INTRODUCCIÓN A LOS B-TREES EN BASES DE DATOS

YESI ESTEBAN PANTOJA CUELLAR

DOCENTE

BRAYAN IGNACIO ARCOS BURBANO

INSTITUTO TECNOLÓGICO DEL PUTUMAYO

TECNOLOGÍA EN DESARROLLO DE SOFTWARE

QUINTO SEMESTRE

DESARROLLO DE BASE DE DATOS

MOCOA - PUTUMAYO

2024

INTRODUCCIÓN A LOS B-TREES EN BASES DE DATOS

Este documento explora el concepto de los B-trees como estructura de datos fundamental en los sistemas de gestión de bases de datos (DBMS). Abordaremos la definición, estructura, propiedades, operaciones básicas y ventajas de los B-trees, comparándolos con otros tipos de índices. También examinaremos sus aplicaciones en la optimización de las consultas de bases de datos y su papel en la eficiencia del almacenamiento y recuperación de datos.

TABLA DE CONTENIDO

Link vídeo: https://youtu.be/vKhNVpTbOUo

Link Github: https://github.com/Nick0oo/DBD_Proyecto

INTRODUCCIÓN AL B-TREE

Un B-tree es una estructura de datos auto-equilibrada que se utiliza ampliamente en la implementación de índices en bases de datos. Es un tipo de árbol de búsqueda que optimiza la búsqueda, inserción y eliminación de datos, especialmente en escenarios donde el conjunto de datos es grande y se necesita un acceso rápido y eficiente.

El B-tree fue desarrollado por Rudolf Bayer y Edward McCreight en 1972, y desde entonces se ha convertido en una estructura de datos esencial en la gestión de bases de datos relacionales. La estructura de un B-tree está diseñada para minimizar el número de operaciones de disco necesarias para acceder a un registro específico. Esto es crucial porque las operaciones de disco son significativamente más lentas que las operaciones de memoria. Al minimizar el acceso a disco, los B-trees garantizan un rendimiento rápido para las consultas de bases de datos.

CARACTERÍSTICAS Y FUNCIONAMIENTO DE UN B-TREE

Un B-tree tiene varias características que lo hacen adecuado para el manejo de índices en bases de datos:

- Equilibrado: El B-tree se mantiene equilibrado, lo que significa que todas las hojas del árbol se encuentran al mismo nivel. Esto asegura que las operaciones de búsqueda y actualización siempre se realicen en tiempo logarítmico, es decir, con complejidad O(log n).
- Nodos con múltiples hijos: A diferencia de un árbol binario de búsqueda, en un B-tree cada nodo puede tener más de dos hijos. La cantidad de hijos está determinada por el grado del árbol, que es un parámetro clave que afecta al rendimiento. Cuantos más hijos tenga cada nodo, menor será la altura del árbol.
- Almacenamiento en disco eficiente: Los B-trees son ideales para sistemas de bases de datos, ya que están diseñados para minimizar las operaciones de lectura y escritura en disco. Los nodos del árbol son relativamente grandes, lo que significa que más datos pueden almacenarse en un solo bloque de disco. Esto reduce la cantidad de accesos al disco necesarios durante las búsquedas.
- Búsqueda eficiente: En un B-tree, las búsquedas se realizan comenzando desde la raíz y siguiendo las ramas hacia las hojas. Debido a que el árbol está equilibrado, se puede llegar a una hoja con una cantidad mínima de accesos a los nodos, lo que garantiza que la búsqueda sea rápida.

ESTRUCTURA Y PROPIEDADES DEL B-TREE

Los B-trees tienen una estructura de árbol jerárquica, donde cada nodo contiene un conjunto de claves ordenadas y punteros a otros nodos. Las claves actúan como puntos de referencia para la búsqueda de datos. Un B-tree se caracteriza por las siguientes propiedades:

- Orden de árbol (t): determina el número mínimo y máximo de claves que un nodo puede contener.
- Altura del árbol: la distancia desde la raíz hasta cualquier hoja del árbol. La altura de un B-tree es logarítmica en relación con el número de claves. Esto garantiza búsquedas eficientes.
- Balanceo del árbol: los B-trees son siempre auto-equilibrados, lo que significa que la altura de cualquier rama del árbol no difiere significativamente. Este equilibrio es fundamental para mantener la eficiencia de las búsquedas.

Cada nodo en un B-tree, excepto la raíz, tiene un número mínimo de claves definido por el orden del árbol (t). La raíz puede tener entre 1 y 2t claves. Los nodos no hoja contienen claves que son mayores que las claves de sus hijos izquierdos y menores que las claves de sus hijos derechos. Los nodos hoja contienen las claves reales y los punteros a los datos asociados.

OPERACIONES BÁSICAS EN UN B-TREE

Las operaciones básicas en un B-tree incluyen la búsqueda, la inserción y la eliminación. Estas operaciones se basan en el orden de las claves dentro del árbol y en la estructura de árbol jerárquica.
La búsqueda de una clave en un B-tree implica un recorrido descendente desde la raíz hasta las hojas. En cada nodo, la clave se compara con las claves del nodo actual. Si la clave coincide con una clave en el nodo, se encuentra la clave y se puede acceder a los datos asociados. Si no coincide, se selecciona el hijo apropiado según el orden de las claves.
La inserción de una nueva clave implica encontrar la posición apropiada en la hoja del árbol. Si el nodo está lleno, se divide en dos nodos. La clave media del nodo dividido se mueve hacia arriba al nodo padre. La eliminación de una clave implica encontrar la posición de la clave y eliminarla. Si el nodo se queda vacío, se fusiona con un nodo hermano o se elimina.

INSERCIÓN Y ELIMINACIÓN DE NODOS

La inserción y la eliminación de nodos en un B-tree son procesos que requieren la actualización de la estructura del árbol para mantener su balanceo y propiedades de orden. La inserción de un nuevo nodo implica encontrar la ubicación correcta en la hoja del árbol donde debe agregarse la clave.
Si el nodo está lleno, se divide en dos nodos, y la clave media se mueve hacia arriba al nodo padre. Este proceso puede propagarse hacia arriba a través del árbol si los nodos padres también están llenos.
La eliminación de un nodo implica encontrar la clave y eliminarla del nodo. Si el nodo se queda vacío, se fusiona con un nodo hermano. Este proceso también puede propagarse hacia arriba a través del árbol si los nodos padres se quedan con menos claves que el mínimo requerido.
La inserción y la eliminación de nodos requieren ajustes en la estructura del árbol para garantizar que el balanceo y el orden se mantienen.
En el caso de la inserción, si un nodo se llena durante la operación, se requiere la división del nodo, lo que genera un nuevo nodo y una nueva clave en el nodo padre. Por otro lado, durante la eliminación, si un nodo se queda vacío, se fusiona con un nodo hermano o se elimina del árbol. La gestión de estas operaciones es crucial para mantener la eficiencia del B-tree.

VENTAJAS DEL USO DE B-TREES EN BASES DE DATOS

Los B-trees ofrecen varias ventajas que los hacen ideales para su uso en bases de datos, como:

- Búsquedas eficientes: la estructura de árbol de búsqueda y el balanceo del árbol garantizan que las búsquedas se puedan realizar en tiempo logarítmico. Esto es esencial para un rendimiento rápido de las consultas de bases de datos.
- Inserciones y eliminaciones eficientes: las operaciones de inserción y eliminación se realizan con un rendimiento comparable a las búsquedas. Esto es crucial para mantener la integridad de los datos y la consistencia en bases de datos dinámicas.
- Acceso secuencial eficiente: los B-trees permiten un acceso secuencial rápido a los datos, lo que es útil para las consultas que requieren la recuperación de registros en orden.
- Uso eficiente del espacio: los B-trees utilizan el espacio de almacenamiento de manera eficiente al almacenar las claves y los punteros de manera compacta en cada nodo.

Además, la naturaleza auto-equilibrada de los B-trees garantiza que la altura del árbol permanezca lo más baja posible, lo que optimiza el rendimiento de las operaciones de disco. Esto es especialmente importante en el contexto de bases de datos, donde la velocidad de acceso a los datos es crítica.

APLICACIONES Y USO DEL B-TREE EN SISTEMAS DE GESTIÓN DE BASES DE DATOS

Los B-trees son ampliamente utilizados en la gestión de bases de datos relacionales (RDBMS) para la implementación de índices. Los índices ayudan a acelerar las consultas de bases de datos al proporcionar un camino rápido para acceder a los registros de datos. En un RDBMS, los B-trees se utilizan para indexar las columnas de las tablas, lo que permite la recuperación rápida de datos en función de los valores de esas columnas.

Las aplicaciones de los B-trees en sistemas de gestión de bases de datos incluyen:

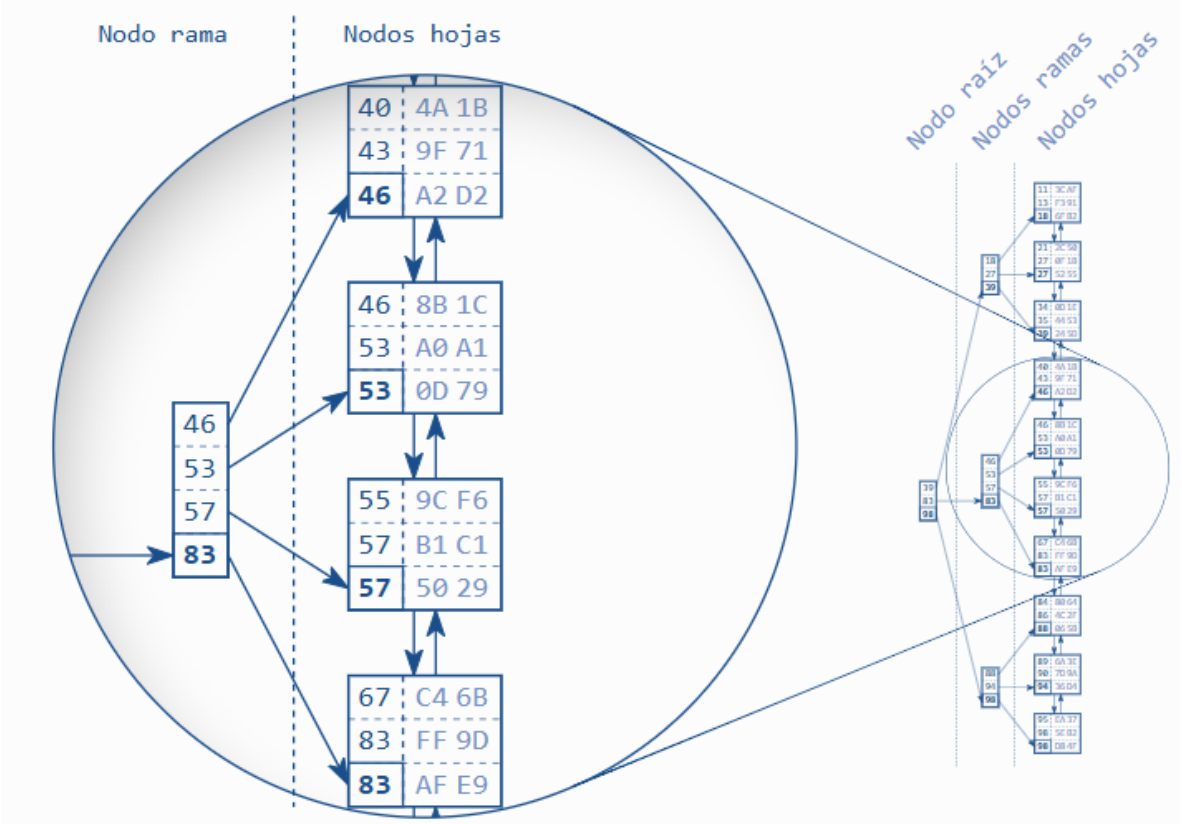
- Indexación de claves primarias: las claves primarias son utilizadas para identificar de forma única cada fila en una tabla. Los índices de B-tree se utilizan para garantizar la eficiencia y la integridad de las claves primarias.
- Indexación de claves foráneas: las claves foráneas se utilizan para establecer relaciones entre diferentes tablas. Los índices de B-tree se utilizan para hacer cumplir las restricciones de integridad referencial, asegurando la coherencia de los datos entre las tablas.
- Indexación de columnas no clave: los B-trees también se pueden utilizar para indexar columnas que no son claves primarias o foráneas. Esto puede mejorar el rendimiento de las consultas que filtran o ordenan datos en función de esos valores de columna.

Al utilizar B-trees para indexar los datos, los sistemas de gestión de bases de datos pueden acelerar significativamente las consultas, mejorar el rendimiento general y optimizar el proceso de almacenamiento y recuperación de datos. La estructura y las propiedades de los B-trees hacen de esta estructura de datos una opción ideal para la gestión de bases de datos eficientes.

Ejemplo 1.

Los nodos hojas del índice están almacenados en un orden aleatorio, es decir su posición en el disco no corresponde a la posición lógica según el orden del índice. Es lo mismo que una guía telefónica con las hojas mezcladas. Si buscas “Sánchez” pero primero abres la guía telefónica por “Rodríguez”, nada nos asegura que Rodríguez siga a Sánchez. Una base de datos necesita una segunda estructura para encontrar rápidamente los datos dentro de las hojas mezcladas: un árbol de búsqueda equilibrado, o sea, un B-Tree.

Ejemplo 2.



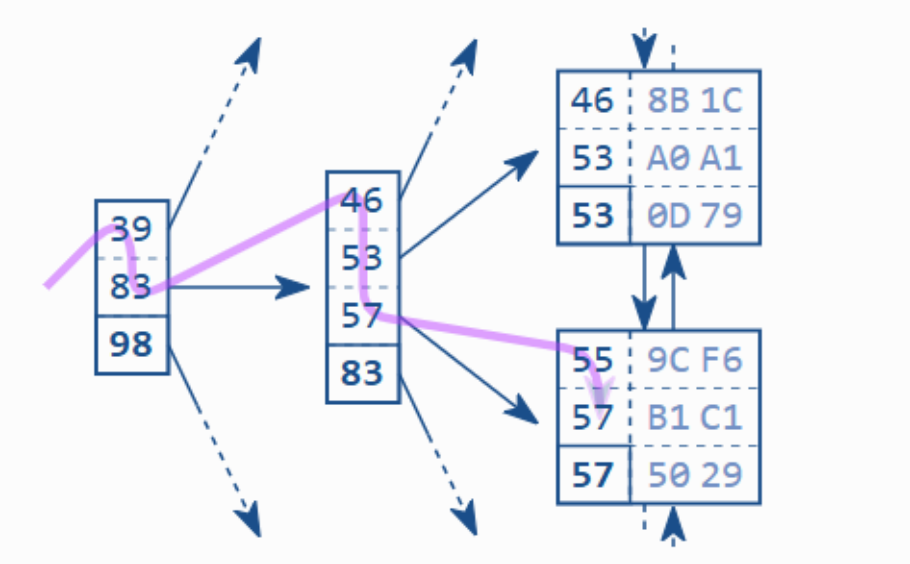
La imagen anterior muestra un ejemplo de un índice con 30 entradas. La lista doblemente enlazada establece el orden lógico entre los nodos hojas. La raíz y sus ramas permiten hacer una búsqueda rápida entre los nodos hojas.

En la ilustración destaca el nodo rama y sus nodos hojas a los que hace referencia. Cada entrada del nodo rama corresponde al valor más grande respecto a su nodo hoja. Por ejemplo, el valor más grande en el primer nodo hoja es 46, que está almacenado en su propia entrada del nodo rama. Lo mismo sucede para todos los nodos hojas, que al final de su nodo rama tienen los valores 46, 53, 57 y 83. De acuerdo con este plan, un nivel de rama está construido hasta que todos los nodos hojas estén cubiertos por un nodo rama.

El siguiente nivel está construido de manera similar, pero encima del primer nivel de rama. Se repite este procedimiento hasta que todas las llaves llenan un único nodo, el nodo raíz. La estructura es

un árbol de búsqueda equilibrado porque la profundidad del árbol es idéntica en cada posición; la distancia entre el nodo raíz y los nodos hojas es idéntica en todas las partes del árbol.

El recorrido de un B-tree viene siendo de la siguiente manera:



Ejemplo para una base de datos

1. Indexación de claves primarias (Búsqueda de clientes por ID)

Una base de datos de clientes necesita identificar rápidamente un cliente por su ID.

Estructura de la tabla customers:

```
CREATE TABLE customers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    email VARCHAR(255),
    phone VARCHAR(20)
);
```

Creación del índice B-tree en la columna `id` (clave primaria):

```
CREATE UNIQUE INDEX idx_customers_id ON customers(id);
```

Consulta que busca un cliente por su ID:

```
SELECT *
FROM customers
WHERE id = 123;
```

La clave primaria id de la tabla customers ya está indexada automáticamente por el sistema de gestión de bases de datos (RDBMS) con un índice B-tree, lo que garantiza búsquedas rápidas. Este

índice asegura la integridad de los datos al ser único y permite la identificación eficiente de registros específicos.

Ventajas:

- Garantiza la unicidad de la clave primaria.
- Acceso directo a los datos asociados al id en tiempo logarítmico.
- Optimiza la integridad referencial cuando otras tablas usan la clave primaria como clave foránea.

2. Indexación de claves foráneas (Filtrar ordenes por ID de cliente)

Tenemos la siguiente tabla de orders (ordenes) en la cual mediante claves foráneas buscaremos que nos muestre las ordenes realizadas por cada cliente, siguiendo esto pues tendríamos la siguiente tabla:

Estructura de la tabla orders:

```
CREATE TABLE orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE,
    idCustomer INT,
    FOREIGN KEY (idCustomer) REFERENCES customers(id)
);
```

Creación del índice B-tree en la columna idCustomer (clave foránea):

```
CREATE INDEX idx_orders_idcustomer ON orders(idCustomer);
```

Consulta que filtra las órdenes por el ID de cliente:

```
SELECT *
FROM orders
WHERE idCustomer = 123;
```

La columna idCustomer en la tabla orders es una clave foránea que establece una relación con la tabla customers. Al crear un índice B-tree sobre idCustomer, mejoramos la eficiencia de las consultas que filtran registros en la tabla orders según el idCustomer. Este índice también ayuda a asegurar la integridad referencial, lo que significa que solo los valores que existen en la tabla customers pueden ser insertados en la tabla orders.

3. Indexación de columnas no clave (Filtrar productos por precio)

En este ejemplo vamos a hacer una consulta del precio de ciertos productos, teniendo en cuenta el significado de la indexación de columnas no clave, la columna price no vendría siendo ni una clave primaria ni foránea, pero se utiliza un índice B-tree para mejorar el rendimiento de las consultas.

Estructura de la tabla products:

```sql
CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    price DECIMAL(10, 2),
    stock INT
);
```

Creación del índice B-tree en la columna price:

```sql
CREATE INDEX idx_products_price ON products(price);
```

Consulta que filtra los productos por precio:

```sql
SELECT *
FROM products
WHERE price BETWEEN 50 AND 100;
```

La columna price no es ni clave primaria ni foránea, pero al indexarla con un B-tree, mejoramos el rendimiento de las consultas que filtran los productos en función de su precio. Esto permite que la base de datos localice rápidamente los productos dentro del rango de precios, sin tener que escanear toda la tabla.

Conclusiones y consideraciones finales

Los B-trees son una estructura de datos fundamental en la gestión de bases de datos. Proporcionan una forma eficiente de organizar y acceder a los datos, lo que optimiza el rendimiento de las consultas y la eficiencia del almacenamiento y recuperación de datos. Su estructura auto-equilibrada, junto con sus propiedades de orden y balanceo, garantiza búsquedas rápidas, inserciones y eliminaciones eficientes. El uso de B-trees en bases de datos es esencial para la optimización de las consultas y la gestión eficaz de los datos. Al entender las ventajas y aplicaciones de los B-trees, los desarrolladores de bases de datos pueden diseñar sistemas de gestión de datos eficientes y de alto rendimiento. Los B-trees son una herramienta esencial en la gestión de grandes conjuntos de datos, y su importancia continuará creciendo a medida que las bases de datos se vuelven más complejas y extensas.