Index Manager 设计文挡

作者：2012 级求是科学班（物理）
戴秉璋(bingzhangdai@zju.edu.cn)

一、模块分析

Index Manager 负责 B+树索引的实现，实现 B+树的创建和删除（由索引的定义与删除引起）、等值查找、范围查找、插入键值、删除键值等操作，并对外提供相应的接口。

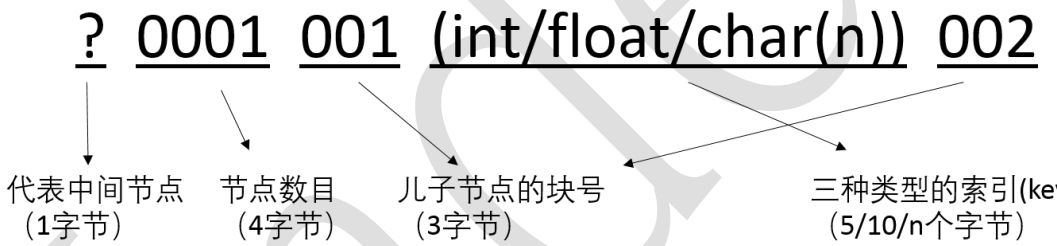
B+树中节点大小应与缓冲区的块大小相同，B+树的叉数由节点大小与索引键大小计算得到。

二、设计思路

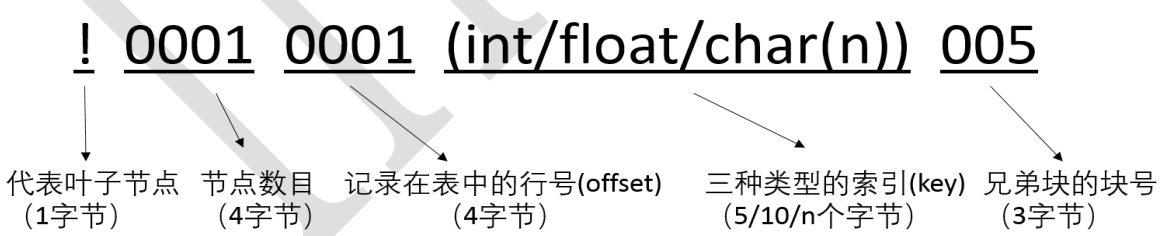
- **节点的设计**

  我们将缓冲区的一个 4k 的 block 作为 B+树的一个节点，每个 block 开始的一个字符来区分中间节点和叶节点，同时紧跟 block 中的 value 的数目以及 value 值，之后对于叶节点紧跟键值行号(offset)和对应的键值(key)，对于中间节点紧跟儿子块的块号(blockNum)和对应的键值(key)。具体细节如图所示。

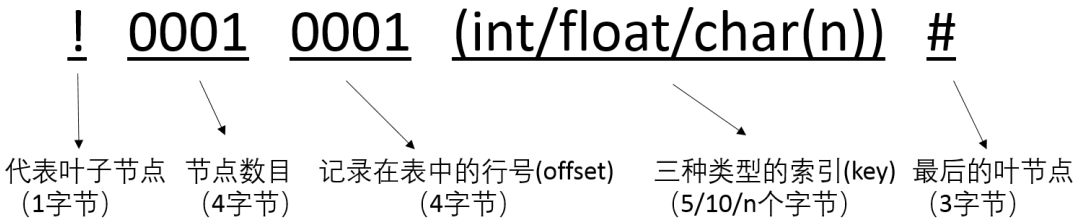
  中间节点：



  叶子节点：



  （非最后一个叶子节点）



  （最后一个叶节点）

- **插入操作**
  1. 找到叶子节点中 search key 可能要插入的位置
  2. a) 如果有空间插入，则将对应的键值和行号插入 B+树相应的位置
     b) 如果已达到节点最大容量，则分裂节点
  3. 更新 B+树（包括父节点是否分裂等操作）
     节点分裂：
     I. 将 n 个键值对排序，分裂为 2 组，前半放入原节点，后半放入新节点
     II. 新节点的键值对插入父节点，并排序。若已满，再分裂，递归向上

- **删除操作**
  1. 找到叶子节点中可能要删除的位置
  2. a) 如果删除后节点大于半满，则直接删除
     b) 如果删除后节点中键值数量小于一半，则：
     I. 若可与相邻的兄弟节点合并为一个节点，则取消右节点，并删除对应的父节点中的项
     II. 否则，与相邻的兄弟节点中的键值对排序后重组，并更新父节点

三、主要功能(外部接口)

Index 主要实现对索引文件的增、删和查找功能。所有传入字符串和结构体尽量使用 const 的 reference 以保证安全并减小开销。除构造函数外，我们一共提供了 4 个外部接口。

```
class IndexManager {
    private:
    /* some code here */
    public:
        IndexManager();
        int search_one(const std::string& database, const std::string& table_name, struct index_info& inform);
        void search_many(const std::string& database, const std::string& table_name, int type, struct index_info& inform, std::deque<int>& container);
        void insert_one(const std::string& database, const std::string& table_name, struct index_info& inform);
        void delete_one(const std::string& database, const std::string& table_name, struct index_info& inform);
};
```

其详细介绍如下：（这里主要解释接口，如何实现的在下一节详细讲解）

- **int search_one(const std::string& database, const std::string& table_name, struct index_info& inform);**

  参数说明：

const std::string& database	数据库名字
const std::string& table_name	表名字
struct index_info& inform	info 结构体

功能：查找一个在数据库名称为 database，表名称 table_name，类型为 infrom.type （包括 int(0)， float(1)，string(2)三种类型），值为 inform.value 的项在表中的行号。

函数返回：

1. 函数正常返回时，将返回一个非负数（即在 index 中的块号）；如该值在 index 中找不到，则返回一个负数。
2. 函数结束返回时，行号保存在 inform.offset 中供调用者使用（仅函数返回非负数时可用）。
3. 外部调用者如果发现返回负数，即说明该项在表中不存在，要及时将信息反馈给用户。

- **void search_many(const std::string& database, const std::string& table_name, int type, struct index_info& inform, std::deque<int>& container);**

参数说明：

const std::string& database		数据库名字
const std::string& table_name		表名字
struct index_info& inform		info 结构体
std::deque<int>& container		返回的行号保存在 container 容器中
int type：	1	找出所有 > inform .value; 的记录
	2	找出所有 >= inform .value 的记录
	3	找出所有 < inform .value 的记录
	4	找出所有 <= inform .value 的记录

功能：查找所有在数据库名称为 database，表名称 table_name，类型为 infrom.type （包括 int(0)， float(1)，string(2)三种类型），并且>（或 >= 或 < 或 <=）inform.value 的项在表中的行号。

函数返回：

1. 函数无返回值，所有找到的行号都保存在 container 容器中。
2. 如果 container.size() == 0，说明没有找到需要的行号，要及时将信息反馈给用户。

- **void insert_one(const std::string& database, const std::string& table_name, struct index_info& inform);**

参数说明：

const std::string& database	数据库名字
const std::string& table_name	表名字
struct index_info& inform	info 结构体

功能：把值为 inform.value，类型为 infrom.type （包括 int(0)， float(1)，string(2)三种类型）的项插入到数据库名称为 database，表名称 table_name 的索引文件中。

函数返回：

1. 函数无返回值，所有找到的行号都保存在 container 容器中。
2. 如果 container.size() == 0，说明没有找到需要的行号，要及时将信息反馈给用户。

- **void delete_one(const std::string& database, const std::string& table_name, struct index_info& inform);**

  参数说明： const std::string& database　　数据库名字
  const std::string& table_name　　表名字
  struct index_info& inform　　info 结构体

  功能：在值为数据库名称为 database，表名称 table_name 的索引文件中把值为 inform.value，类型为 infrom.type （包括 int(0)， float(1)，string(2)三种类型）的索引项删除。

  函数返回：函数无返回值，需事先保证索引文件中有该项。

四. 设计思路与关键函数

- **int search_leaf(const std::string& database, const std::string& table_name, const index_info& inform);**

  从跟节点（0 号 block）开始，层层往下寻找，找到 inform.value 所在的叶节点并返回块号，返回负数表示不存在。

- int search_one(const std::string& database, const std::string& table_name, struct index_info& inform);

  首先调用 search_leaf 函数找到 inform.value 所在叶节点块号，之后将叶节点中每一个 key 做比较，找到 key==inform.value 的项，把叶节点中存储的行号（offset 值）赋给 inform.offset。

- int findPrevLeafSibling(const std::string& database, const std::string& table_name, int blocknum);

  参数 blocknum 应该是某个叶节点的块号，通过从根节点（0 号 block）开始层层往下寻找，到倒数第二层时，进入 blocknum 对应叶节点的左边的一个叶节点，并返回该块号，负数表示未找到。

- int findLeftMostSibling(const std::string& database, const std::string& table_name);

  从跟节点（0 号 block）开始，层层往下寻找，找到最左边的叶节点，并返回块号。

- int findNextLeafSibling(const std::string& database, const std::string& table_name, int blocknum);

  参数 blocknum 应该是某个叶节点的块号，返回 blocknum 的右边一个叶节点的块号。

- int findParent(const std::string& database, const std::string& table_name, const index_info& inform, int blocknum);

  从跟节点（0 号 block）开始，层层往下寻找，如果某个节点是 blocknum 的父节点，则返回该块块号，如果 blocknum 对应的块没有父节点就返回负数。

- void get_index(const std::string& database, const std::string& table_name, int start, int end, int type, struct index_info& inform, std::vector<int>& container);
  start 和 end 是叶节点中起始块和终止块的块号，将其中所有键值的行号(offset)添加进 container 中去。

- void search_many(const std::string& database, const std::string& table_name, int type, struct index_info& inform, std::vector<int>& container);
  首先调用 search_leaf 函数找到 inform.value 所在应该在的叶节点块号，之后根据 type 来确定查找的起点与终点，并调用 get_index 函数把所有找到的行号(offset)添加进 container 中去。

- void insert_leaf(const std::string& database, const std::string& table_name, struct index_info& inform, int Node);
  调用 search_leaf 得到叶节点中需要插入的块的块号，遍历该块，插入键值，如果需要分裂，则分裂节点，并且调用 insert_parent 函数将新的键值对插入父节点。

- void insert_parent(const std::string& database, const std::string& table_name, struct index_info& inform, int Node, const std::string& K, int n);
  插入键值对到该父节点，如果导致该节点分裂，则递归调用该函数，直至根节点。

- void delete_entry(const std::string& database, const std::string& table_name, struct index_info& inform, int n, const std::string& K, int nod);
  nod 是需要被删除的节点，n 是 nod 的父节点，该函数删除节点 n 指向 nod 节点的指针。如果导致父节点 n 需要和兄弟节点合并，则递归调用该函数删除不必要的节点。

- void delete_one(const std::string& database, const std::string& table_name, struct index_info& inform);
  首先调用 search_leaf 确定需要删除键值的叶节点的块号，之后删除叶节点中的该键值对。如果删除后节点小于半满需要合并节点，则调用 delete_entry 函数在父节点中删除指向要删除的节点的指针。

- void write(blockInfo *const node, const std::string& s);
  该函数在每次对节点修改完成后调用，作用是更新节点的 cBlock 里面的内容，并且将 dirtyBit 设为 true。

五、设计总结

Index Manager 是提高数据库系统性能的重要模块，当系统需要检索一个数据的时候，如果不利用 Index，会花更长的时间才能找到目标数据。Index Manager 实现了 Index 的创建，使用，删除等等管理 Index 的功能。

Index Manager 的设计对提高数据库的效率有着重大作用，我们设置当建表时自动对 Primary Key 和 Unique 的项进行 Index 的自动创建，会更好更快地完成检索任务，在验收的时候，我们展示了有无 index 会有肉眼可见的速度差异。

六、附录(预定义结构体)

```
const int Int(0), Float(1), Char_n(2);
const int Greater(1), NotLess(2), Less(3), NotGreater(4);
const int DataFile(0), IndexFile(1);

struct blockInfo;
struct fileInfo {
  int type;                  // 0-> data file
                             // 1 -> index file
  std::string fileName;      // the name of the file
  int recordAmount;          // the number of record in the file
  int freeNum;               // the free block number which could be used for
                                the file
  int recordLength;          // the length of the record in the file
  fileInfo *next;            // the pointer points to the next file
  blockInfo *firstBlock;     // point to the first block within the file
};

struct blockInfo {
  int blockNum;              // the block number of the block
  bool dirtyBit;             // 0 -> flase
                             // 1 -> indicate dirty, write back
  blockInfo *next;           // the pointer point to next block
  fileInfo *file;            // the pointer point to the file, which the block
belongs to
  int charNum;               // the number of chars in the block
  char *cBlock;              // the array space for storing the records in the
block in buffer
  int iTime;                 // it indicate the age of the block in use
  int lock;                  // prevent the block from replacing
};

struct index_info {
  std::string index_name;     //the name of the index file
  int length;                 //the length of the value
  char type;                 //the type of the value
                             //0---int,1---float,2----char(n)
  long offset;               //the record offset in the table file
  std::string value;         //the value
  index_info(){}
};
```