Apuntes Tema 4 SGBD

Contenido

Bases de Datos Distribuidas y Fragmentación de la Información

Explorando las arquitecturas que permiten **escalabilidad y rendimiento** en sistemas de gestión de datos globales

- **¿Qué es una Base de Datos Distribuida?**

Definición

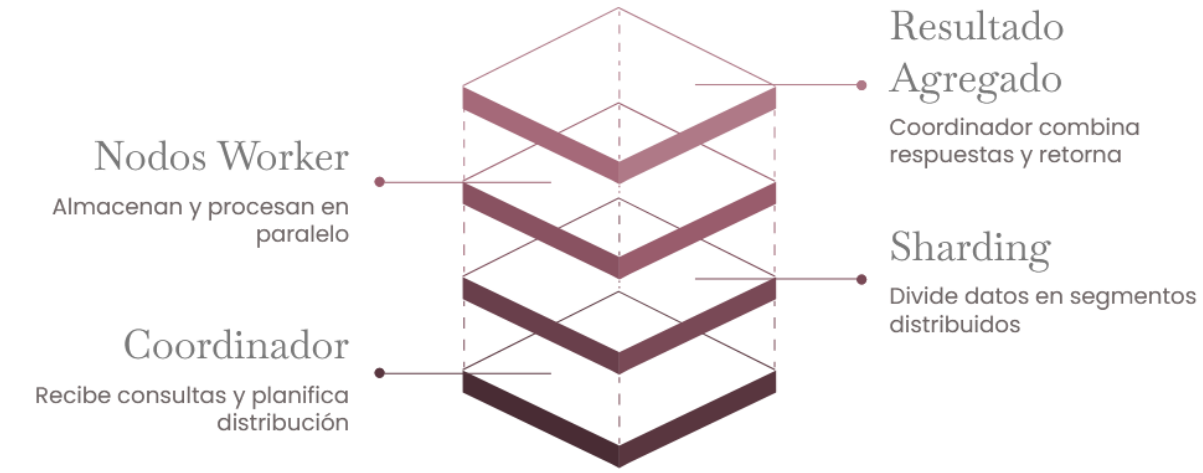
- Sistema que almacena información en múltiples nodos o servidores
- Frecuentemente ubicados en distintas localizaciones geográficas.

Rendimiento

- Esta arquitectura permite escalabilidad horizontal.
- Asegura alta disponibilidad del sistema.
- Tolerancia a fallos, si un nodo falla otros continúan operativos.
- Mejora significativamente el rendimiento mediante procesamiento paralelo de consultas.
- Reducción de latencia, se ubican datos cerca del usuario final.

PostgreSQL Distribuido: Escalabilidad Empresarial con Citus

Aunque PostgreSQL es una base de datos relacional potente, su arquitectura tradicional puede encontrar limitaciones en cargas de trabajo de gran escala. Extensiones como Citus Data transforman PostgreSQL en un sistema distribuido, permitiendo un manejo eficiente de datos masivos y altas concurrencias.



Con Citus, PostgreSQL se convierte en una solución distribuida capaz de ofrecer una escalabilidad horizontal masiva, manteniendo la familiaridad y el robusto ecosistema de PostgreSQL, lo que lo hace ideal para aplicaciones analíticas y transaccionales de alto rendimiento.

Google Spanner: Una Base de Datos Globalmente Consistente

Distribución Global

Escalabilidad horizontal que abarca múltiples regiones y continentes.

Consistencia Externa

Garantiza transacciones atómicas con un orden global de eventos, incluso a través de nodos distribuidos.

Alta Disponibilidad

Replicación automática de datos en múltiples zonas geográficas para una operación continua.

Escalabilidad Ilimitada

Crece dinámicamente con la demanda sin sacrificar rendimiento ni disponibilidad.

Interfaz SQL

Soporte para consultas SQL estándar, facilitando la interacción con datos complejos.

Introducción a Google Cloud Spanner: primeros pasos

https://www.youtube.com/watch?v=cEmEWtOokFo

- Características Claves:

BD relacional diseñada para escalar.

Disponibilidad líder (99%).

Fragmentación automática (aumenta el rendimiento).

Seguridad de nivel empresarial (encriptado de capa de datos, gestión de credenciales y registros para auditorias).

Cambios de esquema online sin periodos inactivos.

- Casos de usos:

Servicios financieros.

Gaming.

Retails. (Cadenas de suministros, orden de inventario, etc)

Industria de Salud (pacientes, facturación).

- Ejemplo en Google Cloud Platform:

Cloud Spanner: Qwik Start

Crear instancias y database

Insertar data y testearlo

Apache Cassandra

Gestionar enormes volúmenes de datos con alta disponibilidad.

Características Clave

- Base de datos NoSQL de código abierto.
- Arquitectura distribuida sin punto único de fallo (peer-to-peer).
- Diseñada para alta disponibilidad y tolerancia a fallos.
- Escalabilidad lineal que permite añadir más nodos según la demanda.
- Consistencia configurable (eventual a fuerte) para adaptarse a diferentes necesidades.

Casos de Uso Principales

- Historial de actividad de usuarios → inicios de sesión, clics, acciones.
- Datos de sensores (IoT) → temperatura, posición GPS, mediciones por tiempo.
- Datos de redes sociales → publicaciones recientes, mensajes, likes.
- Registros del sistema (logs) → errores, eventos del servidor.
- Datos de comercio electrónico → carritos activos, historial de compras, inventario.

Qué es Apache Cassandra - Tutorial en Español

https://www.youtube.com/watch?v=7bM5e2xa6Ic

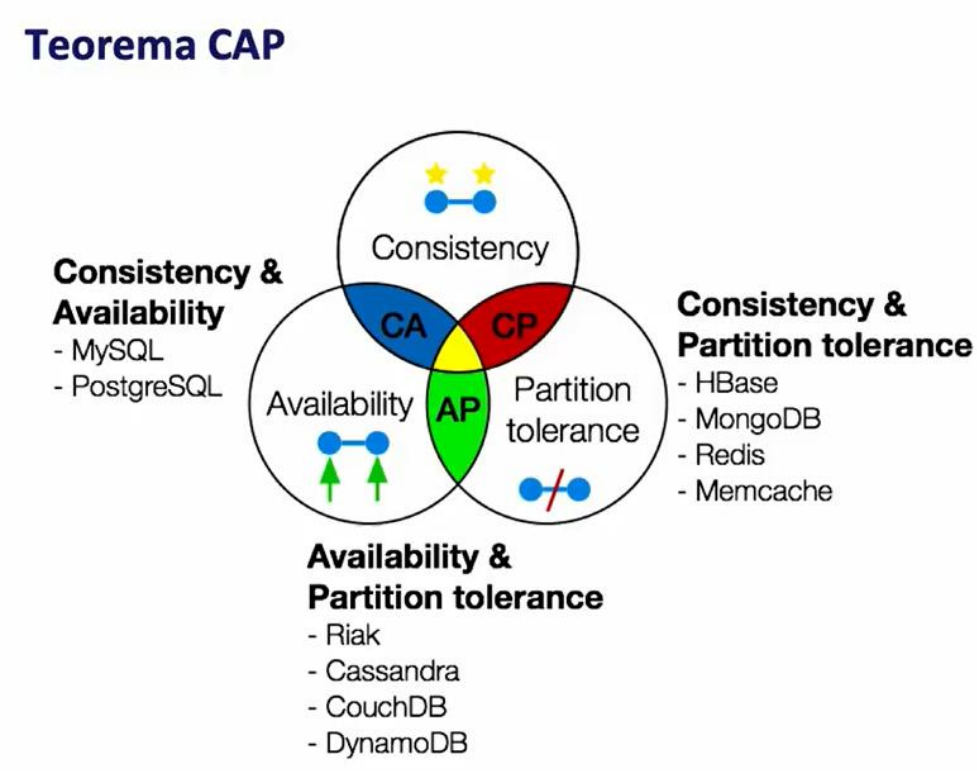
Es una base de datos NoSQL. Más flexibles a la hora de recoger datos no estructurados.

Cuatro tipos de bases de datos:

- Key-Value: tengo una clave y un valor o conjunto de valores. Trabajan en memoria y son muy rápidas.
- Column family: Almacena en columna y no en filas (Esta es la categoría de Cassandra).
- Graph: Nodos y aristas (las relaciones entre nodos la conforman las aristas): Se puede almacenar en los nodos y aristas.
- Document: Almacenan los datos en forma de documento (JSON). Más flexible.

Casandra: Lanzada den 2008, por Facebook. Se hizo OpenSource y pasada a la fundación Apache. Inspirada en Amazon Dunamo y Google BigTable.

Teorema CAP: Consistencia, Alta disponibilidad y Tolerancia a particiones.



Es distribuida. Escala linealmente (A más nodos, más operaciones, sin límites). No sigue un patrón maestro esclavo, es Peer 2 Peer. Si se cae un nodo, el servicio sigue funcionando. Escalabilidad horizontal (meter más equipos).

Es tolerante a fallos (gracias a replicación de datos en nodos).

Permite definir nivel de consistencia.

Usa lenguaje CQL.

Permite replicar en varios datacenters.

Comercializada por Datastax: cobran por soporte.

- Ventajas:

Alta disponibilidad.

Tolerancia a particiones y escalado.

Cantidad de recursos disponibles.

- Desventaja:

La conexión de nuevos nodos no es simple.

Debemos saber que “queries” se van a ejecutar previamente.

MongoDB

Características Principales

- **Orientado a Documentos:** Almacena datos en documentos BSON (similar a JSON).
- **Esquema Dinámico:** Permite cambiar la estructura de los documentos sin interrupciones
- **Alta Disponibilidad:** Replica automáticamente los datos a través de "Replica Sets" para garantizar la continuidad operativa.
- **Escalabilidad Horizontal:** Utiliza "sharding" para distribuir grandes volúmenes de datos entre múltiples servidores.
- **Rendimiento Óptimo:** Diseñado para operaciones de lectura y escritura de alta velocidad, crucial para aplicaciones dinámicas.

Casos de Uso Comunes

- **Aplicaciones de Gran Volumen:** Ideal para gestionar datos de Big Data y aplicaciones con millones de usuarios.
- **Sistemas de Gestión de Contenido:** Almacena metadatos y contenido dinámico para CMS y plataformas de e-commerce.
- **Aplicaciones Móviles:** Su flexibilidad y escalabilidad lo hacen perfecto para los datos de usuarios en aplicaciones móviles.
- **Análisis en Tiempo Real:** Permite procesar y consultar grandes conjuntos de datos rápidamente para dashboards y analíticas en vivo.
- **Internet de las Cosas (IoT):** Capaz de manejar la ingesta masiva de datos de sensores y dispositivos conectados.

¿Qué es MongoDB? Breve explicación animada

https://www.youtube.com/watch?v=DM4gD6Z5zFU

NoSQL. Diseñado para la web moderna. Orientado a documentos JSON pero almacenado como BSON.

Colecciones de documentos y sus relaciones.

Se pueden agregar arrays de valores (no hay que modificar tablas o añadir nuevas tablas).

Cuenta con drivers oficiales: java, php, Python, C++, etc.

Se recomienda usarlo en comercio electrónico, juegos, aplicaciones móviles, manejo de estadísticas, administración de contenido.

No se recomienda en sistema de transacciones, no soporte transaccional de manera nativa.

¿Qué es la Fragmentación de la Información?

- La fragmentación es una técnica fundamental para dividir una base de datos en fragmentos más pequeños y manejables, conocidos como "shards" o particiones.
- Cada fragmento puede residir en un servidor distinto, facilitando el **procesamiento paralelo** y distribuyendo la carga de trabajo de manera eficiente.

Piensa en un bufé con múltiples estaciones de comida: cada estación sirve un tipo de plato específico, evitando cuellos de botella y acelerando el servicio para todos los comensales.

Tipos de Fragmentación

- **Fragmentación Horizontal**

División por filas basada en criterios específicos. Los registros se separan según valores de atributos.

Ejemplo: Clientes europeos en un fragmento, clientes americanos en otro, optimizando consultas geográficas.

- **Fragmentación Vertical**

División por columnas. Las tablas se separan en subconjuntos de atributos relacionados funcionalmente.

Ejemplo: Tabla de clientes dividida en datos personales (nombre, edad) y datos de contacto (email, teléfono).

- **Fragmentación Mixta**

Combinación estratégica de fragmentación horizontal y vertical para optimizar tanto rendimiento como almacenamiento.

Ofrece flexibilidad máxima adaptándose a patrones de acceso complejos.

Fragmentación Horizontal en Sistemas Gestores

MongoDB

Utiliza fragmentación horizontal (sharding) para distribuir documentos según rangos de valores de una clave específica.

Ventajas

- Consultas dirigidas a fragmentos específicos
- Mejora significativa en tiempos de respuesta
- Escalabilidad transparente para aplicaciones

Desafío Principal

Balancear la carga para evitar **"hotspots"** o fragmentos saturados que degraden el rendimiento global.

Fragmentación Vertical en Sistemas Gestores

Oracle Database

Permite dividir tablas grandes en subtablas conteniendo columnas relacionadas, mejorando organización y acceso selectivo.

Mejora de Seguridad

Columnas sensibles pueden aislarse en fragmentos con controles de acceso específicos, reforzando la protección de datos.

Reducción de Transferencias

Consultas específicas solo acceden a los fragmentos necesarios, reduciendo el volumen de datos transferidos por la red.

Inconveniente: Puede **aumentar la complejidad en consultas** que requieren unir múltiples fragmentos (operaciones JOIN distribuidas).

Políticas de Fragmentación y su Impacto

La elección de la política de fragmentación depende críticamente del **patrón de consultas** y la estructura inherente de los datos.

Fragmentación Horizontal

Ideal para sistemas con consultas que filtran por **rangos de valores o regiones geográficas**. Maximiza paralelismo en búsquedas localizadas.

Fragmentación Vertical

Útil cuando diferentes aplicaciones o módulos **acceden consistentemente a subconjuntos específicos** de columnas. Reduce overhead de I/O.

Ejemplo: PostgreSQL

Ofrece particionamiento declarativo que soporta fragmentación horizontal mediante rangos, listas o hash, simplificando la gestión.

Retos y Consideraciones

Complejidad de Gestión

La administración y mantenimiento de múltiples fragmentos distribuidos → requiere herramientas sofisticadas y personal especializado.

Operaciones Distribuidas Costosas

Operaciones como JOIN entre fragmentos o transacciones que abarcan múltiples nodos pueden ser computacionalmente costosas y lentas.

Infraestructura Adicional

Se necesitan mecanismos robustos para balanceo dinámico de carga, replicación automática y recuperación rápida ante fallos de hardware o red.