# 【Doris全面解析】Doris SQL 原理解析

mp.weixin.qq.com/s/v1jl1MxEHPT5czCWd0kRxw

**导读**

本文主要介绍了Doris SQL解析的原理。
重点讲述了生成单机逻辑计划，生成分布式逻辑计划，生成分布式物理计划的过程。对应于代码实现是Analyze，SinglePlan，DistributedPlan，Schedule四个部分。
Analyze负责对AST进行前期的一些处理，SinglePlan根据AST进行优化生成单机查询计划，DistributedPlan将单机的查询计划拆成分布式的查询计划，Schedule阶段负责决定查询计划下发到哪些机器上执行。
由于SQL类型有很多，**本文侧重介绍查询SQL的解析**，从算法原理和代码实现上深入讲解了Doris的SQL解析原理。

## 1 Doris简介

Doris是基于MPP架构的交互式SQL数据仓库，主要用于解决近实时的报表和多维分析。
Doris分成两部分FE和BE，FE 负责存储以及维护集群元数据、接收、解析、查询、设计规划整体查询流程，BE 负责数据存储和具体的实施过程。
在 Doris 的存储引擎中，用户数据被水平划分为若干个数据分片（Tablet，也称作数据分桶）。每个 Tablet 包含若干数据行。多个 Tablet 在逻辑上归属于不同的分区Partition。一个 Tablet 只属于一个 Partition。而一个 Partition 包含若干个 Tablet。Tablet 是数据移动、复制等操作的最小物理存储单元。

## 2 SQL解析简介

SQL解析在这篇文章中指的是**将一条sql语句经过一系列的解析最后生成一个完整的物理执行计划的过程。**

这个过程包括以下四个步骤：词法分析，语法分析，生成逻辑计划，生成物理计划。

如图1所示：

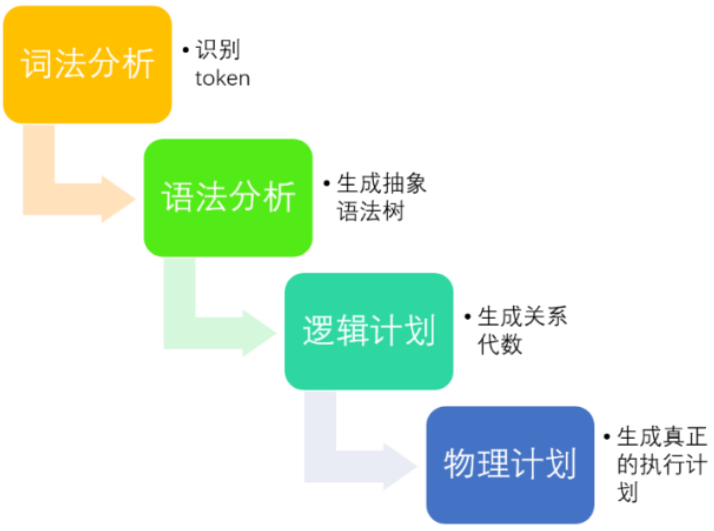


**图 1 SQL解析的流程**

### 2.1 词法分析

词法分析主要负责将字符串形式的sql识别成一个个token，为语法分析做准备。

```
select ......  from ...... where ....... group by ..... order by ......
SQL 的 Token 可以分为如下几类：◦ 关键字（select、from、where）◦ 操作符（+、-、>=）◦ 开闭合标志（(、CASE）◦ 占位符（?）◦ 注释◦ 空格......
```

### 2.2 语法分析

语法分析主要负责根据语法规则，将词法分析生成的token转成抽象语法树（Abstract Syntax Tree），如图2所示。

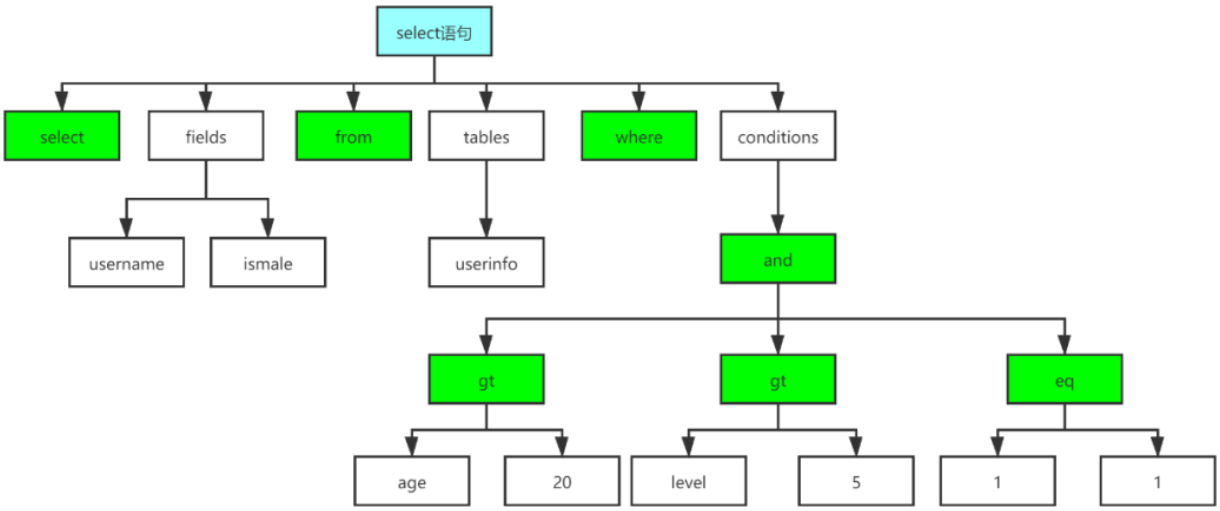



**图 2 抽象语法树示例**

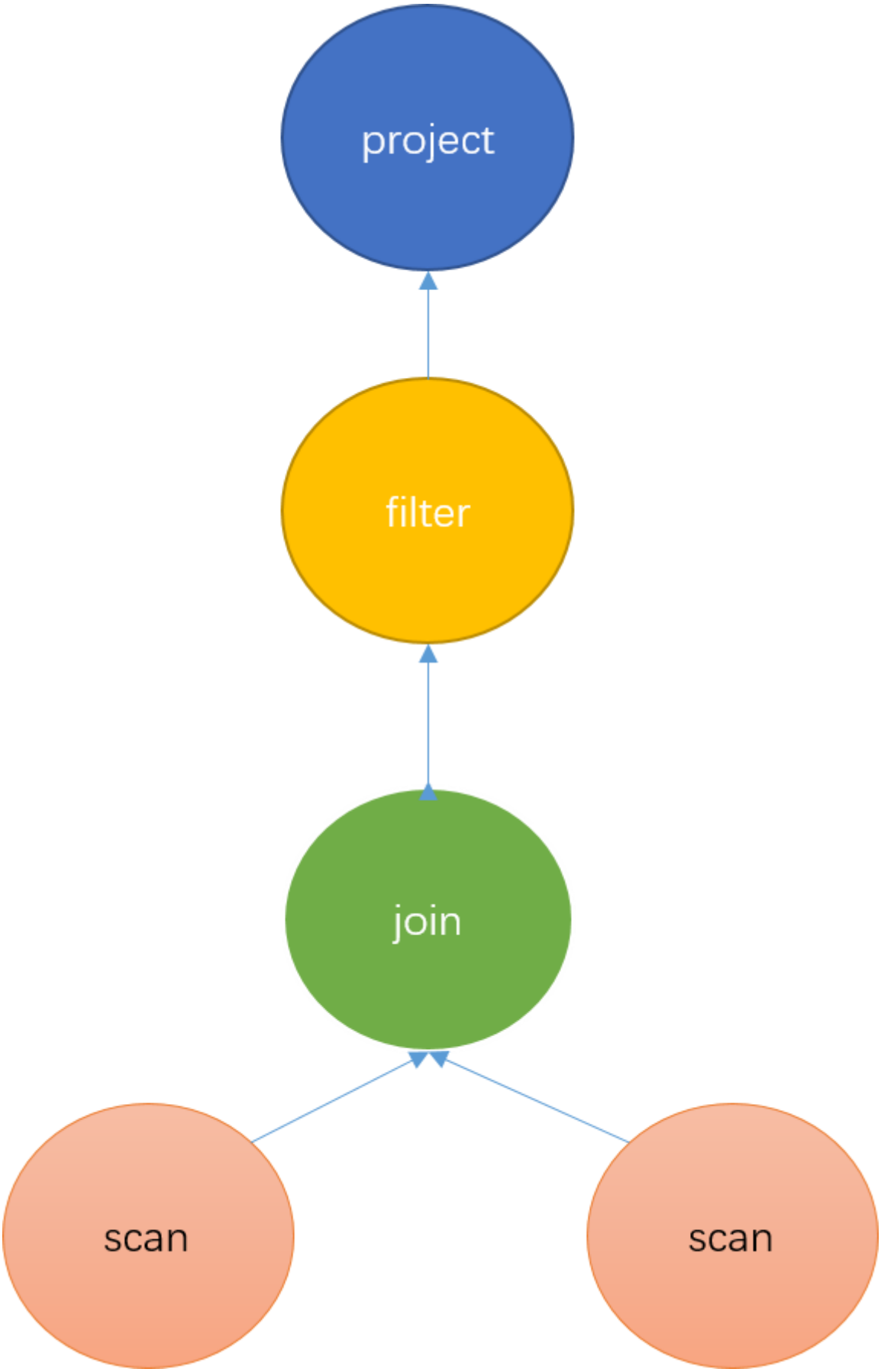
### 2.3 逻辑计划

逻辑计划负责将抽象语法树转成代数关系。代数关系是一棵算子树，每个节点代表一种对数据的计算方式，整棵树代表了数据的计算方式以及流动方向，如图3所示。

**图3 关系代数示例**

### 2.4 物理计划

物理计划是在逻辑计划的基础上，根据机器的分布，数据的分布，决定去哪些机器上执行哪些计算操作。




Doris系统的SQL解析也是采用这些步骤，只不过根据Doris系统结构的特点和数据的存储方式，进行了细化和优化，最大化发挥机器的计算能力。

## 3 设计目标

Doris SQL解析架构的设计有以下目标：

1. **最大化计算的并行性**
2. **最小化数据的网络传输**
3. **最大化减少需要扫描的数据**

## 4 总体架构

Doris SQL解析具体包括了五个步骤：词法分析，语法分析，生成单机逻辑计划，生成分布式逻辑计划，生成物理执行计划。

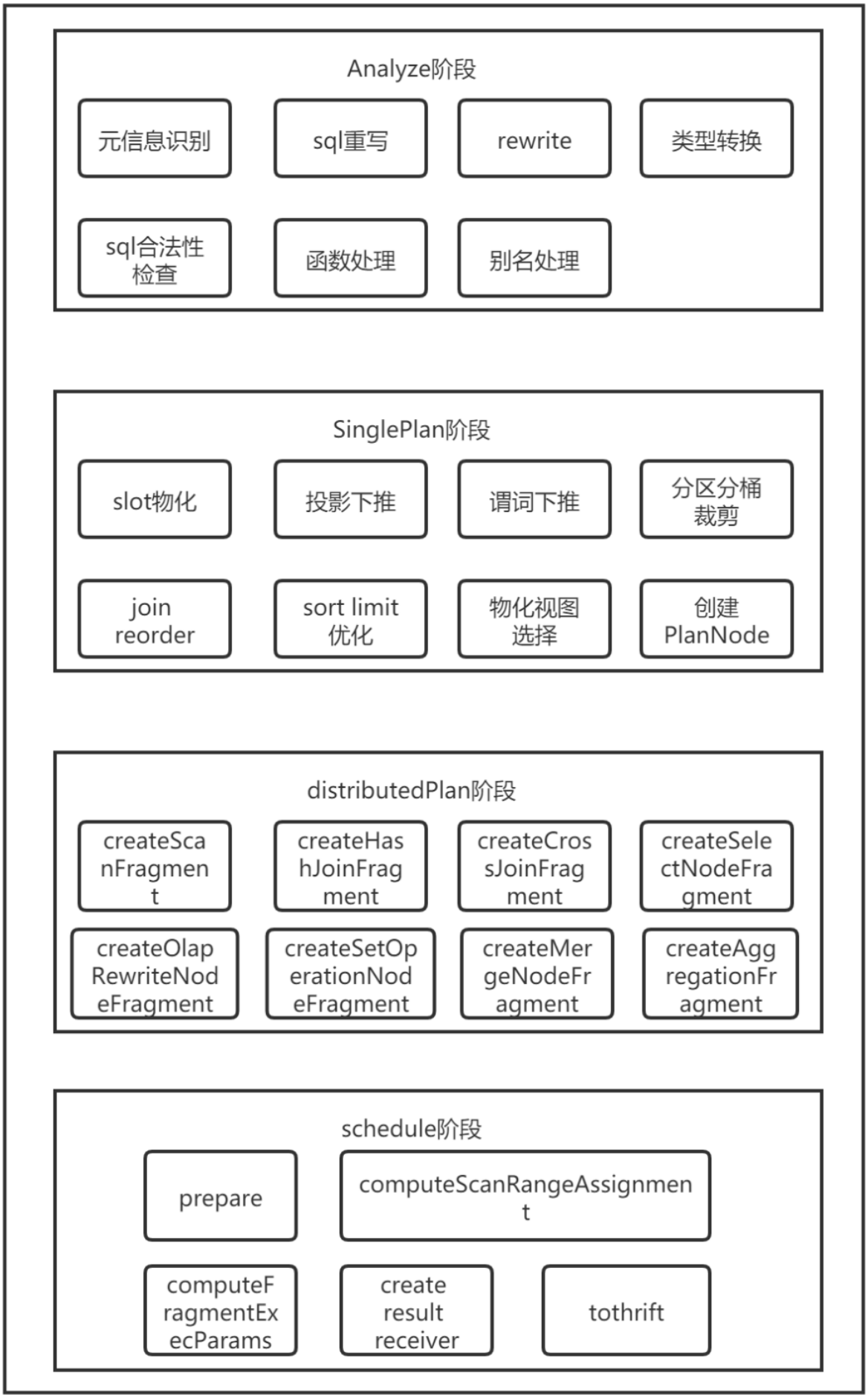
具体代码实现上包含以下五个步骤：Parse, Analyze, SinglePlan, DistributedPlan, Schedule。

**图4 系统总体架构图**

如图4所示，Parse阶段本文不详细讲，Analyze负责对AST进行前期的一些处理，SinglePlan根据AST进行优化生成单机查询计划，DistributedPlan将单机的查询计划拆成分布式的查询计划，Schedule阶段负责决定查询计划下发到哪些机器上执行。

**由于SQL类型有很多，本文侧重介绍查询SQL的解析。**

图5展示了一个简单的查询SQL在Doris的解析实现。


Analyze阶段
元信息识别
sql重写
rewrite
类型转换
sql合法性检查
函数处理
别名处理
SinglePlan阶段
slot物化
投影下推
谓词下推
分区分桶裁剪
join reorder
sort limit优化
物化视图选择
创建PlanNode
distributedPlan阶段
createScanFragment
createHashJoinFragment
createCrossJoinFragment
createSelectNodeFragment
createOlapRewriteNodeFragment
createSetOperationNodeFragment
createMergeNodeFragment
createAggregationFragment
schedule阶段
prepare
computeScanRangeAssignment
computeFragmentExecParams
create result receiver
tothrift

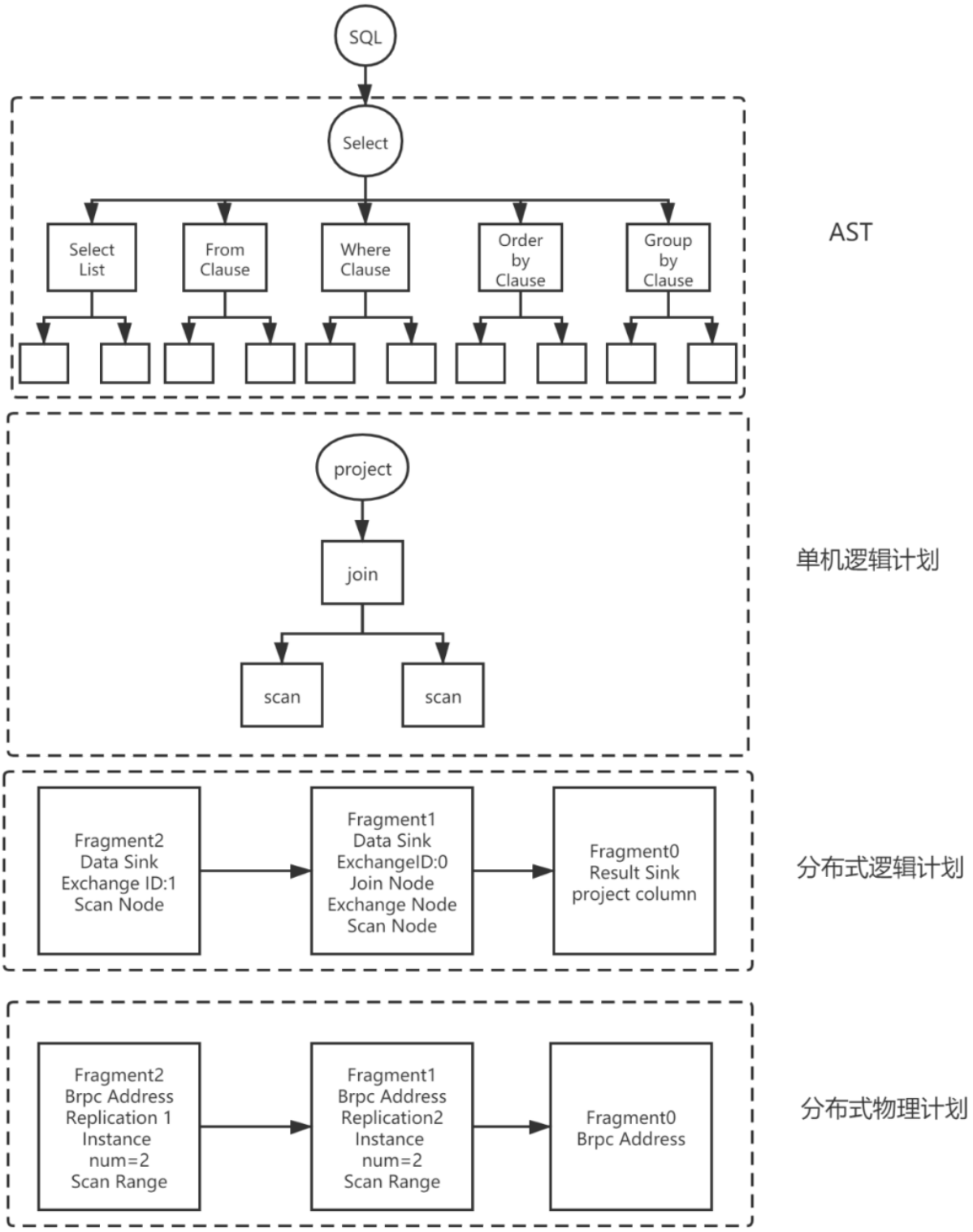



**图 5 查询sql在Doris中的解析过程**

## 5 Parse阶段

词法分析采用jflex技术，语法分析采用java cup parser技术，最后生成抽象语法树（Abstract Syntax Tree）AST，这些都是现有的、成熟的技术，在这里不进行详细介绍。

AST是一种树状结构，代表着一条SQL。不同类型的查询select, insert, show, set, alter table, create table等经过Parse阶段后生成不同的数据结构（SelectStmt, InsertStmt, ShowStmt, SetStmt, AlterStmt, AlterTableStmt, CreateTableStmt等），但他们都继承自Statement，并根据自己的语法规则进行一些特定的处理。例如：对于select类型的sql，Parse之后生成了SelectStmt结构。

SelectStmt结构包含了SelectList，FromClause，WhereClause，GroupByClause，SortInfo等结构。这些结构又包含了更基础的一些数据结构，如WhereClause包含了BetweenPredicate（between表达式），BinaryPredicate（二元表达式），CompoundPredicate（and or组合表达式），InPredicate（in表达式）等。

AST中所有结构都是由基本结构表达式Expr通过多种组合而成，如图6所示。

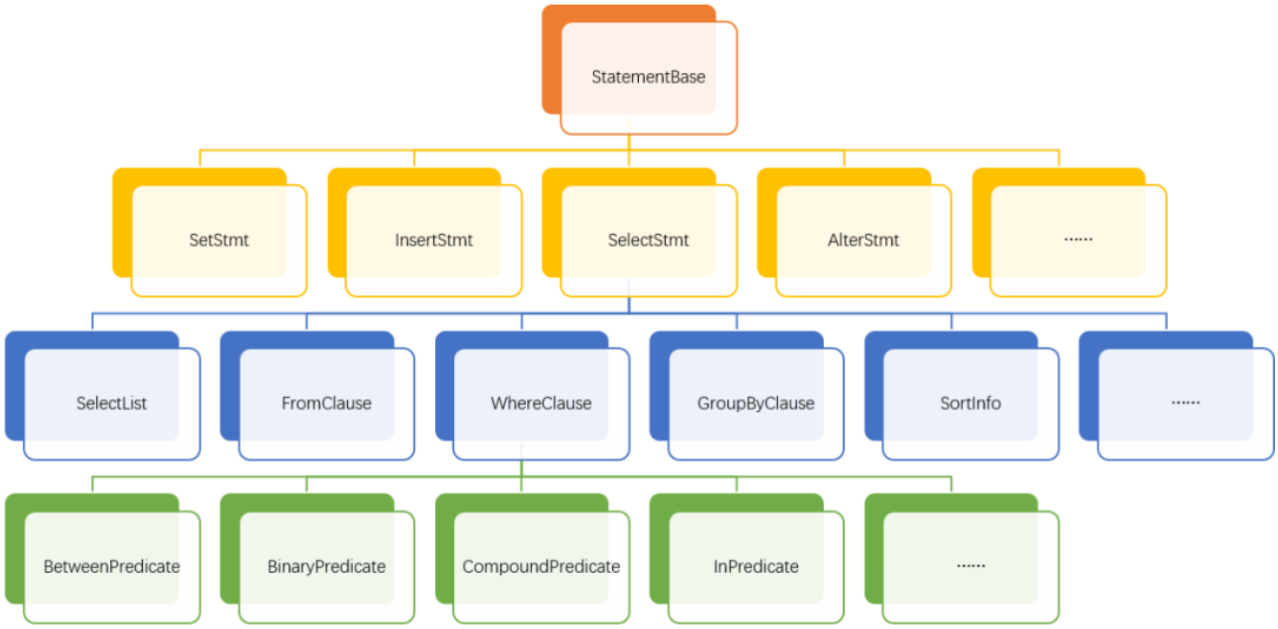



**图 6 Doris中抽象语法树AST的实现**

## 6 Analyze阶段

Analyze主要是对Parse阶段生成的抽象语法树AST进行一些前期的处理和语义分析，为生成单机逻辑计划做准备。

抽象语法树是由StatementBase这个抽象类表示。这个抽象类包含一个最重要的成员函数analyze()，用来执行Analyze阶段要做的事。

不同类型的查询select, insert, show, set, alter table, create table等经过Parse阶段后生成不同的数据结构（SelectStmt, InsertStmt, ShowStmt, SetStmt, AlterStmt, AlterTableStmt, CreateTableStmt等），这些数据结构继承自StatementBase，并实现analyze()函数，对特

定类型的SQL进行特定的Analyze。

例如：select类型的查询，会转成对select sql的子语句SelectList, FromClause, GroupByClause, HavingClause, WhereClause, SortInfo等的analyze()。然后这些子语句再各自对自己的子结构进行进一步的analyze()，通过层层迭代，把各种类型的sql的各种情景都分析完毕。例如：WhereClause进一步分析其包含的BetweenPredicate（between表达式）, BinaryPredicate（二元表达式），CompoundPredicate（and or组合表达式），InPredicate（in表达式）等。

**对于查询类型的SQL，包含以下几项重要工作：**

· **元信息的识别和解析：**识别和解析sql中涉及的 Cluster, Database, Table, Column 等元信息，确定需要对哪个集群的哪个数据库的哪些表的哪些列进行计算。

· **SQL 的合法性检查：**窗口函数不能 DISTINCT，投影列是否有歧义，where语句中不能含有grouping操作等。

· **SQL 简单重写：**比如将 select * 扩展成 select 所有列，count distinct转成bitmap或者hll函数等。

· **函数处理：**检查sql中包含的函数和系统定义的函数是否一致，包括参数类型，参数个数等。

· **Table 和 Column 的别名处理**

· **类型检查和转换：**例如二元表达式两边的类型不一致时，需要对其中一个类型进行转换（BIGINT 和 DECIMAL 比较，BIGINT 类型需要 Cast 成 DECIMAL）。

对AST 进行analyze后，会再进行一次rewrite操作，进行精简或者是转成统一的处理方式。目前rewrite的算法是基于规则的方式，针对AST的树状结构，自底向上，应用每一条规则进行重写。如果重写后，AST有变化，则再次进行analyze和rewrite，直到AST无变化为止。

例如：常量表达式的化简：1 + 1 + 1 重写成 3，1 > 2 重写成 Flase 等。将一些语句转成统一的处理方式，比如将 where in, where exists 重写成 semi join, where not in, where not exists 重写成 anti join。

## 7 生成单机逻辑Plan阶段

这部分工作主要是根据AST抽象语法树生成代数关系，也就是俗称的算子数。树上的每个节点都是一个算子，代表着一种操作。

如图7所示，ScanNode代表着对一个表的扫描操作，将一个表的数据读出来。HashJoinNode代表着join操作，小表在内存中构建哈希表，遍历大表找到连接键相同的值。Project表示投影操作，代表着最后需要输出的列，图7表示只用输出citycode这一列。

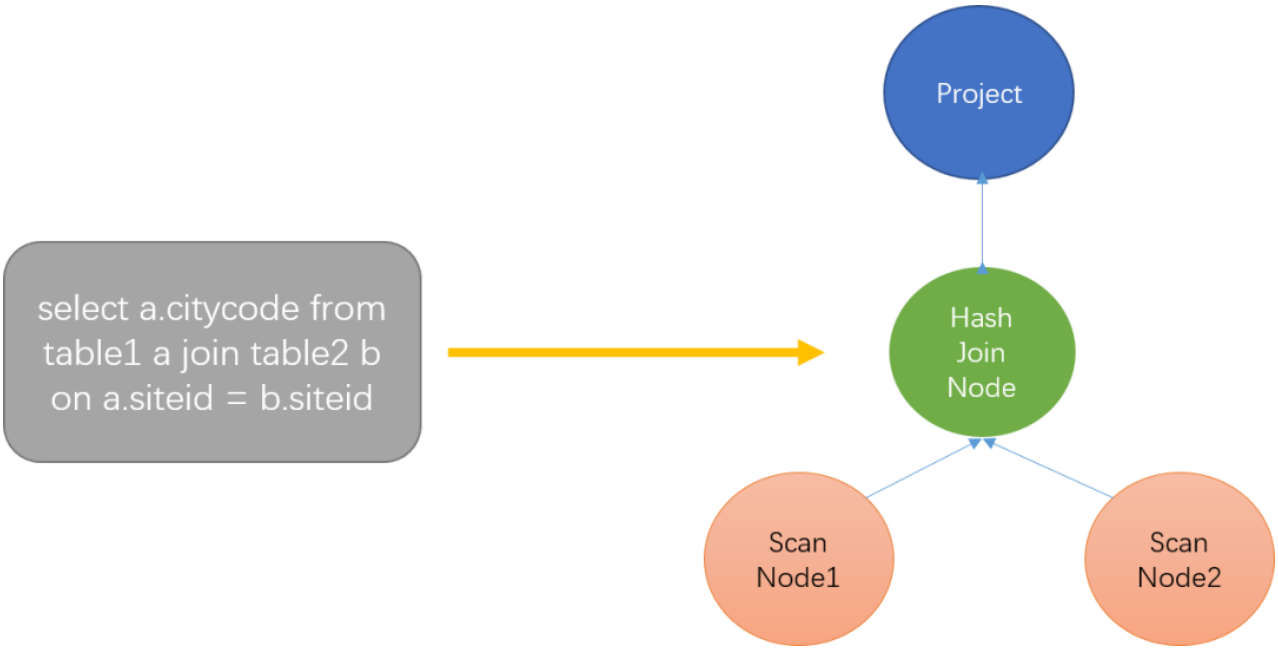



**图7 单机逻辑计划示例**

如果不进行优化，生成的关系代数下发到存储中执行的代价非常高。
对于查询：

```
select a.siteid, a.pv from table1 a join table2 b on a.siteid = b.siteid where a.ci
```

未优化的关系代数，如图8所示，需要将所有列读出来进行一系列的计算，在最后选择输出siteid, pv两列，大量无用的列数据浪费了计算资源。
Doris在生成代数关系时，进行了大量的优化，将投影列和查询条件尽可能放到扫描操作时执行。

**图8 未优化的关系代数**

**具体来说这个阶段主要做了如下几项工作：**

· **Slot 物化：**指确定一个表达式对应的列需要 Scan 和计算，比如聚合节点的聚合函数表达式和 Group By 表达式需要进行物化。

· **投影下推：**BE 在 Scan 时只会 Scan 必须读取的列。

· **谓词下推：**在满足语义正确的前提下将过滤条件尽可能下推到 Scan 节点。

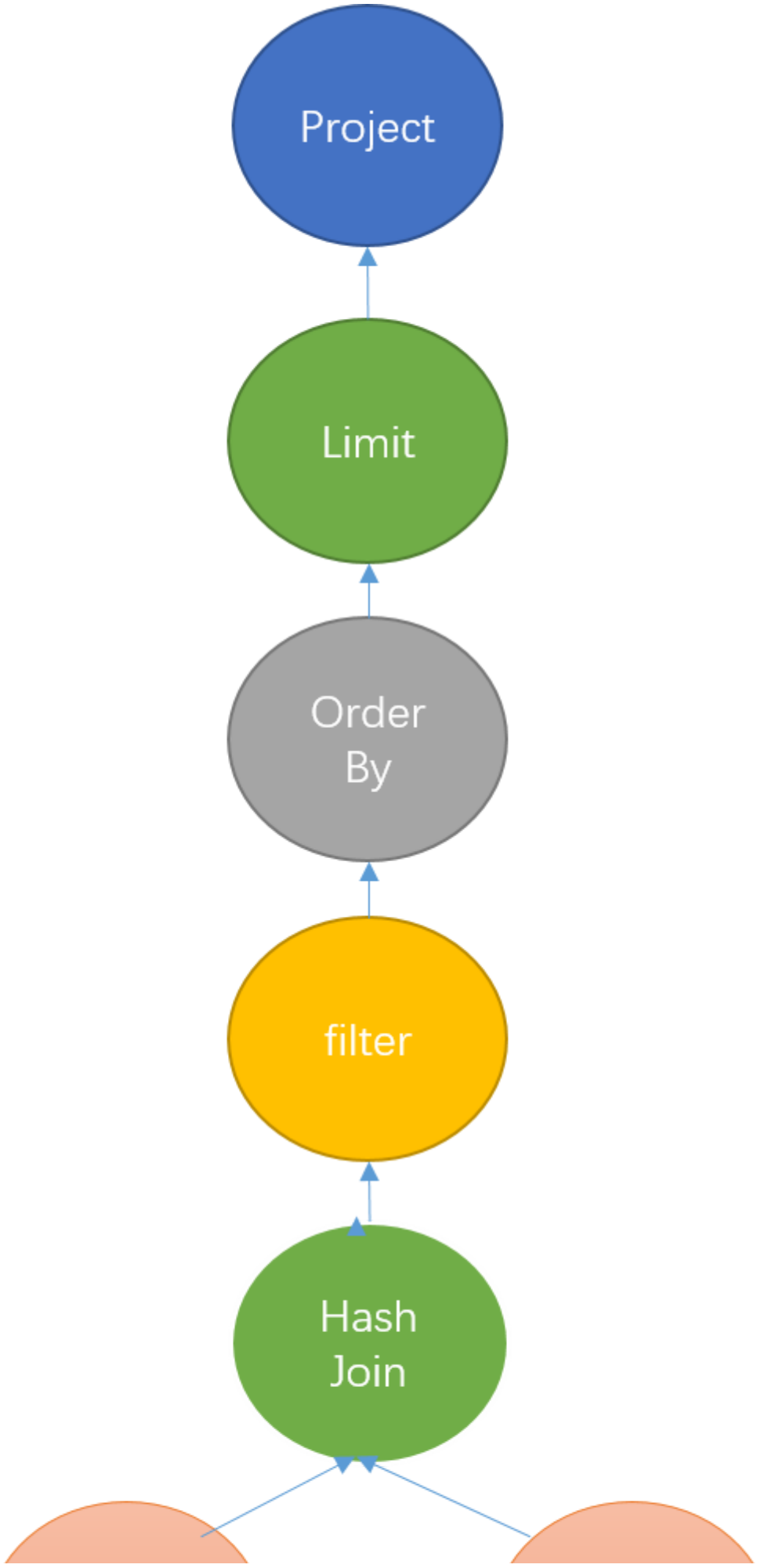
· **分区，分桶裁剪：**根据过滤条件中的信息，确定需要扫描哪些分区，哪些桶的tablet。

· **Join Reorder：**对于 Inner Join, Doris 会根据行数调整表的顺序，将大表放在前面。

· **Sort + Limit 优化成 TopN：**对于order by limit语句会转换成TopN的操作节点，方便统一处理。

· **MaterializedView 选择：**会根据查询需要的列，过滤，排序和 Join 的列，行数，列数等因素选择最佳的物化视图。

图9展示了优化的示例，Doris是在生成关系代数的过程中优化，边生成边优化。



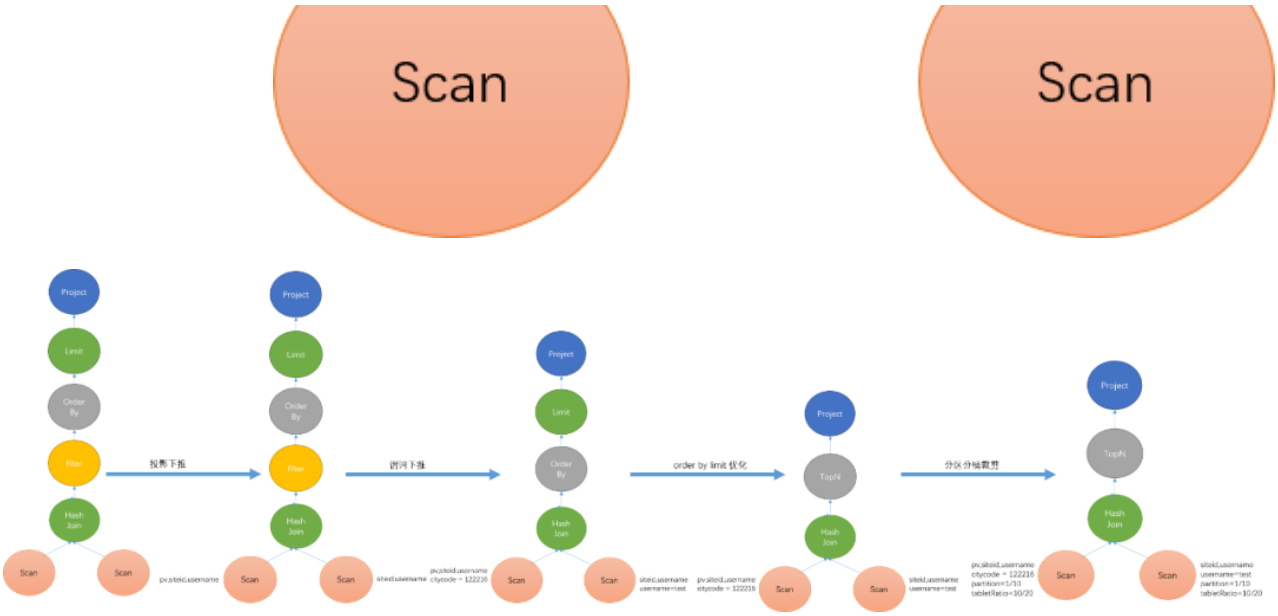



**图 9 单机查询计划优化的过程**

### 8 生成分布式Plan阶段

有了单机的PlanNode树之后，就需要进一步根据分布式环境，拆成分布式PlanFragment树（PlanFragment用来表示独立的执行单元），毕竟一个表的数据分散地存储在多台主机上，完全可以让一些计算并行起来。

这个步骤的主要目标是最大化并行度和数据本地化。主要方法是将能够并行执行的节点拆分出去单独建立一个PlanFragment，用ExchangeNode代替被拆分出去的节点，用来接收数据。拆分出去的节点增加一个DataSinkNode，用来将计算之后的数据传送到ExchangeNode中，做进一步的处理。

这一步采用递归的方法，自底向上，遍历整个PlanNode树，然后给树上的每个叶子节点创建一个PlanFragment，如果碰到父节点，则考虑将其中能够并行执行的子节点拆分出去，父节点和保留下来的子节点组成一个parent PlanFragment。拆分出去的子节点增加一个父节点DataSinkNode组成一个child PlanFragment，child PlanFragment指向parent PlanFragment。这样就确定了数据的流动方向。

对于查询操作来说，join操作是最常见的一种操作。

**Doris目前支持4种join算法：**broadcast join，hash partition join，colocate join，bucket shuffle join。

**broadcast join：**将小表发送到大表所在的每台机器，然后进行hash join操作。当一个表扫描出的数据量较少时，计算broadcast join的cost，通过计算比较hash partition的cost，来选择cost最小的方式。

**hash partition join：**当两张表扫描出的数据都很大时，一般采用hash partition join。它遍历表中的所有数据，计算key的哈希值，然后对集群数取模，选到哪台机器，就将数据发送到这台机器进行hash join操作。

**colocate join：**两个表在创建的时候就指定了数据分布保持一致，那么当两个表的join key与分桶的key一致时，就会采用colocate join算法。由于两个表的数据分布是一样的，那么hash join操作就相当于在本地，不涉及到数据的传输，极大提高查询性能。

**bucket shuffle join：**当join key是分桶key，并且只涉及到一个分区时，就会优先采用bucket shuffle join算法。由于分桶本身就代表了数据的一种切分方式，所以可以利用这一特点，只需将右表对左表的分桶数hash取模，这样只需网络传输一份右表数据，极大减少了数据的网络传输，如图10所示。

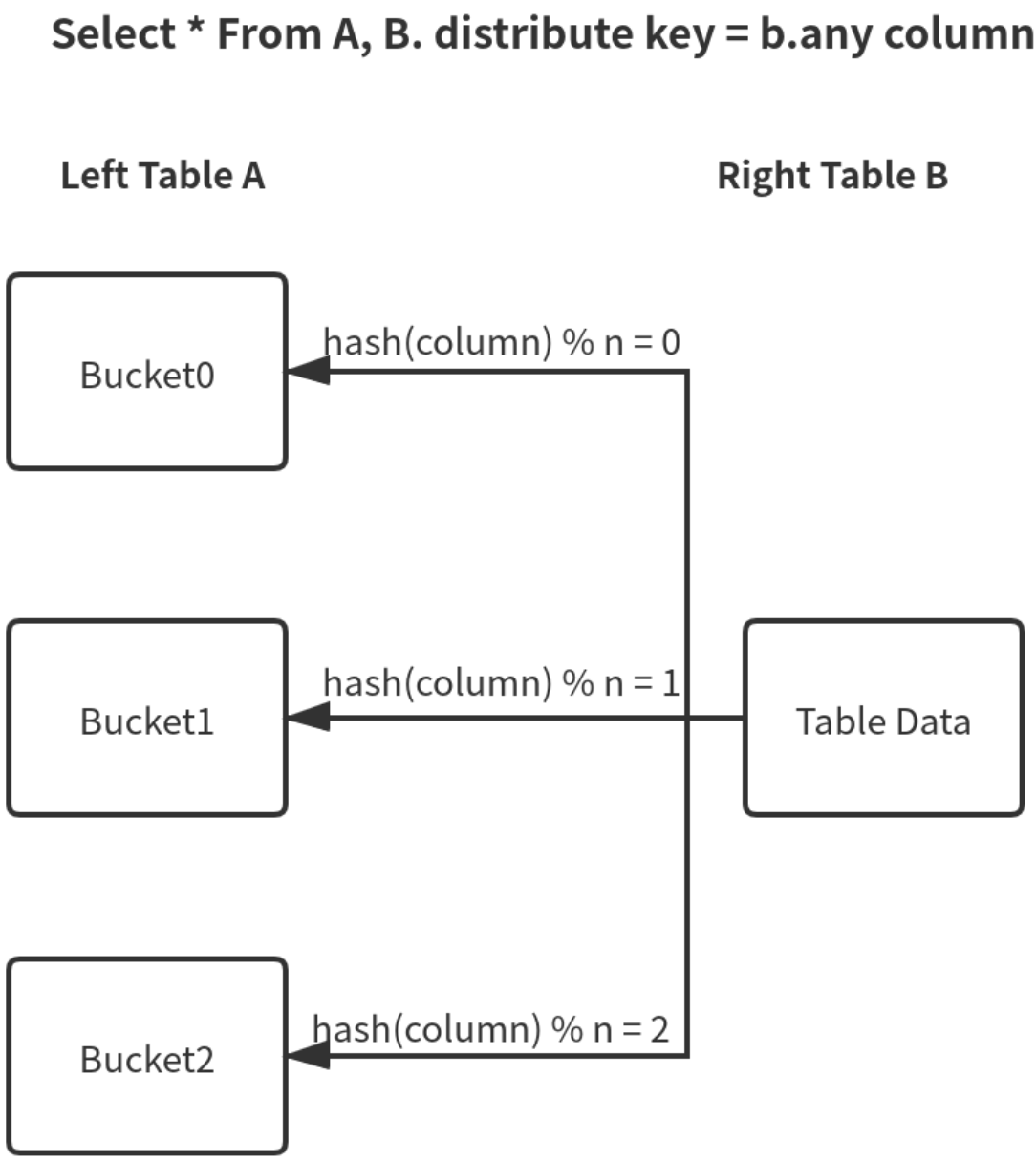



**图 10 bucket shuffle join示例**

如图11展示了带有HashJoinNode的单机逻辑计划创建分布式逻辑计划的核心流程。

· 对PlanNode，自底向上创建PlanFragment。

· 如果是ScanNode，则直接创建一个PlanFragment，PlanFragment的RootPlanNode是这个ScanNode。

· 如果是HashJoinNode，则首先计算下broadcastCost，为选择boracast join还是hash partition join提供参考。

· 根据不同的条件判断选择哪种Join算法

· 如果使用colocate join，由于join操作都在本地，就不需要拆分。设置HashJoinNode的左子节点为leftFragment的RootPlanNode，右子节点为rightFragment的RootPlanNode，与leftFragment共用一个PlanFragment，删除掉rightFragment。

· 如果使用bucket shuffle join，需要将右表的数据发送给左表。所以先创建了一个ExchangeNode，设置HashJoinNode的左子节点为leftFragment的RootPlanNode，右子节点为这个ExchangeNode，与leftFragment共用一个PlanFragment，并且指定rightFragment数据发送的目的地为这个ExchangeNode。

· 如果使用broadcast join，需要将右表的数据发送给左表。所以先创建了一个ExchangeNode，设置HashJoinNode的左子节点为leftFragment的RootPlanNode，右子节点为这个ExchangeNode，与leftFragment共用一个PlanFragment，并且指定rightFragment数据发送的目的地为这个ExchangeNode。

· 如果使用hash partition join，左表和右边的数据都要切分，需要将左右节点都拆分出去，分别创建left ExchangeNode, right ExchangeNode，HashJoinNode指定左右节点为left ExchangeNode和 right ExchangeNode。单独创建一个PlanFragment，指定RootPlanNode为这个HashJoinNode。最后指定leftFragment, rightFragment的数据发送目的地为left ExchangeNode, right ExchangeNode。

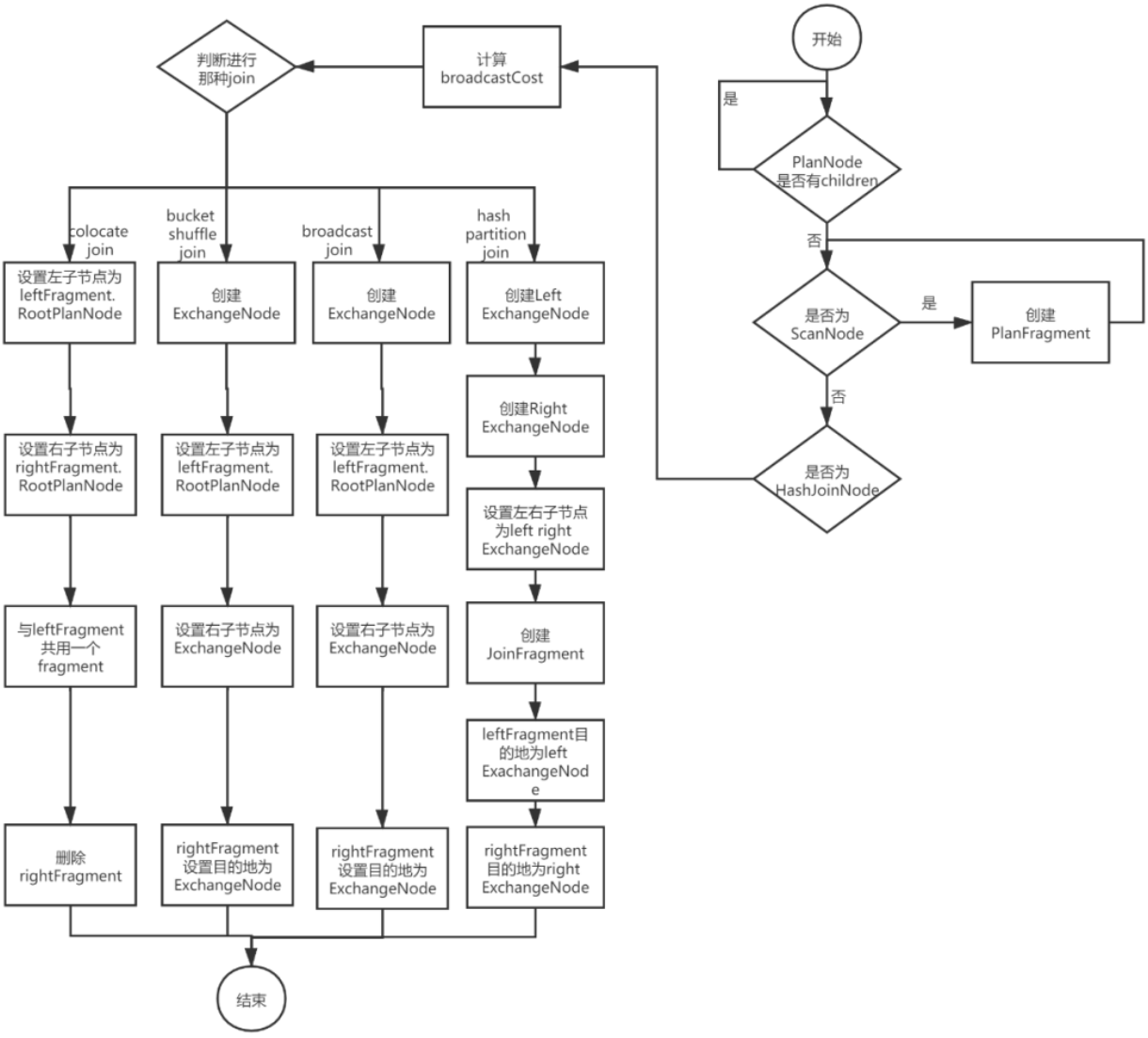



**图 11 HashJoinNode创建分布式逻辑计划核心流程**

图12是两个表的join操作转换成PlanFragment树之后的示例，一共生成了3个PlanFragment。最终数据的输出通过ResultSinkNode节点。

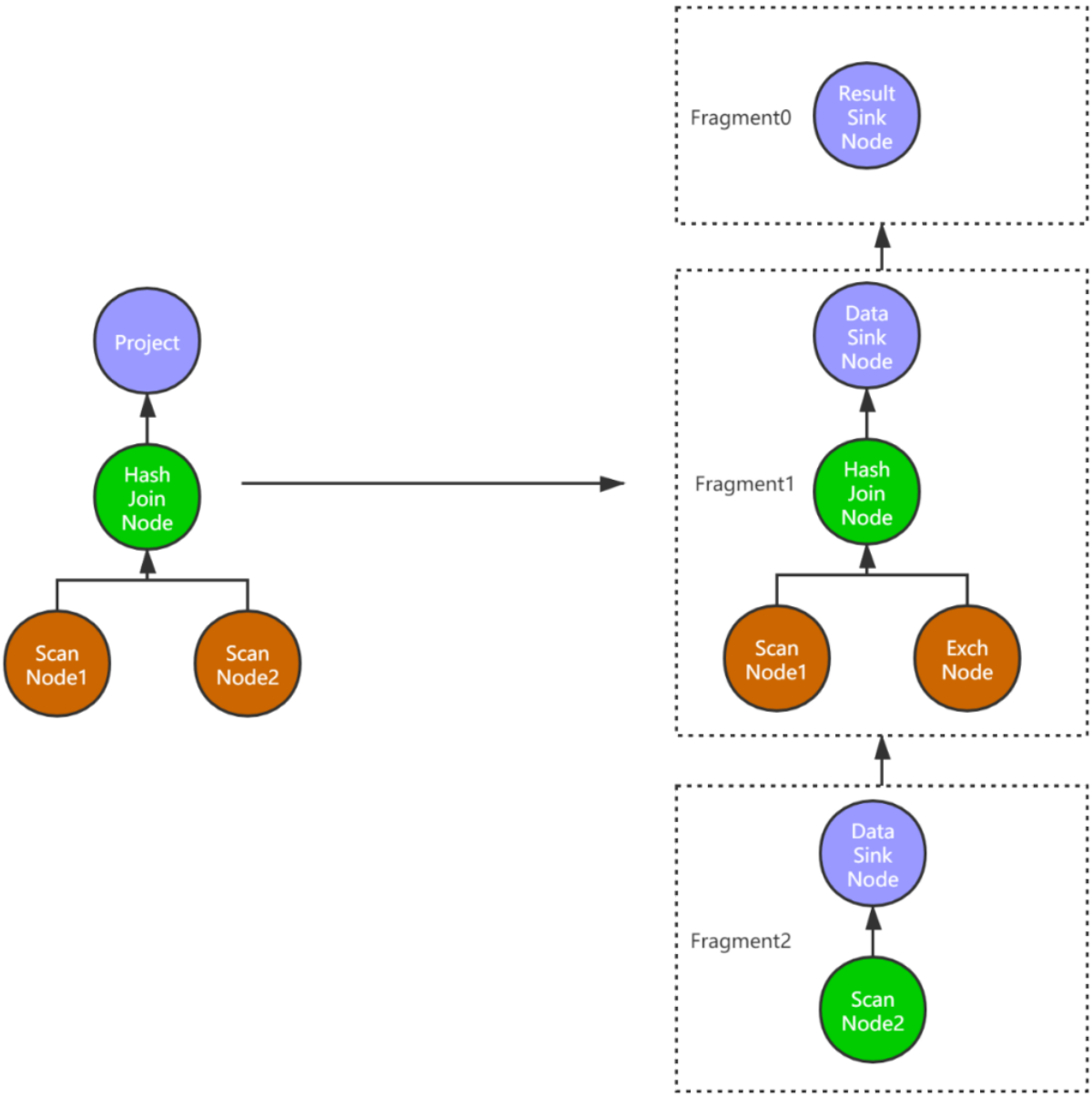



**图 12 从单机计划到分布式计划**

### 9 Schedule阶段

这一步是根据分布式逻辑计划，创建分布式物理计划。主要解决以下问题：

- 哪个 BE 执行哪个 PlanFragment
- 每个 Tablet 选择哪个副本去查询
- 如何进行多实例并发

**图13展示了创建分布式物理计划的核心流程：**

**a. prepare阶段：**给每个PlanFragment创建一个FragmentExecParams结构，用来表示PlanFragment执行时所需的所有参数；如果一个PlanFragment包含有DataSinkNode，则找到数据发送的目的PlanFragment，然后指定目的PlanFragment的FragmentExecParams

的输入为该PlanFragment的FragmentExecParams。

**b. computeScanRangeAssignment阶段：**针对不同类型的join进行不同的处理。

- **computeScanRangeAssignmentByColocate：**针对colocate join进行处理，由于join的两个表桶中的数据分布都是一样的，他们是基于桶的join操作，所以在这里是确定每个桶选择哪个host。在给host分配桶时，尽量保证每个host分配到的桶基本平均。
- **computeScanRangeAssignmentByBucket：**针对bucket shuffle join进行处理，也只是基于桶的操作，所以在这里是确定每个桶选择哪个host。在给host分配桶时，同样需要尽量保证每个host分配到的桶基本平均。
- **computeScanRangeAssignmentByScheduler：**针对其他类型的join进行处理。确定每个scanNode读取tablet哪个副本。一个scanNode会读取多个tablet，每个tablet有多个副本。为了使scan操作尽可能分散到多台机器上执行，提高并发性能，减少IO压力，Doris采用了Round-Robin算法，使tablet的扫描尽可能地分散到多台机器上去。例如100个tablet需要扫描，每个tablet 3个副本，一共10台机器，在分配时，保障每台机器扫描10个tablet。

**c. computeFragmentExecParams阶段：**这个阶段解决PlanFragment下发到哪个BE上执行，以及如何处理实例并发问题。确定了每个tablet的扫描地址之后，就可以以地址为维度，将FragmentExecParams生成多个实例，也就是FragmentExecParams中包含的地址有多个，就生成多个实例FInstanceExecParam。如果设置了并发度，那么一个地址的执行实例再进一步的拆成多个FInstanceExecParam。针对bucket shuffle join和colocate join会有一些特殊处理，但是基本思想一样。FInstanceExecParam创建完成后，会分配一个唯一的ID，方便追踪信息。如果FragmentExecParams中包含有ExchangeNode，需要计算有多少senders，以便知道需要接受多少个发送方的数据。最后FragmentExecParams确定destinations，并把目的地址填充上去。

**d. create result receiver阶段：**result receiver是查询完成后，最终数据需要输出的地方。

**e. to thrift阶段：**根据所有PlanFragment的FInstanceExecParam创建rpc请求，然后下发到BE端执行。这样一个完整的SQL解析过程完成了。

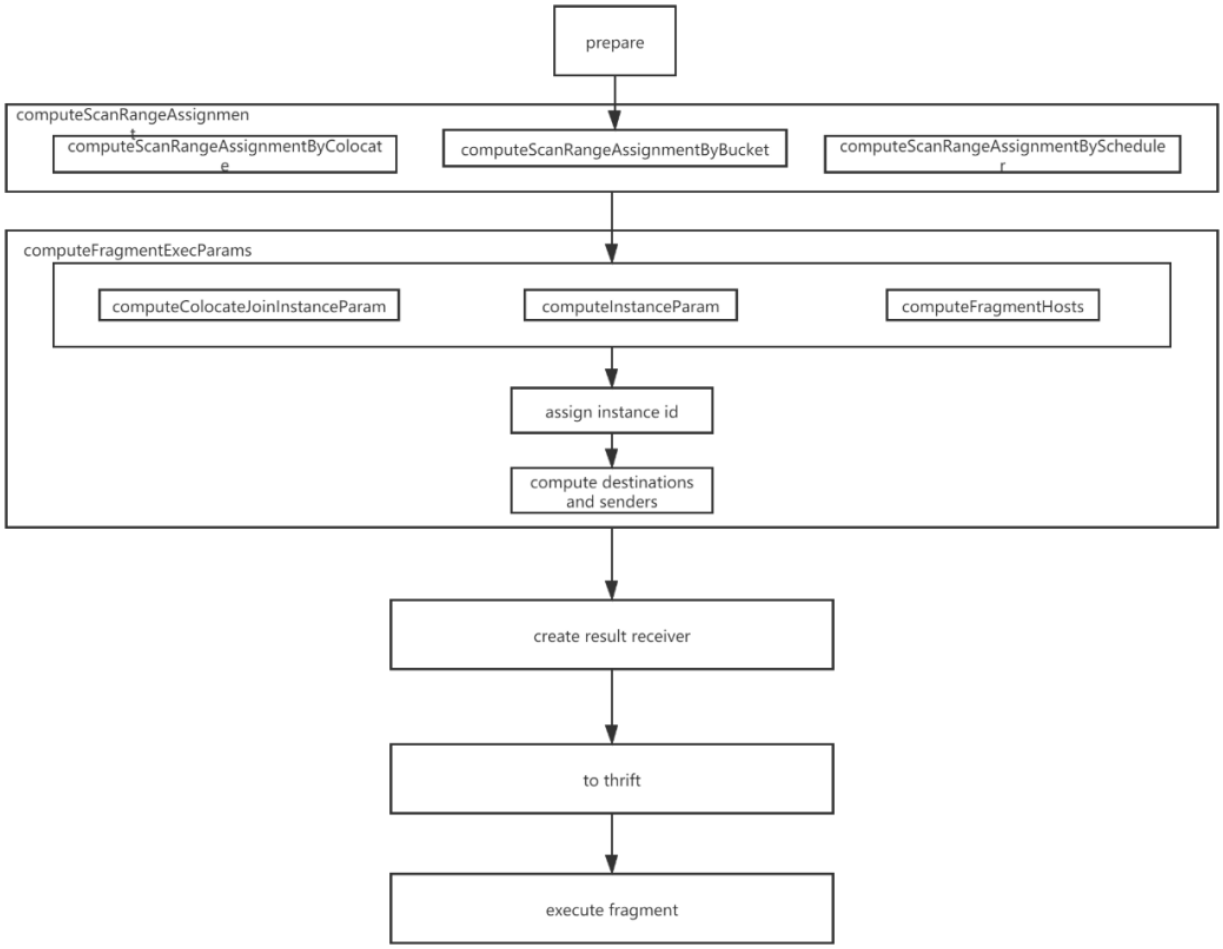



**图 13 创建分布式物理计划核心流程**

如图14所示是一个简单示例，图中的PlanFrament包含了一个ScanNode，ScanNode扫描3个tablet，每个tablet有2副本，集群假设有2台host。

computeScanRangeAssignment阶段确定了需要扫描replica 1,3,5,8,10,12，其中replica 1,3,5位于host1上，replica 8,10,12位于host2上。

如果全局并发度设置为1时，则创建2个实例FInstanceExecParam，下发到host1和host2上去执行，如果如果全局并发度设置为3，这个host1上创建3个实例FInstanceExecParam，host2上创建3个实例FInstanceExecParam，每个实例扫描一个replica，相当于发起6个rpc请求。

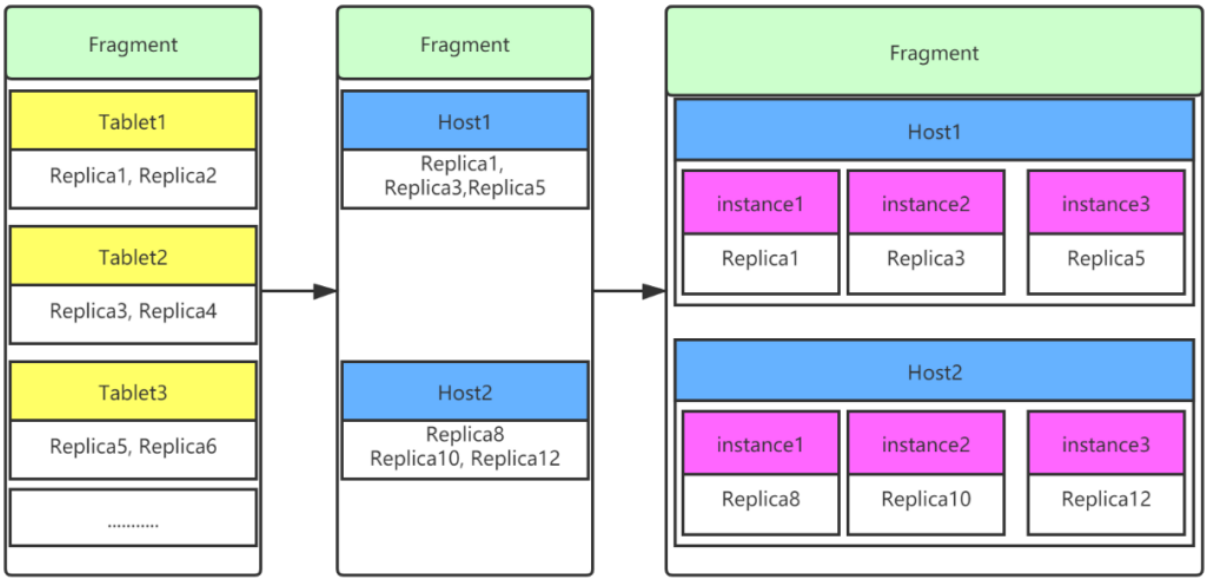



**图 14 生成物理计划的过程**

## 10 总结

本文首先简单介绍了Doris，然后介绍SQL解析的通用流程：词法分析，语法分析，生成逻辑计划，生成物理计划，接着从总体上介绍了Doris在SQL解析这块的总体架构，最后详细讲解了Parse，Analyze，SinglePlan，DistributedPlan，Schedule等5个过程，从算法原理和代码实现上进行了深入的讲解。

Doris遵守了SQL解析的常用方法，但根据底层存储架构，以及分布式的特点，在SQL解析这块进行了大量的优化，实现了最大并行度和最小化网络传输，给SQL执行层面减少很多负担。

欢迎扫码关注：

Apache Doris(incubating)官方公众号

**Apache Doris Github：**

https://github.com/apache/incubator-doris

**Apache Doris 开发者邮件组：**

dev@doris.apache.org