**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。

### Interpreter

Interpreter模块直接与用户交互，主要实现以下功能：

1. 程序流程控制，即“启动并初始化 🡪 ‘接收命令、处理命令、显示命令结果’循环 🡪 退出”流程。
2. 接收并解释用户输入的命令，生成命令的内部数据结构表示，同时检查命令的语法正确性和语义正确性，对正确的命令调用API层提供的函数执行并显示执行结果，对不正确的命令显示错误信息。

### API

API模块是整个系统的核心，其主要功能为提供执行SQL语句的接口，供Interpreter层调用。该接口以Interpreter层解释生成的命令内部表示为输入，根据Catalog Manager提供的信息确定执行规则，并调用Record Manager、Index Manager和Catalog Manager提供的相应接口进行执行，最后返回执行结果给Interpreter模块。

### Catalog Manager

Catalog Manager负责管理数据库的所有模式信息，包括：

1. 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。
2. 表中每个字段的定义信息，包括字段类型、是否唯一等。
3. 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。

Catalog Manager还提供了访问及操作上述信息的接口，供Interpreter和API模块使用。

### Record Manager

Record Manager负责管理记录表中数据的数据文件。主要功能为实现数据文件的创建与删除（由表的定义与删除引起）、记录的插入、删除与查找操作，并对外提供相应的接口。其中记录的查找操作能够支持不带条件的查找和带一个条件的查找（包括等值查找、不等值查找和区间查找）。

数据文件由一个或多个数据块组成，块大小应与缓冲区块大小相同。一个块中包含一条至多条记录，为简单起见，只要求支持定长记录的存储，且不要求支持记录的跨块存储。

### Index Manager

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、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。

### Buffer Manager

Buffer Manager负责缓冲区的管理，主要功能有：

1. 根据需要，读取指定的数据到系统缓冲区或将缓冲区中的数据写出到文件
2. 实现缓冲区的替换算法，当缓冲区满时选择合适的页进行替换
3. 记录缓冲区中各页的状态，如是否被修改过等
4. 提供缓冲区页的pin功能，及锁定缓冲区的页，不允许替换出去

为提高磁盘I/O操作的效率，缓冲区与文件系统交互的单位是块，块的大小为文件系统与磁盘交互单位的整数倍，定为4KB或8KB。

**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);

};

class RecordManager{

bool insertTuple(Table\* table, Tuple tuple, int offset);

bool deleteTuple(Table\* table, int offset);

std::vector<Tuple> selectTuple(Table\* table, std::vector<int> offset);

Table readTable(Schema s, BufferManager \*b);

bool writeTable(Table table, BufferManager \*b);

void deleteTable(std::string tableName, BufferManager \*b);

void triWrite(std::string tableName, BufferManager \*b , int key);

void Split(std::string src, std::string separator, std::vector<std::string>& v);

};

class Tuple{

public:

std::vector<std::string> col;

Tuple()；

Tuple(std::vector<std::string> v)；

};

class Table{

Table(std::string tableName);

bool CreateTable(std::vector<Attribute> attrList);

int getAttrNum(){return attrNum;}

std::string getTableName(){return tableName;}

void printTable();

std::vector<Attribute> getAttrList(){return attrList;}

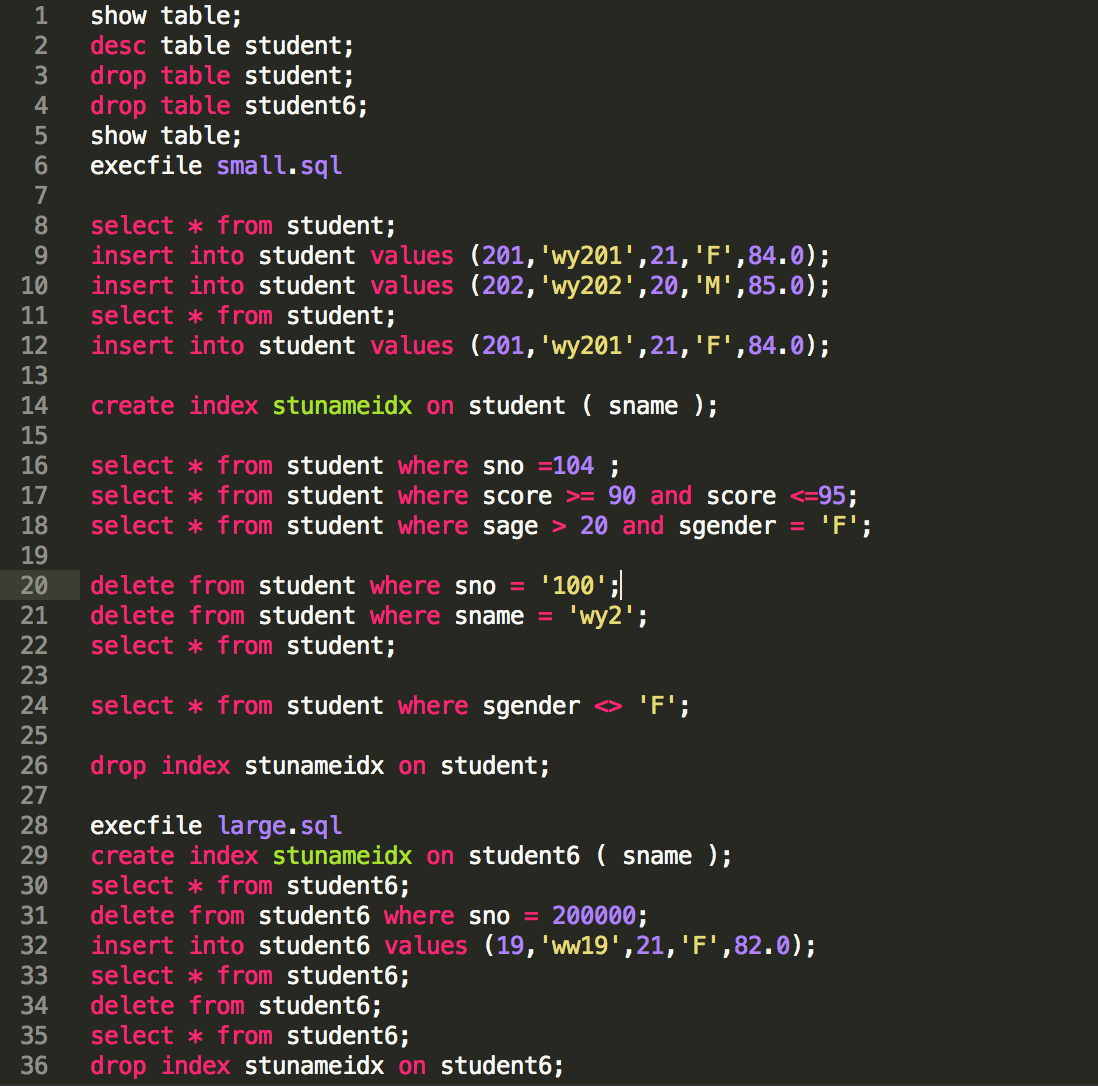
int getTailOffset(){return tailOffset;}

void setTailOffset(int x){tailOffset=x;}

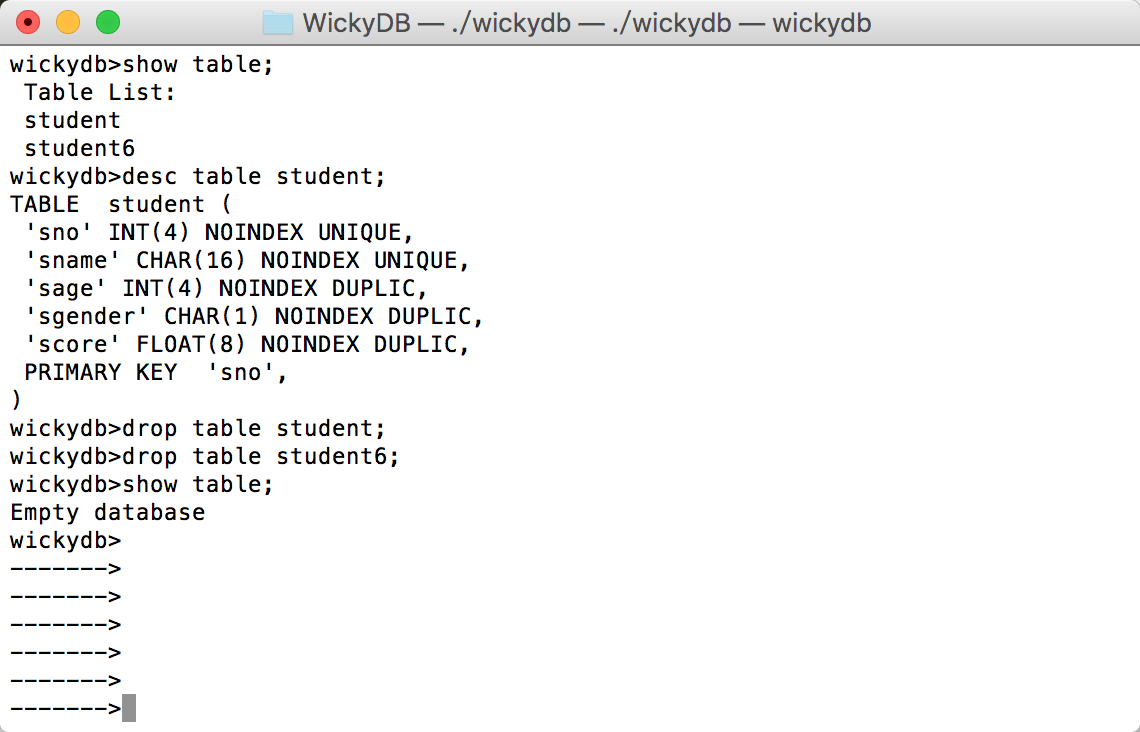
};

**测试结果**

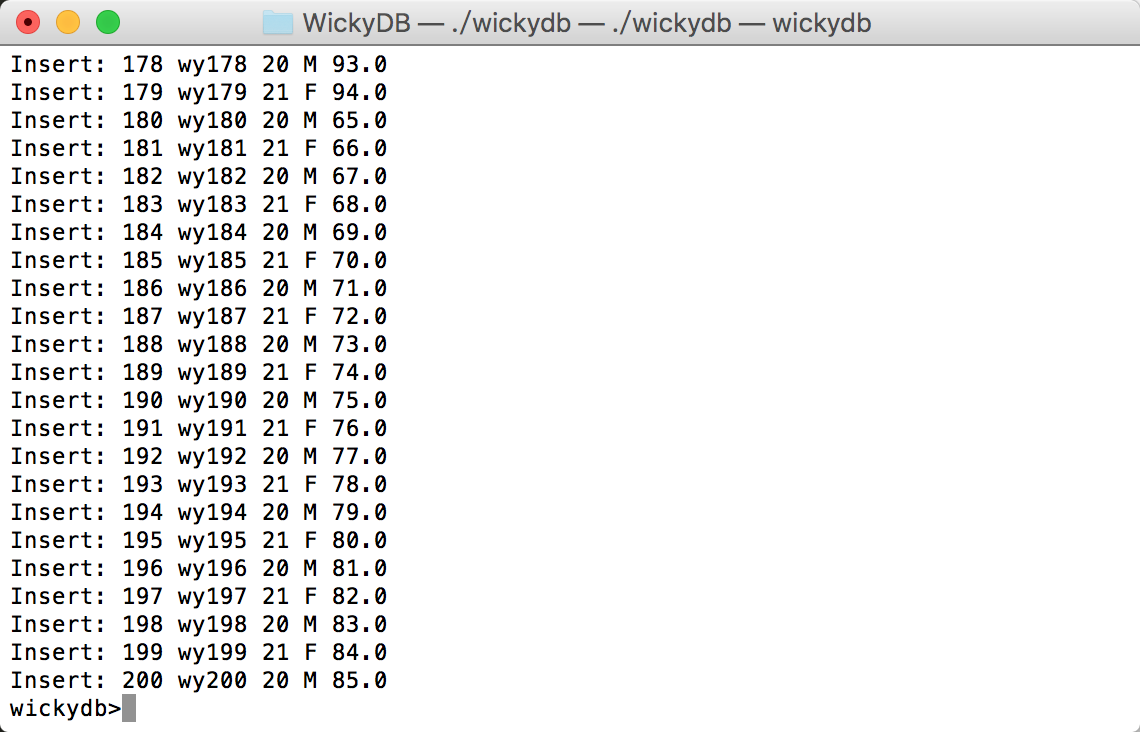
我们的测试展示按照以下指令进行：



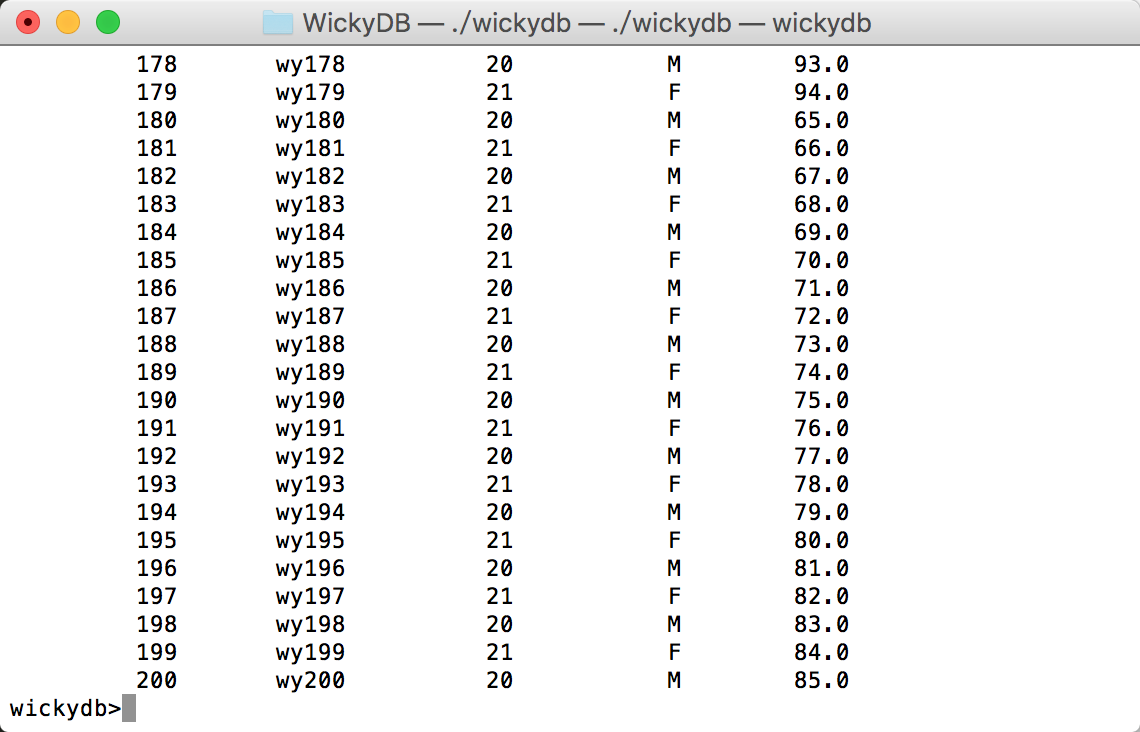
接下来按照上面的流程依次进行操作：

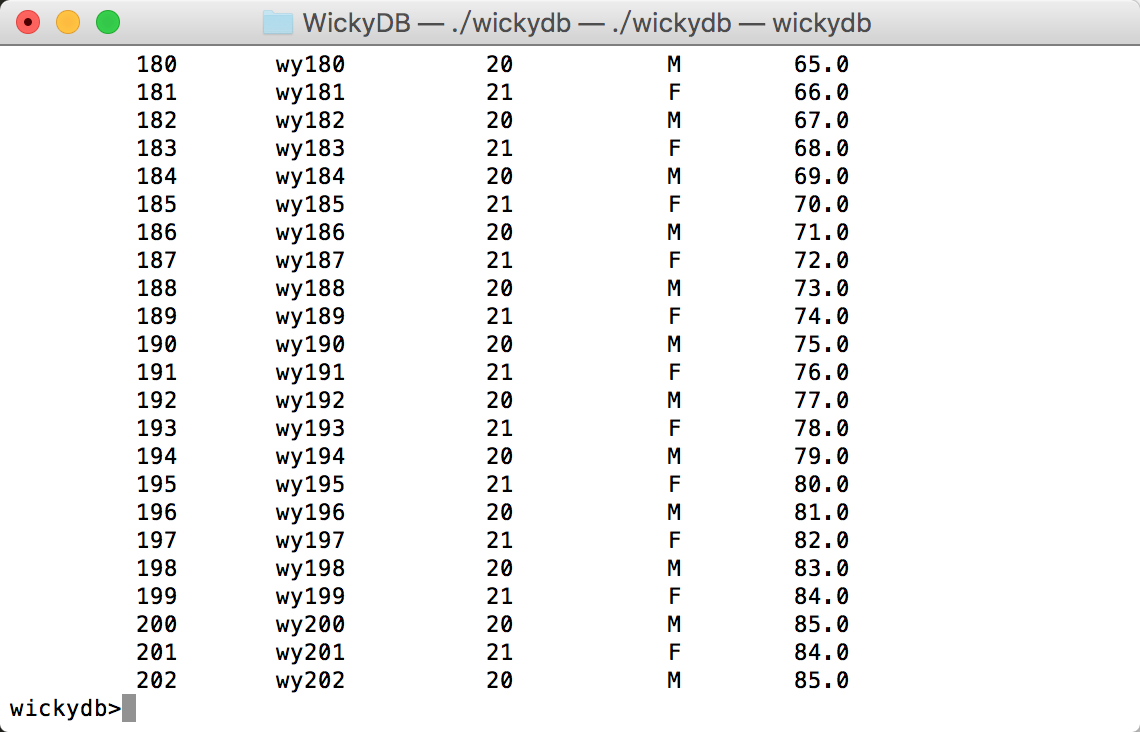


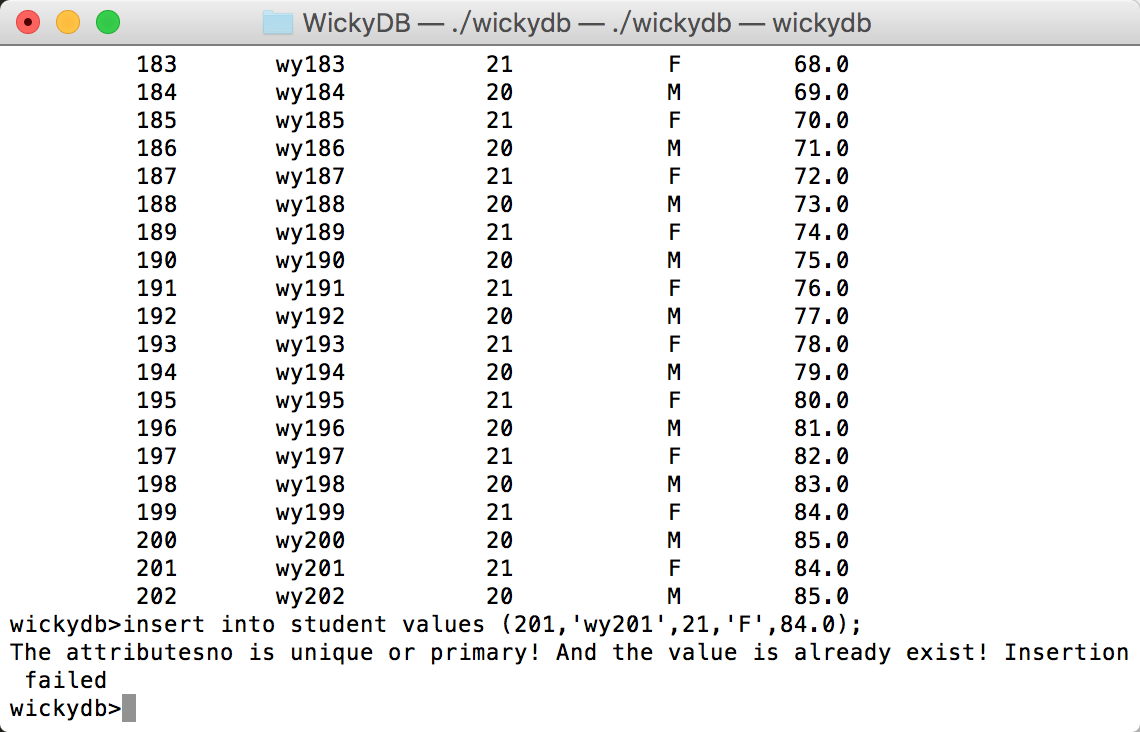
执行execfile small.sql：

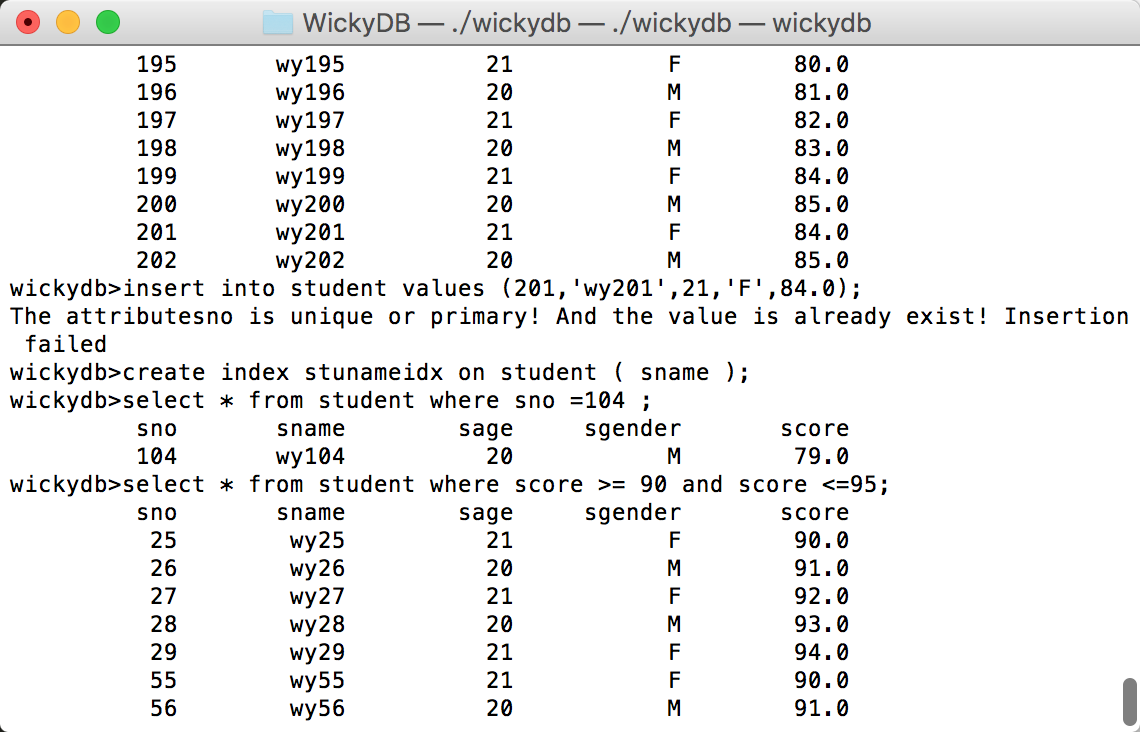


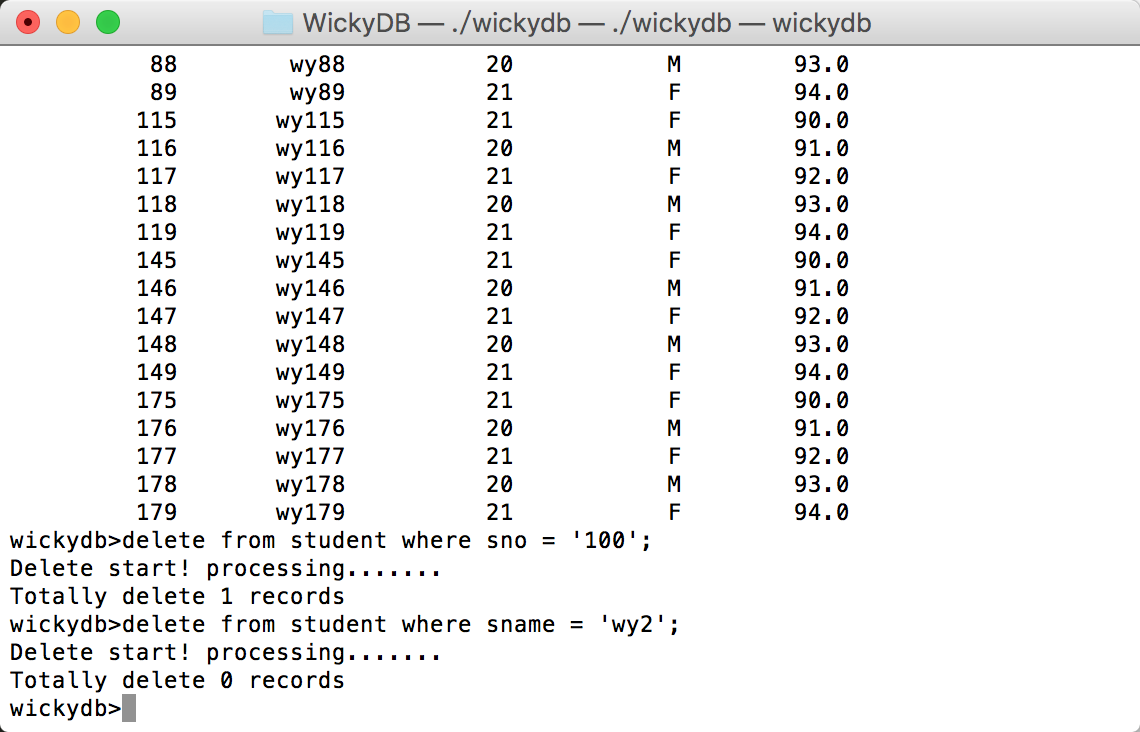
select \* from student:

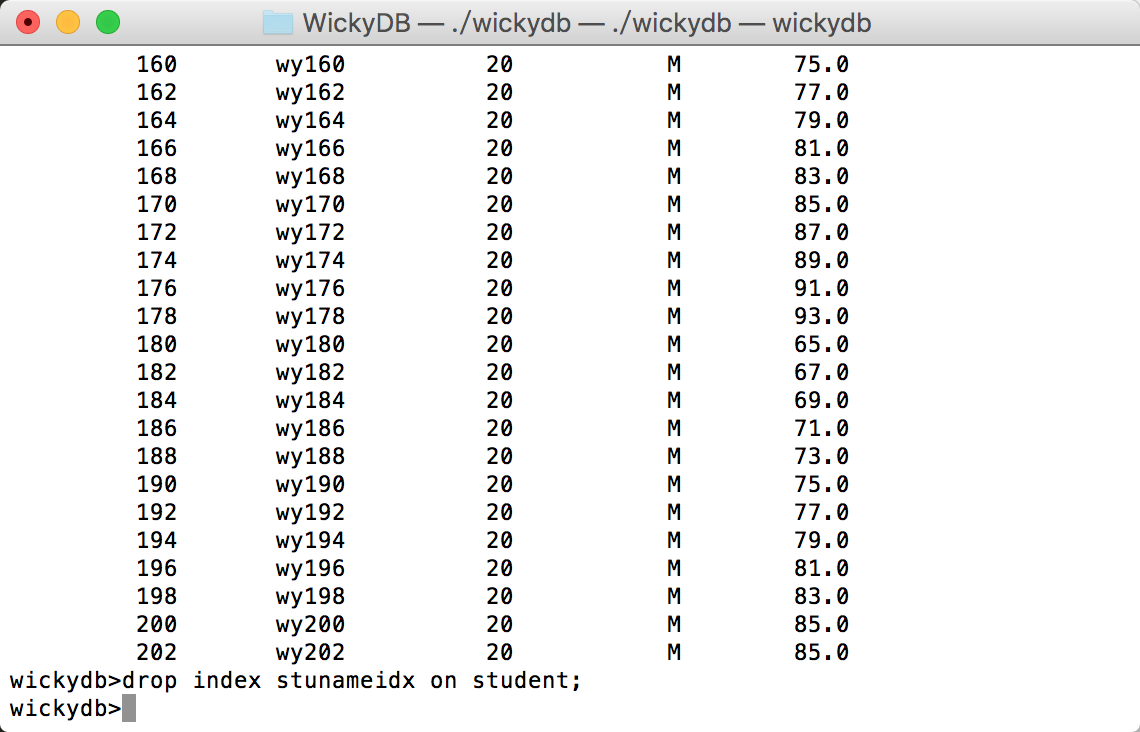


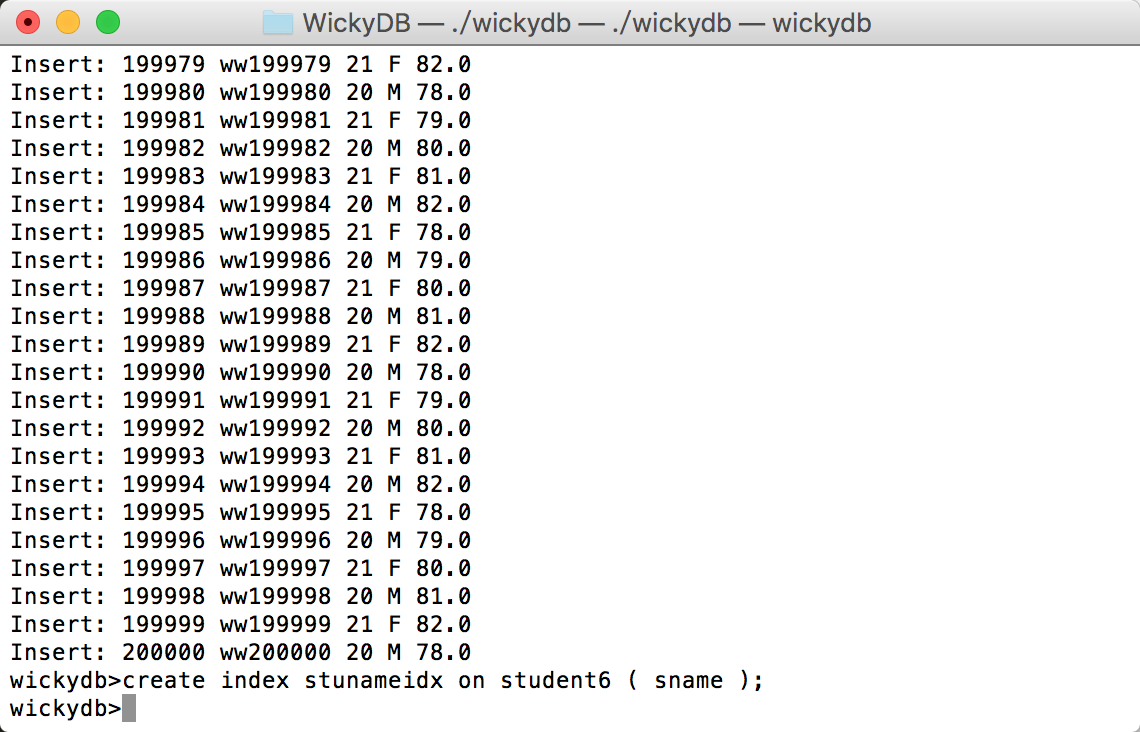


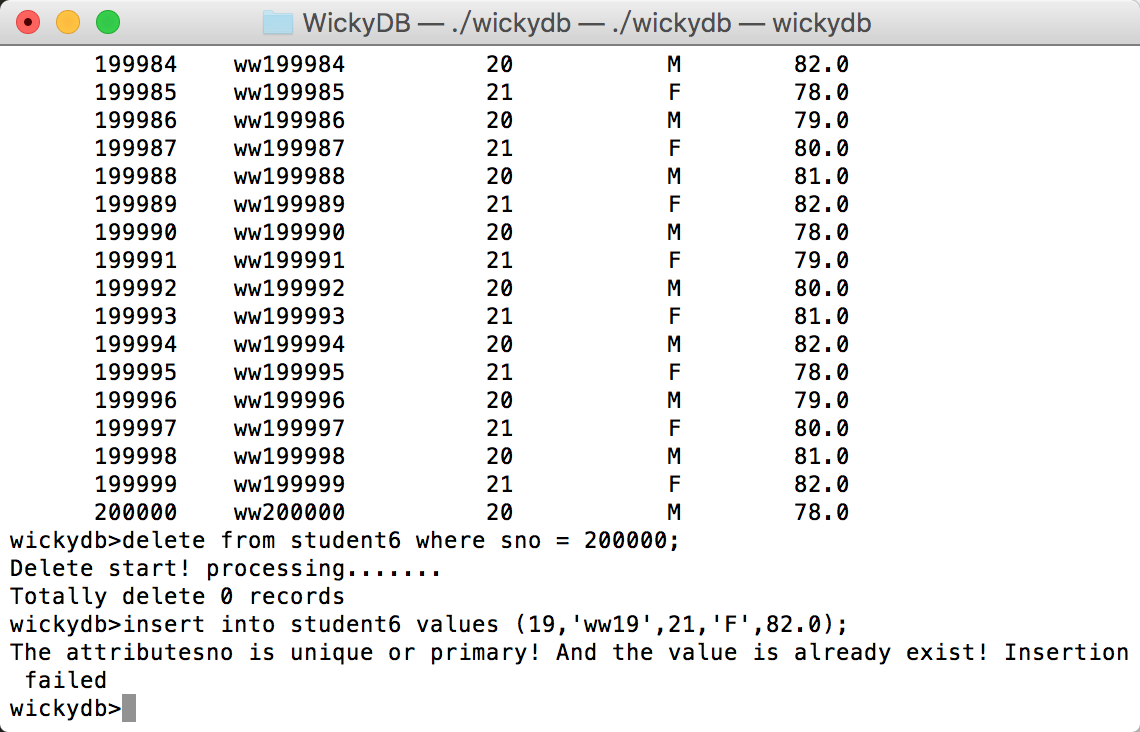


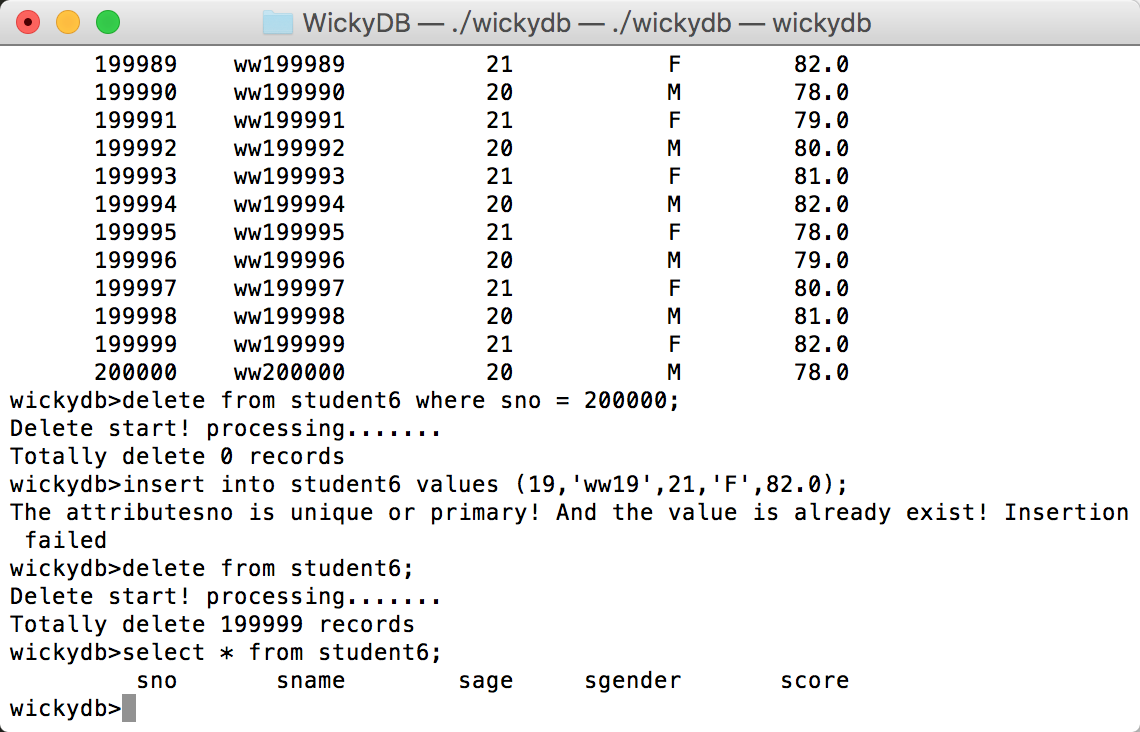












**展望**

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。