2. Métodos de acceso a una BD

Una de las características que hacen que las bases de datos sean poderosas y eficientes es la capacidad de acceder a los datos de múltiples formas. Dependiendo de cómo necesitemos interactuar con los datos, podemos elegir un método de acceso que sea más adecuado.

2.1 Implementación de los accesos por posición

El acceso por posición a una base de datos se refiere a cuando accedemos a los datos basándonos en su ubicación física en la base de datos. Este es el método más rápido de acceder a los datos porque no requiere que el sistema busque en todo el conjunto de datos; sin embargo, este método tiene limitaciones ya que necesitamos saber la ubicación exacta de los datos que estamos buscando. Este tipo de acceso es común en los sistemas de archivos y en las bases de datos orientadas a objetos.

Por ejemplo, si tienes una base de datos con información de estudiantes y conoces la posición exacta de los datos de un estudiante en particular, puedes recuperar esos datos directamente sin tener que buscar en toda la base de datos.

- Demostración práctica de los accesos por posición: Para demostrar el acceso por posición, vamos a utilizar un ejemplo sencillo con un lenguaje de programación, ya que las bases de datos relacionales modernas no suelen utilizar este método. Considera una lista de estudiantes en Python:

```
estudiantes = ["Ana", "Pedro", "Juan", "María", "Carlos"]
```

Si queremos acceder al tercer estudiante en la lista (Juan), podemos hacerlo por posición. En Python, las listas están indexadas desde 0, por lo que "Juan" está en la posición 2:

```
print(estudiantes[2])
```

Esto nos dará "Juan", porque estamos accediendo directamente a esa posición en la lista.

2.2 Implementación de los accesos por valor

El acceso por valor es cuando accedemos a los datos basándonos en un valor específico en lugar de su ubicación. Este es el método más comúnmente utilizado para acceder a los datos en una base de datos relacional. Por ejemplo, podríamos querer encontrar todos los registros de estudiantes con una calificación específica. En este caso, no necesitamos saber dónde están almacenados físicamente los datos; simplemente necesitamos saber el valor que estamos buscando.

Por ejemplo:

SELECT * FROM estudiantes WHERE calificacion = 'A';

- Demostración práctica de los accesos por valor: usaremos una base de datos relacional con SQL. Imagina que tenemos una tabla `estudiantes` con los siguientes datos:

id	nombre	grado
1	Ana	10
2	Pedro	11
3	Juan	12
4	María	10
5	Carlos	11

Si queremos encontrar todos los estudiantes que están en el grado 10, podemos hacer una consulta SQL como esta:

```
SELECT * FROM estudiantes WHERE grado = 10;
```

Esto nos dará todos los estudiantes con el valor "10" en la columna "grado", sin importar su posición en la tabla:

id	nombre	grado
1	Ana	10
4	María	10

Como puedes ver, en el acceso por valor no importa dónde estén ubicados los datos en la base de datos, solo importa el valor que estamos buscando.

- Otra demostración práctica utilizando el acceso por valor en SQL: Supongamos que tenemos una base de datos con una tabla llamada `libros` que tiene los siguientes datos:

id	título	autor	año_publicacion
1	Moby Dick	Herman Melville	1851
2	Orgullo y Prejuicio	Jane Austen	1813
3	1984	George Orwell	1949
4	Matadero Cinco	Kurt Vonnegut	1969
5	Cien años de soledad	Gabriel Garcia Marquez	1967

Ahora, supón que queremos encontrar todos los libros publicados después del año 1900. En este caso, usaríamos el acceso por valor para buscar en la columna `año_publicacion` el valor que queremos:

```
SELECT * FROM libros WHERE año_publicacion > 1900;
```

Este comando devolverá todos los libros que se publicaron después del año 1900:

id	título	autor	año_publicacion
3	1984	George Orwell	1949
4	Matadero Cinco	Kurt Vonnegut	1969
5	Cien años de soledad	Gabriel Garcia Marquez	1967

En esta consulta SQL, estamos utilizando el acceso por valor para encontrar libros basados en el valor de su año de publicación, sin importar dónde se encuentren esos libros en la tabla de la base de datos.

2.2.1 Índices

Los índices son estructuras de datos que mejoran la velocidad de las operaciones de lectura en una base de datos, a expensas de operaciones adicionales de escritura y uso de más espacio de almacenamiento para mantenerlos. Piensa en un índice como la tabla de contenido de un libro: te ayuda a encontrar la información que buscas mucho más rápidamente que si tuvieras que recorrer todo el libro.

Los índices son utilizados en el ámbito de las bases de datos para acelerar el tiempo de consulta. Cuando realizas una consulta en una base de datos sin índices, el sistema debe recorrer cada fila de cada tabla para encontrar los datos relevantes, lo que se conoce como una búsqueda de "tabla completa". Esto puede ser ineficiente y lento, especialmente en bases de datos con millones o incluso miles de millones de filas.

Por otro lado, con los índices, las bases de datos pueden encontrar los datos relevantes rápidamente sin tener que buscar en todas las filas. Los índices son especialmente útiles para las operaciones que recuperan una pequeña proporción de las filas de la tabla, como las operaciones de búsqueda o clasificación.

El manejo de índices es un tema de equilibrio. Por un lado, pueden acelerar las operaciones de lectura. Por otro lado, cada vez que se escribe en la base de datos (es decir, inserciones, actualizaciones y eliminaciones), los índices también necesitan ser actualizados. Esto puede ralentizar las operaciones de escritura. Además, los índices también ocupan espacio de almacenamiento.

Los índices pueden ser simples (basados en una única columna) o compuestos (basados en varias columnas). Un índice simple es útil cuando las consultas buscan valores en una sola columna, mientras que un índice compuesto es útil cuando las consultas buscan valores en varias columnas.

En términos de su implementación, los índices pueden tomar varias formas, siendo las más comunes las estructuras de árbol (como los árboles B o árboles B+), que permiten una búsqueda rápida, inserción y eliminación de entradas.

En resumen, los índices son una herramienta valiosa en la gestión de bases de datos que, cuando se utilizan correctamente, pueden acelerar significativamente las operaciones de consulta. Sin embargo, también es importante tener en cuenta las implicaciones en el rendimiento de las operaciones de escritura y en el uso del espacio de almacenamiento.

2.2.2 Árboles B+

Los Árboles B+ son una especie de estructura de datos utilizada en las bases de datos para facilitar el acceso a los datos de manera rápida y eficiente. Son similares a los Árboles B, otra estructura de

datos, pero con algunas mejoras que los hacen particularmente útiles para ciertas aplicaciones en las bases de datos.

Para entender esto, puedes imaginar un árbol en la naturaleza con su tronco, ramas y hojas. Un Árbol B+ es similar, pero en lugar de tener hojas, tiene punteros (direcciones de memoria) que llevan a los datos.

Una de las características principales de los Árboles B+ es que todas las claves de búsqueda (el dato que estás buscando) se encuentran en las hojas del árbol, y estas hojas están vinculadas entre sí para permitir una rápida búsqueda secuencial de datos. Esto hace que los Árboles B+ sean ideales para las operaciones de búsqueda de rangos y recorridos secuenciales.

Los punteros adicionales en los Árboles B+, en comparación con los Árboles B, se utilizan para apuntar a los datos o a los registros en el disco, en lugar de almacenar los datos dentro del propio árbol. Esta es una de las razones por las que los Árboles B+ son una opción popular para implementar índices en bases de datos.

Los Árboles B+ son eficientes porque reducen la cantidad de veces que el sistema de base de datos tiene que ir a buscar en el disco duro, lo que es un proceso relativamente lento. En su lugar, el sistema puede seguir rápidamente los punteros en la memoria, que es mucho más rápido, hasta que encuentre el dato que está buscando.

En resumen, los Árboles B+ son estructuras de datos esenciales en la implementación de bases de datos eficientes. Son especialmente útiles para la implementación de índices debido a su velocidad y eficiencia en las operaciones de búsqueda y recorrido. Si bien la explicación puede sonar técnica, lo más importante a entender es que ayudan a hacer que las bases de datos funcionen más rápido y de manera más eficiente.

2.2.3 Dispersión

La dispersión, también conocida como hashing, es otra forma efectiva de acceder a los datos en una base de datos. Este método se basa en algo llamado una función hash, que es como una receta matemática que toma una entrada (como el título de un libro) y produce un número único (el valor hash). Este valor hash es luego utilizado para determinar exactamente dónde se almacenan los datos correspondientes en la base de datos.

Imagínate que tienes una caja con muchas ranuras y quieres guardar tus libros en ella. En lugar de poner los libros en las ranuras al azar, utilizas una función hash, que toma el título del libro y te da un número único. Usas ese número para decidir en qué ranura poner el libro. Luego, cuando quieres encontrar el libro de nuevo, sólo tienes que aplicar la misma función hash al título del libro y te dará el número de la ranura donde está guardado. Esto hace que encontrar libros (o, en nuestro caso, datos en una base de datos) sea extremadamente rápido.

El hashing es extremadamente eficiente para buscar valores individuales porque proporciona un acceso directo al lugar exacto donde se almacenan los datos. Esto lo hace muy rápido para recuperar los datos.

Sin embargo, la dispersión tiene sus limitaciones. Una de las más importantes es que no es muy buena para las consultas de rango. Imagina que quieres encontrar todos los libros cuyos títulos comienzan con la letra "A". Con el método de dispersión, no hay manera fácil de hacer esto, ya que los valores hash generados para estos libros podrían estar dispersos por toda la base de datos.

En resumen, la dispersión es una forma muy eficiente de acceder a los datos si estás buscando un valor específico. Sin embargo, tiene sus limitaciones y puede no ser la mejor opción para todos los tipos de consultas. Así como la función hash te ayuda a encontrar rápidamente un libro en tu caja, también ayuda a las bases de datos a recuperar los datos rápidamente y de manera eficiente.

2.2.4 Índices agrupados

Los índices agrupados son un tipo especial de índices en una base de datos. Lo que los hace únicos es la forma en que influyen en cómo se almacenan los datos. Cuando se crea un índice agrupado en una tabla, los datos de esa tabla se reorganizan físicamente para coincidir con el orden del índice. Esto es como reorganizar un estante de libros para que estén en orden alfabético por el autor, lo que hace que sea mucho más fácil y rápido encontrar un libro.

Este tipo de índice es extremadamente eficiente para las consultas que buscan rangos de valores. Imagina que estás buscando todos los libros escritos por autores cuyos apellidos comienzan con "M". Si los libros están ordenados alfabéticamente por autor, es fácil encontrar el comienzo y el final de esa sección y recoger todos los libros entre ellos. Lo mismo sucede con las consultas de rango en una base de datos con un índice agrupado.

Los índices agrupados son particularmente útiles cuando se está buscando un conjunto de datos que tiene una relación natural de ordenamiento. Por ejemplo, si se está buscando una lista de transacciones bancarias entre dos fechas, un índice agrupado en la columna "Fecha" de la tabla "Transacciones" permitiría una búsqueda rápida y eficiente.

Sin embargo, es importante tener en cuenta que cada tabla puede tener solo un índice agrupado, ya que los datos en la tabla solo pueden ordenarse de una forma a la vez.

En resumen, los índices agrupados son una herramienta poderosa en el manejo de bases de datos. Al igual que ordenar un estante de libros hace que sea más fácil encontrar el libro que estás buscando, los índices agrupados hacen que las consultas de rangos sean más rápidas y eficientes. Sin embargo, su uso debe ser cuidadosamente considerado debido a la limitación de tener solo un índice agrupado por tabla.

2.3 Implementación de los accesos por diversos valores

En algunas situaciones, es posible que necesitemos acceder a los datos utilizando múltiples valores. Este tipo de acceso a la base de datos es común en las consultas complejas que requieren condiciones múltiples para recuperar los datos. Por ejemplo, podríamos querer encontrar todos los registros de estudiantes que tienen una calificación específica y están en un grado específico.

Por ejemplo:

SELECT * FROM estudiantes WHERE calificacion = 'A' AND grado = '10';

2.3.1 Implementación de los accesos directos

La implementación de los accesos directos en las bases de datos se refiere a la capacidad de llegar directamente a la información que necesitamos en una sola operación. En otras palabras, no tenemos que pasar por varios datos o registros irrelevantes para encontrar lo que buscamos.

Para ilustrar esto, imaginemos que estás buscando un libro en una biblioteca muy grande. Si conoces el número exacto de la estantería y la ubicación exacta del libro en esa estantería, puedes ir directamente a ese lugar y obtener el libro. No tienes que mirar en todas las estanterías o revisar cada libro. Esta es la esencia del acceso directo en las bases de datos.

Uno de los usos más comunes del acceso directo es cuando conocemos la clave primaria de un registro. La clave primaria es como el número único de la estantería y la ubicación del libro en nuestro ejemplo. Cada registro en una base de datos tiene una clave primaria única, y si conocemos esa clave, podemos recuperar todo el registro en una sola operación.

Por ejemplo, podríamos tener una tabla llamada 'estudiantes' en nuestra base de datos, donde cada estudiante tiene una 'id' única. Si queremos encontrar la información del estudiante con el id '123', podemos utilizar una consulta SQL de acceso directo:

SELECT * FROM estudiantes WHERE id = 123;

Esta consulta irá directamente al estudiante con el id '123' y recuperará toda su información. Este es un método muy eficiente para obtener datos y es una técnica ampliamente utilizada en la gestión de bases de datos relacionales.

2.3.2 Implementación de los accesos secuenciales y mixtos

El acceso secuencial es como recorrer las páginas de un libro una por una, desde el principio hasta el final. En el contexto de las bases de datos, es cuando recorres los registros de la base de datos en orden, uno tras otro. Este método es especialmente útil cuando necesitamos procesar o revisar todos los registros de la base de datos, por ejemplo, para calcular un total, un promedio o realizar algún otro tipo de agregación de datos.

Por ejemplo, si quisiéramos calcular la calificación promedio de todos los estudiantes, podríamos utilizar un acceso secuencial para leer cada registro de estudiante y sumar todas las calificaciones. Al final, dividiríamos la suma total por el número de estudiantes para obtener el promedio.

Sin embargo, el acceso secuencial puede ser ineficiente si tienes una gran cantidad de datos y sólo necesitas procesar una pequeña parte de ellos. En estos casos, puede llevar mucho tiempo leer todos los datos innecesarios.

Por otro lado, el acceso mixto combina lo mejor del acceso directo y secuencial. Nos permite saltar directamente a una ubicación específica en la base de datos (como con el acceso directo) y luego recorrer los registros desde allí (como con el acceso secuencial).

Por ejemplo, si queremos calcular la calificación promedio de los estudiantes en un grado específico, podríamos utilizar una consulta SQL para seleccionar todos los registros de estudiantes en ese grado en particular (esto sería el acceso directo). A continuación, podríamos recorrer esos registros para calcular el promedio (esto sería el acceso secuencial).

En SQL, esta consulta podría verse así:

```
SELECT AVG(calificacion) FROM estudiantes WHERE grado = '10';
```

Esta consulta primero selecciona directamente todos los estudiantes en el grado '10' y luego calcula el promedio de sus calificaciones de manera secuencial. Así, los accesos mixtos nos permiten combinar la eficiencia del acceso directo con la capacidad del acceso secuencial para procesar múltiples registros de manera ordenada.

Espero que este capítulo te haya proporcionado una buena introducción a los diferentes métodos de acceso a una base de datos.