**WickyDB总体设计**

**组长：海杰文 组员：余秋滨、章海威、肖邵斌**

**整体框架**

按照实验要求，WickyDB的主体被分为Interpreter、API、RecordManager、IndexManager、CatalogManager以及BufferManager几个部分。下图是我们设计的几个主要对象及其相互关系，能够比较具象地描绘我们的设计意图。Interpreter使用Yacc和Lex，产生友好的用户交互界面，使得用户能够很方便地使用WickyDB。API与Interpreter紧密相连，所有的SQL语句都能在API层找到相应的API，并进行执行。RecordManager负责管理数据的插入、检索与删除，是WickyDB的重要模块。IndexManager能够创建与删除索引，为API层提供可供使用的索引。CatalogManager负责处理整个系统的所有元数据，包括表的数量、属性、索引的属性、所属等等。从CatalogManager可以获取Schema以完成对表与索引的初始化。最下层的BufferManager直接与文件交流，对上也提供类似文件的接口，基本上透明的设计，用户可以很方便地使用BufferManager而感觉不到Block的存在。

**分工**

**海杰文：**

负责整体框架的设计、各个接口的决定以及Interpreter以及IndexManager的程序编写。其中Interpreter使用Lex与Yacc编写，在具体报告中将给出详细的语法分析树以及如和API层相连接。而IndexManager仅负责制造Index，具体的检索功能在Index中实现。

**余秋滨：**

负责RecordManager的设计与代码编写。并且编写与数据操作有关的API层代码。

**章海威：**

负责CatalogManager的设计与代码编写。并且编写与元数据操作有关的API层代码。

**肖邵斌：**

负责BufferManager的设计、代码编写以及测试工作。BufferManager对上提供透明的读写操作接口，用户感觉不到Block的存在，BufferManager可以很好地管理Block，从而起到Cache的效果。

**主要接口**

class WickyEngine{

static WickyEngine\* getInstance();

void ShowTables();

void DescribeTable(std::string tname);

void createIndex(std::string indexName, std::string tableName, std::string attrName);

void dropIndex(std::string indexName, std::string tableName);

Table\* Select(Table\* t, Condition c);

Table\* Project(Table\* t, std::vector<std::pair<std::string, std::string> > cs);

Table\* Join(Table\* t1, Table\* t2);

int Insert(Table\* t, std::vector<std::pair<std::string, std::string> > values);

int Delete(Table\* t, Condition c);

int InsertByName(std::string name, std::vector<std::pair<std::string, std::string> > values);

int DeleteByName(std::string name, Condition c);

int Update(Table\* t, Condition c);

void CreateTable(Schema sch);

int DropTable(std::string name);

Table\* GetTable(std::string name);

};

class IndexManager{

virtual ~IndexManager();

static IndexManager\* getInstance();

Index\* createIndex(std::string name, std::string type, int keyLen);

Index\* getIndex(std::string name, std::string type, int keyLen);

void deleteIndex(Index\* index);

void dropIndex(Index\* index);

};

class Index{

Index(std::string name, std::string type, int keyLen);

~Index();

int insertKey(Key key, int pointer);

int search(Key k);

int deleteKey(Key k);

}

class BufferManager{

virtual ~BufferManager();

static BufferManager\* getInstance();

bool isFileExists(std::string name);

void redirect(std::string name, int offset=0);

void removeFile(std::string name);

int eof(std::string name);

int readAll(std::string name, int offset, unsigned char\* buf);

int write(std::string name, int offset, int len, unsigned char\* buf);

int read(std::string name, int offset, int len, unsigned char\* buf);

int write(std::string name, int len, unsigned char\* buf);

int read(std::string name, int len, unsigned char\* buf);

int write(std::string name, int offset, int n);

int read(std::string name, int offset, int \*n);

int write(std::string name, int n);

int read(std::string name, int \*n);

int write(std::string name, int offset, double n);

int read(std::string name, int offset, double \*n);

int write(std::string name, double n);

int read(std::string name, double \*n);

int write(std::string name, int offset, std::string str);

int read(std::string name, int offset, std::string \*str, int len);

int write(std::string name, std::string str);

int read(std::string name, std::string \*str, int len);

static void intToBytes(int n, unsigned char\* bytes);

static void doubleToBytes(double n, unsigned char\* bytes);

static void stringToBytes(std::string str, unsigned char\* bytes);

};

**测试结果**

**展望**

最终全部完成了MiniSQL的全部要求，但是我们对整个系统的功能仍有许多不满意的地方。从Interpreter和API方面，我们已经设计好了Join、Projection以及Update等操作的接口，而且从这个系统设计之初就已经在考虑select语句的嵌套等操作。而其他各个更基础的部分，都有各种各样的可供优化的空间。比如我们可以让Index支持数据的范围检索，实现更为精巧的Block替换算法，从各个方面来提高数据库的性能。这次实验是一次非常好的体验，让我们加深了对数据系统的理解，同样也让我们体会到了团队合作的重要性。