

Actividad 1. Bases de datos en memoria

DANIELA VIGNAU, 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, Cristopher Cejudo, and Héctor Reyes. 2021. Actividad 1. Bases de datos en memoria. *J. ACM* 1, 1 (January 2021), 7 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCCIÓN

Kinetica es una base de datos 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. *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. [1]

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]

2.2 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. [2]

Authors' addresses: Daniela Vignau, a01021698@itesm.mx, Tecnológico de Monterrey, Ciudad de México, México; Cristopher Cejudo, a01025468@itesm.mx, Tecnológico de Monterrey, Ciudad de México, México; Héctor Reyes, a01339607@itesm.mx, Tecnológico de Monterrey, Ciudad de México, 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/nnnnnnn.nnnnnnn>

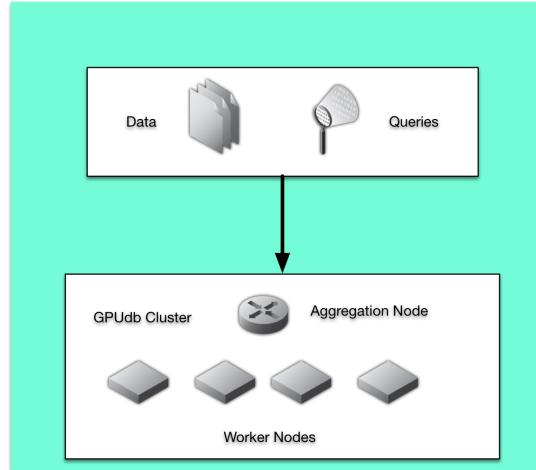


Fig. 1. Arquitectura del DBMS [2]

2.3 Representación en memoria

Dado que el tipo de almacenamiento utilizado por Kinetica es orientado a 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 caso de que se deseé comprimir las, 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 según su 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

Kinetica, además, permite la codificación por diccionario, la cual puede ser aplicada a columnas individuales de los siguientes tipos de datos:

- int
- long
- date
- char1 - char256

De esta manera, se almacenan en memoria los valores únicos por cada columna para después ser asociados a su correspondiente registro. Esto elimina el almacenamiento de valores duplicados por columna, reduciendo, así, el uso de la memoria y el espacio en disco requeridos para almacenar los datos. [3, 4]

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 predefinida, 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. [5]

2.6 Operaciones DML

2.6.1 *Operación de inserción*. En lo que respecta a la operación 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.

2.6.2 Operación de actualización. La operación UPDATE, por otro lado, 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
```

2.6.3 Operación de eliminación. Finalmente, la operación DELETE permite eliminar registros de una tabla mediante el formato DELETE FROM...WHERE. [6]

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

2.7 Buffer diferencial y proceso de mezcla

No se encontró información sobre la implementación del buffer diferencial y el proceso de mezcla en Kinetica.

2.8 Reconstrucción de tuplas

No se encontró información sobre el proceso de reconstrucción de tuplas en Kinetica.

2.9 Tipos de join

Con Kinetica, es posible conectar datos relacionados entre dos o más tablas a través de *join views*. A continuación se enlistan los tipos de *join* permitidos:

- *Inner*: Regresa los registros que coinciden entre dos tablas.
- *Left*: Regresa los registros que coinciden entre dos tablas, así como 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 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 los que no tienen relación con la otra tabla.
- *Cross*: Regresa todos los registros de una tabla junto con su registro correspondiente de la otra tabla.

Asimismo, hay dos tipos de ejecución para los *joins*:

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

Dependiendo del tipo de *join* usado, la memoria se verá afectada positivamente o negativamente. Kinetica busca constantemente la forma de crear el *join* más eficiente posible. [6]

2.10 Logging

Kinetica mantiene un registro que almacena información como interacciones de la base de datos, *startup/shutdown*, errores, etc. Este registro se puede configurar para almacenar información que no está predeterminada.

2.11 Respaldos y *recovery*

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

Se puede realizar un respaldo de todos los datos de la base de datos de forma automática —es decir, siguiendo un programa preestablecido— o cuando el usuario así lo requiera. Las copias de seguridad se almacenan localmente para cada uno de los nodos dentro del clúster.

2.12 Manejo de transacciones

Debido a la potencia que le brinda la compatibilidad con GPU, Kinetica rara vez se usa para el manejo de transacciones. Como sistema OLAP, es mejor aprovechado para consultas de análisis de datos junto con otro DBMS de tipo OLTP. [1]

2.13 Cold/Hot Storage

Kinetica segmenta el almacenamiento de datos, ya sea temporal o permanente, por niveles o *tiers*. Hay cinco *tiers* soportados por Kinetica:

- *VRAM*: Compuesto por memoria RAM de GPU en cada nodo del clúster.
- *RAM*: Compuesto por memoria del sistema en cada nodo del clúster.
- *Caché de disco*: Sirve como espacio temporal para el intercambio de datos que no caben en la memoria RAM o VRAM.
- *Persist*: Contiene registros que se conservan cada vez que se reinicia la base de datos.
- *Cold Storage*: Se utiliza para almacenar datos a los que se accede con poca frecuencia.

En la Fig. 2 se muestra un diagrama de los diferentes niveles de almacenamiento.

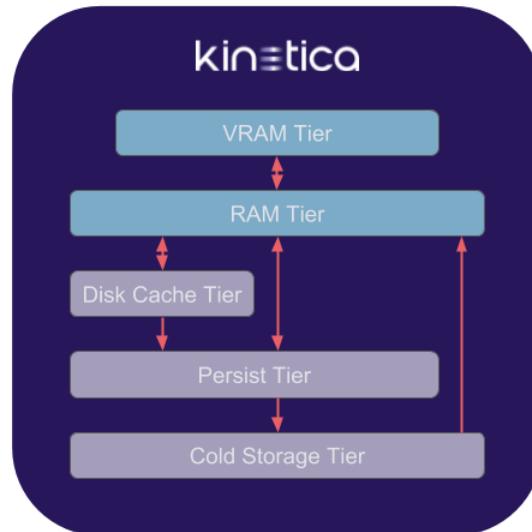


Fig. 2. Niveles de almacenamiento

El rendimiento de cada *tier* es inversamente proporcional a su capacidad de almacenamiento.

2.14 Manejo de datos históricos

Kinetica cuenta con un *streaming data warehouse*, el cual, a diferencia de los *data warehouses* tradicionales, puede analizar los datos almacenados en tiempo real. De este modo, se obtiene un análisis que puede ser utilizado de manera innmediata. [7]

3 RESULTADOS

Aquí van los resultados chidos

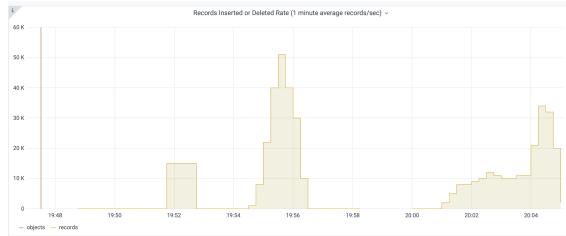


Fig. 3. Registros insertados por minuto

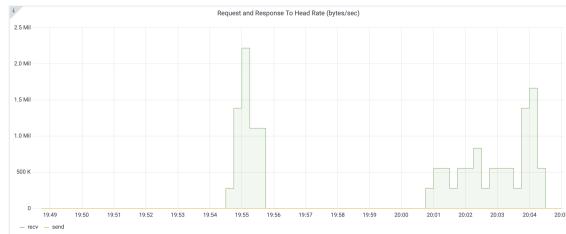


Fig. 4. Velocidad de peticiones y respuesta bytes/sec

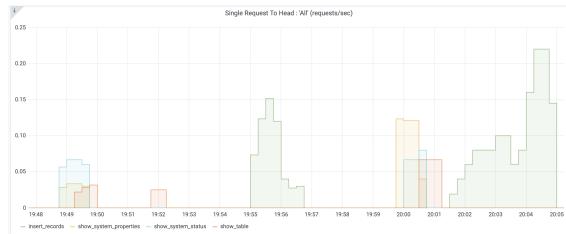


Fig. 5. Petición de inserción por segundo

4 CONCLUSIONES

REFERENCIAS

- [1] «Frequently Asked Questions». <https://www.kinetica.com/product/faq/>
- [2] «Overview». <https://www.kinetica.com/docs/overview/arch.html>
- [3] «Compression». <https://www.kinetica.com/docs/concepts/compression.html>
- [4] «Dictionary Encoding». <https://www.kinetica.com/docs/concepts/compression.html>
- [5] «Tables». <https://www.kinetica.com/docs/concepts/tables.html>
- [6] «SQL Support». <https://www.kinetica.com/docs/concepts/sql.html>
- [7] «What Is a Streaming Data Warehouse?». <https://www.kinetica.com/products/why-streaming-data-warehouse/>

Actividad 1. Bases de datos en memoria

The screenshot shows a SQL tool interface with the following details:

- SQL Statements:** `select * from kl_world.world_population`
- Schema:** <DEFAULT>
- Query Result:**

ID	Name	Gender	Male	CN
1351975	Wolter	Garcia	Male	CN
1351976	Wolter	McKinney	Male	CN
1351978	Della	Nguyen	Male	CN
1351979	Lizette	Simons	Female	CN
1351980	Wolter	Alvarenga	Male	CN
1351983	Eleanor	Norton	Male	CN
1351984	Wolter	Alvarenga	Male	CN
1351987	Carl	Weaver	Female	CN
1351988	Alexander	Watkins	Male	CN
1351989	Wolter	Alvarenga	Male	CN
1351991	Wolter	Herrera	Female	CN
1351992	Wolter	Mitchell	Male	CN
1351993	Brandon	Weber	Female	CN
- Statistics:**
 - Rows read = 18000
 - Query Execution Time: 0.159 s
 - Data Transfer Time: 0.02 s
 - System Results Limit: 10000
- Message:** EXECUTION FINISHED

Fig. 6. Consulta sobre tabla completa

The screenshot shows a SQL tool interface with the following details:

- SQL Statements:** `select count(id) as 'total_by_gender', sum(id) as 'sum_of_ids', gender from kl_world.world_population group by gender`
- Schema:** <DEFAULT>
- Query Result:**

gender	total_by_gender	sum_of_ids
Female	3248581	1855797555889
Male	3251499	18563205694311
- Statistics:**
 - Rows read = 18000
 - Query Execution Time: 1.968 s
 - Data Transfer Time: 0.02 s
 - System Results Limit: 10000
- Message:** EXECUTION FINISHED

Fig. 7. Consulta con funciones de agregación y agrupamiento

The screenshot shows a SQL tool interface with the following details:

- SQL Statements:** `select * from kl_world.world_population where name = 'Dante'`
- Schema:** <DEFAULT>
- Query Result:**

ID	Name	Gender	Male	CN
5248428	Dante	Loeb	Male	CN
5248431	Dante	McKinney	Male	CN
5248432	Dante	Nguyen	Male	CN
5248433	Dante	Simons	Female	CN
5248434	Dante	Francis	Female	CN
5248435	Dante	Alvarenga	Female	CN
5248436	Dante	Lucas	Female	CN
5248437	Dante	Extrada	Male	CN
5248438	Dante	Vargas	Male	CN
5248439	Dante	Qurim	Male	CN
5248440	Dante	Alvarenga	Female	CN
5248441	Dante	Rush	Male	CN
5248442	Dante	Carson	Female	CN
5248443	Dante	Murphy	Male	CN
- Statistics:**
 - Rows read = 18000
 - Query Execution Time: 14.218 s
 - Data Transfer Time: 0.02 s
 - System Results Limit: 10000
- Message:** EXECUTION FINISHED

Fig. 8. Consulta con condición de filtrado