数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

第九章 关系查询处理及查询优化

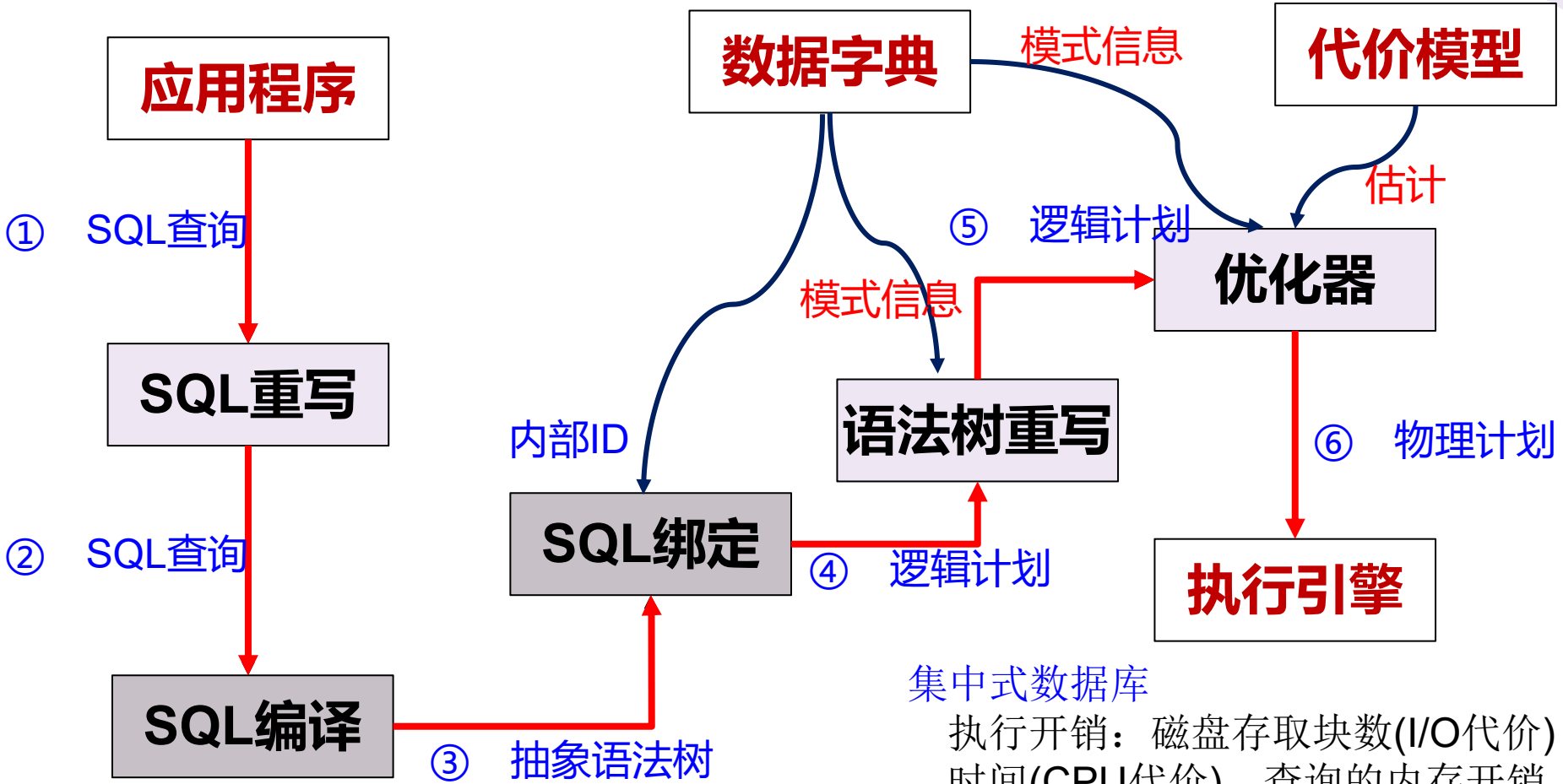
Principles of Database Systems

第九章 关系查询处理及查询优化

9.2 关系数据库系统的查询优化

- 查询优化是在多个查询可执行策略和操作算法中选取较优执行计划的过程。
- 为什么要进行查询优化?
  - SQL语言的“非过程化”特性：不需要用户指明存取路径，实现了数据独立性，但同时也使得用户无法干预系统实现；（进行查询优化的必要）
  - 查询操作的多解性：同一个查询可能存在多种效率不同的实现方案。（进行查询优化的可能）
- 逻辑优化：代数优化、语法级优化。
- 物理优化：通过某种代价模型计算出各种查询执行策略的执行代价，选取代价最小的执行方案。

关系数据库系统的查询优化



集中式数据库

执行开销：磁盘存取块数(I/O代价)*、处理机时间(CPU代价)、查询的内存开销

分布式数据库：I/O代价+CPU代价+内存代价+通信代价

一个实例

[例] 设有如下关系：

- 学生Student（学号，姓名，性别，出生日期，所在系）
- 课程Course（课号，名称，学分）
- 选课SC（学号，课号，成绩）

查询选修了2号课程的学生姓名。

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

可用如下等价的代数表达式来完成这一查询：

$Q_1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge Cno='2'}(Student \times SC))$

$Q_2 = \pi_{Sname}(\sigma_{Cno='2'}(Student \bowtie SC))$

$Q_3 = \pi_{Sname}(Student \bowtie \sigma_{Cno='2'}(SC))$

一个实例（续）

- 假定有1000条学生记录，10000条选课记录，其中选修了2号课程的记录为50条。
- 设一个物理块能装10条学生记录或100条选课记录，内存提供了7块存储空间，其中5块存放学生记录，1块存放选课记录，1块用来存放中间结果，磁盘每秒钟读/写20块数据记录。
- 由于查询策略不同，这三种方案的查询时间相差很大。
- 第一种方案： $Q_1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge Cno='2'}(Student \times SC))$

1) 计算笛卡尔积：学生×选课

读取总块数：1000/10 + 1000/(10×5)×10000/100 = 2100(块)

读取所需时间：T1 = 2100/20 = 105(秒)

学生记录块数

读选课记录遍数

选课记录块数

一个实例（续）

笛卡尔积运算结果的元组个数为：$10^3 \times 10^4 = 10^7$，每块能装10个元组，则该结果集共需10^6块：

写出所需时间：$T2 = 10^6/20 = 5 \times 10^4$(秒)

2) 作选择操作

依次读入笛卡尔积运算结果到内存（10^6块），选择满足条件的记录，假定内存处理时间忽略不计，则读取中间结果的时间T3与T2相等。即：$T3=5 \times 10^4$ (秒)

因为满足条件的记录仅有50条，可留在内存，无需写盘。

3) 作投影操作

将内存中的结果在“Sname”上作投影，得最终结果。内存处理时间忽略不计。

因此，方案一执行查询的总时间为：

$$T = T1+T2+T3 = 105 + 5 \times 10^4 + 5 \times 10^4 \approx 10^5 \text{(秒)}$$

27.8小时

读写总块数：$2100 + 10^6 + 10^6 \approx 2 * 10^6$

第二种方案：$\pi_{Sname}(\sigma_{SC.cno='2'}(Student \bowtie SC))$

1) 计算自然连接：学生⋈选课

读取总块数与方案一相同，故所需时间也与方案一相同：2100块

T1=105(秒)

但自然连接的运算结果只有10^4个元组（why? ），设每块能装10个元组，则该结果集共需10^3块，写出所需时间：T2 = 10^3/20 = 50 (秒)

2) 作选择操作：依次读入自然连接运算结果（10^3块），选择满足条件的记录，假定内存处理时间忽略不计，读取中间结果的时间T3与T2相等。即：T3=50 (秒)

因为满足条件的记录仅有50条，可留在内存。

3) 作投影操作：将内存中的结果在“Sname”上作投影，得最终结果。处理时间忽略不计。

方案二执行查询总时间为：T = T1+T2+T3 = 105 + 50 + 50 ≈205 (秒)

读写总块数：2100+ 10^3 + 10^3 ≈ 4* 10^3

第三种方案：π_{Sname}（Student$\bowtie\sigma_{Cno='2'}$（SC））

1) 对选课表作选择操作

扫描一遍选课表，需要读取的块数为：10000/100 = 100(块)

花费时间为：T1 = 100/20 = 5 (秒)，因满足条件的记录为50条，不必写磁盘。

2) 作自然连接

读取学生表，与内存中的选课记录作连接，需要读取的块数为：1000/10 = 100(块)，连接结果不必写磁盘。

花费时间为：T2 = 100/20 = 5 (秒)

3) 输出投影结果

在内存中操作，处理时间忽略不计。

因此，方案三执行查询的总时间为：T = T1+T2 = 5 + 5 ≈10 (秒)。

读写总块数：100 + 100 = $2*10^2$

由10^5→205→10s，本例充分说明了查询优化的必要性，同时也给出了一些查询优化方法的初步概念。

一个实例(续)

- 假如SC表（内表）的Cno字段上有索引：
  - 第一步就不必读取所有的SC元组而只需读取Cno= ‘2’ 的那些元组(50个)；
  - 存取的索引块和SC中满足条件的数据块大约总共3～4块。

- 若Student表在Sno上也有索引：
  - 第二步也不必读取所有的Student元组；
  - 因为满足条件的SC记录仅50个，涉及最多50个Student记录；
  - 读取Student表的块数也可大大减少

- 总的存取时间将进一步减少到数秒。

一个实例(续)

- 把代数表达式Q_1变换为Q_2、 Q_3，
  - 即有选择和连接操作时，先做选择操作，这样参加连接的元组就可以大大减少，这是代数优化，即：通过对关系代数表达式的等价变换来提高查询效率。
- 在Q_3中，
  - SC表的选择操作算法有全表扫描和索引扫描2种方法，经过初步估算，索引扫描方法较优；
  - 对于Student和SC表的连接，利用SC表上的索引，采用index join代价也较小，这就是物理优化。

9.3 代数优化

通过对关系代数表达式的等价变换来提高查询效率。

9.3.1 关系代数表达式等价转换规则

(课程⋈学生)⋈选课 = (学生⋈课程)⋈选课 = 学生⋈(课程⋈选课)

- 笛卡儿积、自然连接的 交换律 和 结合律
  - $E1 \times E2 \equiv E2 \times E1$;　　$(R \times S) \times T \equiv R \times (S \times T)$
  - $E1 \bowtie E2 \equiv E2 \bowtie E1$;　　$(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
  - $E1 \underset{F}{\bowtie} E2 \equiv E2 \underset{F}{\bowtie} E1$;　　$(R \underset{F1}{\bowtie} S) \underset{F2}{\bowtie} T \equiv R \underset{F1}{\bowtie} (S \underset{F2}{\bowtie} T)$

R

A	B
10	20
20	30
30	40

S

A	C
10	20
20	30
30	40

T

C	D
20	20
10	30
10	40

$(R \bowtie S) \bowtie T$：中间结果$R \bowtie S$产生3条记录

$R \bowtie (S \bowtie T)$：中间结果$S \bowtie T$产生1条记录

查询结果相同，但

查询代价可能不同

关系代数表达式等价转换规则

投影的串接律

$$\pi_{A1,A2,\cdots,An}(\pi_{B1,B2,\cdots,Bm}(E)) \equiv \pi_{A1,A2,\cdots,An}(E)$$

在同一个关系上，只需做一次投影运算，且一次投影时选择多列同时完成。

选择的串接律

$$\sigma_{F1}(\sigma_{F2}(E)) \equiv \sigma_{F1 \wedge F2}(E)$$

选择条件可以合并，使得一次选择运算就可检查全部条件，而不必多次过滤元组

选择与投影的交换律

(1)假设: 选择条件F只涉及属性A1，…，An

$$\sigma_F(\pi_{A1,A2,\cdots,An}(E)) \equiv \pi_{A1,A2,\cdots,An}(\sigma_F(E))$$

先选择后投影

(2)假设: F中有不属于A1，…,An的属性B1,…,Bm

$$\pi_{A1,A2,\cdots,An}(\sigma_F(E)) \equiv \pi_{A1,A2,\cdots,An}(\sigma_F(\pi_{A1,A2,\cdots,An,B1,B2,\cdots,Bm}(E)))$$

先做带选择条件中的列投影，然后选择，再完成最外层投影

关系代数表达式等价转换规则

□ 选择与笛卡尔积的交换律

(1) 若F中涉及的属性都是E1中的属性

$$\sigma_F(E1 \times E2) \equiv \sigma_F(E1) \times E2$$

(2)若F=F1∧F2，并且F1只涉及E1中的属性，F2只涉及E2中的属性，则：

$$\sigma_F(E1 \times E2) \equiv \sigma_{F1}(E1) \times \sigma_{F2}(E2)$$

(3)若F=F1∧F2，F1只涉及E1中的属性，F2涉及E1和E2两者的属性，则：

$$\sigma_F(E1 \times E2) \equiv \sigma_{F2}(\sigma_{F1}(E1) \times E2)$$

部分选择在笛卡尔积前先做。

条件下推到相关的关系上，先做选择后做笛卡尔积运算，这样可以减小中间结果的大小。

关系代数表达式等价转换规则

□ 选择与并的分配率

假设：E=E1∪E2，E1，E2有相同的属性名

$$\sigma_F(E1 \cup E2) \equiv \sigma_F(E1) \cup \sigma_F(E2)$$

条件下推到相关的关系上，先选择后做并运算，可以减小每个关系输出结果的大小

□ 选择与差的分配率

假设：E1与E2有相同的属性名

$$\sigma_F(E1 - E2) \equiv \sigma_F(E1) - \sigma_F(E2)$$

先选择后做差运算

□ 选择与自然连接的分配率

假设：E1和E2是两个关系表达式，

$$\sigma_F(E1 \bowtie E2) \equiv \sigma_F(E1) \bowtie \sigma_F(E2)$$

先选择后做自然连接运算

关系代数表达式等价转换规则

□ 投影与笛卡尔积的分配率

设E1和E2是两个关系表达式，A1，…，An是E1的属性，B1，…，Bm是E2的属性，则：

$$\pi_{A1,A2,\cdots,An,B1,B2,\cdots,Bm}(E1\times E2)\equiv \pi_{A1,A2,\cdots,An}(E1)\times \pi_{B1,B2,\cdots,Bm}(E2)$$

先投影后做笛卡尔积

□ 投影与并的分配率

设E1和E2 有相同的属性名

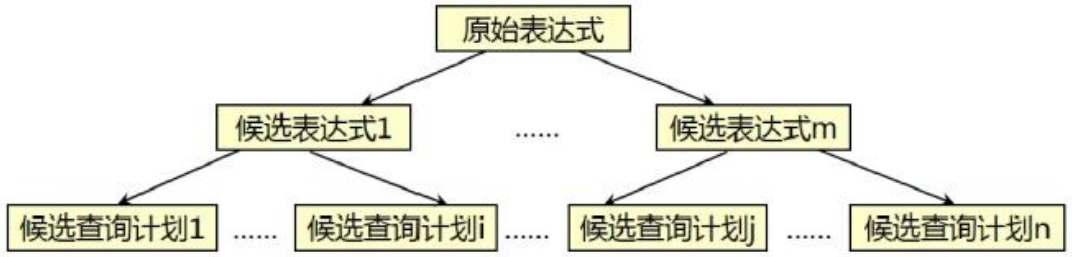
$$\pi_{A1,A2,\cdots,An}(E1\cup E2)\equiv \pi_{A1,A2,\cdots,An}(E1)\cup \pi_{A1,A2,\cdots,An}(E2)$$

先投影后做并操作

注意没有投影和交的分配率

9.3.2 查询树的启发式规则

- 转换的最终目的
  - 减少查询的开销（I/O次数）
- 转换的直接目的
  - 减少中间关系的元组数、元组大小



- 规则1：选择运算尽早执行（减小中间结果集的规模）；
- 规则2：将选择、投影运算同时进行（避免重复扫描）；
- 规则3：把投影同其前或其后的双目运算结合（避免重复扫描次数）；
- 规则4：把选择运算与其前面的笛卡尔积运算结合起来一起计算；（避免重复扫描并减小中间结果集的规模）
- 规则5：把公共子表达式的运算结果存放在外存，作中间结果，使用时读入主存。如：视图表达式（避免重复计算）

关系表达式代数优化算法

1）运用选择的串接定律，得到选择运算“串”；

2）对每个选择运算符，利用等价变换尽量将其移至树的叶端（规则4~9）；

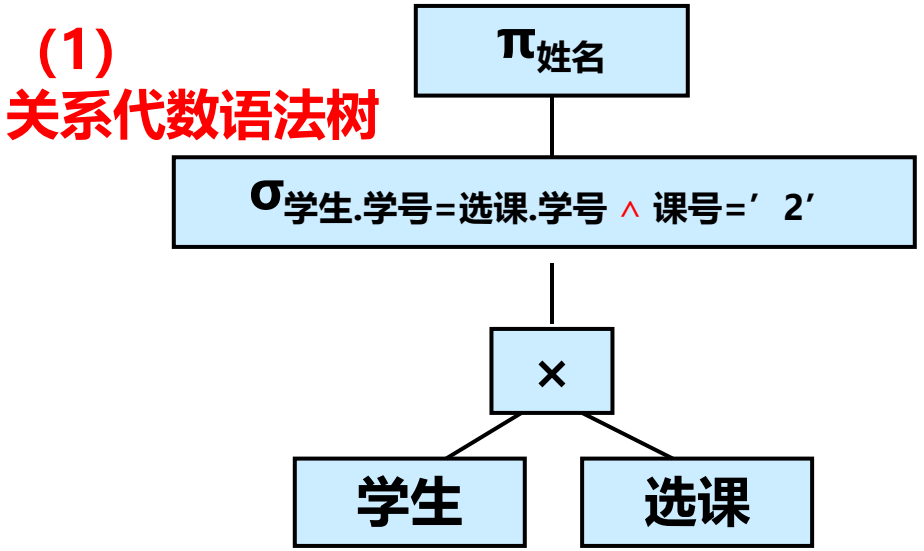
3）对每个投影运算符，利用等价变换尽量将其移至树的叶端（规则3、5、10、11）；

4）尝试将“选择”和“投影”串接合并成单个“选择”或“投影”，或“选择”后跟一个“投影”（规则3~5）；

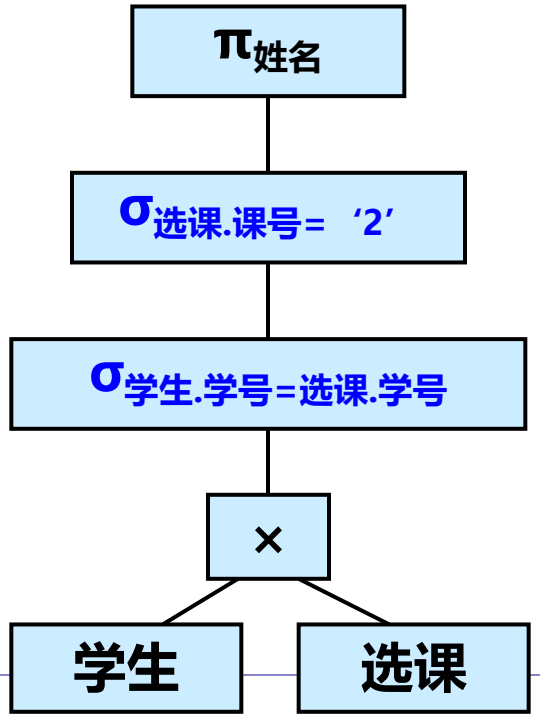
5）上述得到的语法树内结点分组：双目运算和它的父节点为一组。若其后代直至叶节点全是单目运算，也合并为一组。笛卡尔积的子节点若是不能组合成等值连接的“选择”，则二者不合并。

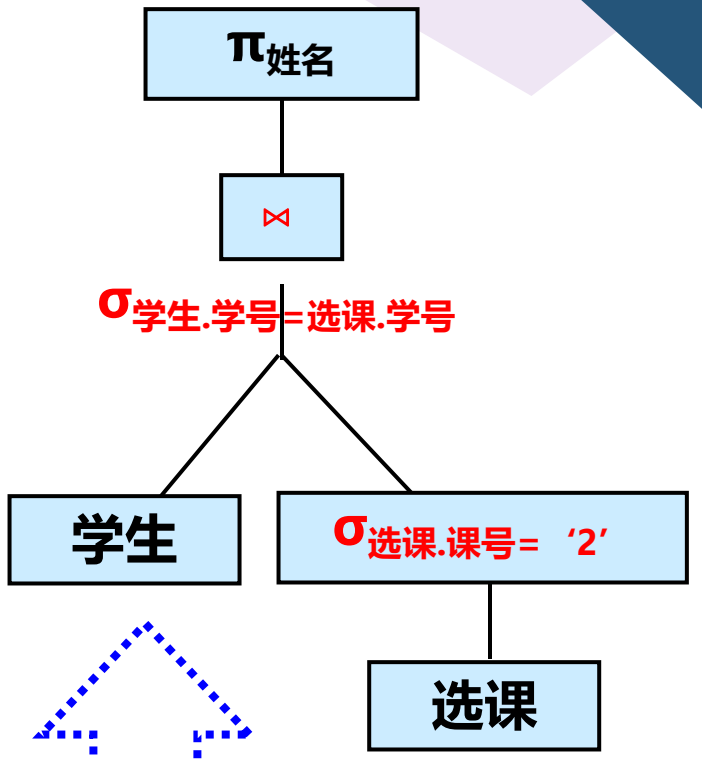
[例] 应用等价变换公式来优化关系表达式

(1)
关系代数语法树



(2)
分解选择运算

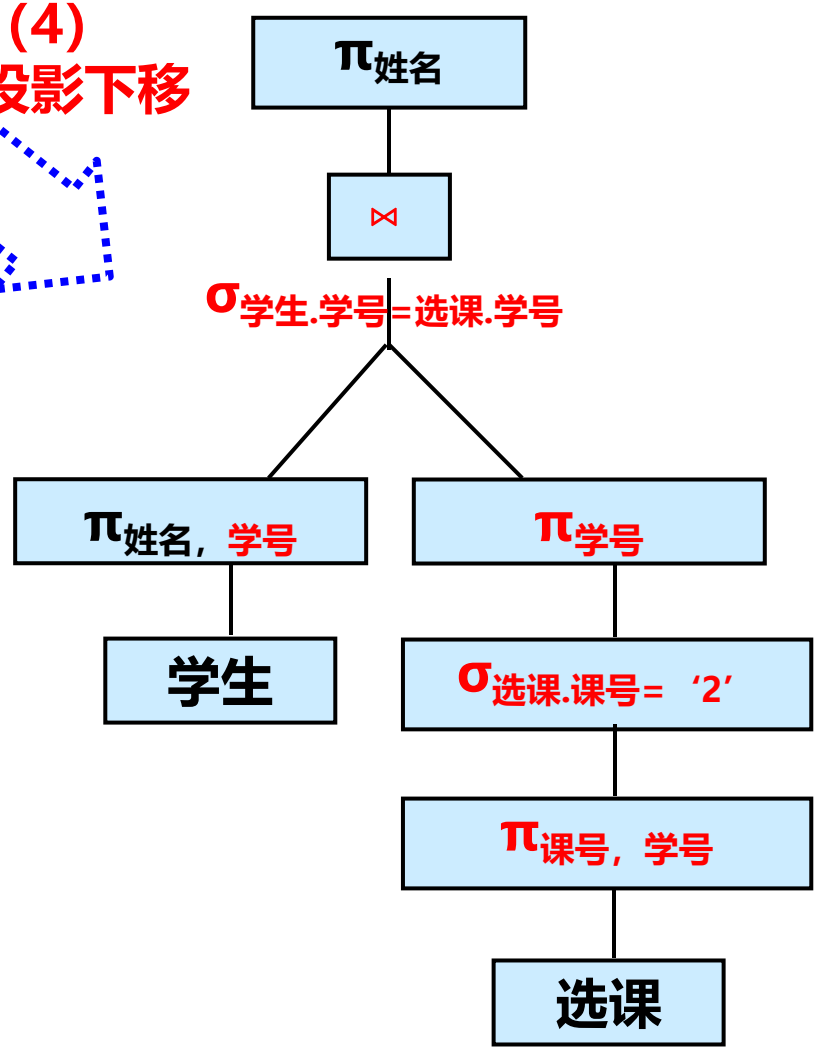




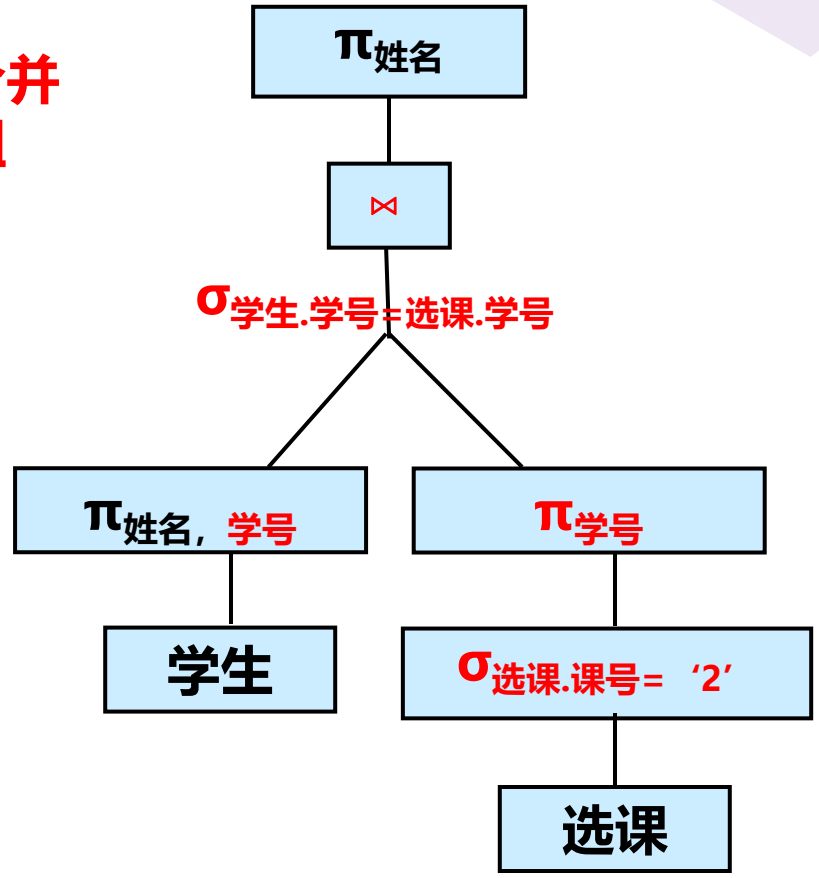
(3)
通过交换选择运算，选择下移，
将其尽可能移到叶端，
将带连接条件的笛卡尔积转化为内连接

[例] 应用等价变换公式来优化关系表达式

(4)
投影下移



(5)
串接合并
与分组



此为代数优化后结果

9.4 物理优化

- 代数优化改变查询语句中操作的次序和组合，不涉及底层的存取路径。
- 物理优化就是要选择高效合理的操作算法或存取路径，求得优化的查询计划。常常先使用启发式规则选取若干较优的候选查询方案，然后分别估算这些候选方案的执行代价，从而选取代价最小的作为执行计划。

- 代价模型
  - 物理代价：CPU，IO，内存，.....，依赖于硬件
  - 逻辑代价：结果集大小估计，依赖于算子算法，统计信息等
  - 算法代价：算子算法的时空复杂度

9.4.1 基于启发式规则的优化

1. **选择操作的启发式规则**:

1) 对于小关系，使用全表扫描，即使选择列上有索引；

2) 对于大关系：

- 如果是主码值上的等值查询“主码=值”，使用主码索引；
- 如果是非主码值上的等值查询，在选择比例<10%的情况下，使用索引（如果有）；否则使用全表扫描；
- 如果非等值或范围查询，在选择比例<10%的情况下，使用索引（如果有）；否则使用全表扫描；
- 对于用AND连接的合取选择条件，如果有涉及这些属性的组合索引，优先采用组合索引扫描方法；如果某些属性上有一般索引，则可以用索引扫描方法；否则使用全表顺序扫描。
- 对于用OR连接的析取选择条件，一般使用全表顺序扫描

9.4.1 基于规则的优化

2. 连接操作的启发式规则：

1）如果2个表都已经按照连接属性排序

选用排序-合并方法

2）如果一个表在连接属性上有索引

选用索引连接方法

3）如果上面2个规则都不适用，其中一个表较小

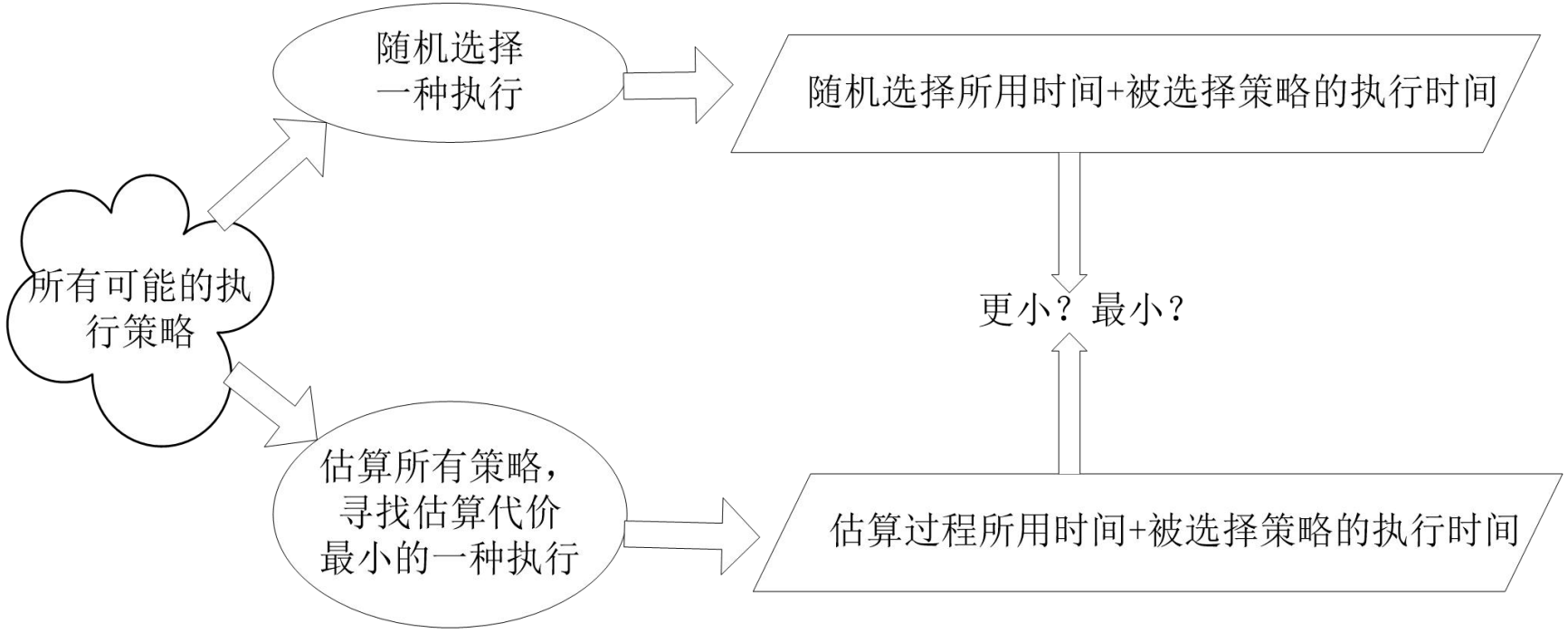
选用Hash join方法

4）可以选用嵌套循环方法，并选择其中占用块较小的表，作为外表(外循环的表)。

9.4.2 基于代价的优化

- 为什么采用启发式规则?

可能的执行策略很多，要穷尽所有的策略进行代价估算需要的计算开销往往与被连接的关系数成指数复杂度关系。



可能造成查询优化过程的开销大于获得的查询开销的减小，得不偿失。

9.4.1 基于规则的优化

- 启发式规则：在一般情况下适用，但不一定保证获得最优执行计划。（试运行、DBA性能调整）

- 和启发式方法类似的其他解决方法：

  贪婪算法、遗传算法。。。

  其思想都是类似——求近似最优解。

- 程序执行方式
  - 解释执行：（优化开销包含在查询总开销中），启发式方法
  - 编译执行：基于代价的优化方法、启发式方法。

9.4.2 基于代价估算的优化

- 查询代价估算
- 查询代价估算基于CPU代价和I/O代价，计算公式如下：

  总代价 = I/O代价 + CPU代价

  COST = P * a_page_cpu_time + W * T

  - 其中：
    - P是计划运行时访问的页面数；a_page_cpu_time是每个页面读取的时间开销，其乘积反映了I/O开销。
    - T为访问的元组数，如果是索引扫描，还要考虑索引读取的开销，反映了数据读取到内存的CPU开销。
    - W为权重因子，表明I/O到CPU的相关性，又称选择率（selectivity）。

9.4.2 基于代价估算的优化

统计信息：

- 基于代价的优化要计算各种操作算法的执行代价，与数据库状态密切相关
- 数据字典中存储优化器需要的统计信息，包括表、属性和索引的信息：

1. 对每个基本表
   - 该表的元组总数(N)
   - 元组长度(l)
   - 占用的块数(B)
   - 占用的溢出块数(BO)

2. 对基表的每个列
   - 该列不同值的个数(m)
   - 选择率(f)
     - 如果不同值的分布是均匀的，f＝1/m
     - 如果不同值的分布不均匀，则每个值的选择率＝具有该值的元组数/N
   - 该列最大值
   - 该列最小值
   - 该列上是否已经建立了索引
   - 索引类型(B+树索引、Hash索引、聚集索引)

3. 对索引(如B+树索引)
   - 索引的层数(L)
   - 不同索引值的个数
   - 索引的选择基数S(有S个元组具有某个索引值)
   - 索引的叶结点数(Y)

常见统计信息

代价估算

- 中间结果的大小估计
- 需要使用一些统计数据(statistics)
  - T(R) : R的元组数
  - S(R) : R中每个元组的大小(bytes)
  - V(A, R) : R的属性A上的不同值数

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

A: 20 byte； B: 4 byte； C: 8 byte； D: 5 byte

关于R的统计数据：

T(R) = 5　　S(R) = 37

V(A,R) = 3　　V(C,R) = 5

V(B,R) = 1　　V(D,R) = 4

关于中间结果W = R1×R2的大小估计如下：

T(W) = T(R1)×T(R2)

S(W) = S(R1) + S(R2)

代价估算

- 选择--中间结果的大小估计

  选择大小的估计 W = σ_p (R)

  SC(A,R) : 选择基数。关系R中对于属性A的每个值的元组平均数量。

  SC(A,R) = T(R) / V(A,R) //这个参数成立的前提假设是数据均匀分布。

- 选择率依赖于谓词P的类型：
  - 相等
  - 范围
  - 非
  - 与
  - 或

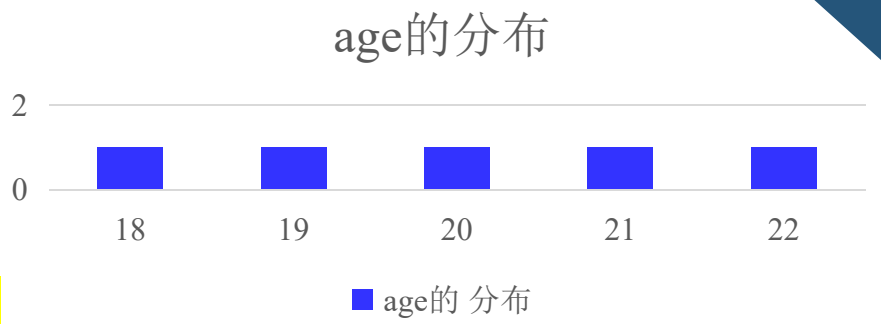
选择率计算方法

- 假设V(age,people)具有5个不同的值（18-22），且T(R)=5。
- 相等谓词：

【例】查询： Select * from People where age=20;

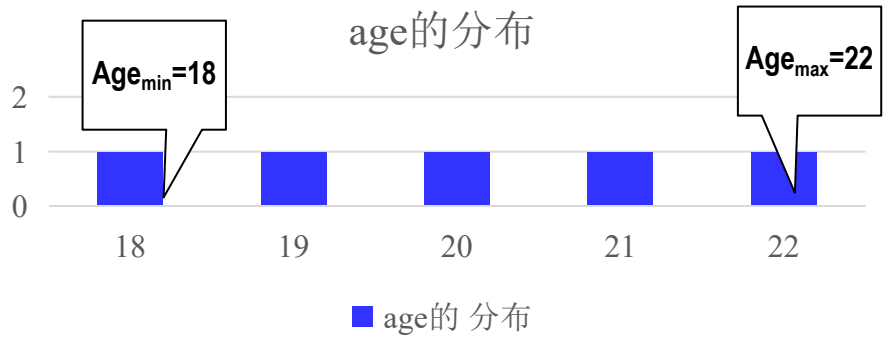
  - 相等谓词：A=常量
  - 相等谓词的选择率： Sel(A=constant)=SC(P) / T(R)

    Sel(age=20)=1/5



- 范围谓词：
  - Sel(A>=a) = $(A_{max}-a+1) / (A_{max}-A_{min}+1)$
  - 查询： Select * from People where age>=20;

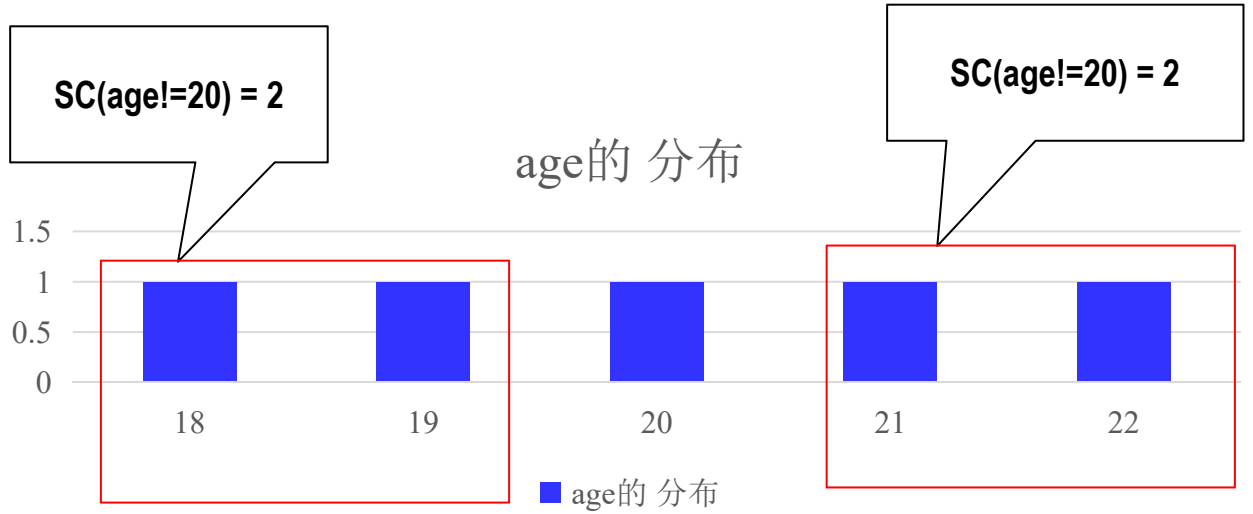
    Sel(age>=20)=(22-20+1)/(22-18+1)=3/5



选择率计算方法

- 非谓词：
  - Sel(!P) = 1-sel(P)
  - 查询： Select * from People where age != 20;

    Sel(age!=20)=1-1/5=4/5



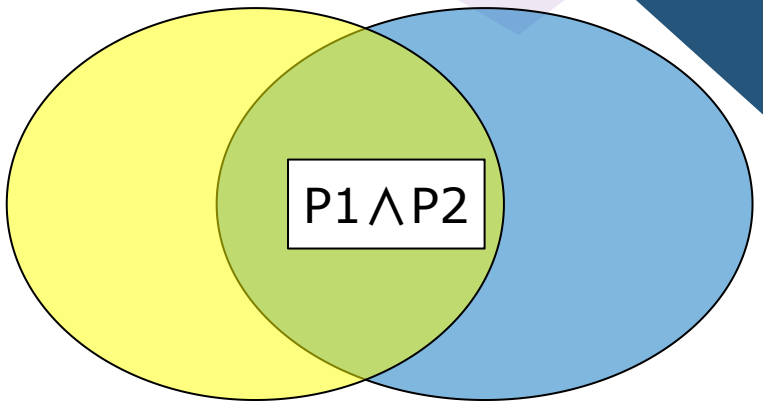
选择率计算方法

- 与谓词:
  - 当谓词之间的选择独立时:

    $Sel(P1 \wedge P2)=sel(P1)\times sel(P2)$

  - 查询： Select * from People

    where age=20 and name like ’ 王%’;



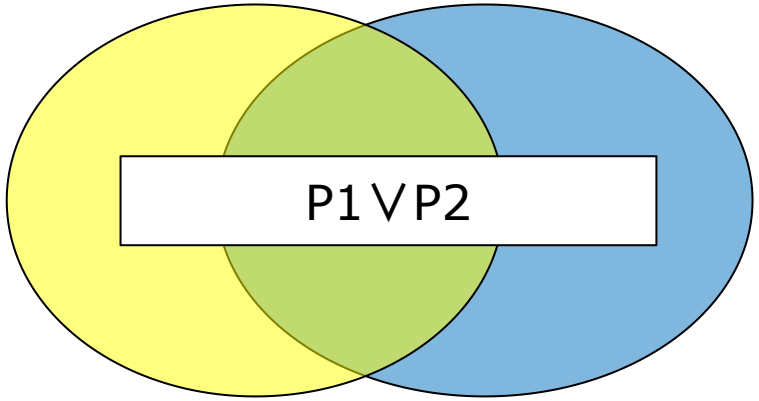
- 或谓词:
  - 当谓词之间的选择独立时:

    $Sel(P1 \vee P2)=Sel(P1)+Sel(P2)-Sel(P1 \wedge P2)$

    $= Sel(P1)+Sel(P2)-Sel(P1)\times Sel(P2)$

  - 查询： Select * from People

    where age=20 or name like ’ 王%’;



选择率计算方法

- Join结果集大小的估计
  - 给定关系R和S，估计R inner join S的结果集大小：
  - 若 $R \cap S = \varnothing$，则 $R \bowtie S$ 等价于 R x S, $T(R \bowtie S) = T(R) \times T(S)$ 。
  - 否则：假设 $R_{cols} \cap S_{cols} = \{A\}$，
    - 若{A}是R的key，则S中的一个元组至多只能连接到R中一个元组，则 $T(R \bowtie S) <= T(S)$ 。若该key是S的外键，则 $T(R \bowtie S) = T(S)$；否则：
      - 假设R中的每一个元组在S中至少能找到一个匹配，产生$R \bowtie S$元组：

        $T(R \bowtie S) = T(R) \times T(S) / V(A,S)$
      - 假设S中的每一个元组在R中至少能找到一个匹配，产生$R \bowtie S$元组：

        $T(R \bowtie S) = T(S) \times T(R) / V(A,R)$

I/O代价估算

- 估计执行查询计划所必须读（写）的磁盘块数目。需要另外一些参数：
  - B(R)：R所需的块数；
  - f(R)：每块可容纳的R的最大元组数
  - M：可用的内存块数；
  - Nr：关系R元组数；
  - Frs：连接选择性；
  - Mrs：连接结果单块记录数；
  - L：索引深度；
  - S：索引选择基数；
  - Y：索引叶结点数
  - HT(i)：索引i的层数；
  - LB(i)：索引i的叶结点所需的块数

I/O代价估算

- 全表扫描算法的代价估算公式
  - 如果基本表大小为 B 块，全表扫描算法的代价 cost = B
  - 如果选择条件是码 = 值，则平均代价 cost = B/2
- 索引扫描算法的代价估算公式
  - 选择条件“码=值”，设索引层数为L，则cost=L+1
  - 选择条件涉及非码，S为索引选择基数，则最坏情况下cost=L+S
  - 若比较条件为>,>=,<,<=，则 cost=L+Y/2+B/2
- 嵌套循环连接算法的代价估算公式
  - 连接代价：Cost=Br+BrBs/(K-1), 且 K< B(R) < B(S)，其中K表示缓冲区大小为K块
  - 中间结果写回磁盘： Cost=Br+BrBs/(K-1)+(Frs*Nr*Ns)/Mrs。Frs为连接选择率，Mrs为块因子
- 排序合并连接算法的代价估算公式
  - 如果连接表已经按照连接属性排好序，则 cost =Br+Bs+(Frs*Nr*Ns)/Mrs 。
  - 如果必须对文件排序需要在代价函数中加上排序的代价对于包含 B 个块的文件排序的代价大约是(2*B)+(2*B*log2B)

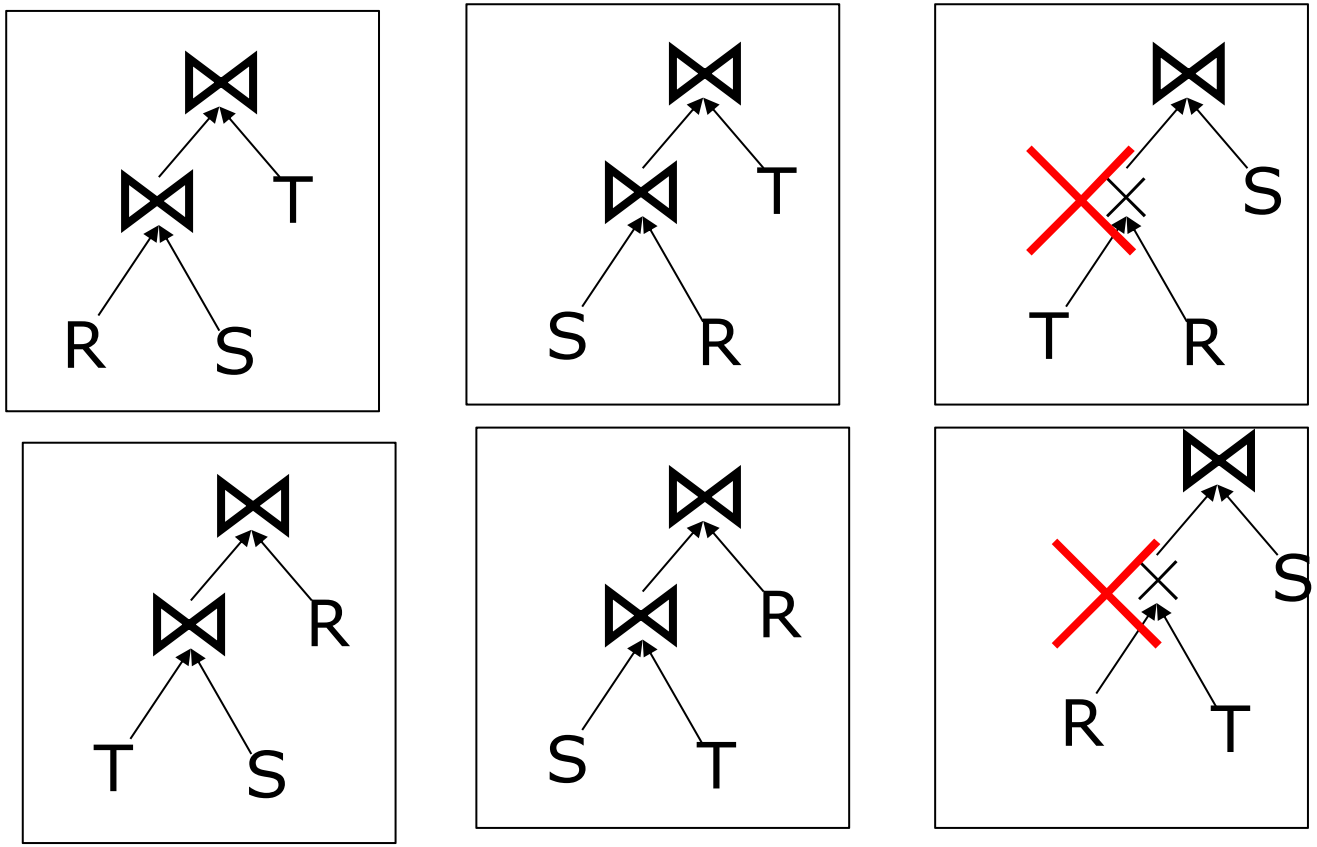
多表连接代价估算

- 单关系查询计划：只考虑表访问的方法：
  - 顺序扫描
  - 二分查找（聚簇索引）
  - 索引扫描
- 两表连接查询计划：连接方式、连接路径
- 多表连接查询计划：在查询路径生成的过程中，根据代价估算，从各种可能的候选路径中找出最优的路径。
- 多表连接算法实现的是主要解决两个问题：
  - 多表连接顺序
  - 多表连接的搜索空间：N个表的连接可能有N！种连接组合

  连接的表越多，枚举的查询计划呈指数级上升

计划枚举

1. 枚举所有可能的连接顺序

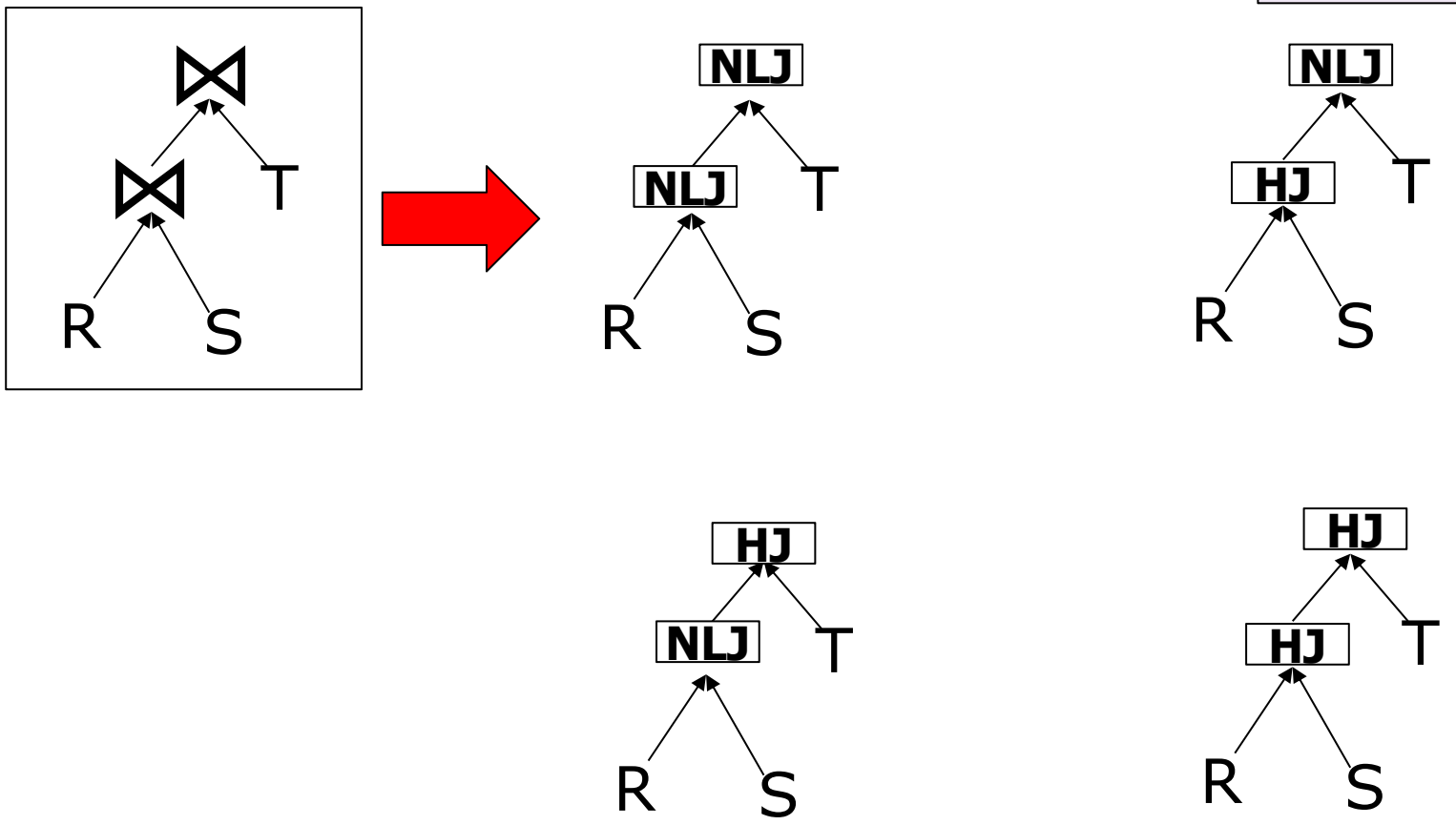


Select * from R,S,T
Where R.a=S.a
And S.b=T.b

计划枚举

Select * from R,S,T
Where R.a=S.a
And S.b=T.b

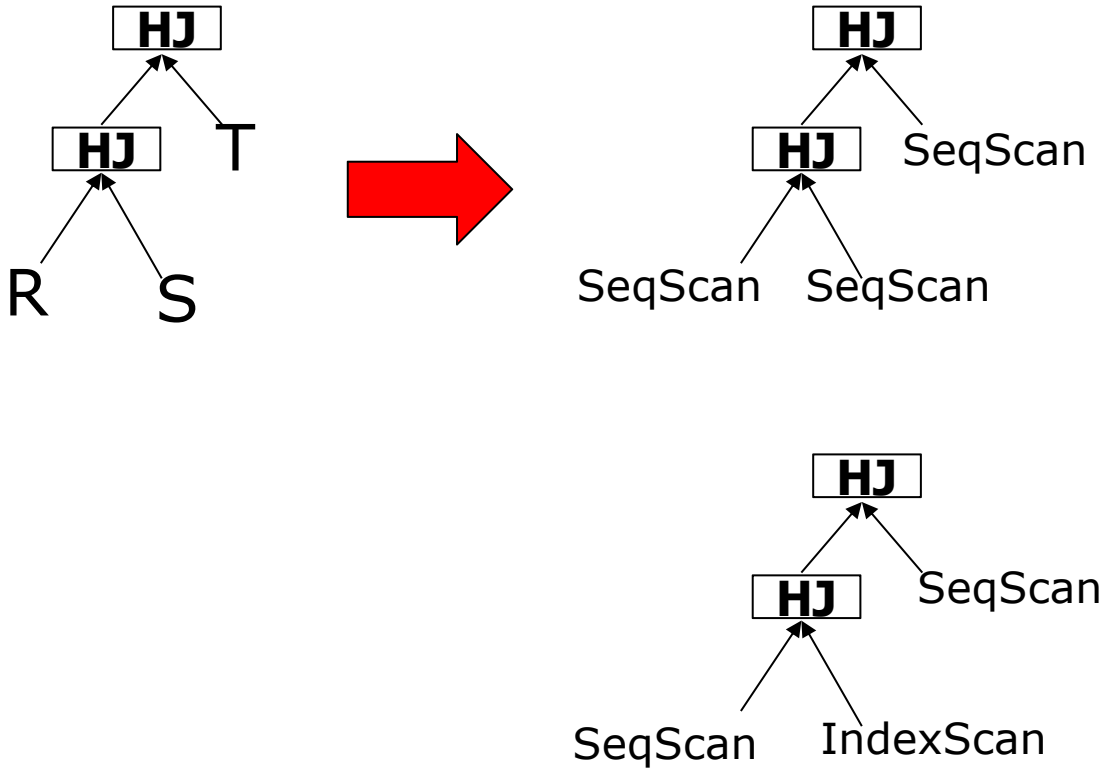
2. 枚举所有可能的连接算法



计划枚举

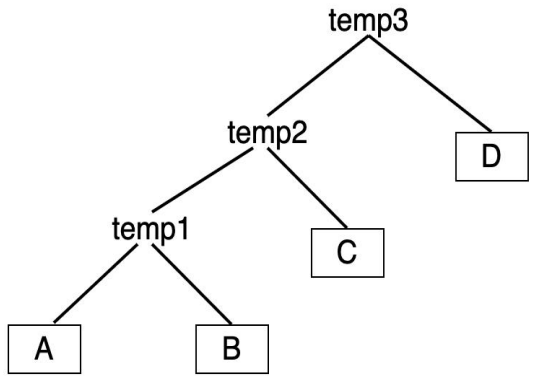
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

3.枚举所有可能的数据存取方法

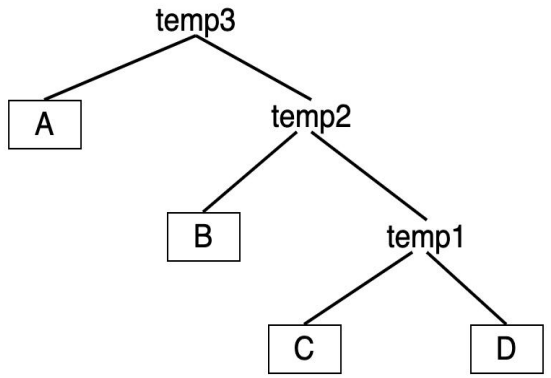


查询树基本形态

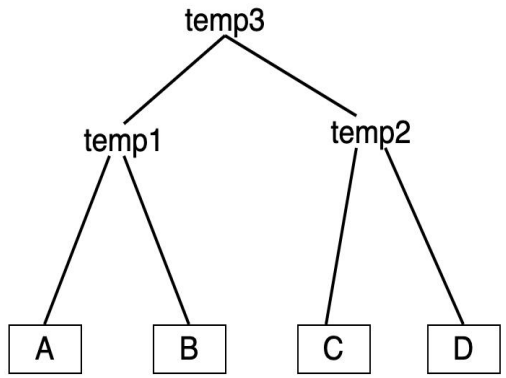
- 多表查询计划：左深树、右深树、紧密树



a) 左深连接树



b) 右深连接树



c) 紧密树

- System R简化这一问题，只考虑左深树

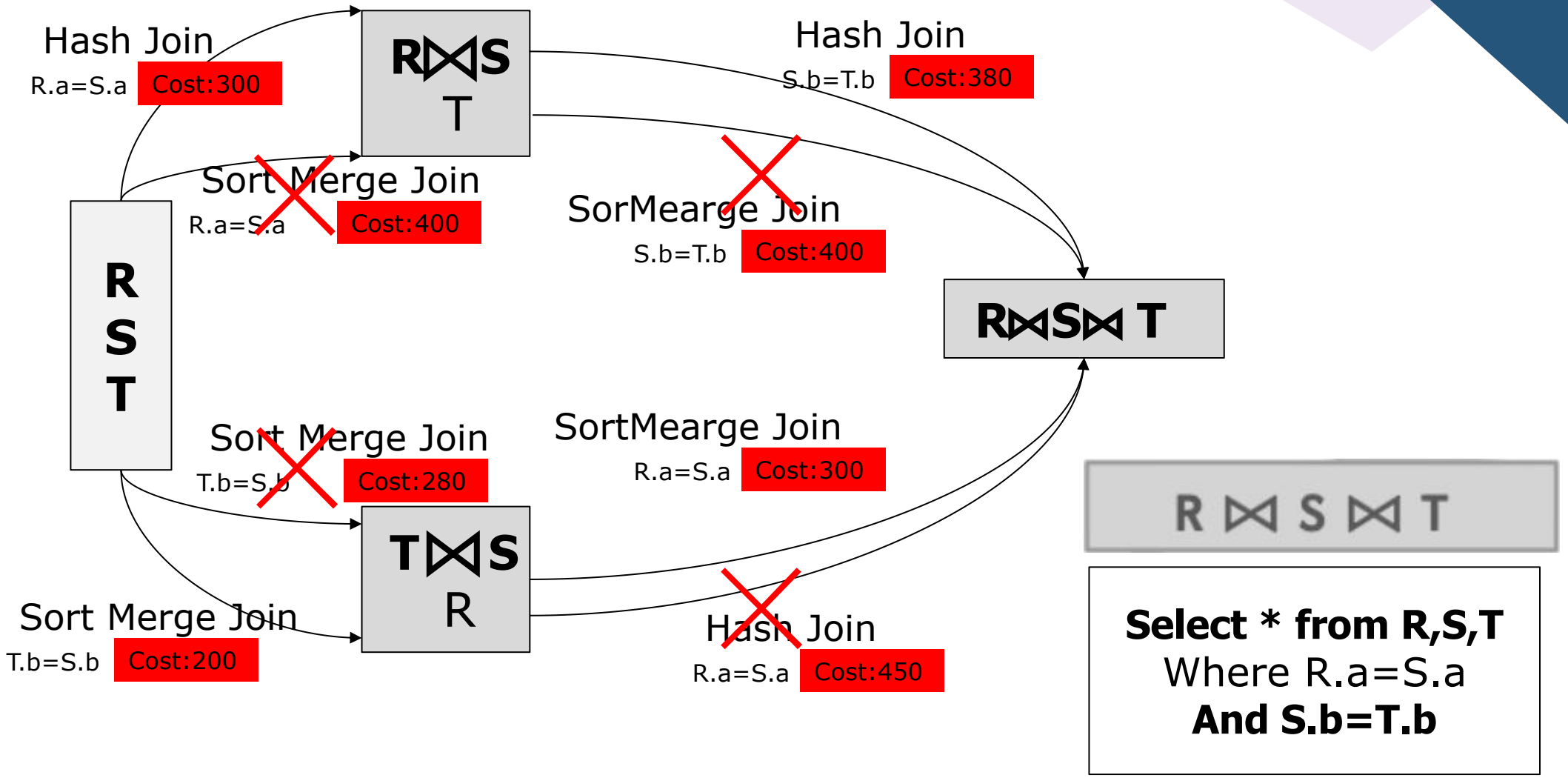
  System R认为非左深树不能流水线计算，不符合火山模型

多表连接代价估算

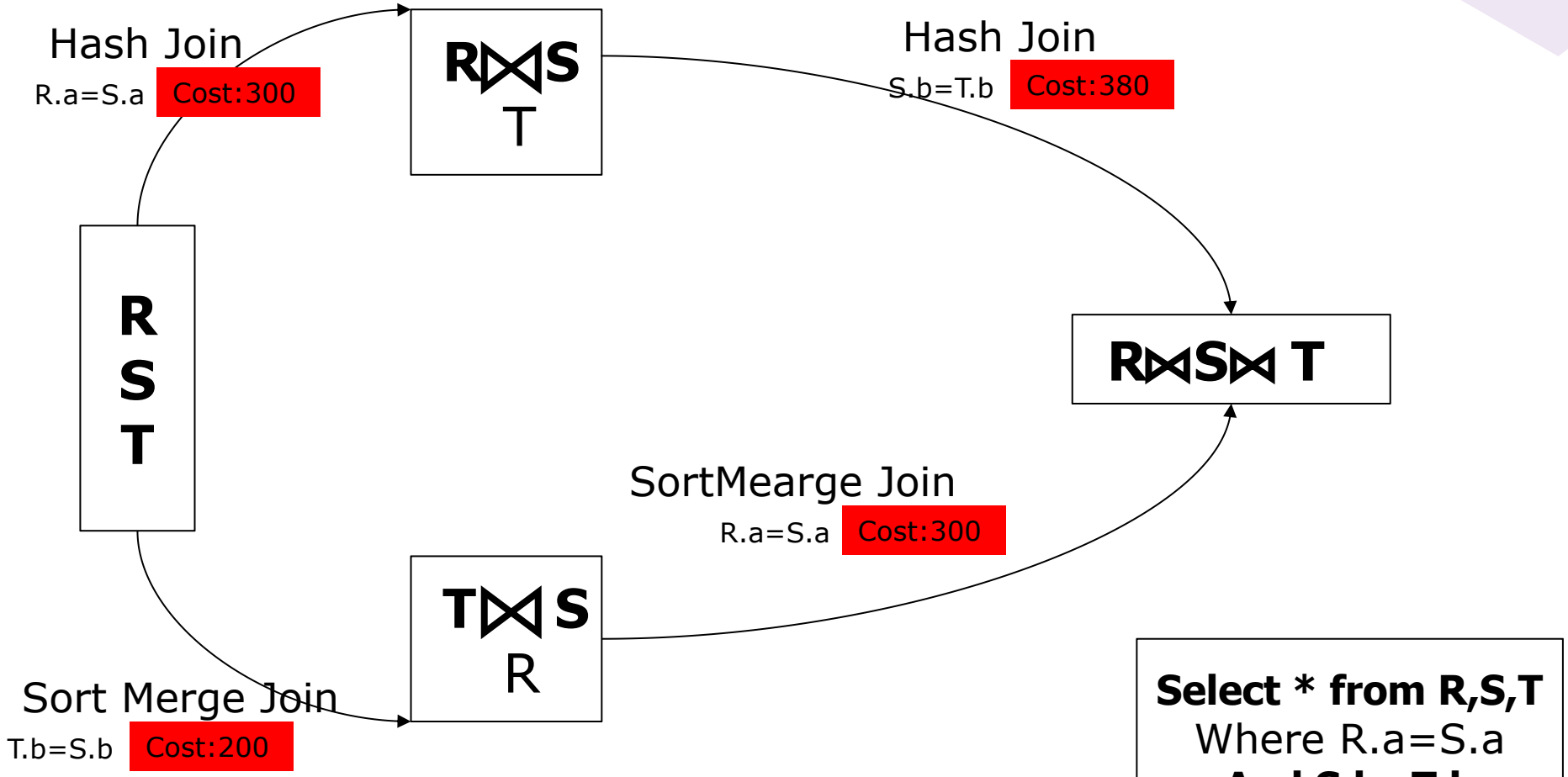
- 多表连接搜索最优查询树方法：
  - 动态规划
  - System R优化方法
  - 贪心
  - 启发式
  - 分枝界定计划枚举
  - 爬山法
  - 遗传算法等。

算法名称	特点与适用范围	缺点
启发式算法	适用于任何范围，与其它算法结合，能有效提高整体效率	不知道得到的解是否最优
贪心算法	非穷举类型的算法。适合解决较多关系的搜索	得到局部最优解
动态规划算法	穷举类型的算法。适合查询中包含较少关系的搜索，可得到全局最优解	搜索空间随关系个数增长呈指数增长
System R 优化	基于自底向上的动态规划算法，为上层提供更多可能的备选路径，可得到全局最优解	搜索空间可能比动态规划算法更大一些

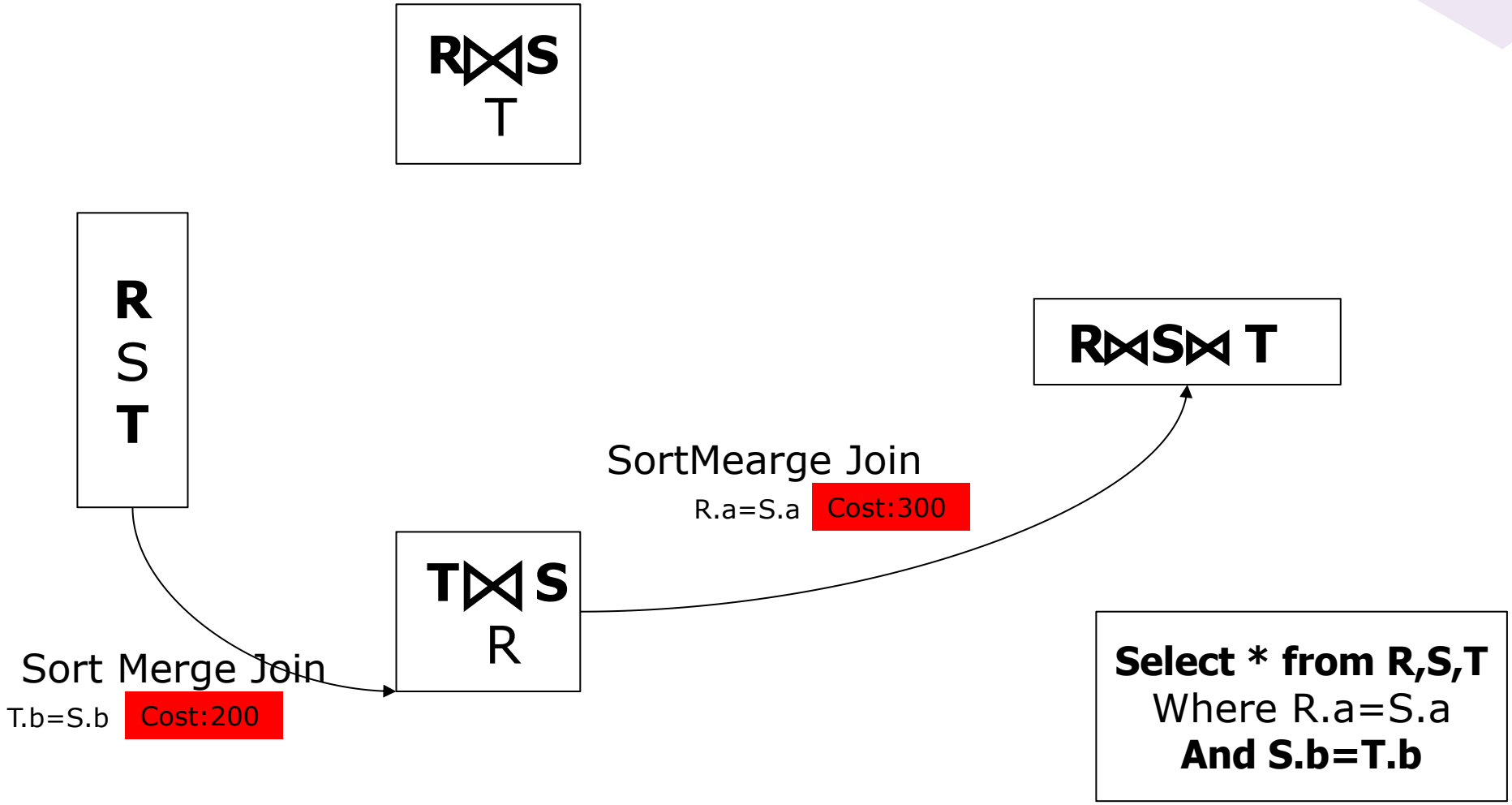
应用动态规划的优化


Hash Join
R.a=S.a
Cost:300
R⋈S
T
Hash Join
S.b=T.b
Cost:380
Sort Merge Join
R.a=S.a
Cost:400
SorMearge Join
S.b=T.b
Cost:400
R
S
T
R⋈S⋈ T
Sort Merge Join
T.b=S.b
Cost:280
SortMearge Join
R.a=S.a
Cost:300
T⋈S
R
Sort Merge Join
T.b=S.b
Cost:200
Hash Join
R.a=S.a
Cost:450
R ⋈ S ⋈ T
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

应用动态规划的优化


Hash Join
R.a=S.a
Cost:300
R⋈S
T
Hash Join
S.b=T.b
Cost:380
R
S
T
R⋈S⋈ T
SortMearge Join
R.a=S.a
Cost:300
T⋈S
R
Sort Merge Join
T.b=S.b
Cost:200
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

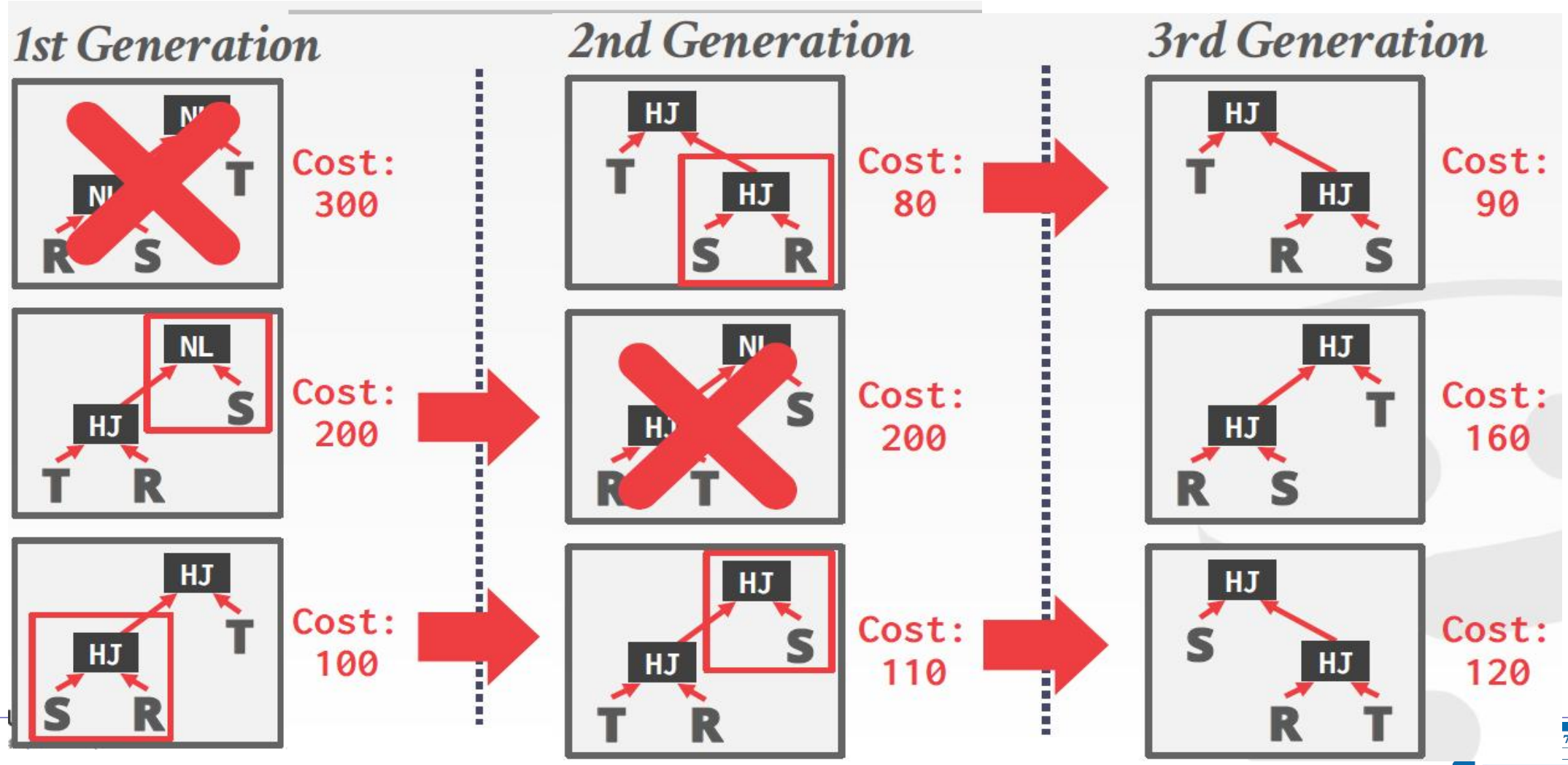
应用动态规划的优化


R⋈S
T
R
S
T
R⋈S⋈ T
SortMearge Join
R.a=S.a
Cost:300
T⋈S
R
Sort Merge Join
T.b=S.b
Cost:200
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

多表连接代价估算

GEQO优化方法主要针对于语法树的重构。每次都淘汰一种花费最多的方案，对于其他的方案都给机会。

- POSTGRES优化器



查询优化小结

- 优化是为了减小查询的代价，包括I/O、CPU、内存和通信代价，是关系数据库的重要问题。
- 导致查询代价较高的一个主要原因是关系代数的笛卡儿积运算。
- 查询优化的过程：
  - 查询树经过变形后得到语法树，
  - 然后根据代数优化的启发式规则对语法树进行逻辑优化，
  - 再考虑存取路径、底层操作算法的不同，根据物理操作的启发式规则提出多种查询计划，
  - 然后可根据某种代价模型评估这些查询计划的执行代价，从中选取评估结果最小的作为执行计划。

DBMS查询优化器

成熟的关系DBMS系统都有比较好的优化器，包括一些开源的RDBMS都有比较好的查询优化算法。

1）优化器可以综合的考虑代数表达式等价变换、物理操作的启发式规则、基于数据字典统计信息的代价评估。

2）出色的优化器可以设计更好的启发式优化算法、更精准的代价评估模型，有助于高效寻找到开销更小的执行方案。

3）影响代价评估的数据逻辑与物理分布等统计信息，只有DBMS内部才能获得最贴近真实情况的数据，并及时更新。

遗传算法

人工智能

本章小结

- 查询处理是RDBMS的核心，查询优化技术是查询处理的关键技术
- 本章讲解的优化方法
  - 启发式代数优化
  - 基于规则的存取路径优化
  - 基于代价的优化
- 比较复杂的查询，尤其是涉及连接和嵌套的查询
  - 不要把优化的任务全部放在RDBMS上
  - 应该找出RDBMS的优化规律，以写出适合RDBMS自动优化的SQL语句

课堂作业

Q: 对于学生选课数据库中的选课关系SC和学生关系S，各表的主关键字上都建有索引，假设两个关系都无法全部放入内存，对SQL语句：

```
SELECT * FROM SC, S
    WHERE SC.SNO=S.SNO
            AND S.SNO=1
            AND (CNO=1 OR CNO=2);
```

按照查询优化的启发式规则，请

1）画出原始查询树及关系代数优化后的语法树；

2）简要分析DBMS可能会 如何执行这句话中的选择运算和连接运算。