

Actividad 1. Bases de datos en memoria

DANIELA VIGNAU LEÓN, Tecnológico de Monterrey, México

CRISTOPHER CEJUDO, Tecnológico de Monterrey, México

HÉCTOR REYES, Tecnológico de Monterrey, México

Este reporte busca profundizar en varios aspectos de las bases de datos en memoria, teniendo como foco el sistema de administración de bases de datos (DBMS) Kinetica. También se analiza el rendimiento de la base de datos en cuestión para las operaciones de inserción y consulta.

ACM Reference Format:

Daniela Vignau León, Cristopher Cejudo, and Héctor Reyes. 2021. Actividad 1. Bases de datos en memoria. *J. ACM* 1, 1 (January 2021), 4 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

1 INTRODUCCIÓN

Kinetica es una base de datos distribuida en memoria que permite ingerir, analizar y visualizar datos simultáneamente. Es un DBMS único en el sentido de que aprovecha el rendimiento de la unidad de procesamiento gráfico (GPU) para realizar operaciones con más rapidez que las bases de datos tradicionales.

Al tratarse de una base de datos orientada a columnas diseñada para el procesamiento analítico (OLAP), Kinetica está optimizada para manejar grandes volúmenes de datos de alta cardinalidad. Por tanto, no es adecuada como sistema de uso transaccional (OLTP). Kinetica organiza los datos de forma estructurada, similar a otras bases de datos relacionales, y los almacena en la memoria RAM o vRAM, en el caso de las GPUs.

2 DESARROLLO

2.1 Arquitectura del DBMS

Kinetica cuenta con una arquitectura distribuida diseñada así para el procesamiento de datos a escala. Un clúster de Kinetica, conformado por nodos idénticos, es capaz de ejecutarse en hardware básico, así como en aquellos equipados con GPU. Uno de esos nodos es seleccionado para ser el nodo de agregación principal. En la Fig. 1 se muestra un diagrama de la arquitectura de Kinetica.

2.2 Tipo de almacenamiento utilizado

La interfaz API nativa a Kinetica para el almacenamiento de datos es de tipo objeto; cada uno de ellos es una fila en la tabla.

Authors' addresses: Daniela Vignau León, Tecnológico de Monterrey, México; Cristopher Cejudo, Tecnológico de Monterrey, México; Héctor Reyes, Tecnológico de Monterrey, México.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2021 Association for Computing Machinery.

0004-5411/2021/1-ART \$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

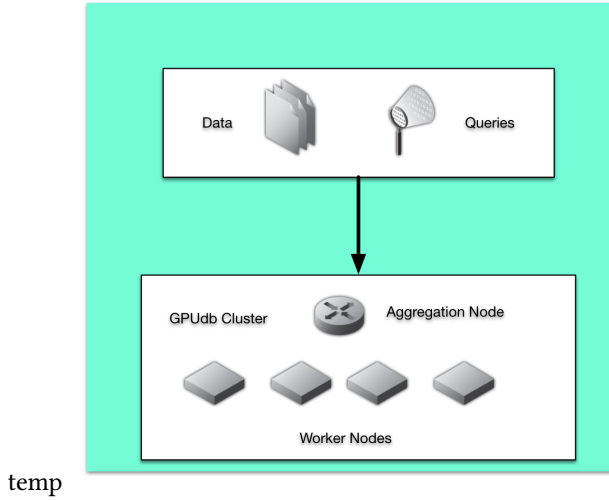


Fig. 1. Arquitectura del DBMS

2.3 Representación en memoria

Dado que el tipo de almacenamiento utilizado por Kinetica es por columnas, la representación en memoria es secuencial. Así, cada registro de una columna va después del otro, e inmediatamente después de terminar esa columna, comienza la siguiente.

2.4 Mecanismos de compresión

Es posible aplicar diferentes mecanismos de compresión a las columnas de las tablas de Kinetica. De manera predeterminada, las columnas son almacenadas sin compresión. En el caso de que se compriman, permanecen en ese estado hasta que son utilizadas. Cuando se recuperan los datos, se realiza una copia temporal de los datos descomprimidos, la cual es descartada una vez que ya no se utiliza.

Kinetica admite cuatro configuraciones de compresión diferentes que varían en velocidad:

- Ninguna (predeterminado)
- snappy: Altas velocidades de compresión y descompresión.
- lz4: Alta velocidad de compresión, mayor velocidad de descompresión.
- lz4hc: Menores velocidades de compresión y descompresión.

2.5 Particionamiento

Los datos de una tabla pueden ser sometidos a un particionamiento para optimizar el rendimiento y el manejo de datos. Los esquemas de particionamiento permitidos en Kinetica son los siguientes:

- Rango: Las *particiones por rango* de datos se definen a partir de un rango donde todos los registros con una *llave de partición* de ese rango son asignados a esa *partición*.
- Intervalo: Las *particiones por intervalo* se definen como intervalos numéricos o intervalos basados en el tiempo. Todos los registros cuya *llave de partición* está dentro del intervalo, o en el rango inferior, son asignados a esa *partición*.
- Lista: Las *particiones por lista* pueden ser manuales o automáticas, donde, en general, cuando los registros cuyos valores de la *llave de partición* estén dentro de la lista prefinida, son asignados a esa *partición*.

- Hash: Las *particiones por hash* se definen por un número exacto de particiones, donde cada uno de los registros es asignado a una *partición* basado en el hash de la *llave de partición*.
- Series: El *particionamiento por serie* segmenta los datos en una secuencia de *particiones* alocadas de manera dinámica.

2.6 Operaciones DML

En lo que respecta a la inserción de datos (INSERT), Kinetica admite dos tipos de formato:

- INSERT INTO...VALUES: Se utiliza para insertar registros con valores literales en una tabla.

```
INSERT INTO example.employee (
    id ,
    dept_id ,
    manager_id ,
    first_name ,
    last_name ,
    sal ,
    hire_date )
VALUES
    (1, 1, null , 'Anne' , 'Arbor' , 200000 , '2000-01-01')
```

- INSERT INTO...SELECT: Se utiliza para insertar registros de una tabla existente.

```
INSERT INTO example.employee (
    id ,
    dept_id ,
    manager_id ,
    first_name ,
    last_name ,
    sal )
SELECT id , dept_id , manager_id , first_name , last_name , sal
FROM example.employee
WHERE hire_date >= '2000-01-01'
```

De forma predeterminada, Kinetica descartará cualquier registro a insertar cuya llave primaria coincida con la de un registro existente.

Por otro lado, la operación de actualización (UPDATE) permite asignar un nuevo valor constante a una columna mediante el formato UPDATE...SET...WHERE.

```
UPDATE example.employee
SET
    sal = sal * 1.10 ,
    manager_id = 3
WHERE id = 5
```

Finalmente, la operación de eliminación (DELETE) permite eliminar registros de una tabla mediante el formato DELETE FROM...WHERE.

```
DELETE
FROM example.employee
WHERE id = 6
```

2.7 Buffer diferencial y el proceso de mezcla

2.8 Reconstrucción de tuplas

2.9 Tipos de *join*

Con Kinetica, es posible conectar datos relacionados entre dos o más tablas a través de *join views*. Los tipos de *join* permitidos son:

- **INNER:** Regresa los registros que coinciden entre dos tablas.
- **LEFT:** Regresa los registros que coinciden entre dos tablas, así como también los registros de la *tabla izquierda* cuyos registros no coinciden con la tabla derecha.
- **RIGHT:** Regresa los registros que coinciden entre dos tablas, así como también los registros de la *tabla derecha* cuyos registros no coinciden con la tabla izquierda.
- **FULL OUTER:** Regresa los registros que coinciden entre dos tablas, así como aquellos que no tienen ninguna relación con la otra tabla.
- **Cross:** Regresa todos los registros de una tabla junto con su correspondiente registro de la otra tabla.

De igual manera, hay dos tipos de ejecución para los *joins*:

- **Native joins:** Realiza un aislamiento de la operación *join* y crea un *join view* nativo de Kinetica.
- **SQL Joins:** Se realiza de manera automática cuando una base de datos recibe una query de SQL con la cláusula JOIN.

Dependiendo del tipo de *join* utilizado, la memoria tiene un cierto impacto negativo. Kinetica constatemente busca la manera de crear el *join* más eficiente posible.

2.10 Logging

Kinetica cuenta con un registro el cual almacena información como: las interacciones de la base de datos, *startup/shutdown*, información de errores y más. Este registro puede ser configurado para que se almacene otra información que no está por default.

2.11 Respallos y *Recovery*

Existe una herramienta para la administración, configuración e instalación de Kinetica llamada *KAgent*, la cual también permite simplificar el proceso de respaldo y *recovery* de una base de datos en Kinetica.

Todos los datos dentro de la base de datos pueden ser respaldados de manera automática (es decir, siguiendo una agenda preestablecida), o bien cuando el usuario así lo requiera. Los respaldos son almacenados de manera local para cada uno de los nodos dentro del cluster.

2.12 Manejo de transacciones

2.13 Cold/hot store

2.14 Manejo de datos históricos