I. Fundamentos, Definición y Componentes Clave

Una BDD es un conjunto de bases de datos que están lógicamente interrelacionadas, pero se encuentran físicamente distribuidas en múltiples nodos o sitios conectados mediante una red de computadoras. A diferencia de los sistemas centralizados, en una BDD, la información se mantiene y gestiona de forma descentralizada.

El Sistema de Administración de Base de Datos Distribuida (SMBDD) es el *software* que coordina todo el sistema. Entre sus numerosos componentes, el más crítico es el Diccionario de Datos Distribuido (DDD), o catálogo global. El DDD almacena la información de metadatos de todo el sistema, incluyendo: el esquema conceptual global, la definición de todos los fragmentos creados, el esquema de asignación (la ubicación de cada fragmento), y los esquemas conceptuales y físicos locales de cada nodo. Otros componentes cruciales son el Procesador de Consultas Distribuido y el Administrador de Transacciones Distribuidas.

II. Características de un SMBDD y sus Ventajas/Desventajas

Un SMBDD debe cumplir un conjunto estricto de características que garantizan su funcionalidad en un entorno distribuido: Autonomía Local (cada sitio puede gestionar sus propios datos), Procesamiento de Consultas y Transacciones Distribuidas, Independencia del *Hardware*, Soporte de Concurrencia Distribuida y Recuperación de Fallos Distribuida.

Aspecto	Ventajas	Desventajas
Operación	Mayor Fiabilidad y Disponibilidad (los fallos de un nodo no detienen todo el sistema).	Mayor Complejidad de diseño y gestión del sistema.
Escalabilidad	Crecimiento Modular (se añaden sitios sin rehacer el sistema).	Altos Costos en el desarrollo y mantenimiento del *software* distribuido.
Rendimiento	Procesamiento Paralelo de consultas, mejorando la velocidad.	Sobrecarga de Comunicación para mantener la consistencia entre réplicas.
Organización	Se ajusta mejor a estructuras empresariales descentralizadas.	Mayor dificultad para mantener la Integridad Global de los datos.

III. La Esencia de la Transparencia

La transparencia es el principio por el cual el SMBDD oculta la complejidad de la red al usuario final, haciendo que la BDD parezca un único sistema centralizado.

- Transparencia de Distribución: Es la más importante. Oculta la realidad física del sistema y se compone de:
    - Transparencia de Localización: El usuario no necesita conocer la dirección física de un fragmento.
    - Transparencia de Fragmentación: El usuario trabaja con las tablas completas (relaciones globales) sin saber que han sido divididas.
- Transparencia de Réplica: Oculta la existencia de múltiples copias de los datos, obligando al sistema a gestionar la propagación de actualizaciones y la consistencia automáticamente.
- Transparencia de Concurrencia: Garantiza que las transacciones ejecutadas en distintos sitios y de manera simultánea se comporten como si se hubieran ejecutado una después de otra (serializabilidad).
- Transparencia de Fallos: El sistema es capaz de detectar fallos de nodos o enlaces de red y recuperarse sin pérdida de datos ni interrupción total del servicio.

IV. Clasificación y Arquitectura de Esquemas

La clasificación define el tipo de sistema que se está construyendo:

- Por la Homogeneidad: Homogéneas (todos los SGBD son idénticos) vs. Heterogéneas (diferentes SGBD, diferentes modelos de datos o ambos). Los sistemas heterogéneos requieren una capa de *middleware* más compleja.
- Por la Distribución de Datos (Asignación): No Replicadas (cada fragmento en un solo sitio) vs. Replicadas (el mismo fragmento en múltiples sitios), lo que puede ser Total o Selectiva.

La Arquitectura se define mediante el mapeo de esquemas: Un Esquema Global Conceptual (vista unificada de la BD) se transforma en Esquemas de Fragmentación, que a su vez se asignan mediante el Esquema de Asignación a los Esquemas Locales de cada nodo.

V. Diseño de BDD: Fragmentación Detallada

El diseño práctico se resuelve con la Fragmentación y la Asignación de Datos. La fragmentación es el proceso de dividir una relación global en subconjuntos. Esta división debe cumplir tres propiedades clave:

1. Completitud: Cada dato de la relación original debe estar contenido en algún fragmento.
2. Reconstrucción: La relación original debe poder ser reconstruida a partir de los fragmentos (usando operaciones como la Unión Natural $\bowtie$ o la Unión $\cup$).
3. Disjunción: Los fragmentos deben ser disjuntos (no solaparse), excepto por el atributo clave que se debe mantener en la fragmentación vertical para la reconstrucción.

Tipos de Fragmentación:

- Fragmentación Horizontal: Divide las filas (tuplas) de una relación usando un predicado de selección. Es ideal cuando los usuarios de un sitio solo necesitan acceso a un subconjunto específico de filas (ej. datos de su región geográfica).
- Fragmentación Vertical: Divide las columnas (atributos) de una relación. Es útil cuando diferentes sitios necesitan acceder a diferentes atributos de la misma fila. Requiere que la clave primaria se mantenga en todos los fragmentos.
- Fragmentación Mixta o Híbrida: Es la aplicación de una fragmentación vertical sobre los resultados de una fragmentación horizontal (o viceversa). El ejemplo práctico del PDF sobre la tabla CLIENTE es el caso típico:
    1. Horizontal: Se separan los clientes por su estado (CLI_EST = 'DF', CLI_EST = 'NL', etc.).
    2. Vertical: Dentro del fragmento 'DF', se realiza una nueva división para crear CLI_M1 (atributos para el departamento de Servicio) y CLI_M2 (atributos para el departamento de Colección), satisfaciendo así las necesidades de información de departamentos distintos que pueden estar localizados en el mismo sitio o en sub-sitios diferentes.

```
CREATE TABLE departamentos (
  id INT PRIMARY KEY,
  nombre VARCHAR(50)
);
```

```
CREATE TABLE empleados (
  id INT PRIMARY KEY,
  nombre VARCHAR(50),
  salario DECIMAL(10,2),
  departamento_id INT,
  FOREIGN KEY (departamento_id) REFERENCES departamentos(id)
);

CREATE TABLE proyectos (
  id INT PRIMARY KEY,
  nombre VARCHAR(50),
  departamento_id INT,
  FOREIGN KEY (departamento_id) REFERENCES departamentos(id)
);
```

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 --replSet shard1 --bind_ip localhost

mongod --shardsvr --port 27019 --dbpath ~/mongodb/shard2 --replSet 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 --replSet 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