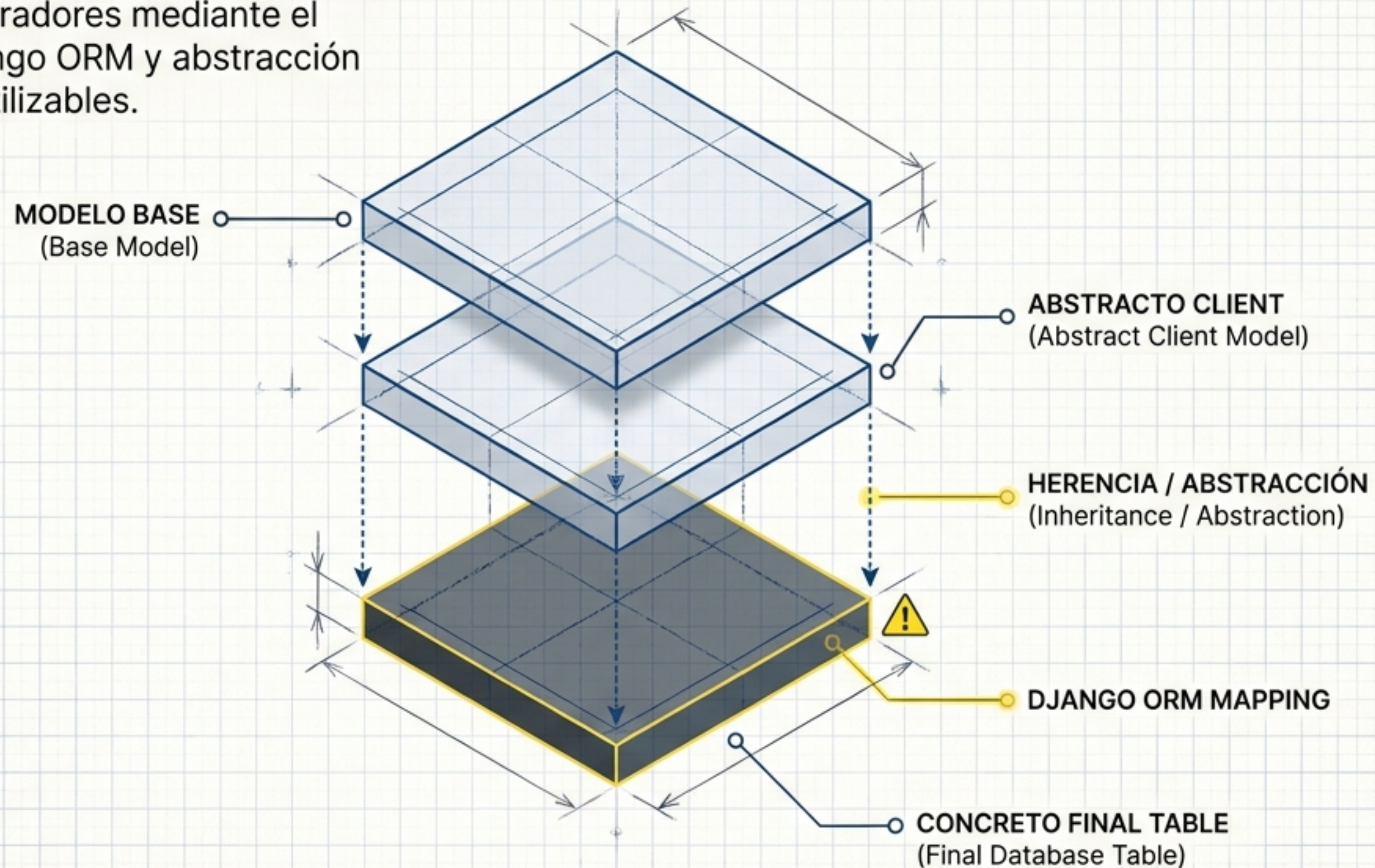


ISS-04 — Modelo Client

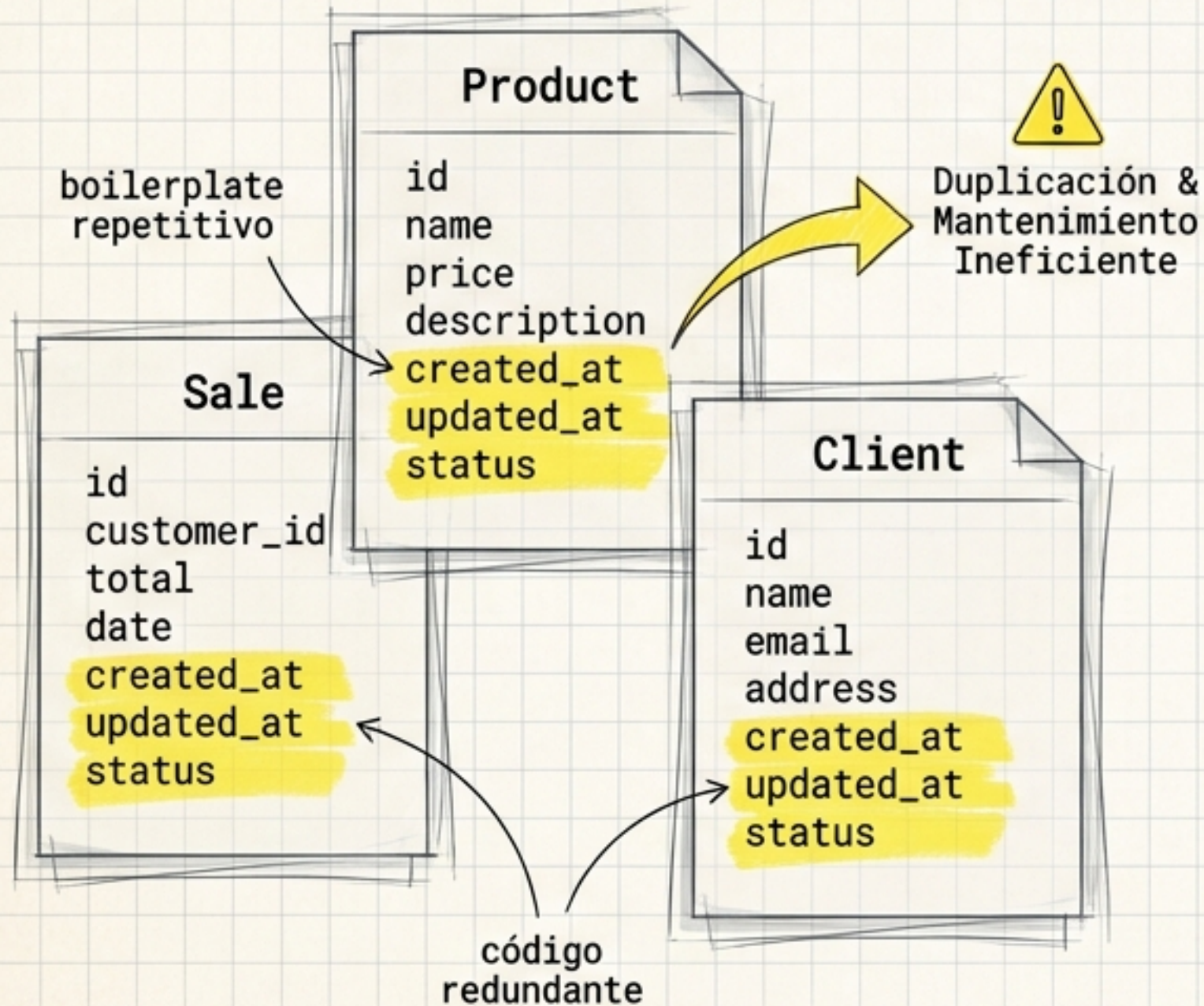
Persistencia de compradores mediante el modelo Client en Django ORM y abstracción de modelos base reutilizables.



El objetivo de arquitectura

Separar el código repetitivo (boilerplate) de la lógica de negocio pura.

El Problema

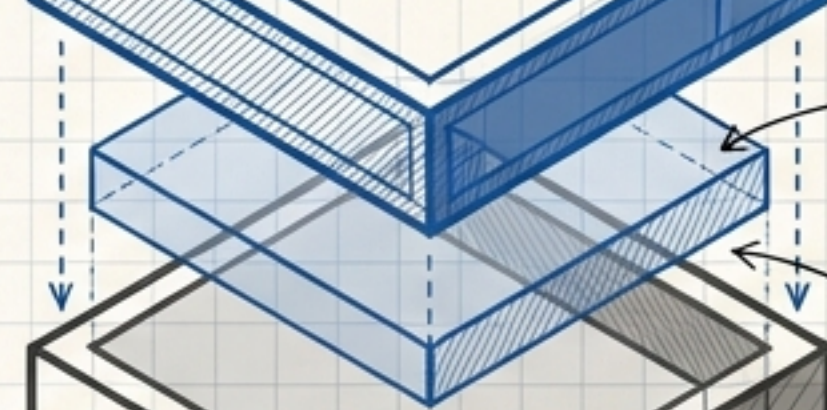


La Solución

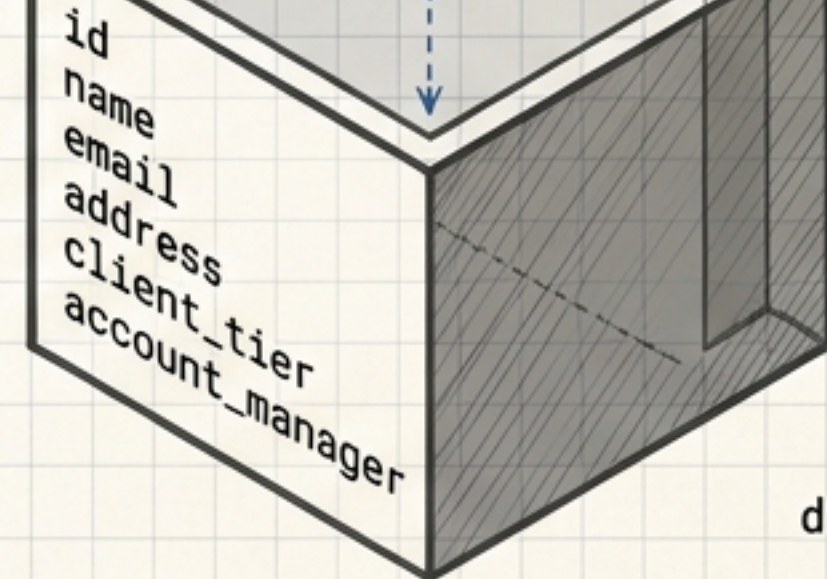
Modelos Abstractos (Common)



Herencia Centralizada (Base)



Lógica de Negocio Pura



Modelo Client (Negocio)

Código Optimizado & Modular:
Separación Clara de Responsabilidades

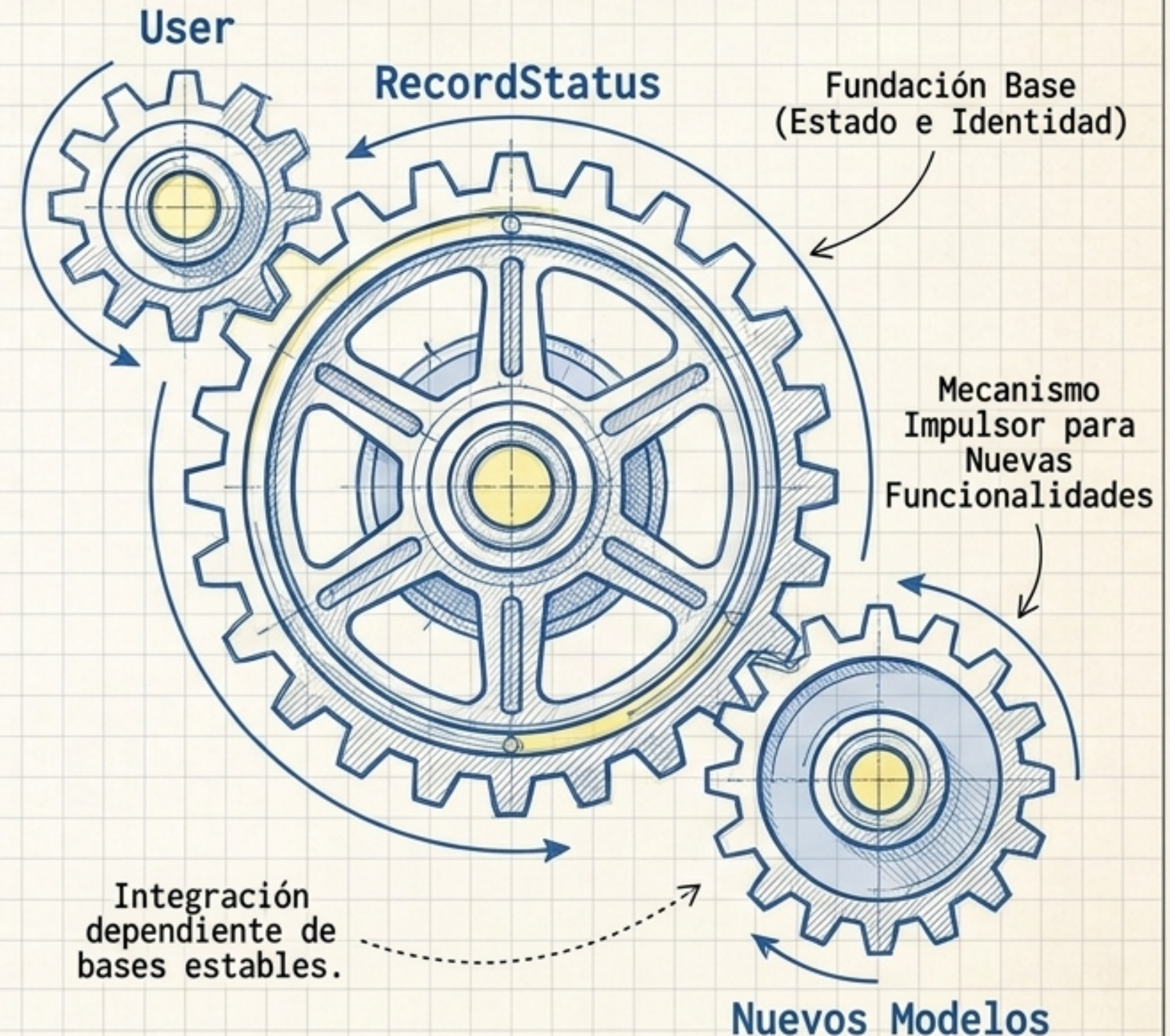
Fundamentos previos (ISS-03 superado)

Antes de construir el cliente, la infraestructura de seguridad base debe estar firme.

QA Inspection

- [PASS]** Proyecto aislado en `.venv` funcionando.
- [PASS]** `apps/common/status.py` con el enumerador `RecordStatus` (ACTIVE / INACTIVE).
- [PASS]** `security.User` operando como `AUTH_USER_MODEL`.

Validación de cimientos de infraestructura para desarrollo modular.



Transformación del sistema de archivos

De la estructura inicial vacía a la creación del paquete de abstracción común.

[ANTES]

```
apps/  
├── client/models.py (vacío)  
└── common/status.py
```

[DESPUÉS]

```
apps/  
├── client/models.py (reescrito: modelo de negocio)  
└── common/  
    ├── status.py  
    └── models.py (NUEVO: modelos abstractos base)
```

→ Modelos abstractos base ahora disponibles para consumo, centralizando la lógica común.

Abstracción Nivel 1: Auditoría de tiempo

Centralizando el rastreo de creación y actualización para todas las tablas futuras.

```
class TimeStampedModel(models.Model):  
    created_at = models.DateTimeField(auto_now_add=True)  
    updated_at = models.DateTimeField(auto_now=True)
```

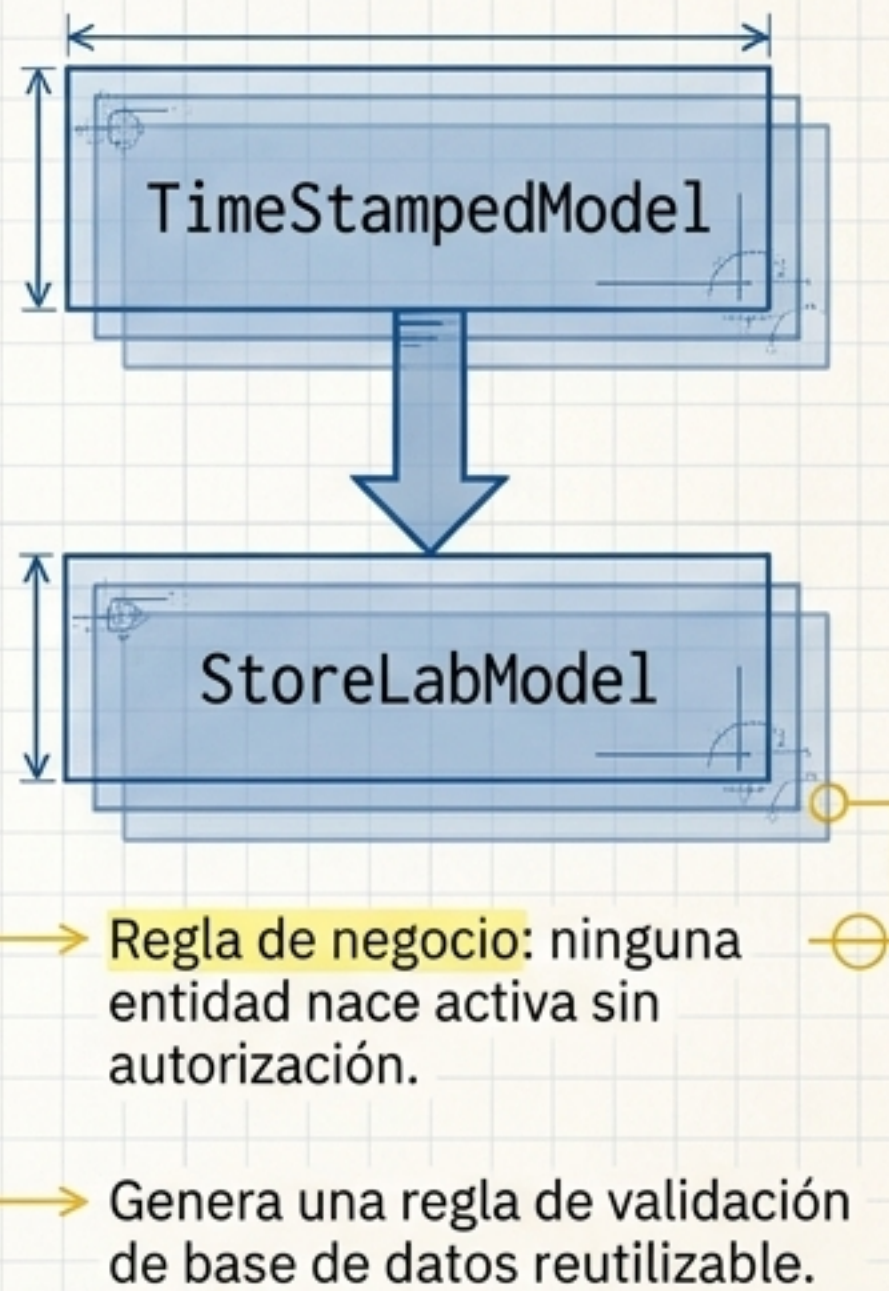
```
class Meta:  
    abstract = True
```

Evita que Django intente crear una tabla física `time\_stamped\_model` en la base de datos. Solo sirve como plantilla.

Abstracción Nivel 2: Estado del registro

Inyectando la baja lógica por defecto y reglas de integridad de base de datos.

```
class StoreLabModel(TimeStampedModel):  
    status = models.CharField(..., default=RecordStatus.INACTIVE)  
  
    class Meta:  
        abstract = True  
  
    @classmethod  
    def status_constraint(cls, name):  
        return models.CheckConstraint(..., name=name)
```



Implementación física: El Modelo Client

Traduciendo las reglas del negocio en atributos físicos de base de datos.

Imports

```
from django.core.validators import MinLengthValidator
```

Herencia

```
class Client(StoreLabModel):
```

Hereda automáticamente timestamps y estado (inactive).

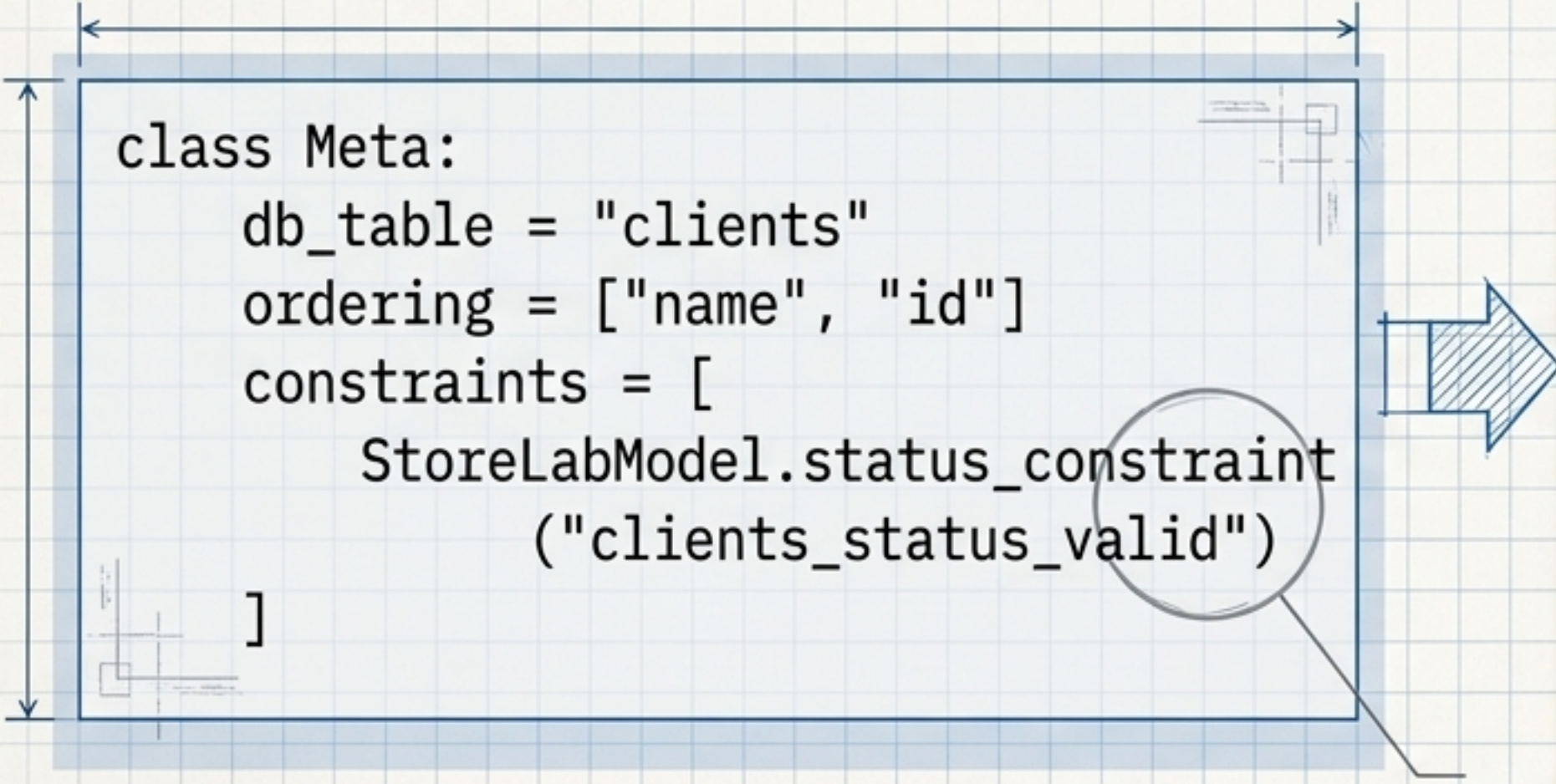
Campos

```
name = models.CharField(max_length=150, validators=[MinLengthValidator(2)])  
email = models.EmailField(..., unique=True, null=True, blank=True)
```

Previene nombres de un solo carácter tanto en la API como en el Admin.

Comportamiento y Restricciones (Clase Meta)

Definiendo cómo se comporta la tabla en el motor físico.



```
class Meta:
    db_table = "clients"
    ordering = ["name", "id"]
    constraints = [
        StoreLabModel.status_constraint
        ("clients_status_valid")
    ]
```

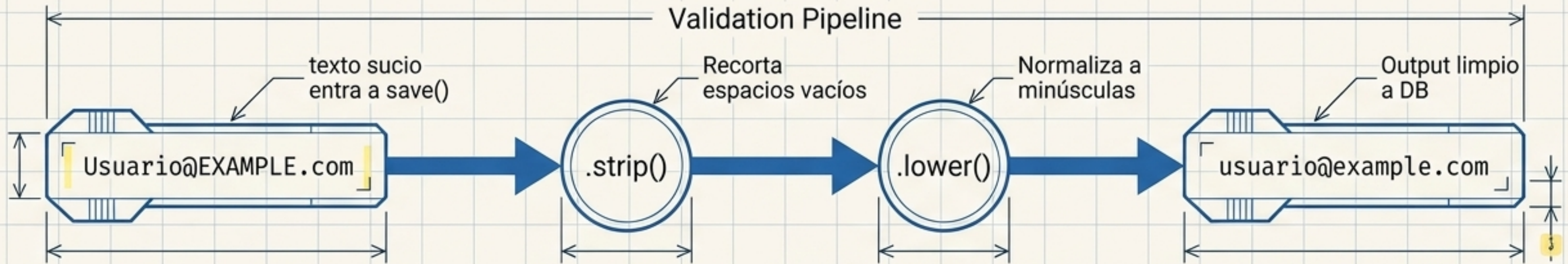
The code block is highlighted with a light blue background. It has dimension lines on the left and top. A blue arrow points from the right side of the code block to a yellow warning box. A circular callout highlights the `status_constraint` method call in the `constraints` list.

Alerta de Motor Multidialecto

El límite histórico de nombres de constraints en Oracle es de 30 caracteres.
'clients_status_valid' tiene 20 caracteres (✓). Superar este límite provocaría fallos de migración en entornos Oracle.

Lógica de Guardado: Normalización de datos

Interceptando la escritura para asegurar la limpieza del correo electrónico.

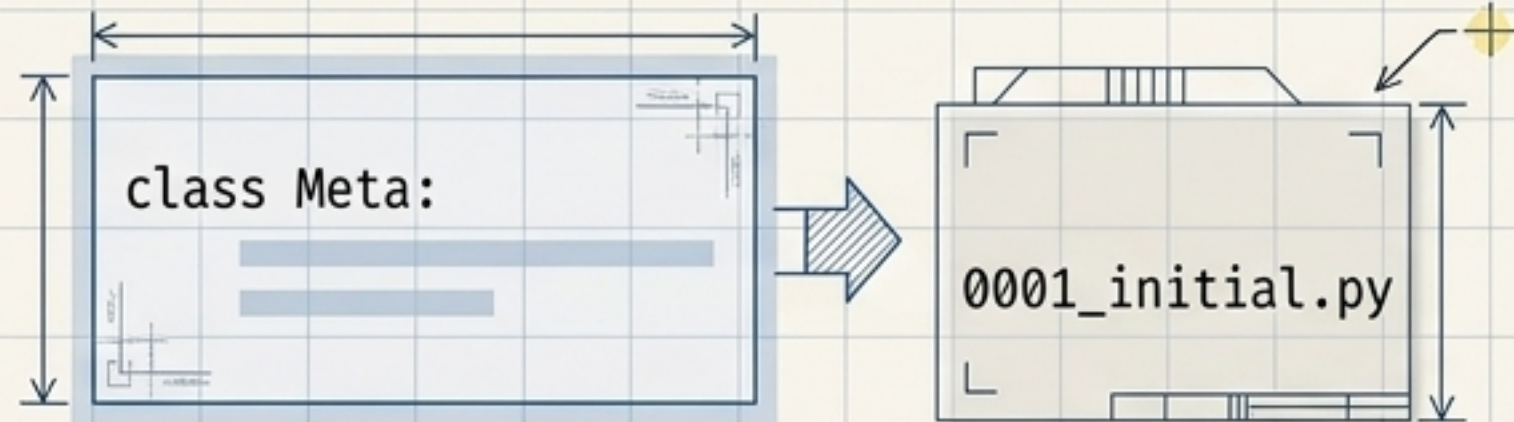


```
def save(self, *args, **kwargs):  
    if self.email:  
        self.email = self.email.strip().lower()  
    super().save(*args, **kwargs)
```

Traduciendo Python a SQL: Migraciones

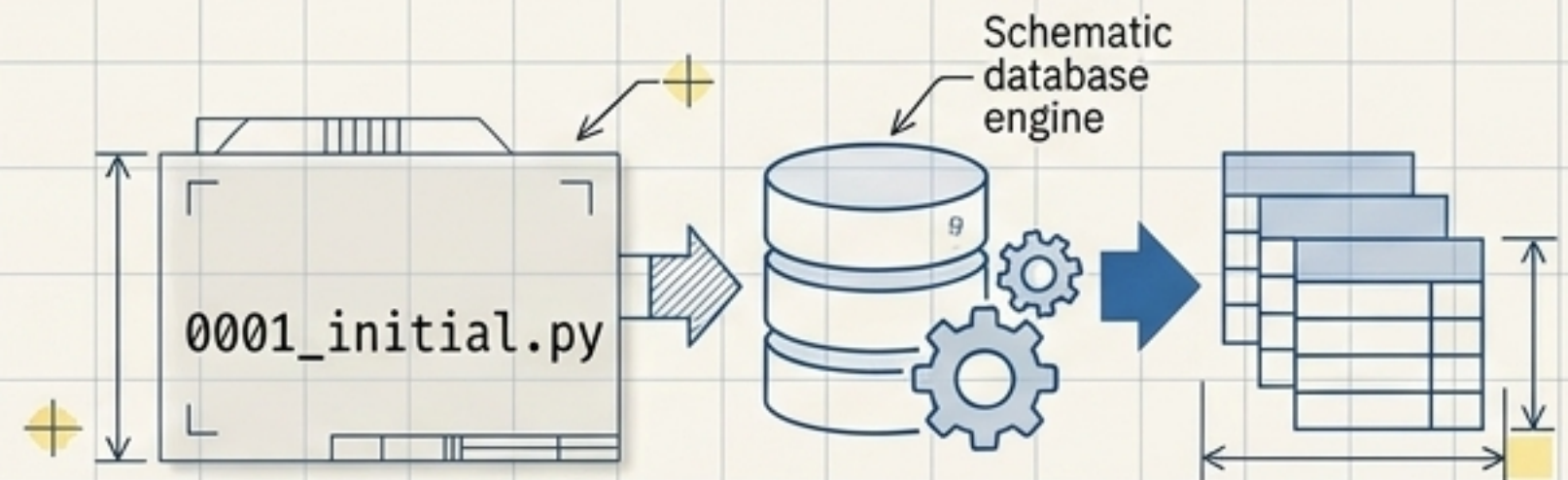
Ejecución del motor de migraciones para materializar el esquema.

```
> python manage.py makemigrations client
Migrations for 'client':
  apps/client/migrations/0001_initial.py
> python manage.py migrate
Applying client.0001_initial... OK
```



1. makemigrations

Escribe el plano físico del esquema (0001_initial.py).

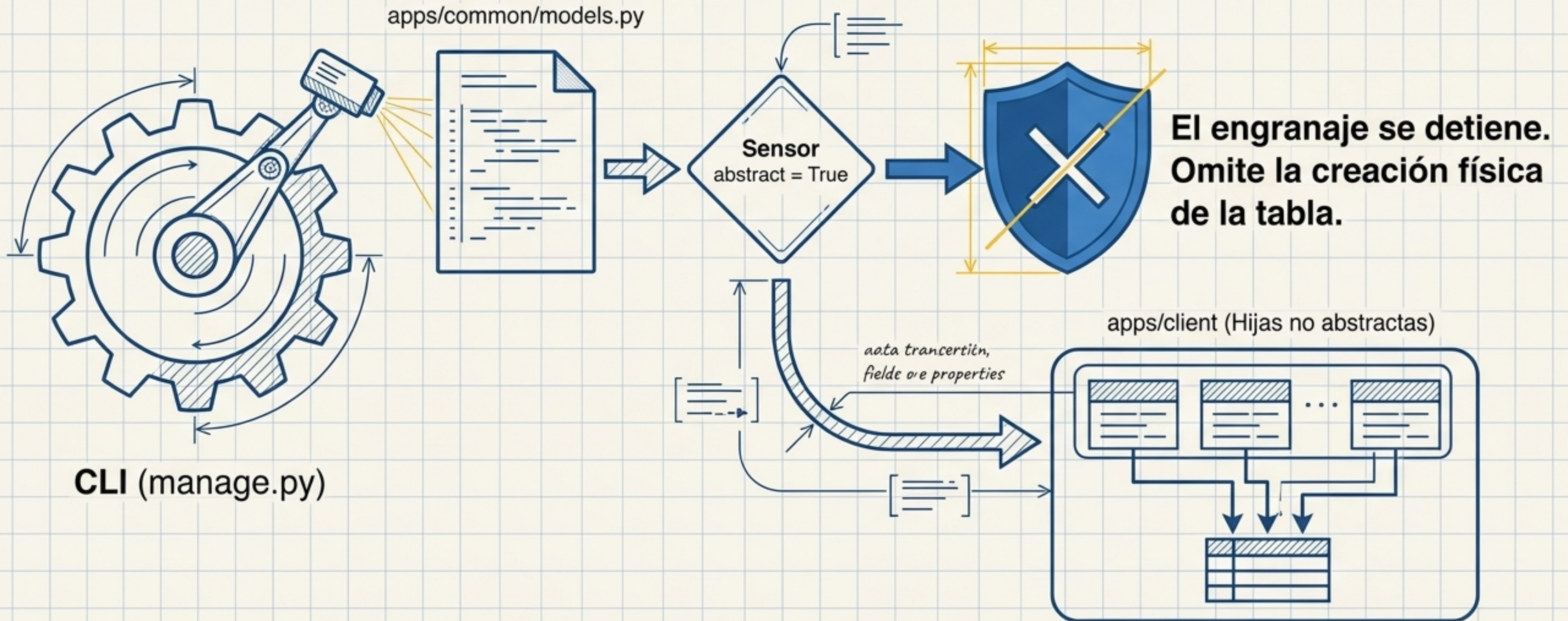


2. migrate

Ejecuta el **DDL** (Data Definition Language) en el motor de base de datos activo.

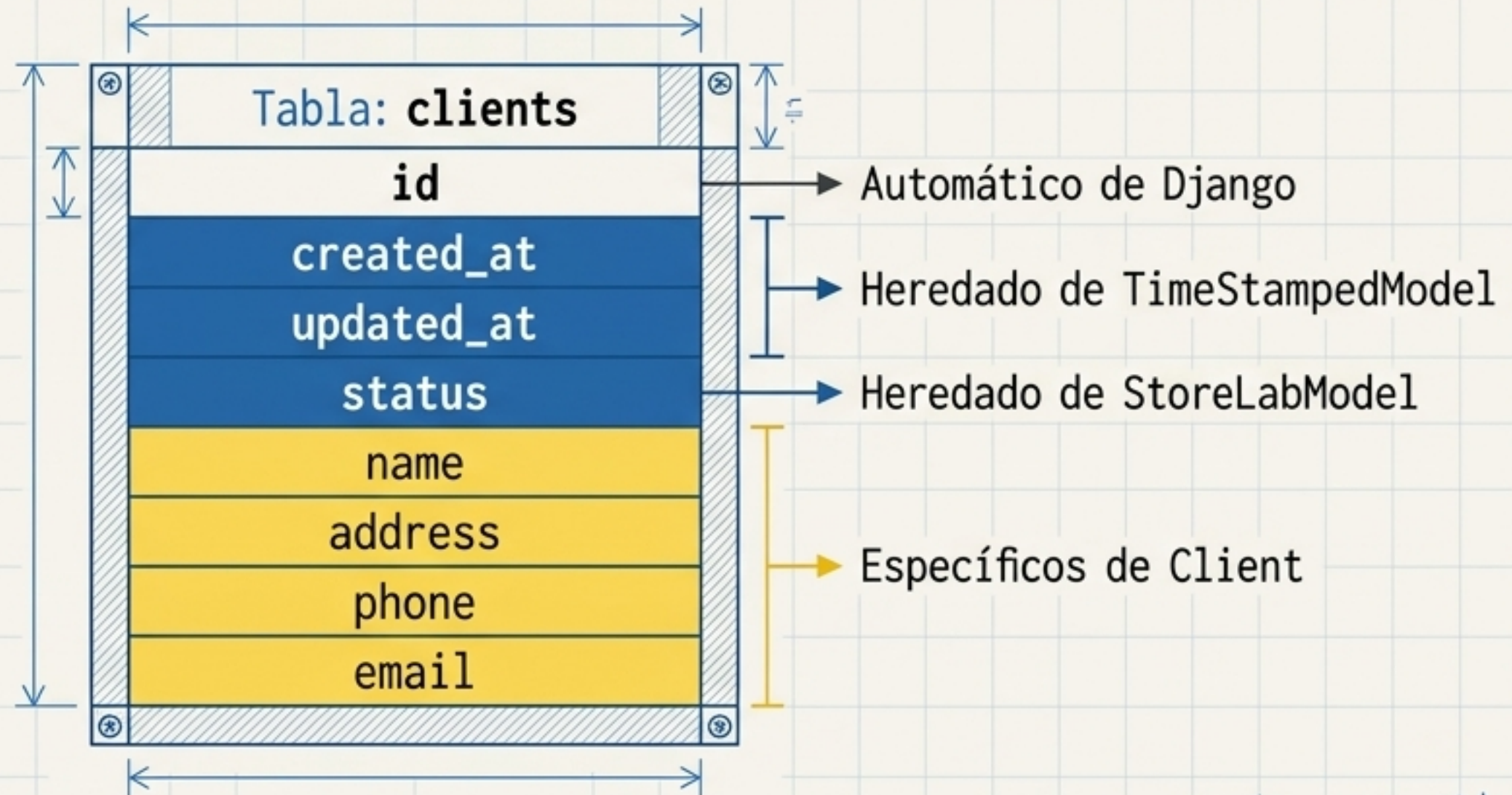
El fenómeno de la migración Fantasma

¿Por qué apps/common no genera un archivo de migración?



Esquema Físico Resultante

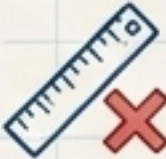
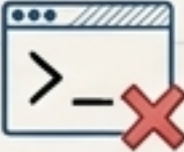
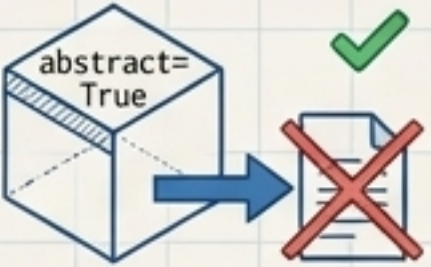
La consolidación de tres clases Python en una única tabla relacional.



La magia del ORM y DRY. En Python vemos tres clases separadas, pero el motor de base de datos recibe una estructura física única, compacta y perfectamente normalizada.

Errores frecuentes vs. Mejores Prácticas

Antipatrones comunes al estructurar modelos y cómo evitarlos.

Error (Antipatrón)	Solución Implementada
Permitir nombres de un solo carácter. ❌	Validar a nivel de modelo con <code>MinLengthValidator(2)</code> . ✅
Emails duplicados por diferencias en mayúsculas (A@test.com vs a@test.com). ❌	Interceptar en el método <code>save()</code> con <code>strip().lower()</code> . ✅
Nombres de CheckConstraint excesivamente largos (>30 caracteres). ❌ 	Usar nombres cortos y portables para garantizar compatibilidad con Oracle (<code>clients_status_valid</code>). ✅
Intentar ejecutar makemigrations common. ❌ 	Entender que <code>abstract=True</code> no genera ni requiere migraciones.  ✅

Criterios de Aceptación (QA Checklist)

Las condiciones innegociables para considerar completado el ISS-04.



[AC-04-01] La tabla física en la base de datos se llama `clients`.



[AC-04-02] Alta de un cliente sin enviar estado asume por defecto `inactive`.



[AC-04-03] Campos `created_at` y `updated_at` son autogenerados.



[AC-04-04] Inserción de `email` duplicado es rechazada (Integridad de datos).



[AC-04-05] Inserción de nombre menor a 2 caracteres es rechazada por el modelo.

Verificación, GATE y Defensa Oral

Evidencias obligatorias para avanzar al siguiente módulo.

	Verificación	Evidencia	Resultado
1	Migración aplicada (db_table='clients')	EVI-04-01: client.0001_initial	PASS
2	Reglas de negocio (defaults, unicidad, longitudes)	EVI-04-02: Tests de cliente	PASS

Preguntas de Defensa Oral

1. ¿Por qué el paquete apps/common no produce un archivo 0001_initial al migrar?
2. ¿Cómo garantiza la clase Meta que nuestra arquitectura soporte bases de datos multidialecto como Oracle?
3. ¿En qué momento exacto del ciclo de vida de Django se ejecuta strip().lower() sobre el email?