



SISTEMAS DE BASE DE DATOS DISTRIBUIDAS

1. DATOS

ESTUDIANTE: Jeremy Ases

CARRERA: Tecnologías de la Información PARALELO: "A"

ASIGNATURA: NIVEL: 05

Semana 02

Las bases de datos distribuidas necesitan un gran computador por ende tiene alto costo de hardware

Base de datos Distribuida (DDB)

es una base de datos tradicional dividida en diferentes partes físicas dispersas que se acceden de manera lógica

Transparencia, es decir que cuando hay un cambio en un nodo se refleja en los demás.

si no hay la transparencia no puede ser una base de datos distribuida

los nodos pueden ser → servidores o (móvil, tablet, computador.

)

DDBMS (Distributed Database Management System)

SGBD (data base management system o DBMS)

Medios de comunicación → la red

Procesador de transacciones → el gestor de base de datos (SGBD)

Procesador de datos → Esto va a estar en el servidor, se debería configurar para poder ejecutar las consultas, recuperación, actualización y manejo físico de los datos

Características de un SMBD BDD

Recibe la solicitud de una aplicación, valida analiza y descompone la solicitud en varias operaciones para evitar errores en el sistema.

Busca localiza lee y valida datos en el procesador de transacciones.

Valida los datos de conformidad con las condiciones, si las hay, es especificadas por la solicitud.

Todas estas actividad son transparentes para el usuario.

Clasificación de las BDD

Se clasifican en base a la distribución de los procesos y datos son soportados: DB centralizada y DB distribuida

DB centralizada un solo gestor llamado anfitrion

DB distribuida servidor de varios gestores en LAN, tiene procesamiento en múltiples sitios y datos en un solo sitio

En el gestor o DBMS radica la base de datos distribuida

Ventajas de Base de Datos Distribuidas

Los datos se localizan cerca del sitio de "mayor demanda"

Tiene acceso más rápido a los datos

Facilita el crecimiento

Independencia del procesador

Problemas que se presentan en las bases de datos distribuidas

El rendimiento puede verse afectado por la carga de trabajo

Mayor complejidad en el diseño e implementación del sistema

Niveles de transparencia de una base de datos distribuida

La transparencia se puede entender como la separación semántica de alto nivel de un sistema, la transparencia habla acerca de que el usuario sienta que es el único que está usando la base de datos

- Transparencia de distribución

Transparencia de distribución

Permite manejar una base de datos físicamente dispersa como si fuera centralizada, se reconocen 3 niveles de transparencia de distribución

- La transparencia de fragmentación
- La transparencia de ubicación.
- La transparencia de ubicación local.

Ejemplo

Crear una base de datos de la universidad con 3 extensiones

```

CREATE TABLE UNIVERSIDAD {
    ID_UNI VARCHAR(3) PRIMARY KEY,
    NOM_UNI VARCHAR(25) NOT NULL
}

INSERT INTO UNIVERSIDAD VALUES ('UTA',"UNIVERSIDAD TECNICA DE AMBATO");

CREATE TABLE CAMPUS{
    ID_CAM VARCHAR(3) PRIMARY KEY,
    NOM_CAM VARCHAR(10) NOT NULL,
    ID_UNI_PER VARCHAR(3) NOT NULL REFERENCES UNIVERSIDAD(ID_UNI)
}

INSERT INTO CAMPUS VALUES ('QUE','QUEROCHACA','UTA');
INSERT INTO CAMPUS VALUES ('ING','INGAHURCO','UTA');
INSERT INTO CAMPUS VALUES ('HUA','HUACHI CHICO','UTA');

CREATE TABLE CARRERAS{
    ID_CAR VARCHAR(3) PRIMARY KEY,
    NOM_CAR VARCHAR(10) NOT NULL,
    CAN_CUP INT NOT NULL,
    ID_CAM_PER VARCHAR(3) NOT NULL REFERENCES CAMPUS(ID_CAM)
}

INSERT INTO CARRERAS VALUES ('TI','TECNOLOGIAS',15,'HUA');
INSERT INTO CARRERAS VALUES ('SOT','SOFTWARE',20,'HUA');
INSERT INTO CARRERAS VALUES ('TEL','TELECOMUNICACIONES',25,'HUA');

INSERT INTO CARRERAS VALUES ('VET','VETERINARIA',15,'QUE');
INSERT INTO CARRERAS VALUES ('AGR','AGRONOMIA',20,'QUE');
INSERT INTO CARRERAS VALUES ('BIO','BIOTECNOLOGIA',25,'QUE');

INSERT INTO CARRERAS VALUES ('MED','MEDICINA',30,'ING');
INSERT INTO CARRERAS VALUES ('ENF','ENFERMERIA',20,'ING');
INSERT INTO CARRERAS VALUES ('FIS','FISIO',18,'ING');

SELECT NOM_CAR
FROM CARRERAS
WHERE CAM_PER = 'HUA';

SELECT NOM_CAR
FROM CARRERAS
WHERE CAM_PER = 'QUE'

```

```
SELECT NOM_CAR  
FROM CARRERAS  
WHERE CAM_PER = 'ING'
```

En una base de datos distribuida es mejor fragmentar de manera vertical (fragmentación vertical se aumentan las tablas, horizontal se agregan campos)

Según el nivel de transparencia de distribución puede tener 3 casos

1. Caso 1: La base de datos soporta transparencia de fragmentación

```
SELECT *  
FROM CARRERAS  
WHERE CAN_CUP >= 20;
```

2. Caso 2: La base de datos soporta transparencia de ubicación

```
SELECT *  
FROM CARRERAS  
WHERE ID_CAM_PER = 'HUA' AND  
CAN_CUP>=20 ;
```

UNION

```
SELECT *  
FROM CARRERAS  
WHERE ID_CAM_PER = 'ING' AND  
CAN_CUP>=20;
```

UNION

```
SELECT *  
FROM CARRERAS  
WHERE ID_CAM_PER = 'QUE' AND  
CAN_CUP>=20;
```

3. Caso 3: la ubicacion local



- 3 requisitos para tener una base de datos distribuida
 - Sistema operativo
 - Red
 - Sistema de base de datos

Clase 2

Transparencia de Transacción

Permite una transacción actualice datos en varios sitios de la red, garantiza que la transacción se realizará o completada en su totalidad o abortada, con lo cual se mantiene la integridad de la base de datos

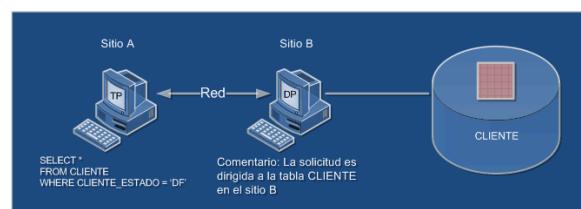
- Configuración de red importante

Formato de transacción:



Solicitud Remota

Trabaja en un solo gesto de base de datos



La sentencia o solicitud SQL puede hacer referencia a un solo sitio remoto

💡 Una transaccion se realiza con el begin y el commit
si hago mal la fragmentacion la transparencia va a alterar la chimba



para que la base de datos esté bien hecho y sea distribuida debe estar bien fragmentada

Una solicitud distribuida permite hacer referencia a datos de varios sitios de procesamiento de datos remotos

proporciona capacidades de procesamiento de base de datos totalmente distribuida:

- Dividir una tabla en varios fragmentos
- Hace referencia a uno o más de esos fragmentos solamente con una solicitud. Se tiene transparencia de fragmentación

Transparencia de replicación

Se refiere a que si existen copias de objetos de la base de datos, su existencia debe ser controlada por el sistema no por el usuario

Transparencia de falla

Es importante en una configuración de la base de datos para que esta no se caiga en caso de que falle un nodo y que sus funciones sean recobradas por otro nodo de la red



desventaja de la replicación: Consumo de recursos

Transparencia de Desempeño

Si hay varias solicitudes debemos hacer un englobamiento para todo lo que estamos pidiendo, reducir al minimo el costo total con la ejecucion de una solicitud

Optimizacion de consultas

orden de ejecucion optimos y seleccion de los sitios a ser accedidos para reducir el costo de comunicacion

con optimizacion de consulta automatica el gestor rolaiza la ruta de acceso barata sin intervencion de usuario

optimizacion de consulta manual: Requiere que la optimizacion sea seleccionada y programada por el usuario o programador.

optimizacion de consulta estatica

optimizacion de consulta dinamica

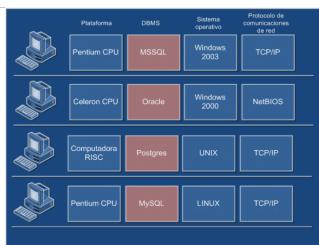
Basado en estadisticas

Transparencia de heterogenidad

Bases de Datos Distribuidas

Niveles de
transparencia de
una base de datos
distribuida

❖ Transparency de
heterogeneidad.



permite la integracion de gestores de base de datos locales diferentes

instalar 2 maquinas virtuales con ubuntu version 20 o 22

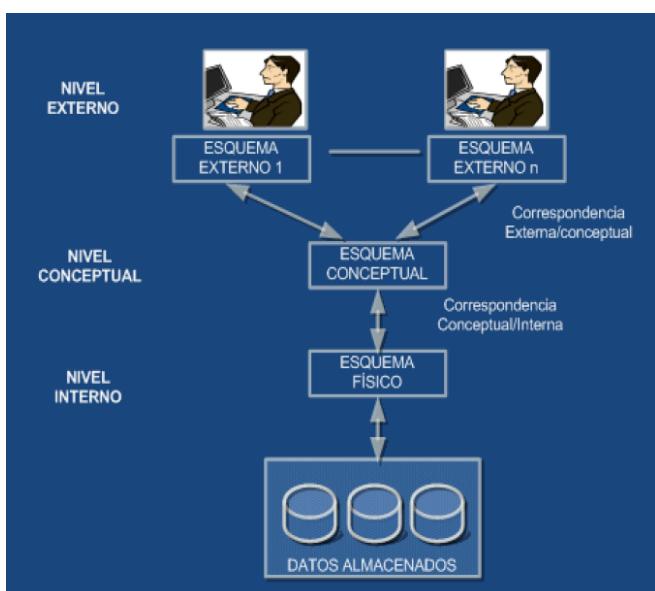
Semana 3

Clase 1



- En UTA_Distribuida, una consulta a global.extensions devuelve filas de Huachi, Ingahurco y Querochaca sin que el usuario conozca dónde vive cada fragmento. ¿Qué tipo de transparencia es la principal en juego?
- global.oferta_carrera une (UNION ALL) las tablas site_* .oferta_carrera y el usuario no escribe manualmente múltiples SELECT por sede. ¿Qué nivel de transparencia se ilustra?
- Una transacción inserta oferta en Querochaca y actualiza cupos en Huachi como una unidad atómica; si falla un sitio se revierte todo. ¿Qué transparencia se busca demostrar?
- Para simplificar consultas, se crean sinónimos carreras, extensiones, oferta que apuntan a las vistas globales; así el usuario no necesita prefijos de esquema. ¿Qué transparencia describe mejor este caso?

Arquitectura ANSI - SPARC de los sistemas de administracion de BDD



Si hay un esquema conceptual (es el diseño de las tablas) hay un esquema físico, el esquema físico es el script esquema externo ⇒ las replicaciones

Nivel interno: descibe la estructura física de la base de datos mediante un esquema interno

Nivel Conceptual: entidades atributos, relaciones, operaciones de los usuarios y restricciones

Nivel Externo : Varios esquemas externos o vistas



Nosotros trabajaremos en el esquema interno, el físico

Es mas facil manejar un error en el esquema global

Sistema de administración de base de datos distribuidas heterogéneos

Existen usuarios locales (interactua directamente con el gestor de base de datos) y globales

En el esquema global tengo todas las transacciones de todo lo que me pide el usuario.

si es heterogeneo se puede editar el modelacmiento de una base de datos

Diseño de una base de datos distribuidas

Diseño conceptual de la base de datos ⇒ Se definen las entidades y relaciones // este es el dolor de cabeza

Diseño lógico de la base de datos

Diseño físico en la base de datos

Diseño Fisico de una base de datos

Como dividir la base de datos en fragmentos

que fragmentos debemos replicar

Donde localizar estos fragmentos y replicas

Objetivos del diseño de la distribución de los datos

Procesamiento Local ⇒ Enfocarnos en donde van a estar los datos pero que este de forma cercana

Distribución de la carga de trabajo ⇒ maximizar el grado de ejecución de paralelismo de las aplicaciones

Costo de almacenamiento y disponibilidad ⇒ es más recomendable hacer en la nube.

Problemas de fragmentación

latencia

perdida de datos, replica de datos

Preguntas para hacer una base de datos distribuida

- ¿Por qué hacer una fragmentación de datos?

Para poder evitar la sobrecarga de procesamiento

- ¿Cómo realizar la fragmentación?

- ¿Qué tanto se debe fragmentar?

No hay que fragmentar mucho, hay de diseñar bien la fragmentación

- ¿Cómo probar la validez de una fragmentación?

Envío de consultas

- ¿Cómo realizar el asignamiento de fragmentos?

- ¿Cómo considerar los requerimientos de la información?

Fragmentacionn de una base de datos dsitribuida

Puedo fragmentar base datos del usuario

base de datos de sistema

Una Tabla

Cada fragmento puede ser guardado en cualquier sitio de una red de computadoras



Catálogo: de cada una de las fragmentaciones que va haciendo

Fragmentación horizontal

Dividir la data através de filas

Fragmentación vertical

Dividir por columnas (por los atributos)

Fragmentación mezclada

mezcla fragmentación vertical y horizontal

Fragmentación de una base de datos distribuida.

Tabla a fragmentar

CLI_NUM	CLI_NOM	CLI_DIR	CLI_EST	CLI_LIM	CLI_BAL	CLI_NIVEL	CLI_DEUDA
1	Selene Aguirre	Las palmas 34	DF	\$3,500.00	\$2,700.00	3	\$1,245.00
2	Martin Porres	Blvd. Lopez M	VE	\$6,000.00	\$12,000.00	1	\$0.00
3	Miriam Gutierrez	Águila 34	DF	\$4,000.00	\$3,500.00	3	\$3,400.00
4	Benito López	Rueda 23	VE	\$6,000.00	\$5,890.00	3	\$1,090.00
5	Victor Pérez	Carlos Carrillo	VE	\$1,200.00	\$550.00	1	\$0.00
6	Nicolás Rosas	20 de nov. 123	NL	\$2,000.00	\$350.00	2	\$50.00

Fragmentación horizontal de una base de datos distribuida.

Nombre del fragmento: CLI_H1								Ubicación: Distrito Federal		Nodo: DF	
CLI_NUM	CLI_NOM	CLI_DIR	CLI_EST	CLI_LIM	CLI_BAL	CLI_NIVEL	CLI_DEUDA				
1	Selene Aguirre	Las palmas 34	DF	\$3,500.00	\$2,700.00	3	\$1,245.00				
3	Miriam Gutierrez	Águila 34	DF	\$4,000.00	\$3,500.00	3	\$3,400.00				
Nombre del fragmento: CLI_H2								Ubicación: Nuevo León		Nodo: MY	
CLI_NUM	CLI_NOM	CLI_DIR	CLI_EST	CLI_LIM	CLI_BAL	CLI_NIVEL	CLI_DEUDA				
6	Nicolás Rosas	20 de nov. 123	NL	\$2,000.00	\$350.00	2	\$50.00				
Nombre del fragmento: CLI_H3								Ubicación: Veracruz		Nodo: XAL	
CLI_NUM	CLI_NOM	CLI_DIR	CLI_EST	CLI_LIM	CLI_BAL	CLI_NIVEL	CLI_DEUDA				
2	Martin Porres	Blvd. Lopez M	VE	\$6,000.00	\$12,000.00	1	\$0.00				
4	Benito López	Rueda 23	VE	\$6,000.00	\$5,890.00	3	\$1,090.00				
5	Victor Pérez	Carlos Carrillo	VE	\$1,200.00	\$550.00	1	\$0.00				

Fragmentación vertical de una base de datos distribuida.

The diagram illustrates the concept of vertical fragmentation in a distributed database. It shows two fragments, CLI_V1 and CLI_V2, each with its own schema and location.

Nombre del fragmento: CLI_V1 Ubicación: Edif. de servicio Nodo: ES

CLI_NUM	CLI_NOM	CLI_DIR	CLI_EST
1	Selene Aguirre	Las palmas 34	DF
2	Martin Porres	Bvd. Lopez Mz VE	
3	Miriam Gutierrez	Águila 34	DF
4	Benito López	Rueda 23	VE
5	Victor Pérez	Carlos Carrillo	VE
6	Nicolás Rosas	20 de nov. 123	NL

Nombre del fragmento: CLI_V2 Ubicación: Edif. De colección Nodo: EC

CLI_NUM	CLI_LIM	CLI_BAL	CLI_NIVEL	CLI_DEUDA
1	\$3,500.00	\$2,700.00	3	\$1,245.00
2	\$6,000.00	\$12,000.00	1	\$0.00
3	\$4,000.00	\$3,500.00	3	\$3,400.00
4	\$6,000.00	\$5,890.00	3	\$1,090.00
5	\$1,200.00	\$550.00	1	\$0.00
6	\$2,000.00	\$350.00	2	\$50.00



ES mucho mejor una fragmentación vertical

CLASE 2



Preguntas del día

(Opción múltiple) ¿Qué describe el nivel interno? A) Vistas de usuario y reportes B) Estructura física mediante un esquema interno C) Entidades, atributos y restricciones lógicas D) Reglas de negocio en la aplicación

El nivel conceptual se centra en entidades, atributos, relaciones, operaciones de usuario y restricciones.

¿Qué nombre reciben las "vistas de usuario" y en qué nivel se ubican?

¿Cuál de estos enunciados corresponde al nivel externo? A) Define índices y estructuras de almacenamiento en disco B) Proporciona múltiples esquemas orientados a distintos usuarios C) Define claves foráneas a nivel físico D) Determina direcciones de bloques en el disco

I. Interno — b) Estructura física II. Conceptual — c) Modelo lógico global III. Externo — a) Vistas de usuario

¿En qué nivel se documentan "entidades, atributos, relaciones y restricciones" del modelo global? A) Externo B) Conceptual C) Interno D) Físico-operativo

Menciona dos ejemplos de decisiones típicas del nivel interno.

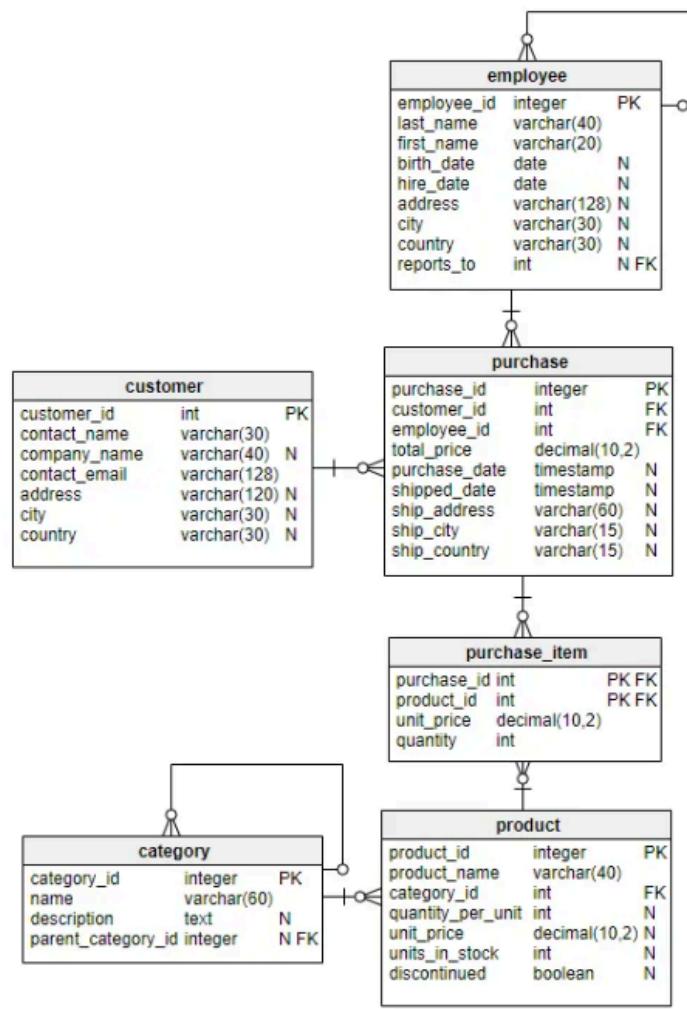
¿Qué afirmación es correcta respecto a la separación entre niveles? A) Los usuarios finales acceden directamente al nivel interno B) El nivel conceptual actúa como puente entre externo e interno C) El nivel externo controla los buffers de disco D) El nivel interno define las vistas de seguridad lógicas

(Caso breve) Una analista necesita una vista con solo Cliente(nombre, ciudad) para un grupo comercial, sin modificar la base física. a) ¿Qué nivel crea esa vista? b) ¿Qué nivel permanece estable aunque cambien índices y archivos?

En conexiones remotas se crea una tabla temporal con from

Cuando te dan las tablas es el nivel conceptual

Taller con la base de tienda



Creación de la tabla

```

CREATE TABLE employee (
    employee_id INT PRIMARY KEY,
    last_name VARCHAR(40) NOT NULL,
    first_name VARCHAR(20) NOT NULL,
    birth_date DATE,
    hire_date DATE,
    address VARCHAR(128),
    city VARCHAR(30),
    country VARCHAR(30),
    reports_to INT,
    FOREIGN KEY (reports_to) REFERENCES employee(employee_id)
);

-- Create the customer table
CREATE TABLE customer (
    customer_id INT PRIMARY KEY,

```

```

contact_name VARCHAR(30),
company_name VARCHAR(40),
contact_email VARCHAR(128),
address VARCHAR(120),
city VARCHAR(30),
country VARCHAR(30)
);

-- Create the category table
CREATE TABLE category (
    category_id INT PRIMARY KEY,
    name VARCHAR(50),
    description TEXT,
    parent_category_id INT,
    FOREIGN KEY (parent_category_id) REFERENCES category(category_id)
);

-- Create the product table
CREATE TABLE product (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(40),
    category_id INT,
    quantity_per_unit VARCHAR(20),
    unit_price DECIMAL(10, 2),
    units_in_stock INT,
    discontinued BOOLEAN,
    FOREIGN KEY (category_id) REFERENCES category(category_id)
);

-- Create the purchase table
CREATE TABLE purchase (
    purchase_id INT PRIMARY KEY,
    customer_id INT,
    employee_id INT,
    total_price DECIMAL(10, 2),
    purchase_date TIMESTAMP,
    shipped_date TIMESTAMP,
    ship_address VARCHAR(60),
    ship_city VARCHAR(15),
    ship_country VARCHAR(15),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (employee_id) REFERENCES employee(employee_id)
);

-- Create the purchase_item table
CREATE TABLE purchase_item (
    purchase_id INT,
    product_id INT,

```

```
unit_price DECIMAL(10, 2),  
quantity INT,  
PRIMARY KEY (purchase_id, product_id),  
FOREIGN KEY (purchase_id) REFERENCES purchase(purchase_id),  
FOREIGN KEY (product_id) REFERENCES product(product_id)  
);
```

Primera consulta

Segunda Consulta

Tercera Consulta

Ejercicio 12: Mostrar los productos comprados con su cantidad máxima comprada

viernes, 12 de septiembre de 2025 8:35

Ejercicio:

Para cada producto adquirido, muestre su nombre, la cantidad máxima comprada y el número de compras realizadas. Muestre tres columnas: `product_name`, `quantity`, y `purchases_number`.

SEMANA 4

Clase 1

The screenshot shows a web-based quiz titled "Introducción a Base de datos distribuida". It includes a greeting message, three questions with answer input fields, and a sidebar with navigation icons.

Introducción a Base de datos distribuida

Hola, Jeremy. Cuando envíe este formulario, el propietario verá su nombre y dirección de correo electrónico.

1. ¿Qué estrategias se pueden utilizar para asegurar la confiabilidad y disponibilidad en una base de datos distribuida?

Escriba su respuesta

2. ¿Cuáles son las diferencias entre la replicación síncrona y asíncrona en una base de datos distribuida?

Escriba su respuesta

3. Se puede utilizar una API en las transacciones de base de datos distribuidas?

verdadero
 falso

Instancia nodo donde se guarda la data, y puedo crearlo con un SGBD

¿Qué es la replicación de datos?

Es cuando se realiza una copia total o parcial de una base de datos A, a otra base de datos B, se puede actualizar con una determinada periodicidad



¿Cómo se replica?

- A través de las transacciones
- A través de agentes ad-hoc

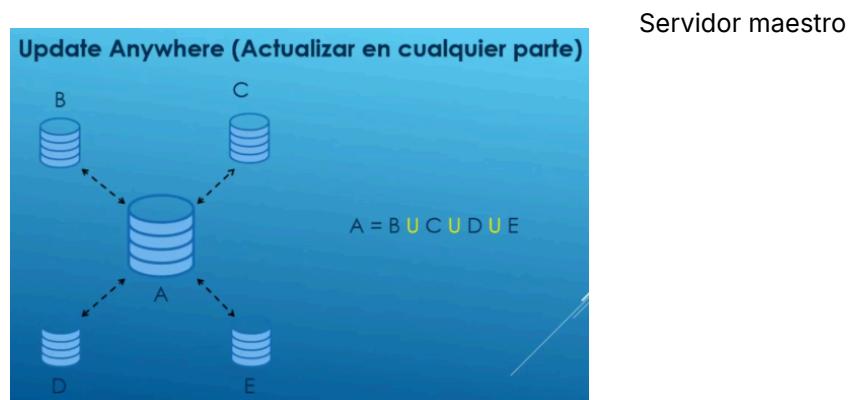
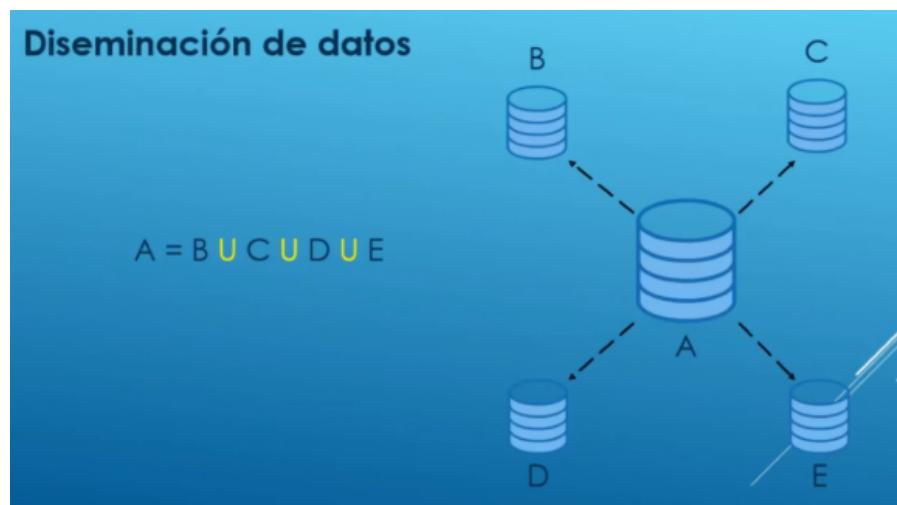
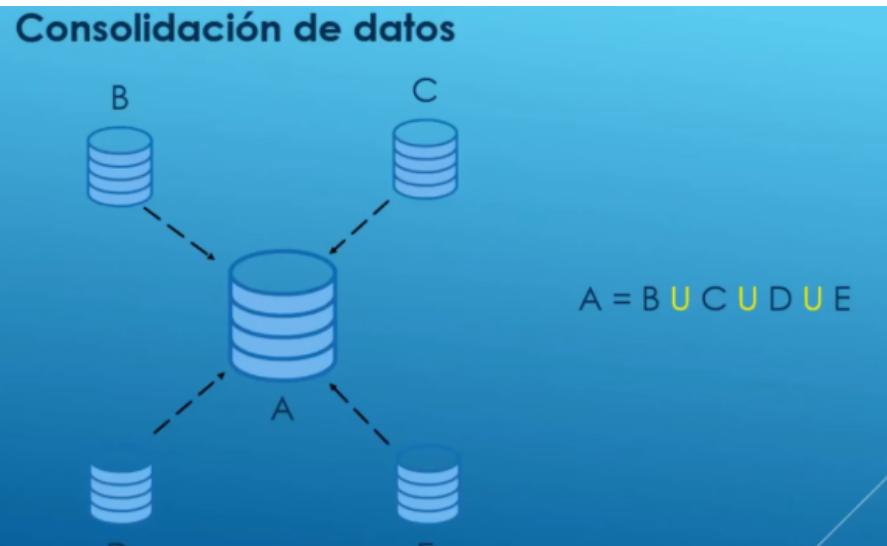
¿Cuando se replica?

- Despues de cada transacción - Asíncrona
- Simultáneamente en todos los nodos - Síncrona
- Periódicamente

Topologías de replicación de datos

Maestro - Esclavo





Para ahcer fragmetnacion debo hacerlo con criterios

SEMANA 5 CLASE 1 Y 2

1. Requisitos Previos

- Windows 10/11 Pro, Enterprise o Education (64-bit).

- **Nota:** En Windows Home, necesitarás **Docker Desktop con WSL 2** (Windows Subsystem for Linux).
- **Hyper-V habilitado** (necesario para Docker Desktop).
- **WSL 2** (recomendado para mejor rendimiento).

2. Pasos para Instalar Docker Desktop

A. Habilitar WSL 2 (Opcional, pero recomendado)

1. Abre **PowerShell como Administrador** y ejecuta:

```
wsl --install
```

2.

3. Reinicia tu computadora.

B. Descargar Docker Desktop

1. Ve al sitio oficial: <https://www.docker.com/products/docker-desktop>.
2. Descarga el instalador para Windows (**Docker Desktop Installer.exe**).
- 3.

C. Instalar Docker Desktop

1. Ejecuta el instalador descargado.
2. Acepta los términos y selecciona:
 - **Install required Windows components for WSL 2** (si usas WSL 2).
1. Haz clic en **OK** y espera a que termine la instalación.
2. Reinicia tu PC cuando se solicite.
- 3.

D. Iniciar Docker Desktop

1. Busca **Docker Desktop** en el menú de inicio y ábrelo.
2. Espera a que Docker se inicie (verás el ícono de Docker en la barra de tareas).
3. Si es la primera vez, te pedirá aceptar los términos.

3. Verificar la Instalación

Abre **PowerShell o CMD** y ejecuta:

```
docker --version
```

Deberías ver algo como:

```
Docker version 24.0.7, build 1110ad01
```

Prueba un contenedor de ejemplo:

```
docker run hello-world
```

Si funciona, verás un mensaje de confirmación:

```
Hello from Docker!
```

This message shows that your installation appears to be working correctly.

4. Configuración Adicional (Opcional)

A. Usar WSL 2 como Backend (Recomendado)

1. Abre **Docker Desktop**.
2. Ve a **Settings (⚙️) > General**.
3. Marca:
 - **Use the WSL 2 based engine.**
1. En **Resources > WSL Integration**, activa la integración con tu distribución WSL (ej: Ubuntu).

B. Aumentar Recursos (Opcional)

Si usas contenedores pesados (BDD, Kubernetes):

1. Ve a **Settings > Resources**.
2. Ajusta:
 - CPUs: 4-8 (según tu PC).
 - Memoria: 4GB-8GB.
 - Swap: 1GB-2GB.

5. Solución de Problemas Comunes

Error: "Docker Desktop requires WSL 2"

- Asegúrate de que WSL 2 esté instalado:
Powershell:
 - wsl --set-default-version 2
- Si falla, actualiza el kernel de WSL: [Instrucciones de Microsoft](#).

Error: "Hyper-V is not enabled"

1. Abre **Panel de Control > Programas > Activar o desactivar características de Windows**.
2. Marca **Hyper-V y Plataforma de máquina virtual**.
3. Reinicia tu PC.

Docker no inicia en Windows Home

- Descarga **WSL 2** manualmente y usa Docker con WSL 2 backend.

6. Alternativa: Docker Toolbox (Para Windows Antiguos)

Si tu Windows no soporta WSL 2 (ej: Windows 7/8):

1. Descarga **Docker Toolbox**: https://docs.docker.com/toolbox/toolbox_install_windows/.
2. Usa VirtualBox para crear una máquina virtual con Docker.

Conclusión

- ✓ Docker Desktop está listo para usar en Windows con WSL 2 (mejor rendimiento).
- ✓ Puedes crear contenedores, redes y volúmenes fácilmente.
- ✓ Para bases de datos distribuidas, usa docker-compose.yml para gestionar múltiples nodos.

Configuración de un Nodo Distribuido en Docker

Objetivo: Crear un entorno distribuido con múltiples nodos (contenedores Docker) que simulen una arquitectura de bases de datos distribuidas (BDD), como PostgreSQL, MongoDB o Cassandra.

Pasos para Configurar un Nodo Distribuido

1. Instalar Docker

Asegúrate de tener Docker instalado en tu sistema:

```
sudo apt-get update && sudo apt-get install docker.io docker-compose
```

2. Crear una Red Docker para Comunicación

Los nodos deben comunicarse entre sí. Crea una red bridge:

```
docker network create bdd-distribuida
```

3. Configurar Nodos con Docker Compose

Ejemplo para **3 nodos PostgreSQL** (docker-compose.yml):

```
version: '3.8'

services:
  postgres-node1:
    image: postgres:14
    container_name: node1
    environment:
      POSTGRES_PASSWORD: admin123
      POSTGRES_USER: admin
      POSTGRES_DB: bdd_distribuida
    ports:
      - "5432:5432"
    networks:
      - bdd-distribuida
  postgres-node2:
    image: postgres:14
    container_name: node2
    environment:
      POSTGRES_PASSWORD: admin123
      POSTGRES_USER: admin
      POSTGRES_DB: bdd_distribuida
    ports:
      - "5433:5432"
    networks:
      - bdd-distribuida
  postgres-node3:
    image: postgres:14
```

```
container_name: node3
environment:
  POSTGRES_PASSWORD: admin123
  POSTGRES_USER: admin
  POSTGRES_DB: bdd_distribuida
ports:
  - "5434:5432"
networks:
  - bdd-distribuida
networks:
  bdd-distribuida:
    external: true
```

4. Iniciar los Contenedores

```
docker-compose up -d
```

Verifica que los nodos estén activos:

```
docker ps
```

5. Configurar la Distribución de Datos

Ejemplo con PostgreSQL (Fragmentación Horizontal)

1. Conéctate a un nodo:

```
docker exec -it node1 psql -U admin -d bdd_distribuida
```

2. Crea tablas fragmentadas y configura la replicación:

- - En node1 (Fragmento 1)

```
CREATE TABLE clientes_region1 (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100),
  region VARCHAR(50) CHECK (region = 'Norte')
);
```

- - En node2 (Fragmento 2)

```
CREATE TABLE clientes_region2 (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100),
  region VARCHAR(50) CHECK (region = 'Sur')
);
```

3. Usa **Foreign Data Wrappers** para consultas distribuidas:

```
CREATE EXTENSION postgres_fdw;
```

- - En node1: Configura acceso a node2

```
CREATE SERVER node2_server FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'node2', dbname 'bdd_distribuida');
CREATE USER MAPPING FOR admin SERVER node2_server
OPTIONS (user 'admin', password 'admin123');
```

6. Probar la Distribución

- Inserta datos en cada nodo y verifica su distribución:

- - En node1

```
INSERT INTO clientes_region1 (nombre, region) VALUES ('Empresa A', 'Norte');
```

- - En node2

```
INSERT INTO clientes_region2 (nombre, region) VALUES ('Empresa B', 'Sur');
```

- Consulta federada desde node1:

```
SELECT * FROM clientes_region1
```

```
UNION ALL
```

```
SELECT * FROM clientes_region2;
```

7. Configuración Avanzada

Para Bases de Datos NoSQL (MongoDB/Cassandra)

- **MongoDB:** Usa réplicas con --replicaSet:

```
docker run --name mongo-node1 --network bdd-distribuida -d mongo:5 --replicaSet rs0
```

- **Cassandra:** Configura un cluster con CASSANDRA_SEEDS:

```
docker run --name cassandra-node1 --network bdd-distribuida -e
CASSANDRA_CLUSTER_NAME=BDD_Cluster -d cassandra:4
```

8. Monitoreo

- Verifica la comunicación entre nodos:

```
docker exec -it node1 ping node2
```

- Usa herramientas como **PgAdmin** (para PostgreSQL) o **MongoDB Compass** para visualizar los datos distribuidos.

Conclusión

Con Docker, puedes simular un entorno distribuido completo para:

Fragmentación (sharding)

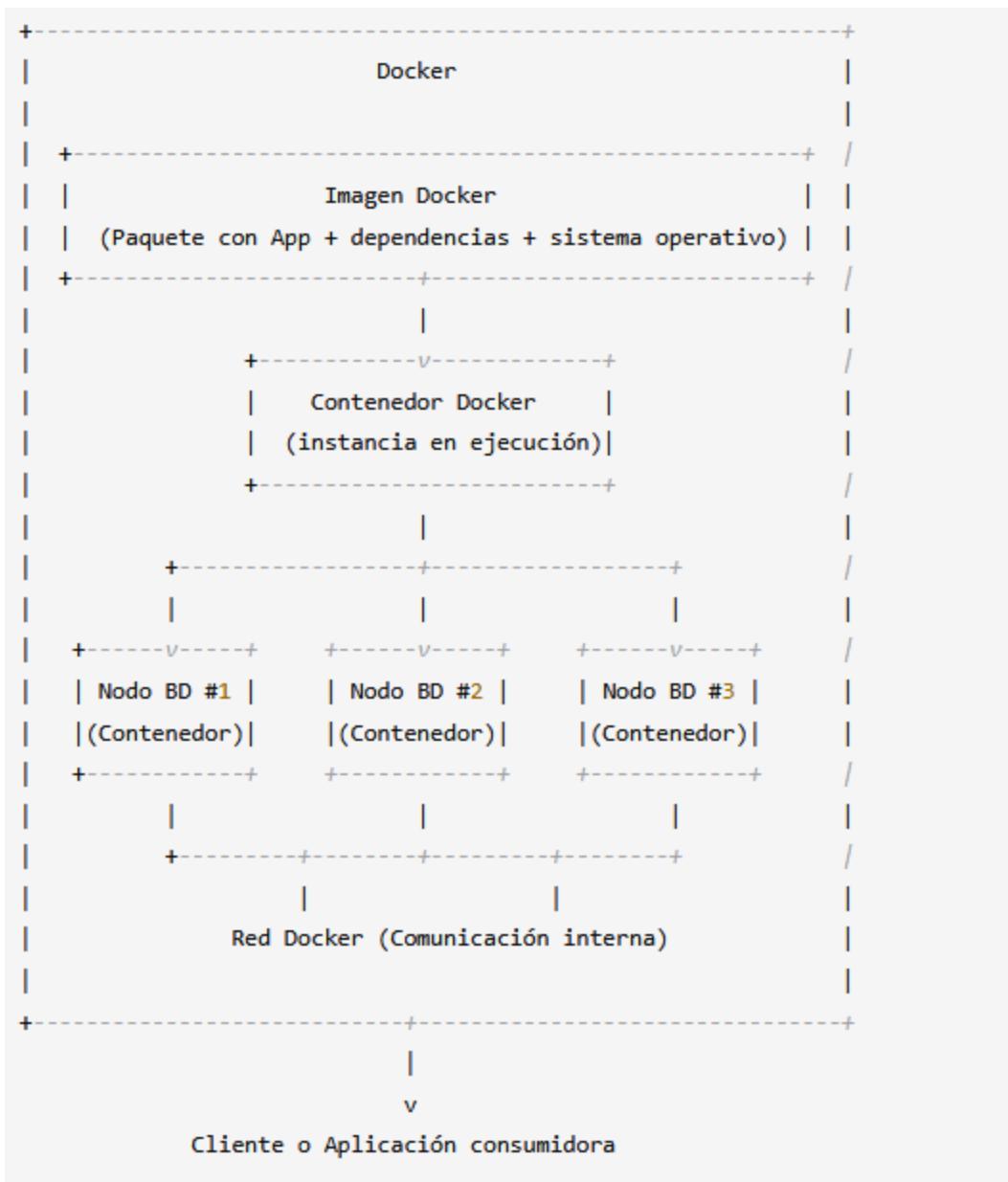
Replicación (alta disponibilidad)

Consultas federadas

Pruebas de escalabilidad

Nota: Para producción, considera orquestadores como **Kubernetes** y configuraciones de seguridad avanzadas (TLS, autenticación).

INSTALACION Y CONFIGURACION DE UN NODO EN DOCKER



Uso de Docker en bases de datos distribuidas:

Docker facilita el despliegue rápido, la escalabilidad horizontal, el aislamiento de dependencias y la consistencia en ambientes distribuidos. Al contenerizar bases de datos distribuidas, obtienes:

- **Escalabilidad:** Puedes añadir rápidamente nodos adicionales según la carga.
 - **Consistencia de ambientes:** Los ambientes de desarrollo, pruebas y producción son homogéneos.
 - **Alta Disponibilidad y Resiliencia:** Si falla un nodo, se crean o eliminan contenedores automáticamente.
 - **Sencillez operativa:** Simplifica la gestión y administración de bases de datos distribuidas mediante herramientas de orquestación como Docker Compose, Kubernetes o Docker Swarm.

SEMANA 6

Proyecto

Ejercicio de fragmentación horizontal en PostgreSQL.(PgAdmin) Base flotilla

Qué es almacenamiento distribuido mediante sharding?

El **sharding** es un método de almacenamiento distribuido en bases de datos, donde la información se divide en fragmentos (o shards) horizontales almacenados en diferentes nodos.

Cada shard guarda un subconjunto del total de datos. De esta forma, se logra:

- **Escalabilidad horizontal:** Puedes añadir más nodos (shards) conforme crecen tus datos.
- **Rendimiento:** Las consultas se reparten entre múltiples nodos.
- **Alta disponibilidad:** Reducción de puntos únicos de fallo.

💡 ¿Cómo funciona sharding?

El sharding divide una colección o tabla en múltiples fragmentos según un criterio (clave de shard), y estos fragmentos se almacenan en diferentes nodos.

Ejemplo visual:

```
Base de datos original (sin sharding):
[ A | B | C | D | E | F ]  
  
Base distribuida con Sharding:
Shard 1: [ A | B ]
Shard 2: [ C | D ]
Shard 3: [ E | F ]
```

🚀 Ejemplo práctico de sharding con MongoDB (paso a paso)

Este ejemplo básico utiliza MongoDB para demostrar cómo implementar almacenamiento distribuido mediante sharding.

✓ Paso 1: Instalar MongoDB

Asegúrate de tener instalado MongoDB Community Server:

<https://www.mongodb.com/try/download/community> <https://www.mongodb.com/try/download/shell>

✓ Paso 2: Ejecutar instancias de MongoDB para shards

Crea directorios para shards:

```
mkdir -p ~/mongodb/shard1 ~/mongodb/shard2 ~/mongodb/config
```

- -para windows

```
mkdir "%USERPROFILE%\mongodb\shard1" "%USERPROFILE%\mongodb\shard2"  
"%USERPROFILE%\mongodb\config"
```

Ejecuta cada shard (instancias MongoDB):

```
mongod --shardsvr --port 27018 --dbpath ~/mongodb/shard1 --repSet shard1 --bind_ip localhost  
mongod --shardsvr --port 27019 --dbpath ~/mongodb/shard2 --repSet shard2 --bind_ip localhost
```

Ejecuta estos comandos en terminales separadas.

✓ Paso 3: Inicializa Replica Sets (obligatorio en MongoDB shards)

Desde otra terminal, inicializa los replica sets de los shards.

```
mongo --port 27018
```

Dentro de Mongo shell:

```
rs.initiate({  
  _id: "shard1",  
  members: [{ _id: 0, host: "localhost:27018" }]  
})
```

Repite en puerto 27019:

```
mongo --port 27019
```

```
rs.initiate({  
  _id: "shard2",  
  members: [{ _id: 0, host: "localhost:27019" }]  
})
```

Paso 4: Ejecuta el servidor de configuración

El servidor de configuración guarda metadatos sobre cómo están distribuidos los shards.

```
mongod --configsvr --replicaSet configReplSet --port 27020 --dbpath ~/mongodb/config --bind_ip localhost
```

Inicializa el Replica Set del servidor de configuración:

Abrir en otra ventana

```
mongo --port 27020
```

```
rs.initiate({  
  _id: "configReplSet",  
  members: [{ _id: 0, host: "localhost:27020" }]  
})
```

Paso 5: Ejecutar MongoDB Router (mongos) abrir en otra terminal

```
mongos --configdb configReplSet/localhost:27020 --port 27017 --bind_ip localhost
```

Paso 6: Configurar los shards desde mongos

Conéctate al router mongos desde otra terminal:

```
mongo --port 27017
```

Añadir shards desde shell de MongoDB Router:

```
sh.addShard("shard1/localhost:27018")  
sh.addShard("shard2/localhost:27019")
```

Paso 7: Crear base de datos y colección shardeada

```
use universidad
```

```
sh.enableSharding("universidad")  
// shardear por campo "codigo"  
db.createCollection("estudiantes")
```

```
sh.shardCollection("universidad.estudiantes", { "codigo": 1 })
```

Paso 8: Insertar datos y verificar distribución

Inserta varios registros de estudiantes:

```
use universidad  
for(let i=1; i<=1000; i++){  
    db.estudiantes.insert({codigo:i, nombre:"Estudiante_"+i})  
}
```

Verifica distribución en shards:

```
db.estudiantes.getShardDistribution()
```

MongoDB mostrará cómo están distribuidos los datos en shards:

Shard shard1 at shard1/localhost:27018

data : 53KiB docs : 493 chunks : 2

estimated data per chunk : 26KiB

estimated docs per chunk : 246

Shard shard2 at shard2/localhost:27019

data : 54KiB docs : 507 chunks : 2

estimated data per chunk : 27KiB

estimated docs per chunk : 253

Conclusión

Con este ejemplo práctico implementaste sharding (almacenamiento distribuido) con MongoDB.

Beneficios obtenidos:

- **Escalabilidad horizontal:** puedes añadir más nodos fácilmente.
- **Distribución eficiente:** consultas distribuidas y rápidas.
- **Alta disponibilidad:** más resistencia ante fallos.

```
sudo apt install mongodb-server-core
```

```
Sudo apt install mongodb-clients
```

Usuario GitHub: joseru82@hotmail.com

Semana 7

Tratamiento de transacciones distribuidas (2PC, Saga).

1) ¿Qué problema resuelven?

En sistemas distribuidos queremos **atomicidad**: que una transacción que toca varios nodos/shards se haga "toda o nada". Hay dos enfoques:

- **2PC (Two-Phase Commit):** asegura *consistencia fuerte* coordinando a todos los participantes para que **o todos confirmen o todos aborten**.
- **Saga:** secuencia de **transacciones locales** (cada una hace commit en su propio servicio) con **operaciones compensatorias** que deshacen pasos previos si algo falla.

Proporciona *consistencia eventual* sin bloquear largo tiempo.

2) Two-Phase Commit (2PC)

Flujo

1. Fase de preparación (PREPARE)

- El **coordinador** envía PREPARE(tx) a todos.
- Cada **participante** ejecuta su trabajo en modo "preparado" (registro en log, bloqueos) y responde VOTE_COMMIT o VOTE_ABORT.

1. Fase de decisión (COMMIT/ABORT)

- Si **todos** votan commit → el coordinador difunde COMMIT(tx).
- Si **alguno** vota abort o hay timeout/falla → difunde ABORT(tx).

Ventajas

- Garantiza atomicidad fuerte entre múltiples nodos (ACID distribuido).

Costes/Riesgos

- **Bloqueo:** participantes quedan bloqueados hasta la decisión.
- **Coordinador único:** si cae tras PREPARE puede dejar a otros en estado **incerto** (protocolo *blocking*).
- Requiere logs de recuperación y timeouts; en redes inestables puede ser caro.

Úsalo cuando la invariante es estricta (p. ej., mover dinero entre cuentas) y los participantes son pocos y de baja latencia.

3) Sagas

Idea

Una **Saga** = T₁ ; T₂ ; ... ; T_n, donde cada T_i es una **transacción local** con su **compensación** C_i (que deshace T_i si la saga falla después).

Si algo falla en T_k, ejecutamos C(k-1), C(k-2), ..., C₁.

Estilos

- **Orquestada:** un *orchestrator* central envía comandos/pasos y, ante fallo, llama a compensaciones.
- **Coreografiada:** puro *event-driven*; cada servicio reacciona a eventos y publica los suyos.

Ventajas/Limitaciones

- **No bloquea** recursos largo tiempo; alta escalabilidad y tolerancia a particiones.
- Requiere **diseñar compensaciones** (idempotentes y seguras).
- No es ideal para invariantes que no toleran inconsistencias transitorias.

Úsalas en procesos de negocio largos (e-commerce: reservar stock → cobrar → programar envío), donde una devolución/refund compensa.