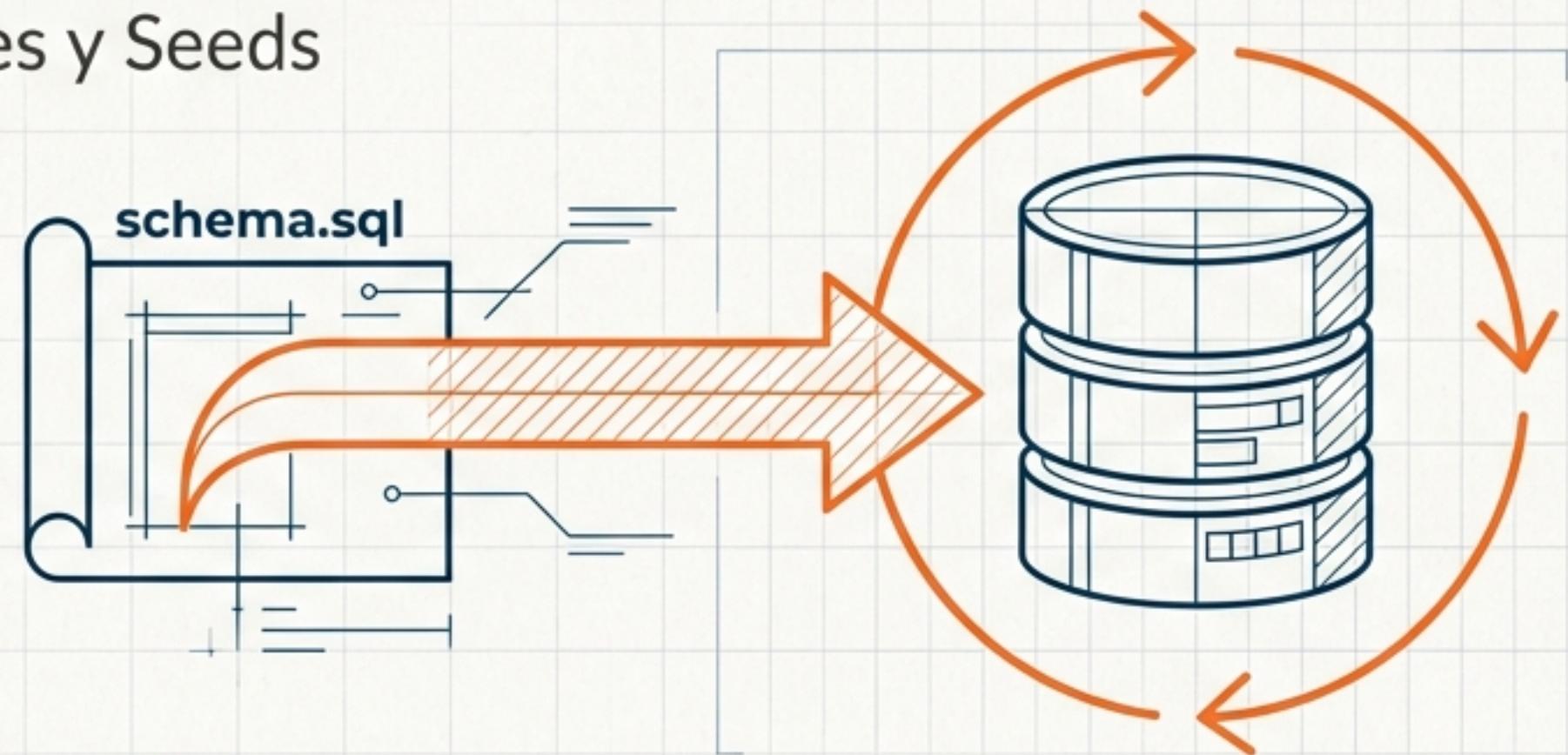


Construyendo una Base de Datos Reproducible

DBA – Clase 2: Modelado, Migraciones y Seeds

EL OBJETIVO DE HOY

Al finalizar esta sesión, tendrás una base de datos que puedes destruir y reconstruir perfectamente desde cero en minutos.



Modelado:
El plano
(**schema.sql**)



Migraciones:
El historial de cambios

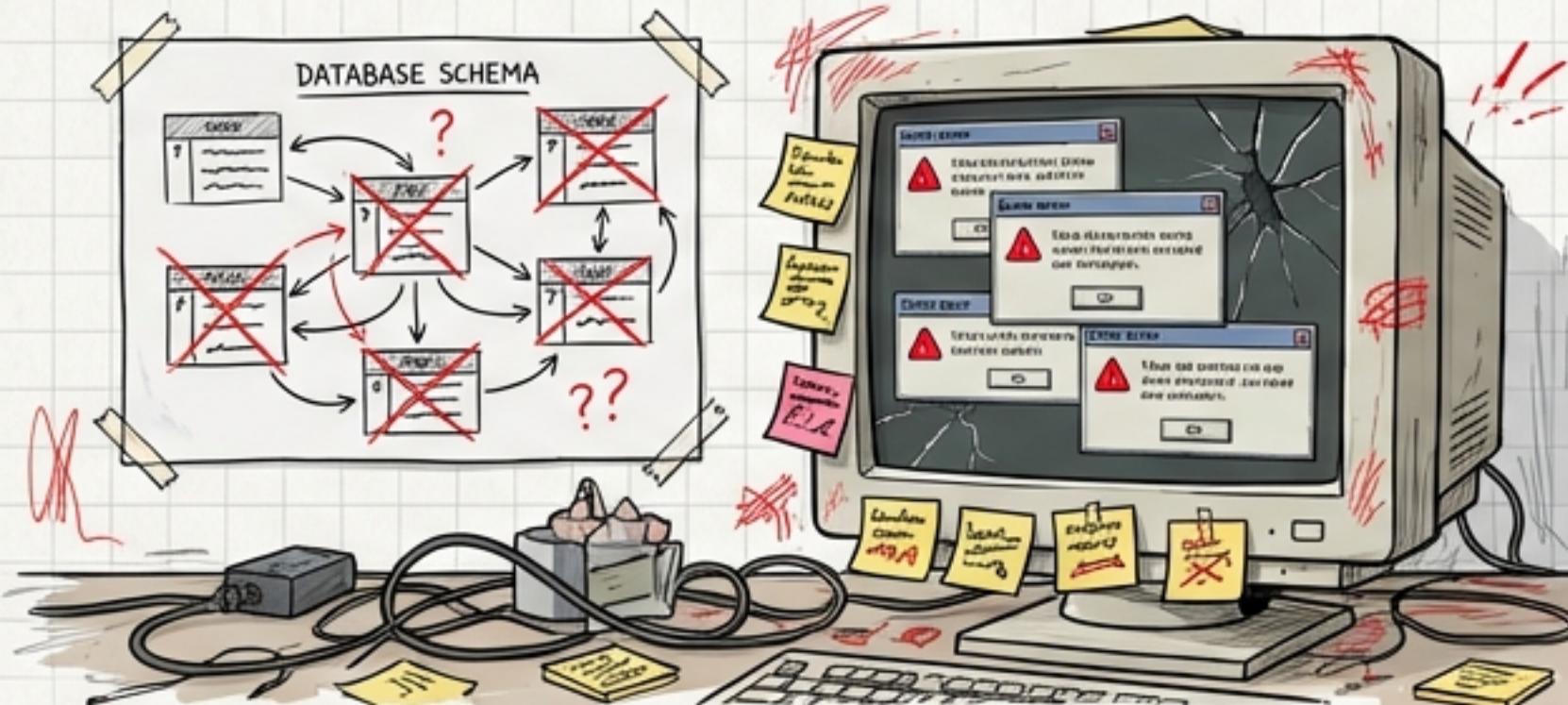


Seeds:
Los datos de prueba iniciales

El Problema: ¿Por Qué Tantos Proyectos Fallan Aquí?

La anarquía del ‘funciona en mi máquina’.

EL ARTESANO



- Cada desarrollador tiene datos diferentes en su base de datos local.
- Un cambio en la estructura rompe el entorno de otro compañero.
- Se pierden horas preparando datos para probar una nueva funcionalidad.
- Los bugs son imposibles de reproducir entre entornos.

EL ARQUITECTO

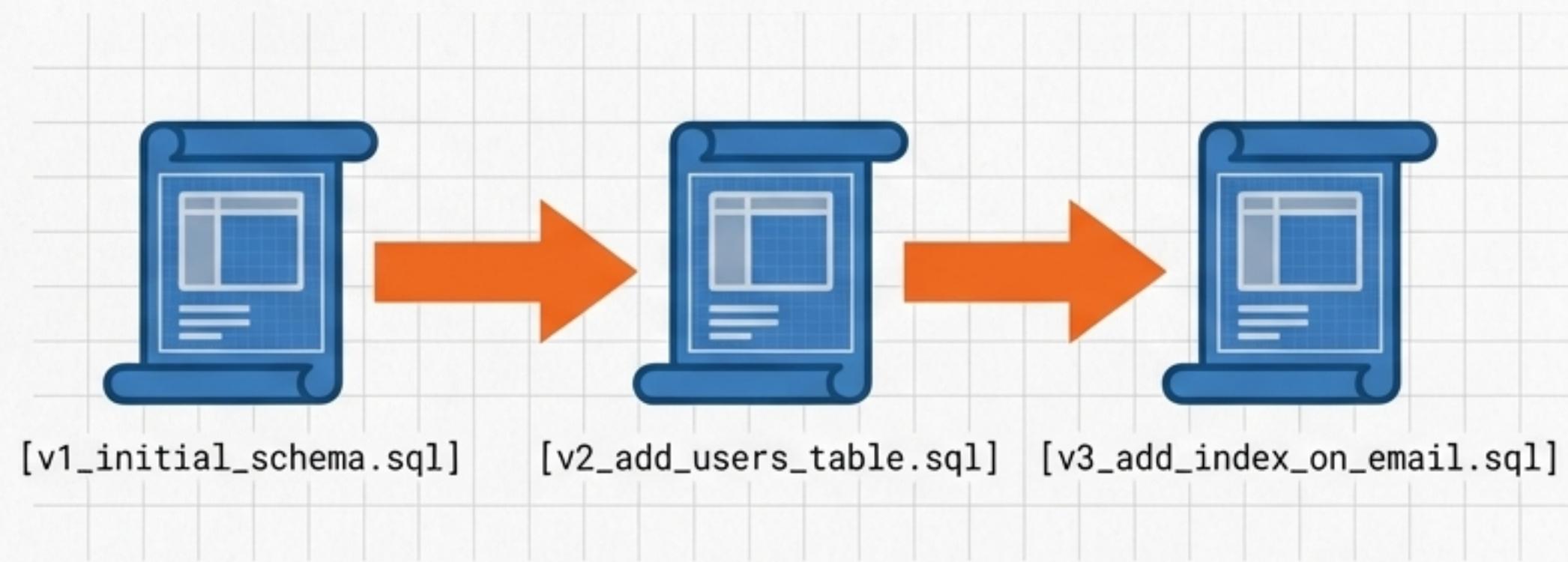


Inconsistencia, pérdida de tiempo
y errores en producción.

Solución Parte 1: Migraciones, el ‘Git’ de tu Base de Datos

Una migración es un archivo que describe un cambio específico en la estructura de tu base de datos.

Juntas, forman un historial versionado y ejecutable de tu esquema.



BENEFICIOS CLAVE

- 📄 **Consistencia:** Asegura que todos los miembros del equipo tengan exactamente la misma estructura de DB.
- 🔍 **Trazabilidad:** Si algo falla, sabemos exactamente qué cambio lo causó y cuándo.
- ⚙️ **Automatización:** Permite construir y modificar entornos de forma programática y confiable.

Solución Parte 2: Seeds, Datos de Prueba Inteligentes

Un script de 'seed' (semilla) puebla tu base de datos con un conjunto de datos predefinido, consistente y limpio.

¿Para Qué Sirven?



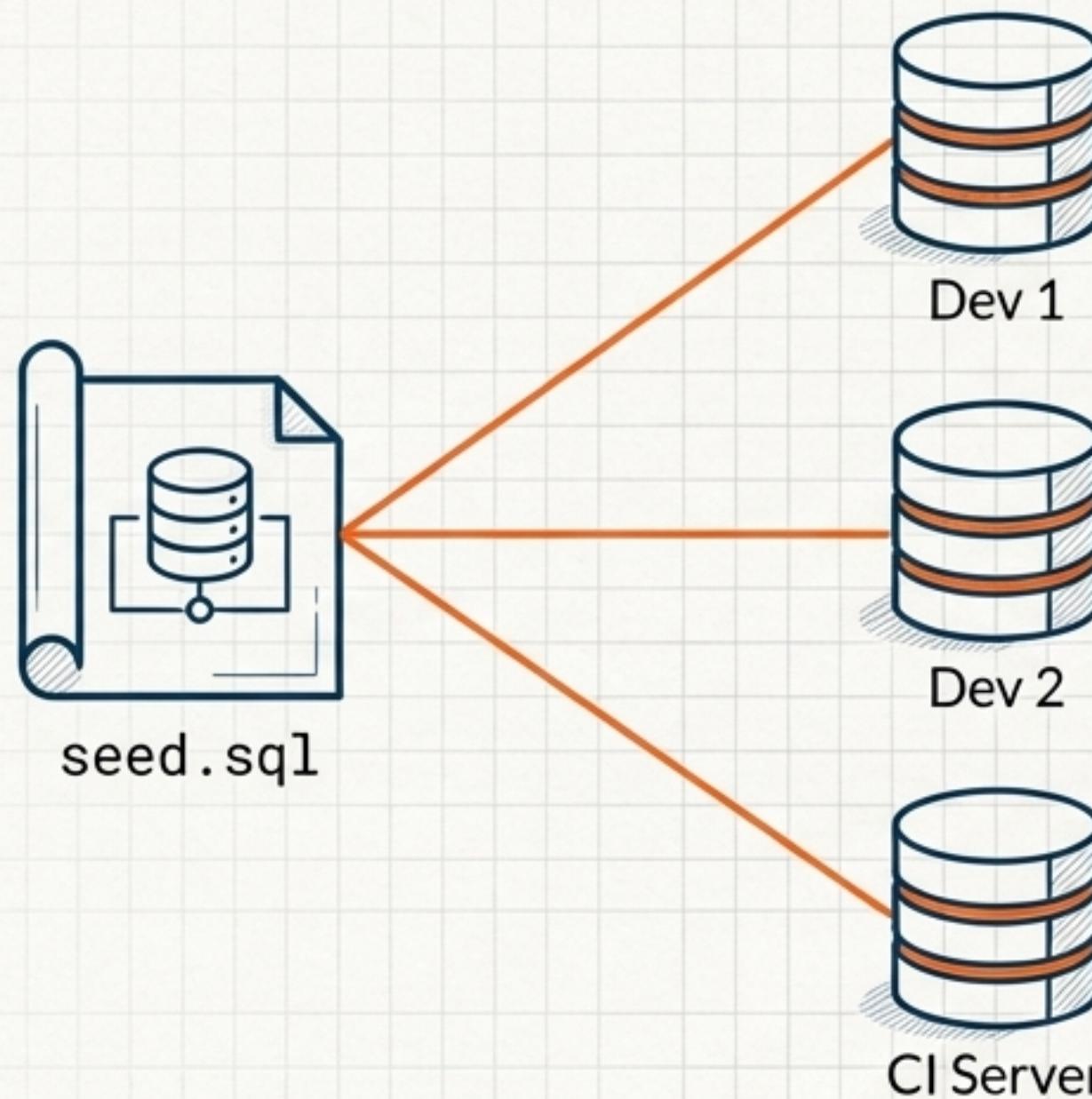
Desarrollo Rápido: Probar endpoints sin inventar datos manualmente cada vez.



Reproducción de Bugs: Recrear las condiciones exactas que causan un error.

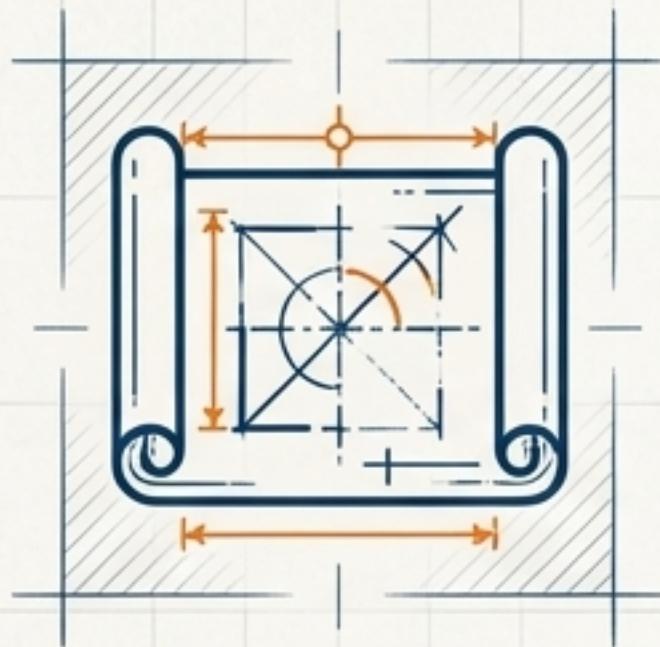


Pruebas Automatizadas: Ejecutar tests sobre un estado conocido y predecible.



El Principio de la Reproducibilidad: Si ejecutas el `seed.sql` hoy o mañana, el resultado es idéntico. Se acabaron las llaves foráneas rotas y los datos huérfanos.

El Plano Arquitectónico: Tus Entregables de Hoy



`db/schema.sql`

El plano maestro.

Define todas tus tablas, columnas, Primary Keys (PK), Foreign Keys (FK) y Constraints (restricciones).

Alternativa: `migrations/001_init.sql`.



`db/seed.sql`

Los datos de cimentación.

Inserta entre 5 y 10 filas de datos coherentes por cada tabla.



`docs/data_dictionary.md`

La leyenda del plano.

Un documento simple que describe cada tabla: campo, tipo de dato, ejemplo y reglas de negocio.

EVIDENCIA DE CONSTRUCCIÓN (lo que presentarás al final): Una captura de pantalla
Una captura de pantalla mostrando el resultado de los comandos `\dt` y `SELECT COUNT(*)`.

La Lógica de Construcción: Respetando el Orden de Inserción

Para evitar errores de Foreign Key Constraint, debes insertar los datos en un orden lógico que respete las dependencias.



Nota Técnica Importante:

Si la base de datos te da un error de Foreign Key, ¡es una buena señal! Significa que tus 'platos' (constraints) son correctos y te están protegiendo de datos inconsistentes.



Tablas de Relación / Transaccionales.
(Ej: `OrdenesDeCompra`, `ItemsDeOrden`).
Dependen de las tablas hijo.

Tablas Hijo.
(Ej: `Productos`, `Usuarios`).
Dependen de las tablas padre.

Tablas Padre / Catálogos.
(Ej: `Categorías`, `Roles`, `Estatus`).
No dependen de nadie.

Tu Responsabilidad como Arquitecto: Justifica tus Decisiones

Política sobre el uso de IA y el 'Copiar/Pegar'.

REGLAS DEL JUEGO



Permitido: Usar IA para generar plantillas (boilerplate) o datos de ejemplo masivos.



Obligatorio: Ser capaz de explicar y justificar tus decisiones de diseño. El código es tu responsabilidad.



Prepárate para explicar:

- **Tipos de Datos:** ¿Por qué `VARCHAR(100)` para un email y no `TEXT`? ¿Por qué `TIMESTAMP` y no `DATE`?
- **Constraints:** Justifica al menos 2 decisiones de diseño.

Ejemplo 1: ¿Por qué `UNIQUE` en la columna de `email`?

Ejemplo 2: ¿Por qué una restricción `CHECK (stock >= 0)`?

Paso 1: Verificación de Cimientos (Docker Sanity Check)

Objetivo: Confirmar que lo mínimo indispensable está funcionando:
Docker está activo y responde.

```
# Verifica la versión de Docker Engine
$ docker --version
Docker version 25.0.3, build 4debf41

# Verifica la versión de Docker Compose
$ docker compose version
Docker Compose version v2.24.5

# Lista los contenedores activos (no debe mostrar errores)
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

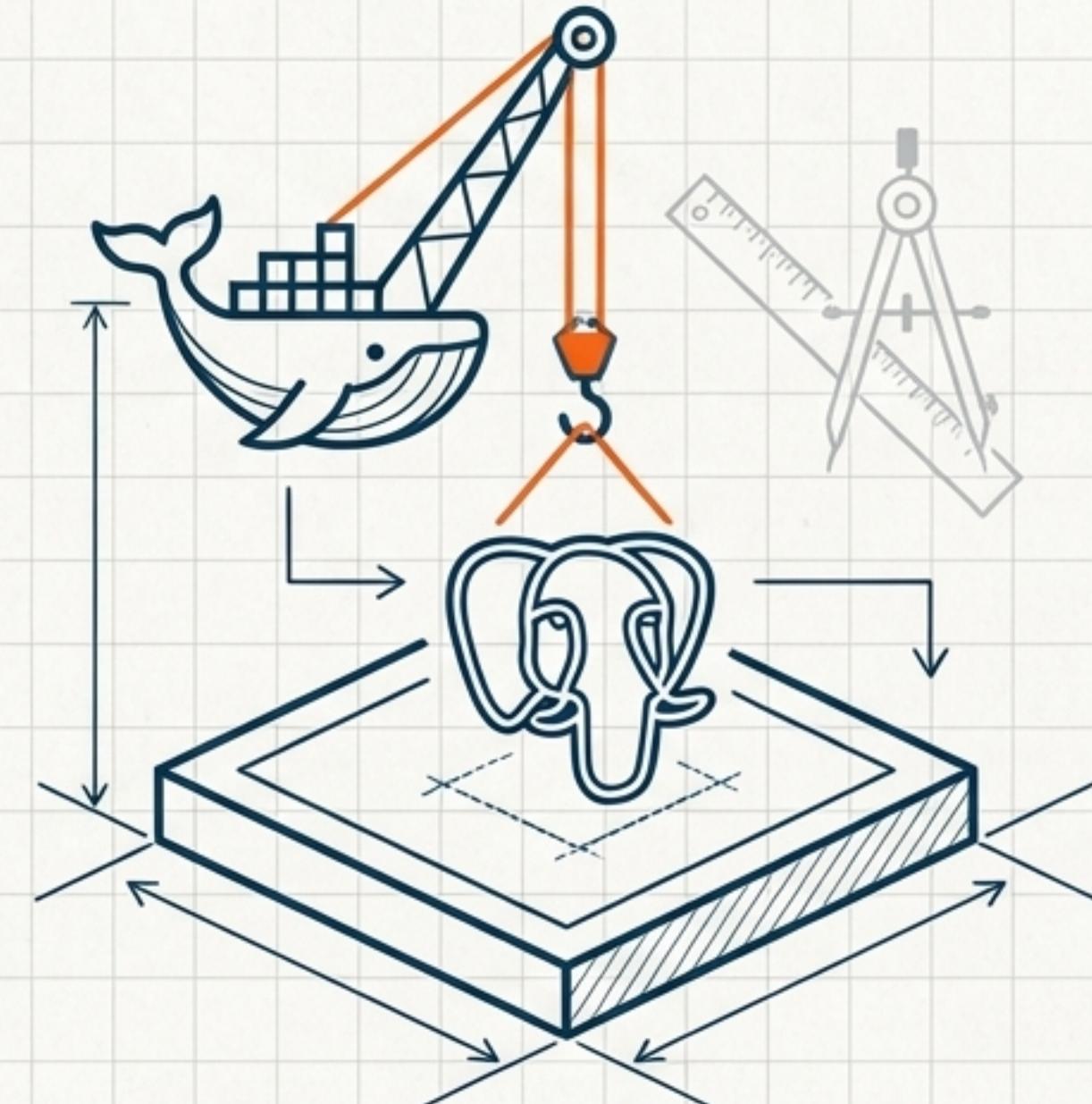


Los comandos se ejecutan sin errores y devuelven las versiones correspondientes.

Paso 2: Levantando la Estructura con Docker Compose

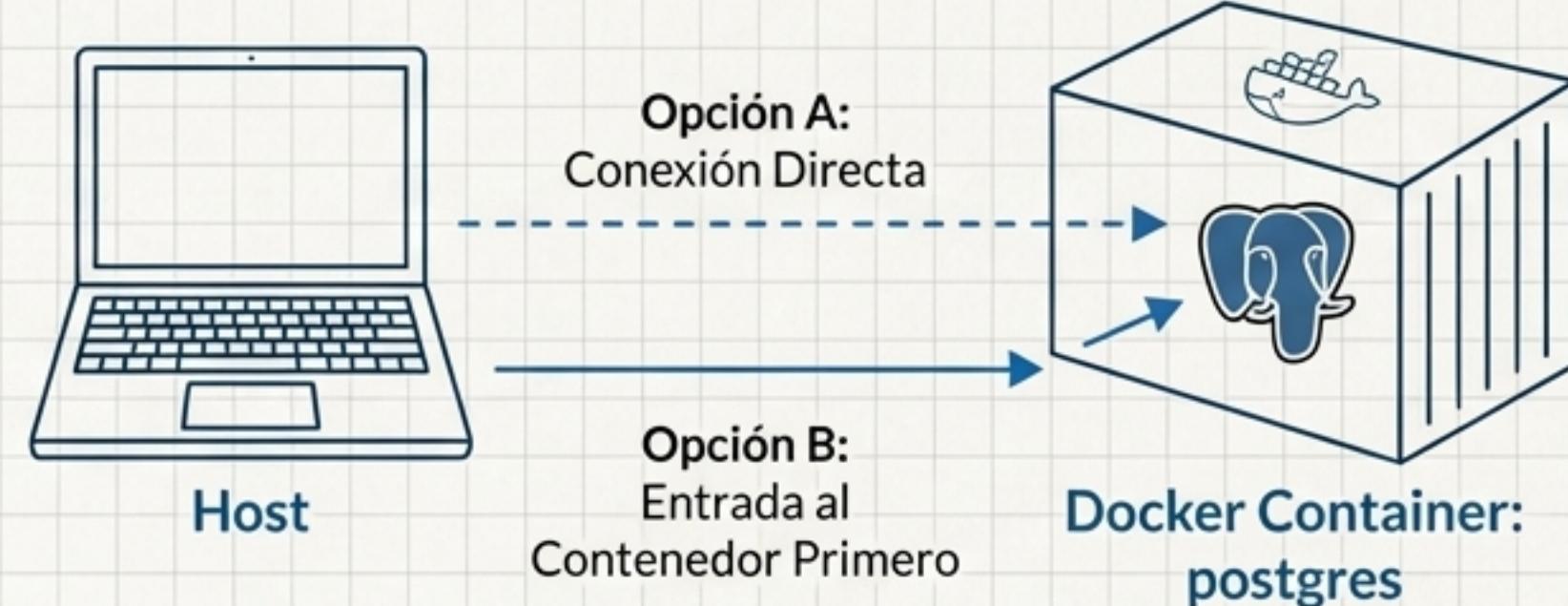
Objetivo: Iniciar un contenedor de PostgreSQL y asegurarse de que esté corriendo correctamente.

```
*1. Levantar servicios en segundo plano:**  
$ docker compose up -d  
Service starting...  
Service starting...  
  
*2. Verificar que el contenedor está corriendo:**  
$ docker ps  
CONTAINER      NAME      PROMOTT      DEFAULT      TIME  
postgres      'postgres'  active      4 minnutes ago  14h 5ms  
  
✓ *(Busca un contenedor con el nombre 'postgres' en la salida).*  
  
3. Ver los logs (bitácora) en tiempo real (opcional, para depurar):**  
$ docker compose logs -f postgres  
Log streams = docker compose logs -f postgres
```



Paso 3: Conectándose a PostgreSQL con `psql`

Objetivo: Acceder a la línea de comandos de PostgreSQL para poder ejecutar queries y scripts.



Opción A (Recomendada, desde tu máquina host)

Conecta `psql` localmente al puerto expuesto del contenedor.

```
psql -h localhost -p 5432 -U postgres
```

Opción B (Avanzada, entrando al contenedor primero)

Ejecuta `psql` dentro del contenedor mediante `docker exec`.

```
docker exec -it <nombre-del-contenedor> psql -U postgres
```

Comandos Esenciales dentro de `psql`



`\l`
Listar todas las bases de datos.



`\dt`
Ver la lista de tablas en la base de datos actual.



`\d nombre_tabla`
Describir la estructura de una tabla específica.



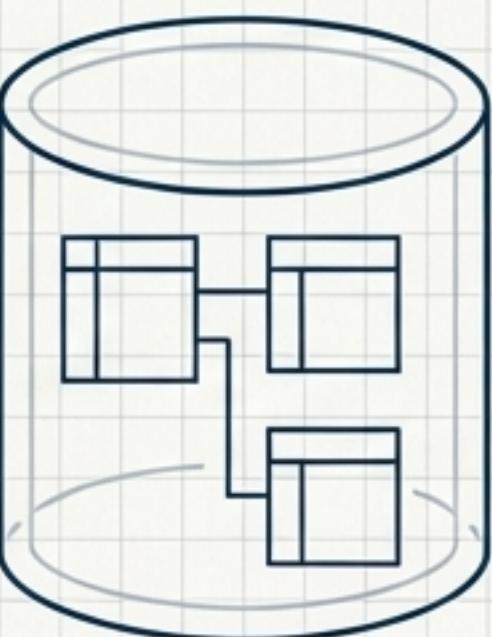
`\q`
Salir de psql.

Paso 4: Ejecutando los Planos (Schema & Seeds)

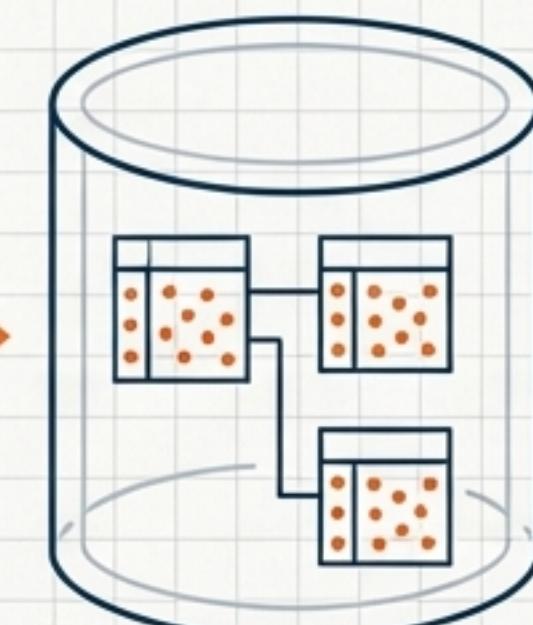


Objetivo: Crear la estructura de la base de datos y cargar los datos de prueba iniciales.

Step 1



Step 2



```
**(dentro de la sesión de `psql`)

postgres=# \i db/schema.sql
Cargar la estructura (el plano)

postgres=# \i db/seed.sql
Cargar los datos (la cimentación)
```

¡VERIFICA TU TRABAJO!

-- Revisa que las tablas se hayan creado
postgres=# \dt

-- Cuenta las filas en una tabla
postgres=# **SELECT COUNT(*) FROM nombre_de_tu_tabla;**

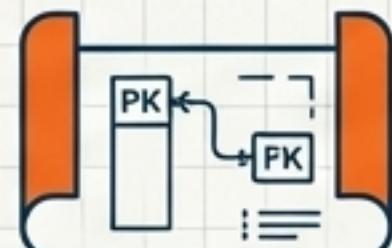


Misión de Hoy: Checklist del Arquitecto

Lograr una base de datos 100% reconstruible con datos coherentes.



Entorno Listo: Docker está OK y el contenedor de PostgreSQL está corriendo exitosamente.



`schema.sql` Creado: El archivo define todas tus tablas, PKs, FKs y al menos 2 constraints significativos.



`seed.sql` Funcional: El archivo inserta entre 5 y 10 filas por tabla, respetando el orden de dependencias.



Diccionario de Datos Completo: El archivo *data_dictionary.md* está creado y documenta tus entidades.

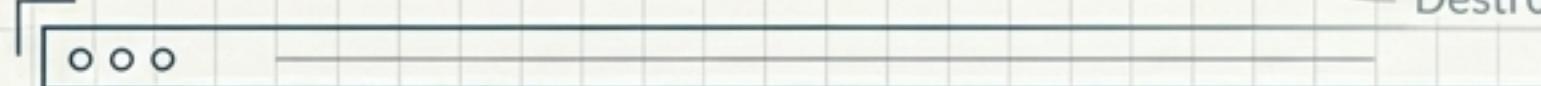


Evidencia Capturada: Tienes una captura de pantalla del output de `\\dt` y de al menos un `SELECT COUNT(*)`.



Éxito Alcanzado: La Verdadera Prueba de un Arquitecto

**Si borro todo mi entorno y ejecuto `docker compose up`,
`\\i schema.sql` y `\\i seed.sql` ,
el sistema vuelve a quedar
exactamente igual en segundos.**



PRÓXIMA CLASE

Ahora que tenemos una base de datos sólida y predecible, empezaremos a construir sobre ella.

Tema: Consultas SQL avanzadas (JOINs, agregaciones y reportes) sobre los datos que acabas de crear.

