Informe de Prácticas de Fundamentos de Bases de Datos

Práctica 3

Rodríguez Marrero, David
Larriba Moreno, Sergio

Índice

Introducción

La práctica se va a centrar en cómo el ordenador lee y entiende las tablas que almacenamos en una base de datos. Esta práctica se va a centrar en tablas, y cómo se gestionan con un fichero de datos en el que se guarda la información y otro de índices que ayuda a buscar una información en concreto según un árbol binario implementado.

Por otra parte, también vamos a hacer uso de las funciones fwrite, fread y fseek que nos ayudaran a gestionar las búsquedas e inserciones en los archivos de datos e índices.

Han incluido en la práctica un zip con scripts para resolver y comprobar el resultado del menú que hemos implementado. Gracias a esos scripts podremos ver qué sentencias tenemos que imprimir en consola para un buen entendimiento del programa.

Al final de la práctica, entregaremos todos los códigos de las consultas y el Makefile que hemos utilizado prácticamente igual al dado en Moodle. También entregaremos este informe en el cual hemos redactado nuestros pensamientos durante toda la práctica.

CreateTable

```
bool createTable(const char * tableName)
{
    FILE *f=NULL;
    char *indexName=NULL;

    /* Abro el fichero */
    f = fopen(tableName, "rb+");
    /* Si el fichero no existe, lo creo */
    if (f == NULL)
    {
        f = fopen(tableName, "wb+");
        if (f == NULL)
            return false;

        /* Escribo -1, que indica que no hay ningun fichero borrado */
        fwrite(&no_deleted_registers, sizeof(int), 1, f);
    }
    /* Reservo memoria para el nombre del indice */
    indexName = (char*) calloc (strlen(tableName), sizeof(indexName[0]) + 1);
    if (indexName == NULL)
        return false;

    /* Cambio el nombre a .idx */
    replaceExtensionByIdx (tableName, indexName);
    if (createIndex (indexName) == false)
        return false;

    free(indexName);
    fclose (f);

    return true;
}
```

```
void replaceExtensionByIdx(const char *fileName, char * indexName) {
    size_t l;
    l = strlen(fileName);
    strncat(indexName, fileName, l - 3);
    strcat(indexName, "idx");
}
```

Esta función abre el fichero con nombre tableName que se pasa por argumento y si no lo puede abrir lo crea e inicializa la cabecera con un -1 que indica que el listado de registros borrados está vacío.

Después de esto, creamos el archivo de índices con mismo nombre que el archivo anterior pero con formato .idx que cambiamos con la función implementada replaceExtensionByIdx.

Luego, llamamos a createIndex, liberamos memoria y retornamos si ha salido todo bien.

CreateIndex

```
bool createIndex(const char *indexName)
{
    FILE *f=NULL;

    /* Abro el fichero de indices */
    f = fopen(indexName, "rb+");
    /* Si el fichero no existe */
    if (f == NULL)
    {
        f = fopen(indexName, "wb+");
        if (f == NULL)
            return false;

        /* Asigno -1 a los 2 primeros punteros iniciales */
        fwrite(&no_deleted_registers, sizeof(int), 1, f);
        fwrite(&no_deleted_registers, sizeof(int), 1, f);
    }

    fclose (f);
    return true;
}
```

Esta función empieza creando el fichero de índices si no existe y escribe dos -1 en la cabecera los cuales indican la raíz del árbol y el nodo borrado.

PrintTree

```c
void printnode(size_t _level, size_t level, FILE * indexFileHandler, int node_id, char side)
{
    Node node;
    size_t pos, i;

    if (node_id == -1 || indexFileHandler == NULL || _level>level)
        return;

    pos = INDEX_HEADER_SIZE + sizeof (node) * node_id;

    fseek(indexFileHandler, pos, SEEK_SET);
    fread(&node, sizeof(Node), 1, indexFileHandler);

    for (i=0; i< _level; i++)
        printf("\t");

    printf("%c %.4s (%d): %d\n", side, node.book_id, node_id, node.offset);

    _level++;
    printnode(_level, level, indexFileHandler, node.left, 'l');
    printnode(_level, level, indexFileHandler, node.right, 'r');

    return;
}

void printTree(size_t level, const char * indexName)
{
    int root;
    FILE *f = NULL;

    if (indexName == NULL)
        return;

    f = fopen(indexName, "r");

    fread(&root, sizeof(int), 1, f);
    if (root == -1)
        return;

    printnode(0, level, f, root, ' ');

    fclose (f);

    return;
}
```

Nuestra función printTree mira si el fichero está vacío, es decir que la raíz sea -1 y si no, llama a la función recursiva printnode la cual lee los nodos del archivo de índices y los va imprimiendo recursivamente teniendo en cuenta su profundidad hasta encontrar un -1 en node_id que significará que no existe ese nodo.

FindKey

```
bool findKey(const char * book_id, const char *indexName, int * nodeIDOrDataOffset)
{
    FILE *f=NULL;
    Node node;
    int root=-1, cmp;
    size_t pos;
    bool condition=true;

    if (book_id == NULL || indexName == NULL || nodeIDOrDataOffset == NULL)
        return false;

    f = fopen (indexName, "rb+");
    if (f == NULL)
        return false;
    /* Leo y guardo la posicion del nodo raiz en root */
    fread(&root, sizeof(int), 1, f);
    *nodeIDOrDataOffset = root;
    while (condition == true) /* M*/
    {
        /* Busco el nodo que ocupa la posicion pos y lo guardo en node */
        pos = INDEX_HEADER_SIZE + sizeof (Node) * (*nodeIDOrDataOffset);
        fseek (f, pos, SEEK_SET);
        fread (&node, sizeof(Node), 1, f);

        /* si los bytes son iguales, he encontrado la llave */
        cmp = memcmp(book_id, node.book_id ,PK_SIZE);
        if (cmp == 0)
        {
            *nodeIDOrDataOffset = node.offset;
            fclose (f);
            return true;
        }
        else if (cmp > 0) /* Quiere decir que el nodo que buscamos esta por la parte derecha, ya que es mayor que el actual */
        {
            if (node.right == -1) /* Si no existe el nodo derecho al actual, no hemos encontrado la llave */
            {
                fclose (f);
                return false;
            }
            else
                *nodeIDOrDataOffset = node.right;
        }
        else /* cmp <0 -> quiere decir que el nodo que buscamos esta a la izquierda, ya que es menor que el nodo actual */
        {
            if (node.left == -1) /* Si no existe el nodo izquierdo al actual, no hemos encontrado la llave */
            {
                fclose (f);
                return false;
            }
            else
                *nodeIDOrDataOffset = node.right;
        }
        else /* cmp <0 -> quiere decir que el nodo que buscamos esta a la izquierda, ya que es menor que el nodo actual */
        {
            if (node.left == -1) /* Si no existe el nodo izquierdo al actual, no hemos encontrado la llave */
            {
                fclose (f);
                return false;
            }
            else
                *nodeIDOrDataOffset = node.left;
        }
        condition = true;
    }
    return true;
}
```

En esta función buscamos un book_id y devolveremos el offset de la posición en la que se encuentra mediante argumentos. Primero abrimos el archivo de índices e inicializo ese offset de salida al de la raíz. Después busco el nodo que ocupa la posición pos que va a ir cambiando si no lo encontramos y lo guardo en node y mediante comparación vamos recorriendo todo el árbol binario según si los bytes son mayores o menores. Si llegamos a un valor -1, es que el nodo buscado no existe, si lo encontramos, guardamos el offset en la variable del argumento y retornamos.

AddIndexEntry

```
bool addIndexEntry(char * book_id,  int bookOffset, char const * indexName) {

    FILE *f=NULL;
    Node node;
    int ret, deleted_reg=-1, nuevo_nodo=-1, res=-1, nodeIDOrDataOffset;
    struct stat st;
    size_t INDEX_REGISTER_SIZE = sizeof(Node);

    if (book_id == NULL || indexName == NULL)
        return false;

    /* Si existe la clave retorno false */
    res = findKey(book_id, indexName, &nodeIDOrDataOffset);
    if (res)
        return false;

    f = fopen(indexName, "rb+");
    if (f == NULL)
        return false;

    /* Miro a ver si hay registros borrados */
    ret = fread(&deleted_reg, sizeof(int), 1, f);
    if (ret == 0)
        return false;
    ret = fread(&deleted_reg, sizeof(int), 1, f);
    if (ret == 0)
        return false;

    /* Si no hay registros borrados */
    if (deleted_reg == -1)
    {
        ret = fseek(f, 0, SEEK_END);
        if (ret != 0)
            return false;
        stat(indexName, &st);
        nuevo_nodo = ((st.st_size - INDEX_HEADER_SIZE)/INDEX_REGISTER_SIZE);
        fprintf (stderr, "Nodo nuevo: %d", nuevo_nodo);
    }
    /* Si hay registros borrados */
    else
    {
        /* Leo el nodo borrado */
        ret = fseek(f,deleted_reg * INDEX_REGISTER_SIZE + INDEX_HEADER_SIZE, SEEK_SET);
        if (ret != 0)
            return false;
        ret = fread(&node, INDEX_REGISTER_SIZE, 1, f);
        if (ret == 0)
            return false;
        nuevo_nodo = deleted_reg;
        deleted_reg = node.left;
        /* Actualizo el nodo de la cabecera */
        ret = fseek(f, sizeof(int), SEEK_SET);
        if (ret != 0)
            return false;
        ret = fwrite(&deleted_reg, sizeof(int), 1, f);
        if (ret == 0)
            return false;
        /* Muevo el puntero */
        ret = fseek(f, nuevo_nodo * INDEX_REGISTER_SIZE + INDEX_HEADER_SIZE, SEEK_SET);
        if (ret != 0)
            return false;
    }

    memcpy(node.book_id, book_id, 4);
    node.offset = bookOffset;
    node.right = -1;
    node.left = -1;
    node.parent = nodeIDOrDataOffset;
    ret = fwrite(&node, INDEX_REGISTER_SIZE, 1, f);
    /* Guardo y actualizo el padre */
    ret = fseek(f, nodeIDOrDataOffset * INDEX_REGISTER_SIZE + INDEX_HEADER_SIZE, SEEK_SET);
    if (ret != 0)
        return false;
    ret = fread(&node, INDEX_REGISTER_SIZE, 1, f);

    /* Lo inserto */
    res = memcmp(book_id, node.book_id, 4);
    if (res > 0)
        node.right = nuevo_nodo;
    else
        node.left = nuevo_nodo;

    ret = fseek(f, nodeIDOrDataOffset * INDEX_REGISTER_SIZE + INDEX_HEADER_SIZE, SEEK_SET);
    if (ret != 0)
        return false;
    ret = fwrite(&node, INDEX_REGISTER_SIZE, 1, f);
    if (ret == 0)
        return false;

    ret = fclose (f);

    return true;
}
```

En esta función queremos introducir una clave por lo que primero miramos a ver si ya existe, si no, abrimos el archivo de índices y miro si hay registros borrados, si no hay añado el nuevo nodo, si hay, leo el nodo borrado y actualizo el nodo de la cabecera que indica que hay nodos borrados.

Luego, inicializamos el nuevo nodo sin hijos, buscamos el padre por el offset dado e inserto el nodo respetando el árbol binario.

AddTableEntry

```
bool addTableEntry(Book * book, const char * dataName,const char * indexName) {
    int nodeoffset,offset;
    size_t len = strlen(book->title);
    FILE *f=NULL;

    f = fopen(dataName, "rb");
    if(!f)
    {
        return false;
    }
    if(fread(&nodeoffset, sizeof(int), 1, f)==0){
        return false;
    }
    if(findKey(book->book_id, indexName, &nodeoffset)==true)
    {
        printf("LA CLAVE EXISTE\n");
        return false;
    }

    if(nodeoffset == -1)
    {
        if(fseek(f, SEEK_END,(int)SEEK_SET)==0){
            return false;
        }
        if(fwrite(book->book_id,PK_SIZE, 1, f)==0){
            return false;
        }
        if(fwrite(&len, sizeof(int), 1, f)==0){
            return false;
        }
        if(fwrite(book->title, len, 1, f)==0){
            return false;
        }
        offset = (int)ftell(f);
    }
    if(addIndexEntry(book->book_id, offset, indexName)==false)
    {
        printf("ERROR INTRODUCIENDO EL ÍNDICE\n");
        return false;
    }
    return true;
}
```

Para insertar un nuevo dato, primero miro a ver si ya existe ese dato, si no, miro si existen registros borrados, si no existen, escribimos al final del archivo la información y después llamamos a la función anterior para editar el fichero de índices.

Interfaz de Usuario

Hemos creado un menú casi idéntico estructuralmente al de la práctica 2 en el que el usuario tiene que elegir use, insert o print para hacer distintas cosas en torno a la creación de tablas inserción en la tabla o impresión del árbol de índices.

- Use: El sistema preguntará por el nombre de la tabla a utilizar con el formato correcto y llamara a la función createTable.

- Insert: El sistema preguntará por la clave y el título para almacenar y llamará a la función addTableEntry para introducirla en los dos ficheros.

- Print: Se muestra el árbol binario de búsqueda por pantalla. Se debe mostrar un mensaje de error si no se ha seleccionado una tabla antes usando use.

Para usar los comandos Insert y Print se debe pasar primero por Use ya que si no no habrá ninguna tabla creada.

Nuestro menú también dispone de una cuarta entrada posible que es el exit la cual sale del programa.

Gracias a este programa implementado, podemos hacer uso de nuestras funciones más fácilmente que si tuviésemos que hacer un main como el tester pero llamando a todas las funciones con unos valores pedidos por pantalla. Ayuda bastante repartirlo en un menú en el que puedes ver la información que quieres, ni más ni menos.

Conclusiones

En esta práctica hemos aprendido cómo el ordenador lee las tablas que llevamos usando durante las dos últimas prácticas y como a través de bytes es posible trasmitir, guardar o editar una información concreta.

Por otra parte, también hemos entendido la dificultad que tienen los programas que hacen estas comunicaciones con el ordenador directamente con mucha más información de la que nosotros tenemos que gestionar en nuestra práctica.

Hemos adquirido nuevos conocimientos del lenguaje C usando funciones como el fwrite, fread y fseek que son fundamentales en el uso de archivos. También hemos entendido las diferencias entre un archivo de información y otro de índices el cual gestiona un árbol binario por lo que estamos tratando con estructuras de datos vistas y aprendidas anteriormente que nos han ayudado a comprender mejor la práctica.

También hemos aprovechado lo aprendido en la práctica 2 ya que nos ha facilitado el menú y hemos entendido que con un programa es mucho más fácil comunicarse con las funciones ya que a lo largo de la práctica nos hemos perdido a la hora de ejecutar los test individualmente para ver si nuestras funciones funcionaban o no.

Finalmente, todas estas prácticas nos han introducido por completo al mundo de la gestión de bases de datos desde lo más externo como es el lenguaje SQL hasta lo más interno como es comprender qué lee el ordenador y como lo gestiona.