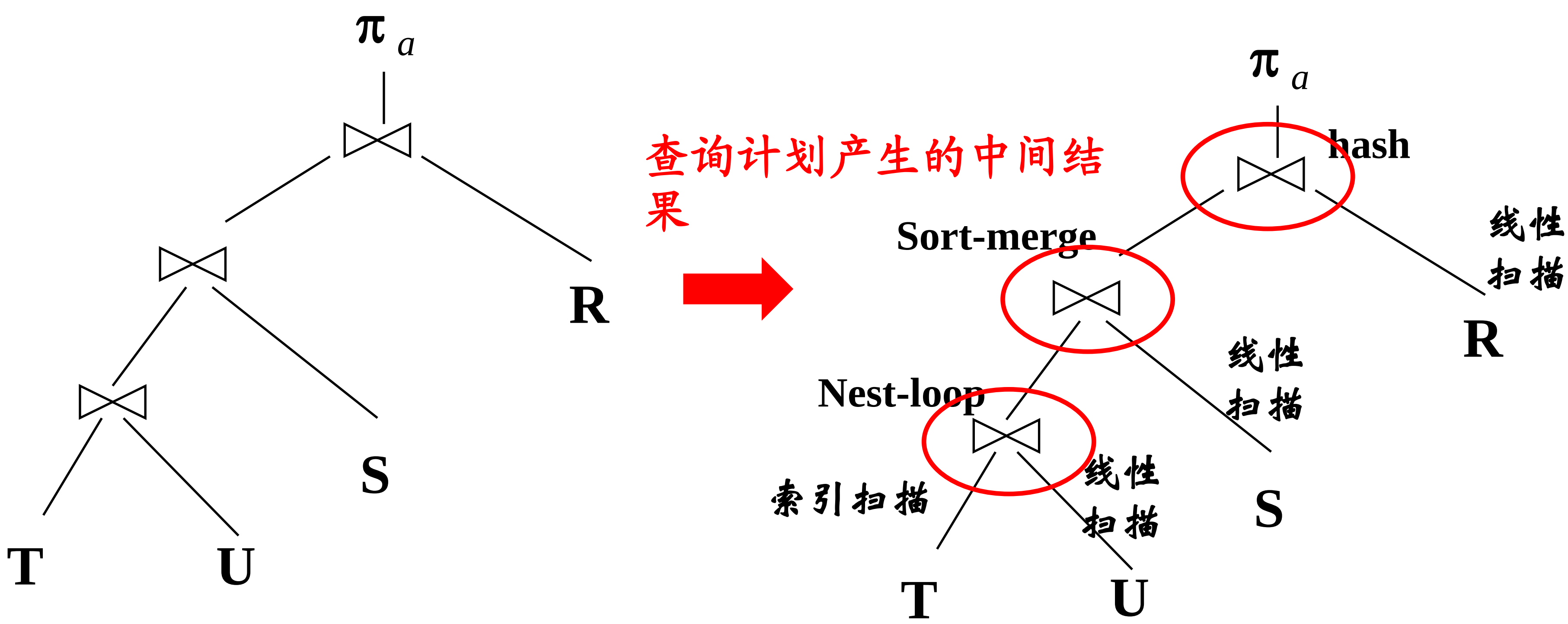
6.2 查询优化

- 查询计划执行方式
- 逻辑查询优化
- 物理查询优化

中间结果如何送达下一个操作符？
存储起来由下一个操作符来读取？
随着结果的生成直接传递给下一个操作符？

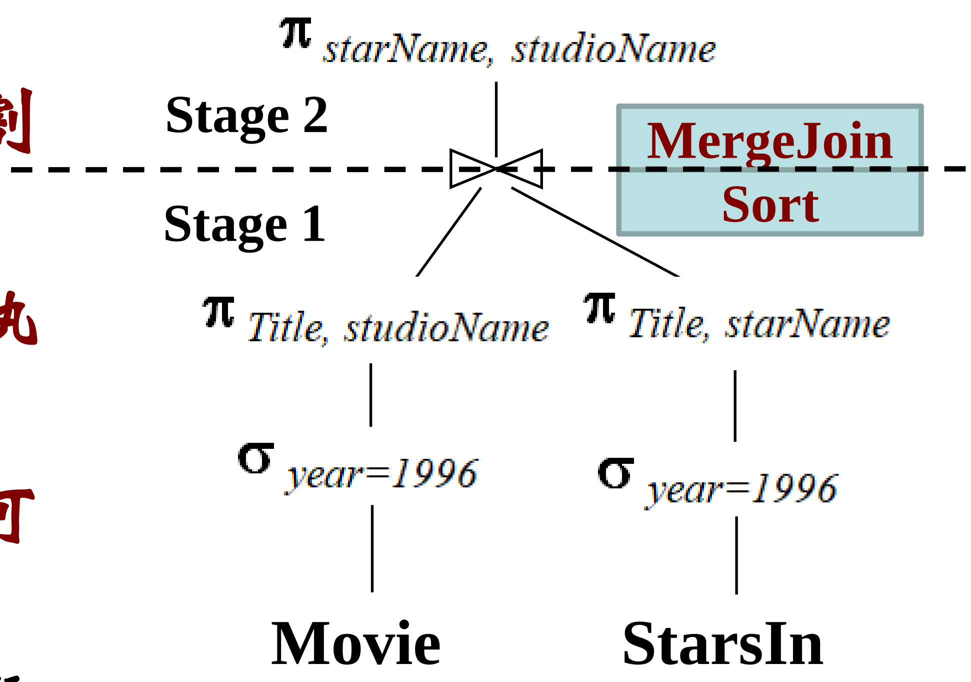


查询执行过程中的物化与流水化

- 流水化
  - 中间结果不被存储，直接传递给下一个操作符作为输入，多个操作符可以并行执行
    - 需求驱动流水化
    - 生产者驱动流水化
    - 流水化的操作符代价中需去除读取输入部分
- 物化
  - 中间结果被存储，下一个操作符需要读取中间结果作为输入
    - 有些中间结果必须物化，例如SortMergeJoin中需要对中间结果进行排序

流水化中的不同阶段

- 指定物理操作算法后，划分流水化中的阶段
  - 用物化的中间结果分割物理执行计划
  - 执行完一个阶段，再执行下一个阶段
  - 同一个阶段中的操作可以流水化执行
    - 利用各自的缓冲区并发执行
    - 上一个操作符的输出直接给下一个操作符作为输入



需求驱动流水化

- **需求驱动流水化**
  - **每个操作符实现Iterator接口，提供open(), next()和close()**
    - **Open()开始操作**
    - **Next()获取下一个结果**
    - **Close()结束操作符的工作**

顺序扫描 $\sigma_{A=a}(R)$

Open():打开文件

Next(): 继续扫描直至发现下一个结果

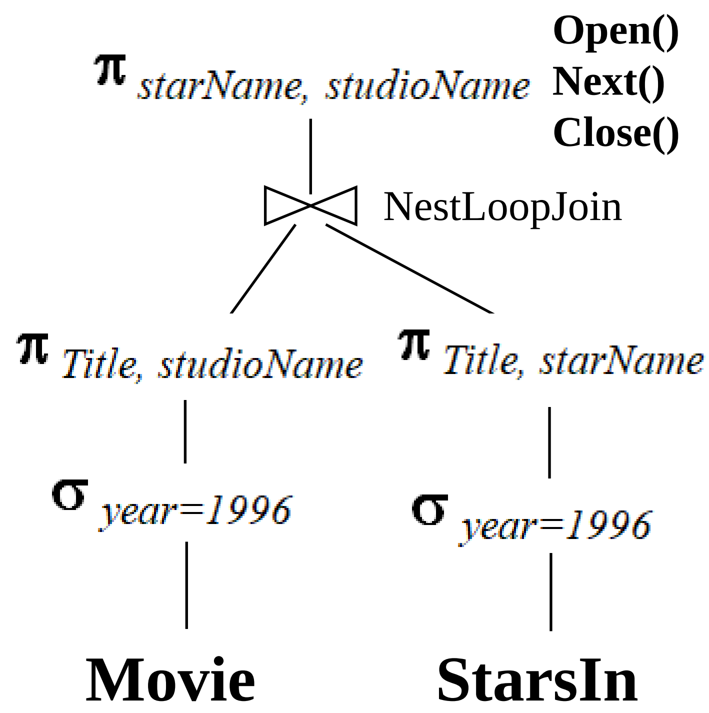
Close(): 扫描完毕时关闭文件

SortMergeJoin(*R*, *S*)

Open():打开文件，可能排序

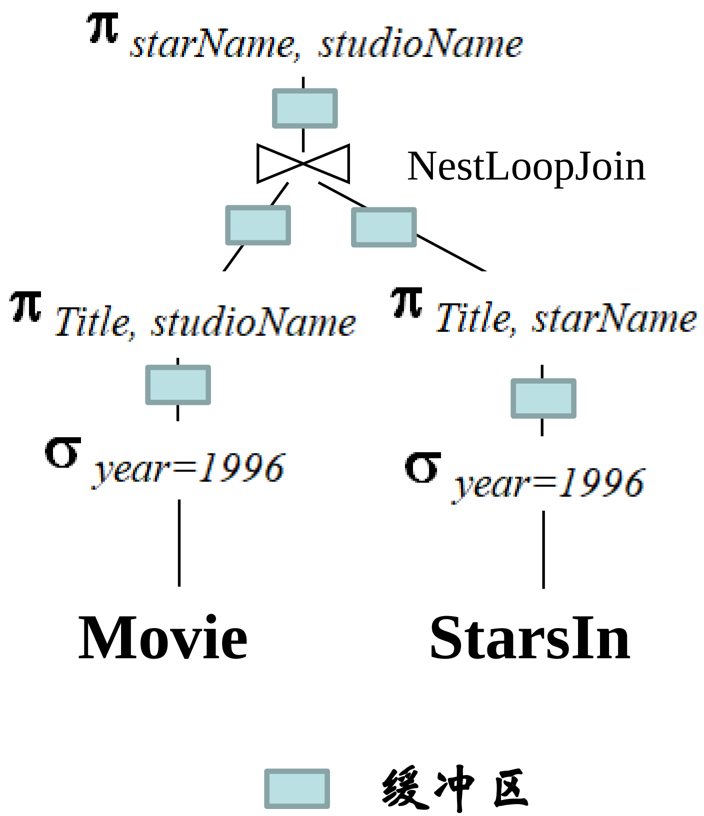
Next(): 继续扫描R和S直至发现下一个连接结果

Close(): 扫描完毕时关闭文件



上层操作符每次调用next()会持续调用下层操作符的next()直至有下一个结果产生

- 生产者驱动流水化
  - 操作符之间有缓冲区
    - 生产者操作符的输出进入缓冲区，缓冲区满后暂停执行
    - 消耗者操作符从缓冲区取出中间结果，缓冲区空后，生产者操作符继续执行



- 查询计划执行方式
- 逻辑查询优化
  - 逻辑查询计划枚举
  - 基数估计
  - 连接顺序选择
- 物理查询优化

代数定律

- 交换律与结合律
  - 交换律
    - $A \otimes B = B \otimes A$, 则“$\otimes$”满足交换律
  - 结合律
    - $A \otimes (B \otimes C) = (A \otimes B) \otimes C$, 则“$\otimes$”满足结合律

代数定律

- 交换律与结合律
  - 关系代数操作中笛卡尔积、连接、集合和包的并、交操作都满足交换律和结合律

交换律	结合律
$R \times S = S \times R$	$(R \times S) \times T = R \times (S \times T)$
$R \bowtie S = S \bowtie R$	$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
$R \cup S = S \cup R$	$(R \cup S) \cup T = R \cup (S \cup T)$
$R \cap S = S \cap R$	$(R \cap S) \cap T = R \cap (S \cap T)$

– 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
    - $\sigma_C(R \cup S) = \sigma_C(R) \cup \ \sigma_C(S)$
    - $\sigma_C(R - S) = \sigma_C(R) - S$
    - $\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S)$
    - $\sigma_C(R \times S) = \sigma_C(R) \times S$
    - $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$
    - $\sigma_C(R \cap S) = \sigma_C(R) \cap S$

    （以上三式：R具有C中提及的全部属性）
    - $\sigma_C(R \times S) = R \times \sigma_C(S)$ C只涉及S中的属性，$\bowtie$和$\cap$同样适用
    - $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie \sigma_C(S)$ R和S都包含C中的全部属性

- 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：

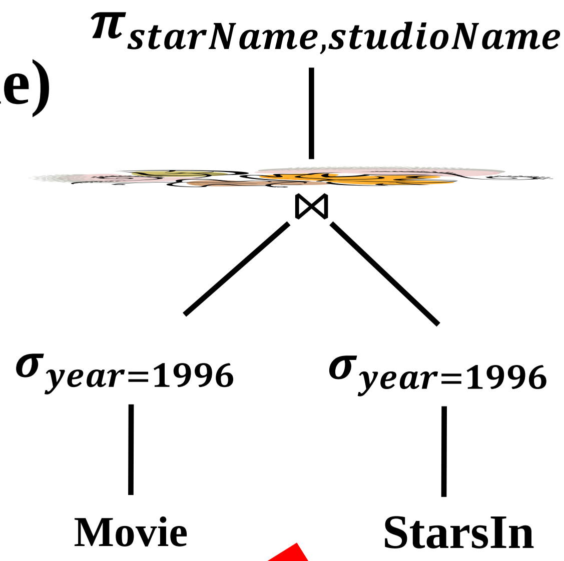
StarsIn(title, year, starName)

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

CREATE VIEW MoviesOF1996 AS

Select * From Movie

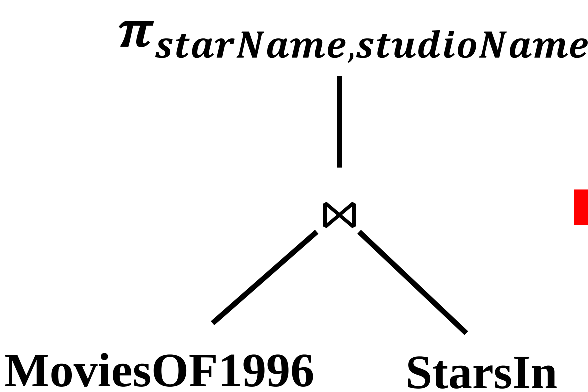
Where year = 1996



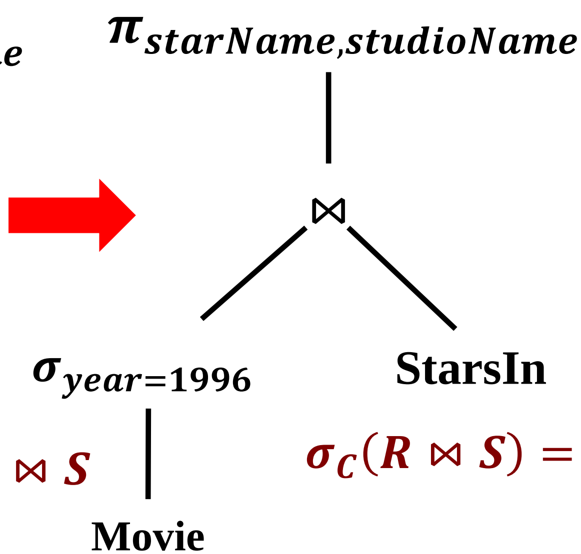
SQL查询：“在1996年有哪些影星为哪些电影制作公司工作”

Select starName, studioName

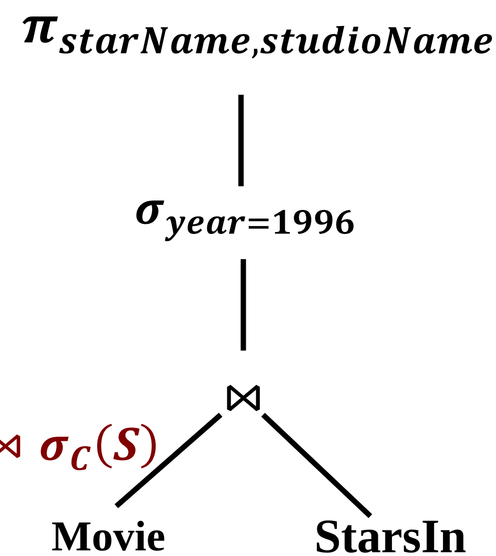
From MoviesOF1996 Natural Join StarsIn;

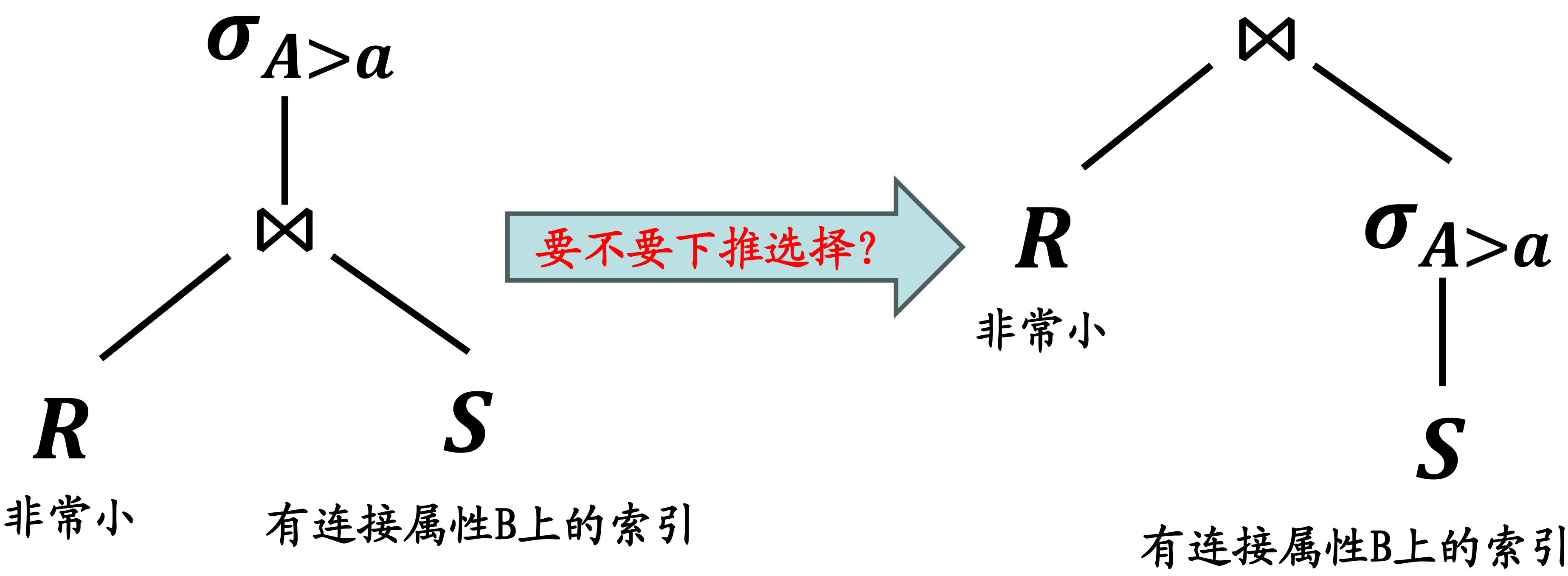


$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$



$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie \sigma_C(S)$




$\sigma_{A>a}$
$\bowtie$
R
非常小
S
有连接属性B上的索引
要不要下推选择?
$\bowtie$
R
非常小
$\sigma_{A>a}$
S
有连接属性B上的索引

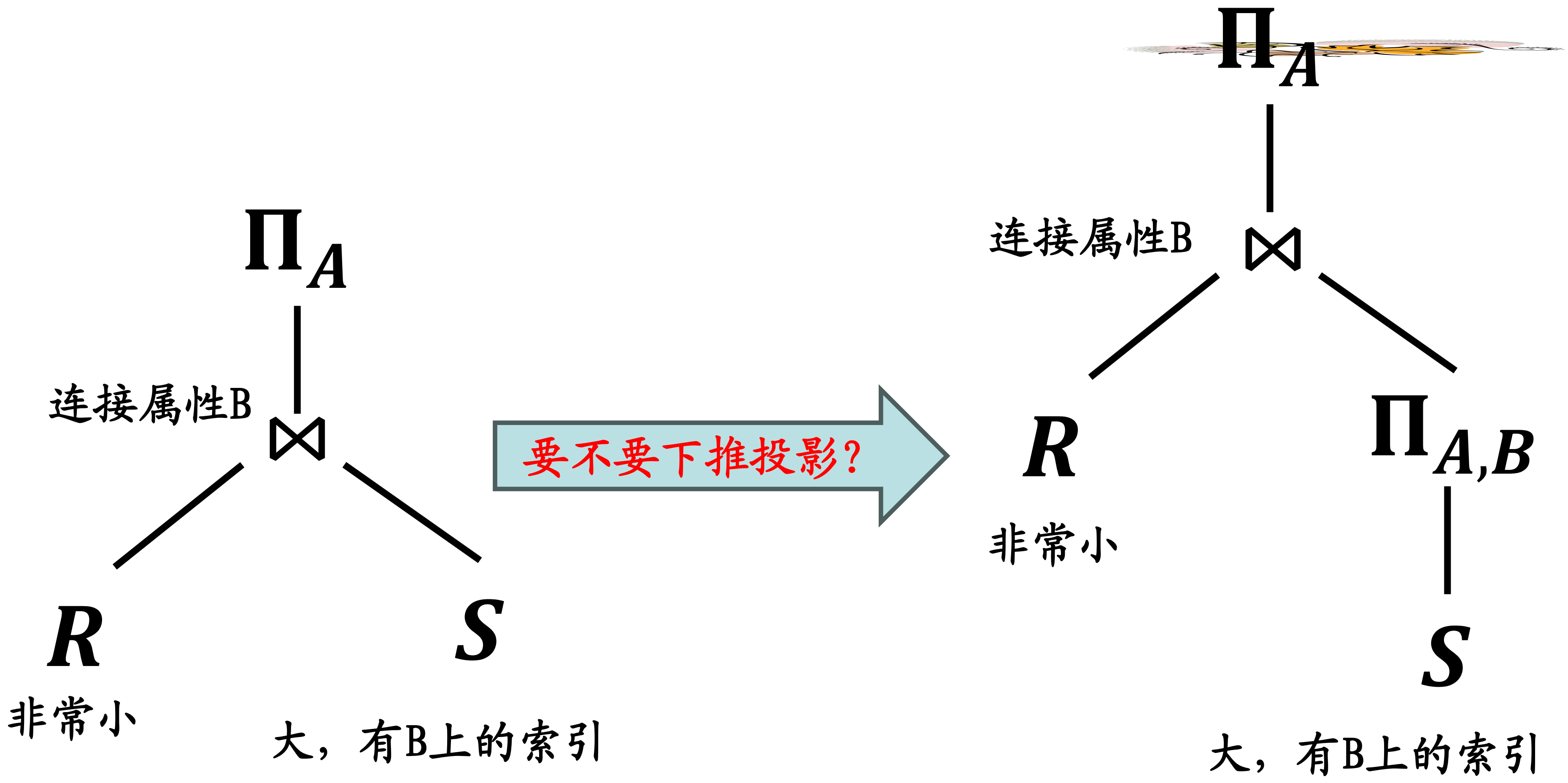
- 涉及投影的定律
  - 投影可以像选择一样下推到多个其他操作符中；
  - 投影不改变元组数，只缩短元组的长度；
  - 有时候投影实际上增加了元组的长度(广义投影)；

$\pi_L(R \bowtie S) = \pi_L\big(\pi_M(R) \bowtie \pi_N(S)\big)$，$M$和$N$包含连接属性以及$R$和$S$中的$L$的输入属性

$\pi_L(R \times S) = \pi_L\big(\pi_M(R) \times \pi_N(S)\big)$，$M$和$N$分别包含$L$中存在于$R$和$S$的全部属性

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

$\pi_L\big(\sigma_C(R)\big) = \pi_L\Big(\sigma_C\big(\pi_M(R)\big)\Big)$，$M$包含$L$和$C$中提及的属性

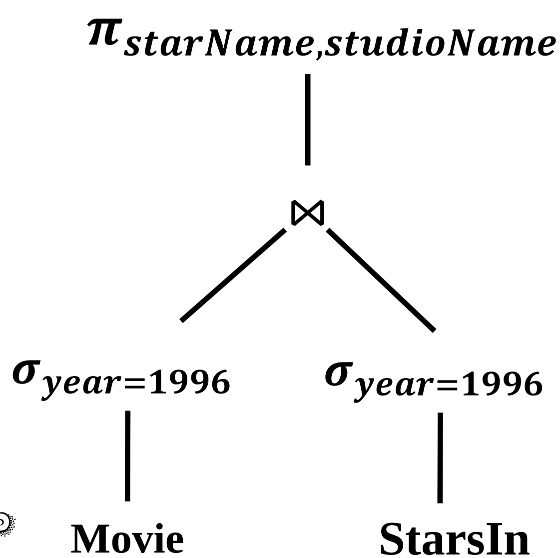

Π_A
连接属性B
$\bowtie$
R
非常小
S
大，有B上的索引
要不要下推投影?
Π_A
连接属性B
$\bowtie$
R
非常小
$\Pi_{A,B}$
S
大，有B上的索引

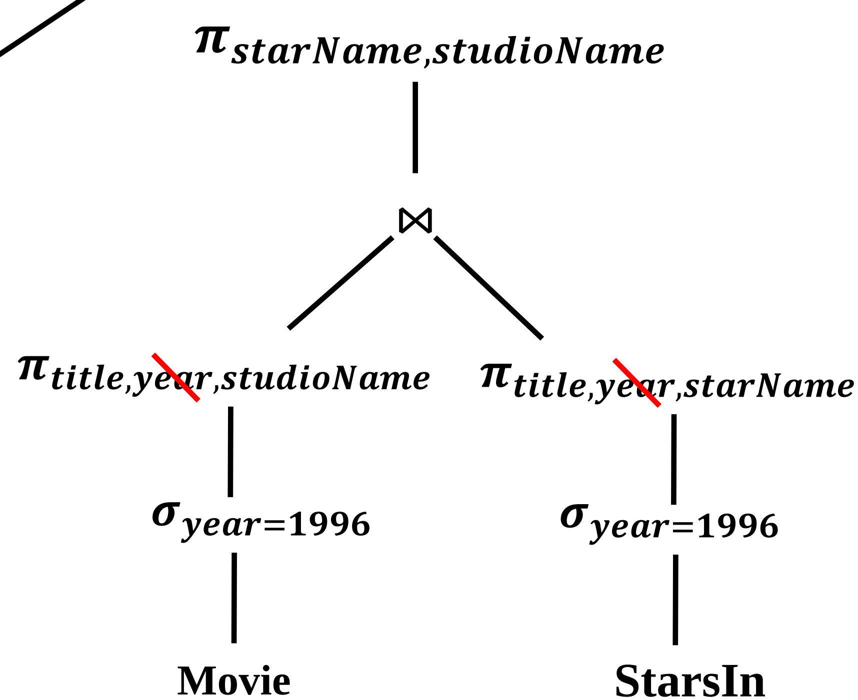
StarsIn(title, year, starName)

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

- 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；
    - $R \underset{C}{\bowtie} S = \sigma_C(R \times S)$
    - $R \bowtie S = \pi_L(\sigma_C(R \times S))$

C是R和S中同名属性对的等值比较，L包含R与S中每一个等值对中的一个属性以及其它所有属性





- 优化器能够枚举所有逻辑查询计划(LQP)
  - 从初始的LQP *E*开始，利用等价规则修改已有的LQP，得到与*E*等价的LQP *E*’

GenerateAllEquivalent(*E*, *RuleSet*)

Input: (1) *E*: 关系代数表达式；(2)*RuleSet*: 等价规则集合

Output: *EQ*: 与*E*等价的关系代数表达式集合

$EQ=\{E\}$

repeat

  for each E_i **in** *EQ* **and each** R_j **in** *RuleSet*

    if E_i 中存在子表达式e_i, e_i出现在*Rj*的一侧 // 可以发动R_j

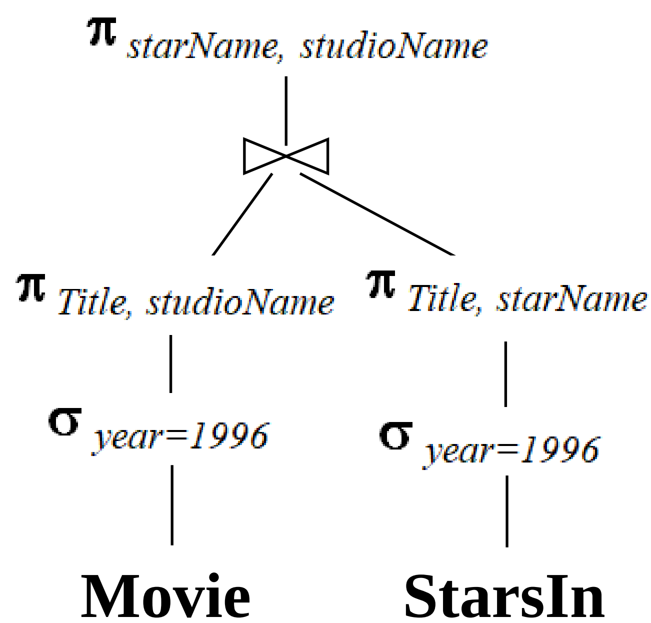
      用*Rj*修改E_i得到*E*’

      EQ 中加入*E*’

until 没有新的表达式可以加入*EQ*

与*E*等价的LQP数量过于庞大！

- 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
- 基于代价的逻辑查询优化
  - 中间结果的大小作为代价



- 查询计划执行方式
- 逻辑查询优化
  - 逻辑查询计划枚举
  - 基数估计
  - 连接顺序选择
- 物理查询优化

操作代价的估计

- 中间结果关系大小的估计
  - 逻辑查询计划中，中间结果的大小是评价逻辑查询计划的一个度量
  - 在物理查询计划方面，也需要使用中间结果的大小，用于估计每个执行算法的代价
  - 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。

操作代价的估计

- 中间结果关系大小的估计
  - 中间结果估计需要满足以下要求
    - 尽量准确
    - 易于计算
    - 逻辑一致：随关系大小递增、与操作顺序无关
  - 估计过程中的假设：
    - 属性值均匀分布、属性彼此不相关
  - 用于估计的统计信息
    - V(R, Y)：表示关系R在属性Y上的值域大小
    - T(R)：表示关系R中元组的数量
    - 关系属性值上的直方图

选择操作结果大小的估计

- 等值条件$S=\sigma_{A=c}(R)$
  - $T(S)=T(R)/V(R,A)$
- 范围条件$S=\sigma_{A<a}(R)$
  - $T(S)=T(R)/3$(或2); $T(S)=T(R)\frac{a-min(A)}{max(A)-min(A)}$
- 复合条件$S=\sigma_{A=10\ AND\ B<20}(R)$
  - $T(S)=T(R)/V(R,A)/3$; $T(S)=\frac{T(R)(20-min(B))}{(max(B)-min(B))V(R,A)}$
- 复合条件$S=\sigma_{A=10\ OR\ B<20}(R)$
  - $T(S)=T(R)\left(1-\left(1-\frac{1}{V(R,A)}\right)\left(1-\frac{1}{3}\right)\right)$ or …

并、交、差操作结果大小的估计

- 并$R \cup S$
  - 包的并:$T(R)+T(S)$
  - 集合的并:$T(R)+T(S)$到$max\{T(R),T(S)\}$之间
    - 估计值: $max\{T(R),T(S)\}+\frac{1}{2}min\{T(R),T(S)\}$
- 交$R \cap S$
  - 结果可以是0到$min\{T(R),T(S)\}$之间的任何值
  - 估计值:$\frac{1}{2}min\{T(R),T(S)\}$
- 差$R-S$
  - $T(R)-\frac{1}{2}T(S)$

消除重复、分组聚集结果大小的估计

- 消除重复$\delta(R)$, $R(A_1, \ldots, A_n)$
  - 上限1: $T(R)$
  - 上限2: $\Pi_{i=1}^{n} V(R, A_i)$
  - 推荐: $min\left\{\frac{T(R)}{2}, \Pi_{i=1}^{n} V(R, A_i)\right\}$
- 分组聚集, 分组属性$A_1, \ldots, A_n$
  - 结果大小等于分组属性不同值的数量
  - 推荐: $min\left\{\frac{T(R)}{2}, \Pi_{i=1}^{n} V(R, A_i)\right\}$

连接大小的估计

- 只考虑自然连接$R(X,Y)\bowtie S(Y,Z)$
- 两个假设条件
  - 值集的包含: $V(R,Y)\le V(S,Y)$
    - R的每个Y值将是S的一个Y值
  - 值集的保持：如果X（Z）是只包含于R（S）的属性，则$V(R\bowtie S,X)=V(R,X)$, $V(R\bowtie S,Z)=V(S,Z)$,
    - 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，部分满足值集保持的约束
  - R.X保留，S.Z不一定保留，假设都保留下来了

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))

$R(a,b,c) \bowtie S(a,b,d)$的大小：$\dfrac{T(R)\times T(S)}{\max\{V(R,a),V(S,a)\}\times\max\{V(R,b),V(S,b)\}}$

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

- 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：

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

$$=950\times500/13\approx36539$$

$R(a,b)$	$S(b,c)$	$U(c,d)$
$T(R)$=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：$R \bowtie S \bowtie U$

$(R \bowtie S) \bowtie U$

首先计算：$T(R \bowtie S)=T(R)\times T(S)/\max(V(R,b), V(S,b))$

$$= 1000\times 2000 / 50 = 40000$$

得到中间结果关系$P(a,b,c)$：

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

$R \bowtie (S \bowtie U)$?

$S \bowtie (R \bowtie U)$?

P与U连接，得到最终结果大小：

$T(P \bowtie U)=T(P)\times T(U)/\max(V(P,c), V(U,c))$

$$= 40000\times 5000 / 500 = 400000$$

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：

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

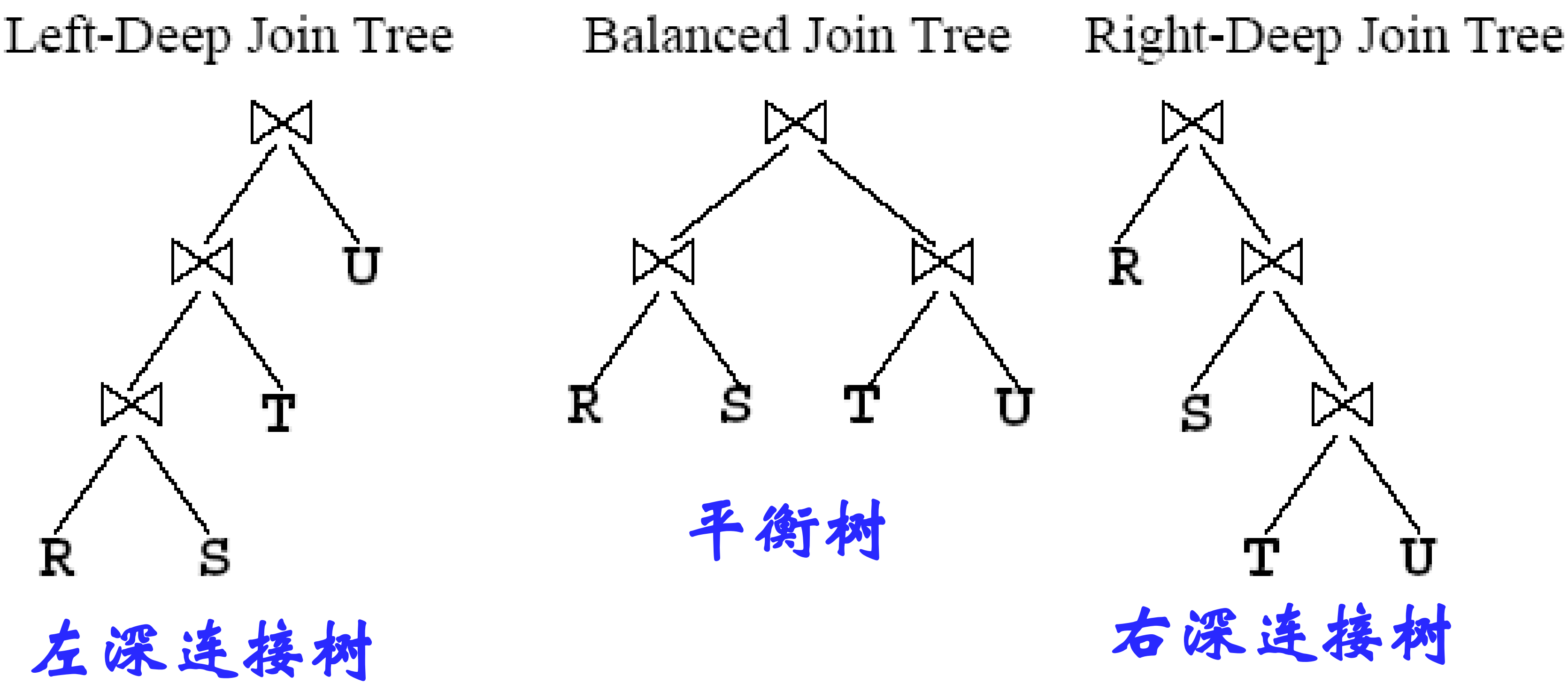
$$=15000+16000+3500+2500+9\times1250$$

$$=48250$$

- 查询计划执行方式
- 逻辑查询优化
  - 逻辑查询计划枚举
  - 基数估计
  - 连接顺序选择
- 物理查询优化

连接顺序的选择

- 连接树



n个关系时，共计$\frac{(2(n-1))!}{(n-1)!}$种连接顺序

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（I/O代价用中间结果衡量）

给定$U=\{R_1, R_2, \ldots\ldots, R_n\}$，求解计算$J(U)=R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$的最小代价$M(U)$，以及产生最小代价$M(U)$的连接计划

建立最优解代价的递归方程：

$$M(S)=\min_{\substack{X\cup Y=S\\X\cap Y=\emptyset\\X,Y\neq\emptyset}}\{M(X)+M(Y)+Cost(J(X)\bowtie J(Y))\}$$

得到J(X)和J(Y)的过程中产生的中间结果

J(X)和J(Y)作为中间结果的大小之和

即$size(\mathrm{J(X)})+size(\mathrm{J(Y)})$

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：表示产生中间结果的最小代价

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	10000	2000	1M	1000
代价	0	0	0	0	0	0
最佳计划	R⋈S	R⋈T	R⋈U	S⋈T	S⋈U	T⋈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	10000	2000	1M	1000
代价	0	0	0	0	0	0
最佳计划	R⋈S	R⋈T	R⋈U	S⋈T	S⋈U	T⋈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	10000	50000	10000	2000
代价	2000	5000	1000	1000
最佳计划	(S⋈T) ⋈R	(R⋈S) ⋈U	(T⋈U)⋈R	(T⋈U)⋈S

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

	{R, S}	{R, T}	{R, U}	{S, T}	{S, U}	{T, U}
大小	5000	1M	10000	2000	1M	1000
代价	0	0	0	0	0	0
最佳计划	R⋈S	R⋈T	R⋈U	S⋈T	S⋈U	T⋈U

	{R, S, T}	{R, S, U}	{R, T, U}	{S, T, U}
大小	10000	50000	10000	2000
代价	2000	5000	1000	1000
最佳计划	(S⋈T) ⋈R	(R⋈S) ⋈U	(T⋈U)⋈R	(T⋈U)⋈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
$((S \bowtie T) \bowtie R) \bowtie U$	12000
$((R \bowtie S) \bowtie U) \bowtie T$	55000
$((T \bowtie U) \bowtie R) \bowtie S$	11000
$((T \bowtie U) \bowtie S) \bowtie R$	3000
$(T \bowtie U) \bowtie (R \bowtie S)$	6000
$(R \bowtie T) \bowtie (S \bowtie U)$	2000000
$(S \bowtie T) \bowtie (R \bowtie U)$	12000

选择连接顺序的贪心算法

初始化：从最小的关系开始

启发式策略：在所有还没有包含在当前树中的关系里，寻找与当前树进行连接能生成估计代价最小的关系.当前树作为连接的左参数，选中的关系作为右参数来形成新的当前树.

- 查询计划执行方式
- 逻辑查询优化
- 物理查询优化

物理查询优化

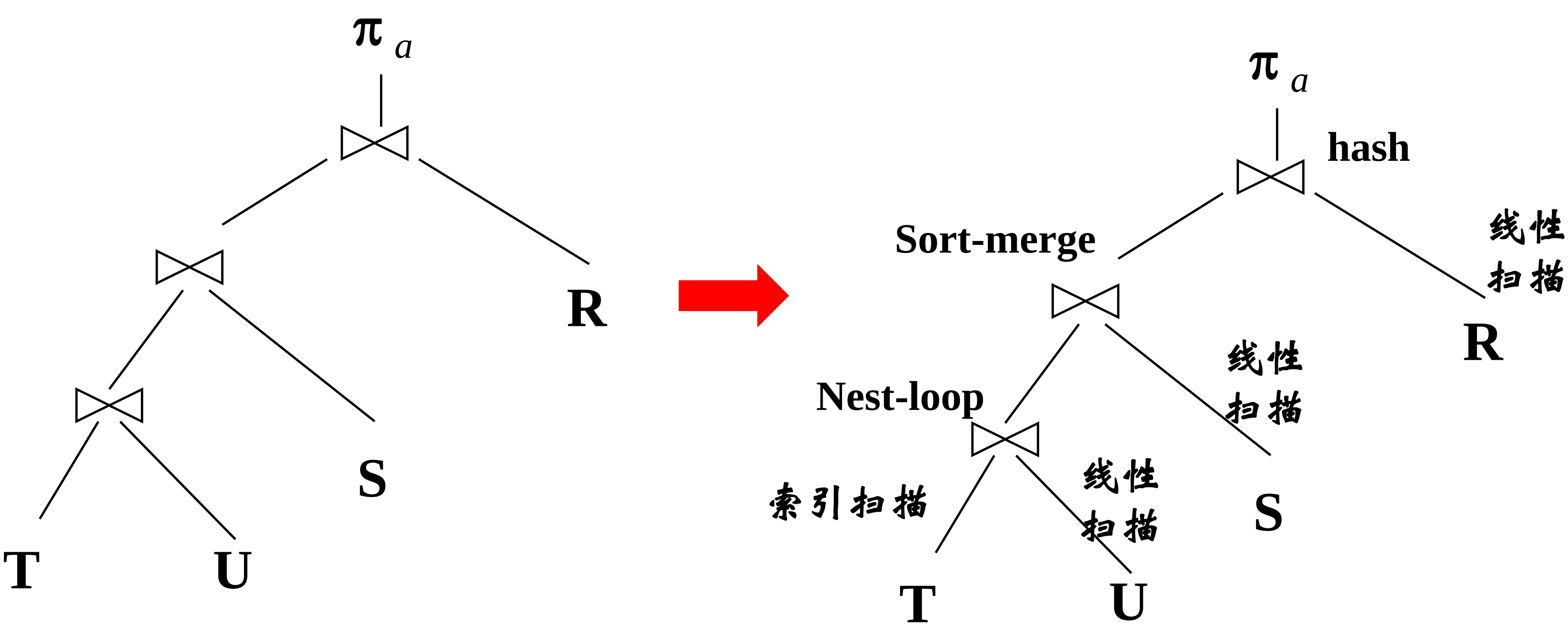
- 物理查询优化的任务
  - 为逻辑查询计划中的每个关系代数操作选择一个物理执行算法
  - 为逻辑查询计划中的中间结果确定传递到下一个操作符的方式（物化 or 流水）
- 物理查询计划中包含以下注释信息
  - 关系代数操作的执行算法
  - 中间结果的传递方式
  - 中间结果是否按照某些属性排序

物理查询优化

- 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，
    - 添加等价规则，表示关系操作符到物理操作符的转变
      - $\sigma_\theta(R) \equiv \sigma_\theta(R) < SeqScan(R) >$
      - $\sigma_\theta(R) \equiv \sigma_\theta(R) < IndexScan(R, BTree(R, Ai)) >$
      - $R \bowtie S \equiv R \bowtie S < NestLoopJoin >$
      - $R \bowtie S \equiv R \bowtie S < SortMergeJoin >$
      - $R \bowtie S \equiv R \bowtie S < HashJoin >$
  - 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。

- 连接算法的选择
  - 一个关系足够小，采用一趟连接算法
  - 大的关系上有连接属性上的聚集索引，采用索引连接
  - 参与连接的关系在连接属性上有序，采用SortMergeJoin
  - 以上不满足，哈希连接往往比较好
  - 如果可用缓冲区太少，采用NestLoopJoin

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



查询优化中的其他策略

- Plan Caching
  - 重用之前生成的查询计划
- 参数化的查询优化
  - 对于查询条件中的不同参数生成不同的查询计划，缓存备用
  - 未来的查询从缓存的多个计划中挑选，或者以之为基础，继续优化
- Adaptive Query Processing
  - 查询执行过程中发现查询优化使用的统计量不准确，查询计划效果不好，重新计算统计量并优化查询计划，重新执行

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:

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

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

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

请写出下推选择、下推选择和投影之后的关系操作表达式

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

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：

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

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属性上有聚集索引

聚集索引容纳于内存

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

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：

$$1+\frac{1000}{10}=101\text{块}$$

- 查询优化
  - 关系代数中的代数定律
    - 下推选择、投影
  - 查询代价估计
    - 中间结果大小的估计
  - 连接顺序的选择
    - 动态规划算法、贪心算法
  - 物理操作符的选择
    - 计算每个物理操作的运行代价

Now let's go to

Next Chapter