SSQL 编译器设计文档

第 6 小组：杨浦城 杨靖 于德强 袁伟佳 吕顺

一、 项目概述

1. 系统概述

本软件实现的是一个 SSQL 语言的编译器，采用 C++语言编写，适当调用 STL 库，利用自动机原理和递归下降设计词法分析和语法分析器，得到一个建议的 SSQL 语言规则，并可以进行增删查的功能。

2. 文档概述

我们实现了课程设计要求的**全部基本项和加分项**，具体项目如下：

1.Context Free Grammar 实现基本的 CREATE、INSERT、SELECT、DELECT 操作

2.可以通过控制台输入，也可以通过文本文件输入

3.对于成功和失败的操作进行相应的提示（具体提示以 MySQL 为模板）

4.支持 expression 表达式，要求的算数、关系和逻辑操作符完美适配 (Bonus 1)

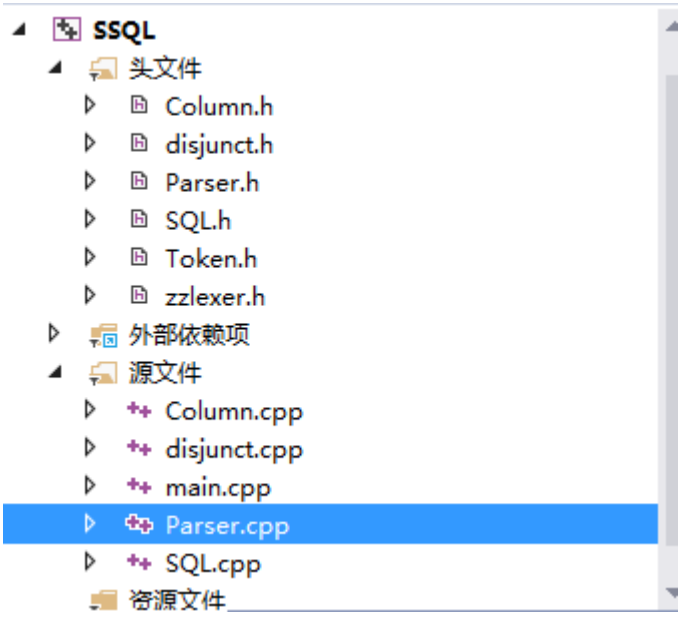
5.DEFAULT 语句和 VALUE LIST 的重载 (Bonus 2)

6. WHERE 从句和 BOOL 变量的重载 (Bonus 3)

7.**创新点：**在使用文件输入和控制台输入的时候会根据输入的语句错误给出相应提示（找不到文件路径、多余的错误字符等），还可以运行-help 查看帮助

二、 文件组织

包中的文件具体组织形式如下



三、 具体实现思路

▪SSQL 语言

1. 数据结构

数据库我们采取列存储方式，将同一列的所有值按照顺序存储在一个 vector 里面，此外，每一列都有列名、默认值等属性，我们构造了一个 Column 类，存放这些属性：

```
class Column {
    private:
        string name;
        vector<int> values;
        bool is_key;
        int default_value;
```

2. SQL 操作

（1） 变量

数据库的建表、插入、删除、查询等操作写在 SQL 类里面。

SQL 类有一个私有变量 database，存储了所有数据：

```
map< string, map<string, Column> > database;
```

第一个 string 是表名，后面的 map 是这张表的数据；第二个 string 是列名，方便快速查询列名是否在表里面。

（2） 建表 create

```
bool create(string table_name, vector<Column> table, vector<string> key_string);
```

语法分析之后将要建立的表名 table_name，以及对应的列表 table 和主键名 key_string 传给 create 函数，函数进行一系列判断（如：表名是否已存在数据库里面，列名会不会重复，主键在不在列表里面），判断没问题就将表单插入数据库 database 里面，否则报出相应错误。

（3） 插入数据 insert

```
bool insert(string table_name, vector<string> names, vector<int> row);
```

插入操作将列名跟每一列的数据插入表单，最难处理的情况是组合主键的组合值不能重复，为了处理这种情况，我们添加了私有变量 key_values 和 key_names :

```
map< string, map<string, int> > key_values;
map< string, vector<string> > key_names;
```

第一个 string 都是表名，key_names 的 vector 存放着组合主键的列名。key_values 的 map 中，string 代表着组合键值的字符串表示，如”123-21”表示值 123 和值 21 的组合，如果这个组合已经存在，则 map 中的 int 值为 1，否则为 0。

一切判断没问题就将数据插入表单中，否则报出相应错误。

（4） 删除数据 deletes

```
bool deletes(string table_name);
bool deletes(string table_name, vector<Token> t);
```

只有一个参数的 deletes 函数表示没有 where 语句的删除，将表单里面的数据都删除掉，但是会留下列名跟列属性，实际操作是将 Column 类中的 vector 清空。

两个参数的 deletes 函数，后面的 vector 是将 where 语句进行词法分析得到的 Token 流。在函数里面，我们将表单的每一行存入`map<string, int> m;`，之后实例化一个 where_dealer 对象进行语法分析：

```
where_dealer w = where_dealer(m, t);
if (w.disjunct() == true)
```

如果 disjunct()函数返回 true，则该行符合 where 语句的条件，我们将其删除。关于 where 语句的语法分析，写在 disjunct 类中，后面还会具体分析。

一切判断没错误则将选中的数据从表单中删除，否则报出相应错误。

（5） 查询数据 select

```
bool select(string table_name, vector<string> names);
bool select(string table_name, vector<string> names, vector<Token> t);
```

names 是要输出的列名，”*”表示输出全部列。只有两个参数的表示不带 where 语句，输出所有行，三个参数的，最后的 vector 是对 where 语句进行词法分析得到的 Token 流，使用方法跟 deletes 函数相似。

一切判断没错误则按照格式输出要查询的数据，否则报出相应错误。

▪Lexer 设计

为了实现 SSQL 编译器的词法分析器部分，我首先阅读并分析了课本《编译原理》附录 A“一个完整的编译器前端”的词法分析器部分的代码逻辑。并基于这一例子，按照本次课程设计中 SSQL 语言的特点以及语法，将词法分析器的实现归结为以下几点。

1）词法分析器基于有穷自动机（DFA）实现。根据读入的字符和当前的状态进行状态转移，DFA 的终止状态为得到单个 Token；

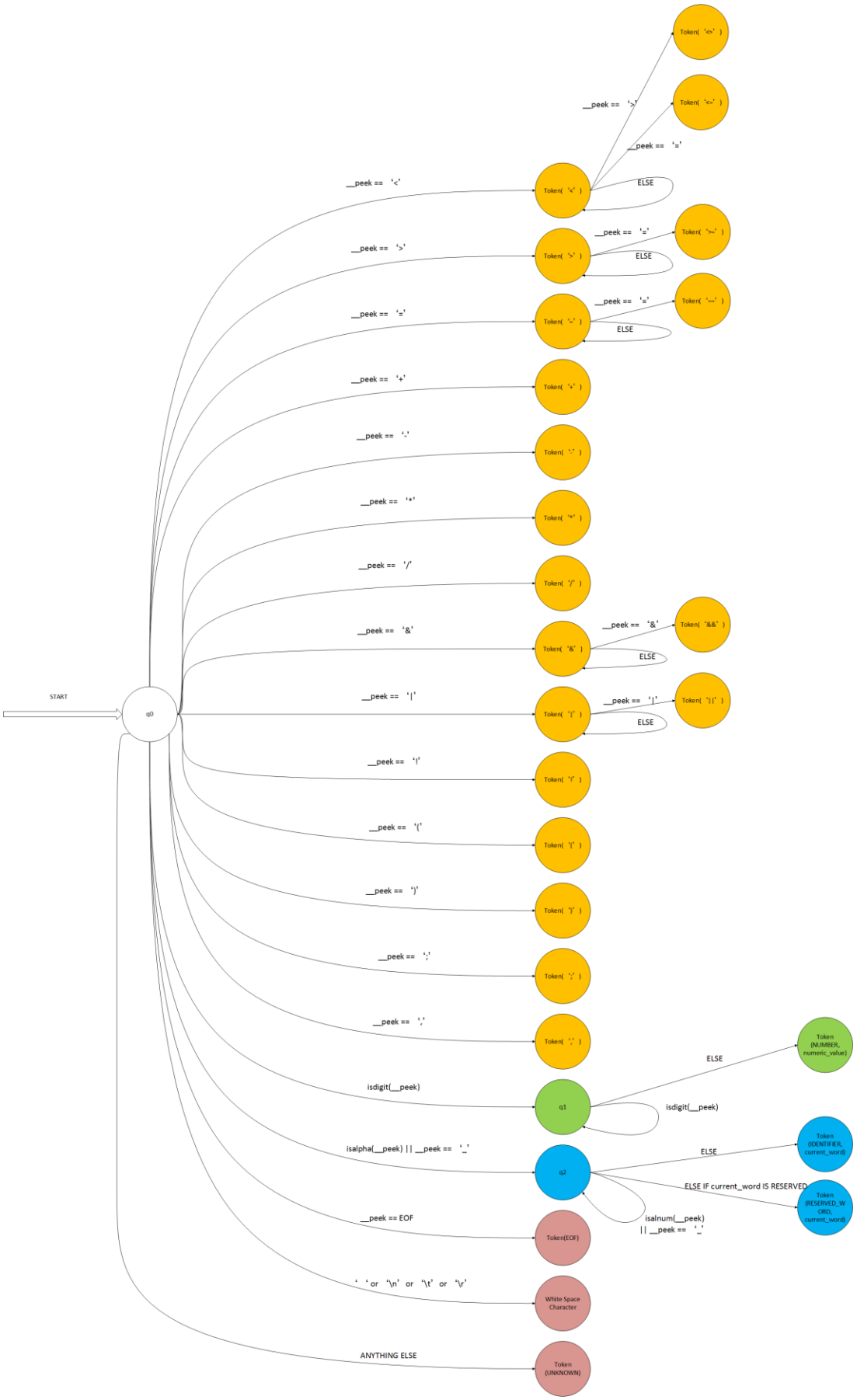
2）该有穷自动机将被循环调用，以对完整的 SSQL 语句进行词法分析，形成连续的 Token 流，供语法分析器读入分析；

3）对于 SSQL 语句，选择以流的形式读入（标准输入流或文件流，对应主程序的两种调用方式）。因此可以借助 C++ istream 的 peek 和 get 函数，实现对流中字符的逐个读入和判断；

4）对于读入的单个字符，可以分为以下四个主要的情况进行判断处理：

i. 运算符和标点。包括算术运算符、逻辑运算符、逗号、分号等；
ii. 数字。注意在本次课程设计中，数字均只为整型数（INTEGER），故不必考虑小数点的问题；
iii. 单词。按照课程设计的要求，保留字（Reserved Words）为不区分大小写的仅包含英文字母的单词，而标识符（Identifier）的构成为“以英文字母或下划线‘_’开头，包含数字、字母或下划线”；
iv. 特殊字符。涉及输入时，要特别注意的一些字符，如空白字符‘ ’，以及一些转义字符，包括制表符‘\t’，回车‘\r’，换行‘\n’，另外还需要注意在读入文件时，会涉及文件结尾 EOF。

5）对于词法分析器返回的单个 Token，考虑语法分析所需的信息，Token 应至少包含其所属的类型以及对应的内容；另外，出于友好用户交互的需要，在输入出错时有必要告知用户错误的位置，故 Token 还需要包含其位置信息（所在的行、列）。


START
q0
__peek == '<'
Token('<')
__peek == '>'
Token('<>')
__peek == '='
Token('<=')
ELSE
__peek == '>'
Token('>')
__peek == '='
Token('>=')
ELSE
__peek == '='
Token('=')
__peek == '='
Token('==')
ELSE
__peek == '+'
Token('+')
__peek == '-'
Token('-')
__peek == '*'
Token('*')
__peek == '/'
Token('/')
__peek == '&'
Token('&')
__peek == '&'
Token('&&')
ELSE
__peek == '|'
Token('|')
__peek == '|'
Token('||')
ELSE
__peek == '!'
Token('!')
__peek == '('
Token('(')
__peek == ')'
Token(')')
__peek == ';'
Token(';')
__peek == ','
Token(',')
isdigit(__peek)
q1
ELSE
Token (NUMBER, numeric_value)
isdigit(__peek)
isalpha(__peek) || __peek == '_'
q2
ELSE
Token (IDENTIFIER, current_word)
ELSE IF current_word IS RESERVED
Token (RESERVED_WORD, current_word)
isalnum(__peek) || __peek == '_'
__peek == EOF
Token(EOF)
' ' or '\n' or '\t' or '\r'
White Space Character
ANYTHING ELSE
Token (UNKNOWN)

▪Parser 设计

按照文档的要求，parser 的实现使用了递归下降分析方法的一种简单的形式：预测分析法。在预测分析法中，各个非终结符号对应的过程中的控制流可以由向前看符号无二义地确定。在分析输入串的时出现的过程调用序列隐式地定义了改输入串的一颗语法分析树。

一个预测分析器程序由各个非终结符对应的过程组成（在程序中就是相应的函数）。对应于非终结符的 A 的过程完成以下两项任务：

1) 检查向前看符号，决定使用 A 的哪个产生式。如果一个产生式的体为 a(这里 a 不是空串)，且向前看符号在 FIRST(a)中，那么就选择这个产生式。对于任何向前看符号，如果两个非空的产生体之间存在冲突，我们就不能对这种文法使用预测语法分析(这应该就是 TA 后来修改文法的原因)。另外，如果 A 有 ε 产生式，那么只有当向前看符号不在 A 的其他产生体的 FIRST 几何中时，才会使用 A 的 ε 产生式。
2) 然后，这个过程模拟被选中的产生式的体。也就是说，从左边开始逐个“执行”此产生式体重的符号。“执行”一个非终结符号的方法是调用该非终结符号对应的过程（即函数），一个与向前看符号匹配的终结符号的“执行”方式则是读入下一个输入符号。如果在某个点上，产生式体中的终结符号和向前看符号是不匹配的，那么语法分析器就会报告一个错误。

语法分析器与其他部分接口的操作可以概括为，从每次从词法分析器拿一出个 token, 进行语法分析，当分析出相应的结果后，调用后台的执行函数进行执行后台的操作，在我们的 project 中就是执行 SQL 的各种操作。

在遇到错误的时候，将结束当前语句的语法分析，进行下一句的语法分析。具体方法就是当遇到错误，该句后续 token 不在进行语法分析，而是一直移动，直到下一句的开始

再者就是递归下降分析方法中普遍存在的问题：左递归会出现死循环。因为这一点，所以所有的文法都要改成右递归的形式才能进行正确的语法分析。

▪WHERE 语句

在 SELECT 和 DELETE 语句中，我们的处理思路是：针对于一个明确的数据表，我们遍历每一行数据，根据 where 语句中的布尔表达式判断该行是否该被选择（删除）。因此在对于 where 语句的处理中，由于 where 语句中包含着 ID Token，ID Token 的值的是数据表中一行里面，每个列 ID 名字所对应的值。在 Parser 部分进行语法解析的时候，是无法读取数据表的值的，所以，我们在 Parser 中处理 WHERE 语句时候，将会吧合法的 WHERE 语句后面的 Token 流传给所调用的 SELECT 和 DELETE 函数，让这小部分 Token 在调用函数中解析，而不是在 Parser 中。

在函数中，对于输入的 Token 流进行解析，应用 TA 给定的文法。由于该文法是左递归的，因此，要实现 LL1 型预测，我们需要把 where 语句后面的文法改成右递归，修改后的文法如下：

```
1  disjunct → conjunct H
2  H → || conjunct H | ε
3  conjunct → bool E
4  E → && bool E | ε
5  bool → (disjunct) | !bool | comp
6  comp → expr rop expr
7  expr → term F
8  F → + term F | - term F | ε
9  term → unary G
10 G → * unary G | / unary G | ε
11 unary → -unary | +unary | id | num
```

因此，根据 First(A)（A 为任意非终结符）的概念，每一个终结符对应于一个处理函数，在函数中，每次查看下一个终结符是什么，并且应用到对应的文法产生式（即实现函数跳转）。但是，只有在某文法产生式中，出现了非终结符且与下一个终结符一致的时候，才会选择继续看下一个终结符，否则，会一直应用其他文法进行展开而不会看一个终结符。

在进行 unary 文法展开时候，如果判断该终结符的 Tag 是 ID 的时候，我们在数据表中进行查询，并且把 ID Token 替换成一个 ID 对应的值放入 bool 表达式的运算中。最后，disjunct 函数（因为 disjunct 是紧跟在 where 这个非终结符后面的）会返回一个 bool 值，完成判断功能，并且协助函数成功筛选或删除对应的行。

附：组员信息和分工

杨浦城	12330370	WHERE 语句设计实现
于德强	12330382	Parser 的设计实现
杨靖	12330363	Lexer 的设计和实现
袁伟佳	12330390	SSQL 语言和操作实现
吕顺	12330227	测试和文案撰写