Universidad Nacional de San Agustín

Escuela Profesional de Ciencia de la Computación

Investigacion Formativa

Sistema Gestor de Base de Datos - Grupo F

Alumnos :
Alejandro Antonio Villa Herrera
Fiorella Estefany Villarroel Ramos
Juan Manuel Soto Begazo
Merisabel Ruelas Quenaya
Docente:
Leydi Beatriz Manrique Tejada
Monica Milagros Jordan Franco

20 de julio de 2022

Índice

1. Introducción

La importancia de recolectar, evaluar y administrar los datos ha aumentado significativamente con el tiempo. El concepto de datos está en constante evolución y transformación en el mundo.

Un gestor de bases de datos, desempeña un papel importante a la hora de importar o crear una base de datos propia. El SGBD proporciona a los usuarios y programadores una forma sistemática de crear, recuperar, actualizar y gestionar datos. Un sistema de gestión de bases de datos (SGBD) es una aplicación informática que interactúa con los usuarios finales, otras aplicaciones y la propia base de datos para capturar y analizar los datos. Un SGBD de propósito general permite definir, crear, consultar, actualizar y administrar bases de datos.

Es por ello que haciendo uso de los conocimientos adquiridos durante el curso de Base de Datos I y Base de Datos II se creará un Sistema Gestor de Bases de Datos.

1.1. Objetivos

1.1.1. Objetivo General

- El objetivo principal de un SGBD es proporcionar un entorno cómodo y eficaz para que los usuarios recuperen y almacenen información.

1.1.2. Objetivos específicos

- Eliminar los datos redundantes.
- Facilitar el acceso a los datos al usuario.
- Prever el almacenamiento de los datos.
- Proteger los datos de daños físicos y de sistemas no autorizados.
- Permitir el crecimiento del sistema de base de datos.
- Hacer que las últimas modificaciones de la base de datos estén disponibles inmediatamente.
- Proporcionar una respuesta rápida a las solicitudes del usuario.

2. Estado del arte

2.1. B+ Tree

El árbol B es una estructura de datos que proporciona datos ordenados y permite realizar búsquedas, accesos secuenciales, anexos y eliminaciones en orden. El árbol B tiene una gran capacidad para los sistemas de almacenamiento que escriben grandes bloques de datos. El árbol B simplifica el árbol de búsqueda binario al permitir nodos con más de dos hijos.

El árbol B almacena los datos de forma que cada nodo contiene claves en orden ascendente. Cada una de estas claves tiene dos referencias a otros dos nodos hijos. Las claves del nodo hijo de la izquierda son menores que las claves actuales y las claves del nodo hijo de la derecha son mayores que las claves actuales. Si un nodo tiene un número "n"de claves, puede tener un máximo de "n+1"nodos hijos.

2.1.1. Ventajas Principales del B Tree

- Nodos intermedios y hoja ordenados: dado que es un árbol equilibrado, todos los nodos deben estar ordenados.
- Recorrido rápido y búsqueda rápida: Uno debería poder atravesar los nodos muy rápido. Eso significa que, si tenemos que buscar algún registro en particular, deberíamos poder pasar por el nodo intermediario muy fácilmente. Esto se logra clasificando los punteros en los nodos intermedios y los registros en los nodos hoja. Cualquier registro debe recuperarse muy rápidamente. Esto se hace manteniendo el equilibrio en el árbol y manteniendo todos los nodos a la misma distancia.
- Sin páginas de desbordamiento: el árbol B+ permite que todos los nodos intermedios y de hoja se llenen parcialmente; tendrá algún porcentaje definido al diseñar un árbol B+. Este porcentaje hasta el cual se llenan los nodos se llama factor de relleno. Si un nodo alcanza el límite del factor de relleno, se denomina página de desbordamiento. Si un nodo está demasiado vacío, se denomina subdesbordamiento. En nuestro ejemplo anterior, el nodo intermediario con 108 está subdesbordado. Y los nodos de hoja no están parcialmente llenos, por lo tanto, es un desbordamiento. En el árbol B+ ideal, no debería tener desbordamiento ni subdesbordamiento excepto el nodo raíz.

2.2. Estructura

2.2.1. Estructura de almacenamiento

2.2.1.1. Estructura física

Control files Es el esqueleto de la base de datos y guarda la información necesaria para levantar la base de datos. Es un archivo de control es un pequeño archivo binario que registra la estructura física de la base de datos e incluye:

- El nombre de la base de datos
- Nombres y ubicaciones de archivos de datos asociados y archivos de registro de rehacer en línea.
- La marca de tiempo de la creación de la base de datos.
- El número de secuencia de registro actual
- Información del punto de control
- El archivo de control debe estar disponible para que lo escriba el servidor de la base de datos de Oracle siempre que la base de datos esté abierta. Sin el archivo de control, la base de datos no se puede montar y la recuperación es difícil.

El archivo de control de una base de datos de forma predeterminada, debe crease con al menos una copia del archivo de control durante la creación de la base de datos. En algunos sistemas operativos, el valor predeterminado es crear varias copias. Debe crear dos o más copias del archivo de control durante la creación de la base de datos. Es posible que también deba crear archivos de control más adelante, si pierde los archivos de control o desea cambiar configuraciones particulares en los archivos de control. Debe estar multiplexado, esto significa que almenos debemos tener dos copias de Control File cada una en discos diferentes de modo que si se dá algún malfuncionamiento y daña un disco, podemos restaruar la información a partir de la segunda copia.

Data files Sirve para el almacenamiento físico del Diccionario de Datos, las tablas, índices, procedimientos y la imagen anterior de los bloques de datos que se han modificado en las transacciones (segmentos de rollback). Estos archivos son los únicos que contienen los datos de los usuarios de la base de datos. Se pueden tener sólo uno o cientos de ellos.

El número máximo de datafiles que pueden ser configurados está limitado por el parámetro de sistema MAXDATAFILES.

Si se decide que utilice varios datafiles, el administrador del sistema puede gestionar que éstos queden localizados en discos diferentes, lo que aumentará el rendimiento del sistema, principalmente por la mejora en la distribución de la carga de entrada / salida. Para obtener información de los Archivos de Datos, consultar la tabla DBA_DATA_FILES.

Redolog files

- Tienen los cambios que se han hecho a la base de datos para recuperar fallas o para manejar transacciones.
- Debe estar conformado por dos grupos como mínimo y cada grupo debe estar en discos separados.
- El principal propósito de estos archivos es de servir de respaldo de los datos en la memoria RAM.
- El proceso LGWR de lo que se encarga es de escribir de la estructura de memoria (Redo) Log Buffer de las Online Redo Logs y muy importante es saber cuales son las circunstancias que hacen que el LGWR escriba al Online Redo Log cuando el usuario hace un commit a la transacción.
- Cuando sucede un cambio (log switch) del archivo de Redo Log
- Cuando han pasado tres segundos , desde la ultima escritura del LGER hacia el Online Redo Log.
- Cuando el Redo Log Buffer esta 1/3 lleno o contiene mas de 1Mb de datos en buffer
- Un aspecto a tener en cuenta es el tamaño de los ficheros redo log. Si son muy pequeños, el LGWR deberá cambiar de ficheros demasiado frecuentemente, lo que reduce su rendimiento. Por otro lado, si los ficheros redo log son demasiado grandes, se necesitará mucho tiempo en las recuperaciones, ya que se tendrán que recuperar muchas transacciones.
- Otro aspecto muy importante es la elección del número correcto de grupos, ya que disponer de demasiados pocos grupos puede acarrear problemas cuando estámos en modos ARCHIVELOG y tenemos una tasa de transacciones muy alta.

- Esto puede suponer que un grupo que todavía está archivando por el proceso ARCH se convierta en el grupo en el que el LGWR necesite escribir, lo que produciría que la BD se parara, ya que el LGWR tienen que esperar a que el grupo esté disponible, una vez que su contenido ha sido archivado.

2.2.1.2. Estructura lógica

Las unidades de asignación de espacio en la base de datos son los bloques de datos, extents y segmentos. El nivel más fino de granularidad en que Oracle almacena los datos es el bloque de datos (también llamados bloques lógicos, bloques de Oracle o páginas). Un bloque de datos corresponde a un número de bytes físicos de espacio de la base de datos en disco.

Un segmento es un conjunto de extents, cada uno de los cuales ha sido asignado a una estructura de datos específicas, pertenecientes a un tablespace. Por ejemplo, cada tabla de datos es almacenada en su propio segmento de datos, mientras que cada índice de datos es almacenado en su propio segmento de índices. Si la tabla o índice es particionado, cada partición es almacenada en su propio segmento

- **Bloque de Datos**

  Se maneja el espacio de almacenamiento en datafiles de una base de datos en unidades llamadas bloques de datos. Un bloque de datos es la unidad más pequeña de datos usada por una base de datos.

  Se solicita los datos en múltiplos de los bloques de datos y no del sistema operativo. El tamaño estándar del bloque se puede conocer consultando el parámetro DB_BLOCK_SIZE. Los tamaños de los bloques de datos deben ser un múltiplo de los tamaños de bloques del sistema operativo, entonado para evitar I/Os innecesarios.

  Formato del Bloque de datos

  - **Encabezado**

    El encabezado contiene información general del bloque, como la dirección del bloque y el tipo de segmento (datos, índices).

  - **Directorio de tablas**

    Contiene información acerca de las tablas que contienen tuplas en el bloque.

  - **Directorio de Registros**

Contiene información de las tuplas actuales en el bloque (incluyendo direcciones para cada row piece "trozo de registro" en el área de datos). Después que el espacio ha sido asignado en el directorio de registros de un bloque de datos, éste no es reclamado cuando el registro es borrado. Se reutiliza el espacio sólo cuando se insertan nuevos registros en el bloque.

- **Overhead**

  El encabezado del bloque de datos, directorio de tablas y directorio de registros son denominados de forma colectiva como overhead.

- **Datos de los registros**

  Contiene la tabla o datos de índices. Los registros puede atravesar bloques

- **Espacio libre**

  El espacio libre es asignado para la inserción de nuevos registros y para la actualización de registros que requieran espacio adicional. En los bloques de datos asignados para segmentos de datos de una tabla o clúster o para los segmentos de un índice, el espacio libre puede almacenar registros de transacciones. Un registro de una transacción es requerido un un bloque para cada instrucción de INSERT, UPDATE, DELETE y SELECT FOR UDATE que accede una o más registros en el bloques En la mayoría de los SOs un registro de una transacción requiere 23 bytes.

- **Tablespace**

Una base de datos consiste de una o más unidades de almacenamiento lógico llamados tablespaces, que almacenan colectivamente todos los datos de la base de datos. Cada tablespace consiste de uno o más archivos llamados datafiles, que son estructuras físicas del sistema operativo. Los datos de una base de datos son almacenados colectivamente en datafiles que constituyen cada tablespace de la base de datos. Internamente se utilizan interfaces estándar de file system para crear y borrar archivos según lo necesite para las siguientes estructuras de la base de datos:

- Tablespaces
- Online redo lo files
- Control files

Tablespaces Dictionary Managed

Para un tablespaces que maneja sus extents mediante el uso del diccionario de datos, se actualiza las tablas apropiadas en el diccionario de datos siempre que un extent es asignado o liberado. Además almacena información de rollback por cada actualización en las tablas del diccionario de datos. Como las tablas del diccionario de datos y los segmentos de rollback son parte de la base de datos, el espacio que ellos ocupan esta sujeto a las mismas operaciones de manejo de espacio de los otros datos.

- **Extent**

  Un extent es una unidad lógica de asignación de espacio de almacenamiento en una base de datos hecha de un número continuo de bloques de datos. Uno o más extents hacen un segmento. Cuando el espacio existente en el segmento está completamente usado, se asigna un nuevo extent para el segmento.

  Cuando se crea una tabla, se asigna el segmento de datos de la tabla con un extent inicial que se corresponde con un número de bloques de datos. Si los bloques de datos del extent inicial del segmento llegan a estar llenos y se requiere más espacio, se asigna un extent incremental para el segmento. Un extent incremental es un extent subsiguiente del mismo tamaño que el anterior o más grande. Por propósitos de mantenimiento, el encabezado del bloque de cada segmento contiene un directorio de los extents en el segmento.

- **Segmentos**

  Un segmento es conjunto de extents que contiene todos los datos de una estructura de almacenamiento lógica (como puede ser una tabla) en un tablespace. Por ejemplo para cada tabla, se asigna uno o más extents para forma un segmento de datos de la tabla. Comúnmente se utiliza 4 tipos de segmentos:

  - Segmentos de Datos
  - Segmentos de Índices
  - Segmentos Temporales
  - Segmentos de Ordenamiento

3. Implementación

3.1. Buffer Manager

3.1.1. Estructura de Buffer Manager

El administrador de búfer BadgerDB utiliza tres clases de C++: BufMgr, BufFramey BufHashTbl. Solo hay una instancia de la clase BufMgr. Un componente clave de esta clase es el grupo de búfer real que consta de una matriz de el numero de buffers , cada uno del tamaño de una página de base de datos.

Además de esta matriz, la instancia de BufManagertambién contiene una matriz de instancias numBufs de la clase BufFrameque se utiliza para describir el estado de cada fotograma en el grupo de búfer. Se utiliza una tabla hash para realizar un seguimiento de las páginas que residen actualmente en el grupo de búfer.

Esta tabla hash se implementa mediante una instancia de la clase BufHashTbl. Esta instancia es un miembro de datos privados de la clase BufMgr. Estas clases se describen en detalle a continuación.

3.1.2. Código de Buffer Manager

3.1.2.1. BufHashTbl Class

La clase BufHashTbl se usa para asignar archivos y números de página a grupos de búfer y se implementa mediante usando çhained bucket hashing". Hemos proporcionado una implementación de esta clase para su uso.

```
1 struct        hashBucket        {
2         File*        file;          //        puntero a un objeto
          ↪  archivo
3         PageId        pageNo;          //        número de página
          ↪  dentro de un archivo
4         FrameId        frameNo;          //        numero de frame de
          ↪  la página en el grupo de búfer
5         hashBucket*        next;          //        siguiente bucket
          ↪  en la cadena
6 };
```

3.1.2.2. BufDesc Class

La clase BufFramese utiliza para mantener el estado de cada frame en el pool de buffers. Se define como sigue: En primer lugar, observa que todos los atributos de la clase BufFrameson privados y que la clase BufManagerestá definida como amiga. Aunque esto puede parecer extraño, este enfoque restringe el acceso a las variables privadas de BufFramesólo a la clase BufMgr. La alternativa (hacer todo público) abre demasiado el acceso. El propósito de la mayoría de los atributos de la clase BufFramedebería ser bastante obvio. El bit dirty, si es verdadero, indica que la página está sucia. indica que la página está sucia (es decir, ha sido actualizada) y por lo tanto debe ser escrita en el disco antes de que el frames sea para contener otra página. El $\text{num}_pinindicacuántasveceshasidofijadalapágina.Elrefbitesutilizadoporelalgoritmodelreloj.Elb$

```
1  class BufDesc{
2            friend class BufMgr;
3            private:
4              File * file;
5            PageId pageNo;
6            FrameId frameNo;
7            int pinCnt;
8            bool dirty;
9            bool valid;
10           bool refbit;
11           void Clear();
12           void Set(File * filePtr, PageId pageNum);
13           void Print();
14           BufDesc();
```

3.1.3. BufManager Class

La clase BufManager es el corazón del gestor de buffers. Aquí es donde se escribe el código para esta tarea.

3.1.3.1. BufManager(const int bufs)

Este es el constructor de la clase. Asigna un array para el pool de buffers con los frames de la página bufs y una tabla BufFrame correspondiente. Tal y como están configuradas las cosas, todos los frames estarán en estado claro cuando se asigne

el pool de buffers. La tabla hash también comenzará en un estado vacío. Hemos proporcionado el constructor.

3.1.3.2. BufManager()

Vacía todas las páginas corruptas y desocupa la reserva de búferes y la tabla BufDesc.

3.1.3.3. void advanceClock()

Avanza el reloj al siguiente frames en el pool de buffers.

3.1.3.4. void allocBuf(FrameId frame)

Asigna un frames libre usando el algoritmo del reloj; si es necesario, escribe una página corrupta de vuelta al disco. Lanza una BufferExceededException si todos los frames del buffer están fijados. Este método privado será llamado por los métodos readPage() y allocPage() descritos a continuación.

3.1.3.5. void readPage(File* file, const PageId PageNo, Page* page)

Primero comprueba si la página ya está en la reserva de búferes invocando el método lookup(), que puede lanzar una HashNotFoundException cuando la página no está en la reserva de búferes, en la tabla hash para obtener un número de frames. En Hay dos casos que se deben manejar dependiendo del resultado de la llamada a lookup():

1. Caso 1: La página no está en la reserva de búferes. Llame a allocBuf() para asignar un frame de buffer y luego llame al método file-¿readPage() para leer la página desde el disco en el frame del buffer pool. A continuación, inserta la página en la hashtable. Finalmente, invoca a Set() en el frames para configurarlo correctamente. Set() dejará el num_pinde la página establecido en 1. Devuelve un puntero al frame que contiene la página a través del parámetro page.

2. Caso 2: La página está en la reserva del buffer. En este caso se establece el refbit apropiado, se incrementa el num_pinde la página, y devuelve un puntero al frames que contiene la página a través del parámetro page.

3.1.3.6. void unPinPage(File* file, const PageId PageNo, const bool dirty)

Disminuye el num_pindel frame que contiene (file, PageNo) y, si dirty == true, establece el bit dirty. Lanza PAGENOTPINNED si la cuenta de pines ya es 0. No hace nada si la página no se encuentra en la búsqueda de la tabla hash.

3.1.3.7. void allocPage(File* file, PageId PageNo, Page* page)

El primer paso de este método es asignar una página vacía en el archivo especificado invocando la función file-¿allocatePage(). Este método devolverá el número de página de la nueva página asignada. Entonces allocBuf() es llamado para obtener un frame de la reserva de búferes. A continuación, se inserta una entrada en la tabla hash y se invoca a Set() en el frames para configurarlo correctamente. El método devuelve tanto el número de página de la nueva página asignada a través del parámetro pageNo y un puntero al frame del buffer asignado a la página a través del parámetro página.

3.1.3.8. void disposePage(File* file, const PageId pageNo)

Este método elimina una página concreta del archivo. Antes de eliminar la página del archivo, se asegura de que si la página que se va a borrar tiene asignado un frames en el buffer pool, ese frames se libera y la correspondiente entrada de la tabla hash también se elimina. de la tabla hash.

3.1.3.9. void flushFile(File* file)

Debería escanear bufTable en busca de páginas que pertenezcan al archivo. Para cada página encontrada debe:

- Si la página está dirty , llama a la funcion writePage() de file , para vaciar la página en el disco y luego establezca el bit dirty de la página en falso
- Eliminar la página de la tabla hash (ya sea que la página esté limpia o sucia)
- Invocar el método Clear() de BufFramepara el marco de página. Lanza PagePinnedException si alguna página del archivo está anclada. Lanza BadBufferException si se encuentra una página no válida que pertenece al archivo

---

```
class BufManager {
    private:
        FrameId clockHand;
    BufHashTbl * hashTable;
    BufFrame* bufDescTable;
    std::uint32_t numBufs;
    BufMethodAccessbufStats;
    void allocBuf(FrameId & frame);
    void advanceClock();
```

```
10      public:
11          Page * bufPool;
12      BufManager(std::uint32_t bufs);
13      ~BufManager(); //        Destructor
14      void readPage(File * file,
15          const PageId PageNo, Page * & page);
16      void unPinPage(File * file,
17          const PageId PageNo,
18              const bool dirty);
19      void allocPage(File * file, PageId & PageNo, Page * & page);
20      void disposePage(File * file,
21          const PageId pageNo);
22      void flushFile(const File * file);
23  };
```

3.1.4. Arbol B+

Operaciones Básicas del árbol

- Inserción

```
1       Nodo *p;
2       if(raiz == NULL){//Si el arbol no contiene elementos
3
4           p =  new Nodo;
5           p->llaves.nuevo();
6           p->direccion.nuevo();
7           p->esHoja = true;
8           p->padre = NULL;
9
10          p->llaves.agregar(a);
11          raiz = p;
12          return;
13
14      }else{//Si el arbol contiene elementos
15      //y se debe buscar donde se agregara el valor
16
17          buscar(a);
18
19          if(encontrado == true){
20              return;
21          }
```

```
22
23          p = donde;
24
25          if(p->llaves.CuantosVal()<orden-1){ //Si despues
26          //de agregar no se debe modificar los nodos del
27          //arbol
28              p->llaves.agregar(a);
29          }else{ //Si al agregar el valor se debera reajustar el arbol
30              dividirHojas(a, p);
31          }
32          return;
33
34      }
35
36  }
```

- Busqueda

```
    int Arbol::buscar(int a){
Nodo *p = raiz;
CajaValor *val;
CajaDireccion *q;

int n = 0;
while(true){//Busca la hoja del valor
    if(p->esHoja == true){//Busca si
    //esta agregado el elemento en esa hoja

        val = p->llaves.dondePrincipio();

        while(val  != NULL){
            if(val->valor == a){
                encontrado = true;
                donde = p;
                return 0;
            }
            val = val->siguiente;

        }

        encontrado = false;
```

```
            donde = p;
            return 1;

        }else{//Busca en que direccion del nodo del arbol debe bajar

            n = 0;
            val = p->llaves.dondePrincipio();
            q = p->direccion.dondePrincipio();

            while(q  != NULL && val  != NULL){
                    if(a>=val->valor){
                        n++;

                        val = val->siguiente;

                        q = q->siguiente;
                    }else{
                        val = NULL;
                    }

            }
            p = q->direccion;
        }

    }
}
```

- Eliminación

```
int Arbol::borrar(int a){
    Nodo *p;
    buscar(a);

    if(encontrado == false){//Si no se encuentra no se hace nada
        return 0;
    }
```

```
        p = donde;//La ubicacion donde se encontro el valor
        if(p == raiz){//Si solo se tiene la raiz
            if(p->llaves.CuantosVal() == 1){//Si la raiz solo tiene un elemento
                p->direccion.limpiar();
                p->llaves.limpiar();
                p->esHoja = false;
                p->padre = NULL;
                delete p;
                raiz = NULL;
                return 1;
            }
            else{
                p->llaves.borrar(a);
                return 1;
            }

        }else{
            if(p->llaves.CuantosVal()>(int)orden/2){
                if(a  != p->llaves.dondePrincipio()->valor ||
                (p->padre)->direccion.dondePrincipio()->direccion == p){
                    p->llaves.borrar(a);
                }else{
                    p->llaves.borrar(a);
                    (p->padre)->llaves.borrar(a);
                    (p->padre)->llaves.agregar
                    (p>llaves.dondePrincipio()->valor);
                }
                return 1;
            }else{
                modificarHojas(a, p);
            }
        }
        return 1;
    }
```

Referencias

[1] Salvatore R. Mangano