Buffer Manager 设计报告

计算机科学与技术学院 计科1405 於航 3140102234

1. 模块概述

MiniSQL中需要独立的Buffer Manager模块专门管理Buffer，从磁盘文件中读写数据。

Buffer Manager负责读取Block缓存至Buffer，同时向其他模块提供Buffer访问、读写功能，并能将Buffer中的Block写回磁盘文件中。Buffer Manger管理有限个Block存放在Buffer中，它遵循LRU（最近最少使用）原则。

1. 主要功能
   1. 写回区块（writeBack）

将指定block写回相应的磁盘文件。若指定block不存在则返回NOTFOUND，否则写回成功返回SUCCESS。

* 1. 获取空块（getEmptyBlock）

首先遍历buffer寻找未被使用的block，若存在则返回块号。若不存在，则找到LRU值最高，即最近最少使用的block，将其写回，然后把此block初始化，再返回块号。

* 1. 更新LRU（useBlock）

对于一次block调用，将该block的LRU值清零，其他所有的增1。

* 1. 获取块号（getBlockNum）

首先在buffer中寻找指定文件指定偏移的block，若存在，则返回块号。若不存在，则获取空块用于存放，并读取磁盘文件数据，然后返回块号。在该操作中，会更新LRU。

* 1. 删除文件（deleteFile）

删除指定文件名的block。

* 1. 查找插入位置（getInsertPosition）

对于某个文件，要插入数据时，根据table信息遍历该文件所有的block，找到插入位置，返回块号和偏移位。若所有block已满则获取空块，新占用一个block开始存放数据。

1. 对外接口
   1. 获取块号

int getBlockNum(const string fileName, const int blockOffset)

* 1. 删除文件

void deleteFile(const string fileName)

* 1. 获取插入位置

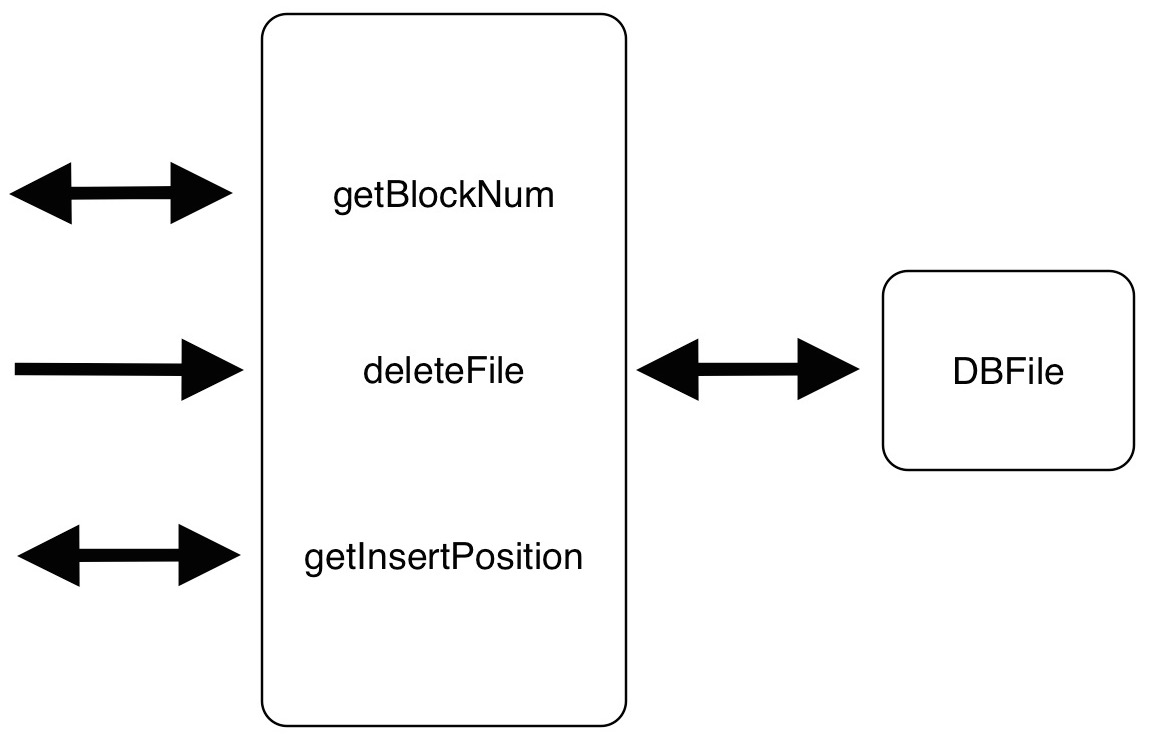
InsertPos getInsertPosition(Table & table)

1. 设计思路

buffer与磁盘的交互均以char＊的形式，虽然存放数据效率降低，但是编写直观。寻找buffer中的block时用的是普通的遍历，没有其他好的优化。LRU值的更新，每当一个block被使用，这个模块的LRU值清零，其他模块增加，即表示当前此模块最近使用的权最大。

1. 整体架构

设计类BufferManager, 由其提供对外接口。外界可以访问buffer，获取BlockNum等值，并可以对buffer进行修改。



1. 关键代码
   1. Block的定义

|  |
| --- |
| class Block {  private:  public:  bool inUse;  int LRUValue;  string FileName;  int BlockOffset;  char Values[BLOCKSIZE + 1];  ……  }; |

* 1. BufferManager的获取块号函数

|  |
| --- |
| int getBlockNum(const string fileName, const int blockOffset) {  int blockNum = getBlockNumInBuffer(fileName, blockOffset);  if (blockNum == NOTFOUND) {  blockNum = getEmptyBlock();  readBlock(fileName, blockOffset, blockNum);  }  useBlock(blockNum);  return blockNum;  }    int getBlockNumInBuffer(const string fileName, const int blockOffset) {  for (int i = 0; i < MAXBLOCKNUM; i++)  if (bufferBlock[i].FileName == fileName && bufferBlock[i].BlockOffset == blockOffset &&  bufferBlock[i].inUse)  return i;  return NOTFOUND;  } // Get blockNum if the block is in the buffer    int getEmptyBlock(const string fileName = "", const int blockOffset = NOTFOUND) {  for (int i = 0; i < MAXBLOCKNUM; i++)  if (!bufferBlock[i].inUse) return i;    int maxLRU = 0, blockNum = 0;  for (int i = 0; i < MAXBLOCKNUM; i++)  if (bufferBlock[i].LRUValue > maxLRU && bufferBlock[i].FileName != fileName &&  bufferBlock[i].BlockOffset != blockOffset) {  blockNum = i;  maxLRU = bufferBlock[i].LRUValue;  }  writeBack(blockNum);  bufferBlock[blockNum].inUse = 0;  return blockNum;  } // If the block we use is not in the buffer, get an idle blockNum for it |

* 1. BufferManager的删除文件

|  |
| --- |
| void deleteFile(const string fileName) {  for (int i = 0; i < MAXBLOCKNUM; i++)  if (bufferBlock[i].FileName == fileName)  bufferBlock[i].inUse = 0;  } // delete all blocks of this file |

* 1. BufferManager的查找插入位置

|  |
| --- |
| InsertPos getInsertPosition(Table & table) {  InsertPos ipos;  if (!table.blockNum) {  ipos.BlockNum = addBlockInFile(table);  ipos.Position = 0;  return ipos;  }  string fileName = table.name + ".table";  int memsize = table.recordLength + 1; // the memory size to store a record  int blockOffset = table.blockNum - 1; // check where is the last block of the table  int blockNum = getBlockNum(fileName, blockOffset);  const int recordNum = BLOCKSIZE / memsize;  for (int offset = 0; offset < recordNum; offset++) {  int position = offset \* memsize;  if (bufferBlock[blockNum].Values[position] == NIL) {  ipos.BlockNum = blockNum;  ipos.Position = position;  return ipos;  } // get an empty position for inserting  }  // There is no empty position in the last block of the table, we need one more block.  ipos.BlockNum = addBlockInFile(table);  ipos.Position = 0;  return ipos;  } // To insert a row to table, find the position in buffer  int addBlockInFile(Table & table) {  int blockNum = getEmptyBlock();  bufferBlock[blockNum].initialize();  bufferBlock[blockNum].FileName = table.name + ".table";  bufferBlock[blockNum].inUse = 1;  bufferBlock[blockNum].BlockOffset = table.blockNum++;  return blockNum;  } // add one more block in the buffer for the table |