Migraciones en Django

Sincronizando el Mundo Real con tu Base de Datos

Objetivo general: Comprender, crear y aplicar migraciones para mantener sincronizados los modelos con la base de datos.

Contexto: Hasta ahora hemos definido modelos y relaciones (uno a uno, uno a muchos, muchos a muchos). Pero... ¿cómo se reflejan esos modelos en la base de datos? 👉 Con migraciones.

Metodología activa: Pregunta inicial: ¿Qué crees que pasaría si cambias un modelo en Django y no actualizas la base de datos?

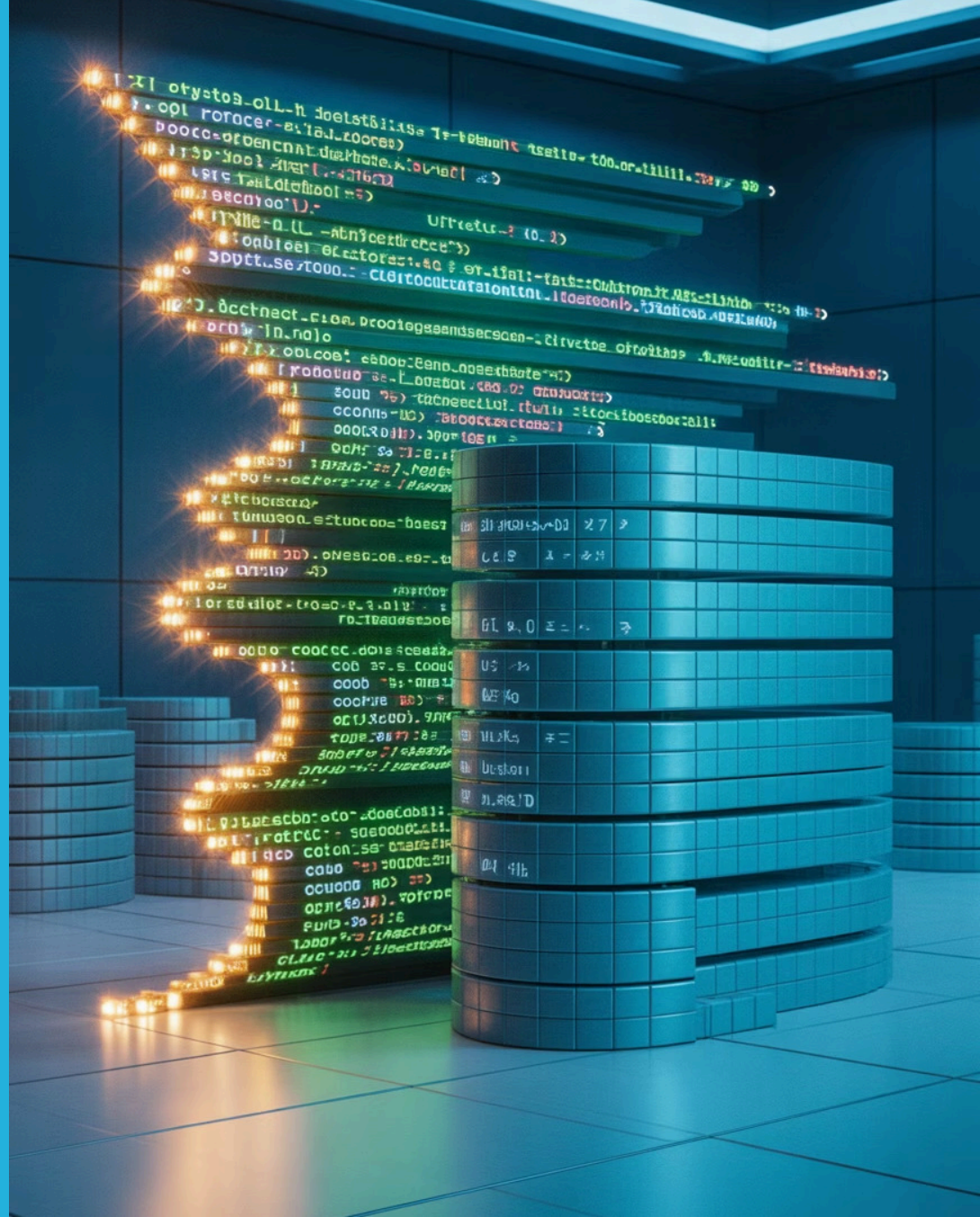
¿Qué es una migración?

Una migración es un conjunto de instrucciones que Django usa para actualizar la base de datos según los cambios en tus modelos.

> 🧠 **"Las migraciones son como un traductor entre tu código y la base de datos".**

Ejemplo simple:

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



Problema que resuelven

Sin migraciones, deberíamos escribir SQL manualmente cada vez que modificamos un modelo 😱. Esto puede causar:

- Inconsistencias
- Errores humanos
- Pérdida de datos

Solución Django:

Automatiza y documenta los cambios, manteniendo sincronía entre modelos y tablas.

Actividad rápida: Explica con tus palabras: ¿Por qué es importante automatizar este proceso?

Cómo funcionan las migraciones

Flujo completo:

Modificas tus modelos

Ejecutas makemigrations

Django genera archivos .py con instrucciones

Ejecutas migrate para aplicar cambios

Archivos de migración

Cada app tiene una carpeta `/migrations/` con archivos como:

```
0001_initial.py
0002_agregar_campo.py
```

Son archivos Python que describen cómo evolucionó tu base de datos.

Ejemplo real (simplificado):

```
# clientes/migrations/0001_initial.py
class Migration(migrations.Migration):
 operations = [
 migrations.CreateModel(
 name='Cliente',
 fields=[
 ('id', models.AutoField(primary_key=True)),
 ('nombre', models.CharField(max_length=100)),
 ],
 ),
 ]
```

Comando makemigrations

Genera las migraciones basadas en los cambios de modelos.

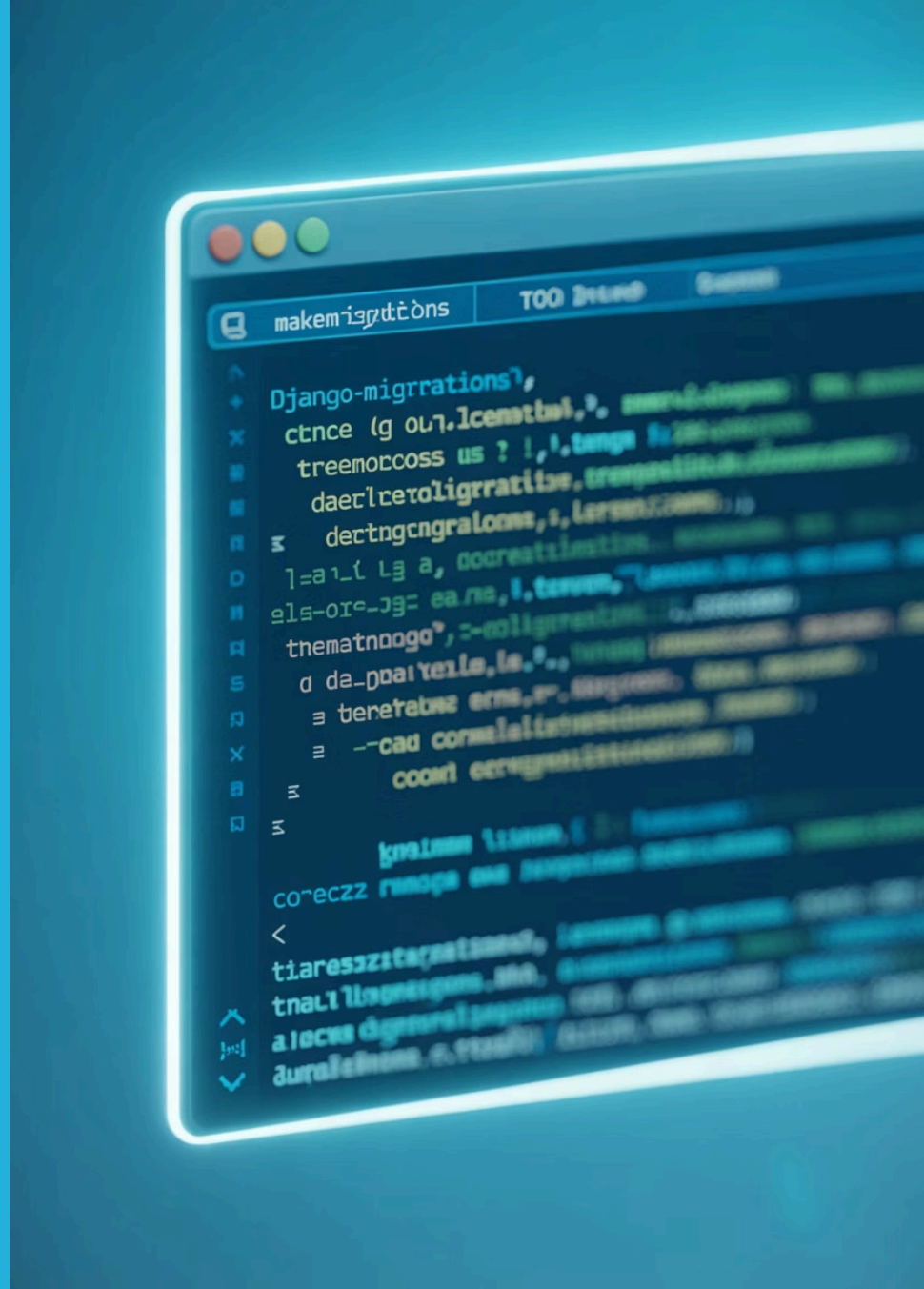
Ejemplo:

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

Salida esperada:

```
Migrations for 'clientes':
  clientes/migrations/0001_initial.py
    - Create model Cliente
```

Actividad: Modifica un modelo en VS Code (agrega un campo "telefono") y genera la migración.



Comando migrate

Aplica los cambios a la base de datos.

```
python manage.py migrate
```

✅ Crea, modifica o elimina tablas según tus migraciones.

Metodología activa: Haz una predicción: ¿Qué pasará si ejecutas migrate sin hacer makemigrations antes?

Visualizando el estado de las migraciones

Para ver cuáles migraciones ya se aplicaron:

```
python manage.py showmigrations
```

Ejemplo de salida:

```
clientes
 [X] 0001_initial
 [ ] 0002_agregar_telefono
```

Ejercicio: Identifica qué migración aún no está aplicada.

Visualizando el SQL

Antes de aplicar una migración puedes ver el SQL real:

```
python manage.py sqlmigrate clientes 0002
```

Salida:

```
ALTER TABLE clientes_cliente ADD COLUMN telefono varchar(20);
```

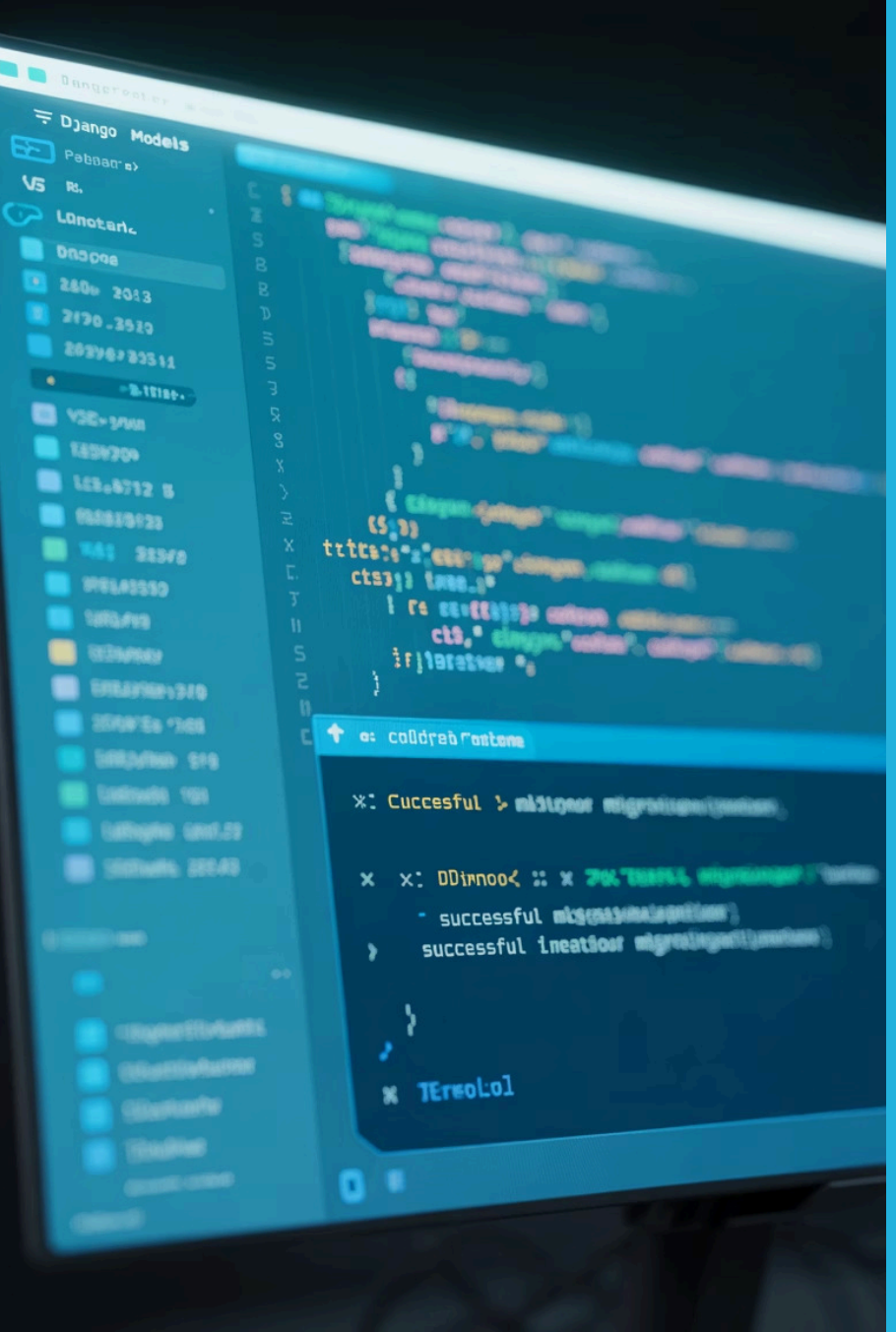
Código Django

```
telefono = models.CharField(
    max_length=20
)
```

SQL Generado

```
ALTER TABLE clientes_cliente
ADD COLUMN telefono varchar(20);
```

Reflexión: Esto es ideal para comprender qué sucede *realmente* en la base de datos.



Ejemplo práctico (Paso 1)

Creamos modelos base

```
# clientes/models.py
class Cliente(models.Model):
    nombre = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
```

Ejecuta:

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

Ejemplo práctico (Paso 2)

Agregamos un campo nuevo:

```
telefono = models.CharField(max_length=20, blank=True, null=True)
```

Volvemos a migrar:

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

1	2	3
Modelo actualizado	**Migración generada**	**Base de datos**
Campo telefono agregado	0002_agregar_telefono.py	Nueva columna creada

Mini desafío: Predice qué hará Django si el campo no permite nulos.

Migraciones en equipo

Cuando varios desarrolladores trabajan en el mismo proyecto:

01

Cada uno crea sus migraciones

02

Al actualizar el repositorio, ejecutan:

```
python manage.py migrate
```

Esto asegura que todos los entornos queden sincronizados.

Migraciones específicas

Puedes aplicar migraciones de una sola app:

```
python manage.py migrate ordenes
```

O regresar una migración:

```
python manage.py migrate clientes 0001
```

⚠️ **Advertencia: Revertir migraciones puede eliminar datos**

Control de versiones y migraciones

Los archivos de migración deben subirse al repositorio (Git) junto al código. Así el equipo:

Mantiene historial

Registro completo de cambios

Sincroniza cambios

Entre todos los entornos

Evita errores

De estructura de datos

Errores comunes

1

Olvidar makemigrations

Antes de ejecutar migrate

2

Modificar sin migrar

Cambiar modelos y no ejecutar migraciones

3

Eliminar migraciones aplicadas

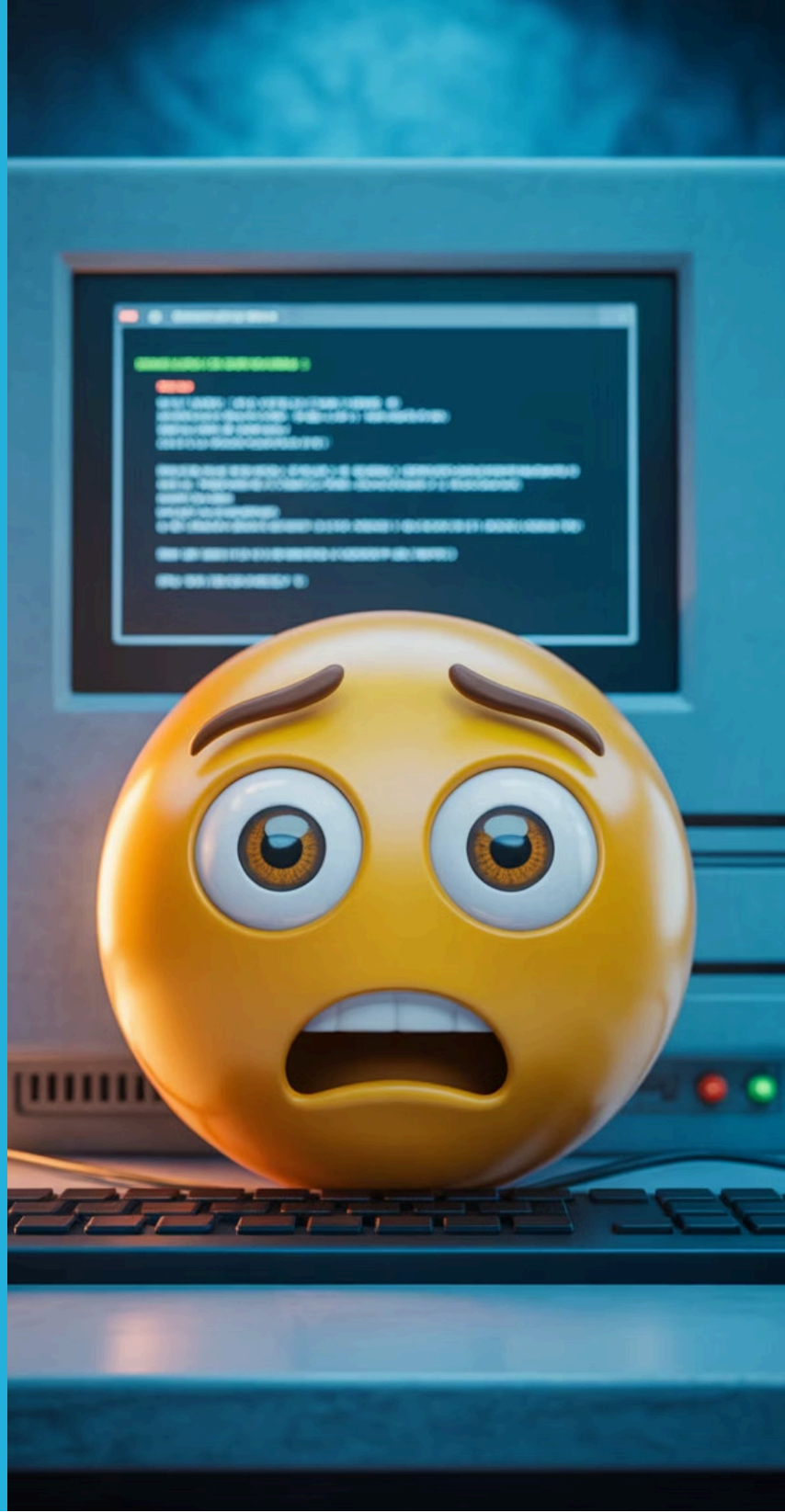
Borrar archivos ya ejecutados

4

Ignorar dependencias

Cambiar modelos sin actualizar relaciones

Solución: 👉 Siempre verificar con `showmigrations` antes de desplegar.



Buenas prácticas

✅ **Migraciones pequeñas**

Generar migraciones pequeñas y frecuentes

✅ **Revisar SQL**

Revisar sqlmigrate antes de producción

✅ **No editar aplicadas**

No editar migraciones aplicadas

✅ **Consistencia**

Mantener consistencia entre entornos

✅ **Documentar**

Documentar cambios grandes

Ejercicio guiado

Crea un modelo Producto y agrega un campo nuevo "stock".

Pasos:

01

Crea modelo en productos/models.py

02

Ejecuta makemigrations

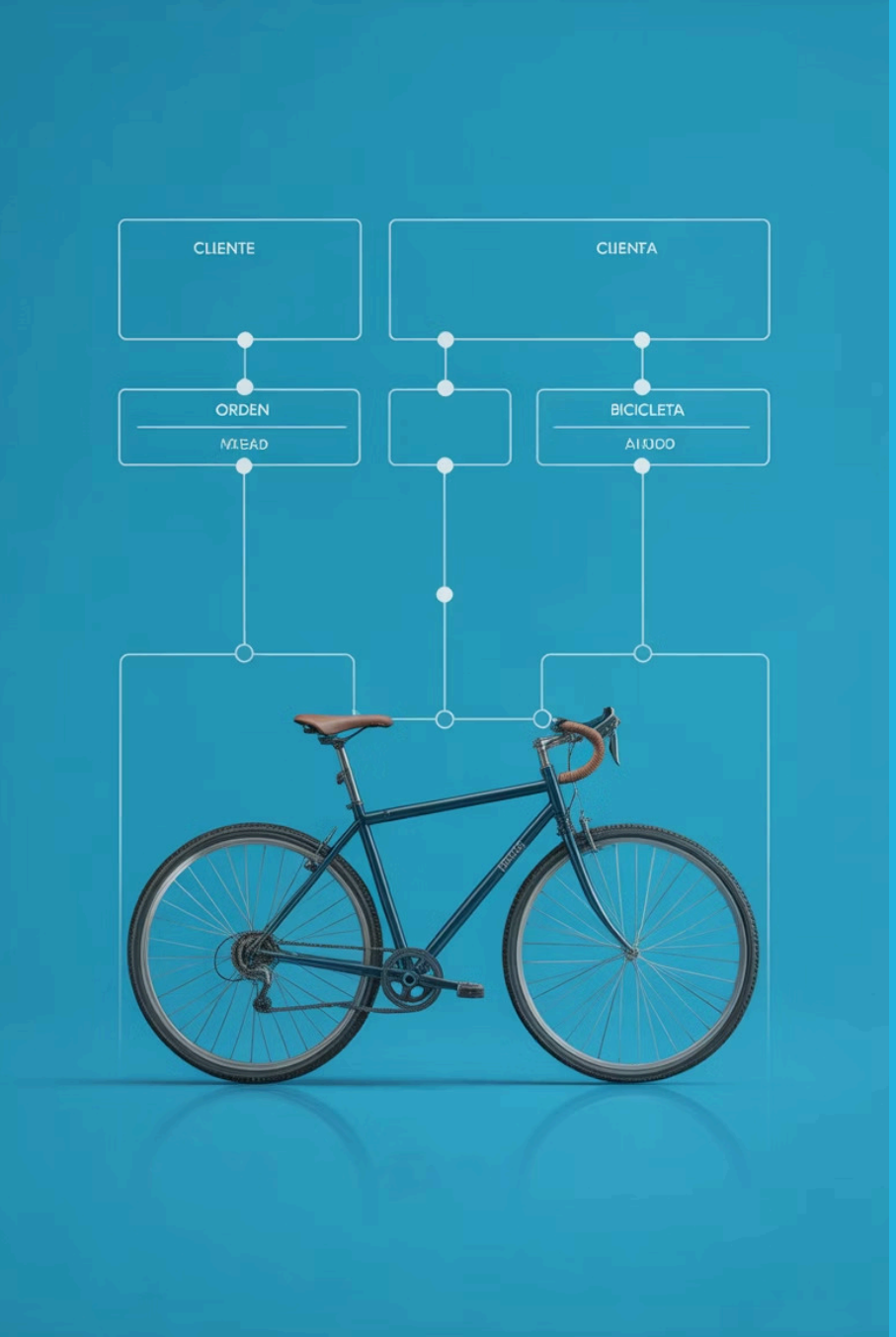
03

Ejecuta migrate

04

Visualiza el SQL generado

Actividad práctica en parejas: Compara con tu compañero qué SQL generó cada migración.



Migraciones y relaciones

Cada vez que creas una relación:

Uno a Uno

Se crea una FK única

Uno a Muchos

Se agrega una FK normal

Muchos a Muchos

Django crea una tabla intermedia

Cada cambio genera su propia migración.

Metodología activa: "Desafío en vivo"

Desafío:

1. **Agrega un campo descuento a la clase Orden**
2. **Ejecuta makemigrations y migrate**
3. **Verifica con showmigrations**
4. **Muestra el SQL con sqlmigrate**

Reto extra: Elimina el campo y genera una nueva migración que lo quite.

Cierre y resumen

Conclusiones:

- Las migraciones son el puente entre modelos y base de datos
- Django automatiza y asegura los cambios estructurales
- Son esenciales para proyectos colaborativos

"Sin migraciones, Django sería como construir una casa sin planos."