Bases de Datos no Estructuradas

UNAM — Semestre 2022-2

Práctica 2@

Integrantes:
Andrés Urbano Guillermo Gerardo
Avitúa Varela Fernando
Santa Rita Vizuet Fernando

§ Bases de Datos NoSQL Cassandra §

Introducción

Apache Cassandra se trata de un software NoSQL distribuido y basado en un modelo de almacenamiento de *clave-valor*, de código abierto que está escrito en Java. Permite grandes volúmenes de datos en forma distribuida. Por ejemplo, lo usa Twitter para su plataforma. Su objetivo principal es la escalabilidad lineal y la disponibilidad. La arquitectura distribudores de Cassandra está basada en una serie de nodos iguales que se comunican con un protocolo *P2P* con lo que la redundancia es máxima.

El objetivo de este trabajo consiste en implementar una plataforma para el manejo de bases de datos de las opiniones de clientes de una librería sobre los libros que ha comprado. El cliente tiene la posibilidad de otorgar una categoría al libro y la calificación del mismo.

Los usuarios que comprenden la lista de clientes de la librería ficticia son estudiantes de la tercera generación de la licenciatura en Ciencia de Datos, con sede en Ciudad Universitaria. A su vez, debe existir registro de los atributos de cada uno de los clientes (nombre, país, membresía, etc).

Los administradores de la base de datos deben poder realizar lo siguiente:

1. Obtener la categoría preferida de un cliente dado.
2. Obtener los clientes que más disfrutaron un libro dado.
3. Acceder a los mejores libros de una categoría dada.

Diseño de la base de datos

El diseño de la base de datos debe tener como objetivo resolver las consultas requeridas por el usuario final con ciertas restricciones. Cassandra elige explícitamente no implementar operaciones que requieran una coordinación a través de los nodos de un cluster pues usualmente son lentas. Cassandra no permite:

- Transacciones a través de particiones
- Joins en esquemas distribuidos
- Llaves foráneas o integridad referencial

En lugar de implementar un esquema relacional con tablas flexibles y normalizadas, lo que se hace es diseñar una tabla para cada una de las consultas solicitadas. Una consecuencia de implementar este esquema es que se puede escribir la misma información en diferentes tablas para satisfacer diferentes consultas, lo cual es ideal para la arquitectura de Cassandra, implicando que sólo se requiere consultar una tabla para una petición en un sistema distribuido.

Los datos que usamos para la base de datos fueron llenados por los alumnos de la carrera, se tienen dos tablas, la primera corresponde a los registros de los libros y tiene los siguientes campos:

- **Nombre Usuario**: Llave única que identifica al usuario que lee los libros
- **Categoría**: Categoría del libro.
- **Libro**: Nombre del libro. Aunque puede haber más de un libro con el mismo nombre, suponemos que esto funciona como un identificador (para usar un identificador real deberíamos usar el ISBN)
- **Calificación**: El número que otorga el usuario de acuerdo a qué tanto le gustó el libro. Este número va de 0 a 5.

La segunda tabla contiene la información de los usuarios, formada de los siguientes campos:

- **Nombre Usuario**: Llave única que identifica al usuario que lee los libros
- **País**: El país de origen.
- **Membresía**: El tipo de membresía que denota que tanto puede comprar libros, qué tanto lee, cuántos libros compra, etc.

Consulta 1 Obtener la categoría favorita de un cliente dado

La primera de las consultas requiere la categoría favorita de un cliente dado, por tanto, requerimos poder efectuar consultas que contemplan una agrupación a nivel de usuario. Establecemos que la *partition key* será el campo *usuario* y como *cluster key* compuesta la *categoría* y *libro*. Por un lado, la *partition key* permite la agrupación por usuario en consultas, mientras que en la *cluster key*, *categoría* permite la agrupación y *libro* provee una identificación unívoca a cada uno de los registros. El diseño de esta tabla, que llamaremos *categoria_por_usuario*, está dado por:

categoria_por_usuario
usuario ***K***
categoria *C*
libro *C*
calificacion

Consulta 2 Obtener los usuarios que más prefieren un libro dado

Dado que necesitamos la información de todos los clientes *dado* un libro la *partition key* natural es el libro. Dentro de cada partición sólo necesitamos la calificación y el nombre del usuario.

Para ordenar por calificación la tenemos que hacer parte de las llaves del cluster. Entonces el diseño de esta tabla que llamaremos **clientes_por_libro** está dado por:

clientes_por_libro
libro ***K***
calificacion *C*
usuario *C*

Consulta 3 Obtener los mejores libros de una categoría dada.

Para conseguir los mejores libros dado una categoría, primero pensamos en el diseño de nuestra tabla, para lograr el mejor desempeño de nuestra base de datos haremos la partion key a la columna de categoría, de este modo cada categoría estará en un nodo diferente en el anillo de cassandra, después tomaremos como cluster column el libro y usuario para su ordenamiento en el disco, quedando el esquema de nuestra tabla de esta forma:

libros_por_categoria
categoria ***K***
usuario *C*
libro *C*
calificacion

Implementación

Consulta 1

Esta consulta requiere la siguiente tabla de acuerdo al diseño previo.

```
CREATE TABLE categories_by_user (
    usuario text,
    libro text,
    categoria text,
    calificacion float,
    PRIMARY KEY((usuario), categoria, libro));
```

Para obtener el resultado deseado (categoría favorita de un cliente), implementaremos una consulta mediante CQL y procesaremos este resultado a través de Python. La consulta vía CQL se realiza de la siguiente forma:

```
SELECT categoria, AVG(calificacion), COUNT(libro)
FROM categories_by_user
WHERE usuario = '{usuario}'
GROUP BY categoria
```

De acuerdo a esto, la consulta devuelve la categoría, promedio de calificación y la cuenta de los libros en cada categoría (pues agrupamos respecto a este atributo). Debemos establecer los criterios para la elección de la mejor categoría para un usuario dado el resultado de la consulta.

Criterios de asignación de la categoría favorita de un usuario

1. Decimos que una categoría se prefiere sobre otra si el promedio de calificación de los libros de dicha categoría es mayor que las demás.
2. En caso de que haya promedios iguales entre dos categorías, entonces establecemos como criterio de desempate elegir la categoría con más libros calificados.
3. Si existe un empate después de los dos filtros anteriores, entonces es no existe criterio para escoger una categoría u otra. Por tanto, permanece la categoría líder hasta el momento.

La implementación en conjunto de estas ideas están dadas por la siguiente función en Python:

```
def obtener_categoria_preferida(usuario) -> str:
    # consulta en CQL
    consulta = session.execute(f"""SELECT categoria, AVG(calificacion), COUNT(libro)
                                   FROM categories_by_user
                                   WHERE usuario = '{usuario}'
                                   GROUP BY categoria
                                   ALLOW FILTERING""")

    # variable que nos indica la categoria con mayor promedio hasta el momento
    avg_max = 0

    # categoria con mejor promedio
    cat_max = 'Empty'

    # cantidad de libros de la categoría
    count_max = 0

    # filtramos a la categoría preferida bajo los siguientes criterios
```

```
    for i in consulta:

        # criterio de asignación
        if avg_max < i[1]:
            cat_max, avg_max, count_max = i[0], i[1], i[2]

        # criterio de desempate
        elif avg_max == i[1]:
            if count_max < i[2]:
                cat_max, avg_max, count_max = i[0], i[1], i[2]

    return f'La categoría preferida de {usuario} es {cat_max}'
```

Una prueba sobre el conjunto de datos es la siguiente.

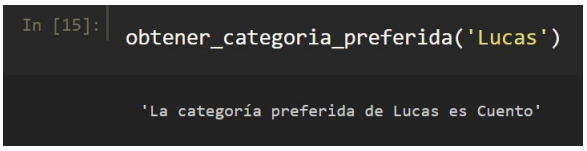

Figura 1: Ejemplo de la consulta 1 sobre la base de datos, donde se busca conocer la categoría favorita del usuario *Lucas*.

Consulta 2

Siguiendo en el *KEYSPACE* de *cheetah_ books*, creamos la tabla usando la siguiente instrucción en CQL

```
CREATE TABLE clientes_por_libro (
    libro text,
    usuario text,
    calificacion int,
    PRIMARY KEY ((Libro), calificacion, usuario)
);
```

Después, dado un libro, en este caso tomamos siddhartha, realizamos la siguiente consulta limitando a los 5 clientes con mayor calificación

```
SELECT usuario,calificacion
FROM clientes_por_libro
WHERE libro='siddhartha'
ORDER BY calificacion DESC LIMIT 5
```

dado que tenemos pocas entradas, sólo obtenemos dos clientes que se muestran a continuación:

Consulta 3

Para la realización de nuestra consulta crearemos primero nuestra tabla con el esquema descrito anteriormente:

```
CREATE TABLE libros_por_categoria (
categoria TEXT,
usuario TEXT,
libro TEXT,
calificacion INT,
PRIMARY KEY ((categoria), usuario, libro)
```

Esto creará nuestra tabla, el siguiente paso será proceder al llenado de los valores con los datos que tenemos de los clientes. Para obtener los mejores libros dada una categoría será posible utilizando esta consulta:

```
SELECT libro, AVG(calificacion) as promedio FROM libros_por_categoria
WHERE categoria = 'cuento'
GROUP BY libro;
```

libro	promedio
el tapiz amarillo	4
erecciones, eyaculaciones, exhibiciones	4
exhalation	4
la metamorfosis	4
the fall of the house of usher	8
the masque of the red death	10
the tell-tale heart	10

Ejemplo para la categoría *Cuento*.

Esta consulta agrupara por libro para posteriormente sacar su promedio para determinar los mejores libros. Veremos en la imagen que falta ordenar por promedio para obtener los mejores libro, esto es, con el promedio más alto, para ello utilizaremos python para ese procedimiento, de esta manera ordenaremos los registro y podremos pedirle el usuario cuantos libros mostrar.

```
def obtener_mejores_libros_categoria(categoria:str, top:int=5) -> list:
    """
    Regresa los mejores libros (mayor promedio de calificación) de una categoría dada

    Parametros
    ----------
    categoria : str
        La categoría en la que buscaremos los mejores libros

    top: int
        El número de mejores libros a regresar. Regresa 5 por defecto

    Regresa
    -------
    salida : list(str)
```

```
        Una lista de los mejores top libros por la categoría
    """
    query = f"""
    SELECT libro, AVG(calificacion) as promedio FROM libros_por_categoria
    WHERE categoria = '{categoria}'
    GROUP BY libro;
    """
    response = session.execute(query)
    mejores_libros = sorted(response.current_rows,
                    key=lambda record: record.promedio, reverse=True)[:top]
    return mejores_libros
```

Ordenamos los valores devueltos de la consulta por promedio y devolvemos una la lista de los mejores libro, a continuación mostramos el resultado con la categoria cuento.

```
top = 4
mejores_libros = obtener_mejores_libros_categoria('cuento', top)

print(f"{' Mejores libros ':{'*'}{'^'}{30}}")
for i, libro in enumerate(mejores_libros):
    print(f'{i+1}. - {libro.libro} ({libro.promedio})')
```

```
******* Mejores libros *******
1. - the masque of the red death (10)
2. - the tell-tale heart (10)
3. - the fall of the house of usher (8)
4. - el tapiz amarillo (4)
```

Conclusiones

Vemos que Cassandra se enfoca en el rendimiento, disponibilidad y escalabilidad, por lo que el diseño es fundamental para aprovechar el mejor uso correcto de la base de datos. Para poder realizar cada consulta, primero tuvimos que pensar en cómo sería la consulta para posteriormente definir como estarían conformadas la *partion key* y *cluster columns* para el esquema, ya que con base en eso, los datos estarán particionados en diferentes nodos y acomodados en el disco de distinta forma. Gracias a estas características presentes en Cassandra, podemos obtener información de manera rápida y poder escalar linealmente aumentando los nodos. Otra característica que nos sorprendió fue la manera en que Cassandra replica la información para que de esta forma cumpla la propiedad de disponibilidad, ya que si un nodo deja de funcionar, encontraremos otro nodo que tiene la misma información por se la replica.

Referencias

[1] Sadalage, Pramod J Fowler: *NoSQL Distilled, Addison Wesley Professional.*

[2] Harrison, Guy: *Next Generation Databases NoSQLand Big Data, Apress, 2015.*

[3] Cassandra, Apache: *Data Definition.* `https://cassandra.apache.org/doc/latest/cassandra/cql/ddl.html`. Accedido en Abril 18 del 2022.