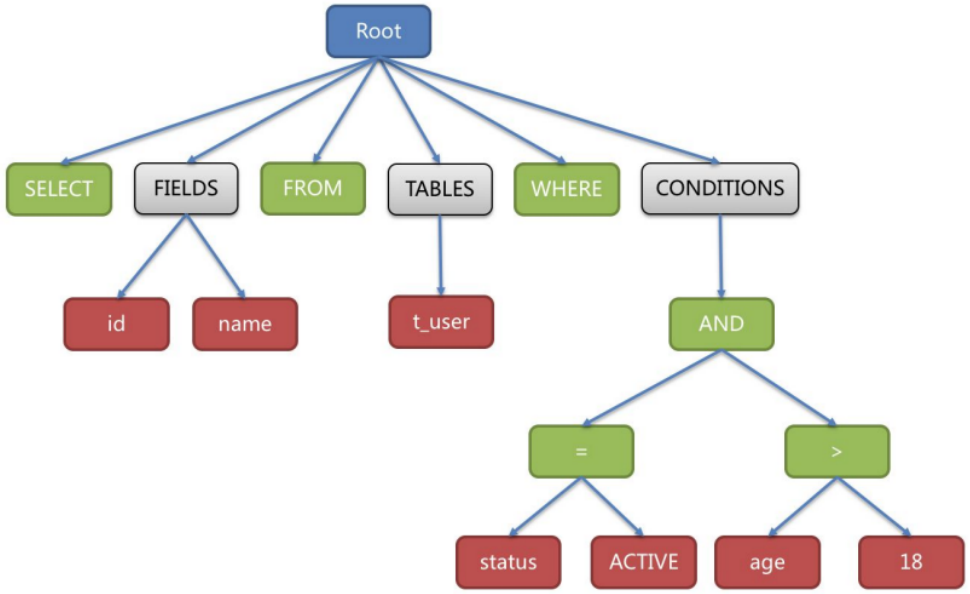
DuckDB核心流程设计分析

本节的内容是对选取的主要功能的类及类间关系设计的分析

DuckDB的SQL parser实际上基于PostgresSQL解析器实现，并对其尽可能地进行了简化。这是因为重新实现一个鲁棒性高的parser是很困难的。

parser模块接受一个SQL查询字符串作为输入，并且返回一个c语言结构体组成的解析树。随后被转换为DuckDB自己的c++类的解析树，其结构如下图所示:



主要类和对象

1. **Parser**
   - Parser 类是解析 SQL 查询的核心类。它包含了 ParseQuery 方法，用于解析输入的 SQL 查询字符串。
2. **Transformer**
   - Transformer 类用于将解析树转换为 SQL 语句对象。它在 ParseQuery 方法中被实例化，并调用其 TransformParseTree 方法来处理解析树。
3. **PostgresParser**
   - PostgresParser 类用于实际解析 SQL 查询字符串。它在 ParseQuery 方法中被实例化，并调用其 Parse 方法来解析查询字符串。
4. **SQLStatement**
   - SQLStatement 类表示解析后的 SQL 语句。解析成功后，生成的 SQL 语句对象会存储在 Parser 类的 statements 向量中。
5. **ParserOptions**
   - ParserOptions 类用于配置解析器的选项，如是否保留标识符大小写、是否启用扩展等。Parser 类在构造函数中接受一个 ParserOptions 对象，并在解析过程中使用这些选项。

6. **ExtensionStatement**
    - ExtensionStatement 类表示通过扩展解析的 SQL 语句。当 PostgresParser无法解析某个查询语句时，会尝试使用扩展来解析，并生成 ExtensionStatement 对象。
7. **ErrorData**
    - ErrorData 类用于存储解析错误的信息。当解析失败时，会生成 ErrorData 对象来记录错误信息和位置。

执行流程

初始化和 Unicode 空格检查

```C++
Transformer transformer(options);
string parser_error;
optional_idx parser_error_location;
{
    string new_query;
    if (StripUnicodeSpaces(query, new_query)) {
        ParseQuery(new_query);
        return;
    }
}
```

设置解析器选项并解析查询

```C++
PostgresParser::SetPreserveIdentifierCase(options.preserve_id
entifier_case);
bool parsing_succeed = false;
{
    PostgresParser parser;
    parser.Parse(query);
    if (parser.success) {
        if (!parser.parse_tree) {
            return;
        }
        transformer.TransformParseTree(parser.parse_tree,
statements);
        parsing_succeed = true;
    } else {
        parser_error = parser.error_message;
        if (parser.error_location > 0) {
            parser_error_location = NumericCast<idx_t>
(parser.error_location - 1);
        }
    }
}
```

解析成功，将解析树转换为 SQL 语句

如果解析全部成功，更新Statements

```C++
if (!statements.empty()) {
    auto &last_statement = statements.back();
    last_statement->stmt_length = query.size() -
last_statement->stmt_location;
    for (auto &statement : statements) {
        statement->query = query;
        if (statement->type ==
StatementType::CREATE_STATEMENT) {
            auto &create = statement->Cast<CreateStatement>
();
            create.info->sql = query.substr(statement-
>stmt_location, statement->stmt_length);
        }
    }
}
```

解析失败，转入错误处理

对于解析失败的情况，又会根据是否有拓展方法进行不同的处理

C++

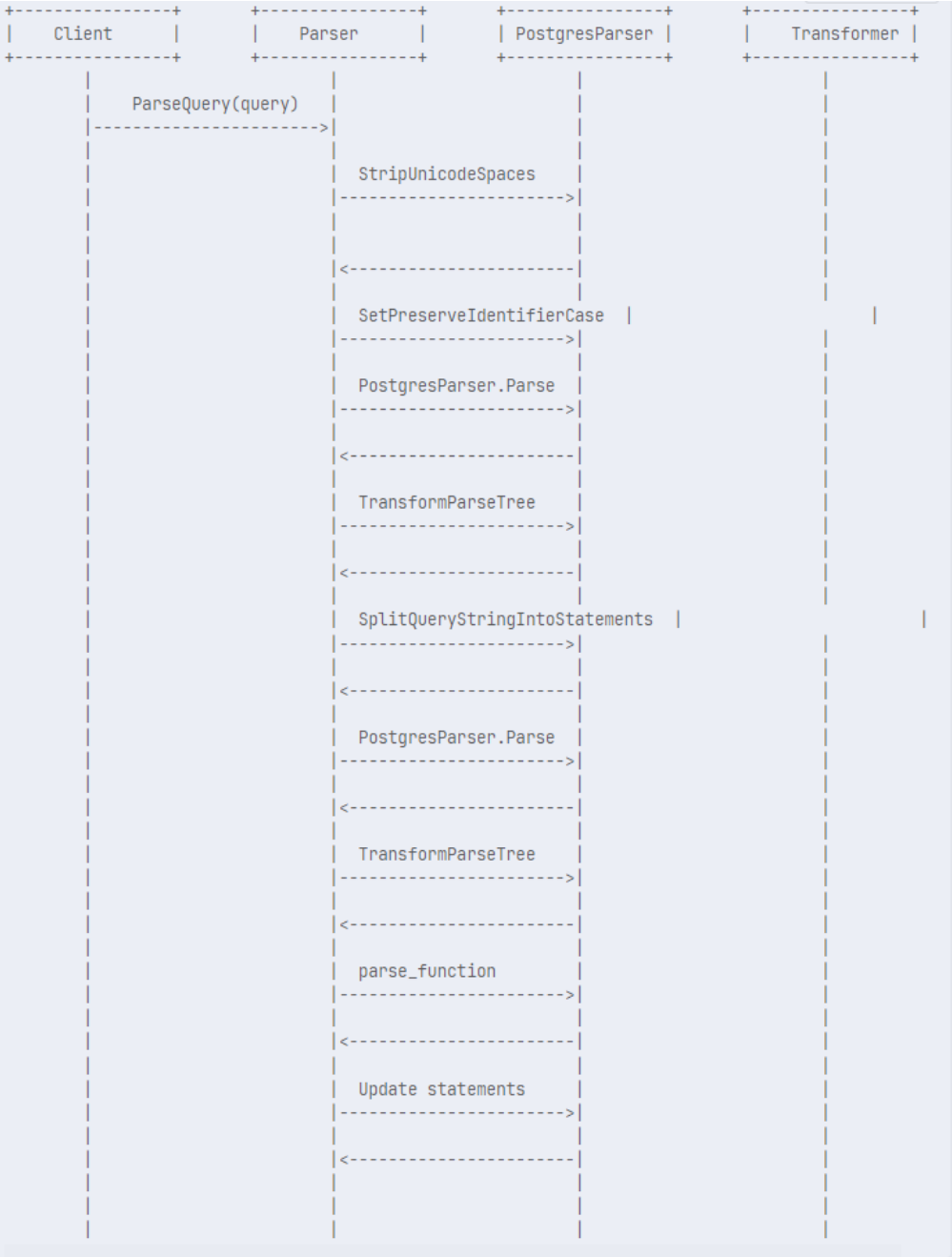
```
if (parsing_succeed) {
    // no-op
} else if (!options.extensions || options.extensions-
>empty()) {
    throw ParserException::SyntaxError(query, parser_error,
parser_error_location);
} else {
    auto query_statements =
SplitQueryStringIntoStatements(query);
    idx_t stmt_loc = 0;
    for (auto const &query_statement : query_statements) {
        ErrorData another_parser_error;
        {
            PostgresParser another_parser;
            another_parser.Parse(query_statement);
            if (another_parser.success) {
                if (!another_parser.parse_tree) {
                    continue;
                }

transformer.TransformParseTree(another_parser.parse_tree,
statements);
                statements.back()->stmt_length =
query_statement.size() - 1;
                statements.back()->stmt_location = stmt_loc;
                stmt_loc += query_statement.size();
                continue;
            } else {
                another_parser_error =
ErrorData(another_parser.error_message);
                if (another_parser.error_location > 0) {

another_parser_error.AddQueryLocation(NumericCast<idx_t>
(another_parser.error_location - 1));
                }
            }
        }
        bool parsed_single_statement = false;
        for (auto &ext : *options.extensions) {
            auto result =
ext.parse_function(ext.parser_info.get(), query_statement);
            if (result.type ==
```

```
ParserExtensionResultType::PARSE_SUCCESSFUL) {
                auto statement =
make_uniq<ExtensionStatement>(ext,
std::move(result.parse_data));
                statement->stmt_length =
query_statement.size() - 1;
                statement->stmt_location = stmt_loc;
                stmt_loc += query_statement.size();
                statements.push_back(std::move(statement));
                parsed_single_statement = true;
                break;
            } else if (result.type ==
ParserExtensionResultType::DISPLAY_EXTENSION_ERROR) {
                throw ParserException::SyntaxError(query,
result.error, result.error_location);
            }
        }
        if (!parsed_single_statement) {
            throw ParserException::SyntaxError(query,
parser_error, parser_error_location);
        }
    }
}
```

时序图


Client
Parser
PostgresParser
Transformer
ParseQuery(query)
StripUnicodeSpaces
SetPreserveIdentifierCase
PostgresParser.Parse
TransformParseTree
SplitQueryStringIntoStatements
PostgresParser.Parse
TransformParseTree
parse_function
Update statements