阿里云

PostgreSQL优化器浅析

康贤

阿里云

目录

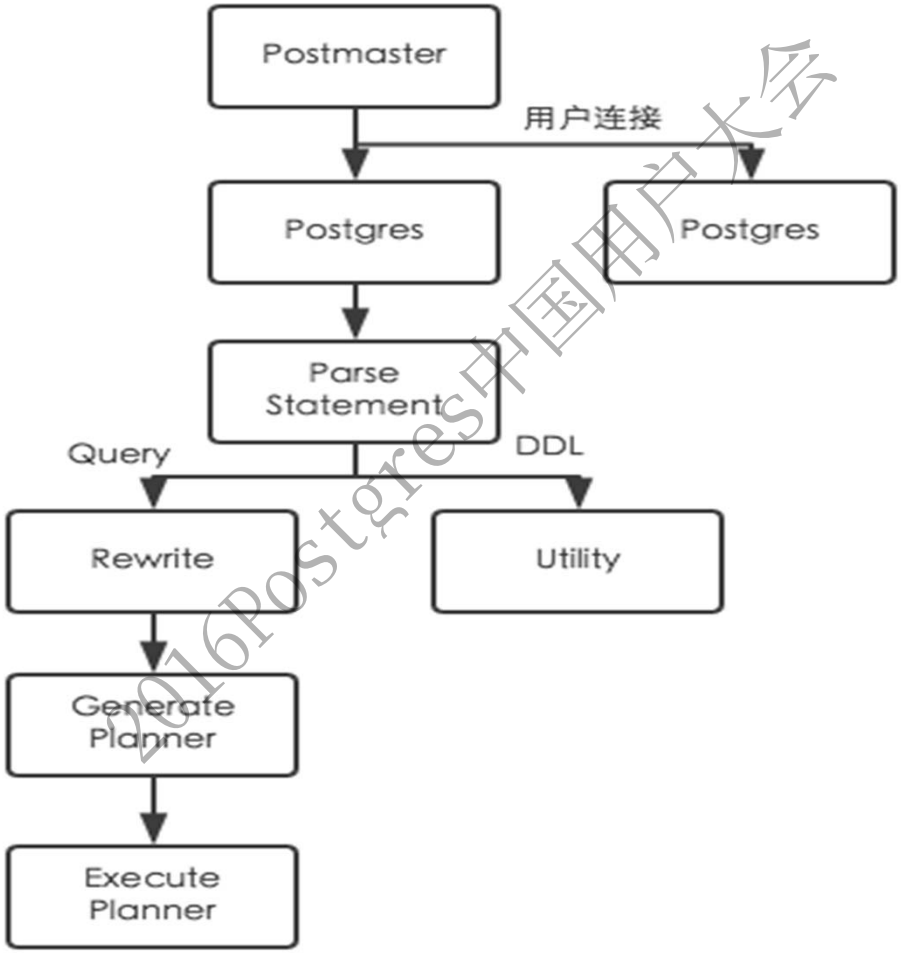
- 优化器介绍
- 逻辑优化
- 物理优化

优化器介绍

- 什么是优化器
  - 数据库的大脑

- SQL处理过程

优化器介绍



优化器介绍

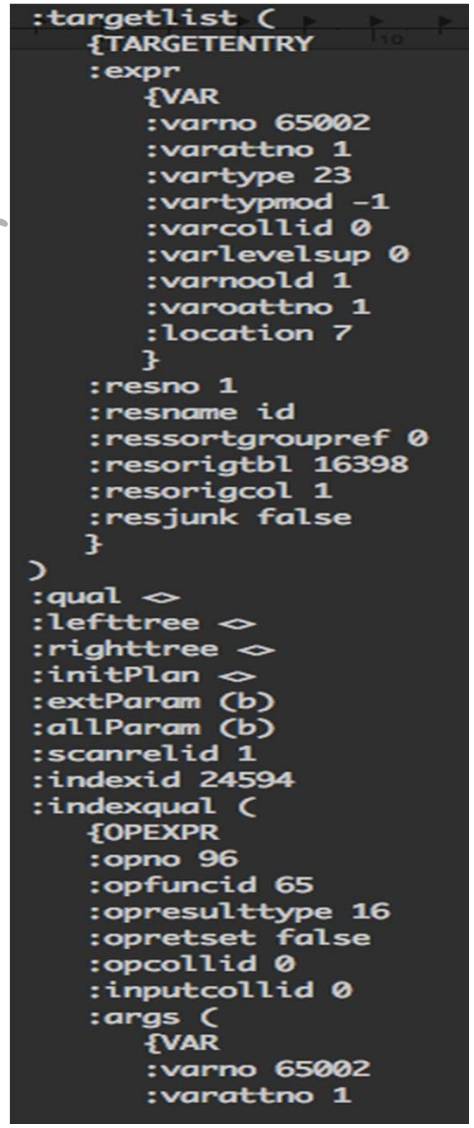
- 查看优化器生成计划
  - explain query，如下

```
postgres=# explain select * from test where id = 10 limit 1;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Limit  (cost=0.28..8.29 rows=1 width=4)
   ->  Index Only Scan using test11 on test  (cost=0.28..8.29 rows=1 width=4)
         Index Cond: (id = 10)
(3 rows)
```

优化器介绍

- Plan结构

```
{PLANNEDSTMT
:commandType 1
:queryId 0
:hasReturning false
:hasModifyingCTE false
:canSetTag true
:transientPlan false
:planTree
   {LIMIT
   :startup_cost 0.28
   :total_cost 8.29
   :plan_rows 1
   :plan_width 4
   :targetlist (
      {TARGETENTRY
      :expr
         {VAR
         :varno 65001
         :varattno 1
         :vartype 23
         :vartypmod -1
         :varcollid 0
         :varlevelsup 0
         :varnoold 1
         :varoattno 1
         :location -1
         }
      :resno 1
      :resname id
      :ressortgroupref 0
      :resorigtbl 16398
      :resorigcol 1
      :resjunk false
      }
   )
   :qual <>
   :lefttree
      {INDEXONLYSCAN
      :startup_cost 0.28
      :total_cost 8.29
      :plan_rows 1
      :plan_width 4
```



```
:targetlist (
   {TARGETENTRY
   :expr
      {VAR
      :varno 65002
      :varattno 1
      :vartype 23
      :vartypmod -1
      :varcollid 0
      :varlevelsup 0
      :varnoold 1
      :varoattno 1
      :location 7
      }
   :resno 1
   :resname id
   :ressortgroupref 0
   :resorigtbl 16398
   :resorigcol 1
   :resjunk false
   }
)
:qual <>
:lefttree <>
:righttree <>
:initPlan <>
:extParam (b)
:allParam (b)
:scanrelid 1
:indexid 24594
:indexqual (
   {OPEXPR
   :opno 96
   :opfuncid 65
   :opresulttype 16
   :opretset false
   :opcollid 0
   :inputcollid 0
   :args (
      {VAR
      :varno 65002
      :varattno 1
```

逻辑查询优化

- 什么是逻辑优化
  - 主要对SQL进行等价或者推倒变换，让其达到更好的执行计划

- 主要优化方向
  - 避免重复工作
  - 减少子集数据量

逻辑查询优化

- SQL组成

```
SELECT s.name
FROM score sc, student s
WHERE s.no IN
(SELECT st_no FROM class c
WHERE c.name = '1101')
AND sc.st_no = s.no
ORDER BY sc.score DESC
LIMIT 1;
```

target list

range table

qualifiter

IN-clause subquery

Join predicate

sort order

limit expression

逻辑查询优化

- 查询重写
  - 消除view、rule等

```
create view v_t_1_2 as SELECT t1.a1, t1.b1, t2.a2, t2.b2 FROM t1, t2;

postgres=> explain select * from v_t_1_2, t1 where v_t_1_2.a1 = 10 and t1.b1 = 20;
QUERY PLAN
------------------------------------------------------------------------------
 Nested Loop  (cost=0.55..41.59 rows=1000 width=24)
   ->  Nested Loop  (cost=0.55..16.60 rows=1 width=16)
         ->  Index Scan using t1_a1_key on t1 t1_1  (cost=0.28..8.29 rows=1 width=8)
               Index Cond: (a1 = 10)
         ->  Index Scan using b1_1 on t1  (cost=0.28..8.29 rows=1 width=8)
               Index Cond: (b1 = 20)
   ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=8)
(7 rows)
```


view又被重写回原来的query

- 提升子链
  - 将IN和exists子句递归提升

```
select * from t1 where t1.a1 in (select t2.a2 from t2 where t2.b2 = 10);
```

假设t2.a2为unique

转化为:

```
select t1.a1,t1,a2 from t1 join t2 where t1.a1=t2.a2 and t2.b2 = 10;
```

执行计划如下:

```
postgres=> explain select * from t1 where t1.a1 in (select t2.a2 from t2 where t2.b2 = 10);
                                QUERY PLAN
---------------------------------------------------------------------------
 Nested Loop  (cost=0.28..25.80 rows=1 width=8)
   ->  Seq Scan on t2  (cost=0.00..17.50 rows=1 width=4)
         Filter: (b2 = 10)
   ->  Index Scan using t1_a1_key on t1  (cost=0.28..8.29 rows=1 width=8)
         Index Cond: (a1 = t2.a2)
```

- 提升子链

```
explain select * from t1 where exists  (select t2.a2 from t2 where t2.a2 = t1.a1) ; 假设t2.a2为unique
```

转化为:

```
select t1.a1, t1.b1 from t1, t2 where t1.a1=t2.a1;
```

执行计划如下:

```
postgres=> explain select * from t1 where exists  (select t2.a2 from t2 where t2.a2 = t1.a1) ;
                            QUERY PLAN
------------------------------------------------------------------
 Hash Join  (cost=26.42..54.69 rows=952 width=8)
   Hash Cond: (t2.a2 = t1.a1)
   ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=4)
   ->  Hash  (cost=14.52..14.52 rows=952 width=8)
         ->  Seq Scan on t1  (cost=0.00..14.52 rows=952 width=8)
(5 rows)
```

- 提升子链
  - 部分IN和exists子句不能提升

```
postgres=> explain select * from t1 where t1.a1 in (select t2.a2 from t2 where t1.b1 = 10);
                                QUERY PLAN
-------------------------------------------------------------------------
 Seq Scan on t1  (cost=0.00..8349.28 rows=476 width=8)
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Result  (cost=0.00..15.00 rows=1000 width=4)
           One-Time Filter: (t1.b1 = 10)
           ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=4)
(6 rows)
```

这里IN子句中使用了主查询的表字段，和主查询有相关性

这里也是可以优化的，把t1.b1移到父查询，但是PG没有支持

- 提升子查询
  - 子查询和子链接区别：子查询是不在表达式中的子句，子链接在表达式中的子句

```
select * from t1, (select * from t2) as c  where  t1.a1 = c.a2;
转化为：
select * from t1, t2 where  t1.a1 = t2.a2;

postgres=> explain select * from t1, (select * from t2) as c  where  t1.a1 = c.a2;
                            QUERY PLAN
------------------------------------------------------------------
 Hash Join  (cost=26.42..54.69 rows=952 width=16)
   Hash Cond: (t2.a2 = t1.a1)
   ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=8)
   ->  Hash  (cost=14.52..14.52 rows=952 width=8)
         ->  Seq Scan on t1  (cost=0.00..14.52 rows=952 width=8)
(5 rows)
```

- 提升子查询
  - 同样子查询也有不支持的情况

```
postgres=> explain select t1.a1 from t1, (select a2 from t2 limit 1) as c where c.a2 = 10;
                                QUERY PLAN
-------------------------------------------------------------------------
 Nested Loop  (cost=0.00..24.07 rows=952 width=4)
   ->  Subquery Scan on c  (cost=0.00..0.03 rows=1 width=0)
         Filter: (c.a2 = 10)
         ->  Limit  (cost=0.00..0.01 rows=1 width=4)
               ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=4)
   ->  Seq Scan on t1  (cost=0.00..14.52 rows=952 width=4)
(6 rows)
```

1. 没有集合操作
2. 没有聚合操作
3. 不包含 sort/limit/with/group
4. 没有易失函数
5. from非空
……

- 化简条件

规则	化简前	化简后
常量传递	a1=a2 and a2=100	a1=100 and a2=100
表达式计算	a1=1+2	a1=3
去除多余括号	(a and b) and (c and d)	a and b and c and d
简化or	false or a > 1	a > 1

- 外连接消除(left/right/full join)

以left join为例，left join(左连接) 返回包括左表中的所有记录和右表中连接字段相等的记录 ，如果右表没有匹配的记录，那么右表将会以NULL值代替，例如：

A表　　B表

ID1　　ID2

1　　1

2

```
select * from A left join B on A.id1 = B.id2;
```

结果如下：

```
ID1  ID2
1       1
2       NULL
```

- 外连接消除(left/right/full join)

```
postgres=> explain select * from t1 left join t2 on true;
                              QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop Left Join  (cost=0.00..11932.02 rows=952000 width=16)
   ->  Seq Scan on t1  (cost=0.00..14.52 rows=952 width=8)
   ->  Materialize  (cost=0.00..20.00 rows=1000 width=8)
         ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=8)
(4 rows)
postgres=> explain select * from t1 left join t2 on true where t1.a1 = t2.a2;
                             QUERY PLAN
--------------------------------------------------------------------
 Hash Join  (cost=26.42..54.69 rows=952 width=16)
   Hash Cond: (t2.a2 = t1.a1)
   ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=8)
   ->  Hash  (cost=14.52..14.52 rows=952 width=8)
         ->  Seq Scan on t1  (cost=0.00..14.52 rows=952 width=8)
(5 rows)
```

- 外连接消除条件
  - where和join条件保证右表不会有NULL值的行产生

```
postgres=> explain select * from t1 left join t2 on t1.b1 = t2.b2 where t2.b2 is not
NULL;
                                QUERY PLAN
-------------------------------------------------------------------------
 Nested Loop  (cost=0.28..23.30 rows=1 width=16)
   ->  Seq Scan on t2  (cost=0.00..15.00 rows=1 width=8)
         Filter: (b2 IS NOT NULL)
   ->  Index Scan using b1_1 on t1  (cost=0.28..8.29 rows=1 width=8)
         Index Cond: (b1 = t2.b2)
(5 rows)
```

- 条件下推
  - 为了连接前，元组数组尽量少

```
postgres=> explain select * from t1,t2 where t1.a1 < 10 and t2.a2 > 900;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop  (cost=0.55..31.20 rows=1000 width=16)
   ->  Index Scan using t2_a2_key on t2  (cost=0.28..10.03 rows=100 width=8)
         Index Cond: (a2 > 900)
   ->  Materialize  (cost=0.28..8.70 rows=10 width=8)
         ->  Index Scan using t1_a1_key on t1  (cost=0.28..8.65 rows=10 width=8)
               Index Cond: (a1 < 10)
```

- 语义优化
  - 约束优化

```
create table tt1(id int not null);
postgres=> explain select * from tt1 where id is null;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on tt1  (cost=0.00..15407.02 rows=1 width=15)
   Filter: (id IS NULL)

set constraint_exclusion = on;

postgres=> explain select * from tt1 where id is null;
                QUERY PLAN
----------------------------------------
 Result  (cost=0.00..0.01 rows=1 width=0)
   One-Time Filter: false
```

默认未优化

• MIN/MAX

select min(a1) from t1;
转换为:
select a1 from t1 order by a1 limit 1;
如果a1没有索引，那么将会是全表扫描

```
postgres=> explain select min(a1) from t1;
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
 Result  (cost=0.32..0.33 rows=1 width=0)
   InitPlan 1 (returns $0)
     ->  Limit  (cost=0.28..0.32 rows=1 width=4)
           ->  Index Only Scan using t1_a1_key on t1  (cost=0.28..45.09 rows=952 width=4)
                 Index Cond: (a1 IS NOT NULL)
```

- group by优化

```
create index tt1_id_key on tt1 using btree ( id);
postgres=> explain select id from tt1 group by id;
                                        QUERY PLAN
-------------------------------------------------------------------------------------
 Group  (cost=0.42..33891.21 rows=1000102 width=4)
   Group Key: id
   ->  Index Only Scan using tt1_id_key on tt1  (cost=0.42..31390.96 rows=1000102 width=4)
(3 rows)
postgres=> explain select name from tt1 group by name;
                                QUERY PLAN
---------------------------------------------------------------------------
 Group  (cost=132169.76..137170.27 rows=1000102 width=11)
   Group Key: name
   ->  Sort  (cost=132169.76..134670.02 rows=1000102 width=11)
         Sort Key: name
         ->  Seq Scan on tt1  (cost=0.00..15407.02 rows=1000102 width=11)
(5 rows)
```

• order by优化

1. 利用索引消除order by

```
postgres=> explain select * from t1 order by a1;
                              QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using t1_a1_key on t1  (cost=0.28..42.71 rows=952 width=8)
(1 row)
```

2. order by下推

```
postgres=> explain select * from t1,t2 where t1.b1=t2.b2 order by b1;
                             QUERY PLAN
--------------------------------------------------------------------
 Merge Join  (cost=126.45..136.22 rows=1 width=16)
   Merge Cond: (t1.b1 = t2.b2)
   ->  Sort  (cost=61.62..64.00 rows=952 width=8)
         Sort Key: t1.b1
         ->  Seq Scan on t1  (cost=0.00..14.52 rows=952 width=8)
   ->  Sort  (cost=64.83..67.33 rows=1000 width=8)
         Sort Key: t2.b2
         ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=8)
(8 rows)
```

distinct优化

```
postgres=> explain select distinct(a1) from t1;
                          QUERY PLAN
---------------------------------------------------------------
 HashAggregate  (cost=16.90..26.42 rows=952 width=4)
   Group Key: a1
   ->  Seq Scan on t1  (cost=0.00..14.52 rows=952 width=4)
(3 rows)
postgres=> explain select distinct(id) from tt1;
                                    QUERY PLAN
-----------------------------------------------------------------------------------
 Unique  (cost=0.42..33891.21 rows=1000102 width=4)
   ->  Index Only Scan using tt1_id_key on tt1  (cost=0.42..31390.96 rows=1000102 width=4)
(2 rows)
postgres=> explain select distinct(name) from tt1;
                               QUERY PLAN
-------------------------------------------------------------------------
 Unique  (cost=132169.76..137170.27 rows=1000102 width=11)
   ->  Sort  (cost=132169.76..134670.02 rows=1000102 width=11)
         Sort Key: name
         ->  Seq Scan on tt1  (cost=0.00..15407.02 rows=1000102 width=11)
(4 rows)
```

hash cost: cheapest_path + hashagg [+ fi
sort]
sort cost: sorted_path [+ sort] + group
final sort]

Hash VS Sort

- 集合操作优化

```
postgres=> explain select a1 from t1 where a1 < 10 union select a2 from t2;
                                        QUERY PLAN
-------------------------------------------------------------------------------------
 HashAggregate  (cost=36.28..46.38 rows=1010 width=4)
   Group Key: t1.a1
   ->  Append  (cost=0.28..33.75 rows=1010 width=4)
         ->  Index Only Scan using t1_a1_key on t1  (cost=0.28..8.65 rows=10 width=4)
               Index Cond: (a1 < 10)
         ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=4)

postgres=> explain select a1 from t1 where a1 < 10 union all select a2 from t2;
                                     QUERY PLAN
-----------------------------------------------------------------------------------
 Append  (cost=0.28..23.75 rows=1010 width=4)
   ->  Index Only Scan using t1_a1_key on t1  (cost=0.28..8.65 rows=10 width=4)
         Index Cond: (a1 < 10)
   ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=4)
```

物理查询优化

- 主要优化方向
  - 单表扫描方式
  - 多表组合方式
  - 多表组合顺序

- 表扫描方式
  - Seq scan, Index scan, Tid scan

```
postgres=> explain select * from t1 ;
                        QUERY PLAN
-----------------------------------------------------
 Seq Scan on t1  (cost=0.00..14.52 rows=952 width=8)

postgres=> explain select * from t1 where a1 = 10;
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using t1_a1_key on t1  (cost=0.28..8.29 rows=1 width=8)
   Index Cond: (a1 = 10)

postgres=> explain select * from t1 where ctid='(1,10)';
                       QUERY PLAN
-------------------------------------------------
 Tid Scan on t1  (cost=0.00..4.01 rows=1 width=8)
   TID Cond: (ctid = '(1,10)'::tid)
```

顺序扫描物理数据页

先通过索引值获得物理数据的位置，再到物理页读取

通过page号和item号直接定位到物理数据

- 选择度计算
  - 无条件

```
EXPLAIN SELECT * FROM tenk1;

                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..458.00 rows=10000 width=244)


 SELECT relpages, reltuples FROM pg_class WHERE relname = 'tenk1';

 relpages | reltuples
----------+-----------
      358 |     10000
```

- 选择度计算
  - 大于或者小于

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 1000;

                                      QUERY PLAN
------------------------------------------------------------------------------------
 Bitmap Heap Scan on tenk1  (cost=24.06..394.64 rows=1007 width=244)
   Recheck Cond: (unique1 < 1000)
   ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..23.80 rows=1007 width=0)
         Index Cond: (unique1 < 1000)
```

- 计算公式

```
SELECT histogram_bounds FROM pg_stats
WHERE tablename='tenk1' AND attname='unique1';

                     histogram_bounds
------------------------------------------------------
 {0,993,1997,3050,4040,5036,5957,7057,8029,9016,9995}
```

selectivity = (1 + (1000 - bucket[2].min)/(bucket[2].max - bucket[2].min))/num_buckets
= (1 + (1000 - 993)/(1997 - 993))/10
= 0.100697

rows = rel_cardinality * selectivity
= 10000 * 0.100697
= 1007 (rounding off)

- 选择度计算
  - 等于计算

```
EXPLAIN SELECT * FROM tenk1 WHERE stringu1 = 'CRAAAA';

                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..483.00 rows=30 width=244)
   Filter: (stringu1 = 'CRAAAA'::name)
```

计算公式

```
SELECT null_frac, n_distinct, most_common_vals, most_common_freqs FROM pg_stats
WHERE tablename='tenk1' AND attname='stringu1';

null_frac         | 0
n_distinct        | 676
most_common_vals|{EJAAAA,BBAAAA,CRAAAA,FCAAAA,FEAAAA,GSAAAA,JOAAAA,MCAAAA,NAAAAA,WGAAAA}
most_common_freqs |  {0.00333333,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003}

selectivity = mcf[3]
            = 0.003

rows = 10000 * 0.003
     = 30
```

备注：如果值不在most_common_vals里面，计算公式为：

```
selectivity = (1 - sum(mvf))/(num_distinct - num_mcv)
```

pg_stats统计信息很重要！！！

- cost计算
  - 代价模型:总代价=CPU代价+IO代价+启动代价

```
postgres=> explain select * from t1 where a1 > 10;
                        QUERY PLAN
-----------------------------------------------------
 Seq Scan on t1  (cost=0.00..16.90 rows=942 width=8)
   Filter: (a1 > 10)
(2 rows)
```

其中：

```
postgres=> select relpages, reltuples from pg_class where relname = 't1';
 relpages | reltuples
----------+-----------
        5 |       952
(1 row)
cpu_operator_cost=0.0025
cpu_tuple_cost=0.01
seq_page_cost=1
random_page_cost=4
```

Cost=16.90怎么计算出来的?

cost计算

总cost = cpu_tuple_cost * 952 + seq_page_cost * 5 + cpu_operator_cost * 952
= 16.90

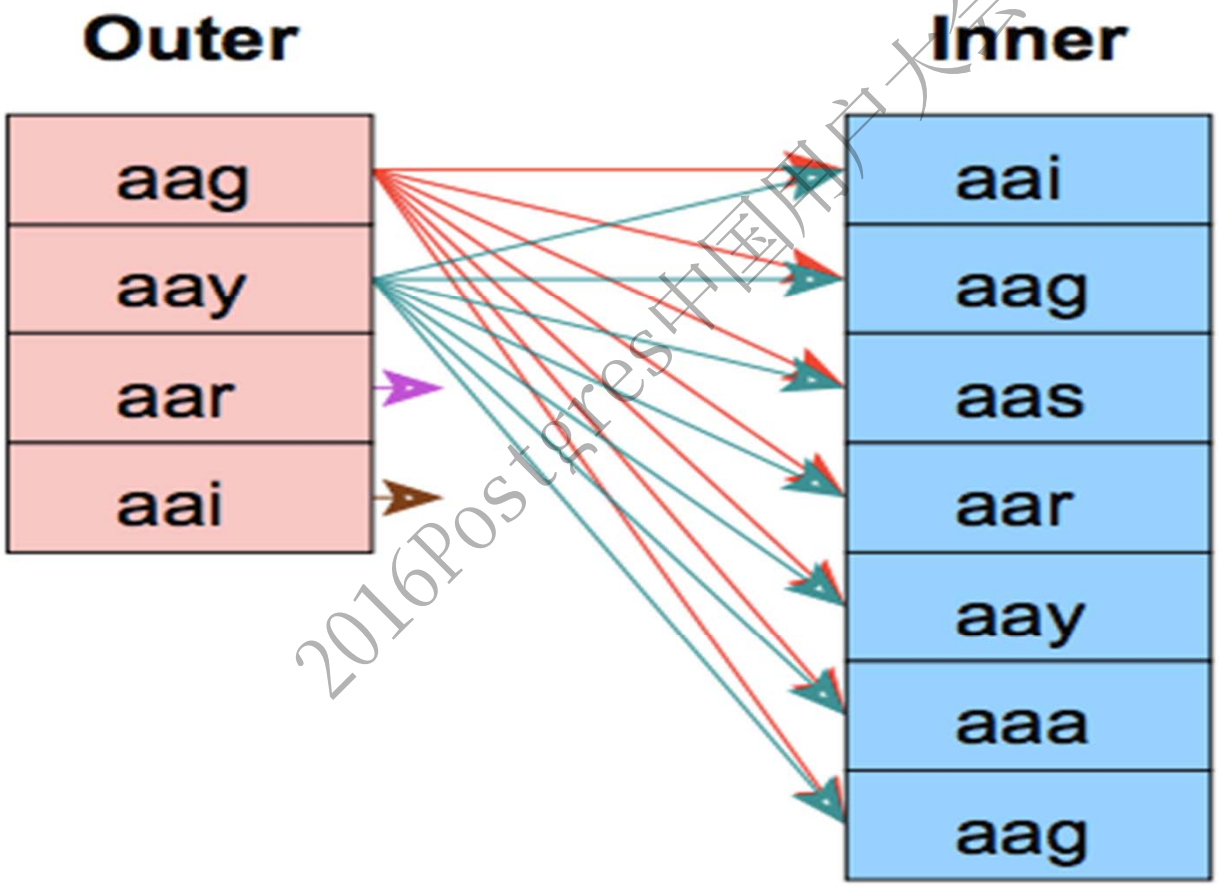
其他扫描方式cost计算可以参考如下函数：

```
postgres=> select amcostestimate,amname from pg_am ;
  amcostestimate   | amname
-------------------+--------
 btcostestimate    | btree
 hashcostestimate  | hash
 gistcostestimate  | gist
 gincostestimate   | gin
 spgcostestimate   | spgist
(5 rows)
```

表组合方式

- Nest Loop
- Hash Join
- Merge Join

(Index)Nest Loop



简单连接代价估算

SELECT * FROM t1 L, t2 R WHERE L.id=R.id

假设：

M = 20000 pages in L, p_L = 40 rows per page,
N = 400 pages in R, p_R = 20 rows per page.

select relpages, reltuples from pg_class where relname= ‘t1’

Nest Loop IO代价计算

L和R进行join

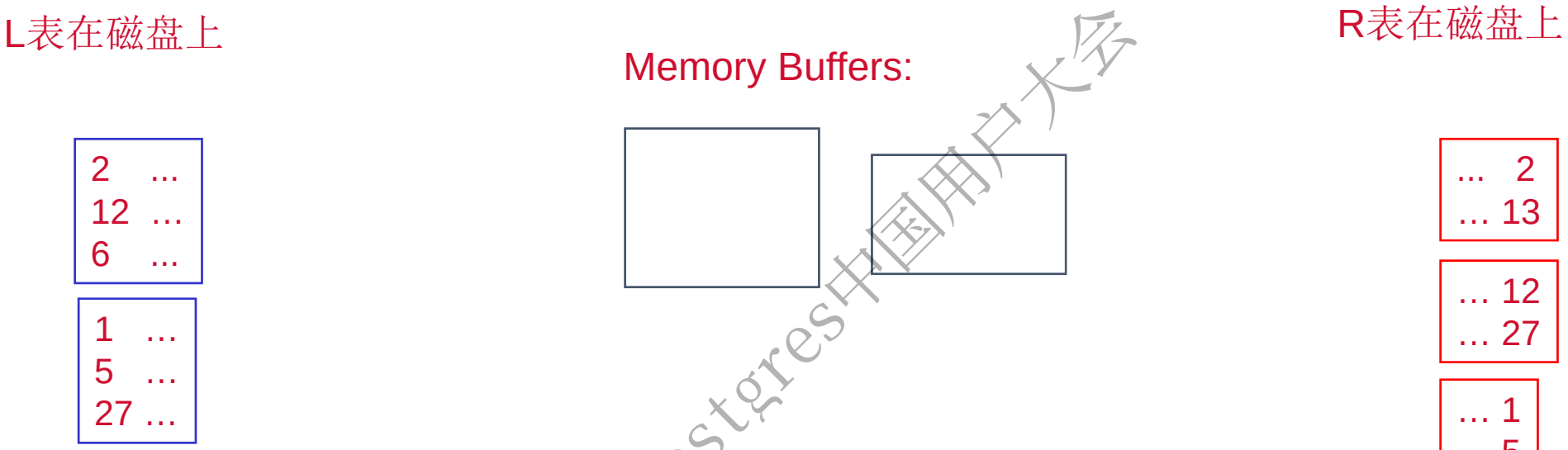
```
for l in L do
  for r in R do
    if r_id == l_id  then ret += (r, s)
```

对于外表L每一个元组扫描内表R所有的元组

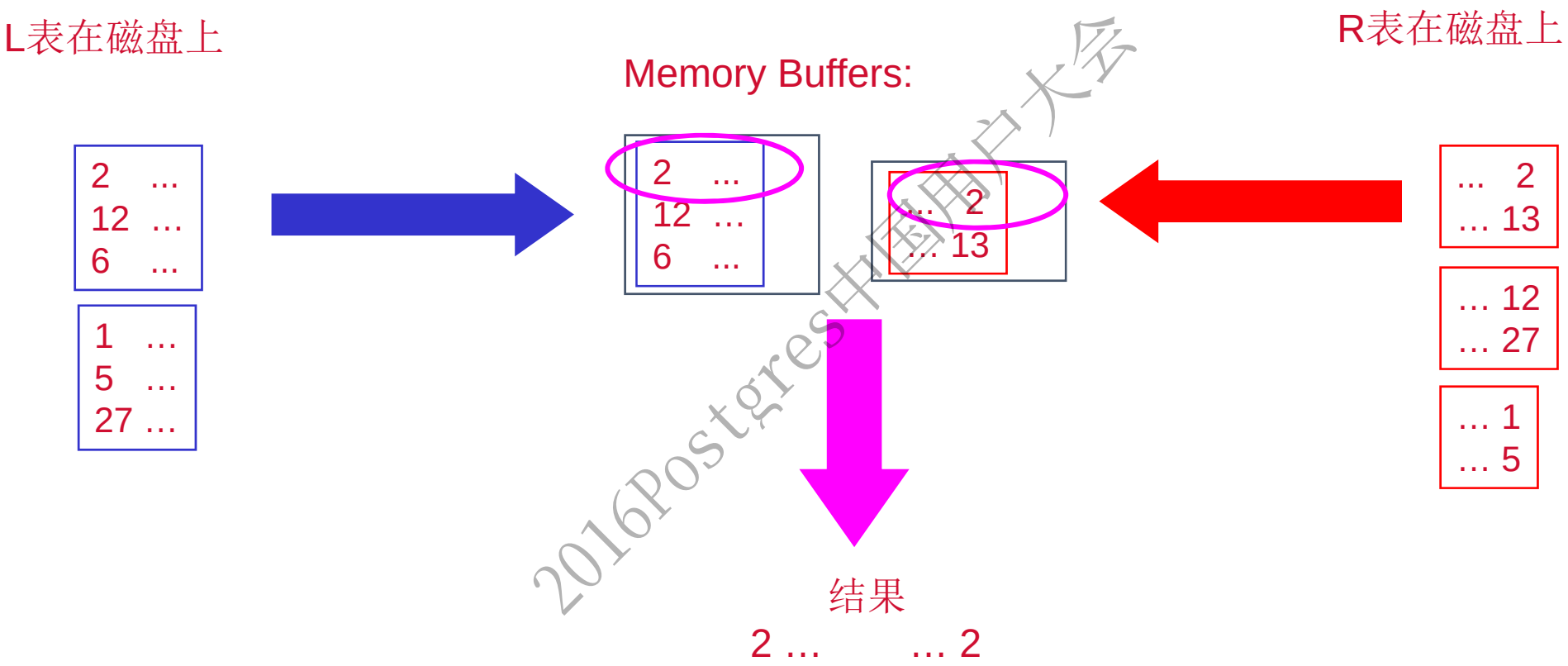
总IO代价： $M + (p_L * M) * N = 20000 + (40*20000)*400$

$= 320020000$

Nest Loop Join

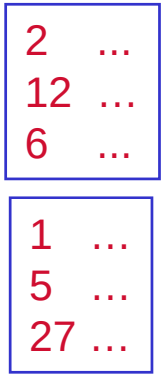


Nest Loop Join

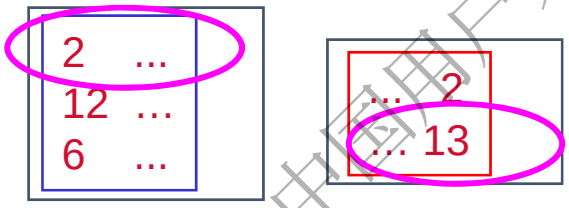


Nest Loop Join

L表在磁盘上



Memory Buffers:



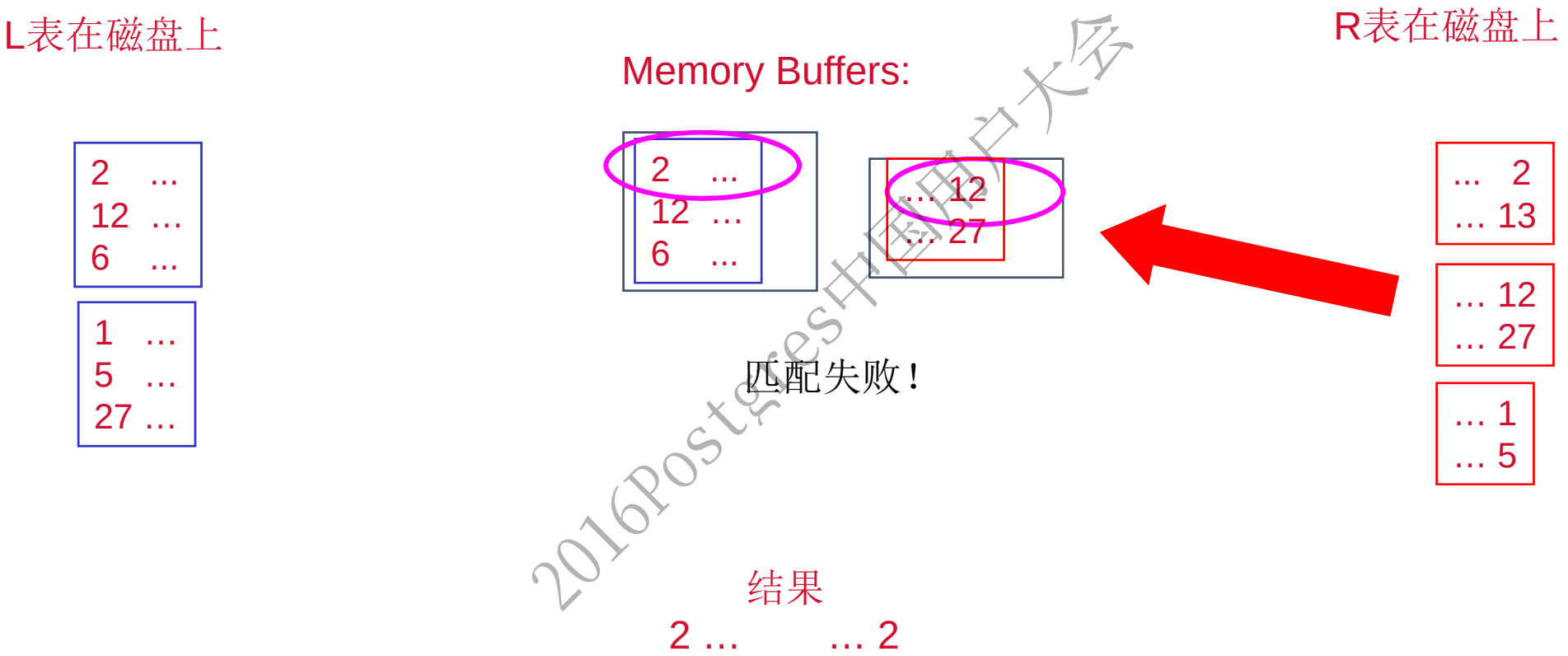
匹配失败！

R表在磁盘上

结果

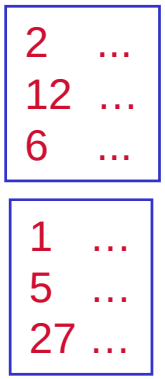
2 2

Nest Loop Join

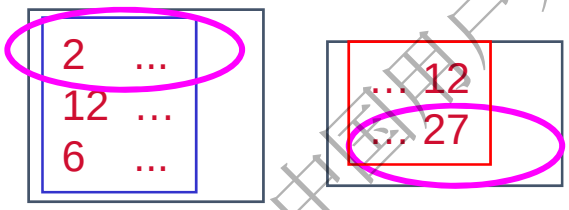


Nest Loop Join

L表在磁盘上



Memory Buffers:



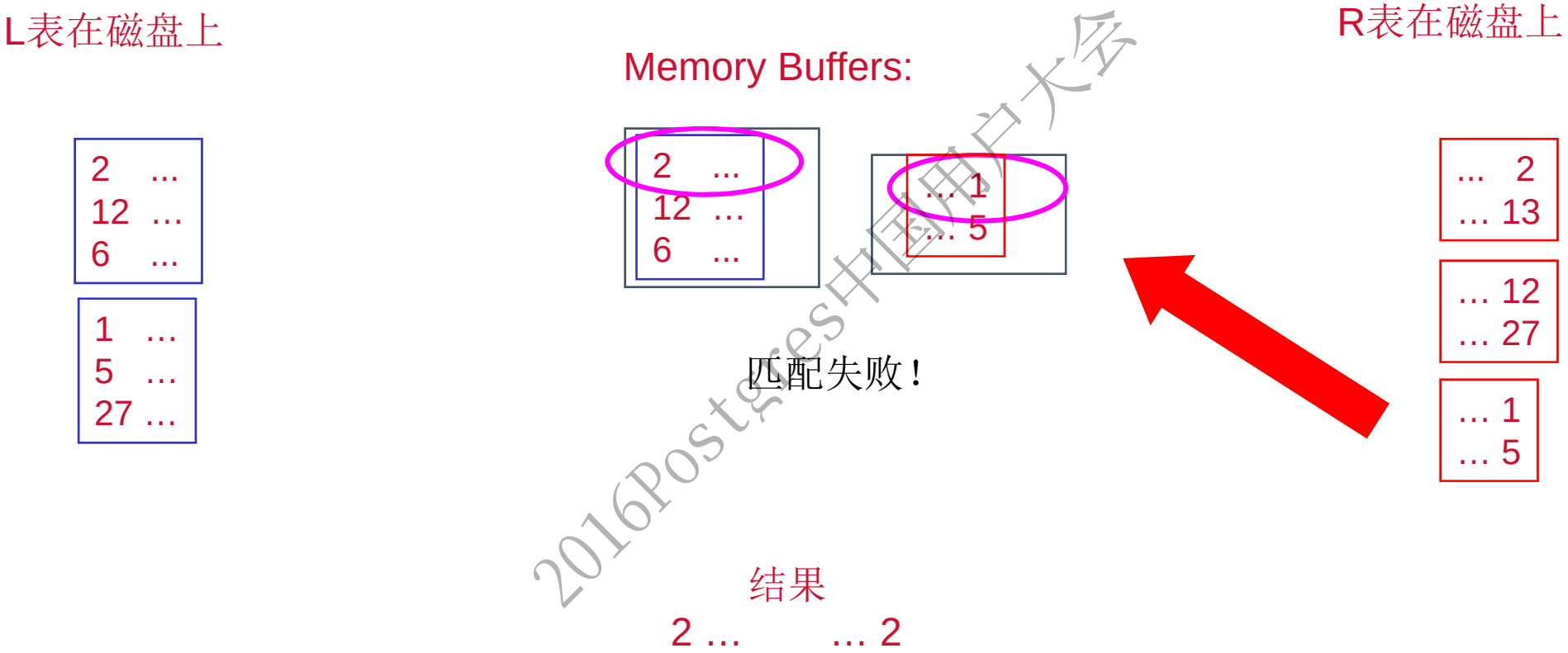
匹配失败！

R表在磁盘上

结果

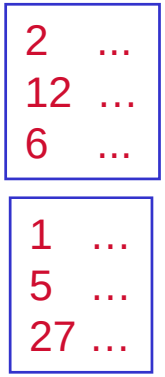
2 2

Nest Loop Join

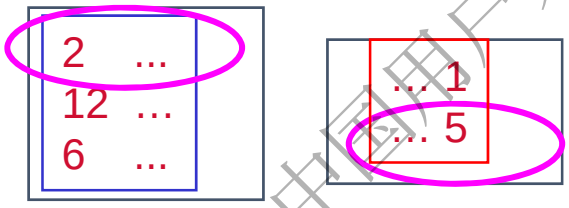


Nest Loop Join

L表在磁盘上



Memory Buffers:



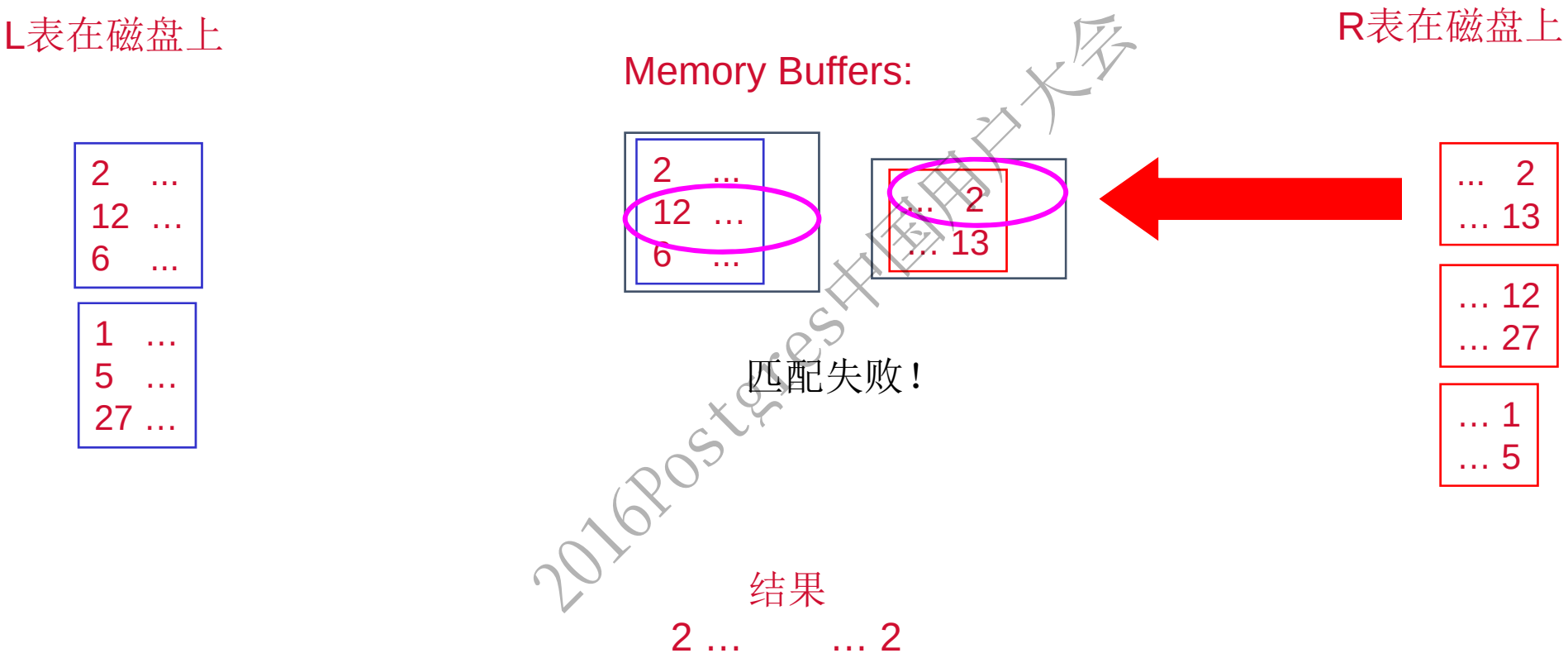
匹配失败！

R表在磁盘上

结果

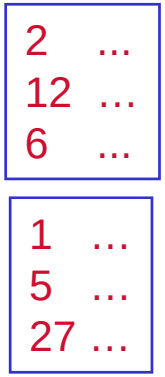
2 2

Nest Loop Join

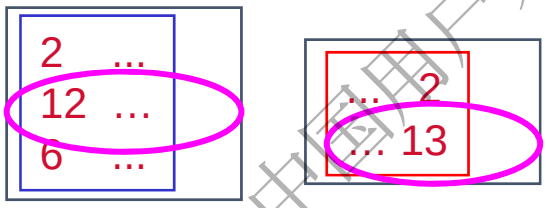


Nest Loop Join

L表在磁盘上



Memory Buffers:



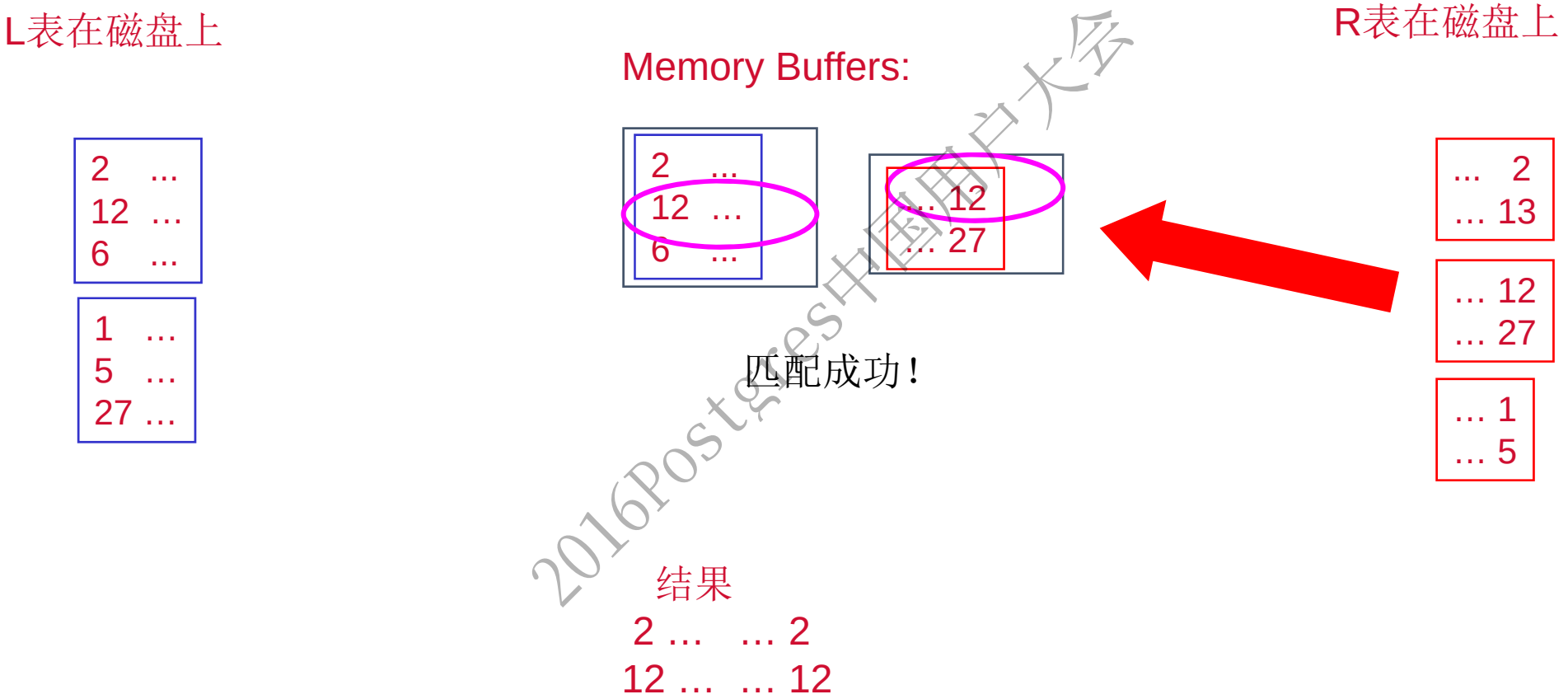
匹配失败！

R表在磁盘上

结果

2 2

Nest Loop Join



Index Nest Loop IO代价

SELECT * FROM t1 L, t2 R WHERE L.id=R.id

假设在表R.id上创建了索引

for l in L do

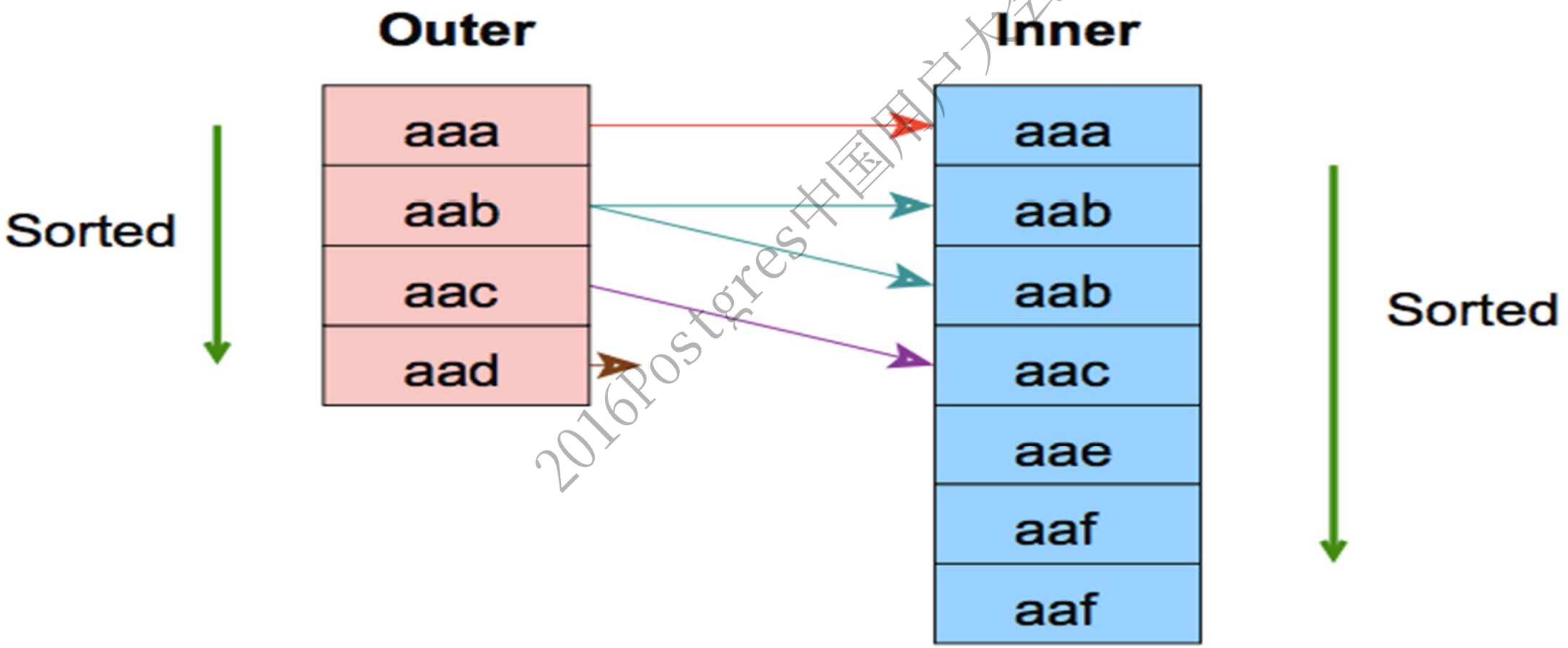
Index Scan r in R where $l_{id} = r_{id}$

ret += (l, r)

总IO: M + ((M*p_L) * cost of finding matching R rows)

= 20000 + ((20000*40)*3) =2420000

Merge Join

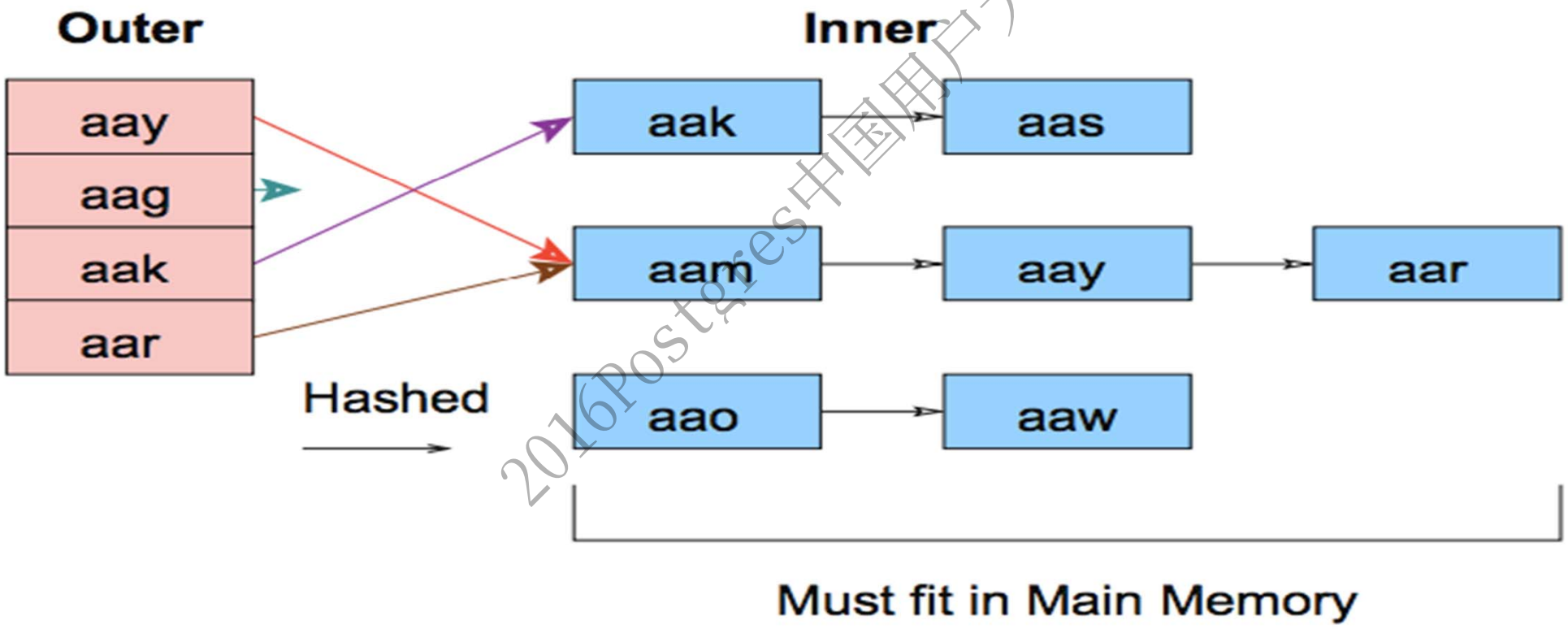


Merge Join

主要分为3步:

1. Sort L on l_{id} 代价MlogM
2. Sort R on r_{id} 代价NlogN
3. Merge the sorted L and R on l_{id} and r_{id} 代价M+N

Hash Join



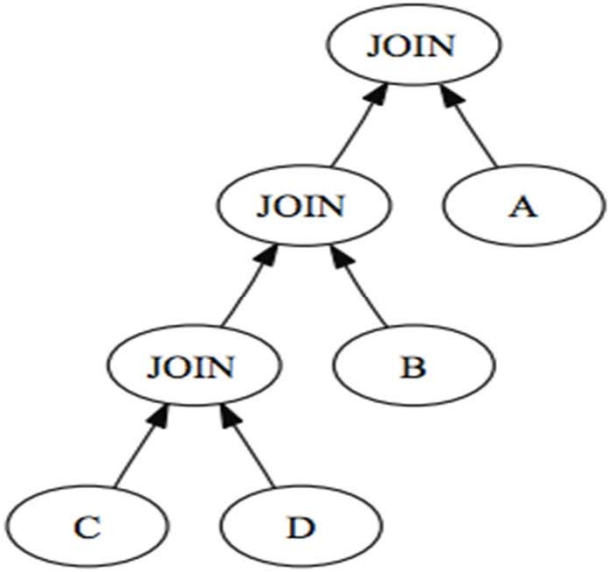
多表连接算法

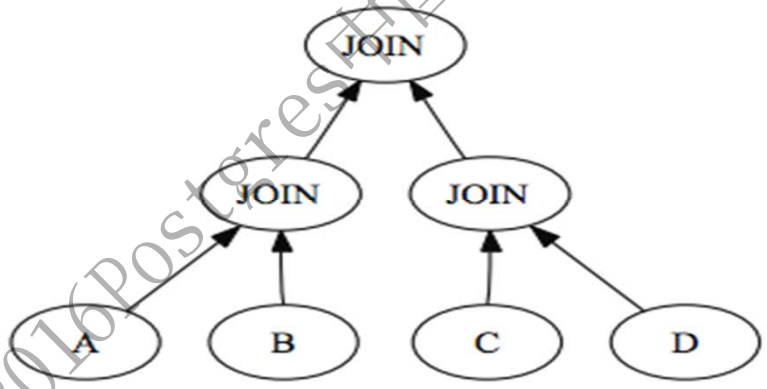
- 动态规划

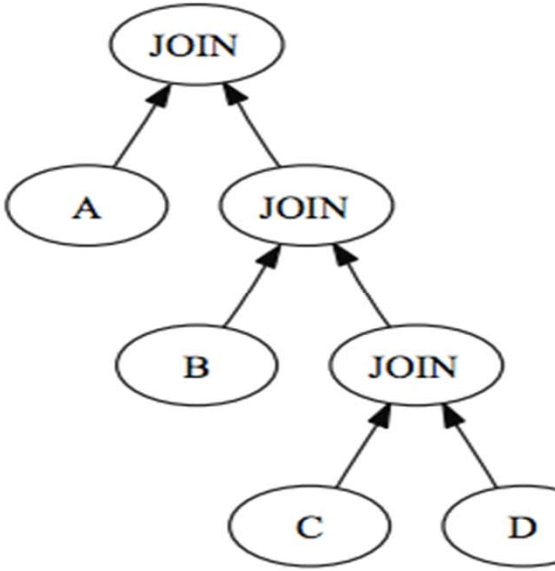
- 遗传算法：适用于join特别多情况

多表连接算法

- 连接树





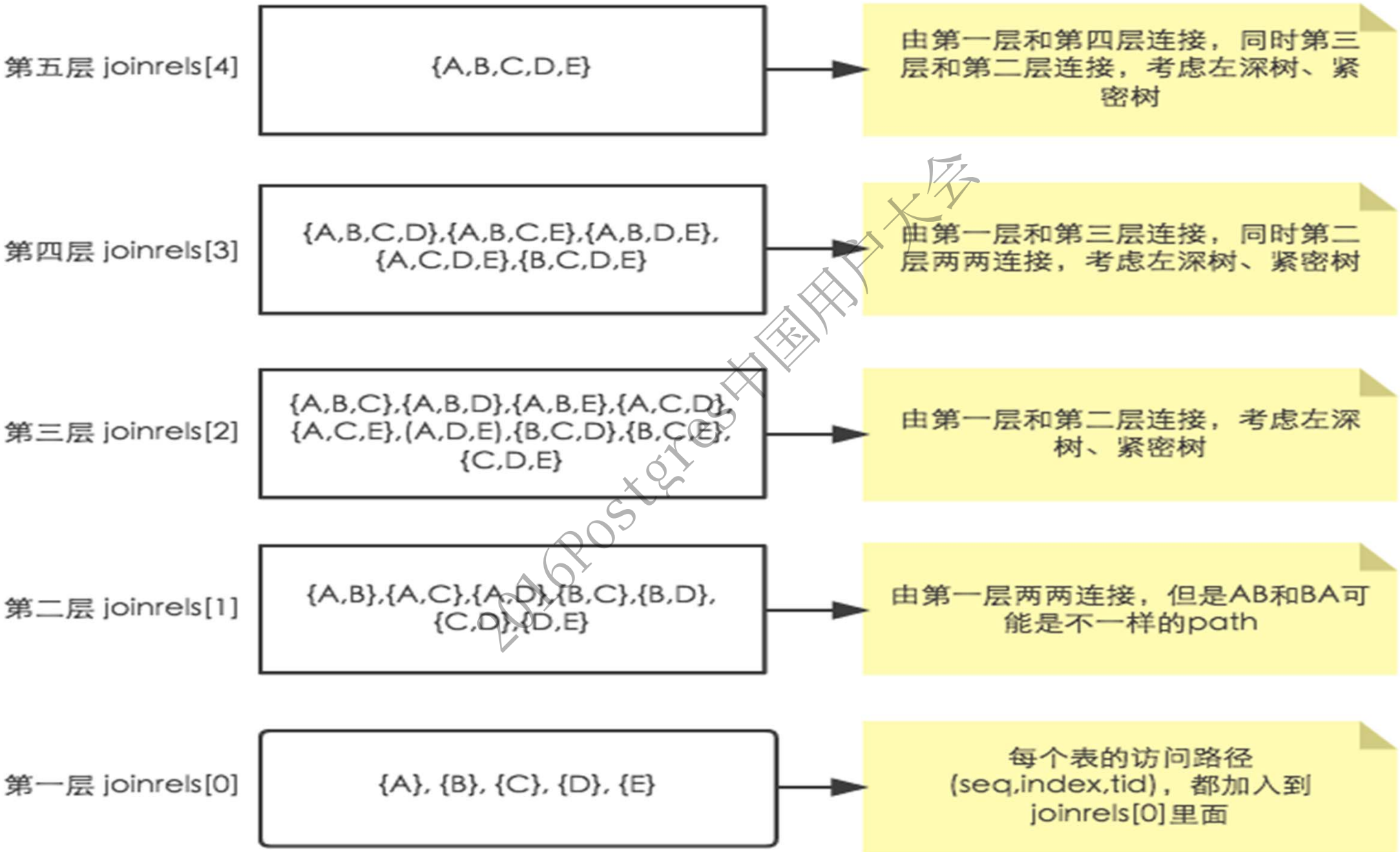


左深树

紧密型树

右深树

动态规划求解过程



动态规划

- 每个中间结果都保留三个路径
  - 1. 最小启动代价
  - 2. 代价最小路径
  - 3. 代价最小排序路径

Thanks!

Q & A