete=models.CASCADE,
related_name='libros')
```

Esto permite:

autor.libros.all()

en lugar de:

autor.libro_set.all()