ZHEJIANG UNIVERSITY
DATABASE SYSTEM, 2020 SUMMER

Interpreter 设计报告

Date:2020.6.25

邢书婷
3180106027
数字媒体技术
计算机科学技术学院

目录

一、 模块概述
二、 主要功能
三、 对外提供的接口
四、 设计思路
五、 整体架构
六、 关键函数和代码

一、 模块概述

模块负责接收并处理用户在前端输入的 SQL 命令，识别命令的类型，解析为有意字符串，执行不同的子函数，分别调用 API 模块和 Catalog 模块的功能函数，并接收返回值。在界面输出执行结果或者抛出异常。对于选择命令，需要进一步在前端输出结果。在处理命令时，对于不同的命令类型分别进行输入规范检查，若出现错误，例如表的重定义，使用不存在的属性，插入数据数目与字段数目不匹配，数据类型不匹配等问题，应给出相应提示。若操作成功，也返回相应提示。

二、 主要功能

接受处理命令：接受输入的 SQL 语句，将命令中的符号去掉，留下字符串存入 vector 容器。

得到命令类型：根据上一步解析后的字符串，根据前两个字符串确定命令类型，并执行相应的子函数，例如：创表语句中的第一二个字符串为“create”“table”，则执行创表函数。

进行命令检查并储存相应的信息：检查命令是否符合要求，若不符合，则抛出异常，若符合，则返回有用的字段信息。

显示执行结果：根据 API 中的执行，在界面输出执行结果

三、 对外提供的接口

1. 接受命令

```
void SetStr(string _srcstr);
```

2. 处理命令

```
void Parse();
```

3. 得到命令类型

```
CmdType GetOpType(vector<string> sen_str);
```

4. 主程序交接入口，输入

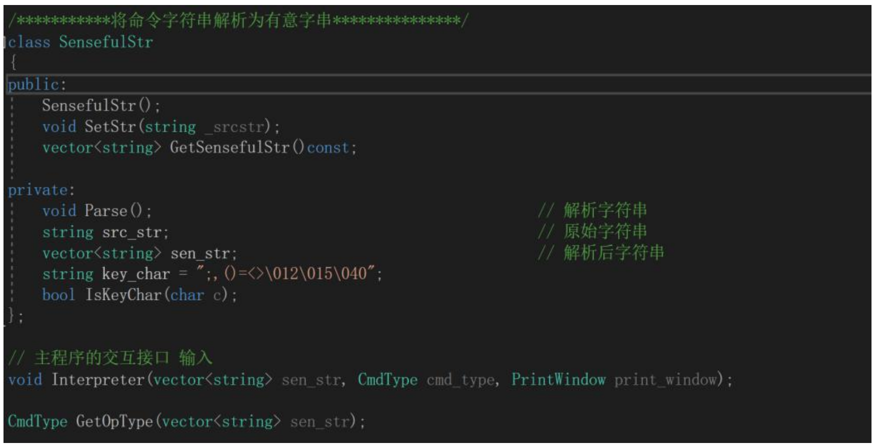
```
void Interpreter(vector<string> sen_str, CmdType cmd_type, PrintWindow print_window);
```

5. 储存有用信息存进 Catalog

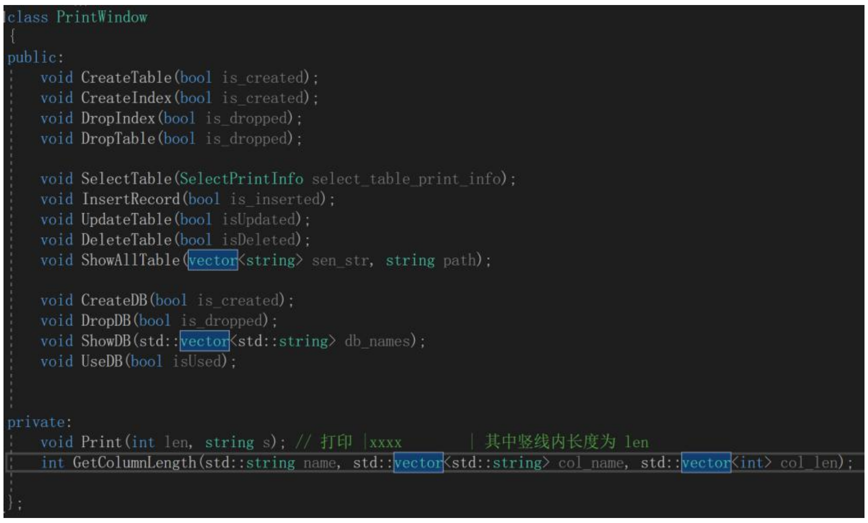
```
TB_Create_Info CreateTableInfo(std::vector<std::string> sen_str);
std::string DropTableInfo(std::vector<std::string> sen_str);
TB_Insert_Info CreateInsertInfo(std::vector<std::string> sen_str);
TB_Select_Info TableSelectInfo(std::vector<std::string> sen_str);
TB_Delete_Info TableDeleteInfo(std::vector<std::string> sen_str);
TB_Update_Info TableUpdateInfo(std::vector<std::string> sen_str);
Idx_Create_Info CreateIndexInfo(std::vector<std::string> sen_str);
std::string DropIndexInfo(std::vector<std::string> sen_str);
```

四、 设计思路

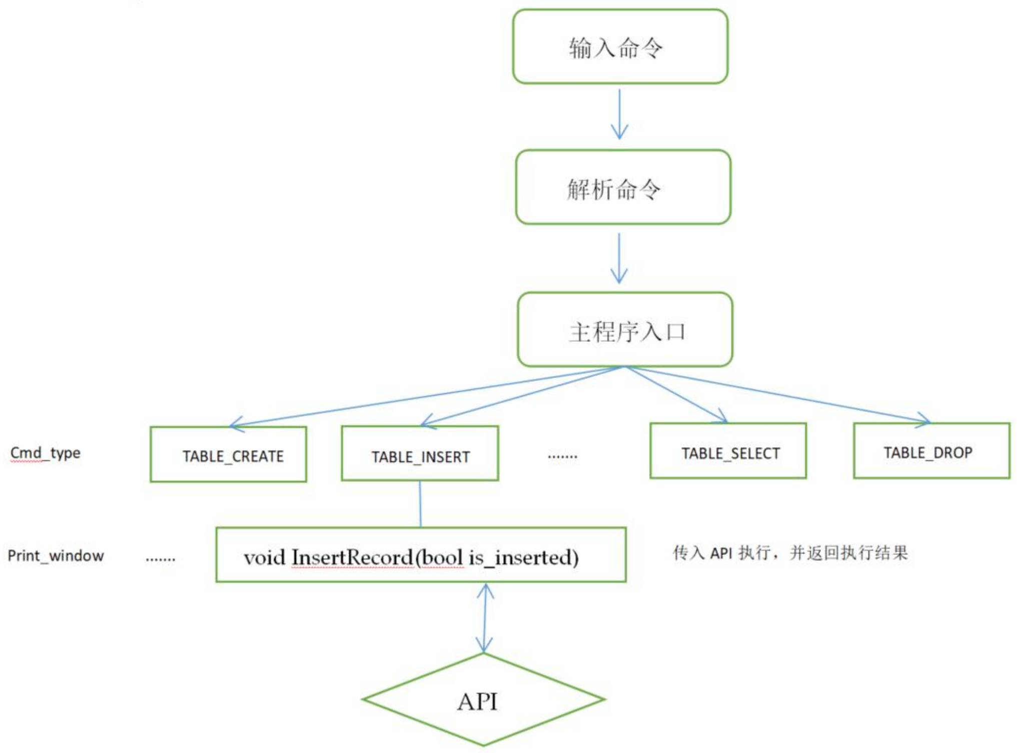
首先我们需要一个类 SensefulStr 来接受处理字符串，另一个类 PrintWindow 用来输出执行结果例：输入语句 insert into student values (‘12345678’,’wy’,22,’M’) 为原始字符串 scr_str，解析后得到存有”insert”,”into”,”student”,”values”,”12345678”,”wy”,”22”,”M” 的有意字符串 sen_str, 根据 sen_str 的第 0 个确定命令类型 cmd_type 为 TABLE_INSERT。然后，将刚刚得到的 sen_str, cmd_type，和定义的 print_window 传入 Interpreter 函数。根据其 cmd_type 执行 print_window 里不同的子函数



```
/***********将命令字符串解析为有意字串***************/
class SensefulStr
{
public:
    SensefulStr();
    void SetStr(string _srcstr);
    vector<string> GetSensefulStr()const;

private:
    void Parse();                                   // 解析字符串
    string src_str;                                 // 原始字符串
    vector<string> sen_str;                         // 解析后字符串
    string key_char = ";, ()=<>\012\015\040";
    bool IsKeyChar(char c);
};

// 主程序的交互接口 输入
void Interpreter(vector<string> sen_str, CmdType cmd_type, PrintWindow print_window);

CmdType GetOpType(vector<string> sen_str);
```

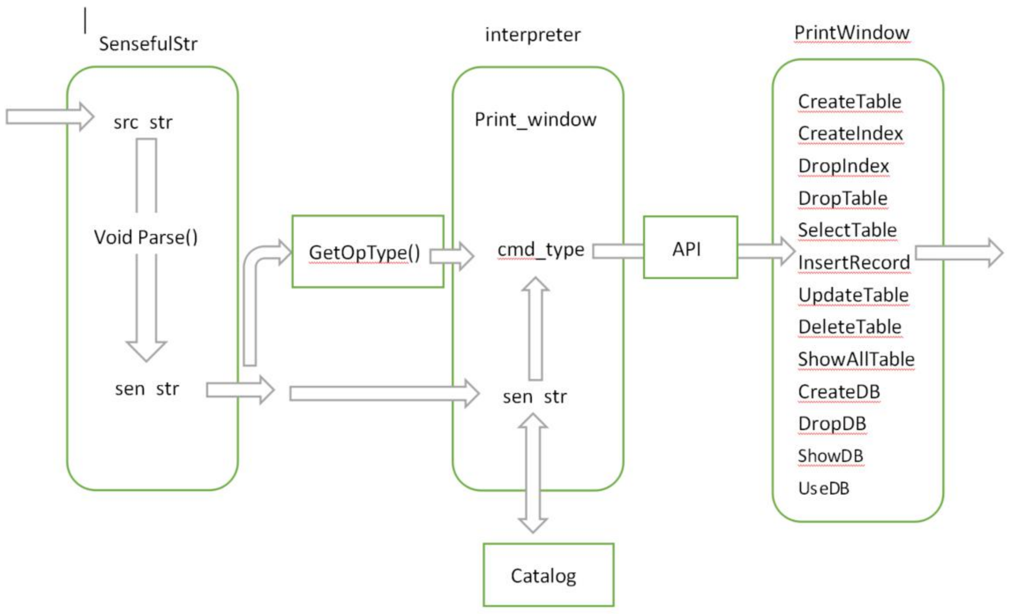


```
class PrintWindow
{
public:
    void CreateTable(bool is_created);
    void CreateIndex(bool is_created);
    void DropIndex(bool is_dropped);
    void DropTable(bool is_dropped);

    void SelectTable(SelectPrintInfo select_table_print_info);
    void InsertRecord(bool is_inserted);
    void UpdateTable(bool isUpdated);
    void DeleteTable(bool isDeleted);
    void ShowAllTable(vector<string> sen_str, string path);

    void CreateDB(bool is_created);
    void DropDB(bool is_dropped);
    void ShowDB(std::vector<std::string> db_names);
    void UseDB(bool isUsed);


private:
    void Print(int len, string s); // 打印 |xxxx        | 其中竖线内长度为 len
    int GetColumnLength(std::string name, std::vector<std::string> col_name, std::vector<int> col_len);

};
```



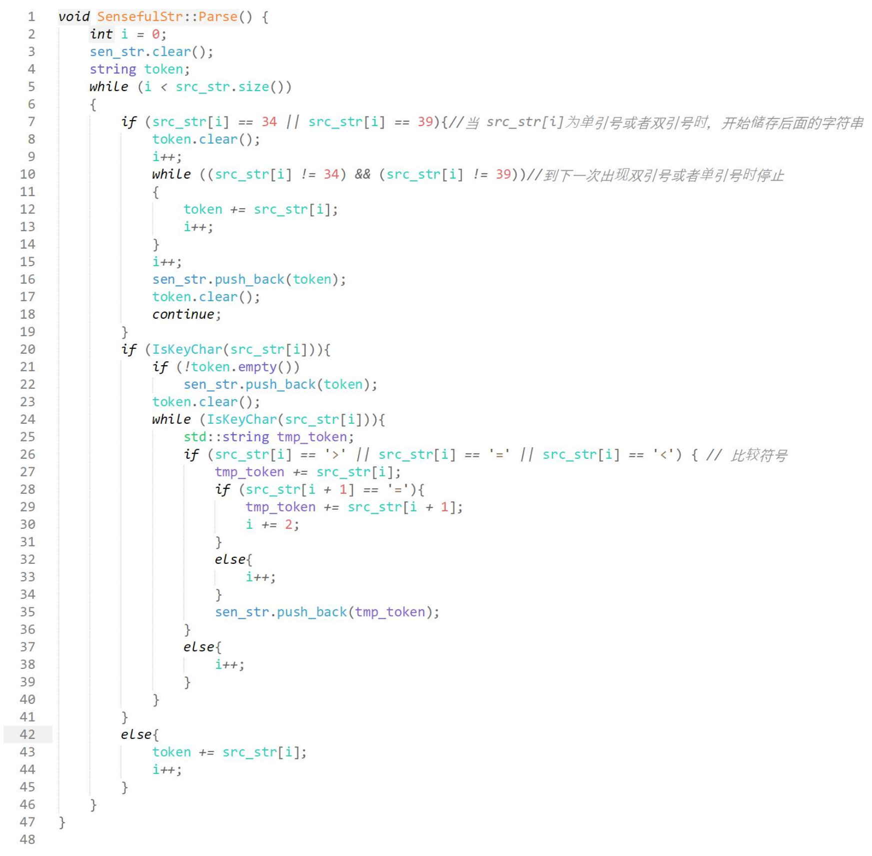
五、 整体架构

在 Interpreter 内，设计一个类 SensefulStr，内部为输入的原始字符串和通过 Parse()函数操作得到的有意字符串。 PrintWindow 则通过 API 里的执行结果输出到界面



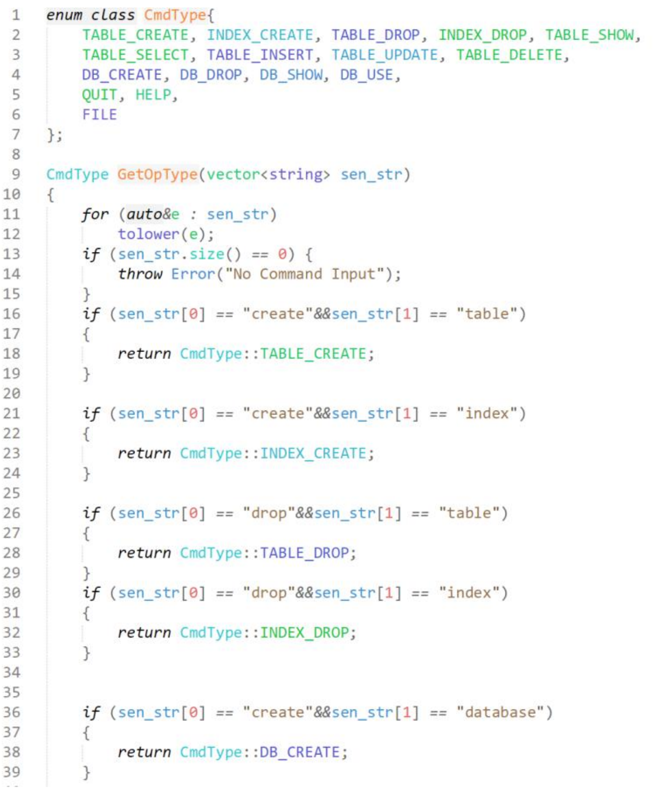
六、 关键函数和代码

SensefulStr 的 Prase()解析函数：、

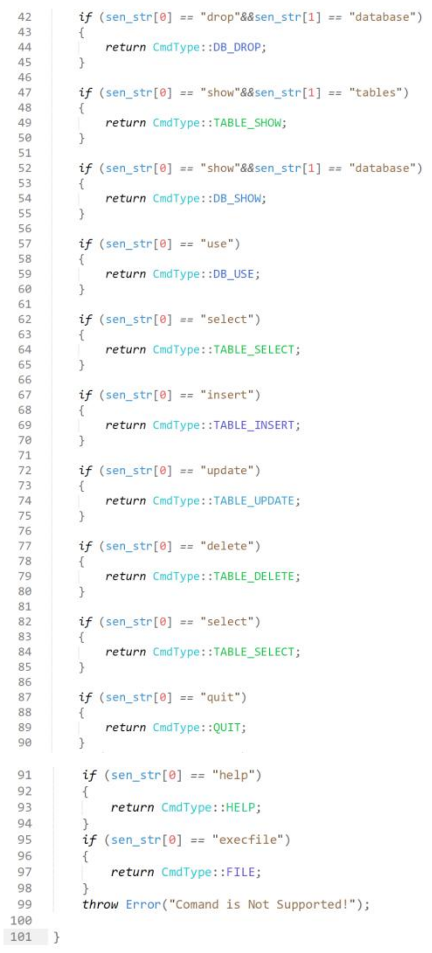


```cpp
void SensefulStr::Parse() {
    int i = 0;
    sen_str.clear();
    string token;
    while (i < src_str.size())
    {
        if (src_str[i] == 34 || src_str[i] == 39){//当 src_str[i]为单引号或者双引号时，开始储存后面的字符串
            token.clear();
            i++;
            while ((src_str[i] != 34) && (src_str[i] != 39))//到下一次出现双引号或者单引号时停止
            {
                token += src_str[i];
                i++;
            }
            i++;
            sen_str.push_back(token);
            token.clear();
            continue;
        }
        if (IsKeyChar(src_str[i])){
            if (!token.empty())
                sen_str.push_back(token);
            token.clear();
            while (IsKeyChar(src_str[i])){
                std::string tmp_token;
                if (src_str[i] == '>' || src_str[i] == '=' || src_str[i] == '<') { // 比较符号
                    tmp_token += src_str[i];
                    if (src_str[i + 1] == '='){
                        tmp_token += src_str[i + 1];
                        i += 2;
                    }
                    else{
                        i++;
                    }
                    sen_str.push_back(tmp_token);
                }
                else{
                    i++;
                }
            }
        }
        else{
            token += src_str[i];
            i++;
        }
    }
}

```

得到命令类型函数 CmdType GetOpType(vector<string> sen_str);

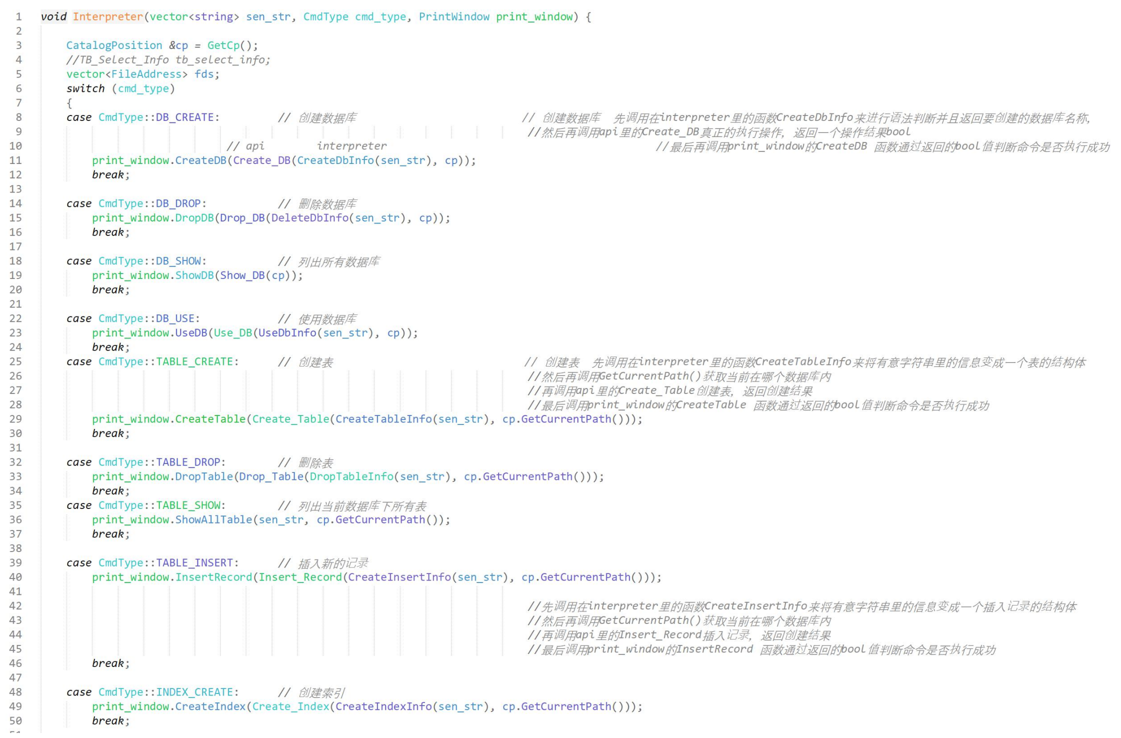


```cpp
enum class CmdType{
    TABLE_CREATE, INDEX_CREATE, TABLE_DROP, INDEX_DROP, TABLE_SHOW,
    TABLE_SELECT, TABLE_INSERT, TABLE_UPDATE, TABLE_DELETE,
    DB_CREATE, DB_DROP, DB_SHOW, DB_USE,
    QUIT, HELP,
    FILE
};

CmdType GetOpType(vector<string> sen_str)
{
    for (auto&e : sen_str)
        tolower(e);
    if (sen_str.size() == 0) {
        throw Error("No Command Input");
    }
    if (sen_str[0] == "create"&&sen_str[1] == "table")
    {
        return CmdType::TABLE_CREATE;
    }

    if (sen_str[0] == "create"&&sen_str[1] == "index")
    {
        return CmdType::INDEX_CREATE;
    }

    if (sen_str[0] == "drop"&&sen_str[1] == "table")
    {
        return CmdType::TABLE_DROP;
    }
    if (sen_str[0] == "drop"&&sen_str[1] == "index")
    {
        return CmdType::INDEX_DROP;
    }


    if (sen_str[0] == "create"&&sen_str[1] == "database")
    {
        return CmdType::DB_CREATE;
    }
```



```cpp
    if (sen_str[0] == "drop"&&sen_str[1] == "database")
    {
        return CmdType::DB_DROP;
    }

    if (sen_str[0] == "show"&&sen_str[1] == "tables")
    {
        return CmdType::TABLE_SHOW;
    }

    if (sen_str[0] == "show"&&sen_str[1] == "database")
    {
        return CmdType::DB_SHOW;
    }

    if (sen_str[0] == "use")
    {
        return CmdType::DB_USE;
    }

    if (sen_str[0] == "select")
    {
        return CmdType::TABLE_SELECT;
    }

    if (sen_str[0] == "insert")
    {
        return CmdType::TABLE_INSERT;
    }

    if (sen_str[0] == "update")
    {
        return CmdType::TABLE_UPDATE;
    }

    if (sen_str[0] == "delete")
    {
        return CmdType::TABLE_DELETE;
    }

    if (sen_str[0] == "select")
    {
        return CmdType::TABLE_SELECT;
    }

    if (sen_str[0] == "quit")
    {
        return CmdType::QUIT;
    }
    if (sen_str[0] == "help")
    {
        return CmdType::HELP;
    }
    if (sen_str[0] == "execfile")
    {
        return CmdType::FILE;
    }
    throw Error("Comand is Not Supported!");

}
```

主程序交接入口



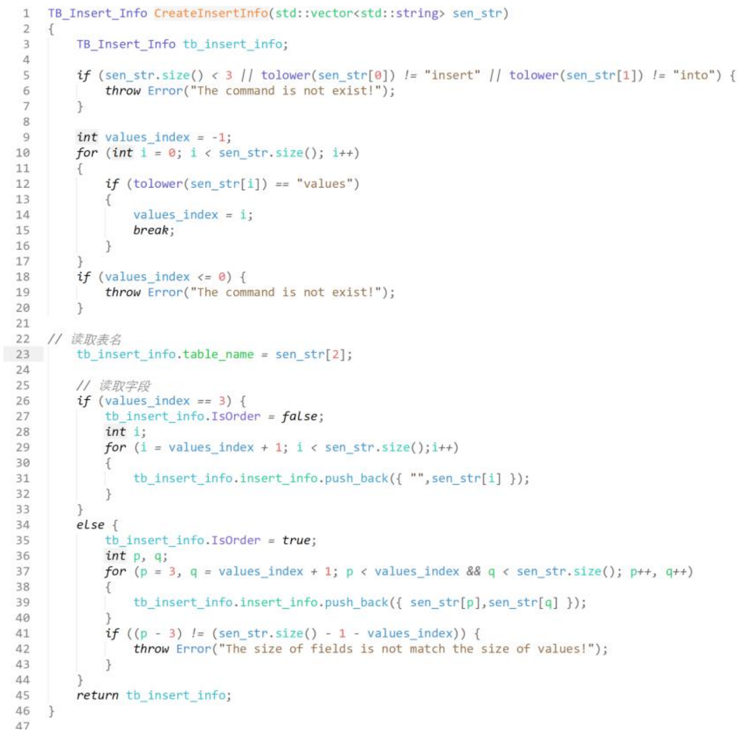
```cpp
void Interpreter(vector<string> sen_str, CmdType cmd_type, PrintWindow print_window) {

    CatalogPosition &cp = GetCp();
    //TB_Select_Info tb_select_info;
    vector<FileAddress> fds;
    switch (cmd_type)
    {
    case CmdType::DB_CREATE:        // 创建数据库                          // 创建数据库  先调用在interpreter里的函数CreateDbInfo来进行语法判断并且返回要创建的数据库名称,
                                                                           //然后再调用api里的Create_DB真正的执行操作, 返回一个操作结果bool
                                // api        interpreter                                  //最后再调用print_window的CreateDB 函数通过返回的bool值判断命令是否执行成功
        print_window.CreateDB(Create_DB(CreateDbInfo(sen_str), cp));
        break;

    case CmdType::DB_DROP:          // 删除数据库
        print_window.DropDB(Drop_DB(DeleteDbInfo(sen_str), cp));
        break;

    case CmdType::DB_SHOW:          // 列出所有数据库
        print_window.ShowDB(Show_DB(cp));
        break;

    case CmdType::DB_USE:           // 使用数据库
        print_window.UseDB(Use_DB(UseDbInfo(sen_str), cp));
        break;
    case CmdType::TABLE_CREATE:     // 创建表                              // 创建表  先调用在interpreter里的函数CreateTableInfo来将有意字符串里的信息变成一个表的结构体
                                                                           //然后再调用GetCurrentPath()获取当前在哪个数据库内
                                                                           //再调用api里的Create_Table创建表, 返回创建结果
                                                                           //最后调用print_window的CreateTable 函数通过返回的bool值判断命令是否执行成功
        print_window.CreateTable(Create_Table(CreateTableInfo(sen_str), cp.GetCurrentPath()));
        break;

    case CmdType::TABLE_DROP:       // 删除表
        print_window.DropTable(Drop_Table(DropTableInfo(sen_str), cp.GetCurrentPath()));
        break;
    case CmdType::TABLE_SHOW:       // 列出当前数据库下所有表
        print_window.ShowAllTable(sen_str, cp.GetCurrentPath());
        break;

    case CmdType::TABLE_INSERT:     // 插入新的记录
        print_window.InsertRecord(Insert_Record(CreateInsertInfo(sen_str), cp.GetCurrentPath()));

                                                                           //先调用在interpreter里的函数CreateInsertInfo来将有意字符串里的信息变成一个插入记录的结构体
                                                                           //然后再调用GetCurrentPath()获取当前在哪个数据库内
                                                                           //再调用api里的Insert_Record插入记录, 返回创建结果
                                                                           //最后调用print_window的InsertRecord 函数通过返回的bool值判断命令是否执行成功
        break;

    case CmdType::INDEX_CREATE:     // 创建索引
        print_window.CreateIndex(Create_Index(CreateIndexInfo(sen_str), cp.GetCurrentPath()));
        break;

    case CmdType::INDEX_DROP:        // 删除索引
        print_window.DropIndex(Drop_Index(DropIndexInfo(sen_str), cp.GetCurrentPath()));
        break;


    case CmdType::TABLE_SELECT:      // 选择表的特定记录
        print_window.SelectTable(Select_Record(TableSelectInfo(sen_str), cp.GetCurrentPath()));
        GetGlobalTimer().PrintTime();
        break;


    case CmdType::TABLE_UPDATE:      // 更新表的记录
        print_window.UpdateTable(Update_Record(TableUpdateInfo(sen_str), cp.GetCurrentPath()));
        break;

    case CmdType::TABLE_DELETE:      // 删除表的记录
        print_window.DeleteTable(Delete_Record(TableDeleteInfo(sen_str), cp.GetCurrentPath()));
        break;

    default:
        throw Error("CmdType Error");
        break;
    }

}

```

Interpreter 和 Catalog 间的接口：

1. 储存建表信息

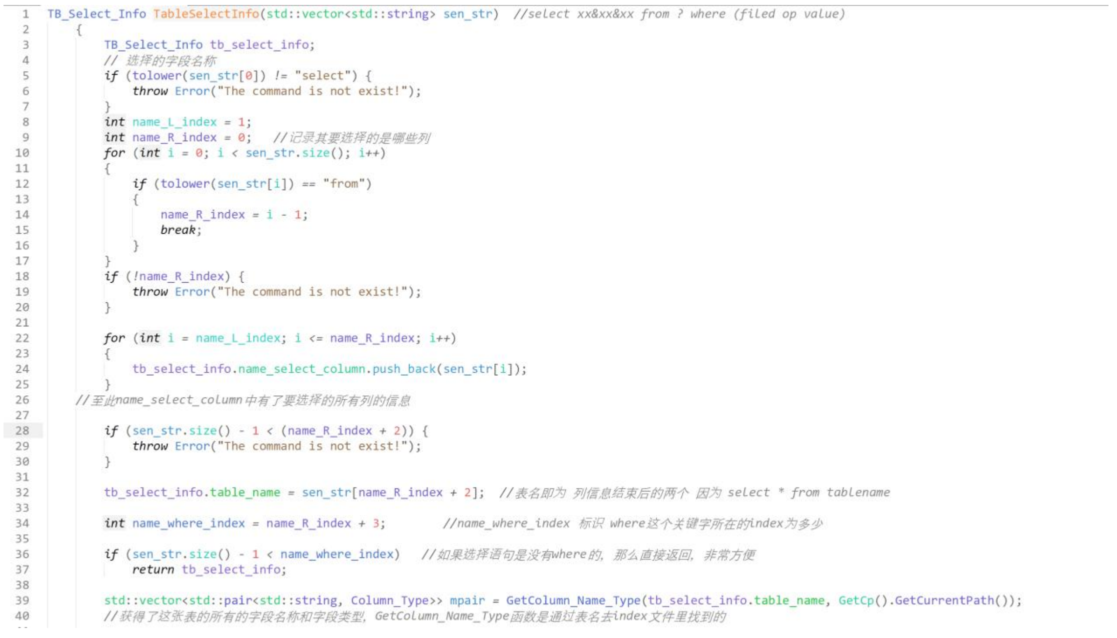
```cpp
TB_Create_Info CreateTableInfo(std::vector<std::string> sen_str)
{
    TB_Create_Info tb_create_info;
    if (sen_str.size() < 3 || tolower(sen_str[0]) != "create" || tolower(sen_str[1]) != "table") {
        throw Error("The command is not exist!");
    }
// 表名
    tb_create_info.table_name = sen_str[2];
    bool HasPrimary = false;
    // 添加各个字段
    for (int j = 3; j < sen_str.size();)   //sen_str 格式如下： create table 'name' 类型1 名字1 ……
    {
        ColumnInfo column_info;
        column_info.isPrimary = false;
        column_info.isUnique = false;
        // 列名
        column_info.name = sen_str[j];
        // 列类型
        if (j + 1 >= sen_str.size()) {
            throw Error("The command is not exist!");
        }

        if (tolower(sen_str[j + 1]) == "int")    // int num
        {
            column_info.type = Column_Type::I;
            column_info.RequiredLength = sizeof(int);
            j += 2;
        }
        else if (tolower(sen_str[j + 1]) == "float")
        {
            column_info.type = Column_Type::F;
            column_info.RequiredLength = sizeof(double);
            j += 2;
        }
        else if (tolower(sen_str[j + 1]) == "char")
        {
            column_info.type = Column_Type::C;
            if (j + 2 >= sen_str.size()) {
                throw Error("The command is not exist!");
            }
            column_info.RequiredLength = stoi(sen_str[j + 2]);   //因为其输入格式为 char name(10)
            if (column_info.RequiredLength >= 255 || column_info.RequiredLength <= 0) {
                throw Error("The length of char is not supported!");
            }
            j += 3;
        }
        else
        {
            throw Error("Unsupported data types!");
        }


        // 是否Unique
        if (j < sen_str.size() && (sen_str[j] == "unique"))
        {
                column_info.isUnique = true;
                j++;
        }
        tb_create_info.columns_info.push_back(column_info);
        // 是否主键
        if (j < sen_str.size() && (sen_str[j] == "primary" )&& (sen_str[j+1] == "key"))
        {
            //cout << sen_str[j] << sen_str[j + 1] << sen_str[j + 2] << endl;
            if (HasPrimary)
                throw Error("More than one primary key!");

            HasPrimary = true;
            for (int i = 0; i < tb_create_info.columns_info.size(); i++)
            {
                if (tb_create_info.columns_info[i].name == sen_str[j+2])
                {
                    tb_create_info.columns_info[i].isPrimary = true;

                }
            }
            j+=3;
            //column_info.isPrimary = true;
        }
    }
    if (!HasPrimary)
        tb_create_info.columns_info[0].isPrimary = true;  // 默认输入的第一个字段为主键

    return tb_create_info;
}
```

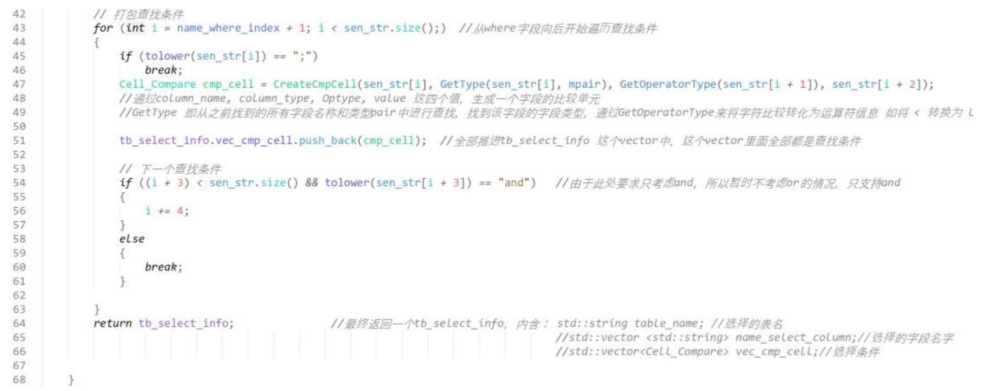
2. 储存插入信息



```cpp
TB_Insert_Info CreateInsertInfo(std::vector<std::string> sen_str)
{
    TB_Insert_Info tb_insert_info;

    if (sen_str.size() < 3 || tolower(sen_str[0]) != "insert" || tolower(sen_str[1]) != "into") {
        throw Error("The command is not exist!");
    }

    int values_index = -1;
    for (int i = 0; i < sen_str.size(); i++)
    {
        if (tolower(sen_str[i]) == "values")
        {
            values_index = i;
            break;
        }
    }
    if (values_index <= 0) {
        throw Error("The command is not exist!");
    }

// 读取表名
    tb_insert_info.table_name = sen_str[2];

    // 读取字段
    if (values_index == 3) {
        tb_insert_info.IsOrder = false;
        int i;
        for (i = values_index + 1; i < sen_str.size();i++)
        {
            tb_insert_info.insert_info.push_back({ "",sen_str[i] });
        }
    }
    else {
        tb_insert_info.IsOrder = true;
        int p, q;
        for (p = 3, q = values_index + 1; p < values_index && q < sen_str.size(); p++, q++)
        {
            tb_insert_info.insert_info.push_back({ sen_str[p],sen_str[q] });
        }
        if ((p - 3) != (sen_str.size() - 1 - values_index)) {
            throw Error("The size of fields is not match the size of values!");
        }
    }
    return tb_insert_info;
}
```

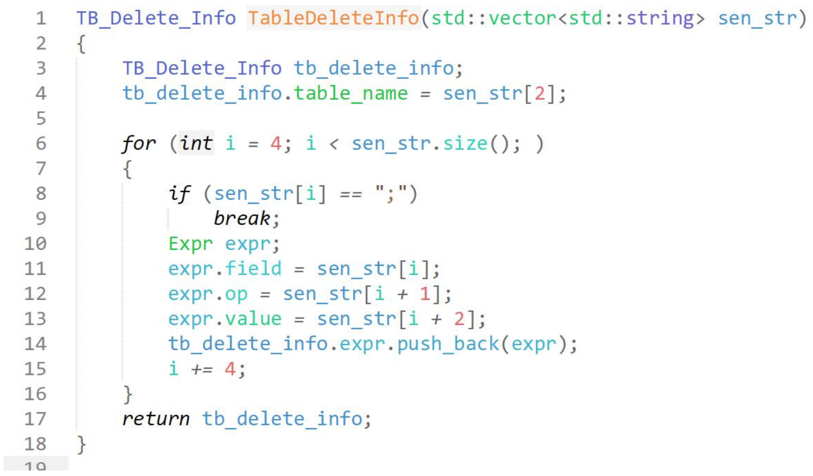
3. 储存选择信息





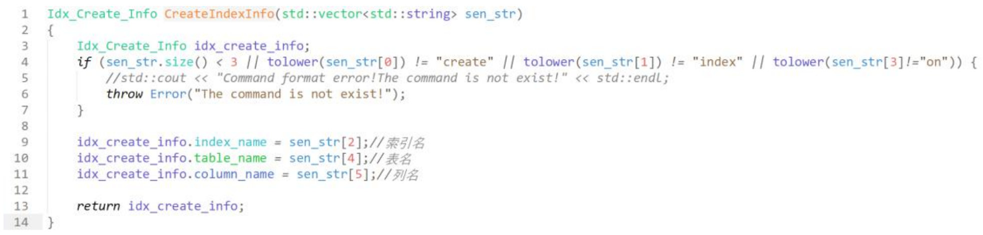
```cpp
TB_Select_Info TableSelectInfo(std::vector<std::string> sen_str)  //select xx&xx&xx from ? where (filed op value)
    {
        TB_Select_Info tb_select_info;
        // 选择的字段名称
        if (tolower(sen_str[0]) != "select") {
            throw Error("The command is not exist!");
        }
        int name_L_index = 1;
        int name_R_index = 0;   //记录其要选择的是哪些列
        for (int i = 0; i < sen_str.size(); i++)
        {
            if (tolower(sen_str[i]) == "from")
            {
                name_R_index = i - 1;
                break;
            }
        }
        if (!name_R_index) {
            throw Error("The command is not exist!");
        }

        for (int i = name_L_index; i <= name_R_index; i++)
        {
            tb_select_info.name_select_column.push_back(sen_str[i]);
        }
    //至此name_select_column中有了要选择的所有列的信息

        if (sen_str.size() - 1 < (name_R_index + 2)) {
            throw Error("The command is not exist!");
        }

        tb_select_info.table_name = sen_str[name_R_index + 2];  //表名即为 列信息结束后的两个 因为 select * from tablename

        int name_where_index = name_R_index + 3;        //name_where_index 标识 where这个关键字所在的index为多少

        if (sen_str.size() - 1 < name_where_index)   //如果选择语句是没有where的, 那么直接返回, 非常方便
            return tb_select_info;

        std::vector<std::pair<std::string, Column_Type>> mpair = GetColumn_Name_Type(tb_select_info.table_name, GetCp().GetCurrentPath());
        //获得了这张表的所有的字段名称和字段类型, GetColumn_Name_Type函数是通过表名去index文件里找到的

        // 打包查找条件
        for (int i = name_where_index + 1; i < sen_str.size();)  //从where字段向后开始遍历查找条件
        {
            if (tolower(sen_str[i]) == ";")
                break;
            Cell_Compare cmp_cell = CreateCmpCell(sen_str[i], GetType(sen_str[i], mpair), GetOperatorType(sen_str[i + 1]), sen_str[i + 2]);
            //通过column_name, column_type, Optype, value 这四个值, 生成一个字段的比较单元
            //GetType 即从之前找到的所有字段名称和类型pair中进行查找, 找到该字段的字段类型, 通过GetOperatorType来将字符比较转化为运算符信息 如将 < 转换为 L

            tb_select_info.vec_cmp_cell.push_back(cmp_cell);  //全部推进tb_select_info 这个vector中, 这个vector里面全部都是查找条件

            // 下一个查找条件
            if ((i + 3) < sen_str.size() && tolower(sen_str[i + 3]) == "and")   //由于此处要求只考虑and, 所以暂时不考虑or的情况, 只支持and
            {
                i += 4;
            }
            else
            {
                break;
            }

        }
        return tb_select_info;              //最终返回一个tb_select_info, 内含: std::string table_name; //选择的表名
                                                                            //std::vector <std::string> name_select_column;//选择的字段名字
                                                                            //std::vector<Cell_Compare> vec_cmp_cell;//选择条件

    }
```

4. 储存删除信息



```cpp
TB_Delete_Info TableDeleteInfo(std::vector<std::string> sen_str)
{
    TB_Delete_Info tb_delete_info;
    tb_delete_info.table_name = sen_str[2];

    for (int i = 4; i < sen_str.size(); )
    {
        if (sen_str[i] == ";")
            break;
        Expr expr;
        expr.field = sen_str[i];
        expr.op = sen_str[i + 1];
        expr.value = sen_str[i + 2];
        tb_delete_info.expr.push_back(expr);
        i += 4;
    }
    return tb_delete_info;
}
```

5. 储存索引创建信息



```cpp
Idx_Create_Info CreateIndexInfo(std::vector<std::string> sen_str)
{
    Idx_Create_Info idx_create_info;
    if (sen_str.size() < 3 || tolower(sen_str[0]) != "create" || tolower(sen_str[1]) != "index" || tolower(sen_str[3]!="on")) {
        //std::cout << "Command format error!The command is not exist!" << std::endl;
        throw Error("The command is not exist!");
    }

    idx_create_info.index_name = sen_str[2];//索引名
    idx_create_info.table_name = sen_str[4];//表名
    idx_create_info.column_name = sen_str[5];//列名

    return idx_create_info;
}
```