数据库内核添加查询语言功能点工作流程

0 引言

本文将对如何在数据库系统中添加查询语言功能点进行简单介绍。首先需要明确不同的系统之间具有差异，但总体来说，添加功能点的整体流程将会被分为以下几个部分：

词法分析→语法分析→语义分析→计划树生成→计划执行

图 1 添加查询功能点整体流程

上述过程以 SPARQL BGP 查询为例，图示如下：

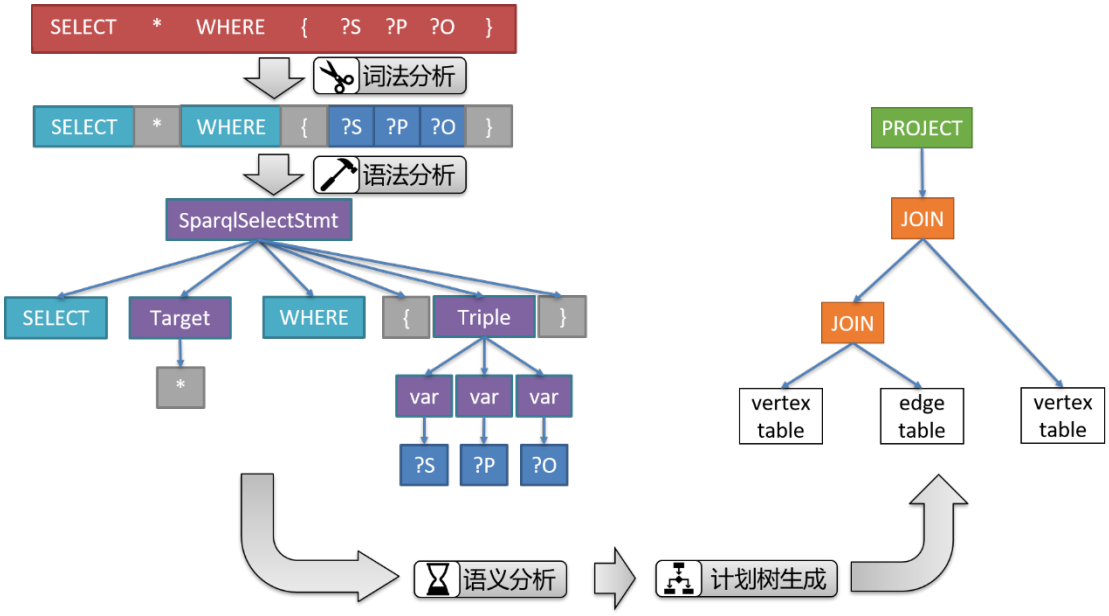

图 2 查询执行过程

其中，语义分析和计划树生成过程往往同时完成。语义分析部分关注“符合词法、语法定义，但查询语句不符合正常查询意图”的语句，例如：出现在结果子句中的变量，没有在 WHERE 子句中进行定义等。而计划树生成过程中，需要确保语句符合正常查询意图，并将查询语句的语义转化为对应的数据库底层的算子（如 IndexScan、JOIN 等）。

为了具有普适性，下文的介绍将尽可能不涉及具体系统。

推荐阅读：openGauss 源码解析系列——SQL 引擎源解析（一）

1 词法分析

- **目标文件**：scan.l
- **作用**：根据正则形式规定的分解规则，将查询语句分解为一系列 token
- **常用工具**：flex
- **词法文件基本结构**：

定义起始状态→定义 token 规则→定义 token 动作→函数定义

图 3 词法文件结构

起始状态的使用目的在于区分 token 划分时的不同路径，比如，在同一个系统中需要匹配两种语言，而两种语言涉及到相同关键字，此时可以定义一个特殊的关键字，当遇到这一特殊关键字时，跳入其中一个语言的词法解析过程，而另一语言的词法规则将不会再被匹配，这样就为两种语言开辟了不同的解析路径。

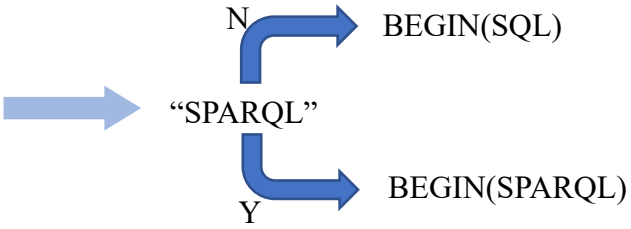

图 4 起始状态

词法文件的关键在于定义 token 的规则，下面先给出正则规则写法：

a	matches a
abc	matches abc
[abc]	matches a, b or c
[a-f]	matches a, b, c, d, e, or f
[0-9]	matches any digit
X+	mathces one or more of X
X*	mathces zero or more of X
[0-9]+	matches any integer
(…)	grouping an expression into a single unit
|	alternation (or)
(a|b|c)*	is equivalent to [a-c]*
X?	X is optional (0 or 1 occurrence)
if(def)?	matches if or ifdef (equivalent to if|ifdef)
[A-Za-z]	matches any alphabetical character
.	matches any character except newline character

图 5 正则表达式

\.	matches the . character
\n	matches the newline character
\t	matches the tab character
\\	matches the \ character
[\t]	matches either a space or tab character
[^a-d]	matches any character other than a,b,c and d

图 6 正则表达式（续）

在词法规则定义中，有一种特殊的规则，将会出现在所有规则的最后面，即“other”，这一规则将匹配所有未定义的 token，写法即“.”。需要注意的是，在任何一个起始状态下，都需要保证若不满足现有规则条件，能够推导至 other 规则，以保证完备性。同理，特殊的规则还有“EOF”、“\n”、“\t”空格符等。

```
355   other           .
```

图 7 词法文件中的 other 规则

在撰写词法动作时，往往需要联合语法文件，确定返回值，如下图中的 COLON_EQUALS 将在对应语法文件中用到。

```
809  {colon_equals}  {
810                      SET_YYLLOC();
811                      yyextra->is_hint_str = false;
812                      return COLON_EQUALS;
813                  }
```

图 8 词法动作

例1. 以 SPARQL_STRING 为例，举例说明词法文件添加过程。

1) 在 gram.y 文件 token 定义部分添加 SPARQL_STRING。token 类型一般为 str。

```
707   %token <str>    BLANK_NODE_LABEL SPARQL_STRING LANGTAG QNAME QNAME_NS
```

图 9 语法文件 token 定义

注：关于 gram 中使用的所有数据类型定义，在 gram.y 文件定义段，大概 200+位置，常用的有 ival、str、boolean、node、list、target。

2) 在 scan.l 文件定义段中添加 token 正则表达式，添加位置应在“other”规则前。同时添加部分匹配成功的错误状态 sparql_string_fail，防止产生 backup。

```
413   sparql_string ([""][^""\\\n]*[""])|([''][^''\\\n]*['']) 
414   sparql_string_fail ([""][^""\\\n]*)|([''][^''\\\n]*)
```

图 10 词法规则定义

注：PG 中强制防止 backup，故每一个词法 token 必须添加错误状态，添加方法类似堆栈回退，即考虑 token 最末尾的符号不能成功匹配时的情况。
参考：flex backup 处理方式

3) 在 scan.l 文件规则段中添加 token 对应词法动作，包括错误状态的动作。

```
1174  <SPARQL,SEMI>{sparql_string}          {
1175                                        SET_YYLLOC();
1176                                        yylval->str = pstrdup(yytext);
1177                                        return SPARQL_STRING;
1178                                    }
1179  <SPARQL,SEMI>{sparql_string_fail}          {
1180                                        yyerror("unterminated sparql string");
1181                                    }
```

图 11 词法动作

注：
a) 词法匹配遵循几个规则：(1) 长规则优先 (2) 最早规则优先，故需要注意 token 动作与其他 token 的规则重复，决定其放置位置。
b) Flex 的起始状态(start state)，允许指定在特定时刻哪些模式可以用来匹配。起始状态在文件开头%x 行定义，任何时刻词法分析器都在一个起始状态中，且只匹配这个状态所激活的模式。SPARQL 的所有功能点凡是 SELECT 引导的都应该在 *SPARQL* 起始状态下定义。

4) 词法部分编译测试，若无报错，且没有产生任何 backup，则词法部分添加成功。

在定义词法动作时，可以考虑使用 flex 提供的某些功能，帮助去除 token 中无意义的部分，例如：字符串中的单引号、双引号等，可能用到的功能有 yyless、yymore 等。

2 语法分析

- **目标文件**：gram.y
- **作用**：添加语法规则，使系统能够识别对应查询语句。
- **常用工具**：bison
- **语法文件结构**：

类型定义→中间状态声明→token 声明→优先级定义→语法规则定义

图 12 语法文件结构

在撰写新的语法功能之前，需要对本系统使用的状态类型进行了解，在语法文件前部，所有系统支持的状态类型可以在以%union 引导的块中找到，在必要时可以向其添加新的类型。

```
%union
{
    core_YYSTYPE        core_yystype;
    /* these fields must match core_YYSTYPE: */
    int                 ival;
    char                *str;
    const char          *keyword;

    char                chr;
    bool                boolean;
    JoinType            jtype;
    DropBehavior        dbehavior;
    OnCommitAction      oncommit;
    List                *list;
    Node                *node;
    Value               *value;
    ObjectType          objtype;
    TypeName            *typnam;
    FunctionParameter   *fun_param;
    FunctionParameterMode fun_param_mode;
    FuncWithArgs        *funwithargs;
    DefElem             *defelt;
    SortBy              *sortby;
    WindowDef           *windef;
    JoinExpr            *jexpr;
    IndexElem           *ielem;
    Alias               *alias;
    RangeVar            *range;
    IntoClause          *into;
    WithClause          *with;
    A_Indices           *aind;
    ResTarget           *target;
    struct PrivTarget   *privtarget;
    AccessPriv          *accesspriv;
    InsertStmt          *istmt;
    VariableSetStmt     *vsetstmt;
/* PGXC_BEGIN */
    DistributeBy        *distby;
    PGXCSubCluster      *subclus;
/* PGXC_END */
    ForeignPartState    *foreignpartby;
    MergeWhenClause     *mergewhen;
    UpsertClause *upsert;
    EncryptionType algtype;
}
```

图 13 类型定义

在所有类型中，需要重点关注的是常见的数据类型：字符 char、整形数字 int、字符串 char*、链表 List 等，除此之外，还有一种特殊的类型 Node，这一类型可以被认为是所有“节点”的父类，在后续的语义处理中通过强制类型转换变为具体节点类型进行处理，而不同节点之间的区分通过 NodeTag 判断（对于节点和 NodeTag 的定义可以在 node.h 中找到）。

在语法分析中，将会涉及到三层结构：语句 statement（或称为句子 sentence）、子句 clause、表达式 expression，单个语句由多个子句组成，单个字句由多个表达式组成，在设计语法规则时应该尽可能遵循上述的结构。

如图 14 所示，给出 SPARQL BGP 查询的语法树：

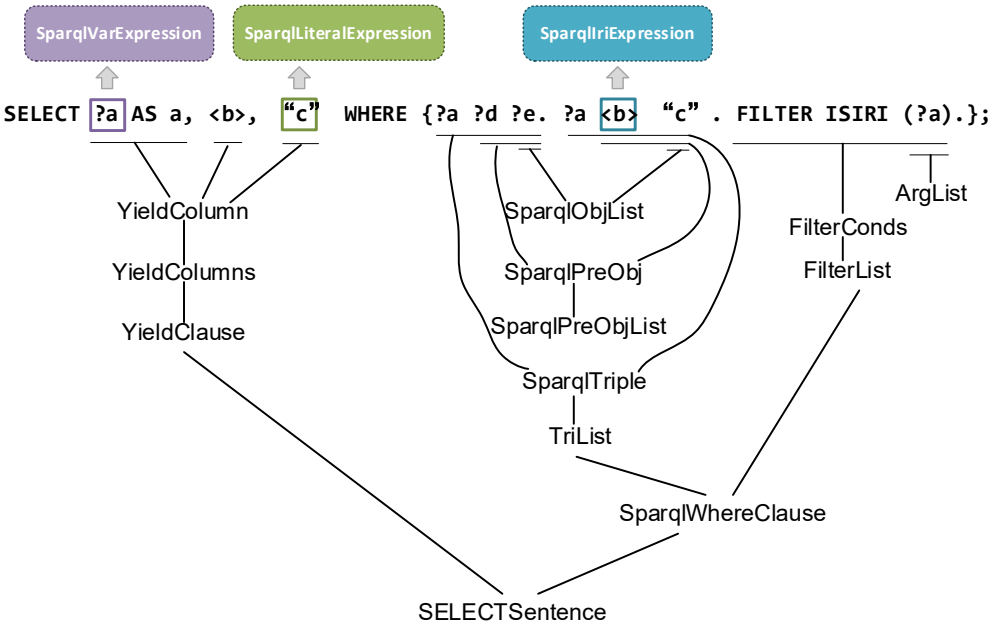

图 14 SPARQL BGP 查询语法树

图 14 中设计的 SPARQL BGP 语法解析方式在不同系统中会有区别，需要关注的是基本结构，结构中的实现细节需要根据不同系统确定。

token 的定义需要结合词法文件，与词法文件中的返回值一致，如图 15 中定义的 token，与图 8 中的返回值一致。

```
700  %token          TYPECAST ORA_JOINOP DOT_DOT COLON_EQUALS PARA_EQUALS
```

图 15 语法文件中的 token 定义

token 总体上可以被分为两类，分别为关键字和普通标识符，普通标识符的定义依靠词法文件和语法文件即可完成解析，而关键字的解析则一般需要联合其他文件进行，例如借助 kwlist 文件中的枚举等。关键字又可以被分为保留关键字和非保留关键字，SQL 语言在 PostgreSQL 系统中的关键字保留性质可以在 PG 中的 SQL 关键字中查到。关键字标识符的类型将被指定为 keyword，所有关键字的列举需要按照字典序。

```
711  %token <keyword> ABORT_P ABSOLUTE_P ACCESS ACCOUNT ACTION ADD_P ADMIN AFTER
712      AGGREGATE ALGORITHM ALL ALSO ALTER ALWAYS ANALYSE ANALYZE AND ANY APP ARCHIVE ARRAY AS ASC
713          ASSERTION ASSIGNMENT ASYMMETRIC AT ATTRIBUTE AUDIT AUTHID AUTHORIZATION AUTOEXTEND AUTOMAPPED AUTOML
```

图 16 关键字定义

在 token 定义之后，将定义不同语句之间的优先级定义，优先级将帮助确定语句具有两种解析方式时确定具体解析方式，常用于数学算符之间。

语法规则定义撰写中，需要遵循一些习惯，如语句对齐方式、最后的分号单独成行、具体规则换行书写等，还需要联合结构体定义进行语法解析树的构建。不同系统语法动作的实现方式差异较大，此处需要符合具体系统的实现习惯。

```
16014  with_clause:
16015          WITH cte_list
16016              {
16017                  $$ = makeNode(WithClause);
16018                  $$->ctes = $2;
16019                  $$->recursive = false;
16020                  $$->location = @1;
16021              }
16022          | WITH RECURSIVE cte_list
16023              {
16024                  $$ = makeNode(WithClause);
16025                  $$->ctes = $3;
16026                  $$->recursive = true;
16027                  $$->location = @1;
16028              }
16029          ;
```

图 17 语法规则举例

对于 SPARQL 的实现，语法推导式的写法可以参照 SPARQL 官方定义。

例2. 语法解析添加过程：

1) 根据查询语句结构，递归的添加语法规则，直至规则中的每一个部分都是 token。

2) 添加语法的规则动作，例如建立某个结构体、链表等。基于 PostgreSQL 的系统常用的函数有：makenode(建立结构体)、list_make1(建立链表)、lappend(头插链表)。

> 注：
> a) 语法动作默认为$$=$1。
> b) 语法动作中的$1/$2 等中的数字指的是语法结构中的顺序，由 1 开始的顺序号。
> 参考：bison 语法动作

3) 语法部分编译测试，无报错，在语义动作为空时在系统中可以识别预定语句，则语法部分添加成功。

> 注：
> a) 当遇到移进/规约冲突、移进/移进冲突时，可以单独编译 yacc 文件，添加-v -t 编译选项，生成状态机文件 gram.output，查看具体哪一句有冲突。冲突的解决往往采用指定语句优先级的方式。
> b) 因 PG 后续编译过程会由 gram.y 自动生成其他文件，所以需要严格遵守语句格式，注意大括号、冒号等位置。

3 结构体定义

结构体的定义应该与实现逻辑相对应，在不同系统上的实现方法会有不同之处，但总体可以参照图 18 中的设计，其中虚线框内的结构体为关键字查询、PageRank 查询、RPQ 查询所需要的结构体，在实现 BGP 功能时可以不关注。绿色实线内的结构体借助了系统内自带的定义，未展开给出。一般来说，在实现 BGP 查询中，结果子句和聚类函数的实现可以参照系统中原有的实现方式。

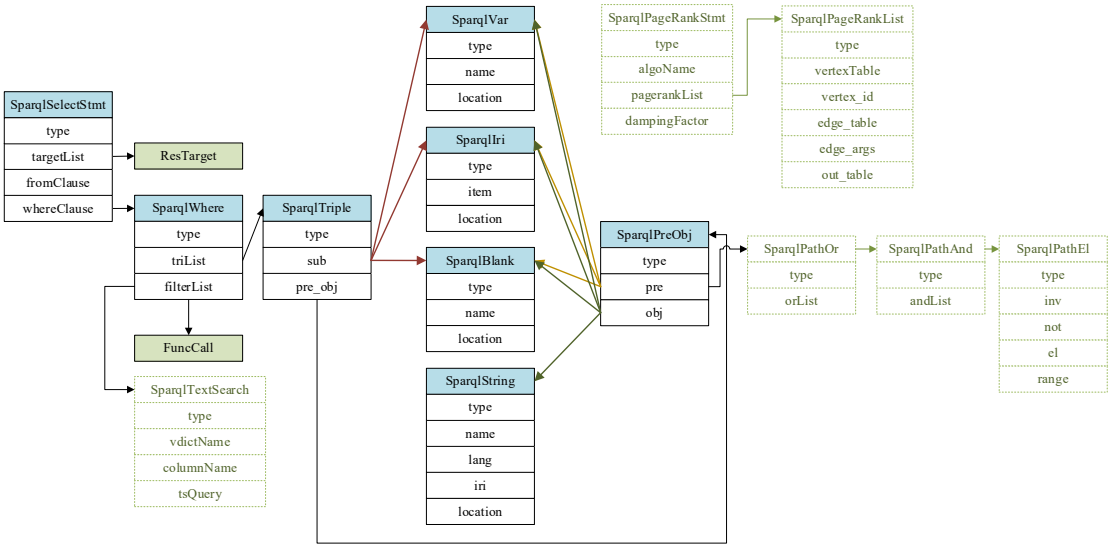

图 18 SPARQL 结构体

例3. 结构体添加

1) 根据需要记录的内容，修改 parsernodes.h 中的结构体。如果在其中添加了新的结构体，则在 node.h 枚举类型中添加 NodeTag 条目。

```
414     T_VacuumStmt,
415     T_ExplainStmt,
416     T_CreateTableAsStmt,
417     T_CreateSeqStmt,
```

图 19 node. h 中的 NodeTag 枚举类型

```
1993  typedef struct ExplainStmt {
1994      NodeTag type;
1995      Node* statement; /* statement_id for EXPLAIN PLAN */
1996      Node* query;     /* the query (see comments above) */
1997      List* options;   /* list of DefElem nodes */
1998      PlanInformation* planinfo;
1999  } ExplainStmt;
```

图 20 parsernode.h 中的结构体定义

2) 在 copyfuncs.c、equalfuncs.c、outfuncs.c 中添加新结构体的 copy、out、equal 函数。首先向 switch 语句添加 case 函数入口，之后实现这个函数的具体内容。如图 21 所示是 SelectStmt 的 copy 函数入口及实现，out 和 equal 函数的添加方式与此相同，不再列举。在添加 copy、out、equal 函数需注意与系统的实现方式自洽，针对不同类型的参数将会调用不同的函数，如在图 21 中，COPY_NODE_FIELD 对应的是 node 节点类型，COPY_SCALAR_FIELD 则对应 int、float、bool、enum 等类型。

```
6702          case T_SelectStmt:
6703              retval = _copySelectStmt((SelectStmt*)from);
6704              break;

4301  static SelectStmt* _copySelectStmt(const SelectStmt* from)
4302  {
4303      SelectStmt* newnode = makeNode(SelectStmt);
4304
4305      COPY_NODE_FIELD(distinctClause);
4306      COPY_NODE_FIELD(intoClause);
4307      COPY_NODE_FIELD(targetList);
4308      COPY_NODE_FIELD(fromClause);
4309      COPY_NODE_FIELD(whereClause);
4310      COPY_NODE_FIELD(groupClause);
4311      COPY_NODE_FIELD(havingClause);
4312      COPY_NODE_FIELD(windowClause);
4313      COPY_NODE_FIELD(withClause);
4314      COPY_NODE_FIELD(valuesLists);
4315      COPY_NODE_FIELD(sortClause);
4316      COPY_NODE_FIELD(limitOffset);
4317      COPY_NODE_FIELD(limitCount);
4318      COPY_NODE_FIELD(lockingClause);
4319      COPY_NODE_FIELD(hintState);
4320      COPY_SCALAR_FIELD(op);
4321      COPY_SCALAR_FIELD(all);
4322      COPY_NODE_FIELD(larg);
4323      COPY_NODE_FIELD(rarg);
4324      COPY_SCALAR_FIELD(hasPlus);
4325
4326      return newnode;
4327  }
```

图 21 selectstmt 的 copy 函数入口及定义

4 语义分析

- **目标文件：**在 parser 文件夹中新建文件以实现对应语义解析。
- **作用：**完成新增语句的语义解析并构建算子级别的查询树。

新建源程序文件及对应的头文件，并在同层的 makefile 中添加编译语句。

```
OBJS= analyze.o gram.o uname_const_decl.o dkpool.o sparql_func.o sparql_p.o sparql_l.o  scan.o parser.o \
      parse_agg.o parse_clause.o parse_coerce.o parse_collate.o parse_cte.o \
      parse_enr.o parse_expr.o parse_func.o parse_node.o parse_oper.o \
      parse_param.o parse_relation.o parse_target.o parse_type.o \
      parse_utilcmd.o scansup.o \
      parse_cypher_expr.o parse_graph.o parse_shortestpath.o \
      parse_sparql_expr.o
```

图 22 makefile 添加编译语句

注：若新增功能点是基于原有查询增加的，则在原有 transform 函数中修改以支持该功能点，否则需要增加新文件，并在 analyse.c 中添加对应的入口。