¿Qué aprenderemos hoy?

Hoy exploraremos el concepto de relaciones uno a uno (1:1) en bases de datos usando Django.

Entenderás qué son, cuándo usarlas y cómo implementarlas paso a paso.

Trabajaremos con el proyecto *bikeshop*, añadiendo modelos de Cliente y PerfilCliente.

Aplicaremos teoría y práctica con ejemplos reales, ejercicios y una mini actividad final.

Metodología activa: 💬 Pregunta inicial: "¿Se les ocurre un ejemplo en la vida real donde algo esté relacionado de a uno, y solo a uno?"

¿Qué es una relación uno a uno?

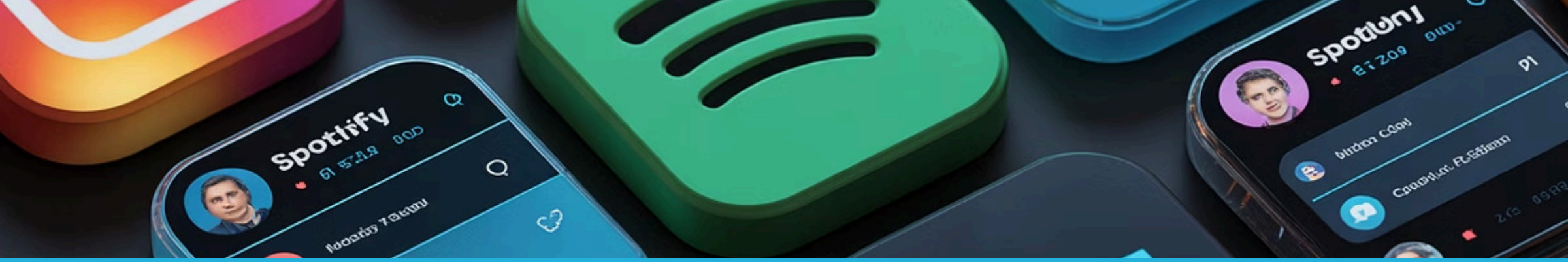
En bases de datos relacionales, una relación 1:1 significa que cada registro en una tabla está vinculado a un único registro en otra tabla.

En Django, se representa con el campo `models.OneToOneField`.

Si un registro intenta tener más de una relación, Django arroja un error de integridad.

Ejemplo visual simple:

Cliente	PerfilCliente
Juan	Perfil de Juan
María	Perfil de María



Cuándo usar relaciones uno a uno

Usa una relación 1:1 cuando quieres extender información de un modelo sin modificarlo. Ejemplos prácticos:

Usuario ↔️ Perfil de usuario	**Vehículo ↔️ Certificado de circulación**
Cliente ↔️ Historial crediticio	**Paciente ↔️ Historial médico**

Ejercicio participativo: 💡 Pide a los estudiantes pensar un ejemplo 1:1 en una app que conozcan (Instagram, Spotify, Uber, etc.).

Diferencias con otros tipos de relaciones

Tipo	Descripción	Ejemplo Django
Uno a uno	1 registro A ↔️ 1 registro B	Cliente ↔️ Perfil `OneToOneField`
Uno a muchos	1 A ↔️ varios B	Cliente ↔️ Órdenes `ForeignKey`
Muchos a muchos	varios A ↔️ varios B	Órdenes ↔️ Bicicletas `ManyToManyField`

Ventajas de usar relaciones uno a uno

Organización

Evita modelos con demasiados campos.

Escalabilidad

Permite agregar información sin alterar tablas base.

Reutilización

Extiende modelos existentes sin duplicar lógica.

Flexibilidad

No todos los registros necesitan datos adicionales.

Dinámica: Pregunta al grupo: "¿Por qué creen que separar el perfil del cliente es una buena práctica?"

Contexto en el proyecto bikeshop

En *bikeshop* tenemos el modelo Bicicleta y ahora agregaremos:

Cliente

Datos básicos.

PerfilCliente

Datos opcionales (dirección, teléfono, fecha de nacimiento).

Objetivo práctico: Implementar la relación 1:1 Cliente ↔️ PerfilCliente.

Crear la app clientes

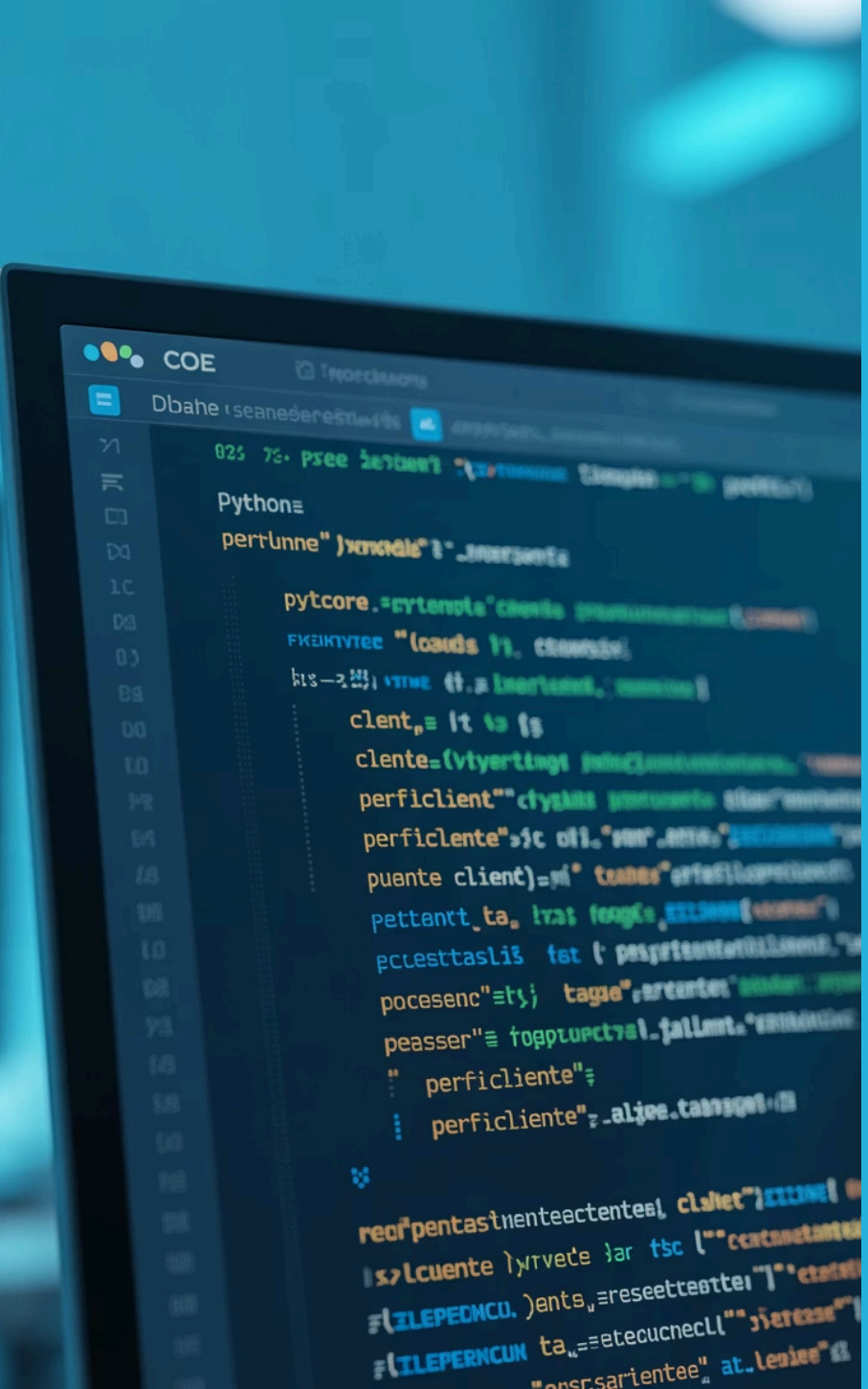
Contenido (paso a paso):

```
python manage.py startapp clientes
```

Luego agregar a `settings.py`:

```
INSTALLED_APPS = [
    ...,
    'clientes',
]
```

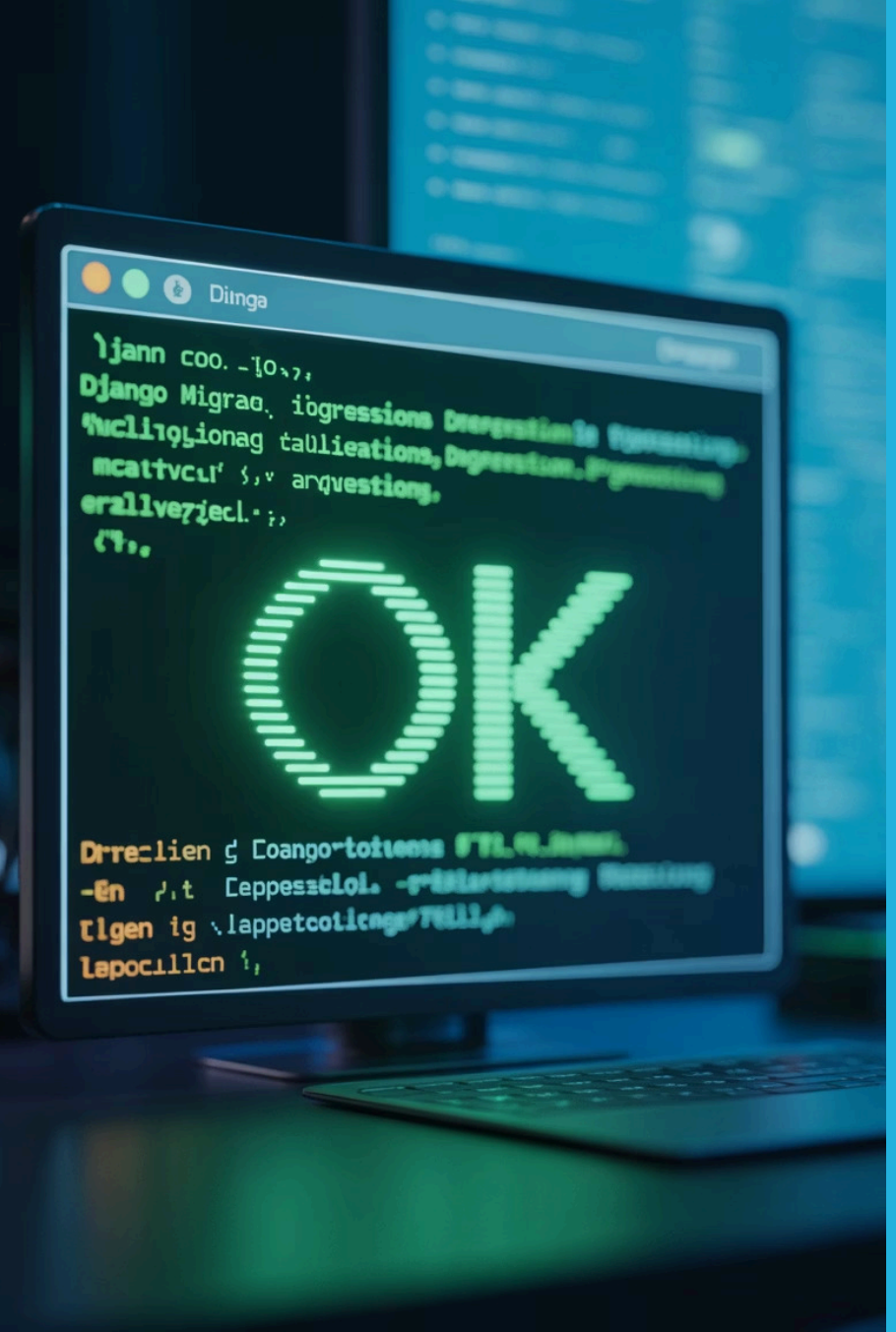
Actividad práctica en VS Code: Cada estudiante crea la app y confirma en `INSTALLED_APPS`.



Definir modelos Cliente y PerfilCliente

```
from django.db import models

class Cliente(models.Model):
    nombre = models.CharField(max_length=100)
    email = models.EmailField(unique=True)

class PerfilCliente(models.Model):
    cliente = models.OneToOneField(
        Cliente,
        on_delete=models.CASCADE,
        related_name='perfil'
    )
    direccion = models.CharField(max_length=200, blank=True, null=True)
    telefono = models.CharField(max_length=20, blank=True, null=True)
    fecha_nacimiento = models.DateField(blank=True, null=True)
```

Explicación: `on_delete=models.CASCADE` → borra el perfil si se elimina el cliente. `related_name='perfil'` → acceso fácil con `cliente.perfil`.

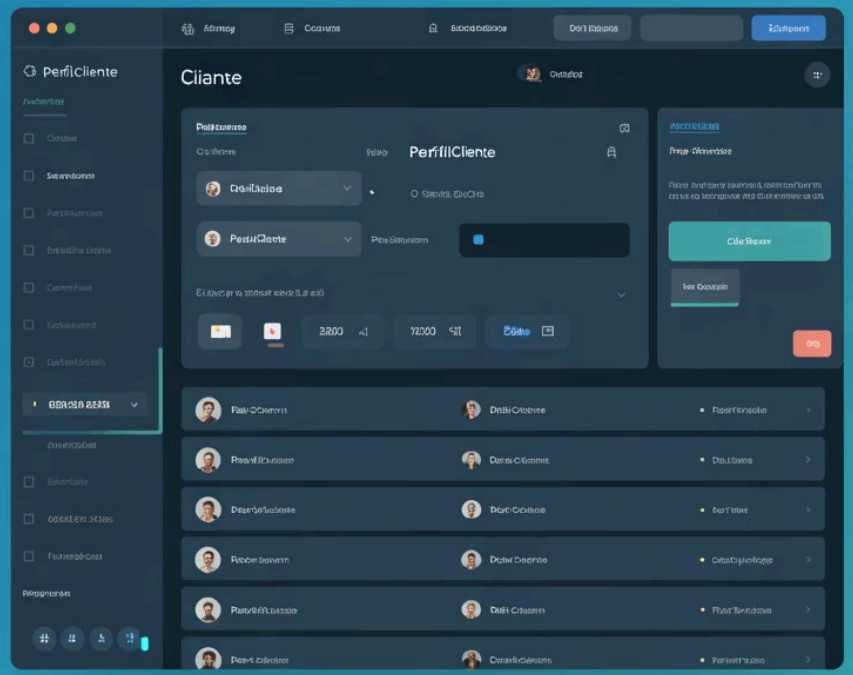


Aplicar migraciones

```
python manage.py makemigrations clientes
python manage.py migrate
```

💡 *Si usas MySQL, asegúrate de que el motor sea InnoDB.*

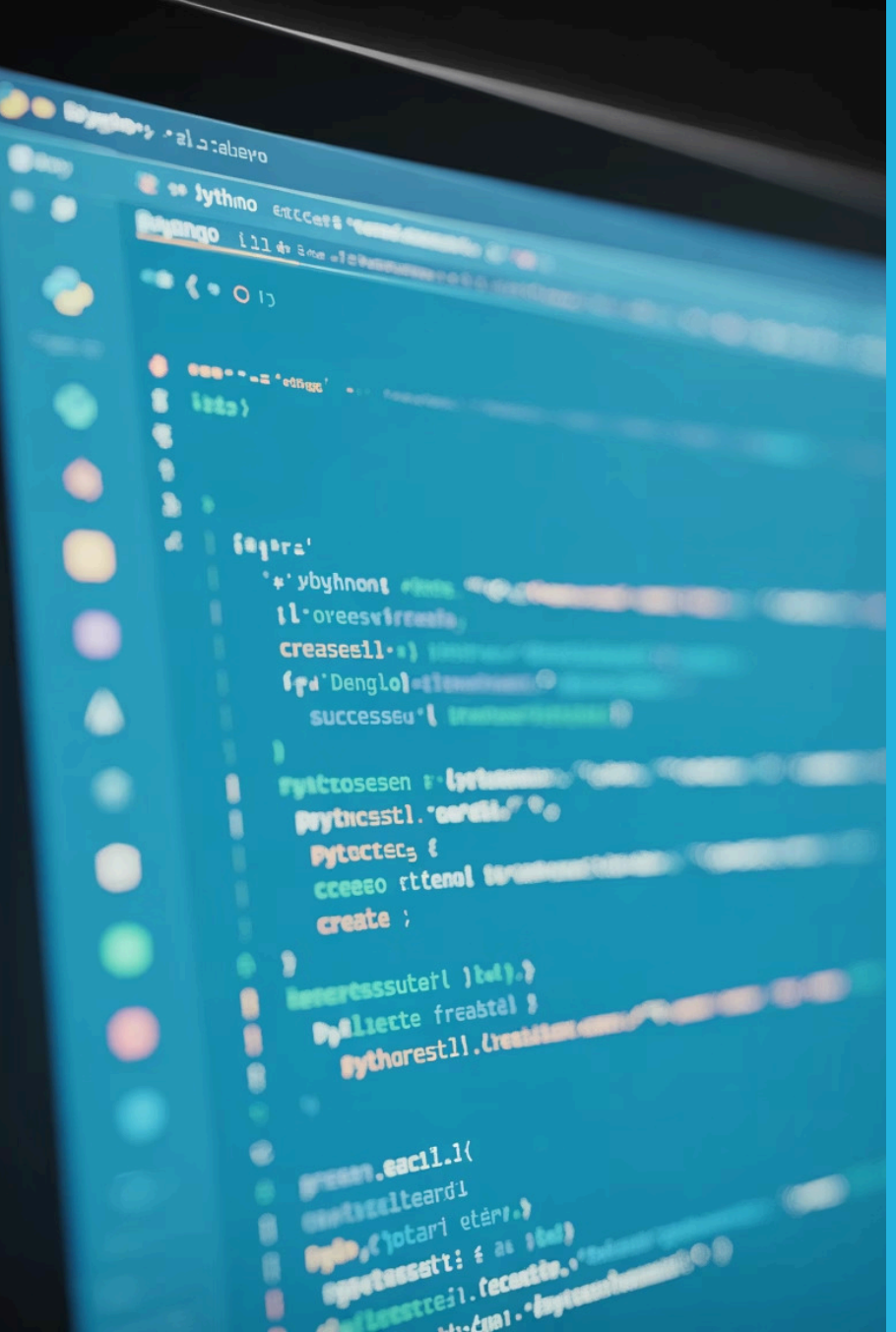
Mini reto: Comprobar que las tablas se crearon correctamente desde MySQL Workbench o `SHOW TABLES;`.



Registrar modelos en el admin

```
from django.contrib import admin
from .models import Cliente, PerfilCliente

class PerfilInline(admin.StackedInline):
    model = PerfilCliente
    can_delete = False

@admin.register(Cliente)
class ClienteAdmin(admin.ModelAdmin):
    inlines = (PerfilInline,)
```

Ventaja didáctica: Permite crear y editar cliente y perfil en una sola vista.



Crear registros en Django Shell

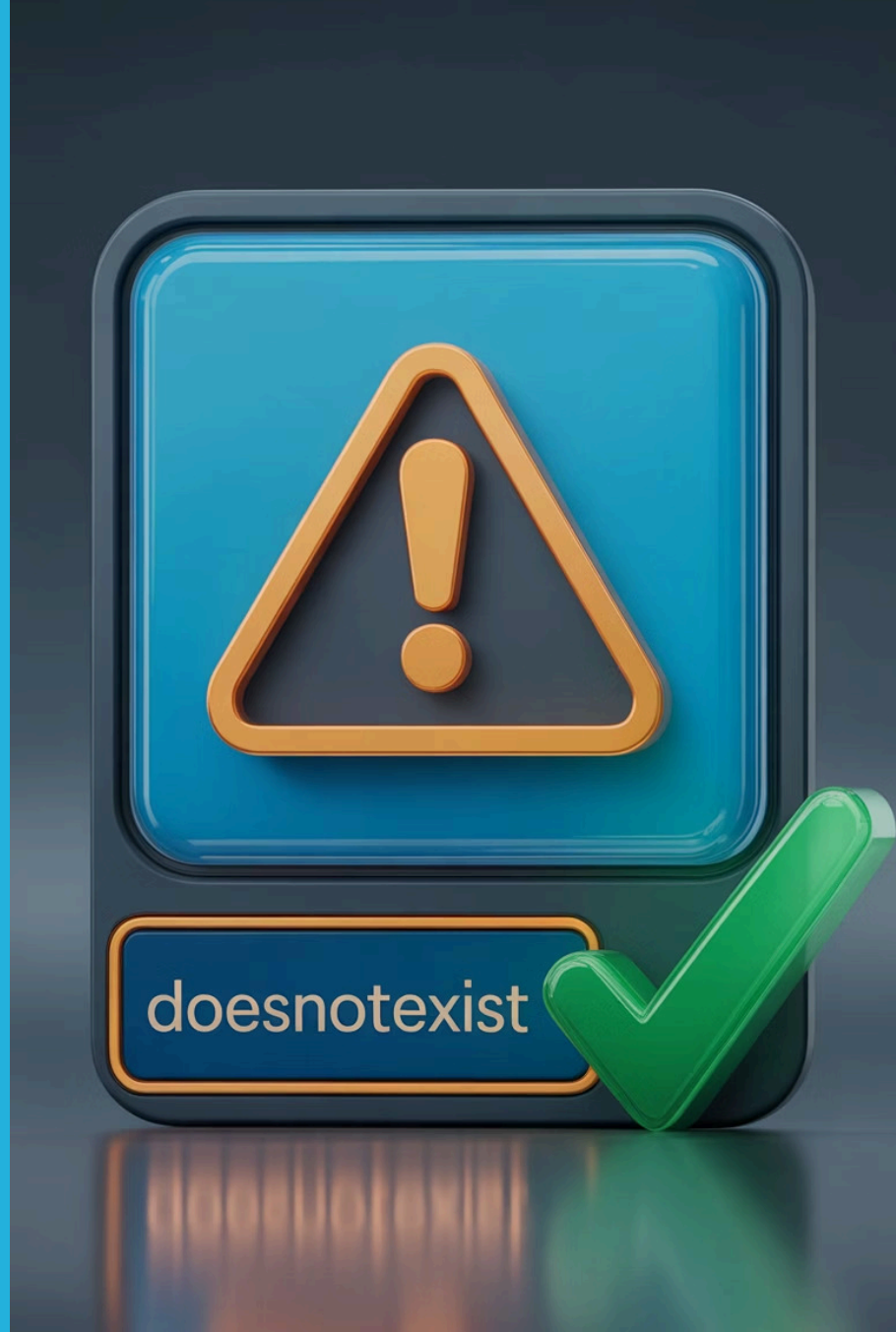
```
from clientes.models import Cliente, PerfilCliente

c = Cliente.objects.create(nombre="Carlos Pérez",
email="carlos@example.com")
PerfilCliente.objects.create(cliente=c, direccion="Av. Siempreviva 123",
telefono="987654321")
```

Reflexión: Cada perfil pertenece a un solo cliente, y cada cliente tiene solo un perfil.

Acceso inverso y manejo de errores

```
try:
    p = c.perfil
except PerfilCliente.DoesNotExist:
    print("El cliente aún no tiene perfil")
```

✅ Usar `hasattr(c, 'perfil')` antes de acceder. Evita errores si el perfil aún no existe.



Consultas optimizadas con select_related

```
clientes = Cliente.objects.select_related('perfil').all()
for c in clientes:
    print(c.nombre, c.perfil.telefono)
```

Beneficio: Django realiza un JOIN y evita múltiples consultas → más rápido y eficiente.

on_delete explicado

Opción	Comportamiento
CASCADE	Elimina perfil con cliente
SET_NULL	Deja perfil con cliente = NULL
PROTECT	Impide borrar cliente con perfil
DO_NOTHING	No hace nada, puede causar error

Ejemplo:

```
cliente = models.OneToOneField(Cliente, on_delete=models.SET_NULL, null=True)
```

Transacciones atómicas

```
from django.db import transaction

with transaction.atomic():
    c = Cliente.objects.create(nombre="María",
email="maria@example.com")
    PerfilCliente.objects.create(cliente=c, telefono="111222333")
```

Explicación: Si una de las operaciones falla, ninguna se guarda.

💡 Ideal para operaciones dependientes.

Automatizar con signals

```
@receiver(post_save, sender=Cliente)
def crear_perfil_cliente(sender, instance, created, **kwargs):
    if created:
        PerfilCliente.objects.create(cliente=instance)
```

Beneficio: El perfil se crea automáticamente al crear un cliente.

Actividad práctica: Implementar esta señal y verificar en admin que el perfil se crea solo.

Ejercicio práctico guiado

Actividad:

1. Crear 3 clientes.
2. Crear sus perfiles.
3. Probar `select_related`.
4. Eliminar un cliente y verificar si su perfil desaparece.
5. Cambiar `on_delete` a `SET_NULL` y volver a probar.

Propósito: Comprender visualmente cómo cambian los datos según la configuración.

Errores comunes

Errores frecuentes:

PerfilCliente.DoesNotExist

Acceso a perfil inexistente.

IntegrityError

Clave duplicada o nula.

Foreign

Error en MySQL por motor incorrecto.

Solución: verificar migraciones, usar try/except, revisar `on_delete`.

Buenas prácticas

- Usar nombres claros en `related_name`.
- Validar emails únicos antes de guardar.
- Evitar campos con acentos o ñ.
- Usar `transaction.atomic()` en operaciones dependientes.
- Registrar PerfilCliente como inline en admin para visualización didáctica.

Cierre y conexión con la próxima clase

✅ **Hoy aprendimos:**

- **Qué es y cuándo usar una relación 1:1.**
- **Cómo implementarla con `OneToOneField`.**
- **Cómo optimizar y proteger nuestros datos.**

➡️ **Próxima clase: Relación Uno a Muchos (Cliente → Orden) — aprenderemos cómo un cliente puede tener múltiples órdenes.**

Cierre activo: 💬 Pregunta final: "¿Qué tipo de relación usarías para vincular un cliente con muchas órdenes?"