miniob 更改yacc

以修改join为例。

1 修改lex_sql.l

添加TOKEN标识，规则为遇见左边的字符，返回右边的token。

```
118     BY                              RETURN_TOKEN(BY);
119     STORAGE                         RETURN_TOKEN(STORAGE);
120     FORMAT                          RETURN_TOKEN(FORMAT);
121     JOIN                            RETURN_TOKEN(JOIN);
122     INNER                           RETURN_TOKEN(INNER);
123     OUTER                           RETURN_TOKEN(OUTER);
124     LEFT                            RETURN_TOKEN(LEFT);
125     RIGHT                           RETURN_TOKEN(RIGHT);
126
```

2 修改 yacc_sql.y

```
65
66      //标识tokens
67      %token  SEMICOLON
68              BY
69              CREATE
70              JOIN
71              LEFT
72              RIGHT
73              INNER
74              OUTER
75              DROP
76              GROUP
77              TABLE
78              TABLES
79              INDEX
80              CALC
```

添加token标识符。

```
    std::vector<AttrInfoSqlNode> *            attr_infos;
    AttrInfoSqlNode *                         attr_info;
    Expression *                              expression;
    std::vector<std::unique_ptr<Expression>> * expression_list;
    std::vector<Value> *                      value_list;
    std::vector<ConditionSqlNode> *           condition_list;
    std::vector<JoinSqlNode> *                join_list;
    JoinSqlNode *                             join;
    std::vector<RelAttrSqlNode> *             rel_attr_list;
    std::vector<std::string> *                relation_list;
    char *                                    string;
    int                                       number;
    float                                     floats;
  }

  %token <number> NUMBER
  %token <floats> FLOAT
//非终结符

/** type 定义了各种解析后的结果输出的是什么类型。类型对应了 union 中的定义的成员变量名称 **/
%type <number>              type
%type <condition>           condition
%type <value>               value
%type <number>              number
%type <string>              relation
%type <comp>                comp_op
%type <rel_attr>            rel_attr
%type <attr_infos>          attr_def_list
%type <attr_info>           attr_def
%type <value_list>          value_list
%type <condition_list>      where
%type <join_list>           join_list
%type <join>                join
%type <condition_list>      condition_list
%type <string>              storage_format
%type <relation_list>       rel_list
%type <expression>          expression
%type <expression_list>     expression_list
%type <expression_list>     group_by
%type <sql_node>            calc_stmt
%type <sql_node>            select_stmt
```

定义块返回的数据结构

定义解析后的结果

```
458 select_stmt:        /*  select 语句的语法解析树*/
459     SELECT expression_list FROM rel_list join_list where group_by
460     {
461       $$ = new ParsedSqlNode(SCF_SELECT);
462       if ($2 != nullptr) {
463         $$->selection.expressions.swap(*$2);
464         delete $2;
465       }
466
467       if ($4 != nullptr) {
468         $$->selection.relations.swap(*$4);
469         delete $4;
470       }
471       if ($5 != nullptr) {
472         $$->selection.join_list.swap(*$5);
473         delete $5;
474       }
475
476       if ($6 != nullptr) {
477         $$->selection.conditions.swap(*$6);
478         delete $6;
479       }
480
481       if ($7 != nullptr) {
482         $$->selection.group_by.swap(*$7);
483         delete $7;
484       }
485     }
```

在块作用域中，$$代表自身，并且在块结束后会自动return $$

$n 代表上面匹配的从左到右的每一个块的值
比如，在本例中，$1代表SELECT,$2代表expression_list。
都是c++语句。

```
487 join_list:
488     /* empty */
489     {
490       $$ = nullptr;
491     }
492     | JOIN join join_list {
493           if ($3 != nullptr) {
494             $$ = $3;
495           } else {
496             $$ = new std::vector<JoinSqlNode>;
497           }
498           $$->emplace($$->begin(), *$2);
499         }
500     | INNER JOIN join join_list {
501       if ($4 != nullptr) {
502         $$ = $4;
503       } else {
504         $$ = new std::vector<JoinSqlNode>;
505       }
506       $$->emplace($$->begin(), *$3);
507     }
508     ;
509 join:
510     relation ON condition_list {
511       $$ = new JoinSqlNode;
512       $$->relation = $1;
513       free($1);
514       $$->conditions.swap(*$3);
515     }
516     ;
```

上面名字定义了就是块语句。
每一个大括号都是一个匹配项。
每次匹配都会递归的进行。如果匹配不上就会报错。
或者输入了未定义的token。

3 在pars_def.h中添加对应的数据结构

```
90  struct SelectSqlNode
91  {
92    std::vector<std::unique_ptr<Expression>> expressions;  ///< 查询的表达式
93    std::vector<std::string>                 relations;    ///< 查询的表
94    std::vector<ConditionSqlNode>            conditions;   ///< 查询条件，使用AND串联起来多个条件
95    std::vector<JoinSqlNode>                 join_list;    ///< join节点
96    std::vector<std::unique_ptr<Expression>> group_by;     ///< group by clause
97  };
98
99
100 /**
101  * 表示一个join relation列表
102  */
103 struct JoinSqlNode
104 {
105     std::vector<ConditionSqlNode>            conditions;  ///< 查询的表达式 on子句的内容
106     std::string                              relation;    ///< 连接的表 join后的表
107     std::string                                   op;     ///< 连接方式 inner join,left join,right join,join TODO 暂时只实现join
108 };
```