数据库系统

课程试验报告

试验题目： **MiniOB 数据库管理系统功能的补充与完善**

学 生 姓 名： 丁海桐

学 生 学 号： 202226010304

专 业 班 级： 软件 2203 班

开 课 时 间： 2023-2024-2 学期

试 验 日 期： 2024 年 4 月 13 日

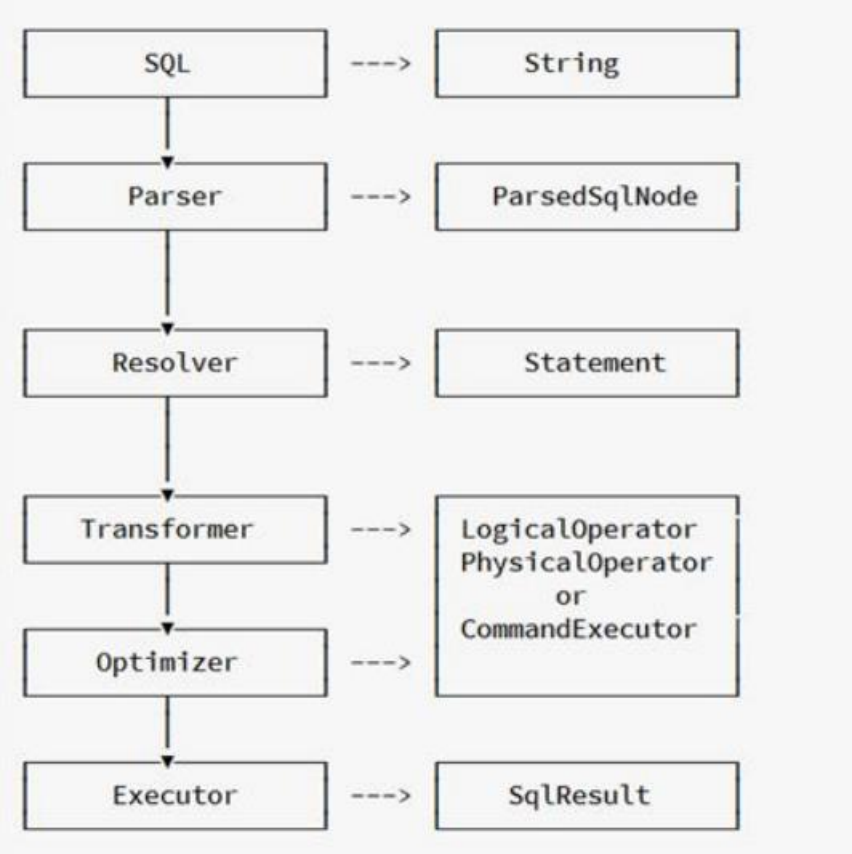
1. 试验任务：

1）实现对 date 这一数据类型的支持；

2）实现聚合运算（sum， avg，count，max， min）；

2. 试验准备情况（对照任务，实验之前给出你的预案）：

SQL 层理解



当用户在 miniob 中输入 SQL 语句时，该请求以字符串形式存储：

1. Parser 阶段：将 SQL 字符串进行词法解析（lex_sql.l）和语法解析（yacc_sql.y）， 最终转换为 ParsedSqlNode（parse_defs.h）。
2. Resolver 阶段：将 ParsedSqlNode 转换为 Stmt（Statement），进行语义解析。
3. Transformer 和 Optimizer 阶段：将 Stmt 转换为 LogicalOperator，并在优化后输 出 PhysicalOperator。这一阶段负责执行查询优化操作。
4. 对于命令执行类型的 SQL 请求，系统会创建相应的 CommandExecutor。
5. 执行阶段 Executor：将 PhysicalOperator（物理执行计划）转换为 SqlResult（执行 结果），或者通过 SqlResult 输出 CommandExecutor 执行后的结果。

3. 试验过程记录（对照任务，对试验方案和结果进行记录和分析）：

（一）实现 Date 数据结构

在 lex_sql.l 中添加 DATE 的 token

```
miniob-2023 > src > observer > sql > parser > lex_sql.l
113  FLOAT          RETURN_TOKEN(FLOAT_T);
114  DATE           RETURN_TOKEN(DATE_T);
115  TEXT           RETURN_TOKEN(TEXT_T);
116  LENGTH         RETURN_TOKEN(LENGTH);
117  ROUND          RETURN_TOKEN(ROUND);
118  DATE_FORMAT    RETURN_TOKEN(DATE_FORMAT);
```

在 ycc_sql.y 中添加 DATE_T

```
miniob-2023 > src > observer > sql > parser > yacc_sql.y
80   %token  SEMICOLON
102          FLOAT_T
103          DATE_T
104          TEXT_T
105          HELP
106          EXIT
107          DOT //QUOTE
```

在 parse_defs.h 中添加 SqlCommandFlag 的枚举

```
miniob-2023 > src > observer > sql > parser > C parse_defs.h
286  /**
289   */
290  enum SqlCommandFlag
291  {
292    SCF_ERROR = 0,
293    SCF_CALC,
294    SCF_SELECT,
295    SCF_INSERT,
296    SCF_UPDATE,
297    SCF_DELETE,
298    SCF_CREATE_TABLE,
299    SCF_DROP_TABLE,
300    SCF_CREATE_INDEX,
301    SCF_DROP_INDEX,
302    SCF_SYNC,
```

在 value.h 中添加 enum 的枚举，并构造 date 的 set 和 get 函数

```
miniob-2023 > src > observer > sql > parser > C value.h
24  /**
27   */
28  enum AttrType
29  {
30    UNDEFINED,
31    CHARS,          ///< 字符串类型
32    INTS,           ///< 整数类型(4字节)
33    FLOATS,         ///< 浮点数类型(4字节)
34    DOUBLES,
35    DATES,          ///< 日期类型
36    LONGS,          ///< Int64
37    TEXTS,          ///< text类型，最大65535字节
38    NULLS,          ///< null类型
39    BOOLEANS,       ///< boolean类型，当前不是由parser解析出来的，是程序内部使用的
40  };
41  // CHARS..DATES 是字段类型，FieldMeta 会进行检查
42  // 所有都是值类型，BOOLEANS 是内部值类型
43
```

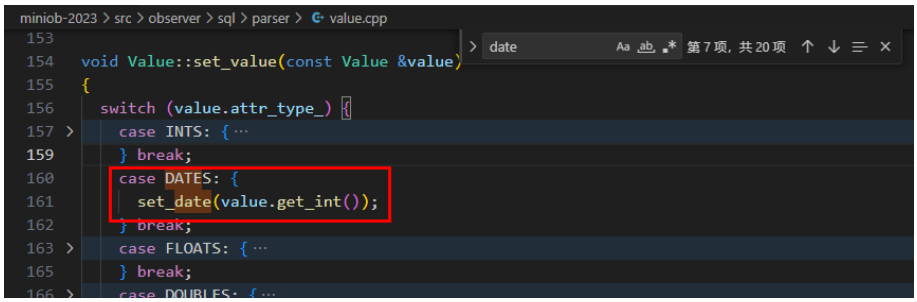
```
miniob-2023 > src > observer > sql > parser > C value.h
53  public:                                   date  Aa ab .* 第 2 项，共 5 项
84    }
85    void set_data(char *data, int length);
86    void set_data(const char *data, int length)
87    {
```

相对应的在 value.cpp 中实现 date 的 set 函数

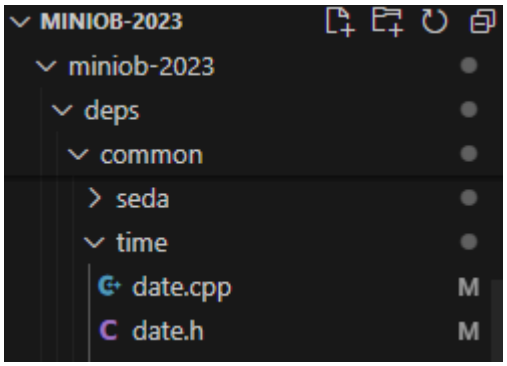
```
miniob-2023 > src > observer > sql > parser > value.cpp
70  {                                         date  Aa ab .* 第 3 项，共 20 项
71    switch (attr_type_) {
87      case BOOLEANS: {
90      } break;
91      case DATES:{
92        num_value_.int_value_ = *(int *)data;
93        length_ = length;
94      } break;
95      case LONGS: {
96        num_value_.long_ = *(int64_t *)data;
97        length_ = length;
98      } break;
99      default: {
100       LOG_WARN("unknown data type: %d", attr_type_);
101     } break;
102   }
```

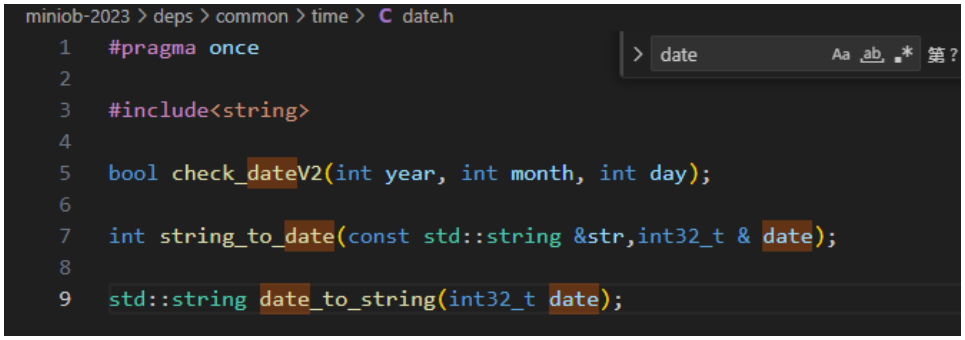
```
miniob-2023 > src > observer > sql > parser > value.cpp
67
68  // NULLS can not set data
69  void Value::set_data(char *data, int length)
70  {
71    switch (attr_type_) {
72      case CHARS: {…
74      } break;
75      case INTS: {…
78      } break;
79      case FLOATS: {…
82      } break;
83      case DOUBLES:{…
86      } break;
87      case BOOLEANS: {…
90      } break;
91      case DATES:{…
94      } break;
95      case LONGS: {…
98      } break;
99      default: {…
101     } break;
102   }
```

在 set_value 中添加 DATES 的 case



在 miniob-2023\deps\common\time 中添加 date.h 和 date.cpp





miniob-2023 > deps > common > time > date.cpp

```cpp
#include"date.h"
#include<string>
#include"stdio.h"
#include "common/log/log.h"
bool check_dateV2(int year, int month, int day)
{
  static int mon[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
  LOG_WARN("check_dateV2: year %d,month %d,day %d");
  bool leap = (year % 400 == 0 || (year % 100 && year % 4 == 0));
  if (year > 0 && (month > 0) && (month <= 12) && (day > 0) && (day <= ((month == 2 && leap) ? 1 : 0) + mon[month]))
    return true;
  else
    return false;
}

int string_to_date(const std::string &str,int32_t & date)
{
    int y,m,d;
    sscanf(str.c_str(), "%d-%d-%d", &y, &m, &d);//not check return value eq 3, lex guarantee
    bool b = check_dateV2(y,m,d);
    if(!b) return -1;
    date = y*10000+m*100+d;
    return 0;
}
std::string date_to_string(int32_t date)
{
    std::string ans = "YYYY-MM-DD";
    std::string tmp = std::to_string(date);
    int tmp_index = 7;
    for(int i = 9 ; i >=0 ;i--)
    {
        if(i == 7|| i == 4)
        {
            ans[i] = '-';
        }
        else
        {
            ans[i] = tmp[tmp_index--];
        }
    }
    return ans;
}
```

至此，DATE 实现就已经完成。

（二）实现聚合函数

在 yacc_sql.cpp 中添加函数 get_aggr_func_type

miniob-2023 > src > observer > sql > parser > yacc_sql.cpp

SUM Aa ab .* 第 1 项，共 5 项

```cpp
{
}

AggrFuncType get_aggr_func_type(char *func_name)
{
  int len = strlen(func_name);
  for (int i = 0; i < len; i++) {
    func_name[i] = tolower(func_name[i]);
  }
  if (0 == strcmp(func_name, "max")) {
    return AggrFuncType::AGG_MAX;
  } else if (0 == strcmp(func_name, "min")) {
    return AggrFuncType::AGG_MIN;
  } else if (0 == strcmp(func_name, "sum")) {
    return AggrFuncType::AGG_SUM;
  } else if (0 == strcmp(func_name, "avg")) {
    return AggrFuncType::AGG_AVG;
  } else if (0 == strcmp(func_name, "count")) {
    return AggrFuncType::AGG_COUNT;
  }
  return AggrFuncType::AGGR_FUNC_TYPE_NUM;
}
```

并且添加相应 token

```
miniob-2023 > src > observer > sql > parser > yacc_sql.cpp
180   # define YYTOKENTYPE
182     {
237       LIKE = 312,
238       UNIQUE = 313,
239       AGGR_MAX = 314,
240       AGGR_MIN = 315,
241       AGGR_SUM = 316,
242       AGGR_AVG = 317,
243       AGGR_COUNT = 318,
244       LENGTH = 319,
245       ROUND = 320,
```

在 parse_def.h 中添加枚举 AggrFuncType

```
miniob-2023 > src > observer > sql > parser > C parse_defs.h > ...
27
28  typedef enum { AGG_MAX, AGG_MIN, AGG_SUM, AGG_AVG, AGG_COUNT, AGGR_FUNC_TYPE_NUM } AggrFun
29  typedef enum { SYS_FUNC_LENGTH, SYS_FUNC_ROUND, SYS_FUNC_DATE_FORMAT,SYS_FUNC_TYPE_NUM } S
30  /**
```

在 src/observer/sql/expr/expression.cpp 中实现 func 的 get_type 和 get_value 函数

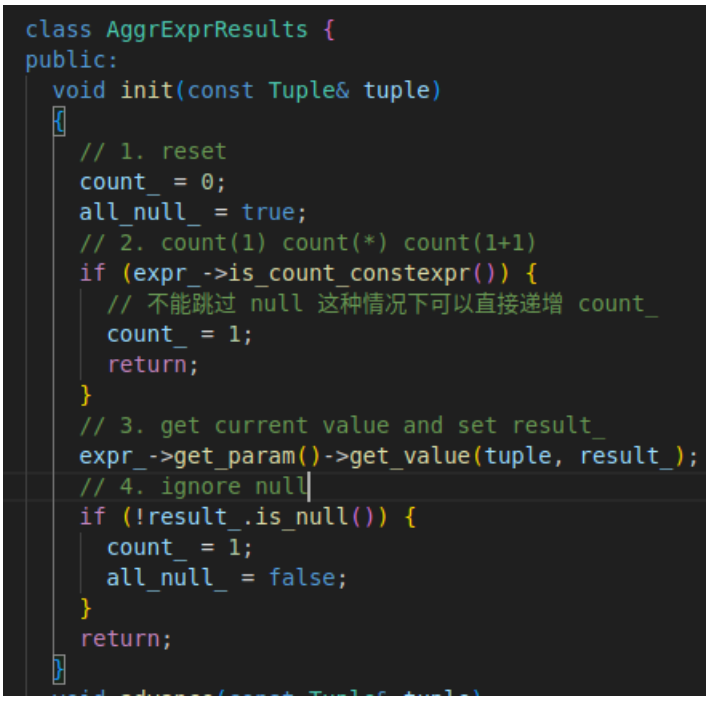
src > observer > sql > expr > expression.cpp > ...

```cpp
std::string AggrFuncExpr::get_func_name() const
{
  switch (type_) {
    case AggrFuncType::AGG_MAX:
      return "max";
    case AggrFuncType::AGG_MIN:
      return "min";
    case AggrFuncType::AGG_SUM:
      return "sum";
    case AggrFuncType::AGG_AVG:
      return "avg";
    case AggrFuncType::AGG_COUNT:
      return "count";
    default:
      break;
  }
  return "unknown_aggr_fun";
}

AttrType AggrFuncExpr::value_type() const
{
  switch (type_) {
    case AggrFuncType::AGG_MAX:
    case AggrFuncType::AGG_MIN:
    case AggrFuncType::AGG_SUM:
      return param_->value_type();
      break;
    case AggrFuncType::AGG_AVG:
      return DOUBLES;
      break;
    case AggrFuncType::AGG_COUNT:
      return INTS;
      break;
    default:
      return UNDEFINED;
      break;
  }
```

```cpp
RC AggrFuncExpr::get_value(const Tuple &tuple, Value &cell) const
{
  TupleCellSpec spec(name().c_str());
  //int index = 0;
  // spec.set_agg_type(get_aggr_func_type());
  if(is_first_)
  {
    bool & is_first_ref = const_cast<bool&>(is_first_);
    is_first_ref = false;
    return tuple.find_cell(spec,cell,const_cast<int&>(index_));
  }
  else
  {
    return tuple.cell_at(index_,cell);
  }
}
```

在 src/observer/sql/expr/tuple.h 中实现具体过程



```
class AggrExprResults {
public:
  void init(const Tuple& tuple)
  {
    // 1. reset
    count_ = 0;
    all_null_ = true;
    // 2. count(1) count(*) count(1+1)
    if (expr_->is_count_constexpr()) {
      // 不能跳过 null 这种情况下可以直接递增 count_
      count_ = 1;
      return;
    }
    // 3. get current value and set result_
    expr_->get_param()->get_value(tuple, result_);
    // 4. ignore null
    if (!result_.is_null()) {
      count_ = 1;
      all_null_ = false;
    }
    return;
  }
```

src > observer > sql > expr > C tuple.h > GroupTuple > AggrExprResults > init(con

```
466   class GroupTuple : public Tuple {
594     class AggrExprResults {
616       void advance(const Tuple& tuple)
624         Value temp;
625         expr_->get_param()->get_value(tuple, temp);
626         // 3. ignore null
627         if (temp.is_null()) { // 直接跳过
628           return;
629         }
630         // 4. update status
631         count_++;
632         all_null_ = false;
633         // 5. init 的时候拿到的是 null
634         if (result_.is_null()) {
635           result_ = temp;
636           return;
637         }
638         // 6. do aggr calc
639         switch (expr_->get_aggr_func_type()) {
640           case AggrFuncType::AGG_COUNT: {
641             // do nothing
642           } break;
643           case AggrFuncType::AGG_AVG:
644           case AggrFuncType::AGG_SUM: {
645             result_.add(temp);
646           } break;
647           case AggrFuncType::AGG_MAX: {
648             if (result_ < temp) {
649               result_ = temp;
650             }
651           } break;
652           case AggrFuncType::AGG_MIN: {
653             if (result_ > temp) {
654               result_ = temp;
655             }
656           } break;
```

至此聚合函数结束。

4. 试验完成情况：

（一）DATE 验证

创建一个 date_test（id int, bir date）表，并插入一项数据，查询验证。

```
miniob > create table date_test(id int, bir DATE);
SUCCESS

miniob > insert into date_test values (1, '2004-09-27');
SUCCESS

miniob > select * from date_test
id | bir
1 | 2004-09-27

miniob > ^[^A
```

（二）聚合函数验证

创建 func_test(sal int)表，并插入一些数据。

```
miniob > create table func_test(sal int);
SUCCESS

miniob > insert into func_test values (1)
SUCCESS

miniob > insert into func_test values (2)
SUCCESS

miniob > insert into func_test values (3)
SUCCESS

miniob > insert into func_test values (4)
SUCCESS
```

验证五个聚合函数

```
miniob > select sum(sal) from func_test
sum(sal)
10

miniob > select min(sal) from func_test
min(sal)
1

miniob > select max(sal) from func_test
max(sal)
4

miniob > select count(*) from func_test
count(*)
4

miniob > select avg(sal) from func_test
avg(sal)
2.5
```

5. 试验总结：（遇到的问题及解决措施，对试验的评价，感想和认识）

实验难度很大，期间遇到了很多问题，比如在实现聚合函数时候，为了不破坏代码框架以及不影响后续实现，要谨慎决定代码之间的联系，后来在 src/observer/sql/expr/tuple.h 进行函数的实现，新创建的文件不影响代码结构，只是提供了接口。

工程代码庞大，所以阅读源码是一项很很艰巨的任务，要耐下性子认真读，可以从 main 中开始，也可以从 SQL 的层次出发，分部分阅读。期间网络上有一些比赛心得可以参考。