



海量数据计算研究中心

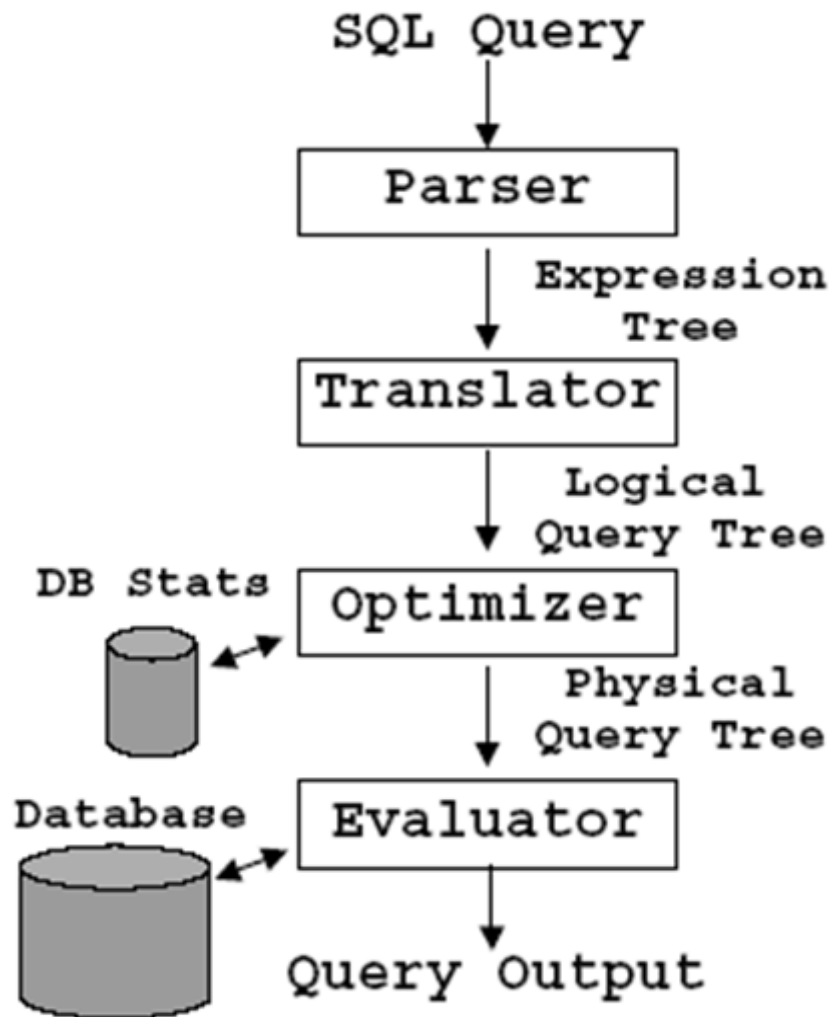
实现篇

第八章查询处理

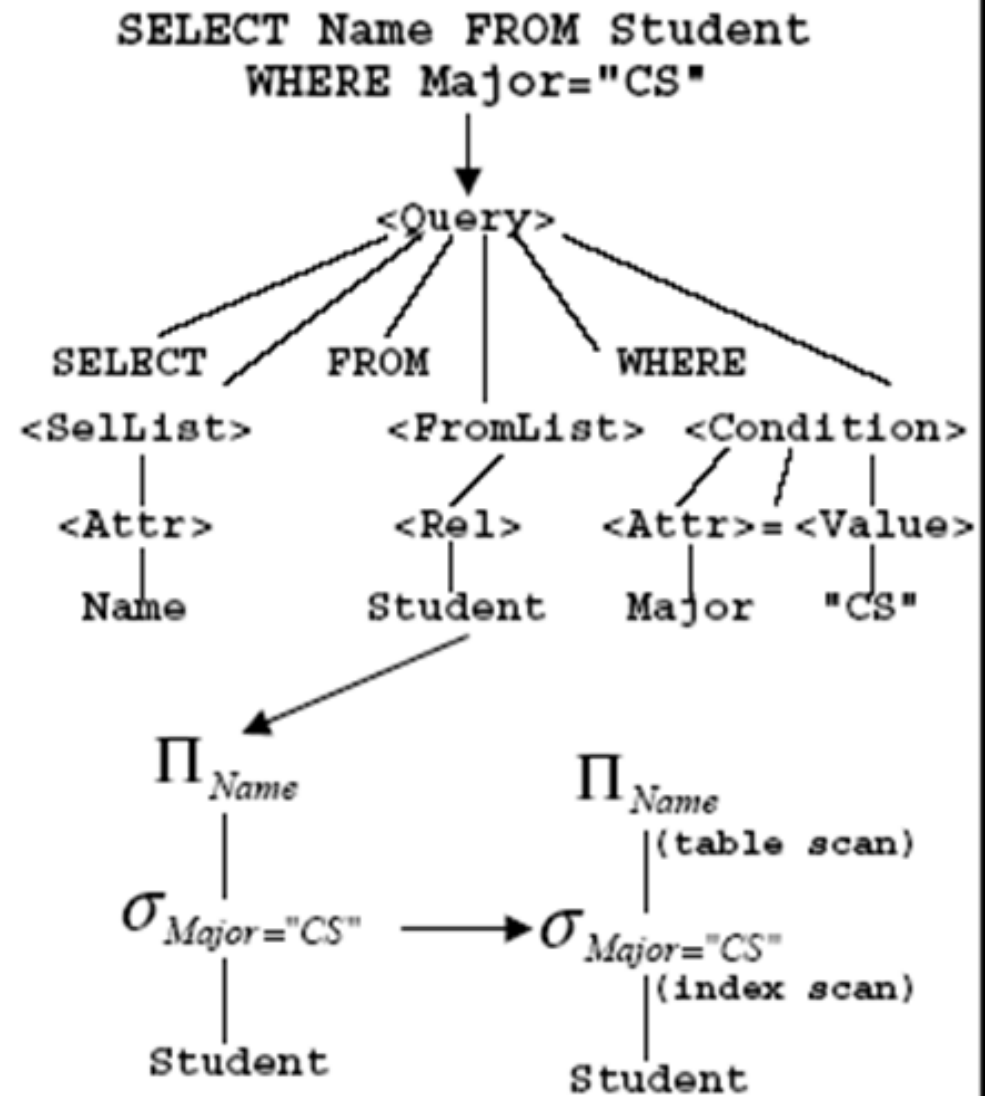
主讲：程思瑶

海量数据计算研究中心





2020-2-24



HIT-DBLAB



几个概念

- Parser Tree (Expression Tree)
 - 由select、from、where组成的语法树
- Logical Query Plan Tree
 - 由基本关系操作符组成的查询树
 - 如：选择、投影、连接等
- Physical Query Plan Tree
 - 由物理操作符组成的查询树
 - 物理操作符
 - 顺序扫描、索引扫描等
 - Hash-join、sort-merge-join等





Example: SQL query

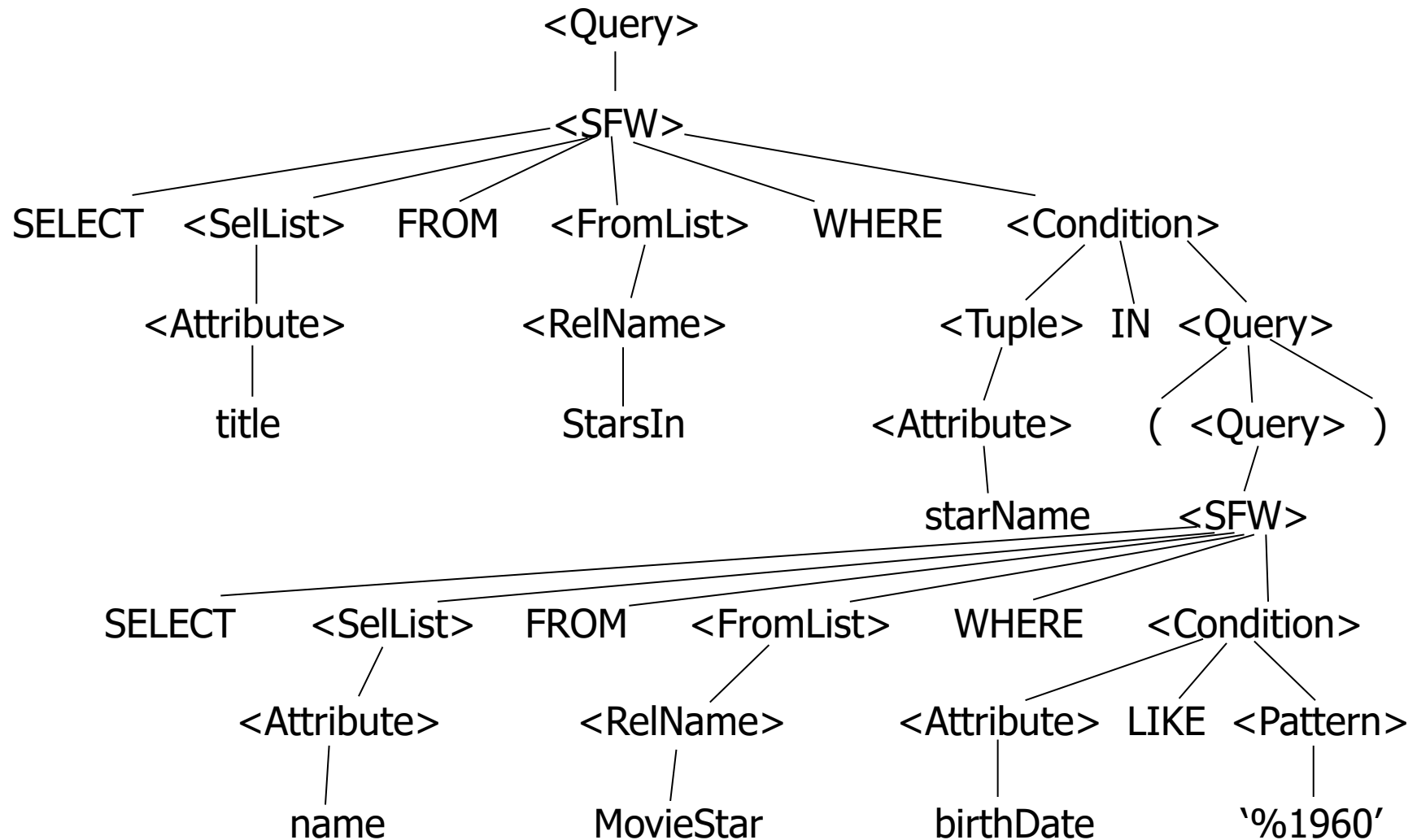
```
SELECT title  
FROM StarsIn  
WHERE starName IN (  
    SELECT name  
    FROM MovieStar  
    WHERE birthdate LIKE '%1960' );
```

(找到影星生于 1960 年的电影名字)





Example: Parser Tree





Example: 生成关系代数

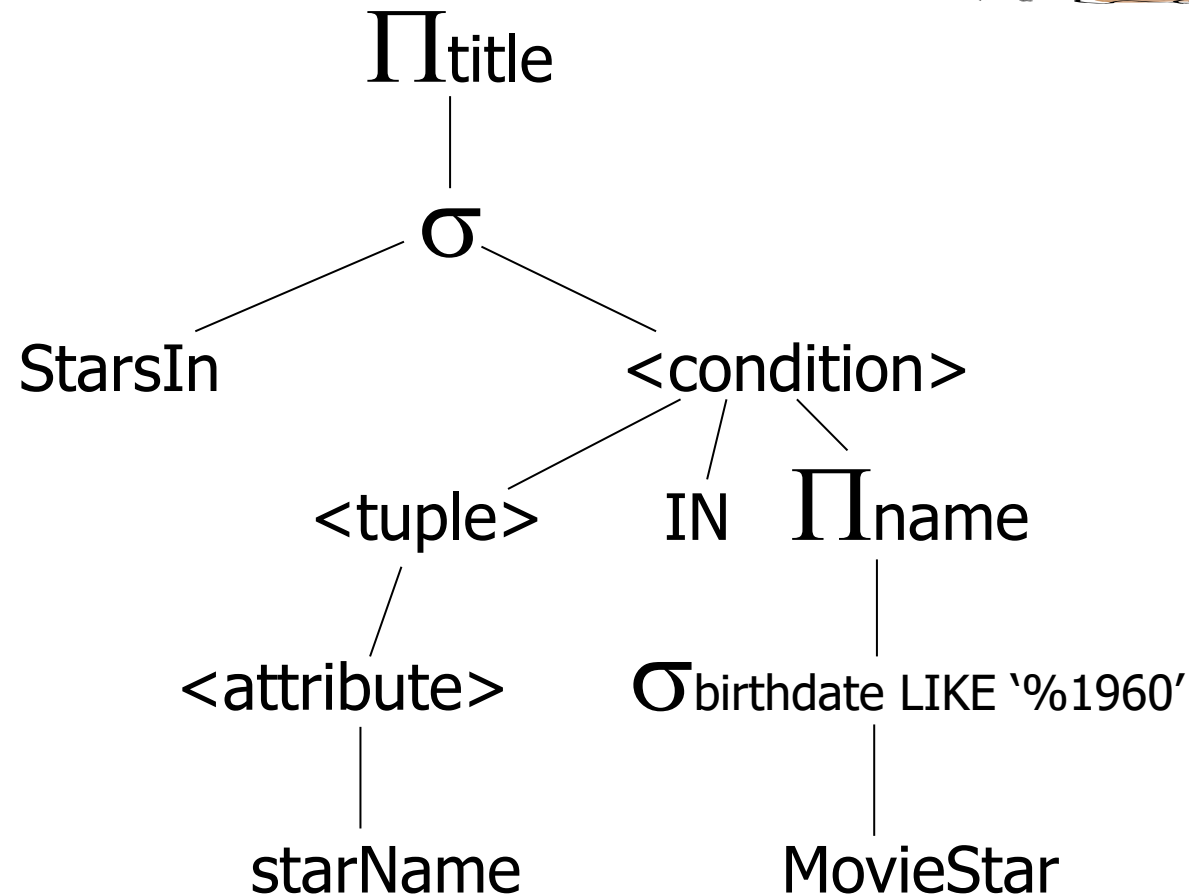


Fig. 7.15: An expression using a two-argument σ , midway between a parse tree and relational algebra





Example: 逻辑查询计划

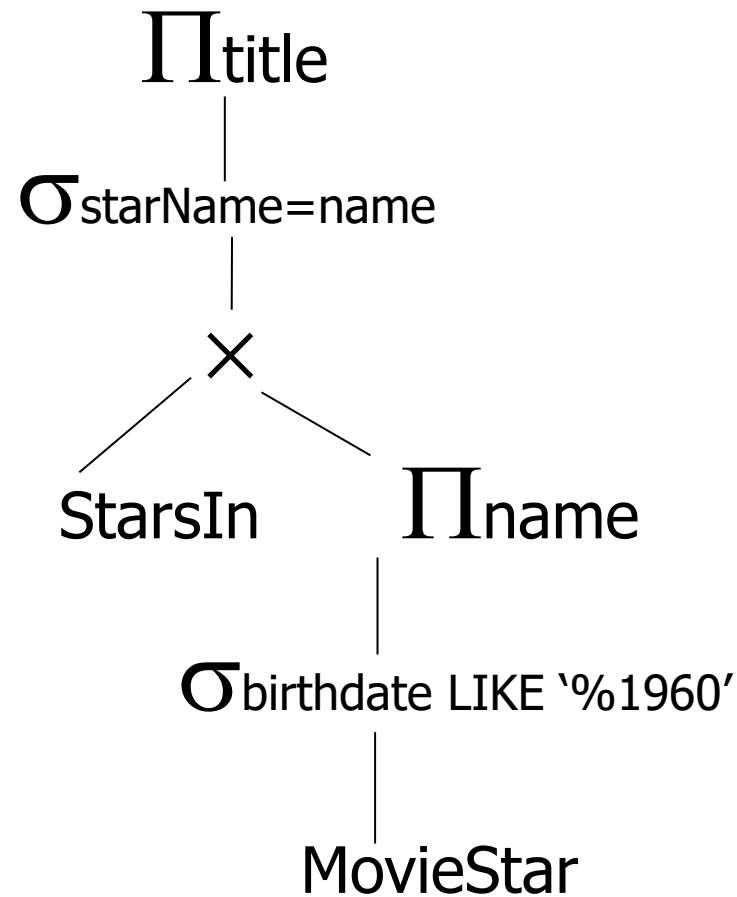
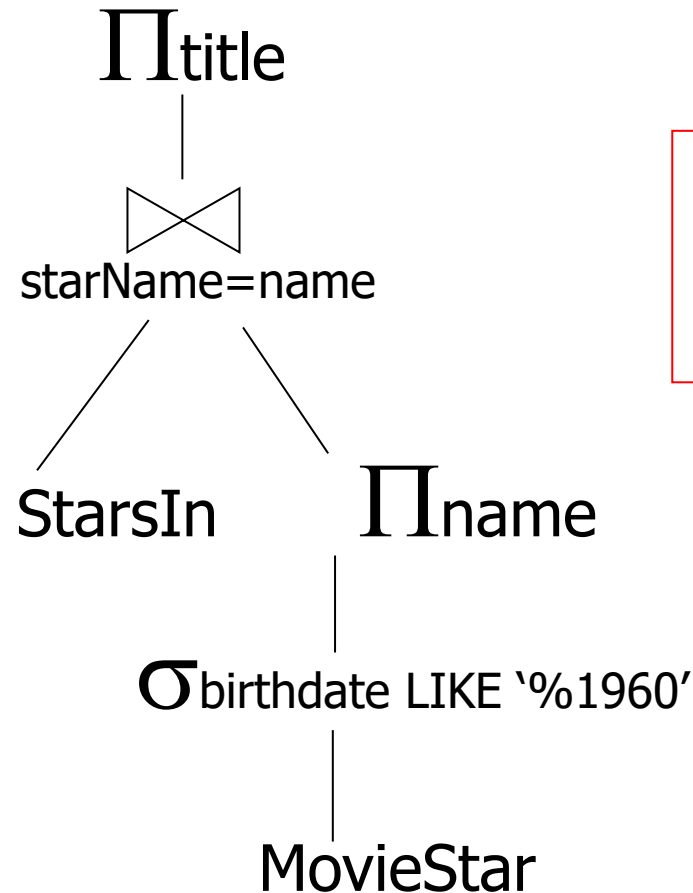


Fig. 7.18: Applying the rule for IN conditions





Example: 改进逻辑查询计划



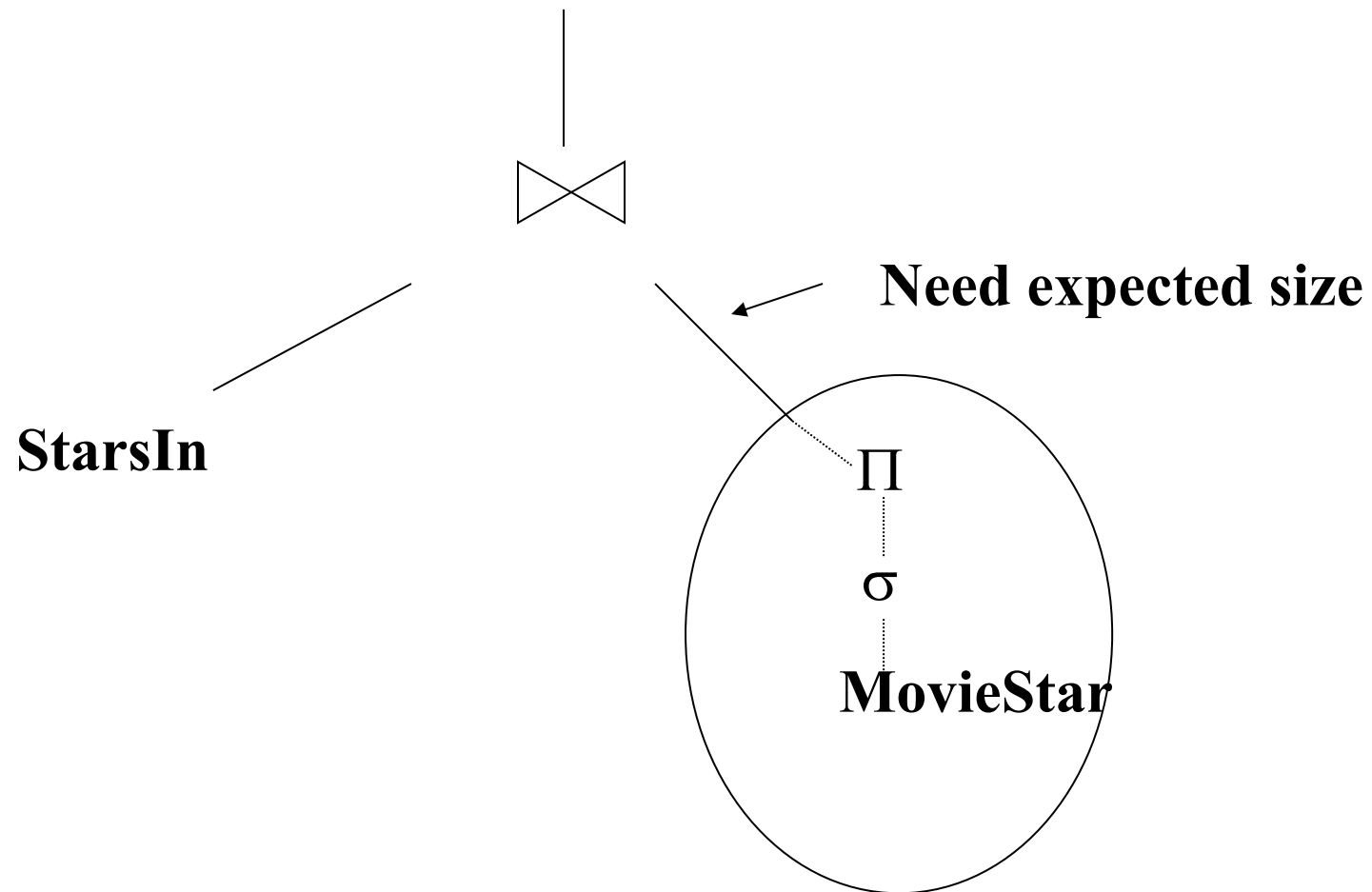
Question:
Push project to
StarsIn?

Fig. 7.20: An improvement on fig. 7.18.



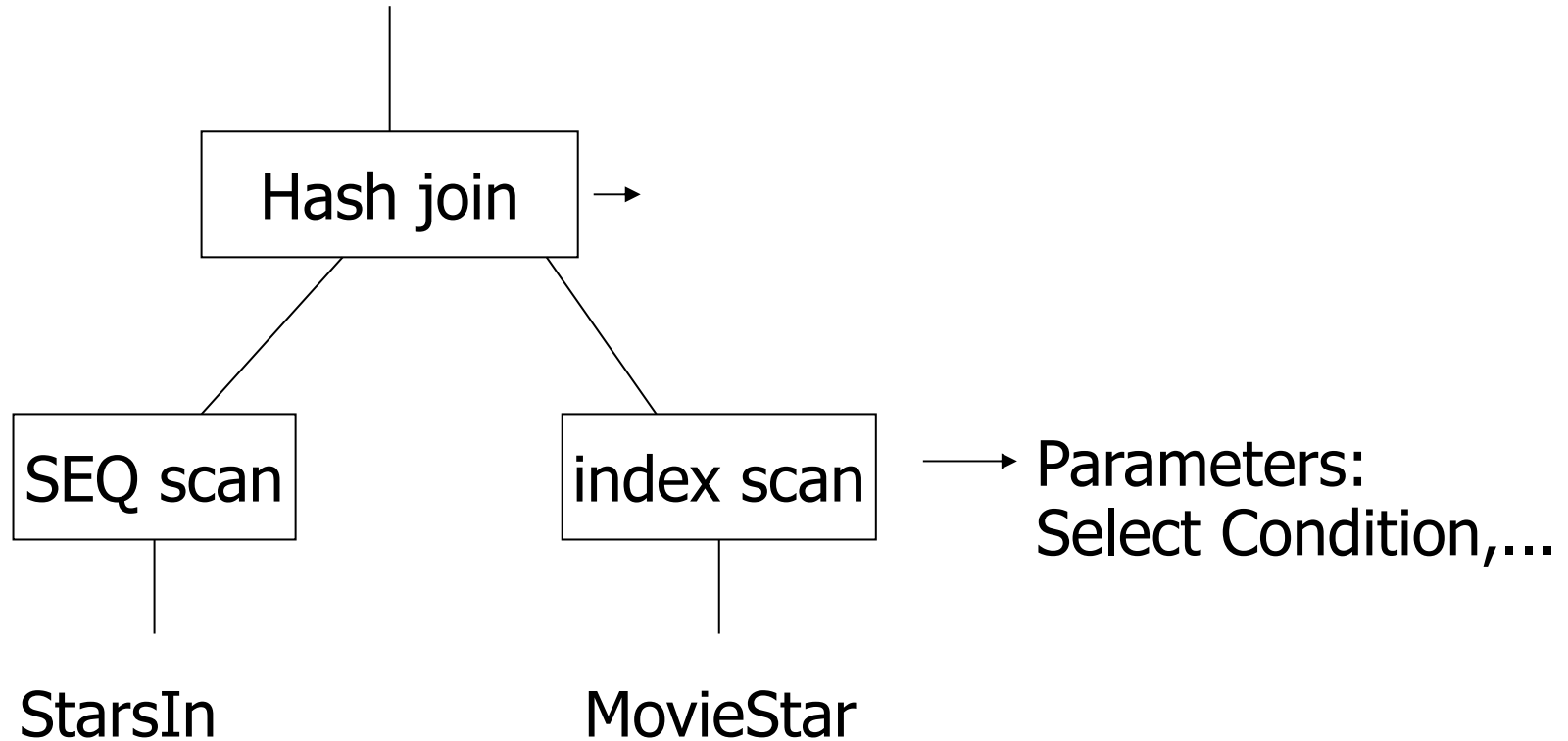


Example: 估计结果大小



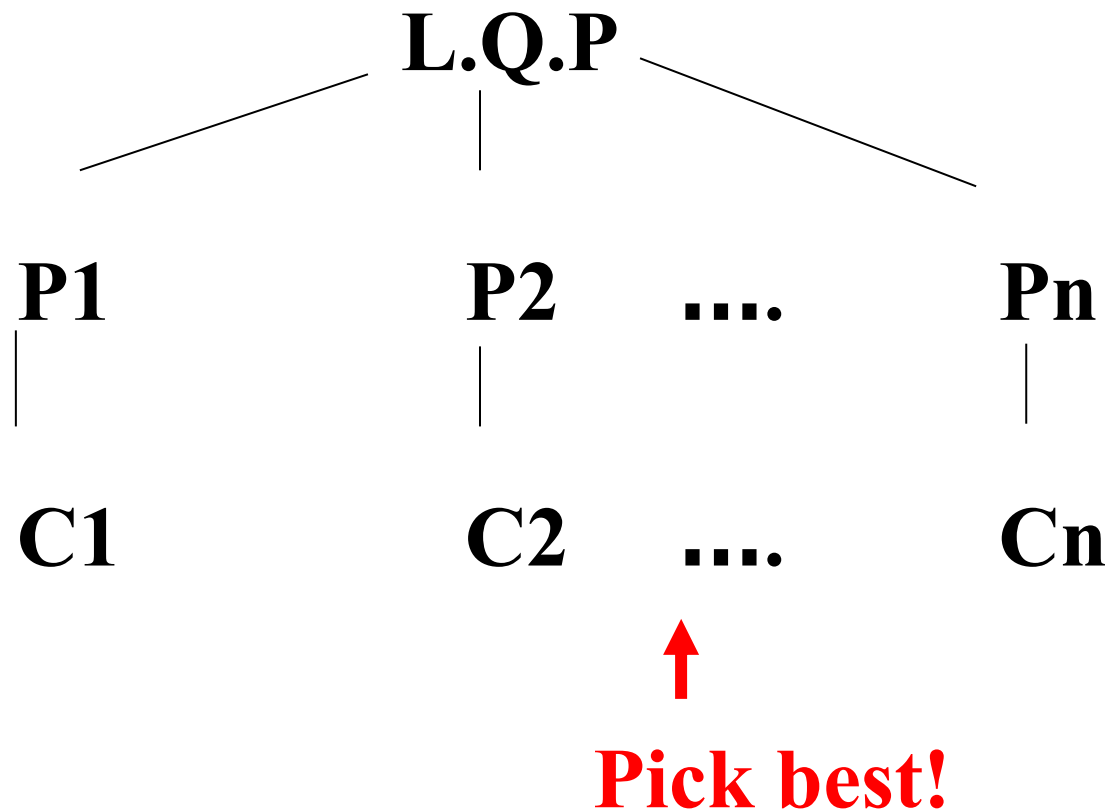


Example: 一个物理查询计划





Example: 估计代价





关系代数操作算法

- 选择操作算法
- 投影操作算法
- 连接操作算法
- 集合操作算法





目录

- 选择操作算法
- 投影操作算法
- 连接操作算法
- 集合操作算法





- 使用SQL语言，选择操作表示如下

```
SELECT *  
FROM R  
WHERE C1 AND C2 OR C3 ...
```

- 选择条件可以是简单条件(简单选择操作)
 - 仅包含关系R的一个属性的条件
- 选择条件也可以是复合条件(复杂选择操作)
 - 由简单条件经AND、OR、NOT等逻辑运算符连接而成的条件





简单选择操作算法

1. 线性搜索算法

- 顺序地读取被操作关系的每个元组；
- 测试该元组是否满足选择条件；
- 如果满足，则作为一个结果元组输出。

2. 二元搜索算法

- 条件：某属性相等比较且关系按该属性排序
- 对操作关系用二元搜索找到元组

如果关系具有 N 个元组

二元搜索需要 $O(\log(N))$ 时间



3. 主索引或HASH搜索算法

- 条件: 主索引属性或Hash属性上的相等比较
- 使用主索引或HASH方法搜索操作关系。

4. 使用主索引查找满足条件的元组

- 条件: 主索引属性上的非相等比较
- 使用主索引选择满足条件的所有元组。

5. 使用聚集索引查找满足条件的元组

- 条件: 具有聚集索引的非键属性上相等比较
- 使用这个聚集索引读取所有满足条件的元组

6. B -树和 B^+ -树索引搜索算法

- 条件: B 树或 B^+ 树索引属性上相等或非相等比较
- 使用 B^+ 树索引搜索查找所有满足条件组



复杂选择操作算法

7. 合取选择算法

- 合取条件中存在简单条件 C
- C 涉及的属性上定义有某种存取方法
- 存取方法适应于上述六个算法之一
- 用相应算法搜索关系, 选择满足 C 的元组, 并检验是否满足其他条件, 若满足, 作为结果元组。

8. 使用复合索引的合取选择算法

- 如果合取条件定义在一组属性上的相等比较
- 而且存在一个由这组属性构成的复合索引
- 使用这个复合索引完成选择操作。





目录

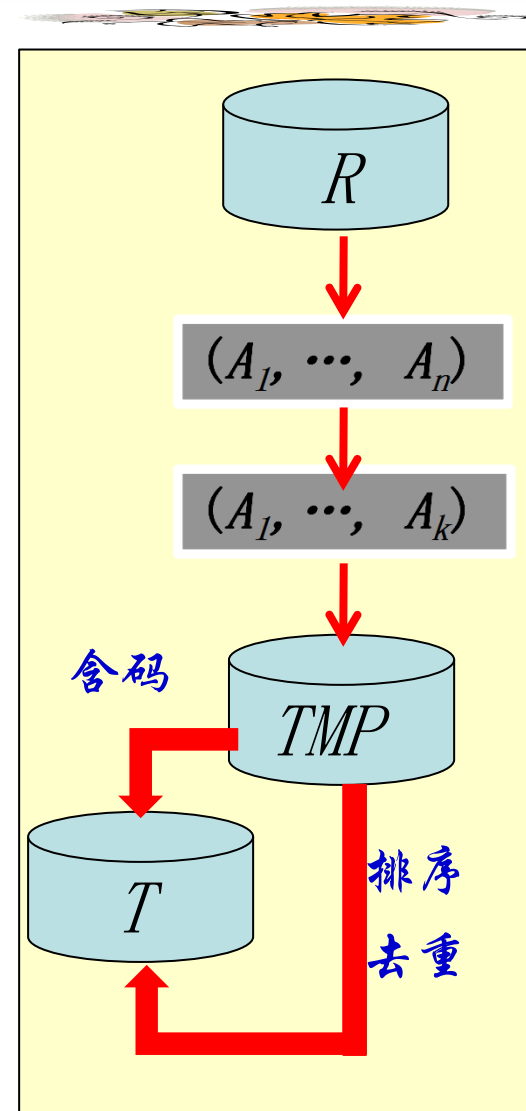
- 选择操作算法
- 投影操作算法
- 连接操作算法
- 集合操作算法





投影操作的实现算法

- 设 $\Pi_{A_1, \dots, A_k}(R)$ 是 R 上的投影操作
 - 若 $\{A_1, \dots, A_k\}$ 中包含 R 的码
 - 存取 R 的所有元组一次即可完成;
 - 操作结果具有与 R 同样, 只是每个元组仅包括 A_1, A_2, \dots, A_k 的值.
 - 如果投影属性表中不包含 R 的码
 - 需要删除操作结果中的重复元组
 - 可利用排序算法来实现投影操作





投影操作的实现算法

- 投影操作算法

输入: 具有 n 个元组的关系 R 。

输出: $T = \Pi_{A_1, \dots, A_k}(R)$ 。

FOR R 中每个元组 r DO

$r[A_1, \dots, A_k]$ 写入 TMP ;

IF $\{A_1, \dots, A_k\}$ 中包含 R 的码属性 THEN $T := TMP$; 结束;

ELSE 排序 TMP ; $i=1$; $j=2$;

 WHILE ($i \leq n$) DO

 写 $TMP(i)$ 到 T ;

 WHILE ($TMP(i) = TMP(j)$) DO

$j=j+1$;

$i=j$; $j=j+1$;





目录

- 选择操作算法
- 投影操作算法
- **连接操作算法**
- 集合操作算法





连接操作的实现算法

- 使用SQL语言, 关系 R 和 S 的连接操作表示为

```
SELECT     $R.*$ ,  $S.*$   
FROM       $R$ ,  $S$   
WHERE      $R.A \theta S.B$ 
```

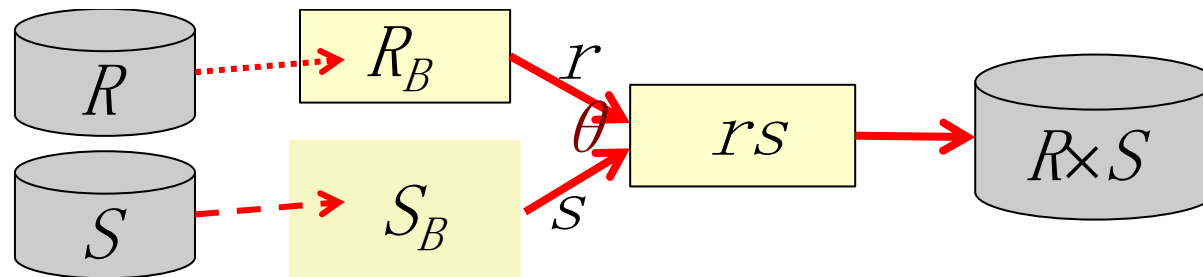
其中, θ 是算术比较符, 这种连接操作简称为 θ 连接.





θ -连接操作算法

- 设 B_R , B_S 为 R , S 的磁盘块数



FOR $i=1$ TO B_S DO

读 S 的第 i 个块到 S_B ;

FOR $j=1$ TO B_R DO

读 R 的第 j 块到 R_B ;

FOR $\forall r \in R_B, \forall s \in S_B$ DO

IF $r.A \theta s.B$

THEN (rs) 存入缓冲区, 写入结果关系;

采用较小的关系作
外层更有效

磁盘存取块数 $B_S + B_S * B_R + B_\theta$ (B_θ 连接结果块数)

磁盘搜索次数 $2B_S$



等值连接操作算法

- 等值连接和自然连接是应用最多的连接操作，两者的操作算法无本质区别。
- 下边主要讨论自然连接
 - 循环嵌套连接(Nest-Loop-Join)算法
 - 排序合并连接(Sort-Merge-Join)算法
 - Hash-连接(Hash-Join)算法





Nest-Loop- Join

输入: $R(A_1, \dots, A_i, \dots, A_n)$,
 $S(B_1, \dots, B_j, \dots, B_m)$,
连接条件 $R.A_i = S.B_j$

输出: R 与 S 的连接 T

FOR R 的每个磁盘块 X DO

读 X 到缓冲区 R_B ;

FOR S 的每个磁盘块 Y DO

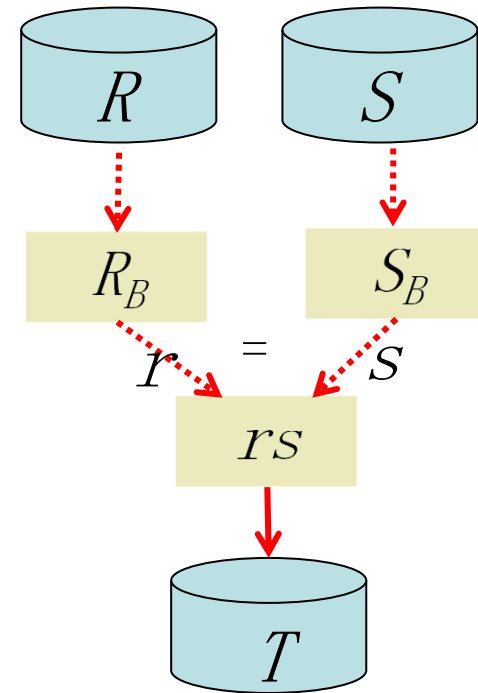
读 Y 到缓冲区 S_B ;

FOR $\forall r \in R_B, \forall s \in S_B$ DO

IF $r[A_i] = s[B_j]$

如何优化?

THEN (rs) 存入缓冲区, 写入 T ;



算法的磁盘存取块数: $B_R + B_R B_S + B_T$





等值连接操作算法

- 优化

- 一次读入尽可能多的元组
- 使用尽可能多的(M)内存块来存储属于关系R的元组，R是外层循环中的关系。
- 假定 $B(R) \leq B(S)$, $B(R) \geq M$





等值连接操作算法

思考：

假定 $B(S) = 1000$ 且 $B(R) = 500$ ，并令 $M = 101$ 。我们将使用100个内存块来为R进行缓冲。因此算法中的外层循环需迭代5次。

每一次迭代中，在第二层循环内必须用1000个磁盘I/O来完整地读取S。因此，磁盘I/O的总数量是5500。

若颠倒R和S的角色，情况如何呢？





等值连接操作算法

Sort-Merge-Join

- 如果关系 R 和 S 的元组已经在连接属性 $R.A_i$ 和 $S.B_j$ 上物理地排序
 - 按排序顺序扫描 R 和 S , 查找在 $R.A_i$ 和 $S.B_j$ 上具有相同值的 R 和 S 的元组, 进行连接.
 - 磁盘存取块数: $B_R + B_S + B_T$
- 如果关系 R 和 S 的元组都未排序
 - 使用下边的 Sort-Merge 连接算法。
 - 用 $X(i)$ 表示已排序关系 X 的第 i 个元组。





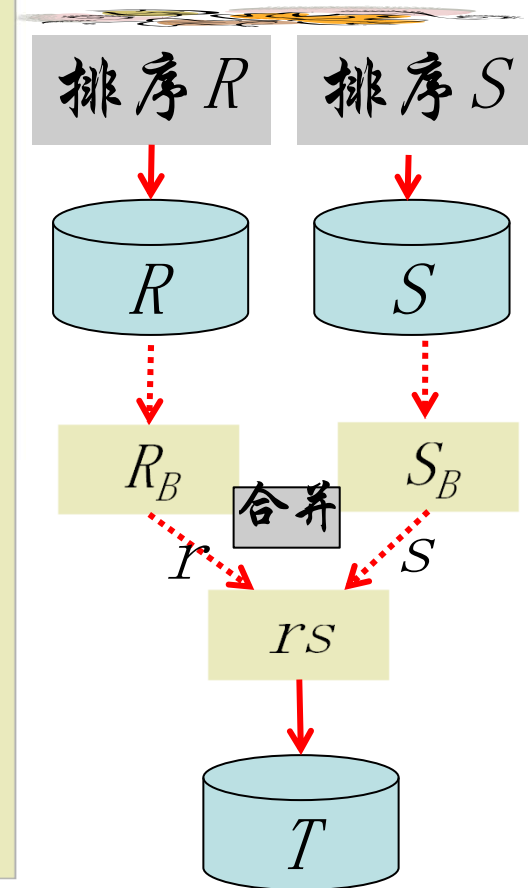
• Sort-Merge Join算法

输入: $R(A_1, \dots, A_i, \dots, A_n)$,
 $S(B_1, \dots, B_j, \dots, B_m)$,
连接条件 $R.A_i = S.B_j$

输出: R 与 S 的连接 T

1. 按属性 $R.A_i$ 值排序 R ;
2. 按属性 $S.B_j$ 值排序 S ;
3. 扫描 R 和 S 一遍, 产生

$$T = \{R(k)S(m) \mid R(k)[A_i] = S(m)[B_j]\}.$$



算法的磁盘存取块数

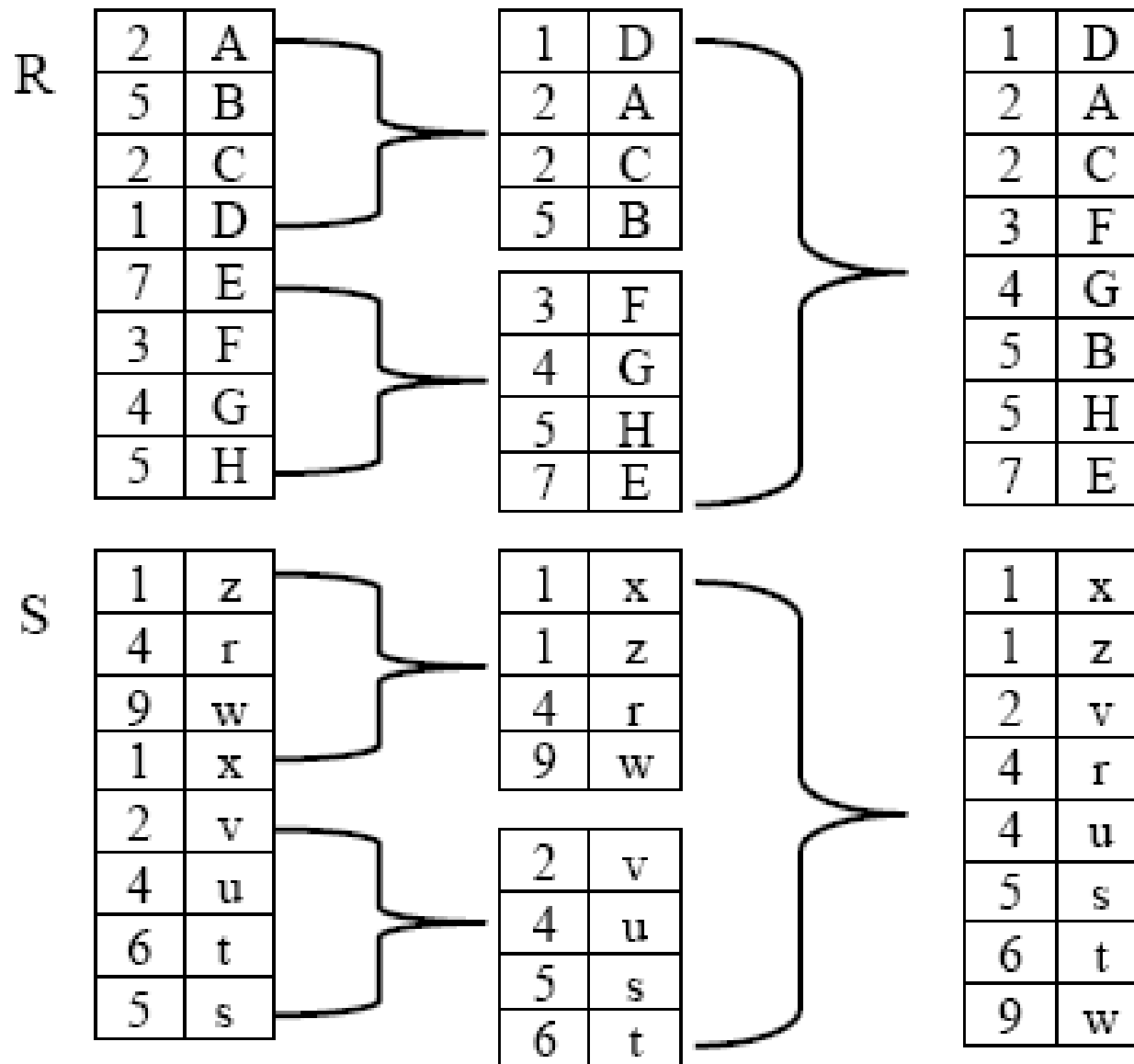
$$O(B_R \log_M B_R + B_S \log_M B_S + B_R + B_S + B_T)$$



Sort-Join Example

Sort Phase

M=4. blocking factor=1.





Sort-Join Example

Merge Phase

M=4, blocking factor=1.



R

1	D
2	A
2	C
3	F
4	G
5	B
5	H
7	E

 In memory after join on 1.


S

1	x
1	z
2	v
4	r
4	u
5	s
6	t
9	w

 Brought in for join on 1.
 In memory after join on 1.

Buffer

1	D	R
1	x	S
1	z	extra
		extra

Output

1	D	x
1	D	z



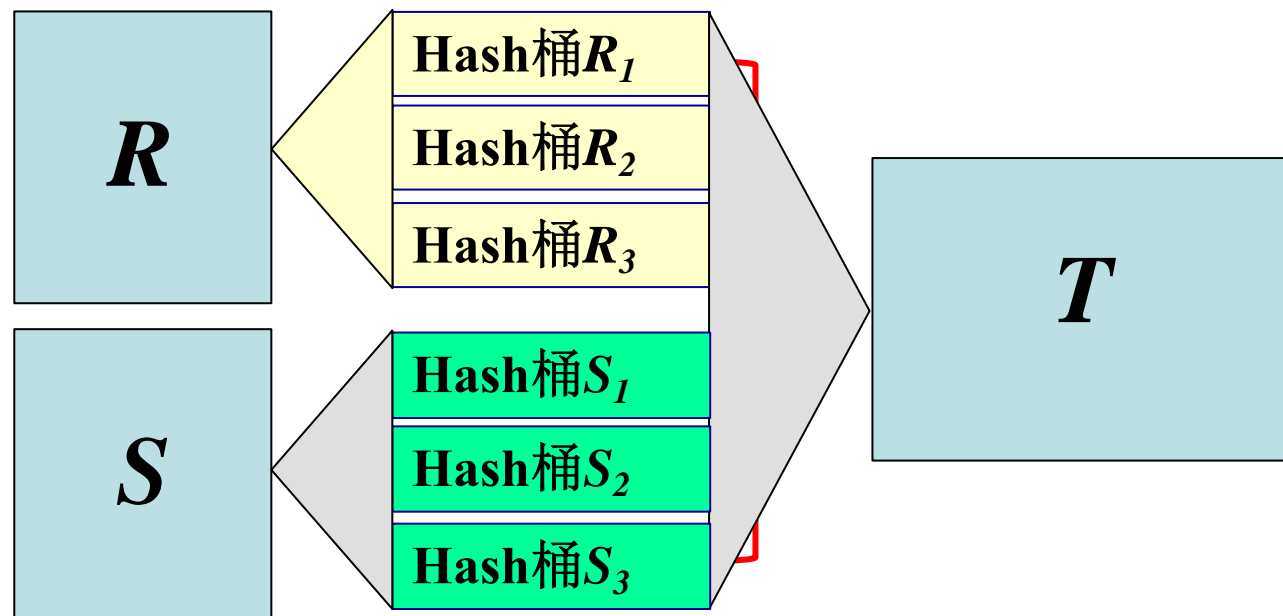
Hash-Join

- 第一阶段(Hash)

- 扫描 R 和 S , 使用定义在连接属性上的Hash函数把 R 和 S 的元组分别构造成Hash文件 HR 和 HS ;

- 第二阶段(Probe)

- 对于 H 问题: 为什么 R_1 不与 S_2 做连接? 中 R 和 S 的元组在连接属性上的值, 产生 R 和 S 的连接结果。





Hash-Join算法

输入: 关系 $R(A_1, \dots, A_i, \dots, A_n)$, $S(B_1, \dots, B_j, \dots, B_m)$, 连接条件
 $R.A_i = S.B_j$, Hash函数 $h(x)$, 值域 $\{1, \dots, N\}$

输出: R 与 S 的连接 T

FOR 每个 $t \in R$ **DO**

t 写入 H_R 的第 $h(t[A_i])$ 个Hash桶;

ENDFOR

FOR 每个 $s \in S$ **DO**

s 写入 H_S 的第 $h(s[B_j])$ 个Hash桶;

ENDFOR;

FOR $i=1$ TO N **DO**

连接 H_R 和 H_S 的第 i 个Hash桶, 结果写入 T ;

ENDFOR



算法的磁盘存取块数:

$$2(B_R + B_S) + B_T$$

Hash Join Example

Partition Phase

R

2	A
5	B
2	C
1	D
7	E
3	F
4	G
5	H

S

1	z
4	r
9	w
1	x
2	v
4	u
6	t
5	s

Partitions for R

$h(x) = 0$

3	F

$h(x) = 1$

1	D	4	G
7	E		

$h(x) = 2$

2	A	2	C
5	B	5	H

Partitions for S

$h(x) = 0$

9	w
6	t

$h(x) = 1$

1	z	1	x
4	r	4	u

$h(x) = 2$

2	v
5	s

$M=4, bfr=2, h(x) = x \% 3$

Hash Join Example

Join Phase on Partition 1

Partition 1 for R

$$h(x) = 1$$

1	D	4	G
7	E		

Partition 1 for S

$$h(x) = 1$$

1	z	1	x
4	r	4	u

Buffers

1	D
7	E

4	G

1	z
4	r

1	x
4	u

Output

1	D	x
1	D	z
4	G	r
4	G	u

Note that both relations fit entirely in memory, but can perform join by having only one relation in memory and reading 1 block at a time from the other one.



目录

- 选择操作算法
- 投影操作算法
- 连接操作算法
- 集合操作算法





集合操作算法

- 输入关系的约束
 - 具有相同的属性集合
 - 并且属性的排列顺序必须也相同
- 实现这些操作的常用算法
 - 首先利用排序算法在相同的键属性上排序两个操作关系；
 - 然后扫描这两个排序后的关系，完成并、交或差操作。





总结

- 本章重点
 - 掌握选择操作的实现算法、连接操作的实现算法、投影操作的实现算法、集合操作的实现算法；
 - 掌握关系代数表达式查询处理方法。





**Now let's go to
Next Chapter**

