
# DTCC

# 数，造，未，来

## 第十二届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2021

2021 年 10 月 18 日 - 20 日 | 北京国际会议中心

Spark
Hadoop
OLAP
HANA
MPP
Oracle
SQL

IT168.com ChinaUnix.net ITPUB

# 外连接符(+)在PostgreSQL中的一种实现方法

Jet C.X. ZHANG 章晨曦@易景科技

# 自我简介

- 现易景科技联合创始人&首席架构师
- 原某金融保险公司技术负责人
- 前Oracle ACE-A
- SOUG（南方Oracle用户组）联合发起人
- oracle_fdw committer
- postgresql reviewer

# 本次主题简介

在PostgreSQL中实现：

```
SELECT a.name, b.class
  FROM ta a, tb b
 WHERE a.id = b.id(+);
```

语句的正确执行。

# 基本思路

```
SELECT a.name, b.class
  FROM ta a, tb b
WHERE a.id = b.id(+);
```

=

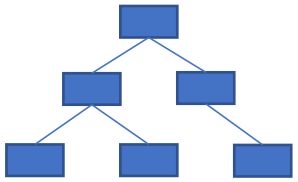
```
SELECT a.name, b.class
  FROM ta a LEFT JOIN tb b
    ON a.id = b.id;
```

# 基本思路

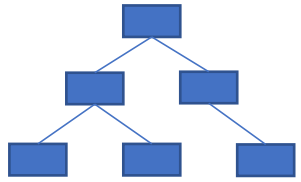
```
SELECT a.name, b.class
  FROM ta a, tb b
WHERE a.id = b.id(+);
```

=

```
SELECT a.name, b.class
  FROM ta a LEFT JOIN tb b
    ON a.id = b.id;
```

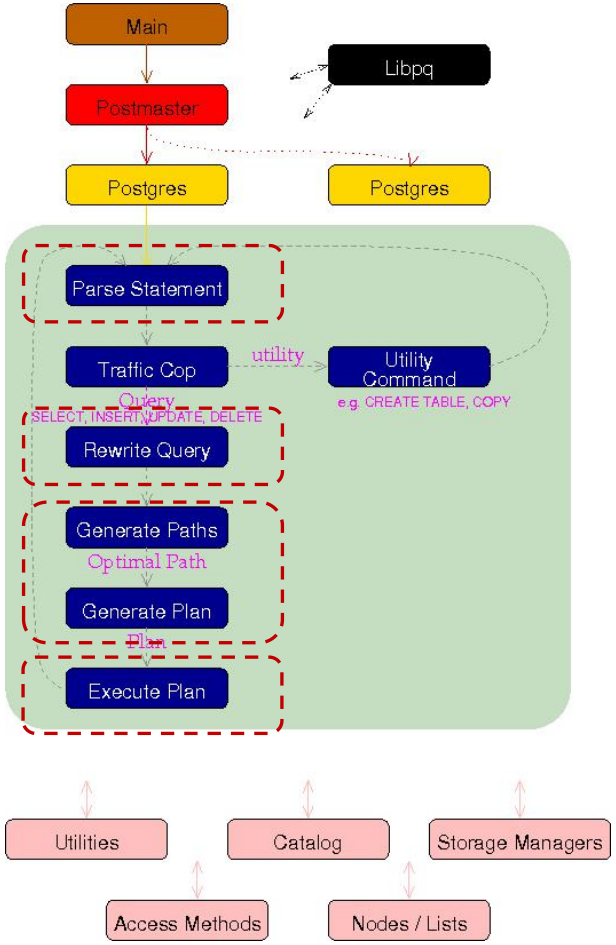


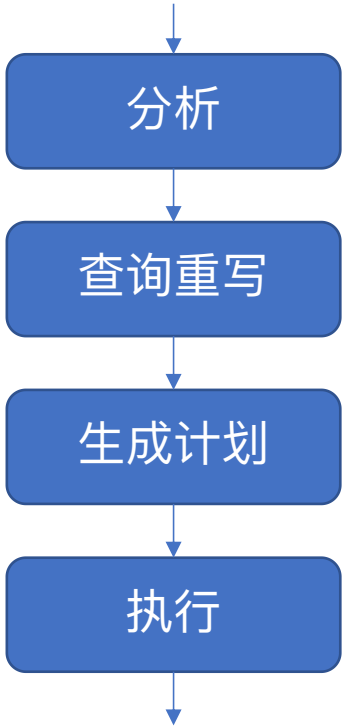
=



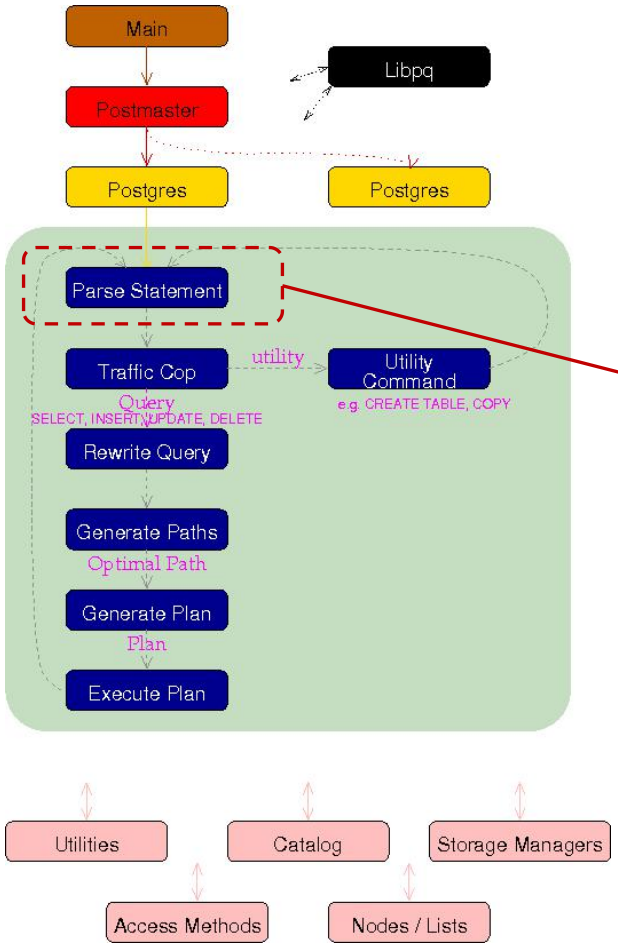
**注意：**
**非字符串层面“翻译”，而是解析为等价的查询树**

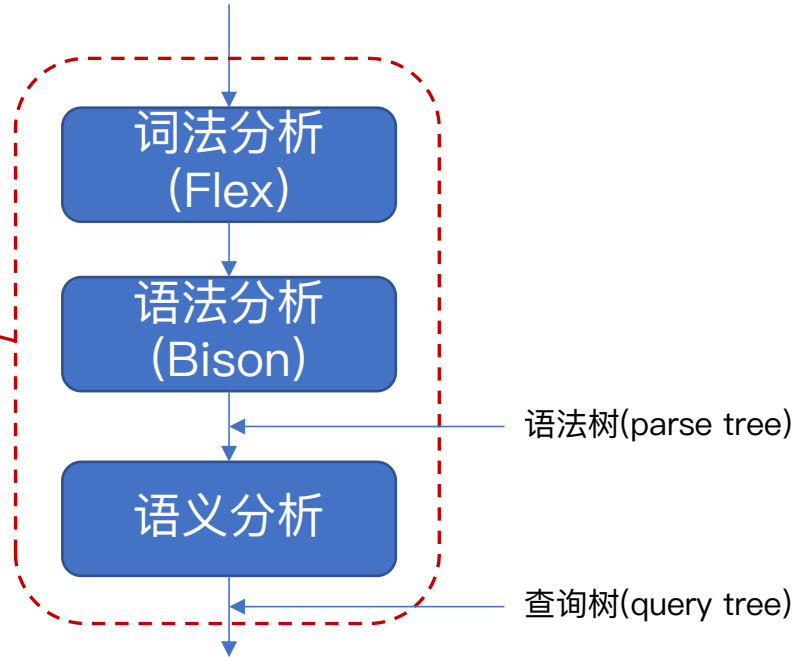
# SQL的查询编译过程


Main
Libpq
Postmaster
Postgres
Postgres
Parse Statement
Traffic Cop
utility
Utility Command
e.g. CREATE TABLE, COPY
Query
SELECT, INSERT, UPDATE, DELETE
Rewrite Query
Generate Paths
Optimal Path
Generate Plan
Plan
Execute Plan
Utilities
Catalog
Storage Managers
Access Methods
Nodes / Lists

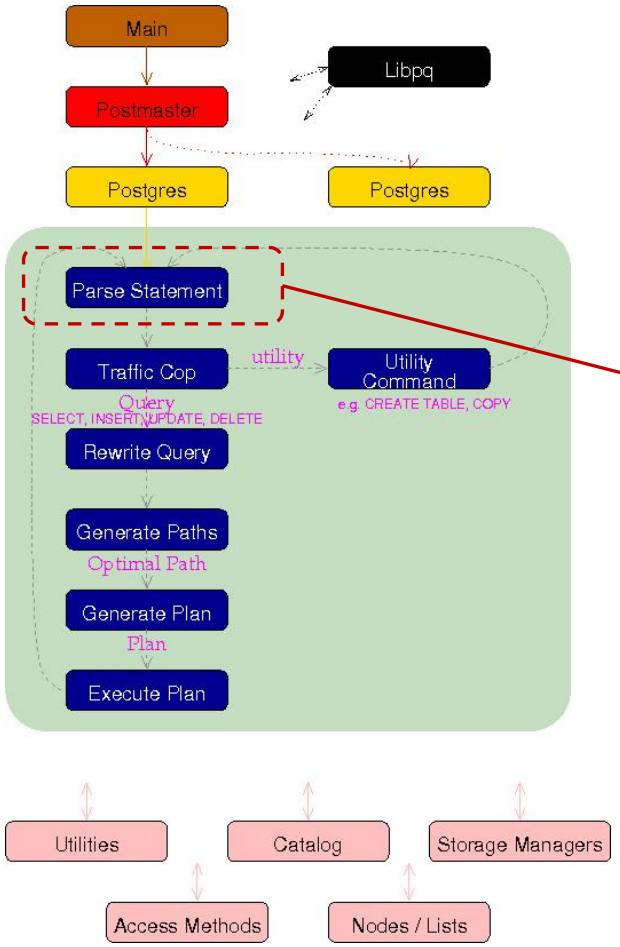

分析
查询重写
生成计划
执行

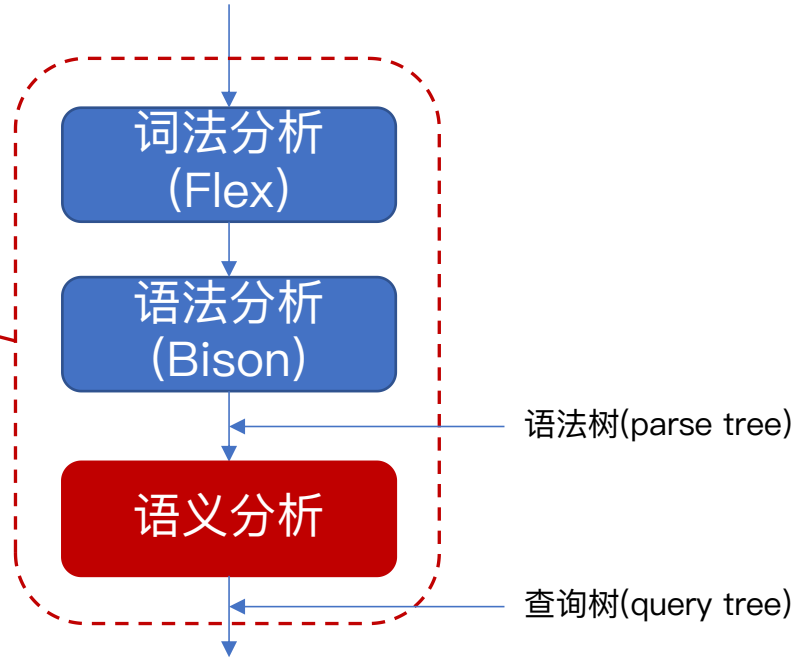
# SQL的查询编译过程


Main
Libpq
Postmaster
Postgres
Postgres
Parse Statement
Traffic Cop
utility
Utility Command
Query
SELECT, INSERT, UPDATE, DELETE
e.g. CREATE TABLE, COPY
Rewrite Query
Generate Paths
Optimal Path
Generate Plan
Plan
Execute Plan
Utilities
Catalog
Storage Managers
Access Methods
Nodes / Lists

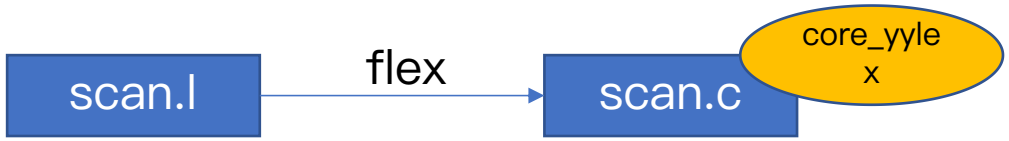

词法分析
(Flex)
语法分析
(Bison)
语义分析
语法树(parse tree)
查询树(query tree)

# SQL的查询编译过程

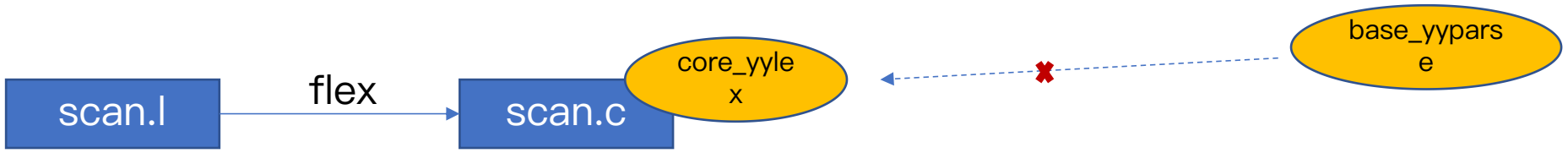

Main
Libpq
Postmaster
Postgres
Postgres
Parse Statement
Traffic Cop
utility
Utility Command
Query
SELECT, INSERT, UPDATE, DELETE
e.g. CREATE TABLE, COPY
Rewrite Query
Generate Paths
Optimal Path
Generate Plan
Plan
Execute Plan
Utilities
Catalog
Storage Managers
Access Methods
Nodes / Lists


词法分析
(Flex)
语法分析
(Bison)
语义分析
语法树(parse tree)
查询树(query tree)

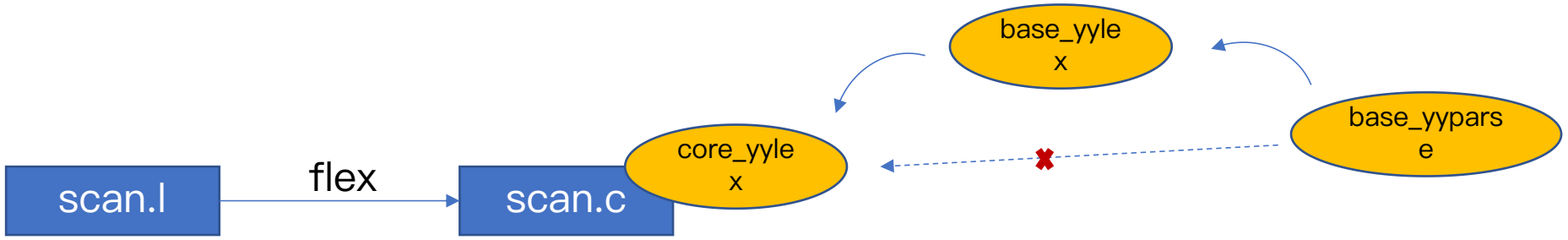
# (+)的识别



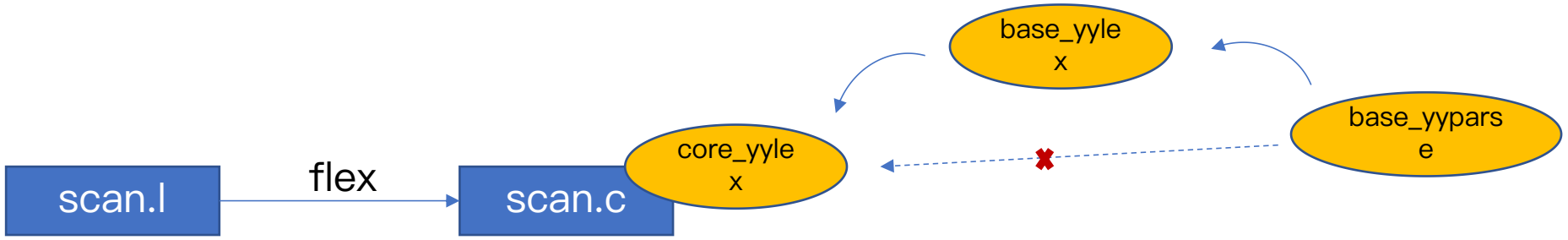
# (+)的识别



# (+)的识别

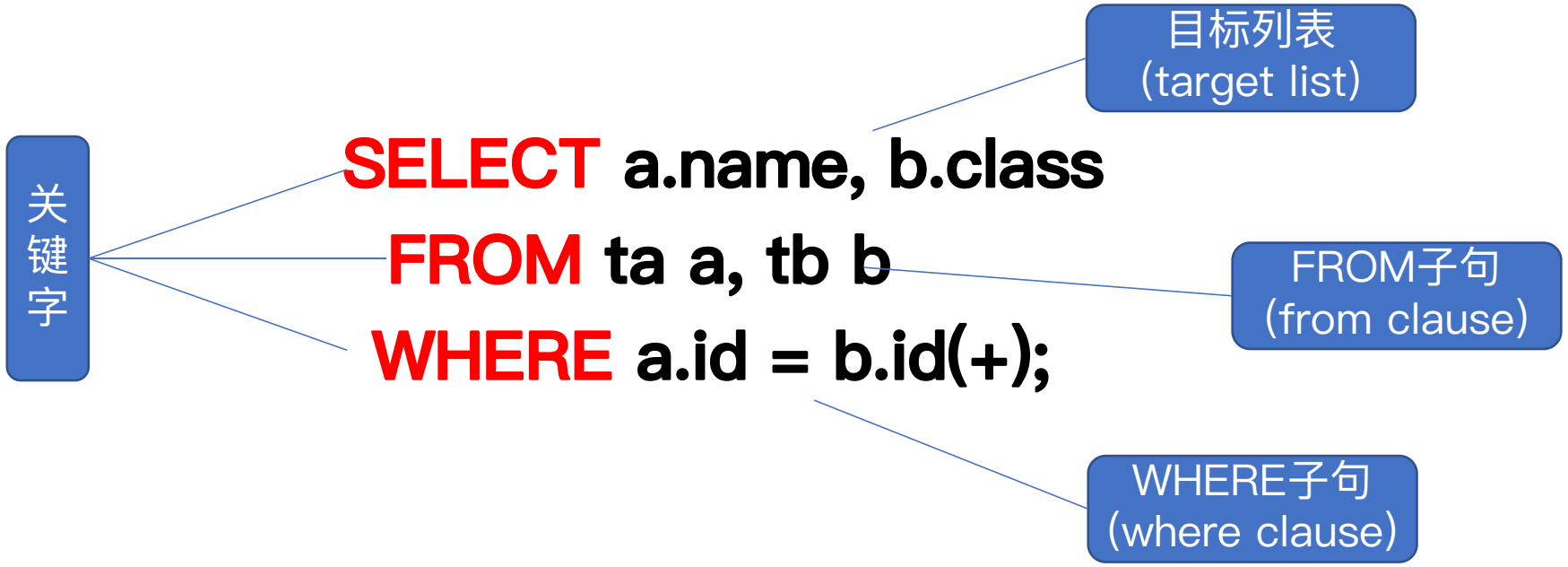


# (+)的识别



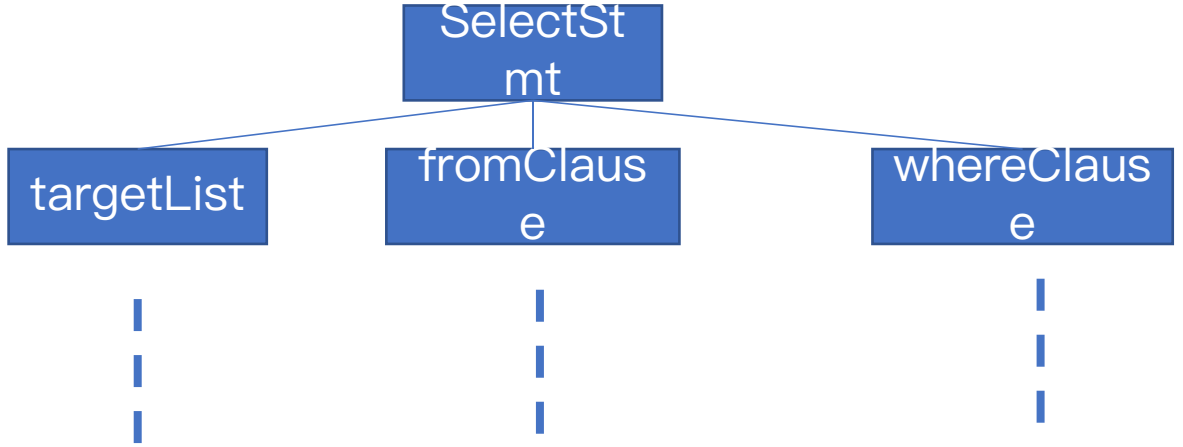

```
case '(':
    if (next_token == '+')
    {
        cur_yylloc = *llocp;
        *(yyextra->lookahead_end) = yyextra->lookahead_hold_char;
        next_token = core_yylex(&(yyextra->lookahead_yylval),
                                llocp, yyscanner);
        if (next_token == ')')
        {
            *llocp = cur_yylloc;
            cur_token = OPT_JOIN;
            yyextra->have_lookahead = false;
        }
    }
    break;
```

# SELECT语句的成分分析



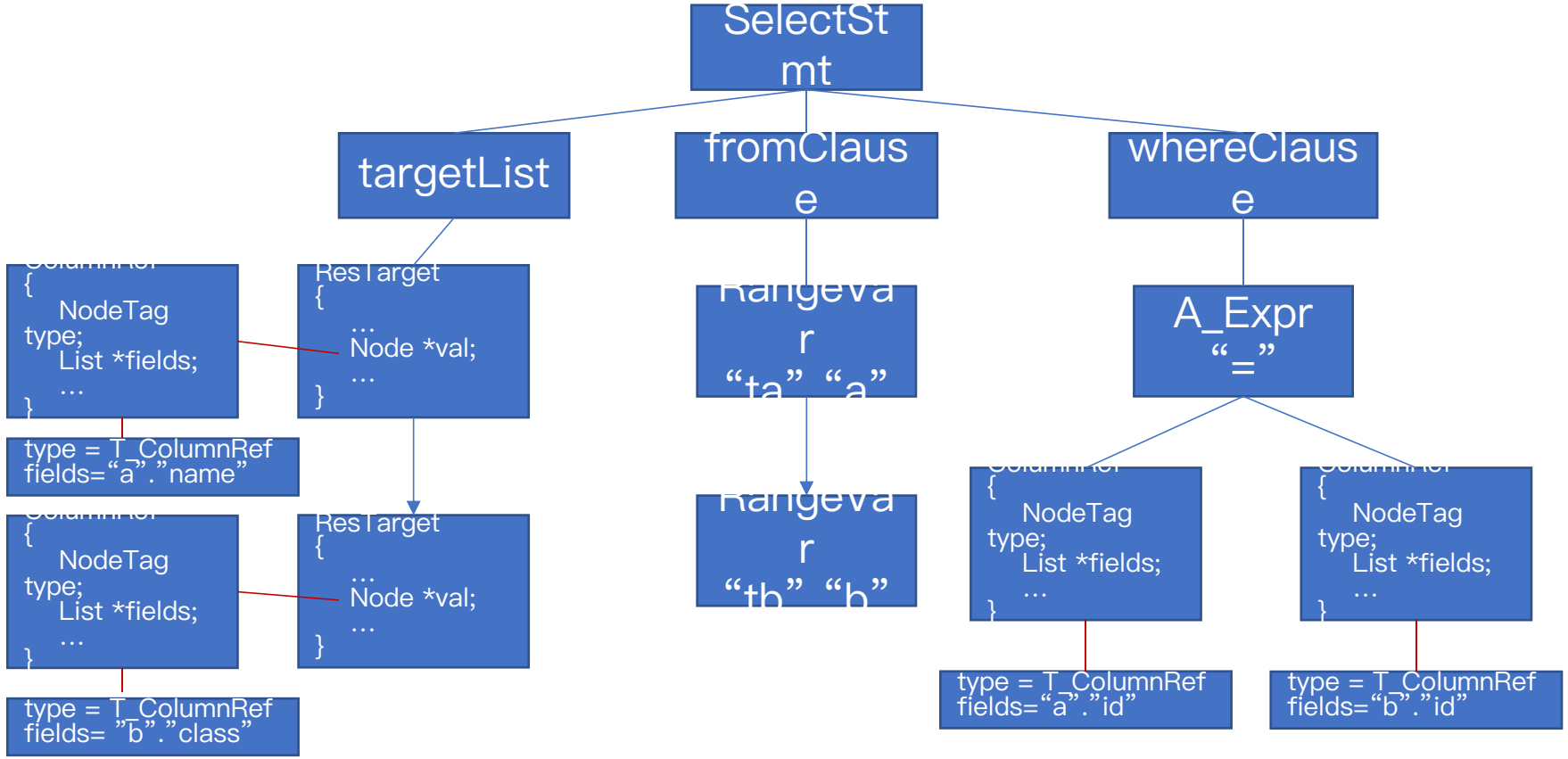
# SELECT语句的语法树

```
struct SelectStmt
{
    ...
    List        *targetList;
    List        *fromClause;
    Node    *whereClause;
    ...
}
```



# 无(+)的语法树

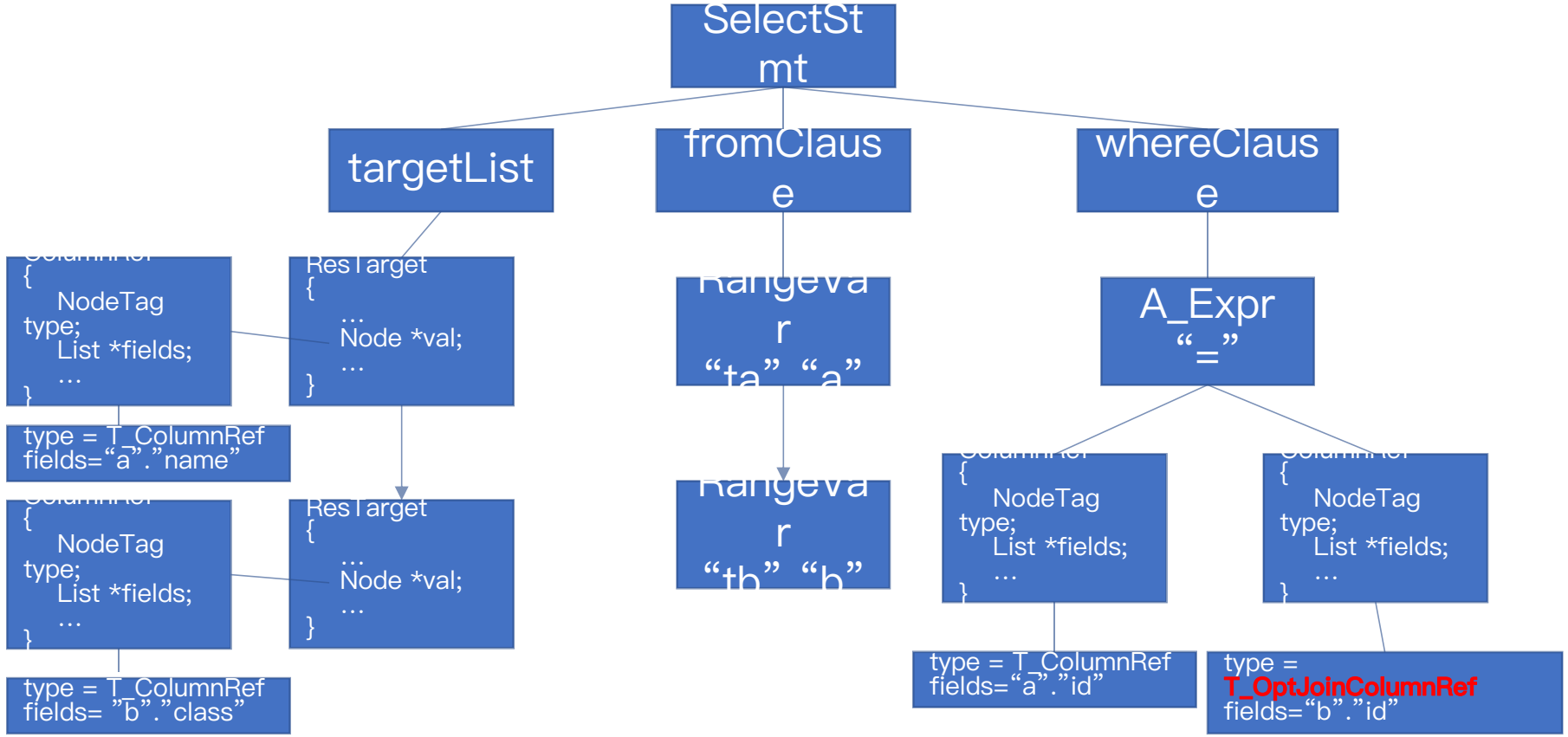
```
SELECT a.name,
b.class
 FROM ta a, tb b
WHERE a.id = b.id;
struct SelectStmt
{
    …
    List        *targetList;
    List        *fromClause;
    Node    *whereClause;
    …
}
```



# (+)的处理技术及语法树

```
SELECT a.name,
b.class
 FROM ta a, tb b
WHERE a.id = b.id(+);
```

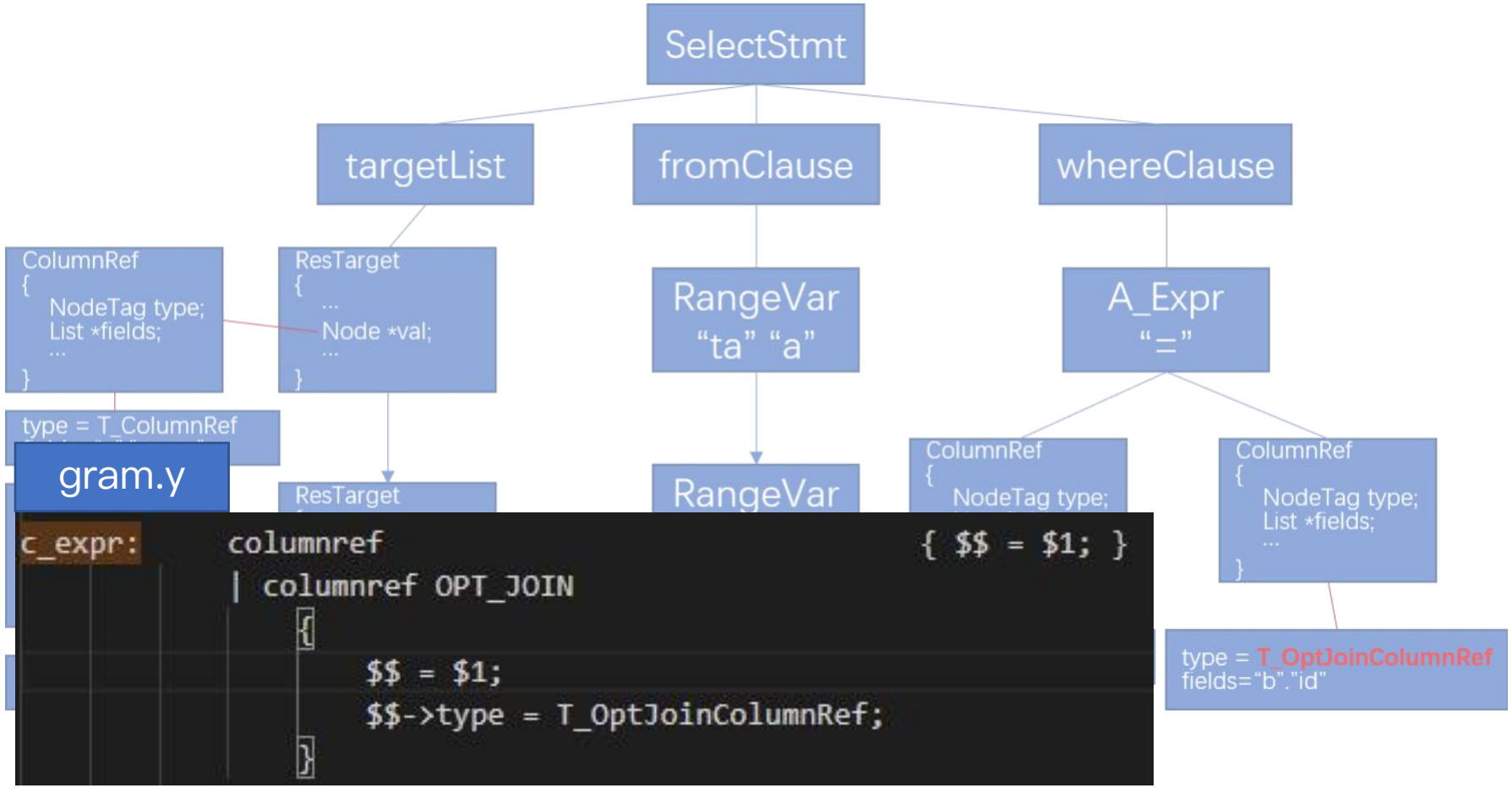
```
struct SelectStmt
{
    …
    List        *targetList;
    List        *fromClause;
    Node        *whereClause;
    …
}
enum NodeTag
{
    …
    T_OptJoinColumnRef
}
```



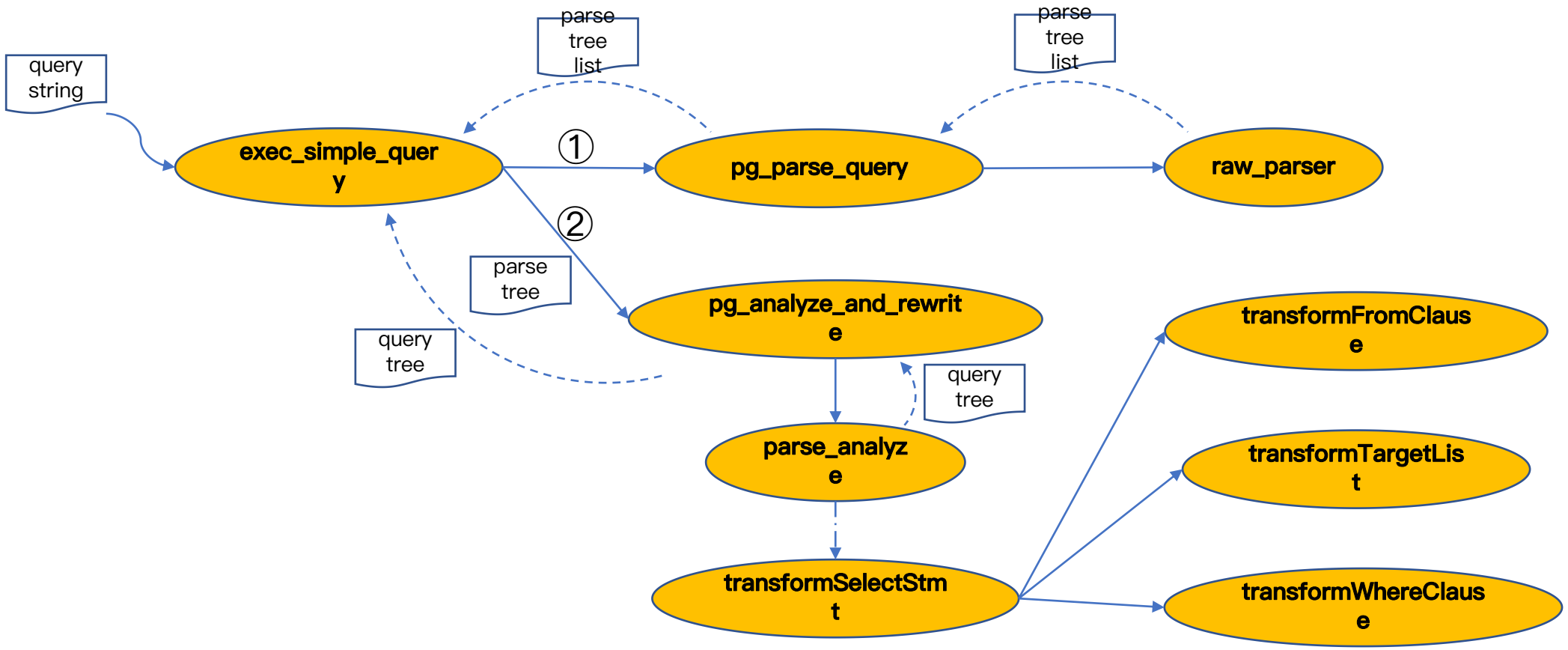
# (+)的处理技术及语法树

```
SELECT a.name,
b.class
 FROM ta a, tb b
WHERE a.id = b.id(+);
```

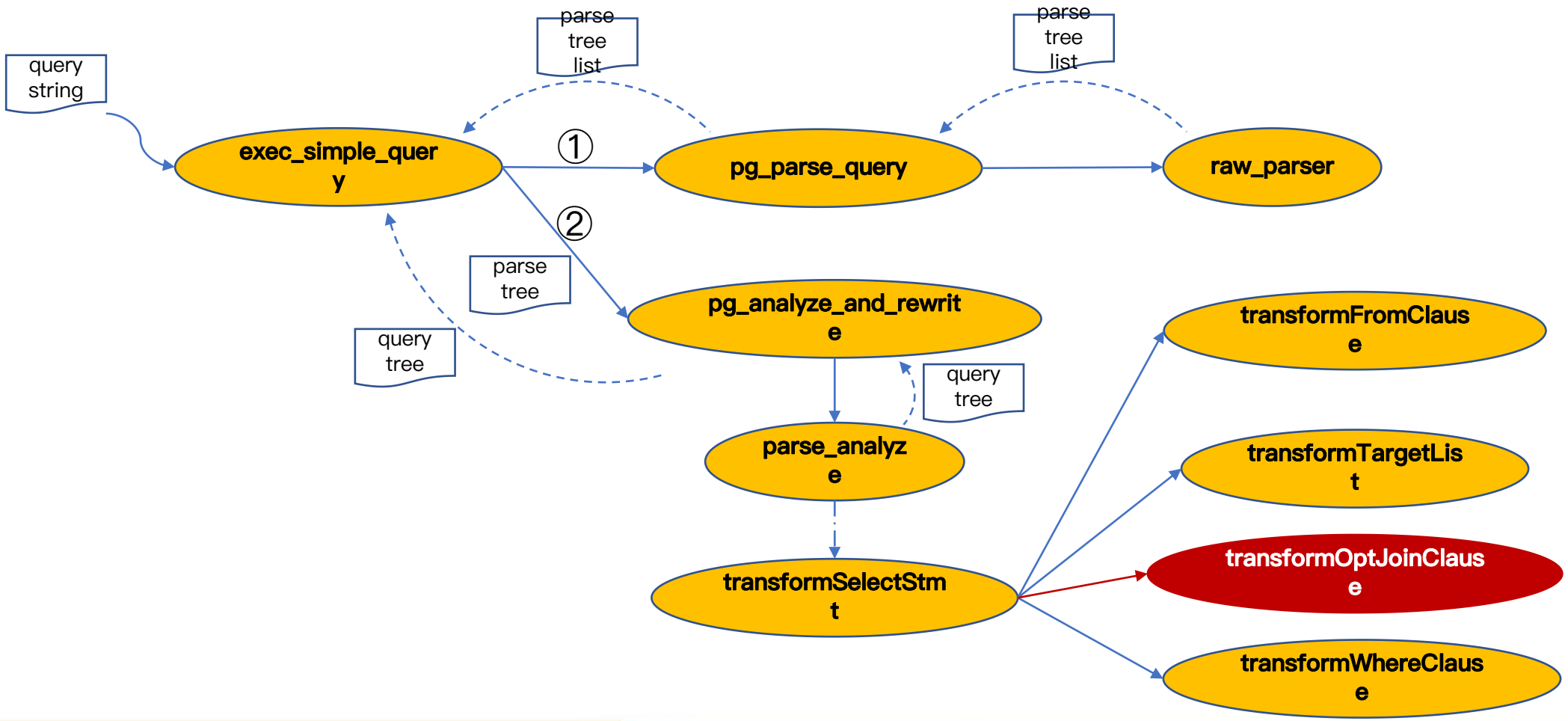
```
struct SelectStmt
{
    ...
    List        *targetList;
    List        *fromClause;
    Node        *whereClause;
    ...
}
enum NodeTag
{
    ...
    T_OptJoinColumnRef
}
```



# 语义分析过程

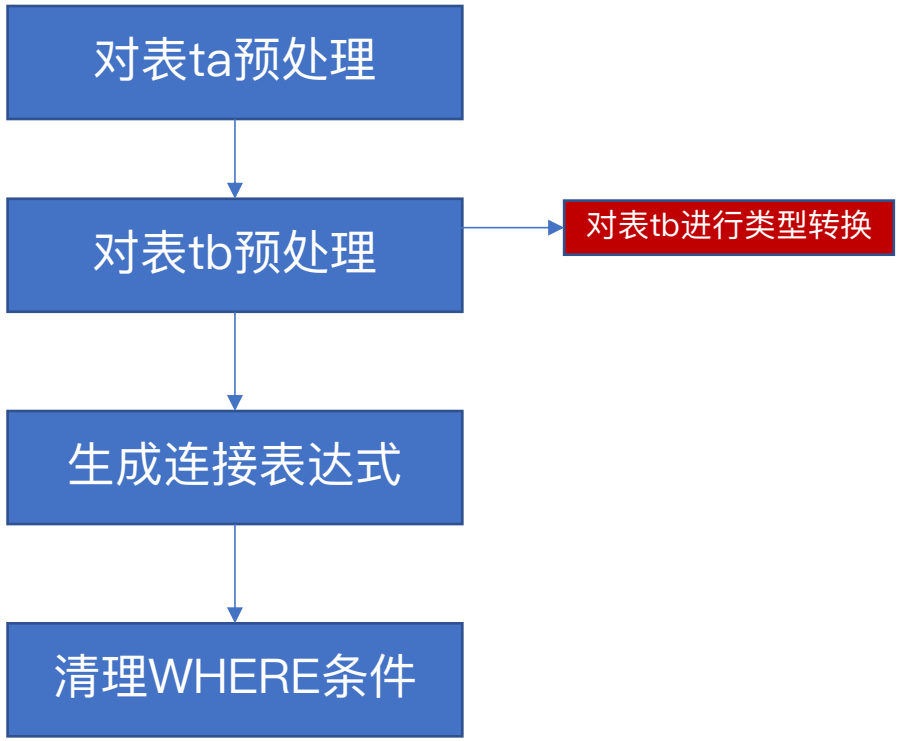

query string
parse tree list
parse tree list
exec_simple_query
①
pg_parse_query
raw_parser
②
parse tree
query tree
pg_analyze_and_rewrite
query tree
transformFromClause
parse_analyze
transformTargetList
transformSelectStmt
transformWhereClause

# 带(+)的语义分析


query string
parse tree list
parse tree list
exec_simple_query
①
pg_parse_query
raw_parser
②
parse tree
query tree
pg_analyze_and_rewrite
query tree
parse_analyze
transformSelectStmt
transformFromClause
transformTargetList
transformOptJoinClause
transformWhereClause

# 带(+)的语义分析

transformOptJoinClause



# 带(+)的语义分析

主要功能代码
讲解及演示

# Demo效果

```
postgres=# SELECT a.name, b.class FROM ta a, tb b WHERE a.id = b.id(+);
 name | class
------+-------
 A    | S1
 B    | S1
 C    | S2
 D    |
(4 rows)

postgres=# EXPLAIN SELECT a.name, b.class FROM ta a, tb b WHERE a.id = b.id(+);
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Left Join  (cost=23.27..101.56 rows=1740 width=156)
   Hash Cond: (a.id = b.id)
   ->  Seq Scan on ta a  (cost=0.00..15.90 rows=590 width=110)
   ->  Hash  (cost=15.90..15.90 rows=590 width=110)
         ->  Seq Scan on tb b  (cost=0.00..15.90 rows=590 width=110)
(5 rows)
```

# 小结

- 不引入新的数据结构
- （+）支持空格、换行（兼容性更强）
- 不损失性能


THANKS
时序
开源
自研
分布式
文档
Spark
HANA
SQL
OLAP
Oracle
Hadoop
MPP
DB2