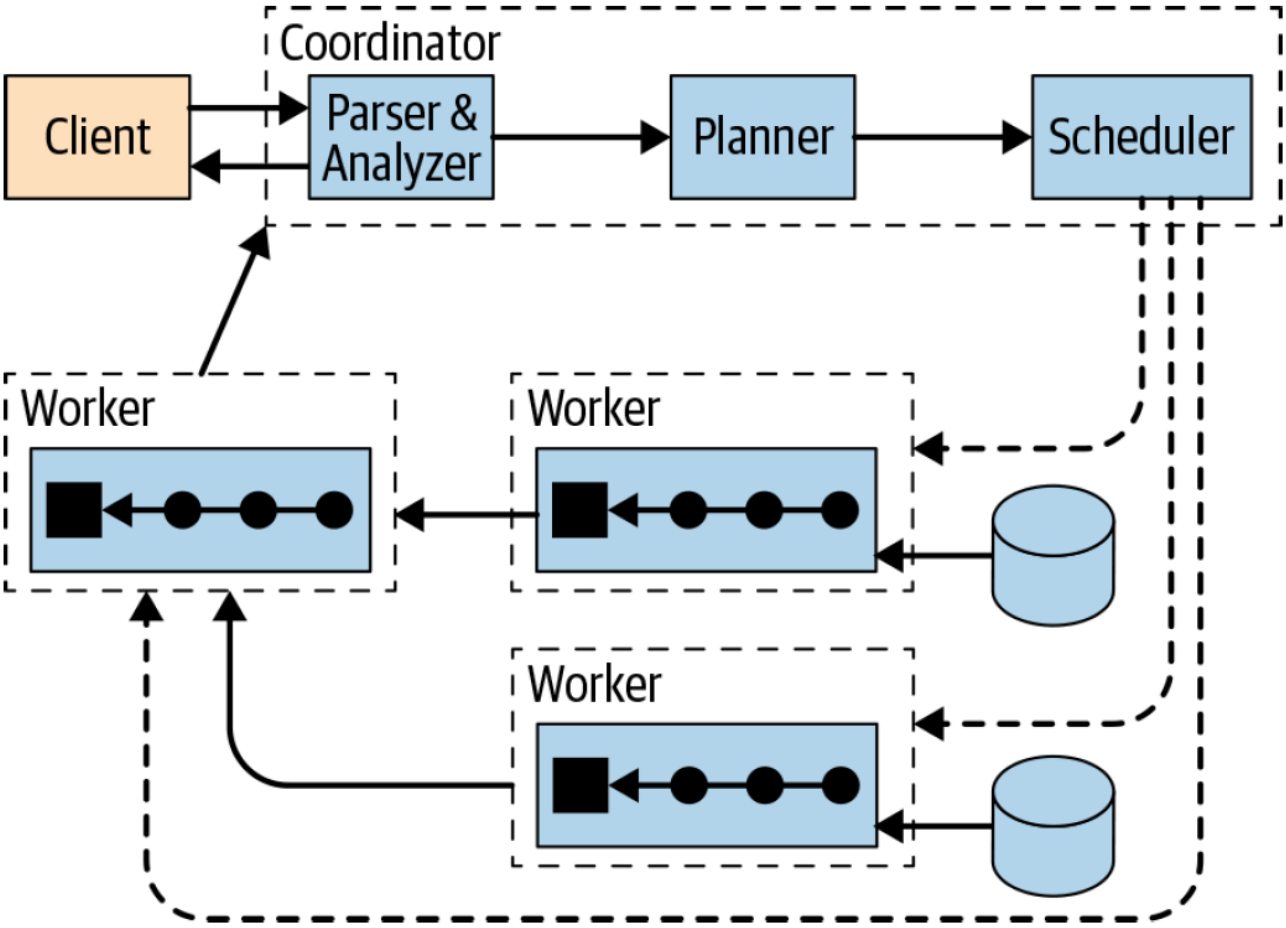
# Presto 计算下推原理与实践

## 背景

在介绍 Presto 计算下推之前，我们先来回顾一下 Presto 从对应的 Connector 上读取数据的流程，过程如下：






从上图可以看出，client 提交 SQL 到 Coordinator 上，Coordinator 接收到 SQL 之后，会进行 SQL 语法语义解析，生成逻辑计划树，然后经过 planner 处理生成物理计划树（这个过程在 这篇 文章里面有介绍），紧接着会生成一个一个有依赖的 Stages，每个 stage 里面有一个或多个 task，这些 task 会被发送到 Worker 上去执行，其中会有一种叫做 Source 的 task，这个 task 就是从对应的数据源里面读取数据，中间结果会发送到其他 worker，最后的计算结果是由 Coordinator 从 worker 上获取再由 Client 获取。

Presto 从数据源读取数据的过程基本可以理解为从数据源读取明细数据（已经经过列裁剪之后的列），然后把明细数据拉到 Worker 上进一步计算。在这个读数据的过程中，Presto 支持把能够用 TupleDomain 表示的 Filter 下推到数据源。比如常见的 a >1、b = 1、a between 1

and 2 这种比较简单的过滤条件下推在所有的数据源都是支持的，只要底层对应的数据源能够支持在读取数据的时候把 filter 带上去就可以使用这个功能。

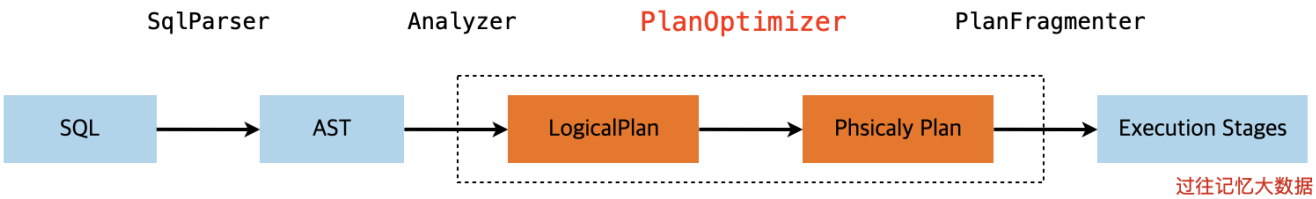
## 复杂算子下推

上面提到 Presto 已经为所有的数据源提供了简单 filter
下推到数据源的能力，但是在大多数场景这个功能其实很有限的。比如我们计算 select sum(a) from mysql.iteblog.tbl，现在的做法是会把 a 的值从 mysql 数据源拉取到 Presto，然后在 Presto 里面计算 sum(a)。如果 tbl 表比较大，那么 MySQL 和 Presto
之间的数据传输可能就会消耗很多时间。如果我们把 sum(a) 的计算放到 MySQL
里面，最后只是把 MySQL 计算出来的 sum(a) 结果传回
Presto，那么整体的计算时间可能会大大减少。

为了做到上面的效果，需要 Presto 从框架层面上提供支持。Presto 从 0.217 到 0.229
版本之间对 Presto planner
的能力进行了优化，使得从架构上支持将更多的算子下推到数据源；相比之前只能把能够用 TupleDomain 表示的 Filter 下推到数据源相比，优化后的 planner 支持将
Filter、Limit、TopN（order by xxx limit N）以及 Aggregation
下推到数据源，从而可以实现上面场景的需求。

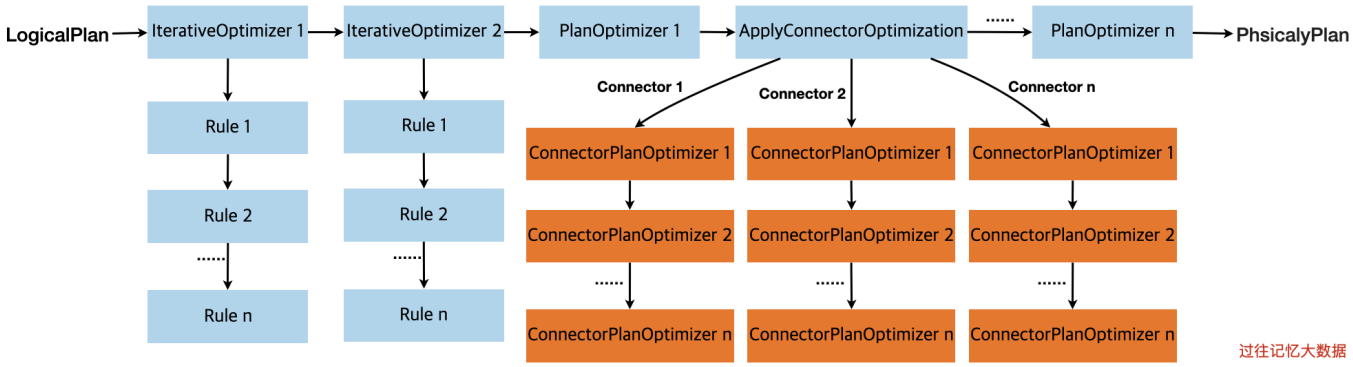
## Presto 计算下推原理

我们前面已经简单介绍了最新版本的 Presto
支持了下推的能力，那么从原理上来说它是如何实现的呢？我们从 SQL 解析开始说起，下面是 Presto 里面 SQL 提交到 Coordinator 端经过解析之后生成可执行的 Stages 的简单过程。




如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

计算下推的过程其实就在逻辑计划到物理计划的过程中，也就是上图虚线框里面的过程中。这个过程的执行大概如下图所示：



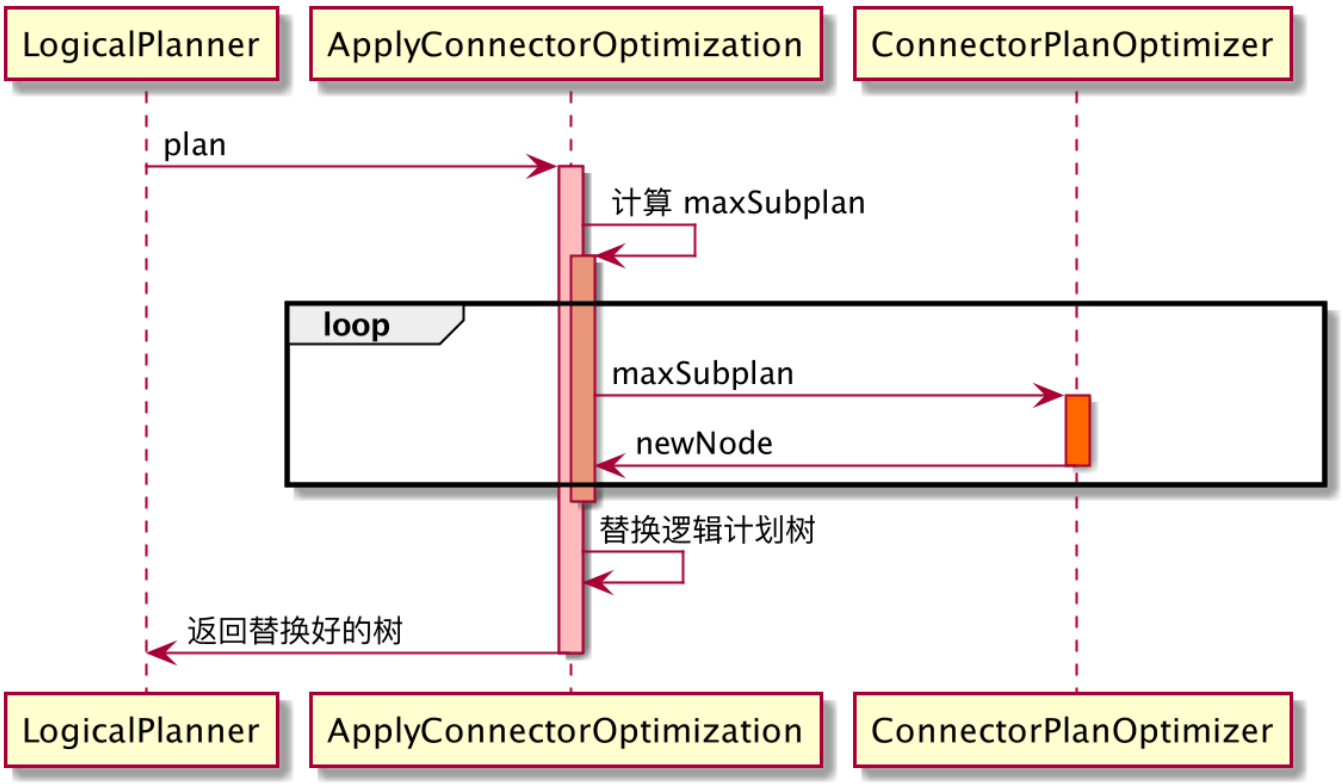

如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

也就是从逻辑计划到物理计划的过程经过了大量的优化规则处理的，这些优化规则全部定义在 PlanOptimizers 里面。主要有两种，IterativeOptimizer 和 PlanOptimizer，这两种是优化器的不同写法，都可以实现对计划树进行优化，其实 IterativeOptimizer 也是实现 PlanOptimizer 接口的类。一个 IterativeOptimizer 里面会包含一个或多个 Rule，每个 Rule 是基于 Pattern 去决定（也就是某个计划树是不是符合对应的 Rule 的 Pattern）要不要处理执行计划树。而除了 IterativeOptimizer 之外的 PlanOptimizer 实现类都是基于访问者模式（比如 visitLimit、visitFilter）来处理执行计划树的。

从上图可以看到，里面有个 ApplyConnectorOptimization，其实 Presto 的计算下推就通过这个类把相关的算子下推到数据源层面的。ApplyConnectorOptimization 是实现 PlanOptimizer 接口的优化器。在 ApplyConnectorOptimization 里面可以拿到各个数据源定义好的下推逻辑（实现 ConnectorPlanOptimizer 接口），每个数据源可以定义多个 ConnectorPlanOptimizer 实现类。ConnectorPlanOptimizer 接口定义如下：

```
public interface ConnectorPlanOptimizer
{
   PlanNode optimize(
         PlanNode maxSubplan,
         ConnectorSession session,
         VariableAllocator variableAllocator,
         PlanNodeIdAllocator idAllocator);
}
```

其中 maxSubplan 就是下推前的执行计划树的一部分，optimize 返回的是下推后的执行计划树。下面是 Presto 计算下推的简单执行流程图：




过往记忆大数据

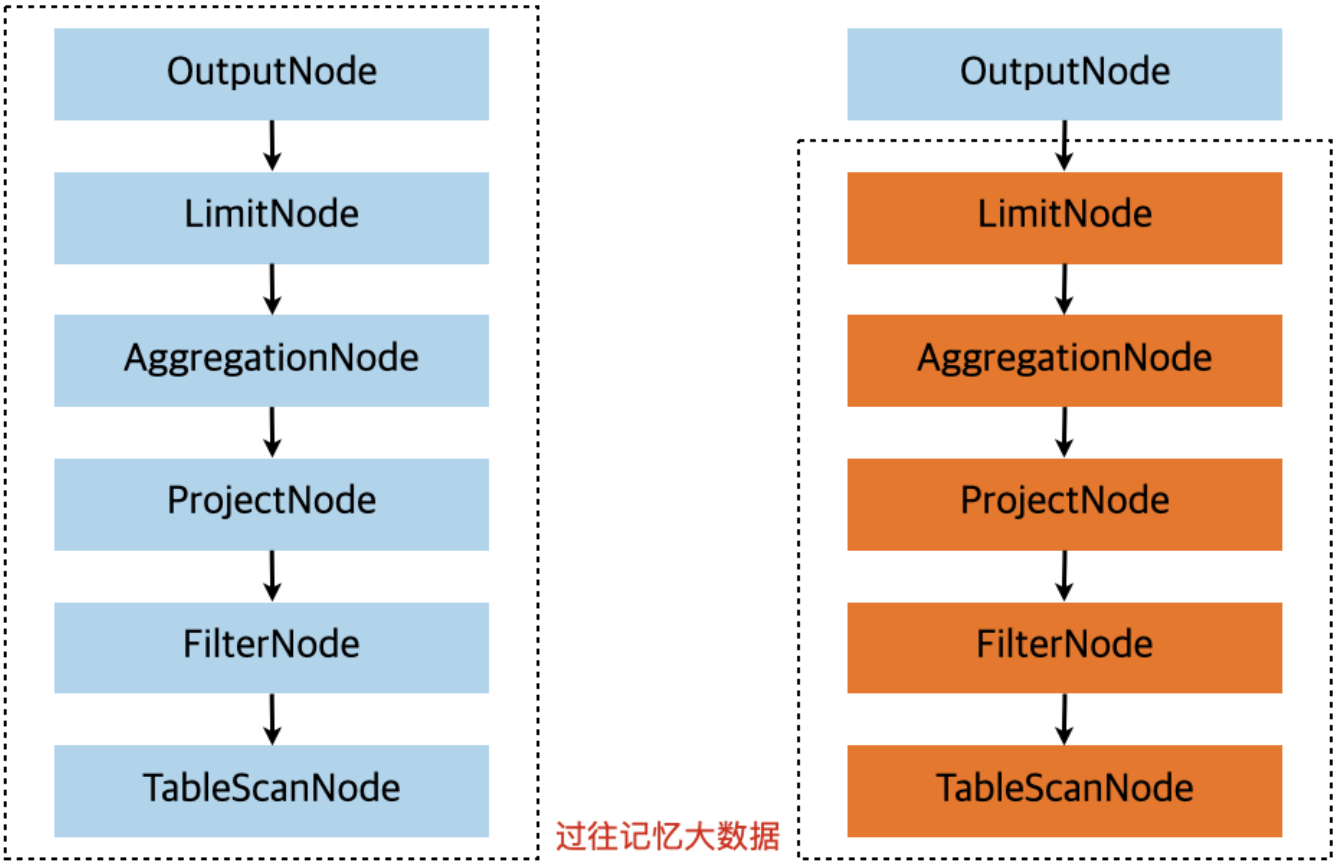
如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

我们前面说过 Presto 里面所有的优化器实现规则都是定义在 PlanOptimizers 里面的，而把这些优化器应用到 PlanNode 上面是在 LogicalPlanner 上做的。LogicalPlanner 里面会循环遍历定义在 PlanOptimizers 里面的一个一个优化规则。当遍历到 ApplyConnectorOptimization 的时候，从 LogicalPlanner 传递到 ApplyConnectorOptimization 里面是下推之前的 plan，在 ApplyConnectorOptimization 里面会先计算出一个 maxSubplan；然后把 maxSubplan 传递到具体数据源定义好的 ConnectorPlanOptimizer 里面，在 ConnectorPlanOptimizer 里面会执行具体的下推逻辑，然后返回一个新的 newNode 到 ApplyConnectorOptimization。如果这个数据源定义了多个 ConnectorPlanOptimizer，会循环遍历的，最后 ApplyConnectorOptimization 会把下推后的执行计划树返回到 LogicalPlanner。

那么 maxSubplan 是什么？和 plan 有什么关系？这里假设我们有以下查询：

```
select count(*) from lineitem where l_orderkey = '4281473' limit 10;
```

下图的左边部分是这个 SQL 的逻辑计划树，整个虚线框部分就是前面说的 plan；而右边部分的虚线框里面其实就是前面说的 maxSubplan，也就是说传到数据源层面上的 PlanNode 也就是右图虚线框里面，只是整个物理计划的一部分。




如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

另外，前面说的其实是对逻辑计划树在数据源层面进行处理的，其实我们也可以将物理计划树传递到数据源层面进行处理，这个时候传进来的 maxSubplan 和 逻辑计划树的 maxSubplan 不一样，具体我就不介绍了。

## Presto 计算下推实践

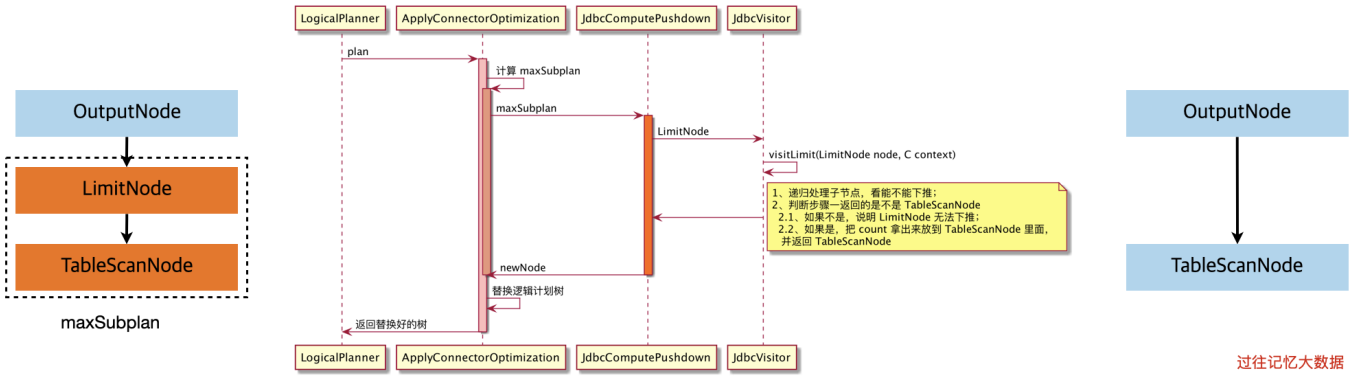
前面说了那么多，那我们如何实现计算下推？这里通过几个例子来简单说明过程。

### 简单 Limit 下推

这里我们使用 MySQL 数据源来进行说明。我们测试的 SQL 如下：

```
select * from lineitem limit 10;
```

下图左边是这个查询的逻辑执行计划树，虚线框里面就是 maxSubplan，也就是传递到 JdbcComputePushdown 里面 optimize 方法里面的。




如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

Limit 下推的逻辑大概如下：

- 先处理 LimitNode 的子节点，看子节点能不能下推；因为我们这里是比较简单的 SQL，所以 LimitNode 的子节点就是 TableScanNode；如果 SQL 带有 Filter，LimitNode 的子节点就是 FilterNode，这时候其实会去调用 visitFilter 去处理;
- 处理完子节点之后，会看下返回的 planNode 是不是 TableScanNode；如果不是说明子节点是不能下推的，那 LimitNode 就不能下推了；
- 如果子节点可以下推，也就是返回 TableScanNode，那么我们把 LimitNode 里面的 count 拿出来，放到 TableScanNode 里面去，然后返回新建的 TableScanNode。

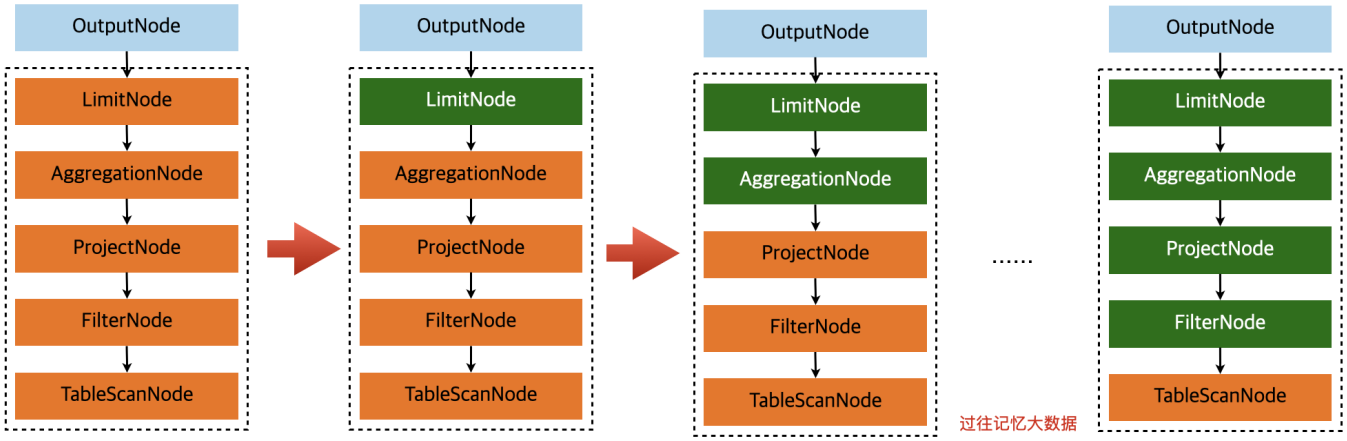
所以经过 Limit 下推处理之后，上图右边就是下推之后的逻辑计划树，可以看到，相比左边的逻辑计划树，右边那个少了 LimitNode 节点，这是因为 LimitNode 里面的信息存放在 TableScanNode 里面了。对这部分感兴趣的可以参见 这里。
TableScanNode

## 带聚合的算子下推

这里同样使用 MySQL 数据源来进行说明。测试的 SQL 如下：

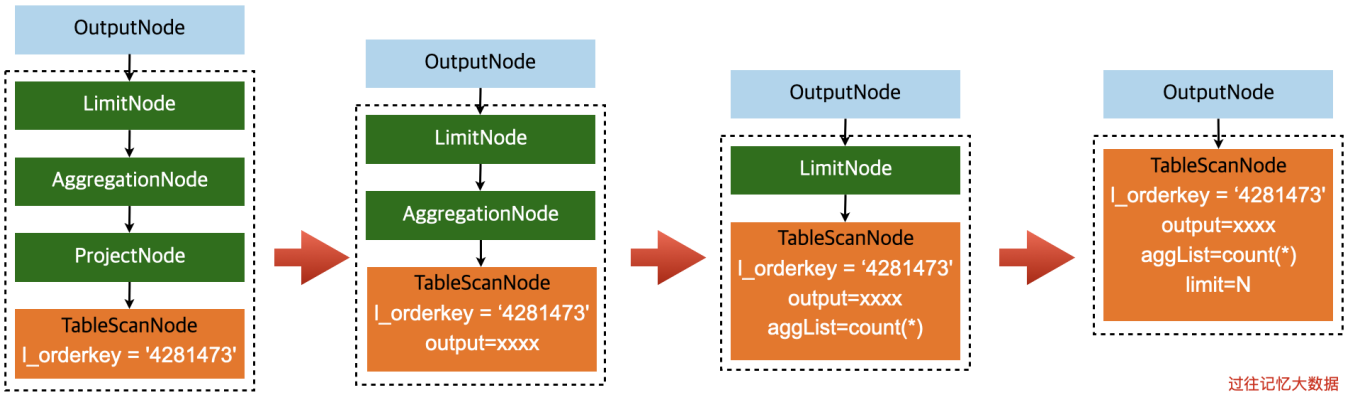
```
select count(*) from lineitem where l_orderkey = '4281473' limit 10;
```

这个 SQL 的逻辑计划树和 maxSubplan 在上一节已经给出来了。这个 SQL 在 MySQL 数据源层面上执行计算下推的流程图如下：






传到 MySQL 数据源层面的 maxSubplan
是上图最左边的虚线框里面的部分，所以我们最先拿到了 LimitNode，在处理 LimitNode 的时候，需要先处理其子节点 AggregationNode；同理处理 AggregationNode 节点之前需要处理 AggregationNode 的子节点 ProjectNode；程序最后处理到 FilterNode 节点，FilterNode 的子节点是 TableScanNode，不需要再往下走了，所以可以直接处理 FilterNode 了，看可不可以下推。在我们的例子中，FilterNode 里面的东西其实就是 l_orderkey = '4281473'，是可以下推的，这时候我们可以把这个信息抽出来，并存储到 TableScanNode 里面；然后返回到 ProjectNode 里面，把 ProjectNode 里面的信息存储到 TableScanNode 里面；同理，按照这个逻辑处理 AggregationNode、LimitNode 节点。过程如下：






可以看到，经过下推之后，最后返回的只有 TableScanNode，里面存储了下推必要的信息。

## 实现部分

截止到 Presto 最新版本，社区对 JDBC 数据源只是实现了一些 Filter 的下推，参见 #13526，这个看起来其实更类似于一个 DEMO，功能不是很完备。而 Trino 社区其实对 JDBC 数据源做了 Limit、TopN、Aggregation 以及 Join 下推，相比 PrestoDB 还是比较完善的。

不过阿里云数据湖分析团队基于 PrestoDB JDBC 数据源已有的 Filter 下推功能做了大量的优化，支持了 Limit、TopN 以及 Aggregation 下推，相比 Trino 而言在 Filter 下推这块做了一些改进，比如支持部分 Filter 下推（#16412）。同时阿里云数据湖分析团队已经把 JDBC 数据源的 Limit、TopN 等下推的代码开源出来了，参见 #16570。Aggregation 下推其实也已经开发完成，相关代码开源需要等 #16570 合并完成。相比 Trino 社区的 Aggregation 下推，支持聚合参数里面有复杂的表达式，比如下面整个 SQL 都可以下推到 MySQL 执行，而 Trino 目前还不支持这个。

```
select
   l_returnflag,
   l_linestatus,
   sum(l_quantity) as sum_qty,
   sum(l_extendedprice) as sum_base_price,
   sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
   sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
   avg(l_quantity) as avg_qty,
   avg(l_extendedprice) as avg_price,
   avg(l_discount) as avg_disc,
   count(*) as count_order
from
   lineitem
where
   l_shipdate <= date '1998-12-01' - interval '93' day
group by
   l_returnflag,
   l_linestatus
order by
   l_returnflag,
   l_linestatus;
```

除了 JDBC 数据源下推功能在走开源流程之外，MongoDB（#16470）以及 TableStore（#16467）数据源下推的功能也会陆续开源出来。

另外，阿里云数据湖分析团队也在开发 Join 下推的功能，也就是把整个 Join 语句下推到对应的数据源。不过目前 PrestoDB 在框架上还不支持 Join 下推，不过 #16583 这个 MR 是提供了 Join 下推的能力的，有了他，我们就可以到数据源层面上操作 JoinNode 节点，并实现 Join 下推的功能。

## 总结

PrestoDB 已经给我们实现了下推的框架，基于它可以很容易地实现对应数据源的下推功能。不过

目前要实现数据源的下推需要到每个数据源里面去实现，可能会存在一定的代码重复。