各文件的基本功能

main.cpp

通过多个if语句获得SQL关键字，并调用相关函数执行相关操作

vague\_equal.h

判断字符串是否相等，对大小写不敏感，主要用来判断SQL关键字

skipspecialchar.h

跳过特殊字符

getstring.h

跳过特殊字符读入字符串

all.h & all.cpp

执行数据库的创建、删除、显示等

databasein.h & databasein.cpp

执行表的创建、删除、显示等

tablein.h & tablein.cpp

执行数据的插入、删除、选择等

继承关系

Node🡪listnode<int>

🡪listnode<std::string>

🡪listnode<double>

数据库结构

All类包括所有的数据库database

数据库类database由多个表datatable组成，同时包含数据库名字databasename的字符串信息。

表table类的主要组成分为表头label\_info与数据node两部分，同时包含表的大小tablesize，表的名字tablename，主键在表头的位置prikey\_position等信息。表头以vector的形式储存，便于增减；数据以map的形式储存，便于使用主键查找。

表头label\_info类，包括了一个列的名字、类别、是否为空等信息。

表除表头外的每一行（即数据）以map的形式储存，key为主键名字，value为指向node的指针向量，vector中各个元素的次序与对应的表头vector中各列次序相同。

数据库接口

All中的接口：

void create\_database(const std::string &name);

创建一个名字为name的数据库

void delete\_database(const std::string &name);

删除一个名字为name的数据库

void show\_databases();

按字典序列出这个所有的数据库名

inline database\* operator[](const std::string &name)

返回一个指向名字为name的table的指针。用于main函数里设置baseptr。

数据库封装

database中的接口：

void create\_table(const std::string &name);

创建一个名字为name的数据表。

void delete\_table(const std::string &name);

删除一个名字为name的数据表。

void delete\_table();

删除这个database下的所有table。

void show\_tables();

按字典序列出这个数据库里所有的表名。

void show\_columns(const std::string &name);

输出名为name的表的制订表项的各项信息。

Inline table\* operator[](const std::string &name)

返回一个指向名字为name的table的指针。用于main函数里设置tableptr。

tablein.cpp中的接口：

table(**const** std::string name, **const** **int** &keyposition, **const** std::vector<label\_info> v, **int** size = 0);

表的构造函数，设定表的名字、主键位置和表头信息。

~table();

表的析构函数，析构表。

**void** insert\_data();

插入数据。会在该函数中读入插入的数据内容。调用方式：

tableptr ->insert\_data();

**void** delete\_data(**const** std::vector <std::string> &keys);

删除数据。删除向量keys中所有主键对应的数据。调用方式：

tableptr->delete\_data(keys);

**void** delete\_data();

删除数据。用于上层database和all释放底层datalist指针所指的内存。调用方式：

it->second.delete\_data()

**void** update();

更新数据。会在该函数中读入更改的数据和条件。调用方式：

tableptr->update();

**void** select(std::vector <std::string> str);

选择数据。参数为FROM前的内容，可能为\*或者被选中的表头名称。会在该函数中读入选择的条件并输出。调用方式：

tableptr->select(all\_lists) ;

**void** show\_columns();

列出本表的制订表项的各项信息。受上层void database::show\_columns(const std::string &name)的调用。调用方式：

tableptr->show\_columns() ;

**inline int** place(const std::string &listname);

返回名为listname的表头在label\_info中的位置序号（下标）。

std::vector <std::string> whereclause\_complete();

whereclause函数，读入where后的条件语句，执行and和or的运算，返回符合要求的主键向量。调用方式：

std::vector <std::string> keys = whereclause\_complete();

std::vector <std::string> whereclause\_frag(**int** x, std::string str1, **char** operate);

whereclause的辅助函数，执行<, >, =的运算。参数为该数据对应的表头在label\_info中的位置序号（下标）x，该数据的内容str1，以及运算符operate。调用方式：

std::vector <std::string> tmp\_keys = whereclause\_frag(list\_index, value, operate);

数据库封装

Database的封装：database的名字在两个地方两个地方有拷贝，一个在all的private中的std::map<std::string, database> info\_database，这里是为了快速按databasename访问database；另一个在database的public里的std::string databasename。 Database中的table封装在database的public中的std::map<std::string, table> datatable中。

Label\_info是表头的类，public里面有表头的名字name，该列所存的信息类型type，是否为可空的枚举类nullornot non，和第一阶段没有考虑的Default和Extra，因为不知道其具体内容，先一律用string存储。

Node类的说明：为了实现一个指针可以指向不同的类型的数据，我们用了向上类型转换的指针。Node类public中有一个用来实现未来可能的扩展功能的std::string name和为了实现向下类型转换而设置的无意义的虚函数virtual ~node() {}。模板类Listnode类public继承于node，在其public中有用来存储数据的data，和一个用来赋值的构造函数和一个析构函数。

Table的封装：table的名字在两个地方有拷贝，一个在database的public中的std::map<std::string, table> datatable中为方便快速按名字访问table；另一个存储在table的private里的std::string tablename中。

在table的private中有存储表头的std::vector<label\_info> label，有存储主键现对于表头所处位置的int prikey\_position，和存储表中每一行数据的std::map <std::string, std::vector<node\*>> datalist。Datalist是以string类型存储的主键到一个node指针的vector的映射。Vecrtor的大小和表头label的大小相同，每个元素都存储一个向上类型转换的listnode类指针，转换成node\*