

# Bases de Datos Avanzadas: Su Viaje de la Teoría a la Producción

Mentalidad, herramientas y buenas prácticas  
para construir sistemas robustos.

# ¿Quien soy?

Mtro. Ramsés Alejandro Camas Nájera

**Head of AI en MAPS Disruptivo  
CEO y Co-founder de RavanTech**



Ingeniero en Desarrollo de Software por la **Universidad Politécnica de Chiapas**, 2019-2022

Maestro en Inteligencia Artificial por la **Universidad Politécnica de Valencia**, 2023-2024

- Becario del **Valencian Graduate School and Research Network of Artificial Intelligence (Valgrai)**
- Asistente de Investigación en el **Valencian Research Institute for Artificial Intelligence (VRAIN)**
- Miembro Activo de la **Sociedad Mexicana de Inteligencia Artificial (SMIA)**

# La Meta: De Saber Consultas a Defender Soluciones

Este curso no se trata de aprender SQL básico. La verdadera meta es transformar su perspectiva.

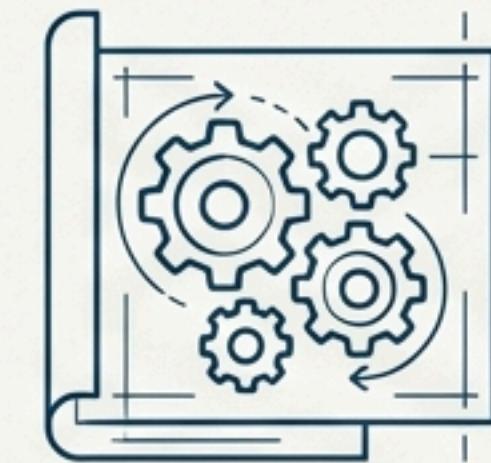
## Punto de Partida



### “Sé hacer consultas”

Conoce la sintaxis SQL. Puede extraer y manipular datos existentes.

## Destino



### “Puedo construir y defender una solución de datos lista para producción”

Diseña esquemas sólidos, justifica decisiones técnicas, asegura la reproducibilidad y opera con buenas prácticas.

El objetivo es desarrollar una **mentalidad de producción**.

# El Mapa del Territorio: ¿Qué Significa 'Avanzadas'?

"Avanzadas" implica dominar las capas que rodean al SQL para construir sistemas que funcionan en el mundo real.

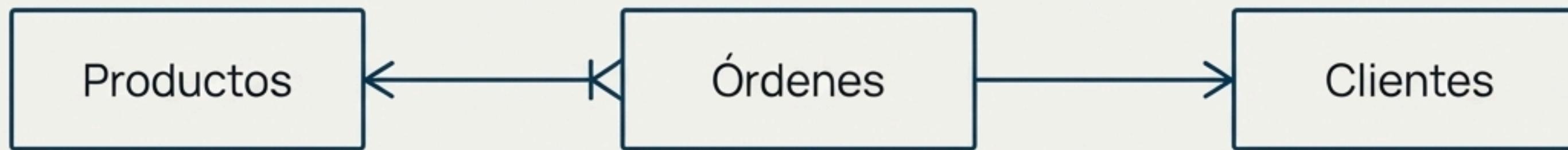


El fin es que desarrollen criterio técnico para tomar decisiones, no solo para seguir tutoriales.

# El Proyecto Integrador: Donde la Teoría se Vuelve Realidad

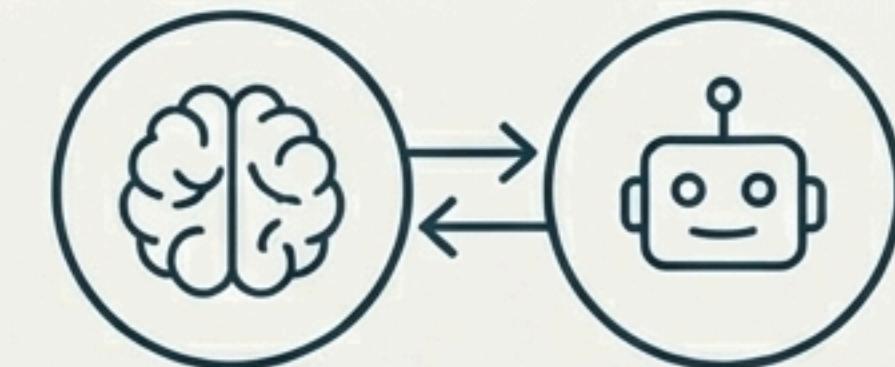
A lo largo del curso, aplicaremos cada concepto en un proyecto con un dominio realista de e-commerce e inventario.

- **Catálogo:** Modelado de productos, categorías y sus atributos.
- **Operaciones:** Gestión de órdenes, pagos y estados de ciclo de vida.
- **Consistencia:** Implementación de reglas de negocio críticas (ej. "el stock no puede ser negativo").
- **Analítica:** Diseño para soportar consultas de negocio (ej. "top ventas por periodo").



No buscamos “muchas tablas por presumir”. Buscamos un **modelo coherente**, con reglas claras y que soporte consultas reales.

# Reglas del Juego: La IA es su Copiloto, no su Sustituto



Pueden y deben usar herramientas como ChatGPT, Gemini o NotebookLM para acelerar su trabajo. Sin embargo, la comprensión es innegociable.

Se evaluará lo siguiente:



- **Evidencia de Comprensión:** Debe poder explicar qué hizo, por qué eligió ese enfoque y qué alternativas o trade-offs existen.



- **Bitácora de IA (Requerida):** Documente los prompts clave, las respuestas que utilizó y, más importante, qué corrigió o cómo verificó la información.



- **Defensa del Trabajo:** Si no puede defender una línea de código o una decisión de diseño, no cuenta como aprendizaje.

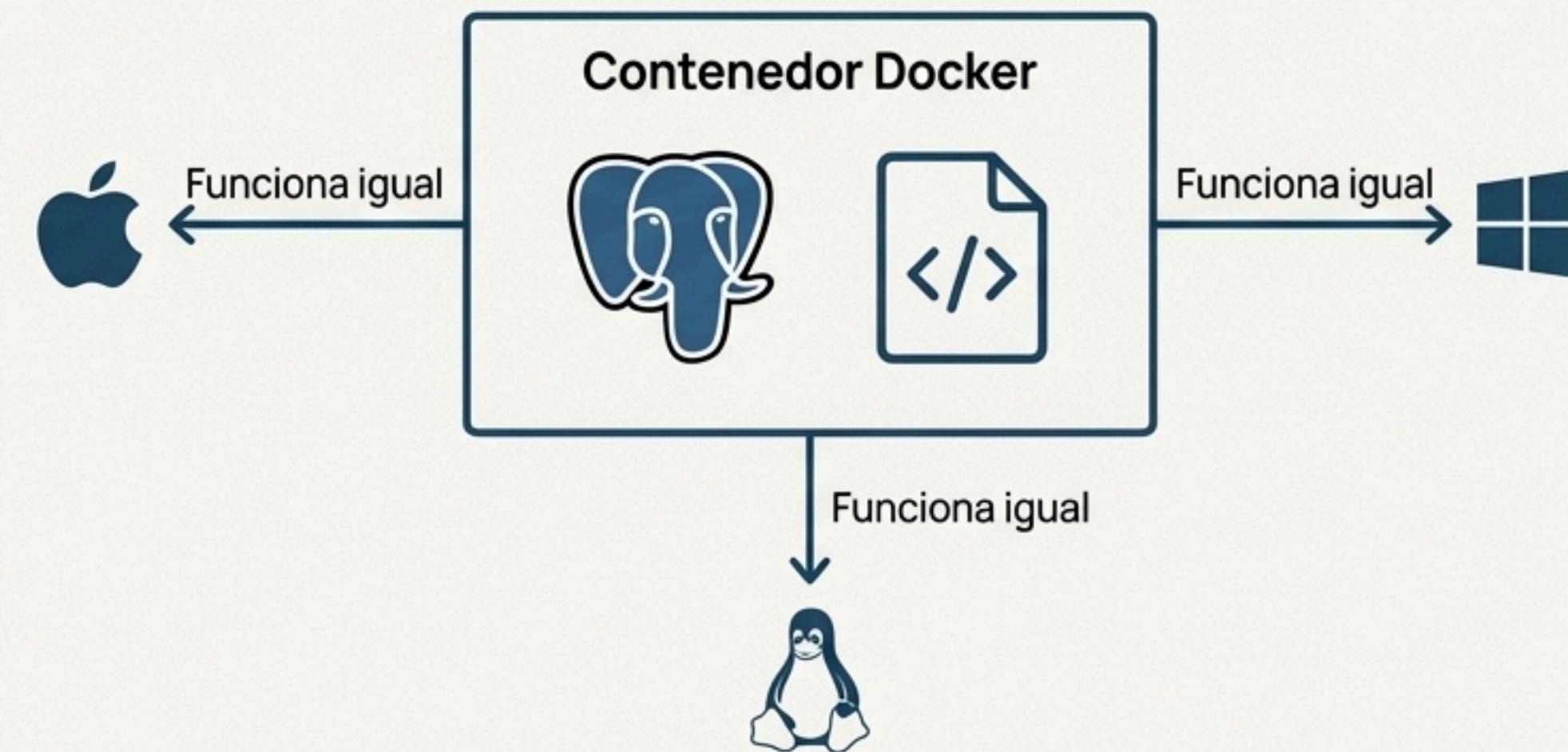
# El Primer Obstáculo en el Camino: “En mi máquina sí funciona”



La inconsistencia de los entornos de desarrollo genera caos, pérdida de tiempo y errores impredecibles.

Un entorno profesional debe ser **reproducible**, no artesanal.

# La Solución: Docker para Entornos Consistentes



Docker es la herramienta estándar para eliminar el caos de los entornos. Funciona empaquetando una aplicación y todas sus dependencias en una unidad estandarizada para el desarrollo de software.

## ¿Por qué Docker resuelve nuestro problema?

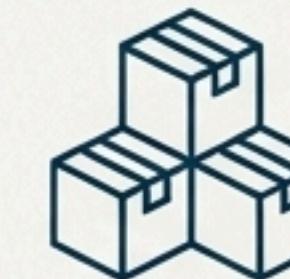
- **Empaqueta el Entorno:** Define la versión exacta de PostgreSQL, su configuración y cualquier extensión necesaria en un archivo.
- **Garantiza la Consistencia:** Todos los miembros del equipo ejecutan exactamente el mismo entorno con un solo comando. Se acabaron las peleas por versiones.
- **Aísla y Simplifica:** Permite crear, destruir y recrear la base de datos de forma segura y rápida, sin 'ensuciar' o 'romper' su máquina local.

# Aclaración Técnica: Contenedores vs. Máquinas Virtuales



## Máquina Virtual (VM)

- **Cómo funciona:** Emula un sistema operativo completo (**Guest OS**) sobre el sistema operativo del anfitrión (**Host OS**).
- **Recursos:** **Más pesada**. Consume significativamente más CPU, RAM y disco.
- **Arranque:** **Lento** (minutos).
- **Caso de uso:** Cuando se necesita un aislamiento completo del SO.



## Contenedor (Docker)

- **Cómo funciona:** Los procesos corren aislados pero comparten el kernel del sistema operativo anfitrión.
- **Recursos:** **Mucho más ligero**. Comparte recursos de manera eficiente.
- **Arranque:** **Rápido** (segundos o milisegundos).
- **Caso de uso:** Ideal para empaquetar y ejecutar servicios individuales, como nuestra base de datos PostgreSQL.

# Orquestando Servicios con Docker Compose

En lugar de ejecutar comandos largos y complejos en la terminal, describimos nuestro entorno en un archivo `docker-compose.yml`. Es nuestro plano para el entorno.

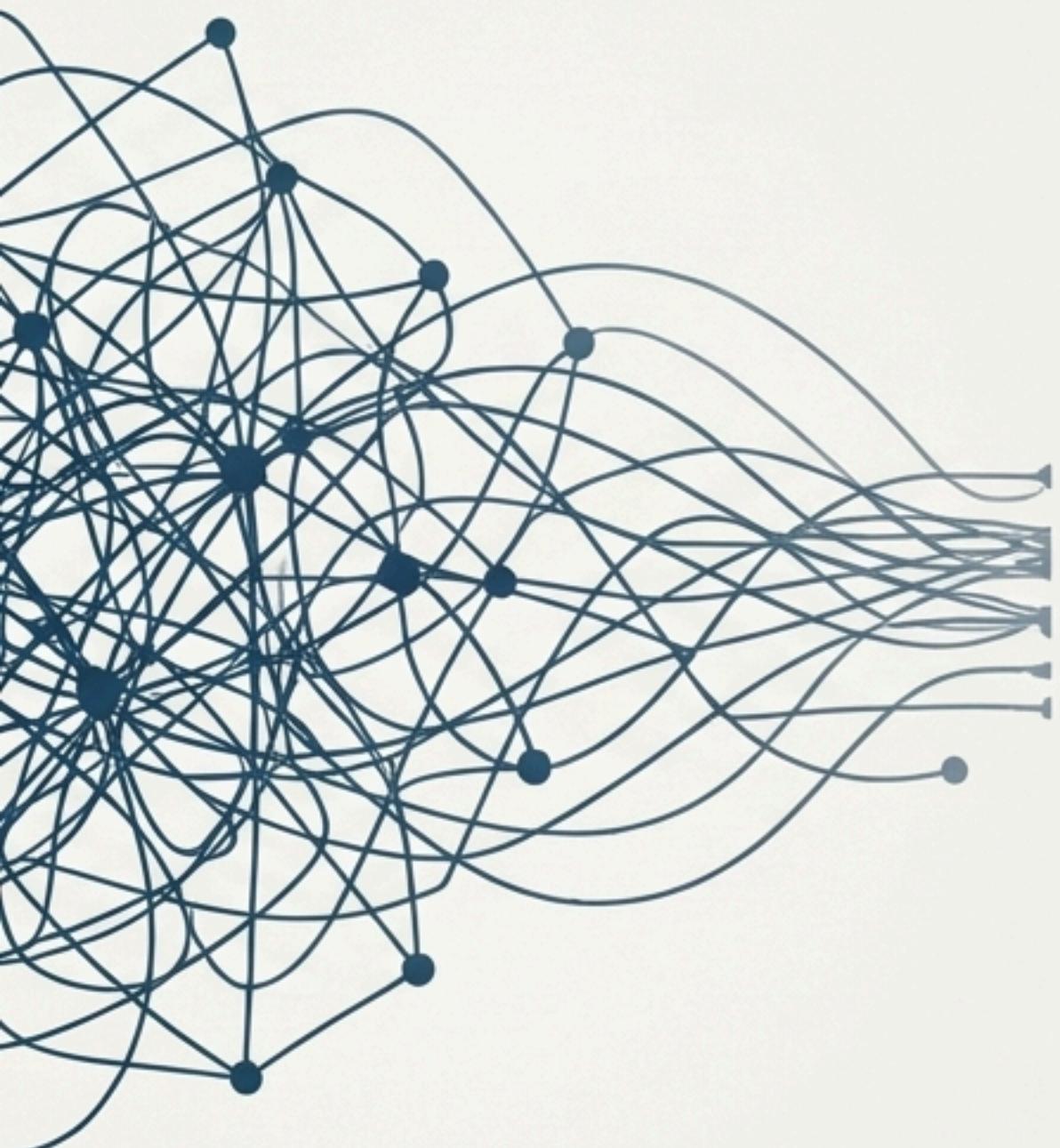
```
services:  
  db:  
    image: postgres:15  
    ports:  
      - "5432:5432"  
    environment:  
      - POSTGRES_USER=user  
      - POSTGRES_PASSWORD=password  
    volumes:  
      - pgdata:/var/lib/postgresql/data  
volumes:  
  pgdata: # Define el volumen para que los datos sobrevivan.
```

The diagram consists of four numbered callouts (1, 2, 3, 4) in orange circles, each with a thin orange line pointing to a specific field in the code above. Callout 1 points to 'image: postgres:15'. Callout 2 points to 'ports: - "5432:5432"'. Callout 3 points to 'environment: - POSTGRES\_USER=user'. Callout 4 points to 'volumes: - pgdata:/var/lib/postgresql/data'.

- 1 1. Qué imagen de software usar.
- 2 2. Qué puertos exponer al exterior.
- 3 3. Qué variables de entorno necesita.
- 4 4. Dónde persistir los datos.

**Concepto Clave Destacado:** **Volúmenes**: Son la pieza fundamental que asegura que sus datos **sobrevivan** incluso si apaga o reinicia el contenedor. Separan los datos del ciclo de vida del proceso.

# El Siguiente Desafío: Del Mundo Real a Tablas Coherentes



Una vez que tenemos un entorno estable, debemos asegurar que la estructura de nuestros datos sea sólida. Un mal diseño es una fuente constante de errores y complejidad.

## El Proceso de Modelado Relacional:

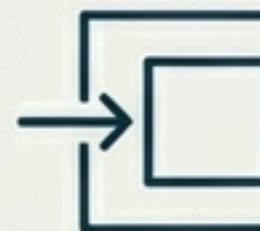
- 1 Identificar Entidades:** Las 'cosas' de nuestro dominio.  
(Ej: `Producto`, `Cliente`, `Orden`).
- 2 Definir Atributos:** Las propiedades de cada entidad.  
(Ej: `nombre`, `precio`, `fecha`).
- 3 Establecer Relaciones:** Cómo se conectan las entidades.  
(Ej: Un `Cliente` tiene muchas `Órdenes`).
- 4 Aplicar Reglas de Negocio:** Las restricciones que deben cumplirse siempre. (Ej: El estado de una orden solo puede ser uno de una lista predefinida).

**Una buena práctica:** primero defina qué significa cada concepto en su dominio, mucho antes de escribir `CREATE TABLE`.

# El Plano Arquitectónico: Normalización para Evitar el Caos

La **normalización** es el proceso de organizar las columnas y tablas de una base de datos para minimizar la **redundancia de datos** y evitar **inconsistencias**.

## Anomalía de Inserción



**Problema:** No se puede registrar un nuevo producto en la tabla `Ordenes\_y\_Productos` si aún no ha sido comprado en ninguna orden.

## Anomalía de Actualización



**Problema:** Si el nombre de un cliente está repetido en 10 filas de órdenes, al actualizarlo, debe cambiarlo en 10 lugares. Si olvida uno, los datos quedan inconsistentes.

## Anomalía de Borrado



**Problema:** Si borra la última orden de un cliente, y los datos del cliente estaban mezclados en esa tabla, podría perder toda la información de ese cliente para siempre.

Normalizar no es 'hacer más tablas por deporte'; es hacer que el modelo sea **confiable** y **predecible**.

# Las Reglas del Diseño: 1NF, 2NF y 3NF en Lenguaje Directo

## Primera Forma Normal (1NF): Valores Atómicos

**Regla:** Cada celda de una tabla debe contener un solo valor. No guarde listas en un campo.

**MAL:** 

`telefonos = '961..., 962..., 963...'`

**BIEN:**  Crear una tabla

`cliente\_telefonos` separada, con una fila por cada número.

## Segunda Forma Normal (2NF): Sin Dependencias Parciales

**Regla:** Aplica a claves primarias compuestas. Todos los atributos no-clave deben depender de la clave primaria \*completa\*, no solo de una parte de ella.

## Tercera Forma Normal (3NF): Sin Dependencias Transitivas

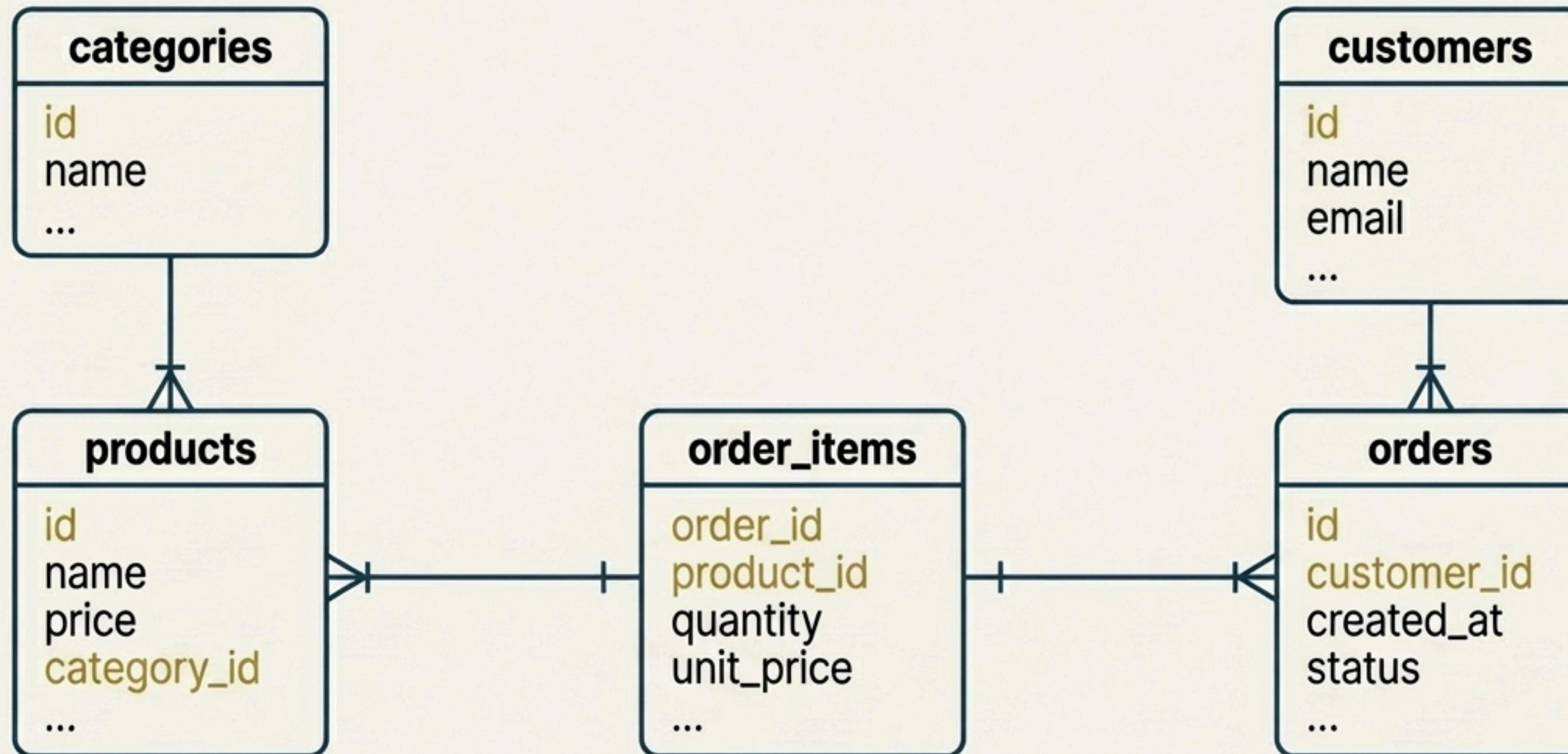
**Regla:** Un atributo no-clave no debe depender de otro atributo no-clave.

**Ejemplo:** Si en la tabla `Productos` tiene `category\_name` y `category\_description`, estos dos dependen de `category\_id`, no directamente de `product\_id`. Deberían estar en una tabla `Categorias`.

**Regla Práctica General:** Si un dato parece 'pertenecer' a otra entidad, probablemente debería estar en su propia tabla.

# Nuestro Punto de Partida: Un Esquema Inicial para el Proyecto

Este es un diseño preliminar y razonable para nuestro dominio de e-commerce. Lo iremos evolucionando y mejorando durante el curso.



# Pensando Como un Desarrollador Avanzado

Un buen esquema no solo almacena datos, sino que responde a preguntas complejas y protege la integridad del negocio. Las preguntas correctas son más importantes que las respuestas iniciales.

- ?
- El Precio:** "¿Dónde debe vivir el precio de un producto? ¿En la tabla `products`, en `order\_items`, o en ambas? ¿Por qué?"
- ?
- Cambios en el Tiempo:** "¿Qué pasa en nuestro sistema si el precio de un producto cambia después de que un cliente lo compró? ¿Cómo se refleja eso en la orden histórica?"
- ?
- Integridad de Datos:** "¿Cómo podemos usar la base de datos para asegurar que el stock de un producto nunca sea negativo? ¿Es solo responsabilidad de la aplicación?"
- ?
- Rendimiento:** "¿Qué índices vamos a necesitar desde el principio para soportar reportes comunes, como 'ver todas las órdenes de un cliente'?"

**El objetivo:** Justificar cada decisión de diseño con un razonamiento sólido.

# Su Misión Cumplida para Hoy

Al terminar la sesión de hoy (teoría + práctica), usted será capaz de:

- ✓ **Levantar un entorno PostgreSQL profesional y aislado usando Docker Compose.**
- ✓ **Conectarse a la base de datos** y verificar que está operativa, tanto por terminal (`psql`) como con una herramienta GUI (DBeaver).
- ✓ **Explicar por qué normalizamos los datos** y poder detectar anomalías comunes en un diseño de esquema.
- ✓ **Tener un esquema preliminar** (versión 0.1) razonable para comenzar a construir el proyecto integrador.

**Tiene las herramientas y el plano inicial. En la práctica, comenzamos la construcción.**