MiniSQL

Documentation

|  |  |  |
| --- | --- | --- |
| 组员 | 学号 | 分工 |
| 崔逸卿 | 1100012950 | Record |
| 鲜染 | 1100012783 | API, User Interface |
| 赵天雨 | 1100012957 | Index, Catalog |
| 郑淇木 | 1100012849 | Interpreter |

**Interpreter Documentation**

1. 语法集：

Create:

* create table address(city char, street\_no int);
* create table address(id int primary key, city char, street\_no int );

Select:

* select \* from address;
* select city, street\_no from address;
* select address.city, address.street\_no, travel.city from address, travel;
* select city, street\_no from address where street\_no = 5;
* select name from address where age = 20 and id > 100;

Insert:

* insert into address (city, street\_no) values ('New York', 5);

Update:

* update address set street\_no = 1;
* update address set street\_no = 2 where city = 'New York';

Delete:

* delete from address;
* delete from address where street\_no < 5;

Drop:

* drop table address;

Quit:

1. Class Interpreter

Gets command from STDIN and parses it. This class parses the command and store command information in *info (private variable, type info\_t).*It can also specify some sorts of syntax error and report it to user, aiming to avoid unexpected error caused by user’s wrong input.

1. Method Summary

|  |  |  |
| --- | --- | --- |
| **public/private** | **Modifier and Type** | **Method and Description** |
| Public | bool | inputCommand()  Get command from STDIN |
| Public | info\_t | getInfo()  Get information after interpretation |
| Public | void | debug()  Print debug information for interpreter |
| Private | bool | parseCommand()  Parse information after command is seperated |
| Private | bool | parseInsert()  Parse command with type of insert |
| Private | bool | parseSelect()  Parse command with type of select |
| Private | bool | parseCreate()  Parse command with type of create |
| Private | bool | parseUpdate()  Parse command with type of update |
| Private | bool | parseDelete()  Parse command with type of delete |
| Private | bool | parseQuit()  Parse command with type of quit |
| Private | bool | parseHelp()  Parse command with type of help |
| Private | bool | parseDrop()  Parse command with type of drop |
| Private | void | clearInfo()  Clear all information before rewriting |
| Private | void | showConditionTree(condition\_tree\_t\* root)  Show information of condition tree for debug |
| Private | void | clearTree(condition\_tree\_t\* root)  Free the memory of condition tree recursively |
| Private | condition\_tree\_t \* | makeTree(int index)  make condition tree with given input |

1. Public Method Details

|  |
| --- |
| bool inputCommand()  Gets command from STDIN and parse it.  Returns:  true: Parses command successfully.  false:Unable to parse the command. Error found. |
| info\_t getInfo()  Gets result after interpretation.  Returns:  A struct storing all interpretation information. |
| void debug()  Prints interpretation information for debug |

**API Manager Documentation**

API 模块负责调度各个子模块，首先，他接受用户的输入，交由interpreter解析，再通过解析的结果，调用各个模块的功能实现用户需求，它的主要实现就是在一个while永真循环中一直从标准输入读出命令。

|  |  |  |
| --- | --- | --- |
| **public/private** | **Modifier and Type** | **Method and Description** |
| Public | int | createTable()  Create table |
| Public | int | dropTable()  Drop table. |
| Public | int | createIndex()  Deprecated |
| Public | record\_t\* | select()  Select record from the database |
| Public | int | insert()  Insert a new record |
| Public | int | update()  update a already saved record |
| Public | int | deleteRecord()  delete record |
| Public | int | getInput()  get input from STDIN |
| Public | int | exit()  exit the program |
| Public | void | help()  Parse command with type of help |
| Public | Index\_node\_t\* | getIndex()  Find index by the condition tree. |
| Public | void | commandType()  Clear all information before rewriting |

**Catalog Manager Documentation**

1. Class Catalog

该模块管理的是“模式信息”，实际上就是每张表的定义，每张表中每个字段的定义。

每张表的信息对应一条 tablt\_t 记录，每个字段的信息对应一条 attr\_t 记录，如下所示：

/\* 字段结构 \*/

struct attr\_t {

string name; // 字段名

bool isPrimary; // 是否主键

int length; // 字段占字节数

attrtype\_t type; // 类型（CHAR, INT）

};

/\* 表的结构 \*/

struct table\_t

{

string name; // 表名

int attrNum; // 字段数

int recordLength; // 一条记录的字节数

attr\_t attributes[MAX\_ATTR\_NUM];

// 字段

};

所有的表信息（内嵌字段信息）都存放在一个文件 ../data/table.list 中。

1. Method Summary

|  |  |  |
| --- | --- | --- |
| **public/private** | **Modifier and Type** | **Method and Description** |
| Public | attr\_t | findAttr(string tableName, string attrName)  Get a certain attribute |
| Public | table\_t | findTable(string tableName)  Get a certain table |
| Public | attr\_t | getPrimaryAttr(string tableName)  Get the primary attribute of a certain table |
| Public | bool | tableExist(string tableName)  Check whether a certain table exists |
| Public | bool | attrExist(string tableName, string attrName)  Check whether a certain table exists |
| Public | Int | createTable(table\_t & table)  Create a certain table |
| Public | Int | deleteTable(table\_t & table)  Delete a certain table |
| Private | void | initTable()  Called in construction to init table list file |
| Private | void | A set of I/O packages  Important and useful |

1. More Details

重点在于对文件读写的操作。如：

void Catalog::writeTable(fstream & fout, table\_t & table)

{

fout.write((char \*)table.name.c\_str(), MAX\_CHAR\_LENGTH);

fout.write((char \*)&(table.attrNum), sizeof(int));

fout.write((char \*)&(table.recordLength), sizeof(int));

for (int i = 0; i < table.attrNum; i++)

{

writeAttr(fout, table.attributes[i]);

}

fout.flush();

}

void Catalog::writeAttr(fstream & fout, attr\_t & attr)

{

fout.write((char \*)attr.name.c\_str(), MAX\_CHAR\_LENGTH);

fout.write((char \*)&(attr.isPrimary), sizeof(bool));

fout.write((char \*)&(attr.length), sizeof(int));

fout.write((char \*)&(attr.type), sizeof(attrtype\_t));

fout.flush();

}

**Index Manager Documentation**

1. Class Index

该模块管理的是所有索引文件。

一个索引文件包含索引建在一张表某一字段上的所有记录的索引项，以及一个统计信息的索引头文件。

一个索引项包含对应的记录在某一字段上的值，称为关键码。以及对应的记录在 Record 管理的实际记录文件中的文件偏移。Record 管理器可以通过一个索引项方便地找到对应的实际记录项。

/\* 索引头 \*/

struct index\_head\_t

{

attr\_t attr; // 做索引的字段

int recNum; // 记录数目

};

/\* 索引节点 \*/

struct index\_node\_t

{

string value; // 关键码

unsigned offset; // basep 的偏移

};

一个索引文件存放在 ../data/TABLENAME\_INDEXNAME.idx。

1. Method Summary

|  |  |  |
| --- | --- | --- |
| **public/private** | **Modifier and Type** | **Method and Description** |
| Public | int | selectIndex(string tableName, condition\_tree\_t \*conditionNode, index\_node\_t \*res)  Put indexs in res according to conditionNode |
| Public | int | createIndex(string tableName, string indexName, attr\_t & attr)  Create an index file |
| Public | int | insertIndex(string tableName, string indexName, index\_node\_t & node)  Insert an index (to an ordered list) |
| Public | int | deleteIndex(string tableName, string indexName, string value)  Delete an index |
| Public | int | updateIndex(string tableName, string indexName, string value, string newValue)  Update an index (list still ordered) |
| Public | int | mergeIndexAND(index\_node\_t \*\*list, int listNum, index\_node\_t \*res)  Merge several index lists into one (AND way) |
| Public | int | mergeIndexOR(index\_node\_t \*\*list, int listNum, index\_node\_t \*res)  Merge several index lists into one (AND way) |
| Private | int | A set of binary search functions  To accelerate search in update and select |
| Private | void | A set of I/O packages  Important and useful |
| Private | bool | lessThan(string value\_1, string value\_2, attrtype\_t type)  Compare two value according to type |

1. More Details

* 同上，重点在于对文件读写的操作。
* 一个索引文件中的索引项是根据关键码有序排列的，故在插入、更新后都要更新排列顺序。
* selectIndex 将查询的结果放在列表 res 中，可能索引项，可能只有一条索引项，也可能有多条索引项。
* 在作大小比较时要根据索引项对应的字段类型来比较，使用 lessThan 函数。

**Record Manager Documentation**

1. Class Record

Record模块用来直接对文件中的数据进行增、删、改、选的操作。期中，所有的操作均由二进制文件读写函数write或read完成。

一个记录文件存放在 ../data/TABLENAME.rec。

1. Method Summary

|  |  |  |
| --- | --- | --- |
| **public/private** | **Modifier and Type** | **Method and Description** |
| Public | int | Insert(info\_t & insert\_info)  Insert a certain record |
| Public | void | Delete(info\_t & delete\_info, index\_node\_t & index)  Delete a certain record |
| Public | void | Update(info\_t & update\_info, index\_node\_t & index)  Update a certain record |
| Public | void | Print(record\_t \*record)  Print a certain record info |
| Public | void | PrintHead(table\_t & table)  Print a record head info |
| Public | record\_t \* | Select(info\_t & select\_info)  Pick a group of records wanted |
| Private | int | Judge(condition\_tree\_t \* tempCondition, int offset, table\_t table, ifstream &input)  Judge if a certain record is valid according to condition tree |
| Private | int | getInfo(table\_t table, string infoName, int &Offset, int &attriLength)  get attribute info |

1. More Details

* int Insert(info\_t & insert\_info)

根据insert\_info的内容能确定表的名字、要插入的字段值等所需数据。首先根据表明确定文件名，用二进制格式打开文件，定位到文件末尾。然后对于这个表中的每个字段，逐个在insert\_info中寻找是否有要插入的对应字段，如果有，则按照规定字节数以二进制格式输入；如果没有相应字段值要插入，则以字符串形式输入“oop”。

* void Delete(info\_t & delete\_info, index\_node\_t & index)

根据输入的delete\_info和index可以确定要打开的文件，并且能直接定位到需要删除的记录的首地址偏移（相对于文件头）。之后，逐个对字段输入字符串“oop”。

* void Update(info\_t & update\_info, index\_node\_t & index)

Update的实现和Insert基本一致。根据update\_info和index定位到相应的文件和地址偏移，进行和Insert一样的操作。

* record\_t \*Select(info\_t & select\_info)

Select函数的实现需要充分利用select\_info中的条件树。基本思路如下：根据select\_info定位到相应文件头，然后逐条记录调用Judge函数，判断是否满足条件树的要求。如果满足要求，就将这一条记录包装成一个record\_t节点，最终形成链表，将record\_t的链表头指针返回。

* void Print(record\_t \*record)

Print输出一条记录中所有的字段值。根据输入的record链表，逐个遍历链表，易于实现。

* void PrintHead(table\_t & table)

直接根据table的内容输出一个表中所有字段名，易于实现。

* int Judge(condition\_tree\_t \* tempCondition, int offset, table\_t table, ifstream &input)

Judge函数根据输入的条件树tempCondition，文件内一条记录的首地址偏移offset，table信息和input流，判断一个表中的某一条记录是否符合条件树的要求。这是一个递归函数，每次返回的是当前两个子节点返回值的“与”或“或”，由条件数中的参数决定。

* int getInfo(table\_t table, string infoName, int &Offset, int &attriLength)

getInfo函数根据某个表中的字段名返回这个字段对应字段值的类型（整形或字符串）、字段值的字节数、字段相对于记录首地址的偏移等基本信息，需要被Judge函数调用。