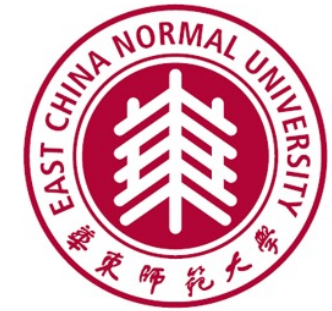
查询-Join算子实现



胡卉芪
华东师范大学
数据科学与工程学院
hqhu@dase.ecnu.edu.cn

Revisit火山模型

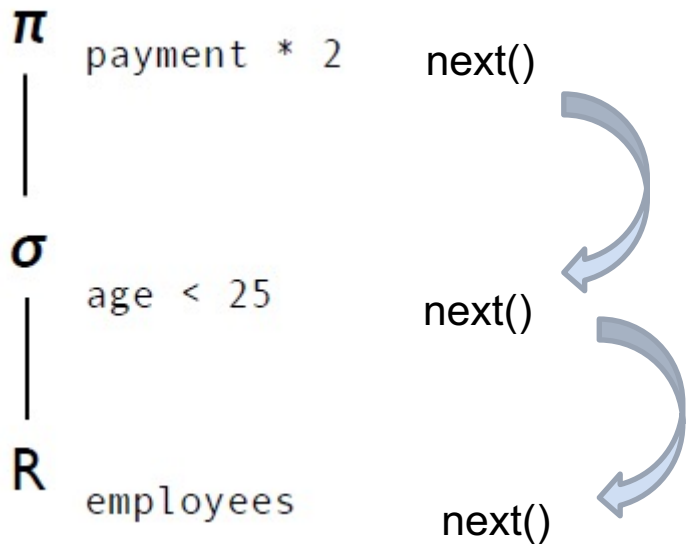
- open
  - 准备资源，准备获得第一个tuple
- next
  - 一次提供一个数据
- close
  - 释放资源
- 名词补充
  - SeqScan、Index Scan

```
Open (){
    b: =R 的第一块;
    t: =b 的第一个元组;
}

GetNext(){
    IF(t 已超过块 b 的最后一个元组){
        将 b 前进到下一块
        IF(没有下一块)
            RETURN NotFound;
            ELSE/* b 是一个新块* /
            t: =块 b 上的第一个元组;
    }/* 现在我们已准备好返回 t 并前进* /
    oldt: =t;
    将 t 前进到 b 的下一元组;
    RETURN oldt;
}

Close(){
}
```

SeqScan算子的接口设计

Revisit火山模型/流水线

- 物理计划是一颗树形结构
- 操作流从上往下，数据从下往上



Revisit数据库中连接操作

- 连接也称为θ连接(Join)
- 连接运算的含义

从两个关系的笛卡尔积中选取属性间满足一定条件的元组

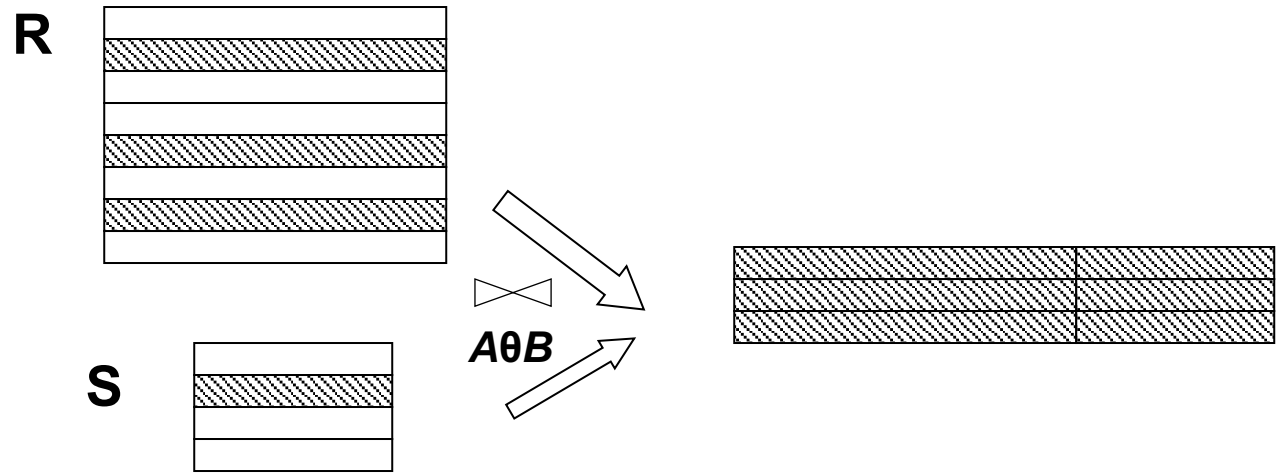
$$R \underset{A\theta B}{\bowtie} S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A] \theta t_s[B] \}$$

  - A和B：分别为R和S上度数相等且可比的属性组
  - θ：比较运算符

- 连接运算从R和S的广义笛卡尔积$R \times S$中选取R关系在A属性组上的值与S关系在B属性组上的值满足比较关系θ的元组
  - 等值连接：θ为“=”的连接运算称为等值连接
  - 自然连接、外连接、非等值连接

等值连接

- 一般的连接操作是从行的角度进行运算。



等值连接

- 关系R和关系S如下所示：

R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

S

B	E
b1	3
b2	7
b3	10
b3	2
b2	2

等值连接

等值连接 $R \underset{R.B=S.B}{\bowtie} S$ 的结果如下：

A	R.B	C	S.B	E
a1	b1	5	b1	3
a1	b2	6	b2	7
a1	b2	6	b2	2
a2	b3	8	b3	10
a2	b3	8	b3	2

连接算法

Nest Loop Join、Hash Join、Merge Join

Nest Loop Join

- R ⋈ S
- 对R和S进行双循环匹配(下图为S ⋈ R)

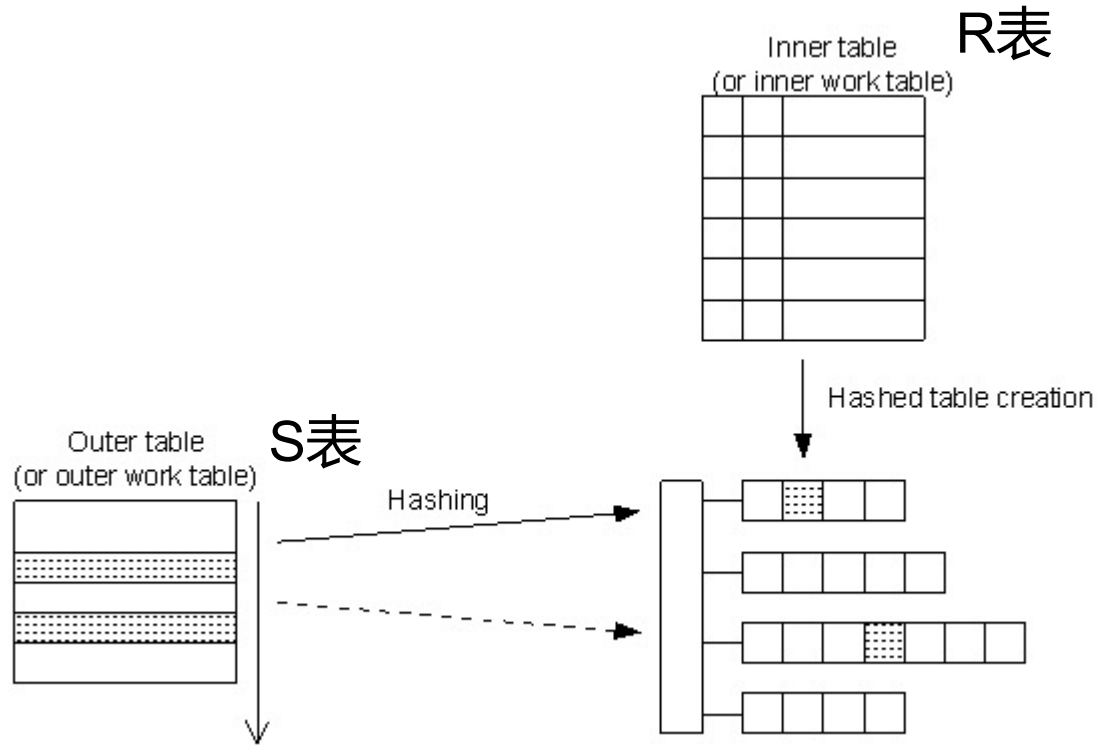
Nested loops join

Function: `nljoin` (R, S, p)

```
/* outer relation R                                          */
foreach record r ∈ R do
    /* inner relation S                                      */
    foreach record s ∈ S do
        /* ⟨r, s⟩ denotes record concatenation               */
        if ⟨r, s⟩ satisfies p then
            append ⟨r, s⟩ to result
```

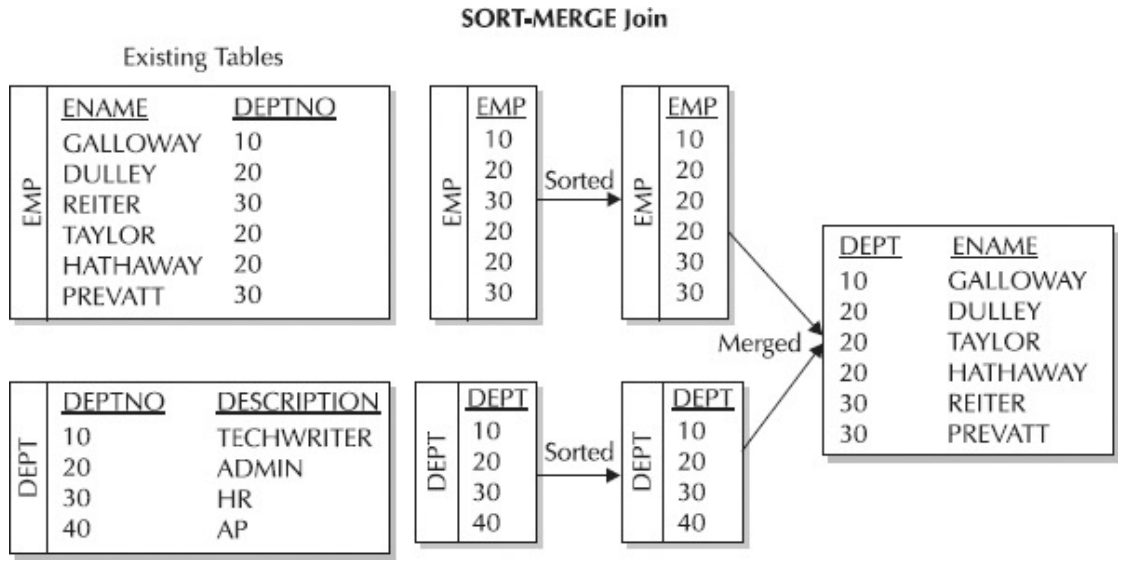
Hash Join

- 对R表构建哈希表，右表进行哈希探测



Merge Join

- 分别对R和S进行排序，然后用归并算法得到Join结果



连接算法结合火山模型

Nest Loop的火山模型

Volcano-Style Nested Loops Join ($\bowtie_p$)

A Volcano-style implementation of nested loops join $R \bowtie_p S$?

```
1 Function: open ()
2 R.open () ;
3 S.open () ;
4 r ← R.next () ;
```

```
1 Function: close ()
2 R.close () ;
3 S.close () ;
```

```
1 Function: next ()
2 while (r ≠ ⟨EOF⟩) do
3     while ((s ← S.next ()) ≠ ⟨EOF⟩) do
4         if p(r, s) then
              /* emit concatenated result      */
5             return ⟨r, s⟩ ;
      /* reset inner join input                */
6     S.close () ;
7     S.open () ;
8     r ← R.next () ;
9 return ⟨EOF⟩;
```

Hash Join的火山模型

- 写法1

```
Open(){
        R.open();
        while((r=R.next())!=EOF)
                将r加入哈希表h
        S.open();
}
```

```
Next(){
        while(){
         s=S.next();
         用s探索哈希表h;
          If(找到一个匹配<r,s>)
             return <r,s>
         }
 }
```

Hash Join的火山模型

- 写法2

```
Open(){
        R.open();
        r=R.next();
        S.open();
}
```

```
Next(){
        if(r是R的第一个元祖){
                将r加入哈希表h
                while((r=R.next())!=EOF)
                        将r加入哈希表h
        }
        while(){
        s=S.next();
        用s探索哈希表h;
         If(找到一个匹配<r,s>)
            return <r,s>
        }
}
```

思考，基于Merge Join的写法？

基于块数据的连接算法

Block Nest Loop与Block Hash Join

基于块的连接算法

- 数据库以Block/page为基本存储单位
- 缓冲区可能存在内存不足，无法将数据全部加在到内存进行计算
- 假设内存有B个Block用于Join，其中一个Block用于缓存Join结果(注意火山模型中我们不需要缓存所有结果)

Nest Loop Join

- Recall Nest Loop Join

Nested loops join

Function: `nljoin` (R, S, p)

```
/* outer relation R                                          */
foreach record r ∈ R do
    /* inner relation S                                      */
    foreach record s ∈ S do
        /* ⟨r,s⟩ denotes record concatenation                */
        if ⟨r,s⟩ satisfies p then
            append ⟨r,s⟩ to result
```

- 假定R和S的blocks数量分别为N_R和N_S, 那么块访问次数一种为 $N_R + |R| * N_S$

扩展Indexed Nest Loop Join

- 对R表的每个数据，直接对S表做Index Scan
- 减少磁盘访问次数和访问数据量

Index nested loops join

Function: `index_nljoin` (R, S, p)

foreach record $r \in R$ **do**

    scan S-index using (key value in) r and concatenate r with all matching tuples s ;
    append $\langle r, s \rangle$ to result ;

Block Nest Loop Join

- 为减少磁盘访问, 假设缓冲区中b_r和b_s个块用于缓存R和S的数据，$b_r + b_s$=B-1

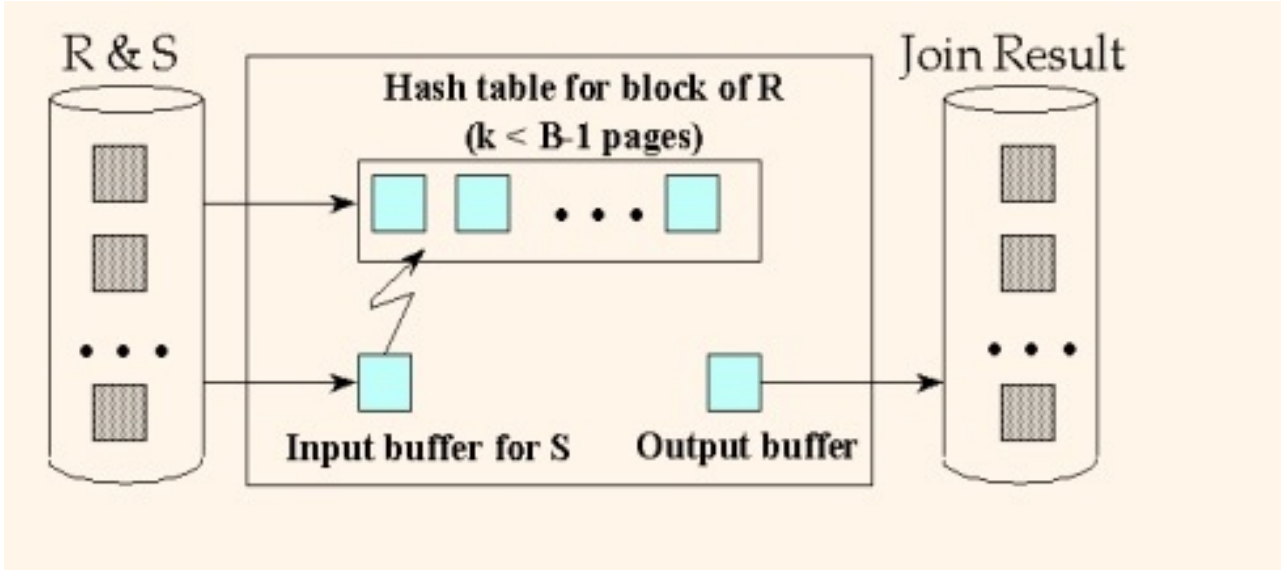
Block nested loops join: build hash table from outer row block

```
1 Function: block_nljoin'(R, S, p)
2 foreach b_R-sized block in R do
3     build an in-memory hash table H for the current R-block ;
4     foreach b_S-sized block in S do
5         foreach record s in current S-block do
6             probe H and append matching ⟨r, s⟩ tuples to result ;
```

- 磁盘访问次数 $\lceil N_R/b_R \rceil \cdot \lceil N_S/b_S \rceil$
- 访问block数量 $N_R * N_S$

Block Nest Loop Join

- 通常b_r=B-2, b_s=1,内存中建哈希表优化匹配
  - *如果R表能一次性全部放在内存，事实上和我们经常说的哈希连接是会比较接近



Block Join与火山模型关系

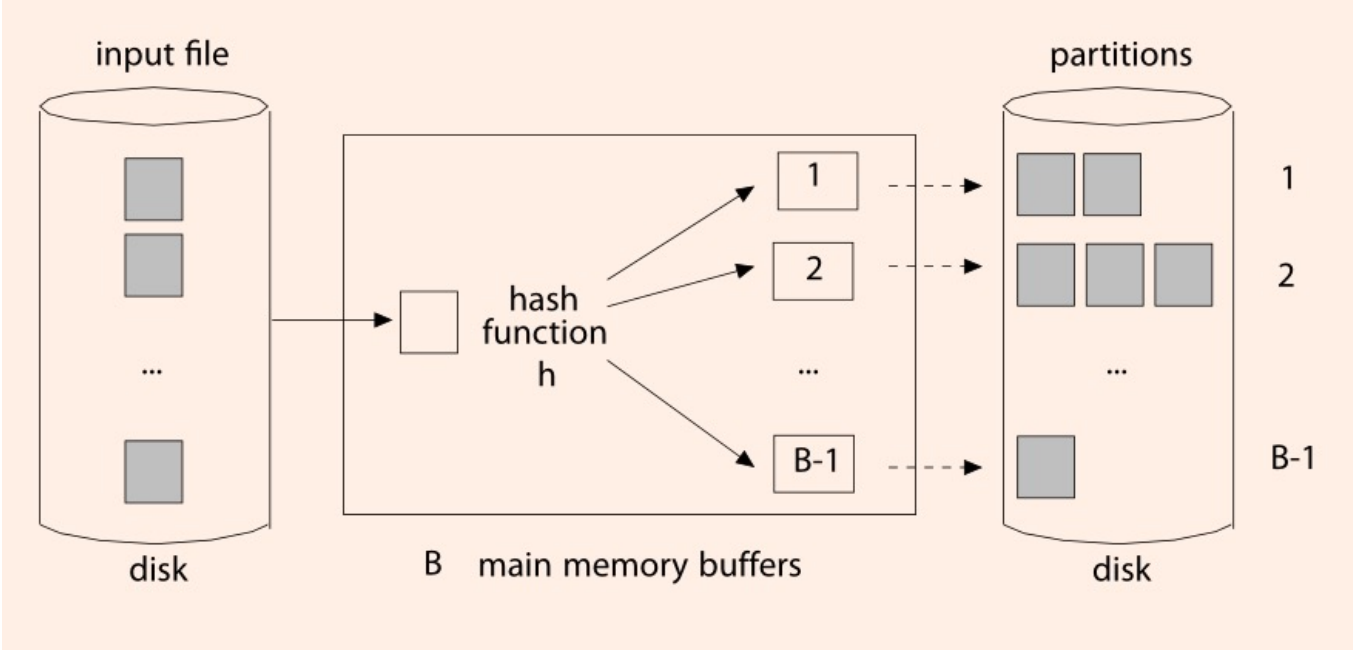
- 两者没有矛盾，火山模型只是实现上的接口形式
- 参考P13, 我们可以对Join的next函数实现P24的算法

Block Hash Join

- 如果R表可以完全放入内存
  - R和S只访问一趟, 一趟算法
  - 和P25页Nest Loop 算法接近
- 如果R表无法完全放入内存
  - 课本上的Block Hash Join
    - 数据库系统实现(Hector)

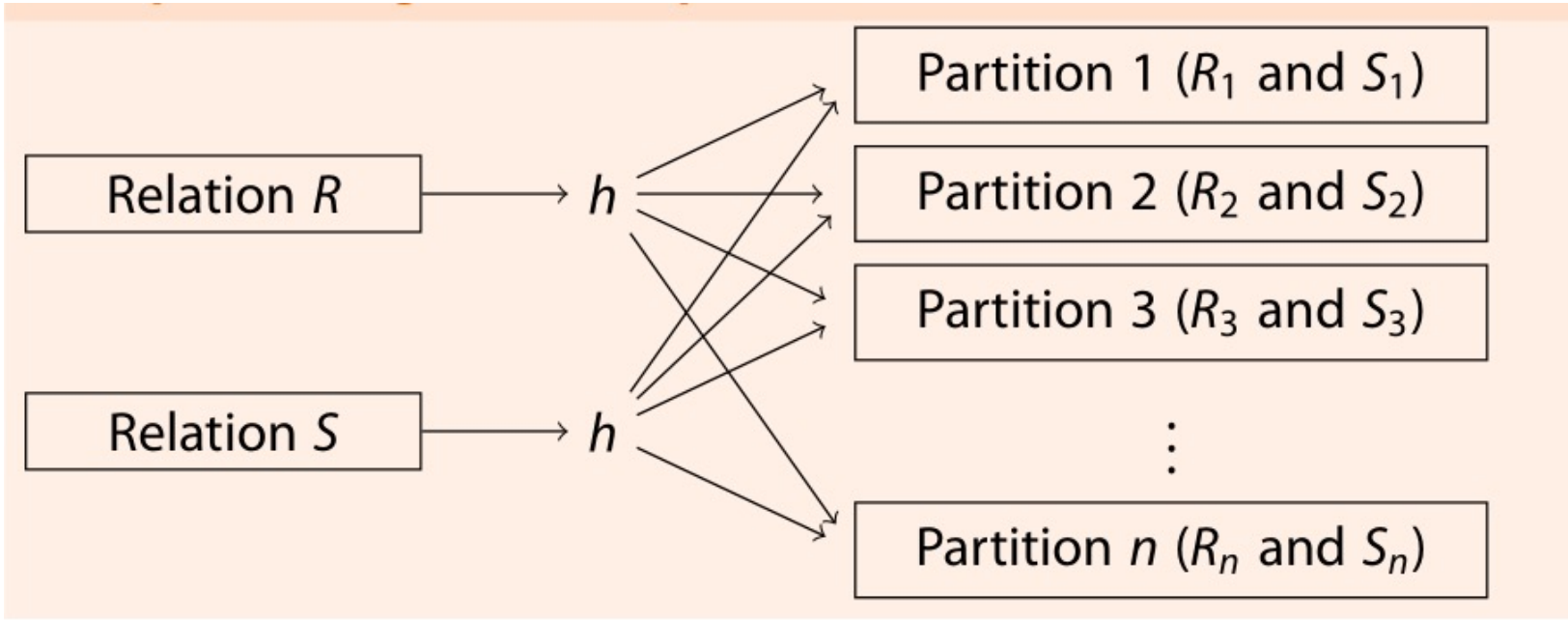
R表的Block Hash算法

- 建B-1个桶，每个桶写满一个Block就刷盘，一个桶可能占多个Block



Block Hash Join

- 对S表同样可以做Block Hash算法
- 对每个Partition做一趟哈希连接



Block Hash Join

- 又称Grace Join

Hash join

Function: `hash_join` $(R, S, \alpha = \beta)$

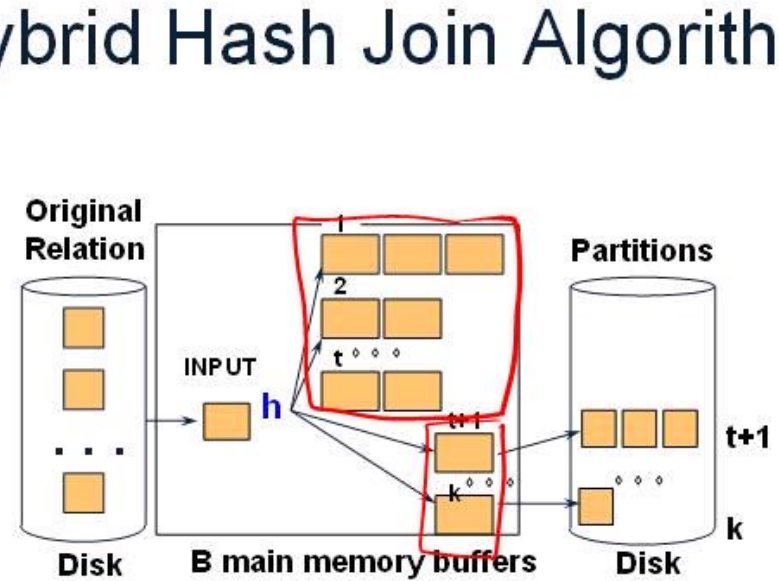
```
/* Partitioning phase                                        */
foreach record r ∈ R do
    append r to partition R_{h(r.α)}
foreach record s ∈ S do
    append s to partition S_{h(s.β)}
/* Intra-partition join phase                                */
foreach partition i ∈ 1, ..., n do
    build hash table H for R_i, using hash function h';
    foreach block b ∈ S_i do
        foreach record s ∈ b do
            probe H via h'(s.β) and append matching tuples to
            result ;
```

Block Hash Join的限制

- 如果每个Partition中R表的Block数量都少于B-1
  - 则算法供需对R,S表访问两趟,两趟算法
- 否则
  - 需要Partition进一步哈希，多趟算法
- 总体上，两趟算法的限制
  - R表数据量少于(B-1)*(B-1)
- Grace Join访问磁盘block数量
  - $3N_R + 3N_S$

Hybrid Hash Join

- 部分Partition做一趟算法，部分用二趟算法
  - 假设构建k个桶，对其中m个桶完全保留在内存中，其他k-m各桶只保留一个block



优化代价估计

- 桶的数量限制
  - $k - m + \frac{m * N_R}{k} \leq \mathrm{B} - 1$
- 对于内存中每一块，节省2次I/O。因为内存中桶比例为m/k，所以节省 2m/k($N_R + N_S$)
- 选m=1, 那么k应该尽可能小. k至少为 N_R/B,实际K还要更大，估算优化为 2B/N_R*($N_R + N_S$)

思考，基于块的排序算法？