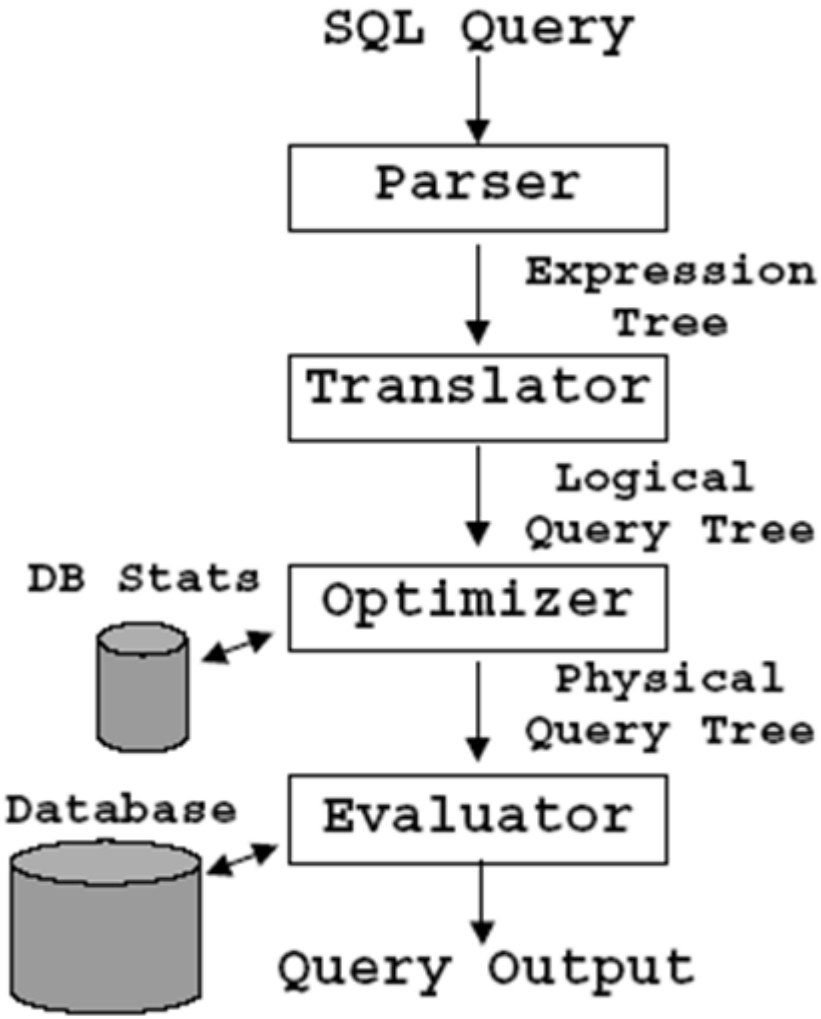
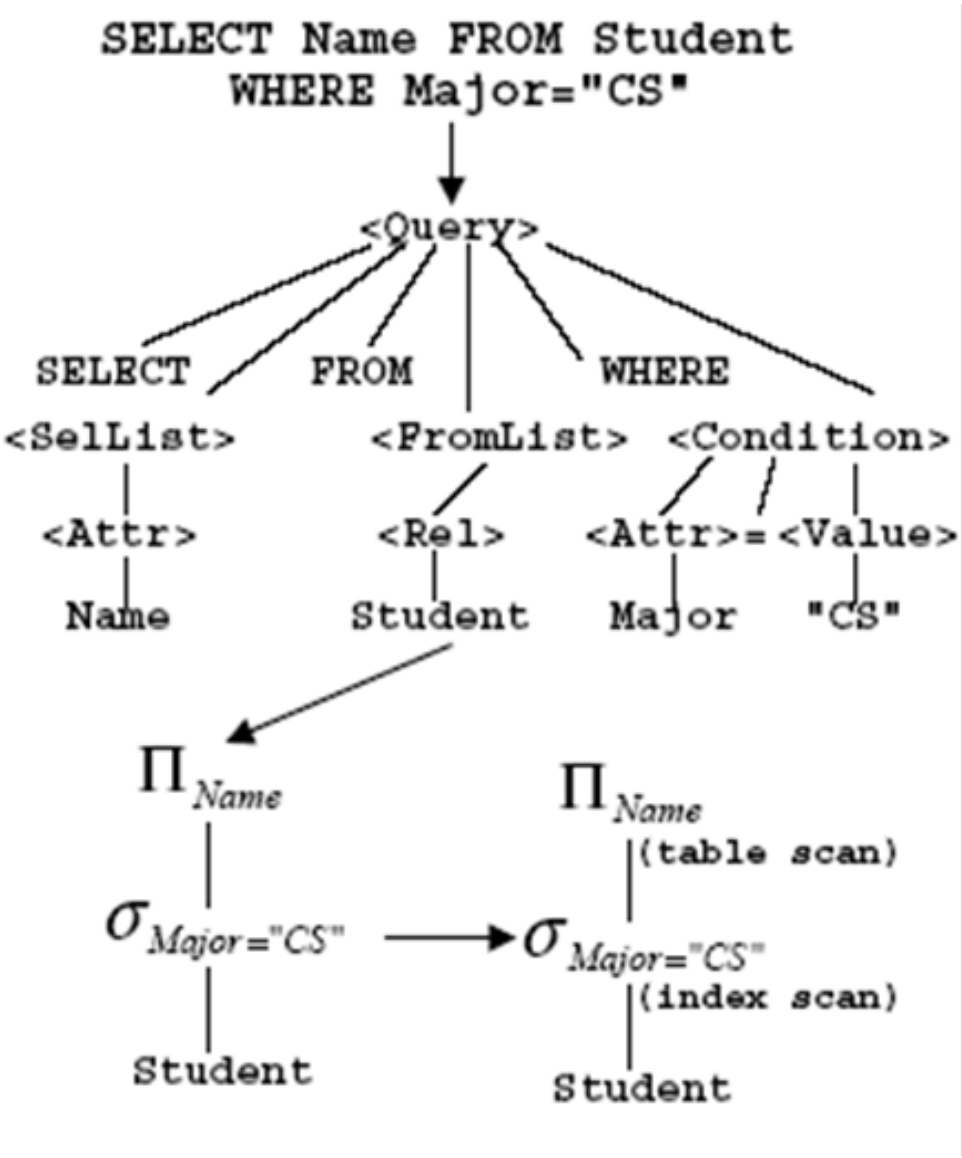
实现篇

第六章 查询处理与优化

主讲：高宏

海量数据计算研究中心


SQL Query
Parser
Expression Tree
Translator
Logical Query Tree
DB Stats
Optimizer
Physical Query Tree
Database
Evaluator
Query Output


SELECT Name FROM Student
WHERE Major="CS"
<Query>
SELECT
FROM
WHERE
<SelList>
<PromList>
<Condition>
<Attr>
<Rel>
<Attr>=<Value>
Name
Student
Major
"CS"
Π Name
σ Major="CS"
Student
Π Name
(table scan)
σ Major="CS"
(index scan)
Student

几个概念

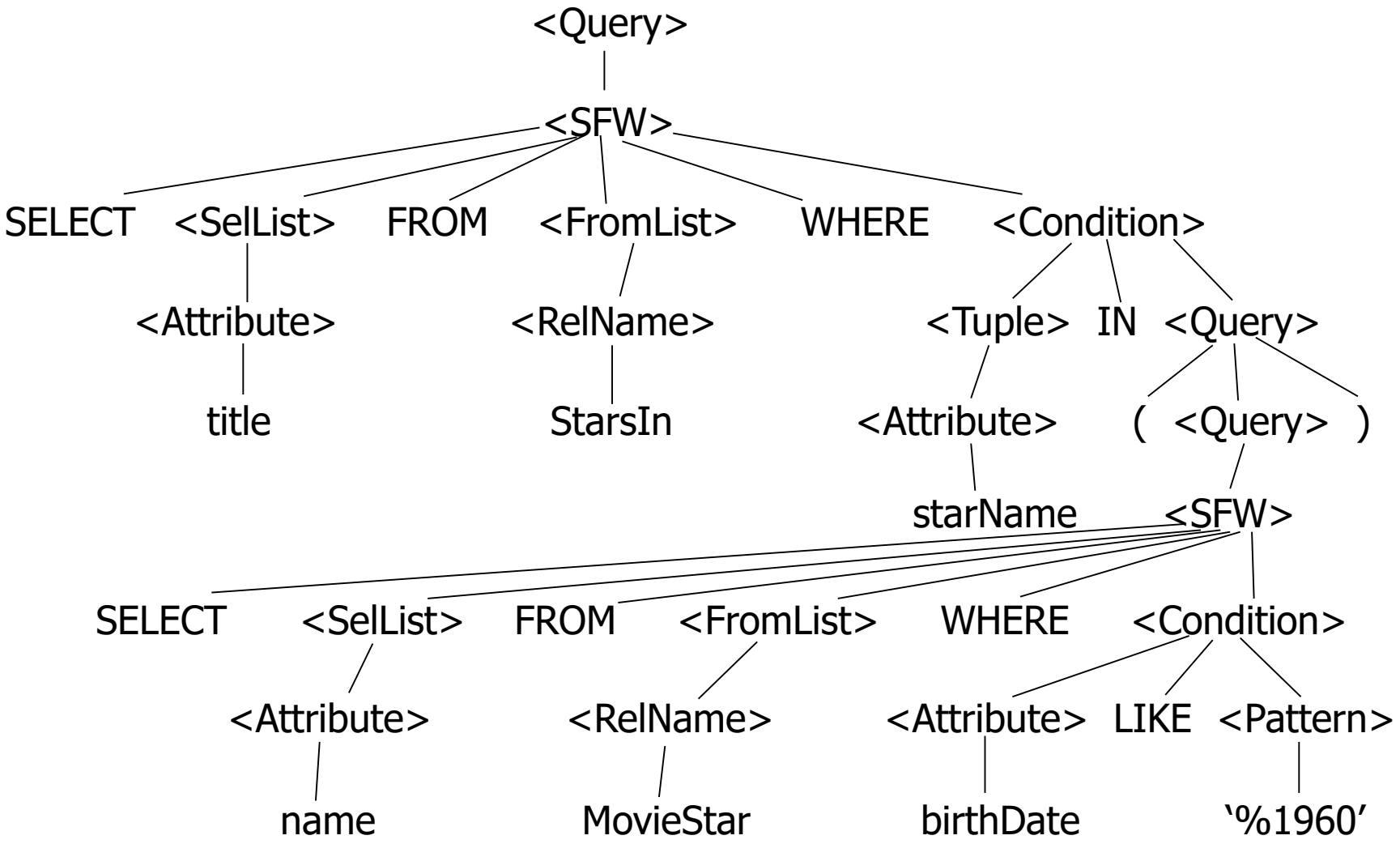
- Parser Tree (Expression Tree)
  - 由select、from、where组成的语法树
- Logical Query Plan Tree
  - 由基本关系操作符组成的查询树
    - 如：选择、投影、连接等
- Physical Query Plan Tree
  - 由物理操作符组成的查询树
  - 物理操作符
    - 顺序扫描、索引扫描等
    - Hash-join、sort-merge-join等

Example: SQL query

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

(找到1960 年出生的影星参演的电影名字)

Example: Parser Tree



Example: Generating Relational Algebra

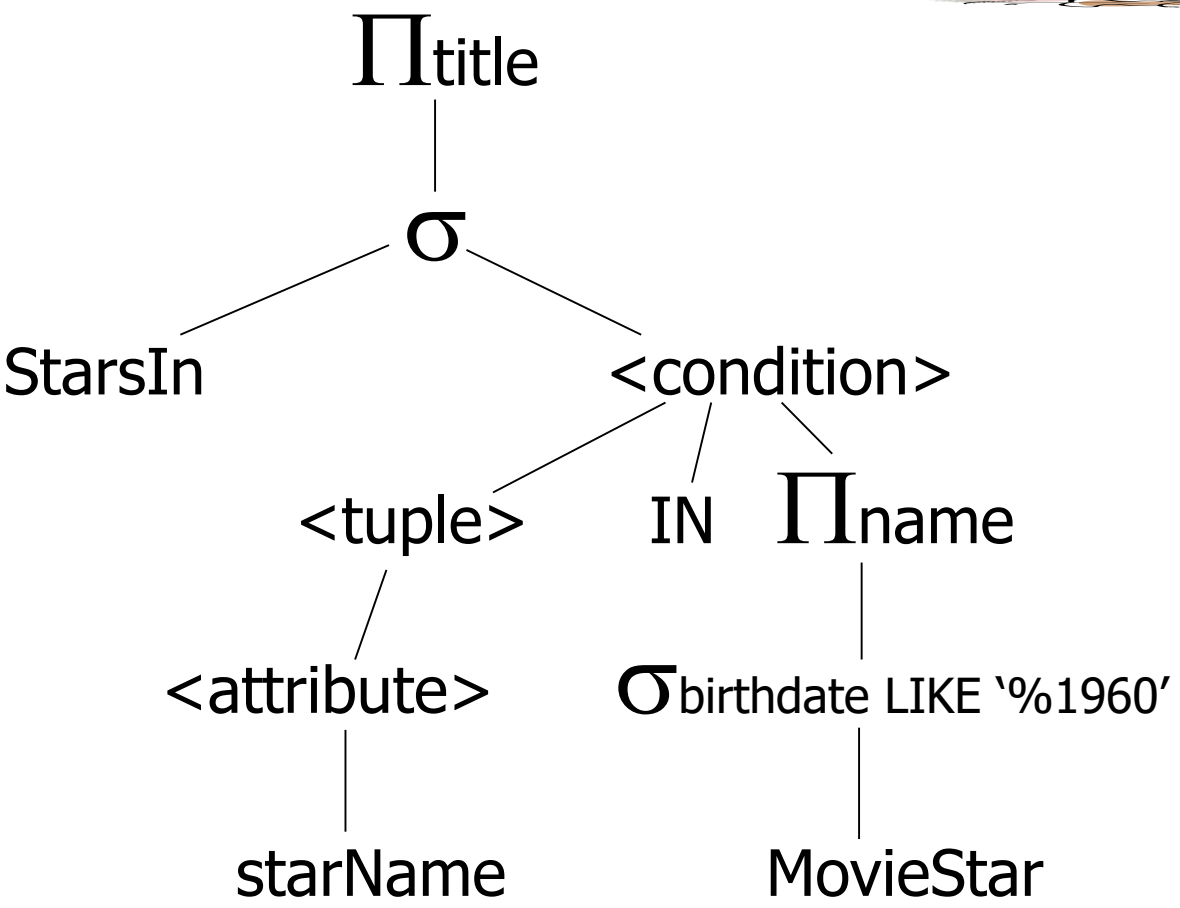


Fig. 7.15: An expression using a two-argument σ, midway between a parse tree and relational algebra

Example: Logical Query Plan

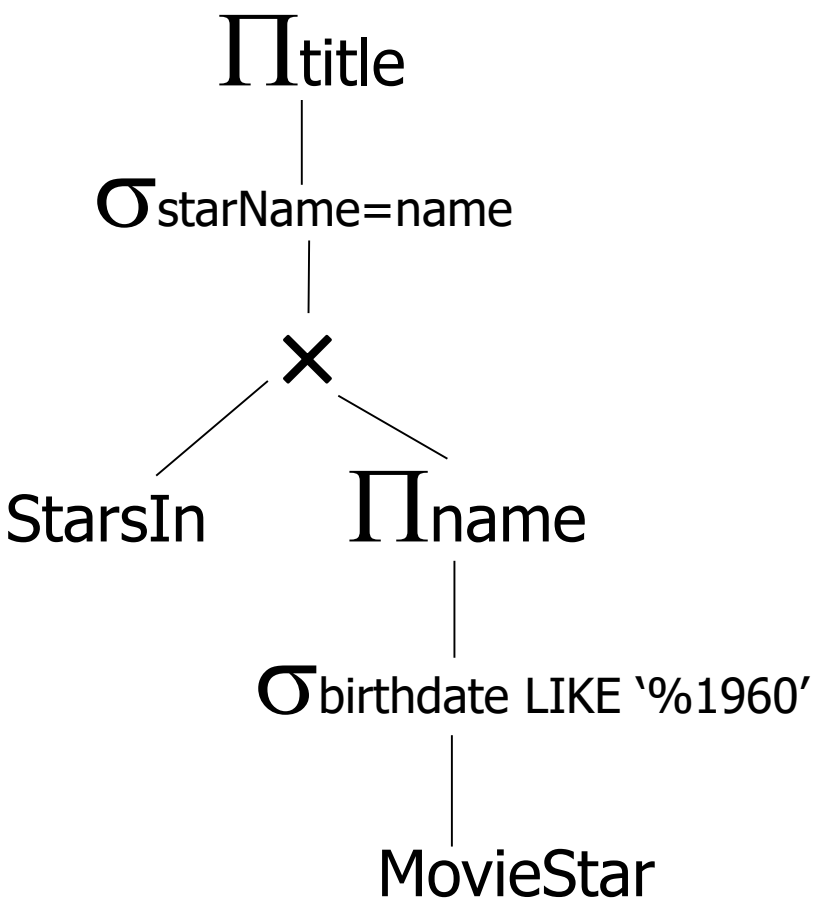
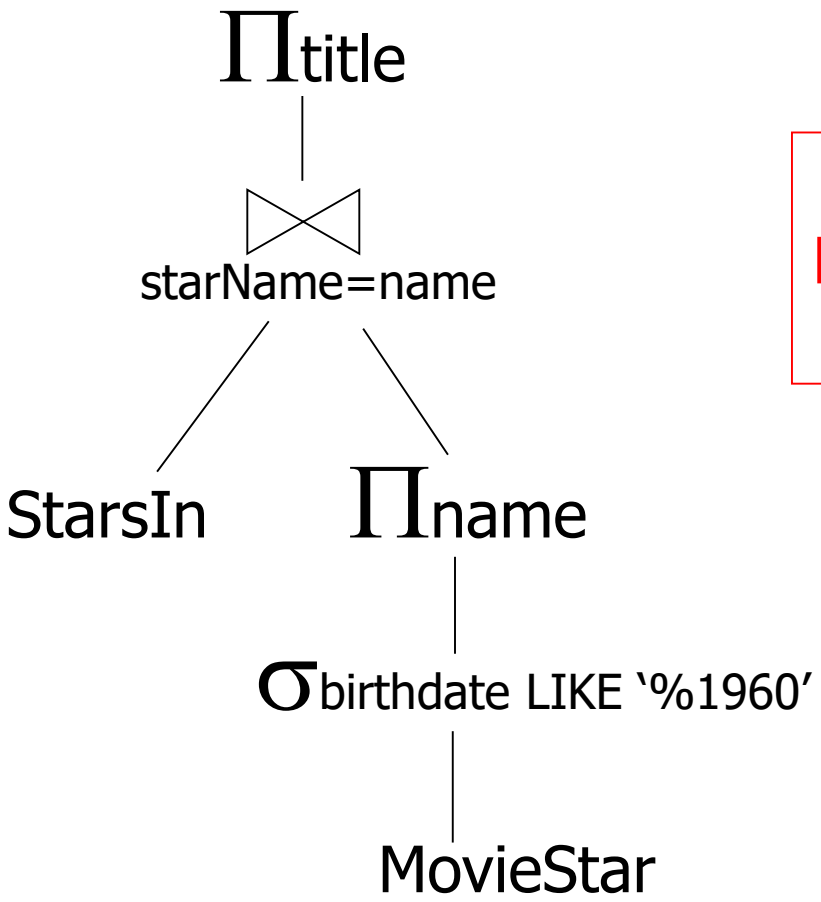


Fig. 7.18: Applying the rule for IN conditions

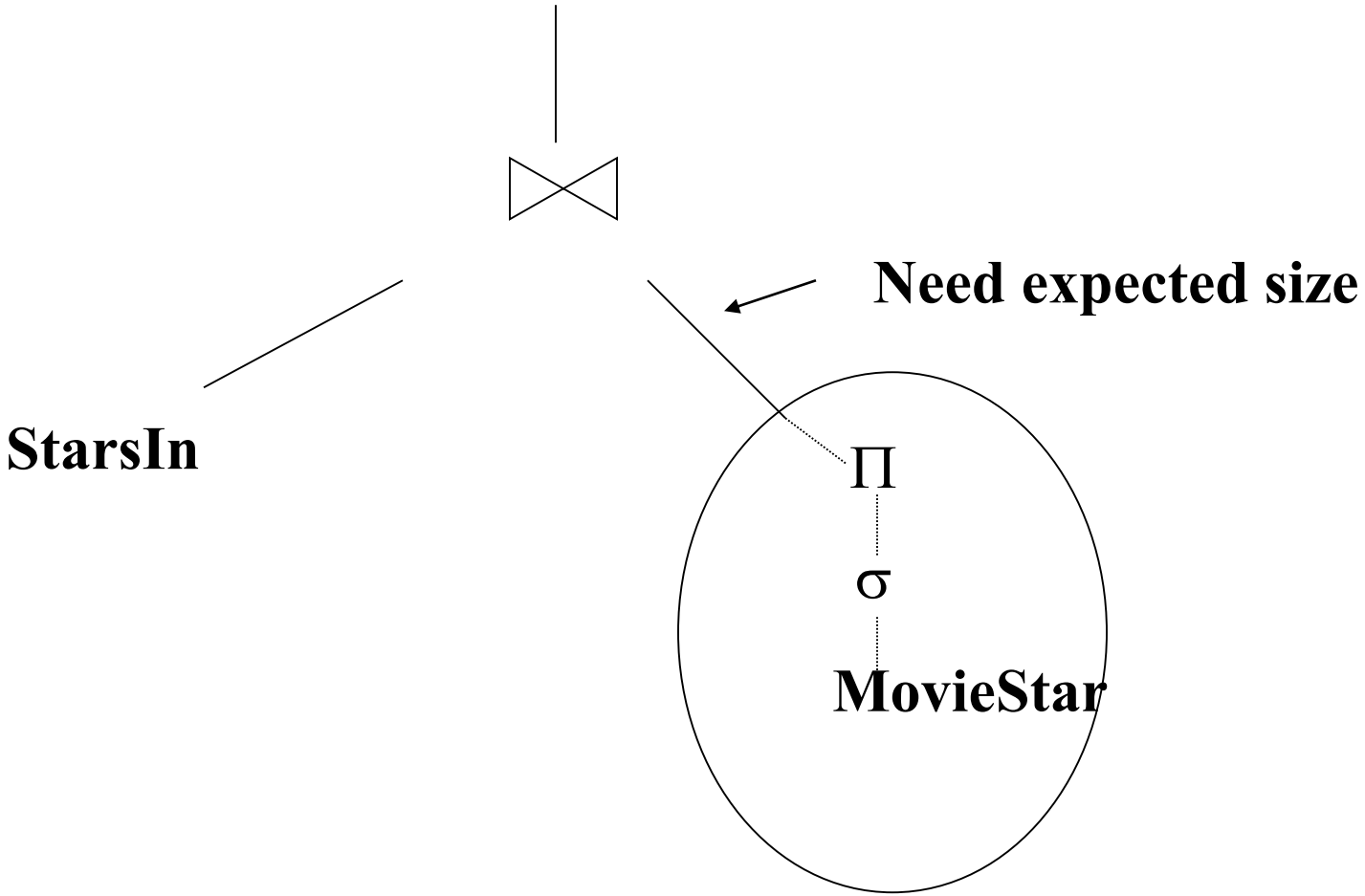
Example: Improved Logical Query Plan



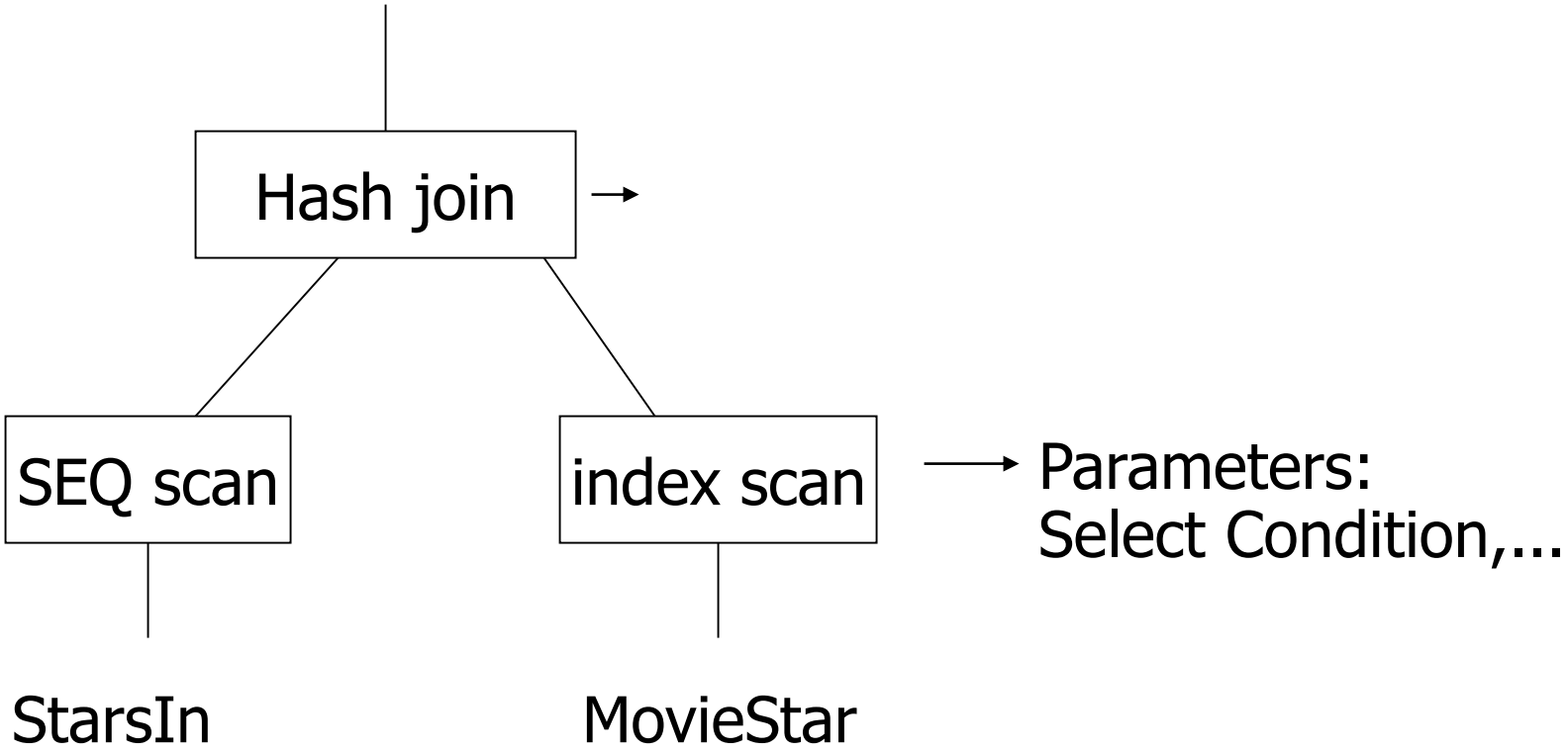
Question:
Push project to StarsIn?

Fig. 7.20: An improvement on fig. 7.18.

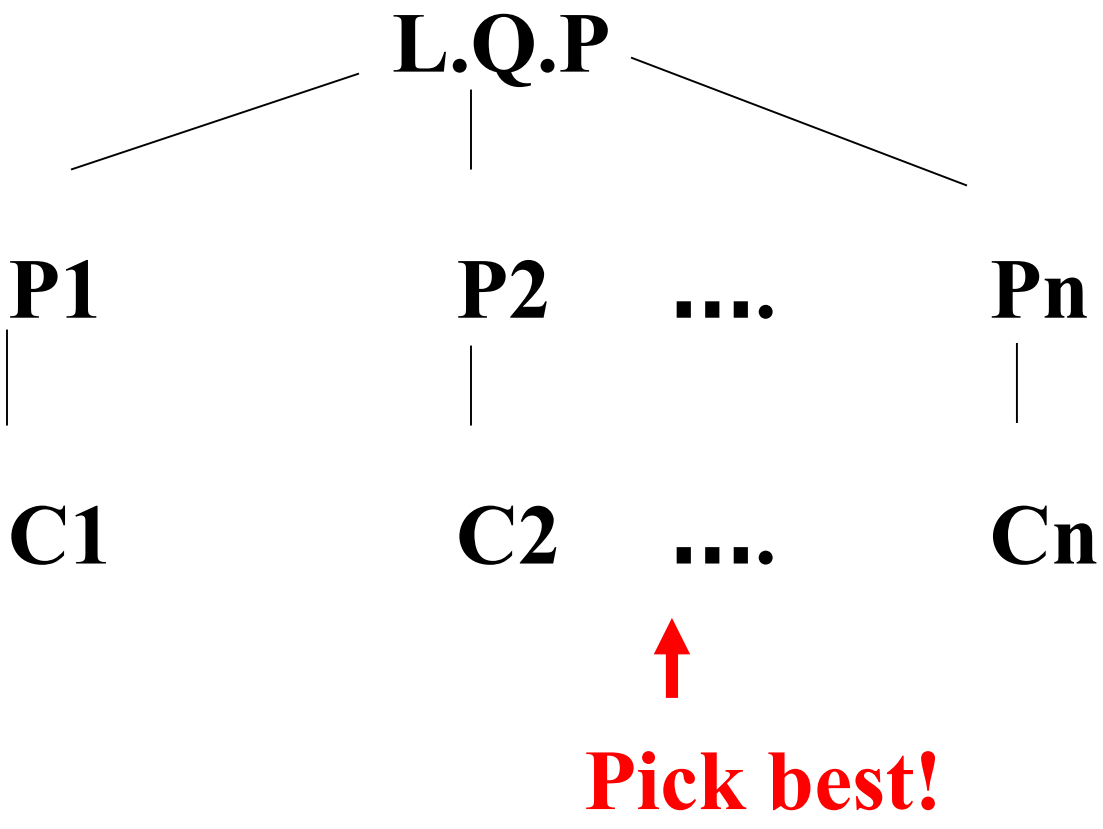
Example: Estimate Result Sizes



Example: One Physical Plan



Example: Estimate costs



Outline

- 关系代数操作算法
- 查询优化

6.1 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

- 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

- 顺序地读取被查询关系的每个元组；
- 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. 二元或插值搜索算法

- 条件:某属性相等比较且关系按该属性排序.
- 对查询关系用二元或插值搜索找到元组

如果关系具有N个磁盘块

二元搜索需要$O(\log(N))$时间

插值搜索需要$O(\log(\log(N)))$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。

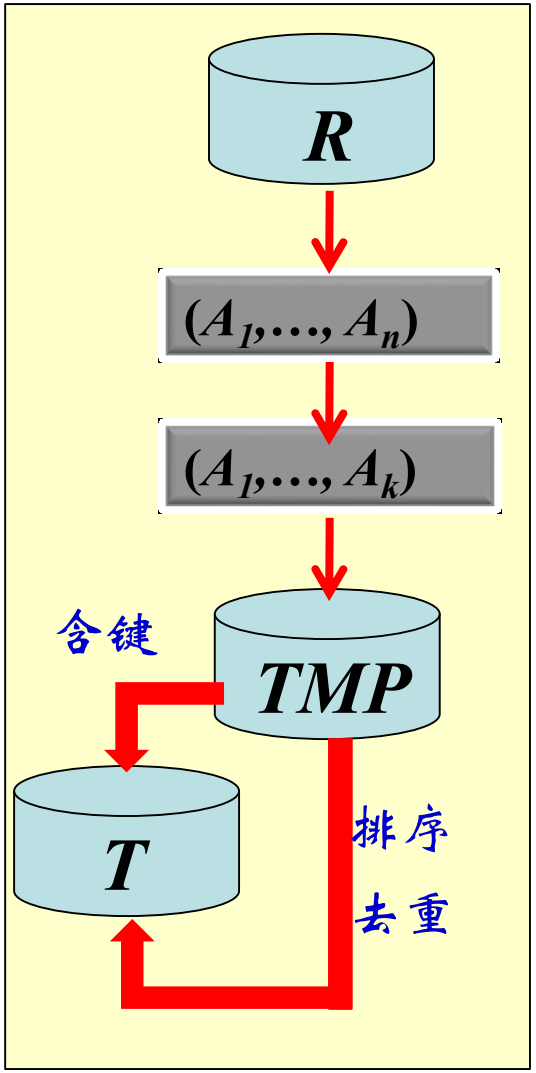
8. 使用复合/多维索引的合取选择算法

- 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/多维索引
- 使用这个复合/多维索引完成选择操作。

- 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$\Pi_{A_1, \ldots, A_k}(R)$是$R$上的投影操作
  - 若$\{A_1, \ldots, A_k\}$中包括R的键
    - 存取R的所有元组一次即可完成;
    - 操作结果有与R同样个数元组，每个元组仅包括A_1、A_2、…、A_k的值.
  - 如果投影属性表中不包含R的键
    - 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



- **投影操作算法**

输入：具有n个元组的关系R。

输出：$T=\Pi_{A_1, \ldots, A_k}(R)$。

FOR　R中每个元组r　DO

　　$r[A_1, \ldots, A_k]$写入TMP;

IF　$\{A_1, \ldots, A_k\}$中包含R的键属性 THEN　$T := TMP$;　结束；

ELSE　排序TMP;　$i=1$;　$j=2$;

　　WHILE　$(i \le n)$　DO　　//去重

　　写$TMP(i)$到T;

　　WHILE　$(TMP(i)=TMP(j))$　DO

　　$j=j+1$;

　　$i=j$;　$j=j+1$;

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

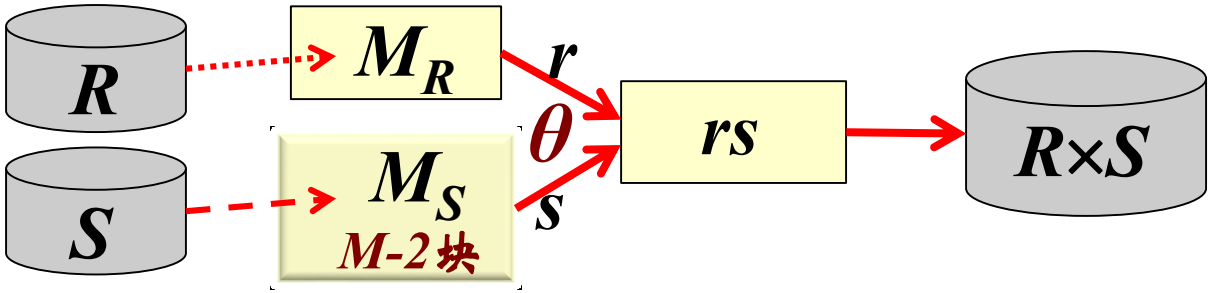
- 使用SQL语言，关系R和S的连接操作表示为

```
SELECT  R.*, S.*
FROM    R, S
WHERE   R.A θ S.B
```

  - 其中，θ是算术比较符，该连接操作简称为θ连接.

θ连接操作算法

- 设$M \le B_R$, $M \le B_S$, $B_S \le B_R$. S划分为$B_S/(M-2)$个子集.



FOR i=1 **TO** $B_S/(M-2)$ **DO**
  读S的第i个子集合到M_S;
  FOR j=1 **TO** B_R **DO**
    读R的第j块到M_R;
    FOR $\forall r \in M_R, \forall s \in M_S$ **DO**
      IF ***r.Aθs.B***
      THEN (rs)存入缓冲区，写入结果关系;

磁盘存取块数$B_S+(B_S/(M-2))B_R+B_\theta$($B_\theta$连接结果块数)

等值连接操作算法

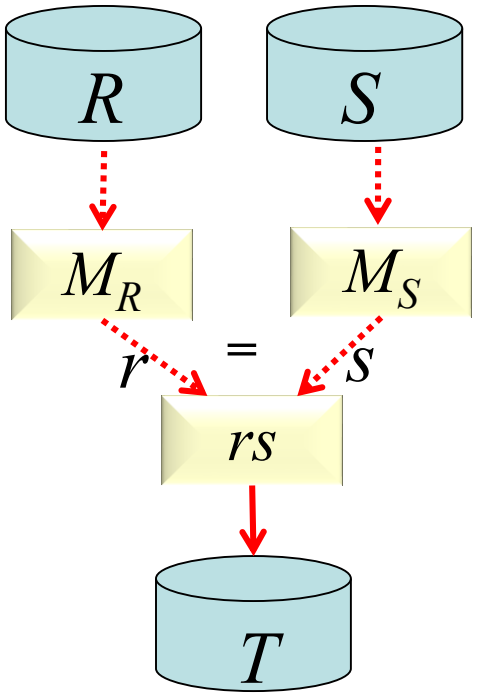
- 等值连接和自然连接是应用最多的连接操作，两者的操作算法无本质区别.
- 下边主要讨论自然连接
  - 循环嵌套连接(Nest-Loop-Join)算法
  - 排序合并连接(Sort-Merge-Join)算法
  - Hash-连接(Hash-Join)算法

Nest-Loop- Join

输入：$R(A_1, \ldots, A_i, \ldots, A_n)$，
$S(B_1, \ldots, B_j, \ldots, B_m)$，
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M_R;
  FOR S的每个磁盘块Y DO
    读Y到缓冲区M_S;
    FOR $\forall r \in M_R, \forall s \in M_S$ DO
      IF $r[A_i] = s[B_j]$
      THEN (rs)存入缓冲区，写入T;



如何优化？

算法的磁盘存取块数：$B_R + B_R B_S$

- 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$B(R) \leq B(S)$， $B(R) \geq M$
  - 性能分析：
    - 外层循环的迭代次数为$B(R)/M$
    - 每一次迭代时，读取R的M个块，和S的$B(S)$个块
    - 这样，磁盘I/O的数量为：

$$\frac{B(R)}{M}(M + B(S)) \quad \text{即：} \quad B(R) + \frac{B(S) \times B(R)}{M}$$

思考：

假定B(S) = 1000且B(R) = 500，并令M = 100。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500+5×1000=5500。

若颠倒R和S的内外层嵌套关系，情况如何呢？

1000+10×500=6000

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.

R

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

S

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

Output

1	D	x
1	D	z

  - 磁盘存取块数：至少(B(R) + B(S))

Sort-Merge-Join

- 如果关系R和S的元组都未排序
  - 分别按照连接属性R.A和S.B排序关系R、S
  - 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.

• Sort-Merge Join算法

输入：$R(A_1, \ldots, A_i, \ldots, A_n)$，

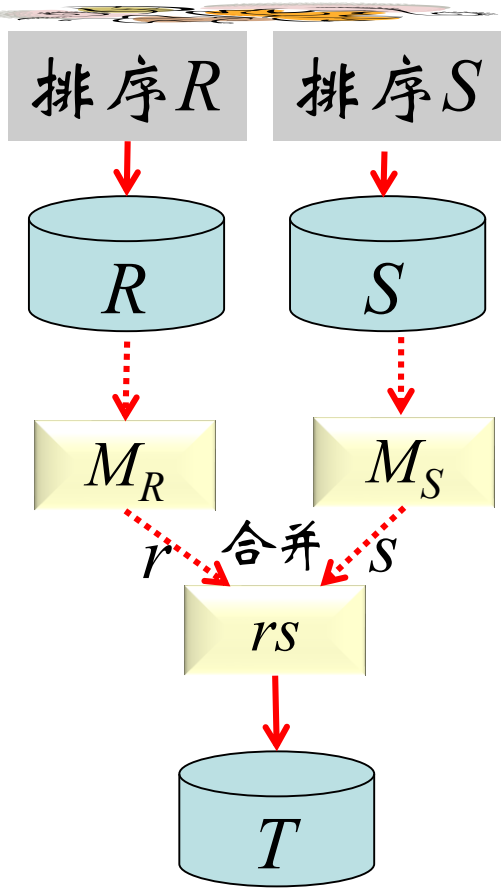
$S(B_1, \ldots, B_j, \ldots, B_m)$，

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

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

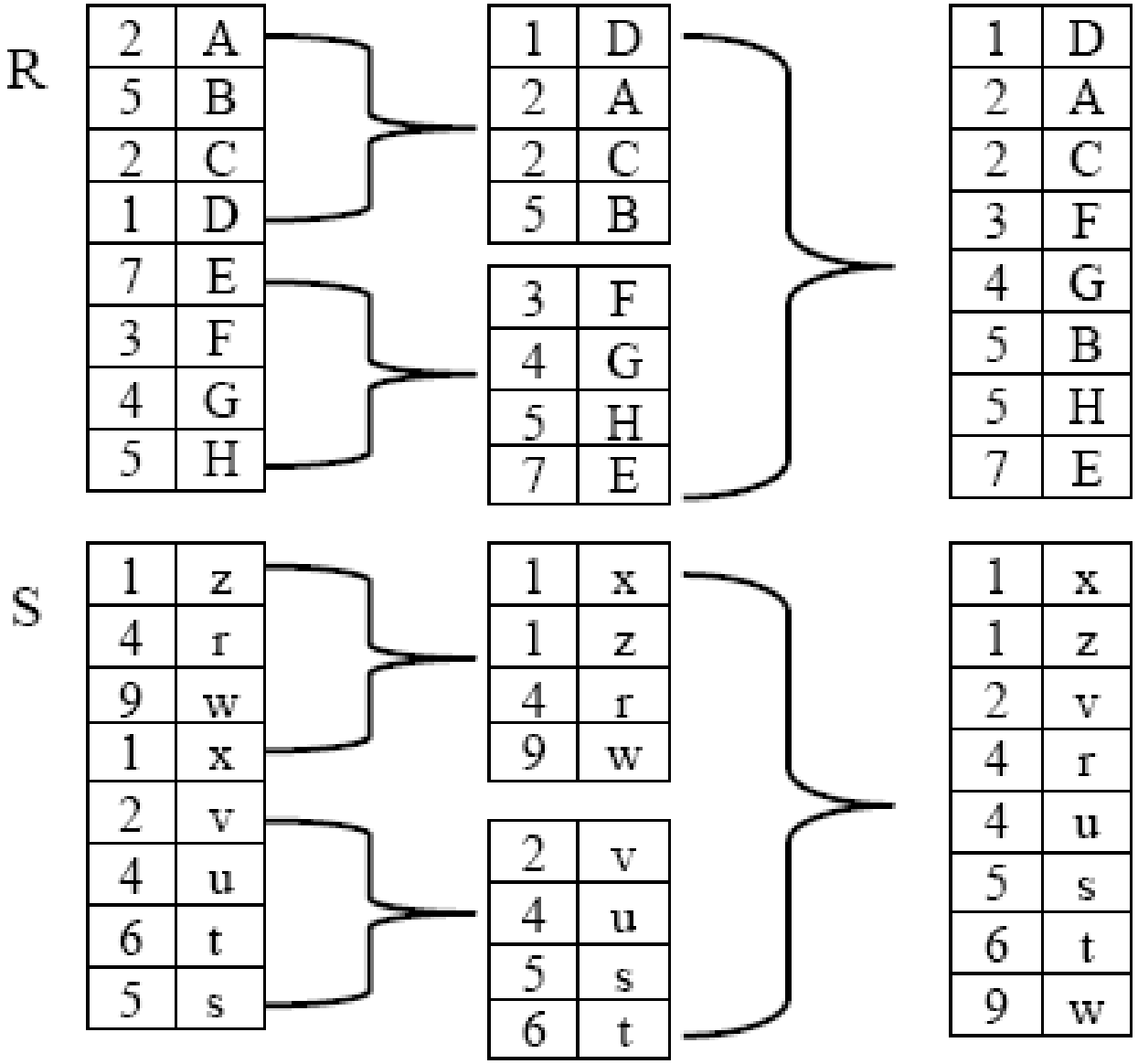


算法的磁盘存取块数：

$O(B(R)\log B(R) + B(S)\log B(S)+B(R)+B(S))$

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

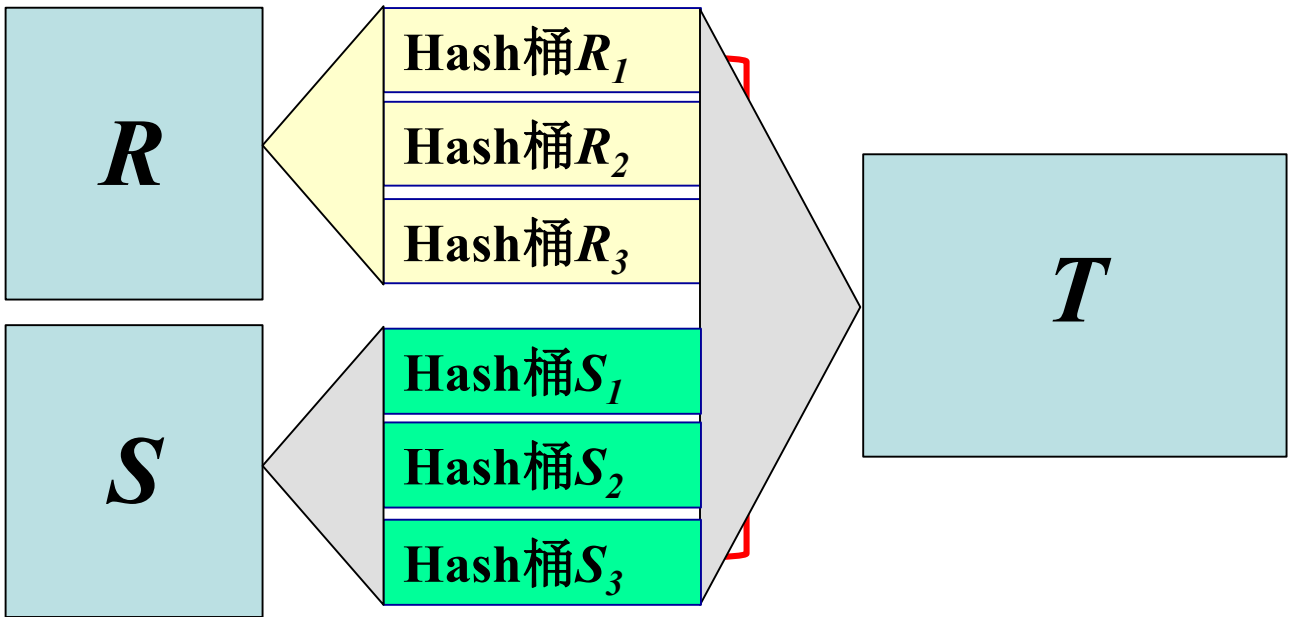
Output

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[illegible]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。

问题：为什么R_1不与S_2做连接？



Hash-Join算法

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

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：　　$2(B(R) + B(S))$

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
7	E

4	G

h(x) = 2

2	A
5	B

2	C
5	H

Partitions for S

h(x) = 0

9	w
6	t

h(x) = 1

1	z
4	r

1	x
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
7	E

4	G

Partition 1 for S

h(x) = 1

1	z
4	r

1	x
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.

- 选择操作算法
- 投影操作算法
- 连接操作算法
- **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。

Outline

- 关系代数操作算法
- 查询优化

代数定律

- 交换律与结合律

The relational algebra operators of cross-product (×), join (⋈), set and bag union ($\cup_S$ and $\cup_B$), and set and bag intersection ($\cap_S$ and $\cap_B$) are all associative and commutative.

Commutative	Associative
$R \times S = S \times R$	$(R \times S) \times T = R \times (S \times T)$
$R \bowtie S = S \bowtie R$	$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
$R \cup S = S \cup R$	$(R \cup S) \cup T = R \cup (S \cup T)$
$R \cap S = S \cap R$	$(R \cap S) \cap T = R \cap (S \cap T)$

涉及选择的定律

- 分解律
  - $\sigma_{C1 \text{ and } C2}(R) = \sigma_{C1}(\sigma_{C2}(R))$
  - $\sigma_{C1 \text{ or } C2}(R) = \sigma_{C1}(R) \cup \sigma_{C2}(R)$　　只有在R为集合时成立。
  - $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$

例如，令R(a,b,c)是一关系，则$\sigma_{(a=1 \text{ or } a=3) \text{ AND } b<c}(R)$可分解为：$\sigma_{(a=1 \text{ or } a=3)}(\sigma_{b<c}(R))$，或

$\sigma_{a=1}(\sigma_{b<c}(R)) \cup \sigma_{a=3}(\sigma_{b<c}(R))$

涉及选择的定律

- 对二元操作符进行下推选择
  - $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
  - $\sigma_C(R - S) = \sigma_C(R) - S$
  - $\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S)$
  - $\sigma_C(R \times S) = \sigma_C(R) \times S$
  - $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$
  - $\sigma_C(R \cap S) = \sigma_C(R) \cap S$
  - $\sigma_C(R \times S) = S \times \sigma_C(S)$
  - $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie \sigma_C(S)$

– 下推选择

- 当查询中涉及视图时，某些情况下：
  - 首先将选择操作尽可能往树的上部移动是很重要的
  - 然后再把选择下推到所有可能的分枝

例如：

StarsIn(title, year, starName)

Movie(title, year, length, inColor, studioName)

CREATE VIEW MoviesOF1996 AS

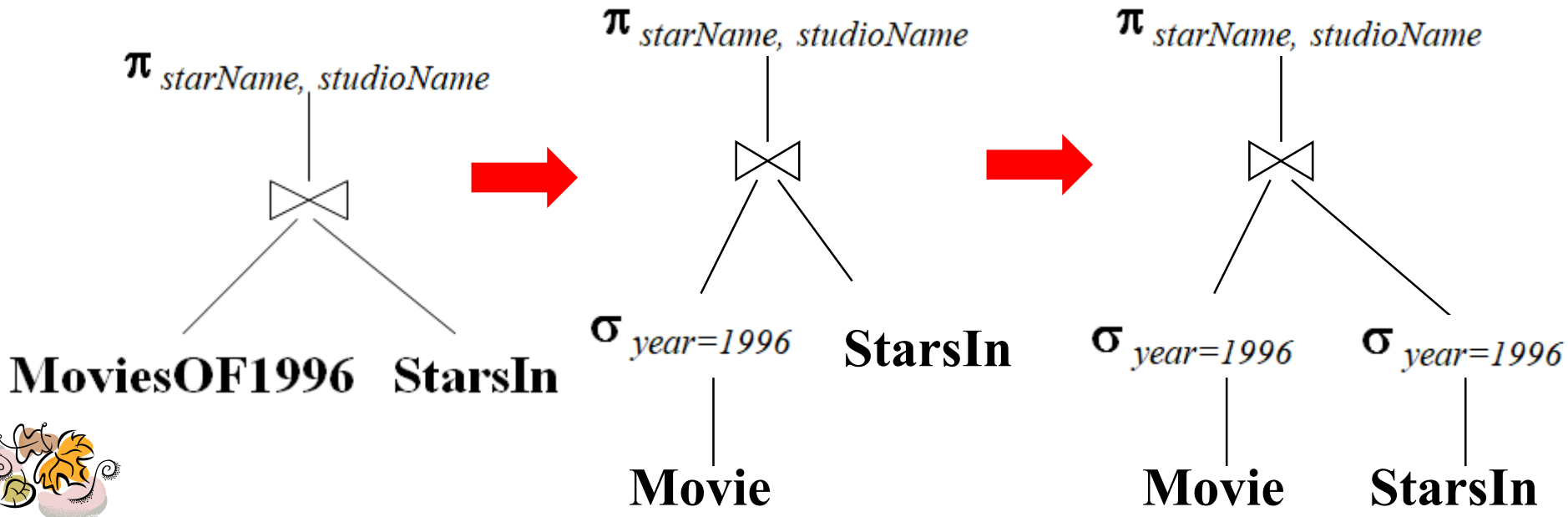
Select * From Movie

Where year = 1996

SQL查询：“在1996年有哪些影星为哪些电影制作公司工作”

Select starName studioName

From MoviesOF1996, StarsIn



涉及投影的定律

- 投影可以像选择一样下推到多个其他操作符中；
- 投影不改变元组数，只缩短元组的长度；
- 有时候投影实际上增加了元组的长度；

$$\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))$$

$$\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))$$

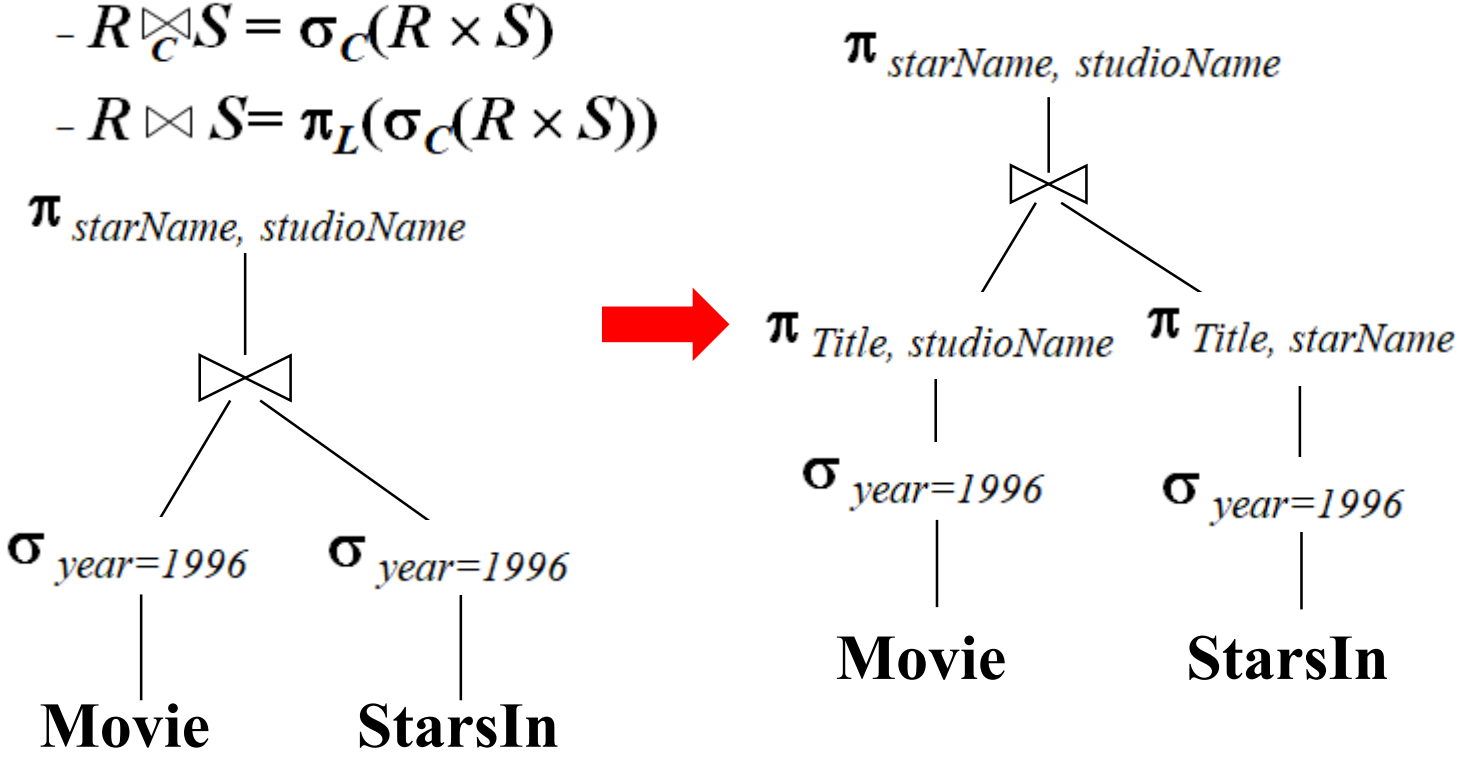
$$\pi_L(R \cup_B S) = \pi_L(R) \cup_B \pi_L(S)$$

$$\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$$

有关连接与笛卡尔积的定律

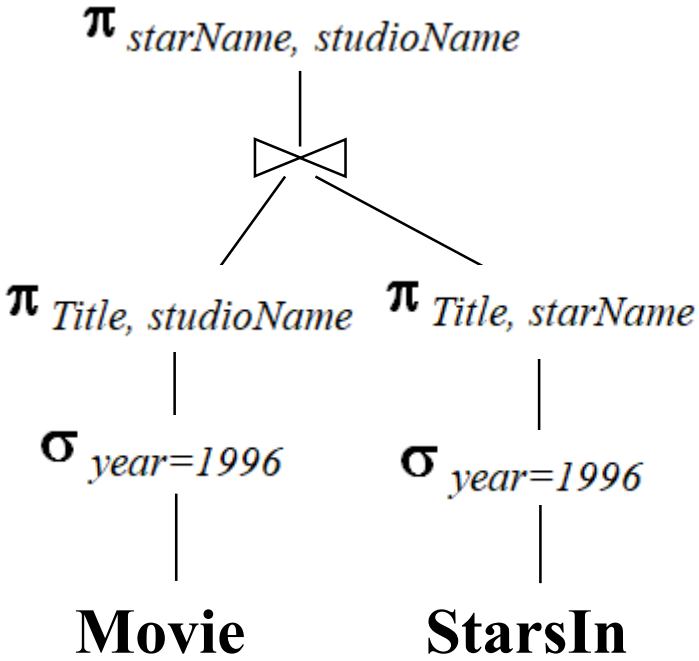
- Join操作满足交换律和结合律；
- 选择可以下推到join操作符之下；
- 可以将投影下推到join之前；

- $R \bowtie_C S = \sigma_C(R \times S)$
- $R \bowtie S = \pi_L(\sigma_C(R \times S))$



启发式优化的原则

- 尽可能地将选择条件下推，尽早执行选择，使得过滤后的中间结果尽可能地小；
- 用join替换笛卡尔积
- 尽可能下推投影，在适当的位置增加投影操作
- 利用pipeline，减少查询计划的执行时间



操作代价的估计

- 逻辑计划转换成物理计划
  - 由逻辑计划可派生出多个不同物理计划，
  - 对每个物理计划进行评价，或估计实现这个转换的代价。(称为基于代价的枚举)
  - 从中选择具有最小估计代价的物理查询计划。

如何对代价进行估计？

操作代价的估计

- 中间结果关系大小的估计
  - 用于处理中间关系的磁盘I/O数是描述查询计划代价大小的一个函数
  - 由于未经计算，一般难以准确地获得中间关系的元组数。因此，只能通过一些原则，对中间结果关系的大小进行尽可能准确地估计。
  - 符号
    - V(R, Y)：表示关系R在属性Y上的值域大小

连接大小的估计

- 只考虑自然连接R(X,Y)⋈S(Y,Z)
- 两个假设条件
  - 值集的包含: V(R,Y)≤ V(S,Y)
    - R的每个Y值将是S的一个Y值
  - 值集的保持：如果A是R的一个属性但不是S的属性，则V(R⋈S, A)=V(R,A)
    - R与S的连接次序并不重要
- 例如，S为主键表，R为外键表时，满足值集包含和值集保持的约束

连接大小的估计

- 令V(R,Y)≤V(S,Y)
- R的每个元组*t*有1/ V(S,Y)概率与S中的一个元组进行连接
- 因S中有T(S)元组，与*t*连接的期望元组数为T(S)/V(S,Y)
- 由于R有T(R)个元组，R⋈S的估计大小为

  T(R)T(S)/V(S,Y)
- 一般地，有T(R)T(S)/MAX(V(R,Y), V(S,Y))

T(R)T(S)/MAX(V(R,Y), V(S,Y))

- 假定有关系$R(a,b)$, $S(b,c)$
- V(R,b)=V(S,b)=13
- T(R)=950, T(S)=500

连接$R(a,b) \bowtie S(b,c)$

结果大小的估计是：

T(R)T(S)/MAX(V(R,Y), V(S,Y))

=950×500/13≈36539

b	0	1	2	3	4	5	6	7	8	9	10	11	12
R	150	200	50	50	50	100	50	50	50	50	50	50	50
S	100	80	70	25	25	25	25	25	25	25	25	25	25

- 假定有关系R、S
- V(R,b)=V(S,b)=13
- *R*、*S*在*b*属性上的取值分布分别为：
  - 1: 200, 0: 150, 5: 100, 其它值：500
  - 0: 100, 1: 80, 2: 70 , 其它值：250

连接$R(a,b) \bowtie S(b,c)$

结果大小的估计是：

$150\times100+200\times80+50\times70+100\times25+9\times50\times25$

$=15000+16000+3500+2500+9\times1250$

$=48250$

$R(a,b)$	$S(b,c)$	$U(c,d)$
$T(R)=1000$	$T(S)=2000$	$T(U)=5000$
$V(R,b)=20$	$V(S,b)=50$	
	$V(S,c)=100$	$V(U,c)=500$

计算自然连接：R⋈S⋈U

(R⋈S)⋈U

首先计算：T(R⋈S)=T(R)×T(S)/max(V(R,b), V(S,b))

= 1000×2000 / 50 =40000

得到中间结果关系*P*(*a*,*b*,*c*)：

T(*P*)=40000, V(P, c)=V(S,c)=100

R⋈(S⋈U)?

P与U连接，得到最终结果大小：

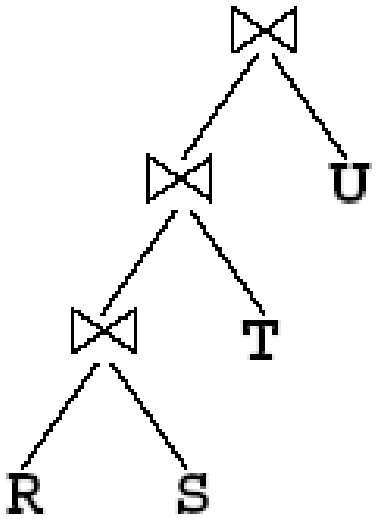
T(*P*⋈*U*)=T(P)×T(U)/max(V(P,c), V(U,c))

= 40000×5000 / 500 =400000

连接顺序的选择

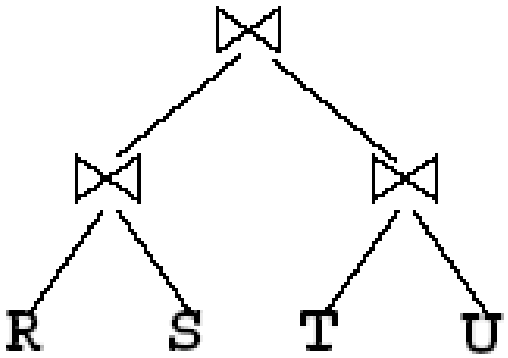
- 连接树

Left-Deep Join Tree



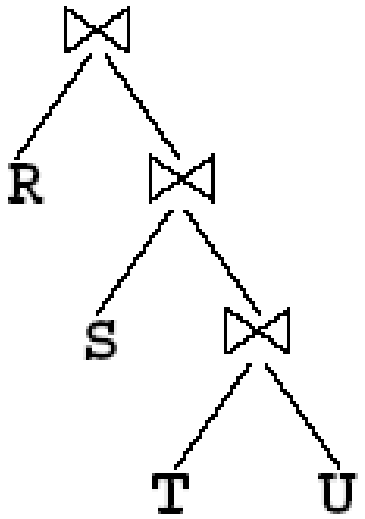
左深连接树

Balanced Join Tree



平衡树

Right-Deep Join Tree



右深连接树

连接顺序的选择

- 基于动态规划的连接顺序选择算法
  - 输入：$R_1,R_2,\ldots\ldots,R_n$
  - 输出：具有最小I/O代价的连接顺序

$$R_1,R_2,\ldots\ldots,R_k\ ,R_{k+1},\ldots\ldots,R_n \qquad 1\le k\le n-1$$

建立代价的递归方程：

$$M[i,j]=\min_{i\le k\le j-1}\{M[i,k]+M[k+1,j]+Cost(R_{1\sim k}\bowtie R_{k+1\sim j})\}$$

考虑四个关系R、S、T和U的连接，且每个关系有1000个元组。相应的估计值如下：

$R(a,b)$	$S(b,c)$	$T(c,d)$	$U(d,a)$
$V(R,a)=100$			$V(U,a)=50$
$V(R,b)=200$	$V(S,b)=100$		
	$V(S,c)=500$	$V(T,c)=20$	
		$V(T,d)=50$	$V(U,d)=1000$

1. 首先以单个表作为入口，建立如下表格：

	$\{R\}$	$\{S\}$	$\{T\}$	$\{U\}$
大小	1000	1000	1000	1000
代价	0	0	0	0
最佳计划	R	S	T	U

其中，

大小：表示运算结果的大小

代价：表示产生结果的代价

最佳计划：产生结果的计划

考虑四个关系R、S、T和U的连接，且每个关系有1000个元组。相应的估计值如下：

$R(a,b)$	$S(b,c)$	$T(c,d)$	$U(d,a)$
$V(R,a)=100$			$V(U,a)=50$
$V(R,b)=200$	$V(S,b)=100$		
	$V(S,c)=500$	$V(T,c)=20$	
		$V(T,d)=50$	$V(U,d)=1000$

2. 两个关系的情况

枚举所有可能的两个关系的连接，并计算相应代价：

	$\{R,S\}$	$\{R,T\}$	$\{R,U\}$	$\{S,T\}$	$\{S,U\}$	$\{T,U\}$
大小	5000	1M	10,000	2000	1M	1000
代价	0	0	0	0	0	0
最佳计划	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

$R(a,b)$	$S(b,c)$	$T(c,d)$	$U(d,a)$
$V(R,a)=100$			$V(U,a)=50$
$V(R,b)=200$	$V(S,b)=100$		
	$V(S,c)=500$	$V(T,c)=20$	
		$V(T,d)=50$	$V(U,d)=1000$

	$\{R,S\}$	$\{R,T\}$	$\{R,U\}$	$\{S,T\}$	$\{S,U\}$	$\{T,U\}$
大小	5000	1M	10,000	2000	1M	1000
代价	0	0	0	0	0	0
最佳计划	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

3. 三个关系的情况

枚举所有可能的三个关系的连接，并计算相应代价：

	$\{R,S,T\}$	$\{R,S,U\}$	$\{R,T,U\}$	$\{S,T,U\}$
大小	10,000	50,000	10,000	2,000
代价	2,000	5,000	1,000	1,000
最佳计划	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$

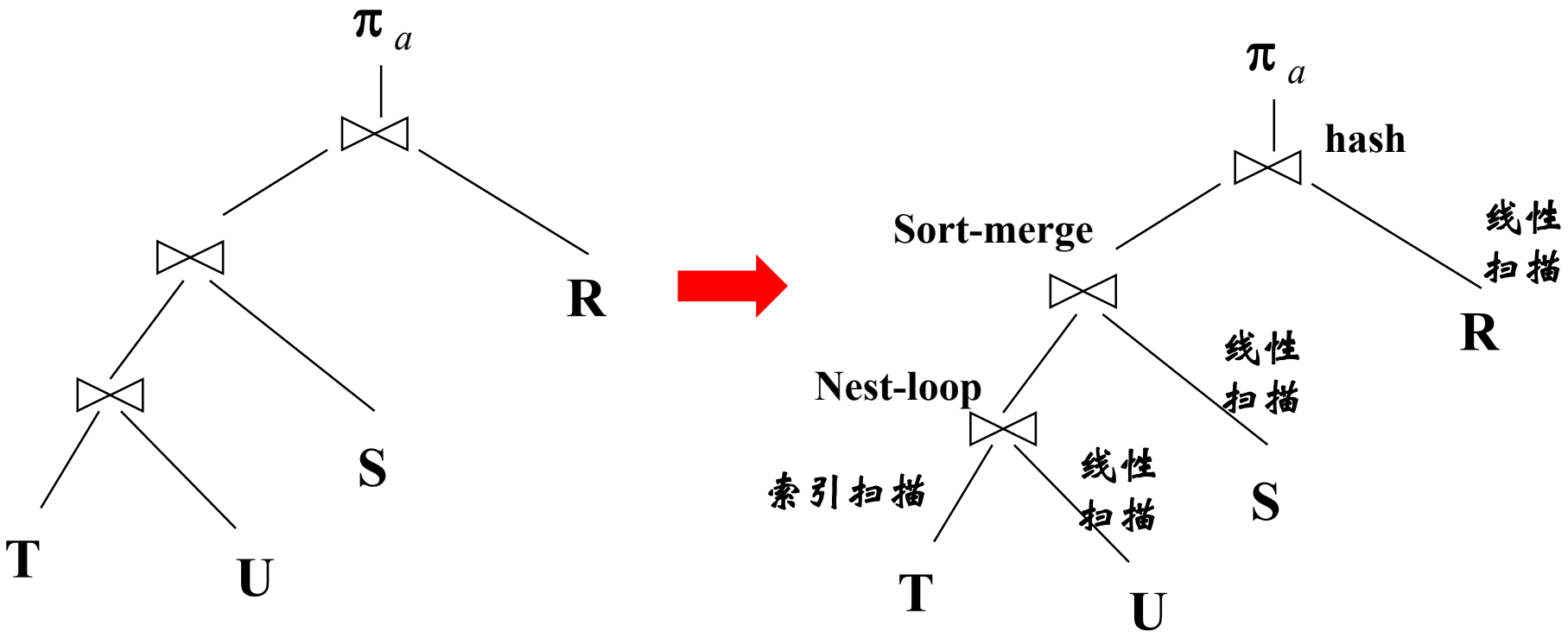
4. 四个关系的情况

(1). 以可能的最佳方法选择三个关系进行连接，然后与第四个连接

(2). 将四个关系划分为两对，将每一对进行连接，再将两个结果连接。

分 组	代 价
$((S \bowtie T) \bowtie R) \bowtie U$	12 000
$((R \bowtie S) \bowtie U) \bowtie T$	55 000
$((T \bowtie U) \bowtie R) \bowtie S$	11 000
$((T \bowtie U) \bowtie S) \bowtie R$	3 000
$(T \bowtie U) \bowtie (R \bowtie S)$	6 000
$(R \bowtie T) \bowtie (S \bowtie U)$	2 000 000
$(S \bowtie T) \bowtie (R \bowtie U)$	12 000

$R(a,b)$	$S(b,c)$	$T(c,d)$	$U(d,a)$
$V(R,a)=100$			$V(U,a)=50$
$V(R,b)=200$	$V(S,b)=100$		
	$V(S,c)=500$	$V(T,c)=20$	
		$V(T,d)=50$	$V(U,d)=1000$



Student(Sno, Sname,Sage, Sdept)

SC(Sno, Cno, Grade)

[例] 求选修了2号课程的学生姓名。

```
SELECT  Student.Sname
FROM  Student，SC
WHERE  Student.Sno=SC.Sno AND
       SC.Cno='2'；
```

其中：

假定Student中有1000个学生记录，SC中有10000个选课记录

其中选修2号课程的选课记录为50个

有多种等价的关系代数表达式来完成这一查询：

$Q1=\pi_{Sname}(\sigma_{Sc.Cno='2'}(Student \bowtie SC))$

$Q2=\pi_{Sname}(Student \bowtie (\sigma_{Sc.Cno='2'}(SC)))$

逻辑查询计划树？

$Q3=\pi_{Sname}((\pi_{Sname, Sno}Student) \bowtie (\pi_{Sno, Cno}\sigma_{Sc.Cno='2'}(SC)))$

Student(Sno, Sname,Sage, Sdept)

SC(Sno, Cno, Grade)

假定Student中有1000个学生记录，SC中有10000个选课记录其中选修2号课程的选课记录为50个

对于查询计划$Q1$：$\pi_{Sname}(\sigma_{Sc.Cno='2'}(Student \bowtie SC))$

1. 计算自然连接(Nest-Loop)

设一个块能装10个Student元组或100个SC元组，在内存中存放5块Student元组和1块SC元组，则读取总块数为：

$$\frac{1000}{10}+\frac{1000}{10\times 5}\times\frac{10000}{100}=100+20\times 100=2100\text{块}$$

2. 计算自然连接(Sort-Merge)

Student关系默认在主键Sno上有序，则连接操作读取总块数：

$$\frac{10000}{100}\times\log_5\frac{10000}{100}+\frac{1000}{10}+\frac{10000}{100}\approx 300+100+100=500\text{块}$$

Student(Sno, Sname, Sage, Sdept)

SC(Sno, Cno, Grade)

假定Student中有1000个学生记录，SC中有10000个选课记录，其中选修2号课程的选课记录为50个　　SC在Cno属性上无索引

对于查询计划$Q2$：$\pi_{Sname}(\text{Student} \bowtie (\sigma_{Sc.Cno='2'}(SC)))$

1. 计算自然连接(Nest-Loop)

顺序扫描SC，利用选择条件过滤SC，得到中间结果关系T_SC，由于T_SC只有50个记录，则直接保存在内存缓冲区中，不必写到磁盘。

此时T_SC为小关系，连接操作读取总块数为：

$$\frac{10000}{100}(\text{选择代价})+1\times\frac{1000}{10}(\text{连接代价})=100+100=200\text{块}$$

2. 计算自然连接(Sort-Merge)

Student关系默认在主键Sno上有序，则连接操作读取总块数：

$$\frac{10000}{100}(\text{选择代价})+0(\text{排序代价})+1\times\frac{1000}{10}(\text{连接代价})=100+100=200\text{块}$$

若SC在Cno属性上有索引？

Student(Sno, Sname, Sage, Sdept)

SC(Sno, Cno, Grade)

假定Student中有1000个学生记录，SC中有10000个选课记录，其中选修2号课程的选课记录为50个　　SC在Cno属性上有索引

对于查询计划$Q2$：$\pi_{Sname}(\text{Student} \bowtie (\sigma_{Sc.Cno='2'}(SC)))$

1. 计算自然连接(Nest-Loop)

若SC在Cno上存在索引，则经过选择后的中间结果T_SC大小为50个记录，仍然设一个块能装10个Student元组或100个SC元组，此时T_SC为小关系，占一个磁盘块。连接操作读取总块数为：

$$1+1\times\frac{1000}{10}=1+100=101\text{块}$$

2. 计算自然连接(Sort-Merge)

Student关系默认在主键Sno上有序，则连接操作读取总块数：

$$1+\frac{1000}{10}=101\text{块}$$

Student(Sno, Sname, Sage, Sdept)

SC(Sno, Cno, Grade)

假定Student中有1000个学生记录，SC中有10000个选课记录其中选修2号课程的选课记录为50个

对于查询计划Q3：$\pi_{Sname}((\pi_{Sname, Sno}Student) \bowtie (\pi_{Sno, Cno}\sigma_{Sc.Cno='2'}(SC)))$

1. 计算自然连接(Nest-Loop)

连接操作读取总块数为？

若SC在Cno属性上有/无索引？

2. 计算自然连接(Sort-Merge)

连接操作读取总块数？

Now let's go to

Next Chapter