Buffer manager 设计报告

1. 模块概述

Minisql 作为一个数据库关键的是对数据的处理，而内存中的数据处理快，但是容量有限，而磁盘中数据的容量大，但处理慢，如何利用着而 Buffer manager 就是在管理内存 和磁盘中间的数据交互，

2. 主要功能

a) Catalog Manager 负责管理数据库的所有模式信息，包括：

b) 数据库中所有表的定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引，这个表所存储在那个文件中，最后面的一个 block 是在哪里

c) 表中每个字段的定义信息，包括字段类型、是否唯一等。

d) 数据库中所有索引的定义，包括所属表、索引建立在那个字段上等。

e) Catalog Manager 还必需提供访问及操作上述信息的接口，供 Interpreter 和 API 模块使用。

3. 对外提供的接口

```
bool CreateTable_catalog(const Table &); //创建关于表的记录（不含 index，初始 index 个数为 0），
 bool CreateIndex_catalog(const Index&);          //创建关于 index 的记录信息

 bool DropTable_catalog(const Table &);           //删除表
 bool Dropindex_catalog(const Index &);           //删除 index

 //table
 int get_Lblockindex(const Table &);              //得到这个表的最后面的一个 block 的编号
 bool add_Lblockindex(Table &);                   //对表这个最后面的 block 编号更新


 //index
```

```
int get_Lblockindex(const Index &);                    //得到这个表的最后面的一个block 的编号
bool add_Lblockindex(Index &);                         //对表这个最后面的 block 编号更新
bool check_PK(const Table&);                           //检查 attribute 是否是 PK
bool check_Table(const Table &);                       //check whether table exist
bool check_attribute(const Table&);                    //检查 table 里面是否有这个属性，输入的 Table 里只有一个 attribute
bool check_index(const Index &);                       //check whether index exist
vector<Index> check_index(Table &);                    //查看这个 table 中哪一个元素是 index
Table gettable(const Index&);                          //返回 index 所属的 table 的 information
Table get_table_info(const Table&);                    //输入 table 的名字，返回 table 的详细 information
```

4. 设计思路

a) 采用 xml 文件的格式存储信息，将 index 和表的信息都存储在 xml 文件中，存放在 catalog 目录下

b) 每次启动程序的时候，从磁盘中读出 xml 文件读取信息（调用 tinyxml 库的相关函数），将信息存储在内存中，每次当执行 drop 等更新信息的操作的时候，直接在内存中对 catalog 进行修改，并不写会内存，每次程序结束的时候，将内存中的信息写会内存，

c) 因为 catalog 的信息比较少，并且要频繁使用，而内存中的速度要快于磁盘，我们采用了这样的操作来实现 catalog

d) 每次结束的时候会直接将原先写在硬盘的 xml 文件覆盖，写入内存中最新的

e) 在内存中的操作就是简单的 vector 的遍历和访问，没有难度

f) Xml 文件节点信息的具体构造和内存中的 index table 的信心存储结构见本报告的

整体架构

5. 整体架构：

1）内存中的构架

整体为一个 catalog 类，类中 2 个 vector，存储信息

```
Catalog

    Vector<Index>

    Vector<Table>


class Table{
public:
    string name; //表的名称
    int attriNum;   //表中的属性总数
    int totalLength; //一条记录的总长度
    int primaryindex;   //标记哪一个元素为 Primary index
    int biggestnum;      //这个表的最后一个的 block 的编号（文件编号+block 编号） 用于 insert  操作
    vector<Attribute>attributes; //属性的信息
    Table() :name(""), attriNum(0), totalLength(0){}
    ~Table(){}
};

class Index
{
public:
    string index_name;                  //索引名
    string table_name;                  //对应的表名
    string attribute_name;              //对应的属性名字
    int BBN;                            //记录的是文件中最大 block 的编号
    int columnLength;                   //记录的是 index 对应的 value 的总长度
    int blockoffset;                    //block 的偏移量
    int fileoffset;                     //这个 index 是哪一个文件中的   记录他文件偏移
    int type;                           //属性对应类型
    Index() : columnLength(0){}
    ~Index(){}
};
```

2）XML 文件的构件的格式

将 index 和 table 的信息都存储在一个 xml 文件，并且，index 依赖于 table 存储

```
/*调用 tinyxml 文件库存储信息
将 tinyxml 文件存储在当前目录的 calatog 文件下只有一张个 xml 文件*/
格式为
MYcalatog---(number 表示有几个 table number_index)
    <table name>
        Tableinfo（name,attributenum，PI（是一个数值，表示第几个属性是 primary key），biggest_blocknum,TotalLength,fi）
        attributename（name,type(字段类型),length, IsUnique(是否唯一),IsPK）
        attributename（name,type(字段类型),length, IsUnique(是否唯一),IsPK）
        ....
        Index()
            name( indexname,attributename (对应的是第几个元素),BBN（biggest block number）， columnLength,blockoffset(根节点的对应第几个文件的 block),type)
        Index().
        ....
    <>.....
    <table name>
```

6. 关键函数和代码

1.Catalog 的构造函数

```
//初始化 calatog，将 catalog.xml 文件中的内容读到内存中来
catalog::catalog()
{

    //数据库第一次运行，没有生成过 xml 文件，故要这个文件
    if (false == cal.LoadFile(file.c_str())){
        新建节点，并初始化节点信息

    //数据库已经在运行过了，xml 文件已经生成了，则将 xml 文件读入内存
    Else
        解析 table 节点信息，写入内存
            解析 table 下的 attribute 节点信息写入内存
            解析 table 下的 index 的节点信息写入内存
//备注：解析是调用 tinyxml 的库函数，

}
```

2.catalog 的析构函数

```
/*将内存中的 index 和 table 写会 disk 中*/
catalog::~catalog()
    新建一个节点
    遍历内存，逐个将 table 插入那个节点
        遍历内存中的 vector，将 attribute 信息插入 table，作为其子结点
        遍历内存中的 vector，将属于这个表的 index 信息插入 table，作为其子节点
```

3 catalog 里查找 table index 等其他信息

直接在类中循环遍历，找到满足的类，然后返回所需信息

4 更新 catalog 里面 table 和 index 的信息

直接在内存中训啊还遍历，找到对应的元素，进行更新操作