Apuntes 3: Martes 10 de octubre

Diego Granados Retana 2022158363

Bases de datos II

10 de octubre, 2023

Apuntes de la clase:

¿Qué es un índice? Cuando estamos desarrollando una base de datos, tenemos el disco y adentro guardamos el raw data. El raw data es los datos crudos, sin procesar. El raw data pasar por algún tipo de transformación. Una transformación es una normalización de datos, una limpieza. Una limpieza es agarrar los datos y quitar las cosas que no aportan al problema que estamos desarrollando. En la parte de WikiText, la limpieza es eliminar la información de los marcadores de lenguaje. La limpieza sirve para reducir la tamaño en la base de datos.

Una normalización es pasar todos los datos al mismo formato. A nivel de una base de datos relacional, puede ser que el dato se considere diferente aún si semánticamente es igual. Se va a convertir en un formato consistente.

Aumentación de datos: Sacar nuevos datos a partir de los que ya se tienen o agregar nuevos datos. Para hacer una búsqueda en una base de datos SQL, puede ser que en nuestro contexto un dato se relacione con dos palabras que en el idioma español no son sinónimos, pero en este caso sí pueden referirse al mismo objeto. Por ejemplo, un yogurt bueno o saludable. En esta aumentación se añadió información para tener más formas de referenciar al mismo objeto.

Transformaciones de datos: Aumentar, limpiar y normalizar son transformaciones de datos.

Una vez que tenemos el texto raw, en una base de datos SQL, vamos a tener la implementación con una tabla. En una tabla vamos a tener nombres de columnas y vamos a tener datos. En una base de datos NoSQL de tipo documental, vamos a tener una representación de documento. En una base de datos SQL, tomamos la información raw, la transformamos y la metemos dentro de 1 o más tablas (muchas por lo general).

En una base de datos SQL, no es obligatorio completamente tener normalización. Usualmente, de un lado ponemos el rendimiento y de otro ponemos la duplicación de datos. Agarramos el raw data, lo transformamos y lo metemos en una o más tablas. En una búsqueda lineal o secuencial, la única forma de hacer la búsqueda es si nos ponemos desde el primer registro y nos empezamos a buscar. Este tipo de búsqueda se conoce como un scan.

Una base de datos documental que no use un full-text index hace igual. Incluso es más difícil porque el formato de los documentos es un JSON, entonces hay que procesarlo y hacer la comparación. En este caso la base de datos SQL es más rápida que la NoSQL si no hay índices.

Cuando tenemos una tabla, esta utiliza el concepto de registro. Este registro de todas las columnas puede medir por ejemplo 1 Mb. El archivo de datos se va a organizar de 1 Mb a 1 Mb. Cuando hago una búsqueda, le digo al SO que traiga esa información. Inmediatamente me trae ese registro. Cuando suben ese registro a memoria, la representación de esos datos es muy simple. Una fracción se usa para guardar cierto dato. Puedo

usar desplazamientos sobre memoria para poder recuperar esos datos. Le sumamos un 512k a una posición y llegamos a los siguientes campos. En C, los structs funcionan igual.

En una base de datos documental, lo que tiene es un JSON. Cuando la información binaria está guardada en disco, convierte la representación binaria a textual, luego hace la manipulación de los campos, pero no tengo la seguridad que los desplazamientos son iguales, por lo que necesito más lógica para hacer el análisis del documento y extraer la información. De esta forma, una búsqueda lineal es más rápida en SQL que en NoSQL pero el manejo de memoria y almacenamiento.

¿En qué ayudan los índices? Me permiten que las búsquedas sean más rápidas. Básicamente, vamos a meternos en el mundo de lo que es search. Dejamos por un lado el scan.

Cuando tenemos los datos rows, organizamos los datos en registros. Como vamos a organizar los datos en registros, la única forma de encontrar uno es hacer una búsqueda y preguntar si lo que estoy buscando es el registro. Esto es un scan. Hacer un scan no es necesariamente malo. Los datos normalmente podemos escoger un field, una columna para hacer una organización de datos para la búsqueda de datos. Nosotros buscamos una estructura de datos que sea balanceada. Si agarramos esto y lo estiramos y lo vemos como una lista, la tabla se ve como una lista simple. En el peor de los casos, la complejidad sería O(n). Si convierto la lista a una estructura de árbol balanceada, la complejidad del algoritmo se convierte en la profundidad del árbol. Con un índice, intentamos movernos de una lista a una estructura balanceada. Existen ciertas estructuras que me van a permitir hacer un índice sin necesidad de tener una estructura balanceada.

Los índices son separados de la tabla de datos. No todos los índices son buenos para todo tipo de datos. Para guardar texto, es mejor tener una estructura B+ o R+, pero si es con números, podemos usar un B balanceado. Si conocemos la distribución de datos, podemos usar un hash.

Los índices suelen ser muy grandes para almacenarlos en memoria y en disco. En la definición del archivo del índice, tenemos un header del file. El header contiene metadatos de los datos que estoy almacenando. El metadato más importante en la representación de un árbol es el Head, o la cabeza de la estructura de datos.

Si el archivo de datos está vacío, el Head apunta a -1, que indica que está vacío.

Si el motor de base de datos ve que en el Head hay -1, debemos tener otro metadato que me diga a dónde está el inicio del archivo. Luego de insertar en la base de datos, creo un registro en el header, tengo el HijoIzquierdo, el HijoDerecho, el Data y un puntero. El Head se actualiza a la posición 1 y el start a la nueva vacía. Hay un metadato como un “start” que apunta a la posición donde se inserta un nuevo elemento. Ejemplo. : 89: -1 | -1 | 89 | 0 (Desplazamiento de datos sobre el archivo de datos). Con el desplazamiento, hace un seek. Este registro empieza en la posición 0 del disco.

Seek es la función de archivos del SO que desplaza la cabeza lectora hacia donde está el registro.

Si se inserta un 120, creo un nuevo registro en los datos. En el header, decido si el dato debe ser hijo derecho o izquierdo. Creo el nuevo registro del dato insertado en el header y actualizamos el hijo derecho del registro padre. En el momento que inserto nuevos registros, es posible que tengo que rebalancear el árbol. Para hacer esto, cambio el Head a la posición raíz nuevo y les pongo hijos nuevos a los nodos. Reacomodo el árbol dentro el disco. Si yo sigo echando y echando información, tengo que actualizar el índice constantemente.

Si alguien quiere recuperar un dato, se pregunta por el índice para encontrarlo. Dentro de un disco puedo tener una representación de un árbol. La gracia de un índice como estos es que sea pequeño. Esto lo hago para poder subirlo completamente a memoria, entonces no hago paginación y context switches. La única

entrada de datos que tengo que hacer es a la tabla del disco. Esto se conoce como un índice non-clustered. Lo que me está diciendo es que crea un archivo afuera que va a apuntar hacia el archivo de datos.

En un clustered index, voy a tener un archivo que va a tener un header. El registro que vamos a manejar en la base de datos, va a tener una columna de identificación y una columna de data. La base de datos le va a sumar otras cosas. Le va a sumar el hijo izquierdo y el hijo derecho. Esto quiere decir que la información de organización para armar la estructura de datos dentro del disco está metida en el mismo registro de datos que vamos a almacenar. El árbol está montado o incrustado en el mismo disco. La representación del árbol está dentro del registro de datos. No vamos a repetir el campo de la columna de identificación.

El archivo empieza igual. Tiene un header. El root está vacío y el head o start inicia en 1. 89 / data / -1 / -1

En este otro, tengo todo el registro dentro del nodo. Es difícil mantener todo el índice clustered dentro de la memoria principal. Con cada inserción, se actualizan los punteros a los hijos. Estoy implementando la estructura de datos que me implementa el índice dentro de la misma estructura de datos. Solo podemos tener 1 solo clustered index dentro del archivo de datos. Si el data set es muy grande, es prácticamente imposible subir el clustered a memoria.

En un archivo clustered, si tenemos un archivo de datos que pesa un terabyte. Si tenemos un índice non-clustered sobre ese archivo, dependiendo del tamaño es más rápido que un clustered. En un clustered tengo que bajar al disco la cantidad de veces de profundidad del índice. Con un nonclustered index pequeño, subo el índice a memoria una sola vez, hago toda la búsqueda dentro de memoria y solo recupero el dato una sola vez cuando obtengo el puntero. Cuando hago la búsqueda, me meto por memoria y no hago el direct memory access, context switches, no llamo al sistema operativo. El único momento en que uso al OS, es cuando ya encuentra el puntero y va a recuperar el dato al disco.

Un índice clustered lo vemos con una estructura de datos armada dentro del archivo de datos y en las hojas tengo los datos. En uno non-clustered, tengo punteros hacia las hojas donde se encuentran los datos.

Puedo tener 1 clustered y ‘n’ non-clustered. Normalmente, las cláusulas where me dicen dónde podría definir índices.

Índice de hash: Una tabla de hash es organizar datos a partir del resultado de una función. La función de hash va a tener un dominio, lo cual es el conjunto de valores que puede generar esa función.

Esto genera una tabla de hash, la cual es una lista o arreglo en memoria donde cada espacio se conoce como bucket. Un bucket va a estar definido por el dominio de esa función, o los posibles valores de la función. El hash table va a meter los datos de acuerdo con el bucket correspondiente a la función de hash. Cuando hay datos que caen al mismo bucket ocurre una colisión. Aquí entra un algoritmo de manejo de colisiones. Lo más fácil es tener una lista enlazada en el bucket.

Dentro de un índice una tabla de hash vive dentro de un disco duro. La estructura de datos en el bucket va a tener un puntero al archivo de datos donde está la información.

Con solo utilizar una función de hash bien definida, reducimos el problema sustancialmente.

Toda operación que hagamos contra la base de datos va a pasar sobre la tabla de hash, ocupamos un puntero y recuperamos los datos. No vamos a usar una lista simple. El primer nivel de la tabla de hash es meter los datos, luego en cada bucket podemos meter otra tabla de hash u otra estructura de datos, como un árbol. Podríamos tener 3 tablas de hash anidadas para hacer las búsquedas más rápidas, por ejemplo.

Qué riesgo tiene un índice por hash? Si uno no está haciendo una base de datos y define un índice de hash sin saber la distribución de los datos, es un grave problema. Lo peor que puede pasar es que todos los registros caigan en el mismo bucket, entonces se convierte en un scan. La escogencia de la función de hash y el dato con el cual se escoge es sumamente relevante.

Expression Index Create based on WHERE expression SELECT * FROM T1 WHERE year > 2005; Si tenemos un índice que organiza los datos en cuanto al año, si descendemos en el árbol 20 niveles, hago 20 comparaciones y ejecutamos la operación de comparación todas las veces para los registros 1 por 1. En lugar de guardar dentro del nodo, utilizamos la expresión dentro del WHERE ya evaluada. Voy a indexar el resultado de la operación del WHERE más que indexar los datos. Así evito realizar el procesamiento del where. Por ejemplo, guarda True o False de acuerdo con si se cumplen los datos. Se usa cuando conocemos el patrón exacto de uso de los usuarios y sabemos que esa consulta es la que más se va a usar en la base de datos. Cuando conocemos el workload que ejecuta la base de datos, el índice de expresión nos ahorra todos los cálculos que hay que hacer.

Partial Index: Created over a subset of the dataset. Lo que nosotros andamos buscando es utilizar un subset de los datos. Por ejemplo, si armamos un índice con solo los estudiantes con carnét mayor a 2010, nos ahorramos el espacio de indexar los datos que casi no son recuperados. Esto hace que el índice sea más pequeño. Esto implica que lo puedo subir a memoria más fácilmente, por lo que las búsquedas son más rápidas. Esto se relaciona con la calidad de servicio para los usuarios que tienen prioridad. Cuando el índice no devuelve resultados, manda a hacer un scan a la tabla.