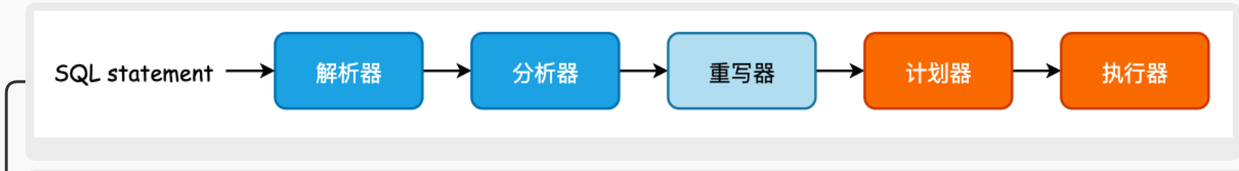
为什么需要 Motion

Pre-Content



首先，不管是 PostgreSQL 还是 MySQL，或者是 Greenplum，它们对查询处理的方式都逃不过上图中的 5 个基本步骤，这是 RDBMS 的查询处理基石

其中解析器对一条 SQL 进行语法解析，得到一棵语法解析树；分析器则是对语法解析树进行语义分析，得到一棵查询树；重写器则是按照既定的规则对查询树进行重写；计划树则是基于查询树来生成一棵执行效率最高的计划树，这也是数据库最为复杂的一部分；执行器则按照计划树的顺序访问表和索引，执行相应的查询

在 PG 中，我们可以使用 ` EXPLAIN SQL ` 来得到一条 SQL 的执行计划，这个执行计划其实就是计划器生成的计划树的简化版本。在 MySQL 中可以通过 ` EXPLAIN FORMAT=tree SQL` 的方式得到类似的计划树

我们用一张非常非常简单的 table 来说明问题

- CREATE TABLE t (a int);
- CREATE TABLE t (a int) distributed by (a);

在单机 PostgreSQL 和 Grenplum 中建立了相同的 table，并且 table t 在 Greenplum 中根据字段 ` a` 进行哈希分布

关于 Join

- Join 在 RDBMS 中通常会有 3 种实现方式: Nested-Loop Join、Hash Join 以及 Merge Join。在 OLAP 种绝大多数的 Join 都会使用 Hash Join，因为 OLAP 中不是所有的字段都会存在索引
- 关于 Join 的细节实现可参考： https://smartkeyerror.oss-cn-shenzhen.aliyuncs.com/Phyduck/database/MySQL%20JOIN%20%E5%B7%A5%E4%BD%9C%E5%8E%9F%E7%90%86%E6%B5%85%E6%9E%90.pdf

Simple Query

```
explain select * from t;
```

PostgreSQL

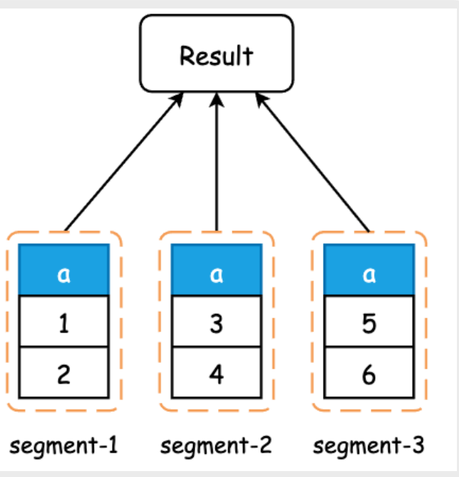
```
                        QUERY PLAN
-----------------------------------------------------
Seq Scan on t  (cost=0.00..35.50 rows=2550 width=4)
```

Greenplum

```
                                        QUERY PLAN
-------------------------------------------------------------------------------
Gather Motion 3:1  (slice1; segments: 3)  (cost=0.00..1063.00 rows=96300 width=4)
  ->  Seq Scan on t  (cost=0.00..1063.00 rows=32100 width=4)
Optimizer: Postgres query optimizer
```

可以看到，在 PG 的执行计划中，只需要扫描一遍表 t 就能够得到最终结果，而在 Greenplum 的查询计划中却多出了一个 ` Gather Motion 3:1` 节点，其含义将在后续描述

由于表 t 在 Greenplum 是分布式存储的，并且分片字段为 ` a` ，因此数据会大致地均匀分布在各个 segment 节点上（本例中为 3 个 segment 节点）。那么如果我们想要获取全部数据的话，就必须访问所有的 segment 节点



如左图所示，由于数据是根据某一个分片键并使用某种规则分布式存储的，那么当我们执行诸如 select * from t 这样的操作时，就必然存在数据的移动，这是分布式查询中绕不开的一个步骤

更进一步地，如果我们执行的是 select * from t where a < 1024; 的话，就需要一次额外的 Filter，这个 Filter 同样必须在所有的 segment 上执行

Gather Motion

实际上，Gather Motion 的含义就是收集数据，Gather 有收集之意，Motion 的本意为移动，我们就可以将 Gather Motion 理解成"收集移动的数据"

数据移动

Join Query

首先，我们需要另外创建两张 table

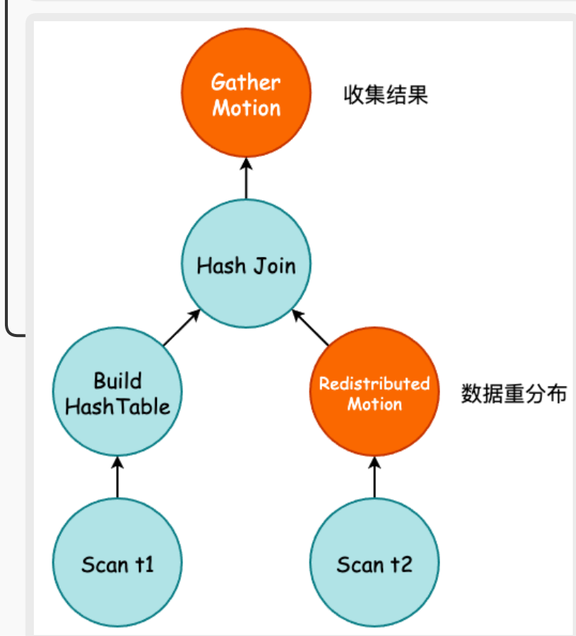
```
create table t1 (a int, b int) distributed by (a);
create table t2 (a int, b int) distributed by (b);
```

```
explain select * from t1, t2 where t1.a = t2.a;
```

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
Gather Motion 3:1  (slice2; segments: 3)  (cost=2037.25..1005718.85 rows=7413210 width=16)
  ->  Hash Join  (cost=2037.25..1005718.85 rows=2471070 width=16)
        Hash Cond: (t2.a = t1.a)
        ->  Redistribute Motion 3:3  (slice1; segments: 3)  (cost=0.00..2683.00 rows=28700 width=8)
              Hash Key: t2.a
              ->  Seq Scan on t2  (cost=0.00..961.00 rows=28700 width=8)
        ->  Hash  (cost=961.00..961.00 rows=28700 width=8)
              ->  Seq Scan on t1  (cost=0.00..961.00 rows=28700 width=8)
Optimizer: Postgres query optimizer
```

如上查询计划所示，由于 t1 是通过 a 分布的，而 t2 则是通过 b 进行分布的，那么即使 t1 和 t2 拥有完全相同的数据，它们在同一个 segment 上的数据也是不一样的。因此，如果想要对其进行 join 操作的话，就必须将 t1 或者是 t2 的数据进行重分布

重分布的含义也很简单，以 t2 为例，只需要扫描一遍 t2，然后再对其根据字段 a 进行哈希重分布即可



分布式查询计划的本质其实就是数据在各个节点之间的移动

综上，当一个 segment 实例执行某些查询时，如果仅考虑自身的数据的话，将无法得到正确的查询结果，必须要对数据进行重新分布，Greenplum 把这个数据的移动抽象成了 Motion，包括 Gather Motion、Redistributed Motion 以及 Broadcast Motion