Proyecto 1: Organización de Archivos y control de Concurrencia

Mauricio Pinto, Jonathan Prieto, Francesco Uccelli

Mayo 2020

1 Introducción

1.1 Objetivo

A través de este trabajo se busca analizar algunas tecnicas de organización e indexación de archivos, así como sus algoritmos de acceso y modificación, y también la concurrencia en planes de transacciones. Para esto, se buscó implementar estas técnicas en archivos reales, observar sus operaciones y medirlas. Esto se realizó haciendo uso del lenguaje C++ y de algunas librerías que serán mencionadas posteriormente.

1.2 Definiciones Previas

Comenzaremos definiendo algunos términos importantes

- Organización de archivos : Para los archivos que están almacenados dentro de una Base de Datos, la forma en la cual estos se organizan es clave para tener poder controlarlos y administrarlos. Existen muchos acercamientos para las tecnicas de organizacion de archivos, según algos factores como los tamaños de archivo, la frecuencia de cambios o el tipó de archivo, se busca siempre escoger la técnica más eficiente.
- Hash : Un hash es una función criptografica que se usa(en este contexto) para calcular la ubicacion de un registro en una tabla hash que representa un archivo.
- Índice : Un índice es una estructua que se usa para "marcar" donde se encuentra un registro en un archivo. El uso de índices permite acceder directamente a los registros en un archivo.
- Transacciones : Una transaccion consta de una interaccion sobre un archivo con multiples instrucciones. Estas instrucciones se deben ejecutar de manera indivisible y atomica para mantener la integridad de datos.

1.3 Dominio de Datos y Planteamiento del Problema

Para este proyecto hemos generado manualmente los datos para hacer las pruebas. El problema, en vista general, es poder justificar el uso de las tecnicas de organizacion de archivo esocgidas. De la misma manera, también se quiere comparar el rendimiento (en terminos de tiempo y accesos a disco duro) del Random file y Static hashing que son las tecnicas seleccionadas. Por último, mostrar gráficamente la ejecución de nuestras consultas para que el mini-SGDB sea amigable para cualquier publico.

1.4 Resultados Esperados

Esperamos poder ver en la practica que las tecnicas de organizacion si ayudan a mejorar el rendimiento de las operaciones. Esto se debería ver reflejado en el tiempo que se tarda ejecutar cada funcion de los archivo y en la cantidad de veces que se accede al disco duro.

2 Fundamentación y Descripción de Tecnicas Usadas

2.1 Descripción de las tecnicas de indexación elegidas

2.1.1 Static Hashing

La tecnica de static usa un diccionario donde se guradan la key y el registro. Esto permite acceder al registro simplemente accediendo al diccionario lo que hace que la busqueda de un registro sea muy eficiente. De la misma manera, el insert tambien es eficiente ya que se mantiene el diccionario ordenado. El tradeoff en este caso es el espacio de memoria, ya que como se tiene que usar la estructura auxuliar que contiene las keys, se usa más memoria.

2.1.2 Random file

Los archivos que usan la tecnica de Random file tienen una estructura interna que permite el acceso directo a un registro por su numero. Esto significa que se puede acceder a leer o escribir un registro en mitad del archivo directamente, sin necesidad de recorrer la primera mitad para encontrarlo. Similar a la organización con índice, esto hace que se reduzca significativamente el tiempo de bsuqueda y de insercion pero aumente el costo de memoria.

2.2 Aspectos Importantes de la Implementacion

- **Static Hash:** Para el static hashing se utilizó una estructura bucket

2.3 Manejo de Memoria Secundaria

Para el manejo de memoria secundaria, principalmente, usamos la libreria de C++ fstream, que nos permite escribir y leer archivos en el código. Para la carga de la tabla a la interfaz gráfica utilizamos un archivo en memoria donde se almacena el resultado de la consulta. Finalmente, cada vez que se hace una operacion con el objeto que maneja los rachivos, se llama un destructor que escribe el indice en memoria.

2.4 Simulación de Transacciones

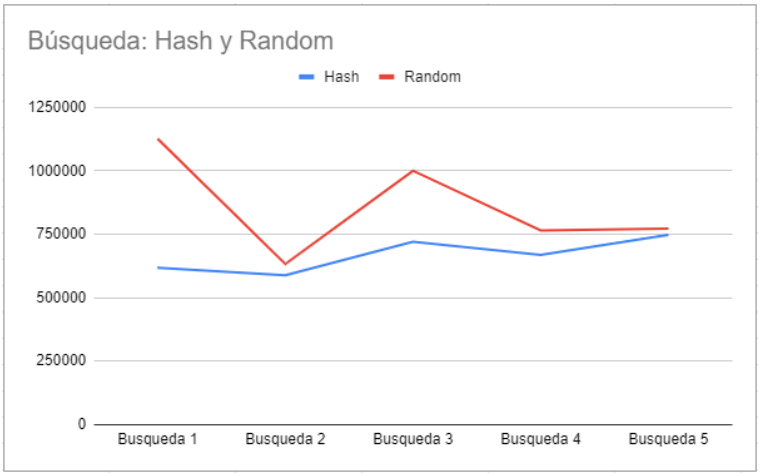
Para simular transacciones se utilizó el concepto de hilos de ejecución. Cada hilo ejecutó una transacción, es decir, una serie de operaciones. Cuando estas hacían cambios sobre el recurso compartido, se buscó hacer un bloqueo exclusivo de este.
Para implementar esto se utilizó la librería de POSIX ¡pthread.h¿ y la librería ¡mutex¿. Se simuló la ejecución de dos transacciones en paralelo, usando un hilo para cada una, y llamando a una función que se encargaría de ejecutar las operaciones correspondientes a cada una. Estas operaciones fueron leídas de un archivo de texto y adaptadas a la estructura *transaction*, la cual se define como:

```
struct transaction {
        int t_id;
        int index;
        const char *data_filename;
        const char *index_filename;
        int n_buckets;
        int bucket_size;
        operation op;
        record data;
};
```

Cuando una transacción realiza una operación de write, se hacía un bloqueo a esta con mutex mientras la realizaba, para mantener la integridad de los datos. (La prueba de las transacciones se encuentra en el anexo pruebas de uso)

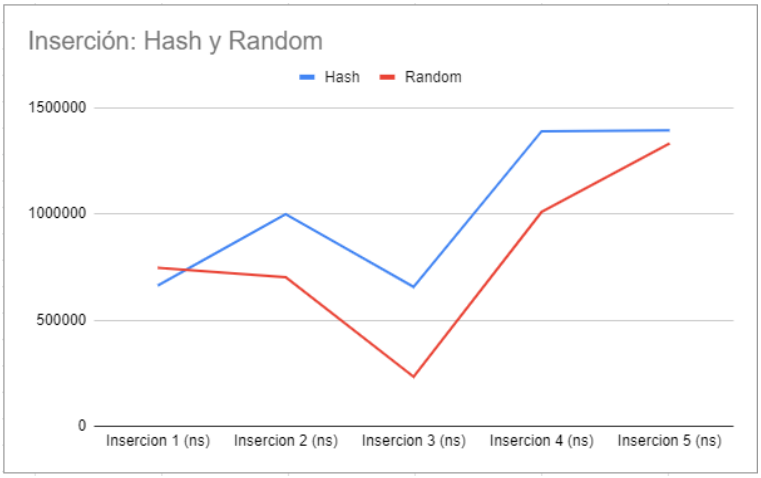
2.5 Resultados de experimentación

2.5.1 Busqueda



Como se observa en el gráfico, para un caso promedio, el random hace la busqueda más eficiente, esto se debe a que ese es su propósito principal y está diseñada para eso.

2.5.2 Inserción

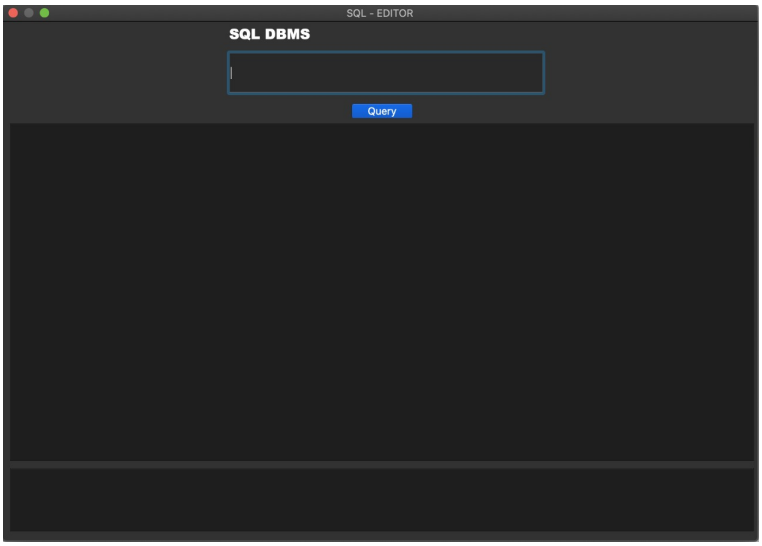


En el caso de la inserción se ve que es en promedio más rápida en el hash, esto se debe a que la canctidad de registros con la que probamos es relativemente pequeña, por lo que agregar los componentes de la tabla hash no aumentan el costo mucho

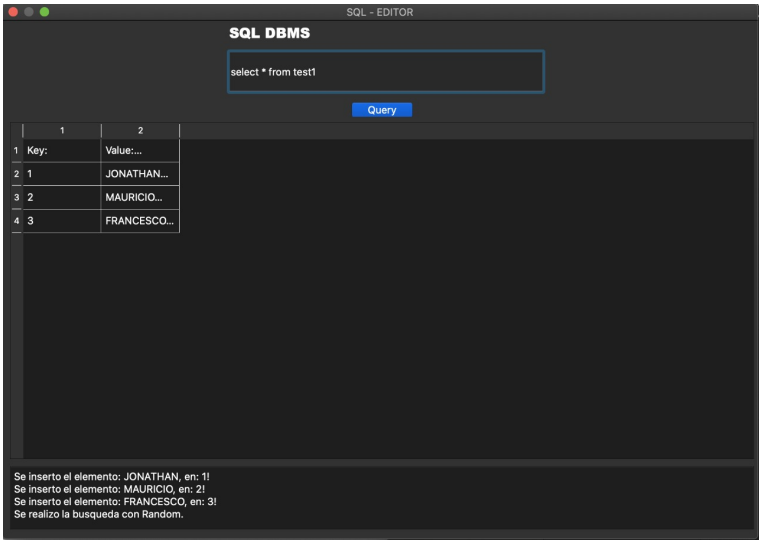
2.6 Pruebas de uso y aplicación

2.6.1 Aplicación

Cuando se ejecuta el programa, primero se muestra la pantalla que recibe un query en texto plano y un boton para mandar el query.

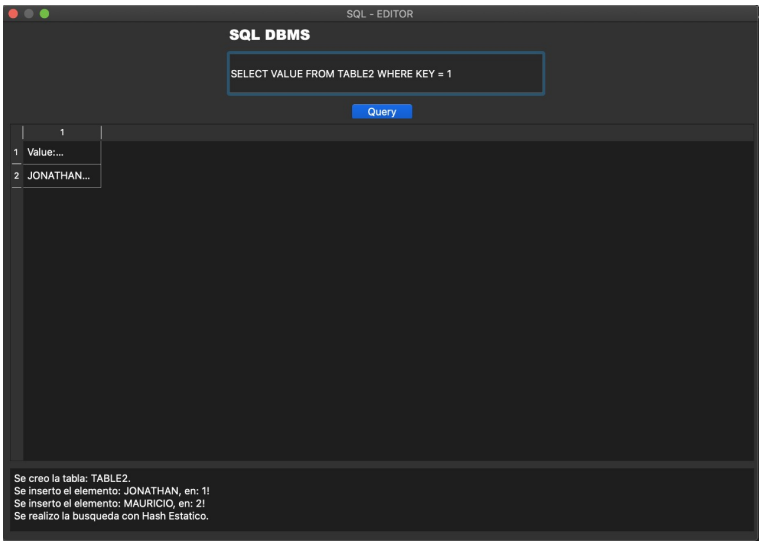


Luego de ejecutar la consulta, se muestra la tabla resultante dentro de una ventana. Además, se muestra en un cuadro de texto el output correspondiente a la consola, en este caso, que se han insertado los registros y se ha ejecutado la busqueda.

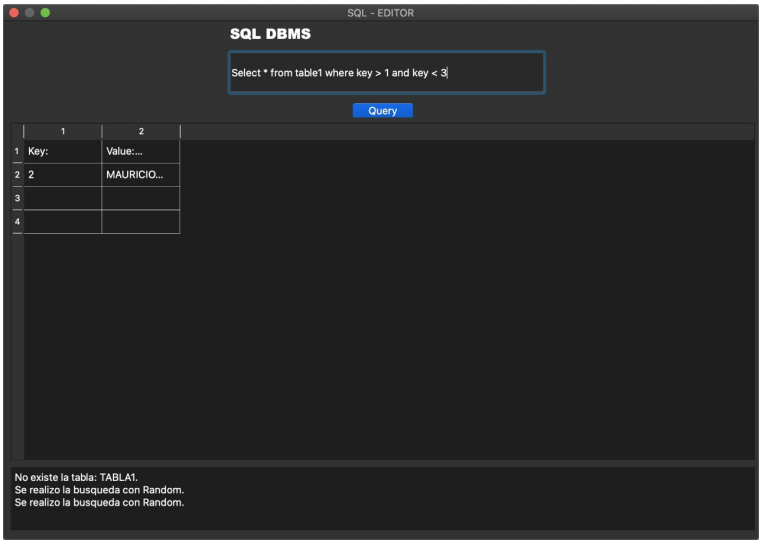


2.6.2 Pruebas de uso

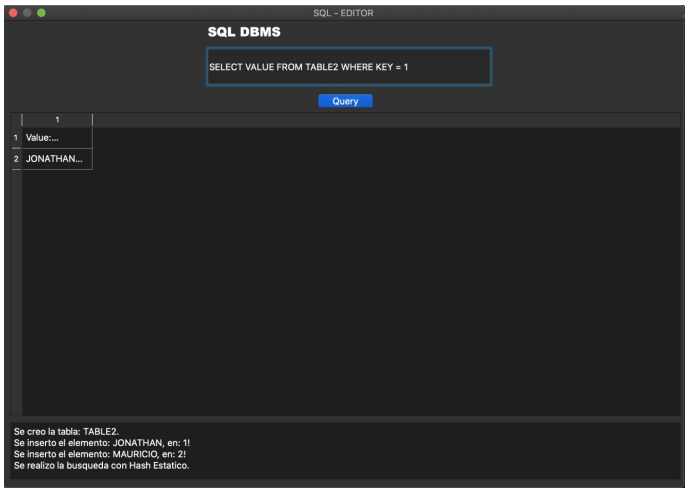
En el anexo de aplicación definido anteriormente, se ve un ejemplo de prueba de usa para un un select de toda la tabla usando random file. En esta parte se muestran un select con hash y un atributo especifico.



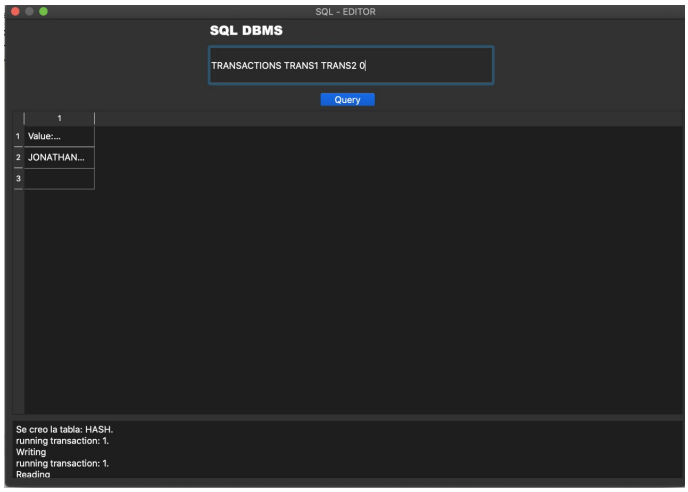
Busqueda por rango



- Transacciones en Random file



- Transacciones en Hash



2.7 Conclusión

Para concluir, se puedieron implementar ambas tecnicas de organización de archivos, así como la interfaz grafica para el mini SGBD y las transacciones. Esto nos permitió hacer la experimentación y obtener resultados explicables y lógicos.

En temrinos de las complicaciones que tuvimos, la principal fue la integracion de la interfaz desarrollada en QT (un sistema completamente nuevo para nosotros) y los codigos de manejo de archivos. Además, no se pudieron pasar parametros del codigo en c++ a qt, lo que nos forzó a usar archivos .Por otro,

no se pudieron almacenar direcciones de punteros en los indices ya que cada vez que se leen pueden tener direcciones que no corresponden.

2.8 Anexos

Solo utlizamos la libreria de QT para la interfaz grafica.
Link del proyecto en github:
`https://github.com/mauriciopinto/BD2_Proyecto_1`