

Curso: BISOFT-20.

Estructuras de datos 2

Consigna Actividad: Estudio de caso #1

Temática: BTREES Y TABLAS HASH

Profesor:

Ana Isabel Méndez Brenes.

Estudiantes:

Falon Carmona Arias

Malcom Enrique Quirós Madriz,

Daniel Alberto Rodríguez Orozco,

Santiago Zeledón Marín

Fecha de entrega: noviembre 2024.

Tema 1: Tablas Hash.....	2
1. Infografía resumen del tema Tablas Hash:.....	2
2. Infografía de los algoritmos Hash.....	2
3. Infografía de los algoritmos Hash.....	3
Tema 2: BTrees (Árboles B).....	3
Sección Teórica:.....	3
1. Infografía: ¿Cómo funciona la indexación de bases de datos en forma de árbol B?.....	4
2. Infografía ¿Cómo funciona la indexación de bases de datos en forma de árbol B+?.....	4
3. Infografía ¿Cómo implementar una indexación en SQL (puede ser SQLite o SQLServer? Deberá incluir las operaciones para la creación y para el funcionamiento básico CRUE.....	4
4. Infografía con la opinión del grupo.....	4
Sección Práctica:.....	4
1. Infografía HASHMAP en JAVA.....	4
2. Implemente un programa usando la librería de HashMap de Java con las siguientes operaciones:.....	4

Tema 1: Tablas Hash

Sección Teórica:

1. Infografía resumen del tema Tablas Hash:

- Concepto lógico
- Tablas Hash Abiertas
- Tablas Hash Cerradas

TABLAS HASH



Concepto

Las tablas hash son estructuras de datos que se utilizan para almacenar un número elevado de datos sobre los que se necesitan operaciones de búsqueda e inserción muy eficientes. Funciona transformando la clave con una función hash en un hash, un número que identifica la posición (casilla o cubeta) donde la tabla hash localiza el valor deseado.



Tablas hash abiertas

En este método, cada posición o "ranura" de la tabla hash contiene una lista enlazada (o cualquier otra estructura de datos) donde se almacenan los elementos que tienen el mismo valor de hash.

Tablas hash cerradas

En este método, todos los elementos se almacenan directamente en la tabla hash sin utilizar listas enlazadas.



Operaciones básicas

- inserción(llave, valor)
- búsqueda(llave) que devuelve valor
- La mayoría de las implementaciones también incluyen borrar(llave). T

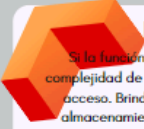
2. Infografía de los algoritmos Hash

- Objetivo de un algoritmo hash
- Importancia de un buen algoritmo hash
- Descripción de los algoritmos Hash:
 - i. Hash multiplicativo
 - ii. Hash restas sucesivas
 - iii. Hash prueba lineal

- iv. Hash doble lineal v. Método de división
- vi. Método Hash Medios Cuadrados,
- vii. Método Hash Random Probing,
- viii. Truncamiento
- ix. Método del Pliegue.

Objetivo

Brindar una estructura de datos almacenar y acceder de manera rápida, realizar inserción y eliminación de pares clave-valor.



Importancia

Si la función hash es eficiente se puede lograr una complejidad de tiempo constante para las operaciones de acceso. Brinda eficiencia en tiempos de búsqueda y almacenamiento de grandes volúmenes de datos con optimización del uso de memoria

Descripción de los algoritmos Hash:

- i. Hash multiplicativo: Esta técnica trabaja multiplicando la clave k por sí misma o por una constante, usando después alguna porción de los bits del producto como una localización de la tabla hash.
- ii. Hash restas sucesivas: Esta función se emplea con claves numéricas entre las que existen huecos de tamaño conocido, obteniéndose direcciones consecutivas.
- iii. Hash prueba lineal : Consiste en que una vez detectada la colisión se debe recorrer el arreglo secuencialmente a partir del punto de colisión, buscando al elemento
- iv. Hash doble lineal: El hash doble es una técnica de programación informática que se utiliza junto con el direccionamiento abierto en las tablas hash para resolver colisiones de hash, mediante el uso de un hash secundario de la clave como compensación cuando se produce una colisión .
- v. Método de división : En este caso la función se calcula simplemente como $h(k) = k \bmod M$ usando el 0 como el primer índice de la tabla hash de tamaño M .
- vi. Método Hash Medios Cuadrados: El método del cuadrado medio eleva al cuadrado el valor de la clave y luego toma los r bits del medio del resultado, lo que da un valor en el rango de 0 a $2^r - 1$.
- vii. Método Hash Random Probing: El sondeo implica encontrar otra ubicación en la tabla hash para el elemento de datos cuando se produce una colisión .
- viii. Truncamiento: Consiste en tomar algunos dígitos de la clave y formar con ellos una dirección. Este método es de los más sencillos, pero es también de los que ofrecen menos uniformidad en la distribución de las claves.
- ix. Método del Pliegue: Consiste en partir demasiado grande, pudiendo ocurrir que no pueda ser almacenado en memoria.

3. Infografía de los algoritmos Hash

- MD2, MD4, MD5
- SHA-1, SHA-2.

ALGORITMOS HASH

Algoritmos MD

1. MD2 (Message Digest Algorithm 2)
 - Desarrollado por: Ronald Rivest en 1989.
 - Longitud del Hash: 128 bits.
 - Usos: Fue utilizado principalmente para firmas digitales.
 - Seguridad: Vulnerable a ataques modernos y no es recomendado en la actualidad.
2. MD4 (Message Digest Algorithm 4)
 - Desarrollado por: Ronald Rivest en 1990.
 - Longitud del Hash: 128 bits.
 - Usos: Diseñado para sistemas de alta velocidad.
 - Seguridad: Vulnerable a ataques de colisión, ya no se considera seguro.
3. MD5 (Message Digest Algorithm 5)
 - Desarrollado por: Ronald Rivest en 1991.
 - Longitud del Hash: 128 bits.
 - Usos: Amplio uso en la verificación de integridad de archivos.
 - Seguridad: Vulnerable a colisiones y ataques de fuerza bruta; desaconsejado para aplicaciones de seguridad.

Algoritmos SHA

4. SHA-1 (Secure Hash Algorithm 1)
 - Desarrollado por: NSA en 1993 y revisado en 1995.
 - Longitud del Hash: 160 bits.
 - Usos: Fue ampliamente usado en certificados SSL y firmas digitales.
 - Seguridad: Vulnerable a ataques de colisión; desaconsejado en aplicaciones de seguridad.
5. SHA-2 (Secure Hash Algorithm 2)
 - Desarrollado por: NSA en 2001.
 - Versiones: SHA-224, SHA-256, SHA-384, y SHA-512 (longitud de hash).
 - Longitud del Hash: Varía entre 224 y 512 bits, según la versión.
 - Usos: Ampliamente utilizado en criptografía moderna (ej. Bitcoin, SSL, etc.).
 - Seguridad: Considerado seguro y resistente a colisiones y ataques de fuerza bruta en la mayoría de aplicaciones.

4. Infografía de la implementación de tablas hash en SQL SERVER

HASHBYTES (TRANSACT-SQL)

Sintaxis básica de HASHBYTES

```
HASHBYTES ( '<algorithm>', { @input | 'input' } )  
<algorithm>::= MD2 | MD4 | MD5 | SHA | SHA1 | SHA2_256 | SHA2_512
```

Argumentos

<algorithm>
Identifica el algoritmo hash que se va a utilizar para realizar el hash de la entrada. Es un argumento requerido y no tiene valor predeterminado. Las comillas simples son necesarias.

<@input
Especifica una variable que contiene los datos en los que se va a realizar el hash. @input es varchar, nvarchar o varbinary.

"input"
Especifica una expresión que se evalúa como un carácter o una cadena binaria que se va a aplicar el algoritmo hash.

Ejemplos

Devuelve el valor hash de una variable

En el siguiente ejemplo se devuelve el hash SHA2_256 de los datos nvarchar almacenados en la variable @HashThis.

```
DECLARE @HashThis NVARCHAR(32);  
SET @HashThis = CONVERT(NVARCHAR(32),'dsldkjLK85klahny$no00#knf');  
SELECT HASHBYTES('SHA2_256', @HashThis);
```

Devuelve el valor hash de una columna de tabla

En el ejemplo siguiente se devuelve el valor hash SHA2_256 de los valores de la columna c1 de la tabla Test1.

```
CREATE TABLE dbo.Test1 (c1 NVARCHAR(32));  
INSERT dbo.Test1 VALUES ('This is a test.');
```

```
INSERT dbo.Test1 VALUES ('This is test 2.');
```

```
SELECT HASHBYTES('SHA2_256', c1) FROM dbo.Test1
```

(HASHBYTES (Transact-SQL))

Sección Práctica: 1. Implemente en SQL (de su gusto) una tabla de tipo HASHBYTES y proceda a realizar las operaciones básicas CRUE además de la instrucción que permita desplegar el código Hash generado.

```
CREATE TABLE Usuarios (  
    Id INT PRIMARY KEY IDENTITY,  
    Nombre NVARCHAR(50) NOT NULL,  
    Email NVARCHAR(100) NOT NULL,  
    Password NVARCHAR(100) NOT NULL,  
    HashPassword VARBINARY(64)  
);
```

```
INSERT INTO Usuarios (Nombre, Email, Password, HashPassword)  
VALUES ('Juan Pérez', 'juan.perez@example.com', 'password',  
HASHBYTES('SHA2_256', 'password'));
```

```
SELECT Id, Nombre, Email, Password, HashPassword  
FROM Usuarios;
```

```
UPDATE Usuarios  
SET Password = 'newpassword',  
    HashPassword = HASHBYTES('SHA2_256', 'newpassword')  
WHERE Id = 1;
```

```
DELETE FROM Usuarios  
WHERE Id = 1;
```

```
SELECT HashPassword  
FROM Usuarios  
WHERE Id = 1;
```

Tema 2: BTrees (Árboles B)

Sección Teórica:

1. Infografía: ¿Cómo funciona la indexación de bases de datos en forma de árbol B?

ÁRBOLES B EN BASES DE DATOS



¿Qué es un árbol B?

Un Árbol B es una estructura de datos equilibrada que permite acceder rápidamente a registros.

Su objetivo es mejorar el rendimiento de consultas de búsqueda, inserción, y eliminación en bases de datos, organizando datos en forma jerárquica.



Estructura de Árbol

- **Nodos:** Compuestos por un número fijo de claves y punteros a hijos.
- **Raíz y hojas:** La raíz contiene las claves principales, y las hojas son los últimos nodos donde se almacenan los valores o punteros.
- **Profundidad:** Mantiene un equilibrio, asegurando que todas las ramas tengan la misma longitud, lo que garantiza un acceso constante en tiempo.



Funcionamiento de las Búsquedas

- **Comparación Secuencial:** Se compara la clave buscada con las claves en el nodo.
- **Recorrido del árbol:** Si la clave no está en el nodo actual, se usa el puntero correspondiente para pasar al siguiente nodo.
- **Acceso a hojas:** Una vez localizada la clave, se llega al nodo hoja que contiene el dato deseado o su ubicación en disco.

ÁRBOLES B EN BASES DE DATOS



Inserción de Datos

- **Espacio disponible en el nodo:** Si el nodo tiene espacio, se inserta la clave en orden.
- **División de nodo (Split):** Si el nodo está lleno, se divide en dos, promoviendo una clave al nodo padre.
- **Actualización del árbol:** Al dividirse los nodos, el árbol sigue balanceado, manteniendo la profundidad uniforme.



Eliminación de Datos

- **Eliminar clave:** Si se encuentra la clave, se elimina y se reorganizan las claves restantes.
- **Fusión de nodos (Merge):** Si la eliminación causa que el nodo tenga pocas claves, puede combinarse con un nodo vecino para mantener el equilibrio.
- **Redistribución:** Las claves pueden redistribuirse entre nodos adyacentes si es necesario, conservando el orden.

2. Infografía ¿Cómo funciona la indexación de bases de datos en forma de árbol B+?

¿CÓMO FUNCIONA LA INDEXACIÓN DE BASES DE DATOS EN FORMA DE ÁRBOL B+?



¿Qué es la Indexación en Árbol B+?

Es una estructura organizada de datos en la base de datos que permite búsquedas rápidas y eficientes.



Estructura del Árbol B+

- **Nodos internos:** Contienen claves para guiar la búsqueda hacia los datos correctos.
- **Nodos hoja:** Almacenan las claves y enlazan directamente a los datos, además de estar conectados secuencialmente.

Cómo Funciona la Búsqueda

La búsqueda comienza en el nodo raíz, sigue hacia los nodos internos y termina en los nodos hoja, donde están los datos.



Ventajas de un Árbol B+ en Bases de Datos

- **Búsqueda rápida:** Optimiza el tiempo de acceso a los datos.
- **Eficiencia en disco:** Minimiza la cantidad de lecturas/escrituras en disco.
- **Consultas de rango:** Permite obtener datos en un rango específico fácilmente.

3. Infografía ¿Cómo implementar una indexación en SQL (puede ser SQLite o SQLServer? Deberá incluir las operaciones para la creación y para el funcionamiento básico CRUE.



¿CÓMO IMPLEMENTAR UNA INDEXACIÓN EN SQL? (SQL SERVER)

¿Qué es una Índice en SQL?
Los índices mejoran la velocidad de búsqueda y consulta en las tablas al organizar los datos.

Creación de un Índice en SQL

```
CREATE INDEX idx_nombre ON tabla (columna);  
CREATE UNIQUE INDEX idx_unico ON tabla (columna);  
CREATE INDEX idx_multi ON tabla (columna1, columna2);
```

Operaciones CRUD con Índices

Insertar
INSERT INTO tabla (columna1, columna2) VALUES (valor1, valor2);

Consultar
SELECT * FROM tabla WHERE columna = valor;

Actualizar
UPDATE tabla SET columna = nuevo_valor WHERE condicion;

Eliminar
DELETE FROM tabla WHERE condicion;

4. Infografía con la opinión del grupo.

¿Las bases de datos utilizan estructuras BTree o BTree++?

¿LAS BASES DE DATOS UTILIZAN ESTRUCTURAS BTREE O BTREE++?

BTree vs B+ Tree

Eficiencia en Búsquedas: B+ Tree es más rápido en consultas de rango.

Estructura: BTree almacena claves en nodos internos y hojas; B+ Tree, solo en hojas.

Espacio y Mantenimiento: B+ Tree es más eficiente en uso de espacio y más fácil de mantener.

Opinión del Grupo

- Preferencia por B+ Tree en bases de datos modernas (SQL Server, MySQL, PostgreSQL).
- **Ventajas:** Optimización en almacenamiento en disco, búsqueda rápida y mejor manejo de grandes volúmenes de datos.

Conclusión

B+ Tree es la elección dominante en bases de datos modernas por su rendimiento superior en consultas y manejo de datos secuenciales.

Sección Práctica:

1. Infografía HASHMAP en JAVA.

La infografía deberá incluir:

- a. ¿Qué es un HashMap en Java ?
- b. ¿Cómo funciona la implementación en Java?
- c. Características del HashMap de Java
- d. Nombre y descripción de las rutinas de la librería

2. Implemente un programa usando la librería de HashMap de Java con las siguientes operaciones:

- a. Crear una tabla Hash
- b. Agregar un elemento
- c. Consultar un elemento
- d. Remover un elemento
- e. Listar elementos

f. Modificar la información de un elemento

Para este ejercicio, puede considerar que el ítem puede ser una persona (llave cédula), carro (llave placa) o curso (llave id del curso). Además, deberá incluir un menú para probar el funcionamiento de la aplicación.