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:

2. 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.

3. 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

4. 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。

2. 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

3. 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。

2. 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

3. 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。

2. 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

3. 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。