关系查询处理和查询优化

9.1 问题的提出

非过程化，无需显示指明存取路经。

数据的处理说到底还是有先后顺序和过程的，非过程化只是用户层面使用数据操纵语言的一种“用户体验”

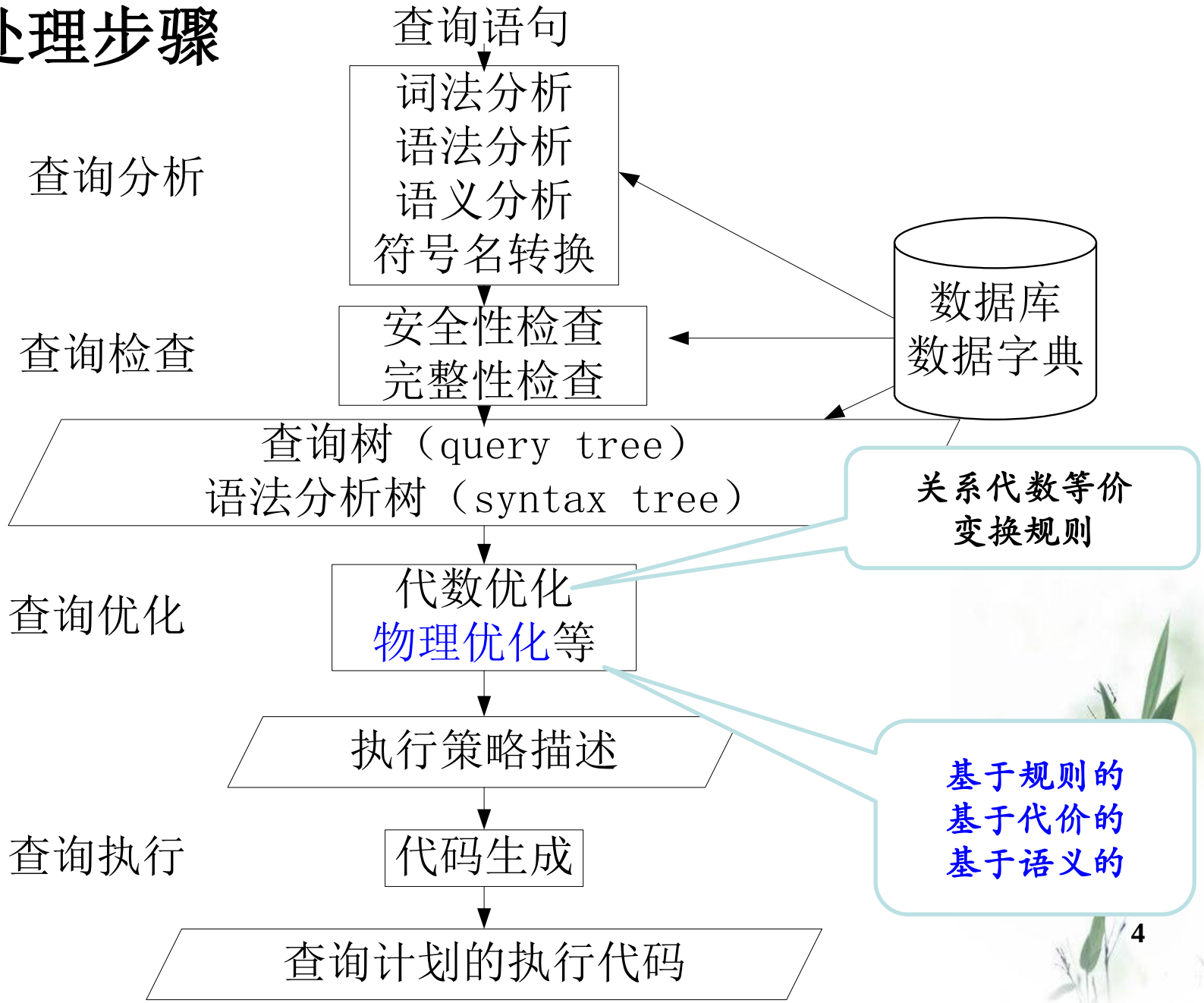
从非过程化的“用户体验”到实际的操作处理过程（例如两个集合并运算，是先读R1集合还是先读R2集合就涉及到过程），如何决策的任务留给了DBMS

基于规则的

- 代数优化——优化关系代数表达式

- 物理优化——优化存取路径和底层操作算法，可进一步分为基于规则的（rule based）、基于代价的（cost based，代价模型）和基于语义的（semantic based）。

查询处理步骤


查询语句
词法分析
语法分析
语义分析
符号名转换
查询分析
安全性检查
完整性检查
查询检查
数据库
数据字典
查询树（query tree）
语法分析树（syntax tree）
关系代数等价
变换规则
代数优化
物理优化等
查询优化
执行策略描述
基于规则的
基于代价的
基于语义的
代码生成
查询执行
查询计划的执行代码

一个实例

[例] 求选修了2号课程的学生姓名。

S(sno,sname,sage,ssex,sdept)

Sc(sno,cno,grade)

- 系统可以用多种等价的关系代数表达式来完成这一查询

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

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

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

实现查询操作的算法示例

一、 选择操作的实现

二、 连接操作的实现

一、 选择操作的实现

- ［例1］Select * from student where <条件表达式> ；

  考虑<条件表达式>的几种情况：

  C1：无条件；

  C2：Sno＝'200215121'；

  C3：Sage>20；

  C4：Sdept＝'CS' AND Sage>20；

选择操作的实现（续）

- **选择操作典型实现方法：**
  - **1. 简单的全表扫描方法**
    - 对查询的基本表顺序扫描，逐一检查每个元组是否满足选择条件，把满足条件的元组作为结果输出
    - 适合小表，不适合大表
  - **2. 索引(或散列)扫描方法**
    - 选择条件中的属性上有索引(例如B+树索引或Hash索引)
    - 通过索引先找到满足条件的元组主码或元组指针，再通过元组指针直接在查询的基本表中找到元组

选择操作的实现（续）

- ［例1-C2］ 以C2为例，Sno＝ ‘200215121’，并且Sno上有索引
  - 使用索引得到Sno为 ‘200215121’ 元组的指针
  - 通过元组指针在student表中检索到该学生

- ［例1-C3］ 以C3为例，Sage>20，并且Sage 上有B+树索引
  - 使用B+树索引找到Sage＝20的索引项，以此为入口点在B+树的顺序集上得到Sage>20的所有元组指针
  - 通过这些元组指针到student表中检索到所有年龄大于20的学生。

选择操作的实现（续）

- ［例1-C4］ 以C4为例，Sdept＝ ‘CS’ AND Sage>20，如果Sdept和Sage上都有索引：
  - 算法一：分别找到Sdept＝ ‘CS’的一组元组指针和Sage>20的另一组元组指针
    - 求这2组指针的交集
    - 到student表中检索
    - 得到计算机系年龄大于20的学生
  - 算法二：找到Sdept＝ ‘CS’的一组元组指针，
    - 通过这些元组指针到student表中检索
    - 对得到的元组检查另一些选择条件(如Sage>20)是否满足
    - 把满足条件的元组作为结果输出。

二、 连接操作的实现

- 连接操作是查询处理中最耗时的操作之一
- 本节只讨论等值连接(或自然连接)最常用的实现算法

［例2］ SELECT * FROM Student，SC
WHERE Student.Sno=SC.Sno；

连接操作的实现（续）

1. 嵌套循环方法(nested loop)

2. 排序-合并方法(sort-merge join 或merge join)

3. 索引连接(index join)方法

4. Hash Join方法

连接操作的实现（续）

1. 嵌套循环方法(nested loop)

- 对外层循环(Student)的每一个元组，检索内层循环(SC)中的每一个元组
- 检查这两个元组在连接属性(sno)上是否相等
- 如果满足连接条件，则串接后作为结果输出，直到外层循环表中的元组处理完为止

简单通用，对于参与连接的表没有要求。由于对内层循环表需要进行多次扫描，因此性能较差

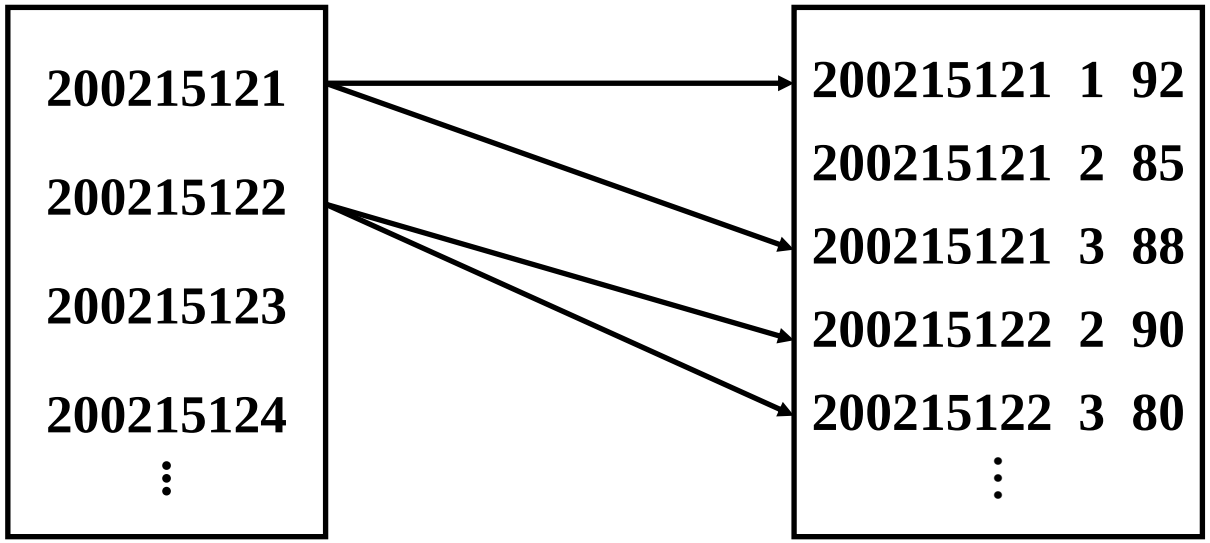
适用于一个集合大而另一个集合小的情况(将小集合做为外循环)

连接操作的实现（续）

2. 排序-合并方法(sort-merge join 或merge join)

- 适合连接的诸表已经排好序的情况
- 排序—合并连接方法的步骤：
  - 如果连接的表没有排好序，先对Student表和SC表按连接属性Sno排序
  - 取Student表中第一个Sno，依次扫描SC表中具有相同Sno的元组

连接操作的实现（续）



排序-合并连接方法示意图

- 当扫描到Sno不相同的第一个SC元组时，返回Student表扫描它的下一个元组，再扫描SC表中具有相同Sno的元组，把它们连接起来
- 重复上述步骤直到Student 表扫描完

连接操作的实现（续）

- Student表和SC表都只要扫描一遍
- 如果2个表原来无序，执行时间要加上对两个表的排序时间
- 对于2个大表，先排序后使用sort-merge join方法执行连接，总的时间一般仍会减少

连接操作的实现（续）

3. 索引连接(index join)方法

- 步骤：

① 在SC表上建立属性Sno的索引

② 对Student中每一个元组，由Sno值通过SC的索引查找相应的SC元组

③ 把这些SC元组和Student元组连接起来

循环执行②③，直到Student表中的元组处理完为止

若参与连接的表在连接属性上有索引，就可以采用索引连接算法。利用索引可以迅速定位元组，可以加快连接过程中的元组查找速度，提高连接效率。

连接操作的实现（续）

4. Hash Join方法

- 把包含较少元组的表（student）进行一遍处理，把它的元组按hash函数（ hash 码是连接属性）分散到hash表的桶中。
- 对另一个表（sc）进行一遍处理，把该表的元组按同一个hash函数（ hash 码是连接属性）进行散列，找到适当的hash桶，并把sc元组与来自student表与之匹配的元组连接起来。

把连接属性作为hash码，用同一个hash函数对参与连接的两张表分别进行hash分桶。在连接时，只需要将两张表中hash值相同的桶相互连接从而减少了计算量。
一般适用于较小的那个连接表在hash后能完全放入内存的情况。

查询优化概述（续）

- RDBMS通过某种代价模型计算出各种查询执行策略的执行代价，然后选取代价最小的执行方案
  - 集中式数据库
    - ➢执行开销主要包括：
      - 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
    - ➢I/O代价是最主要的
  - 分布式数据库
    - ➢总代价=I/O代价+CPU代价+内存代价＋通信代价

查询优化概述（续）

- 查询优化的总目标：
  - 选择有效的策略
  - 求得给定关系表达式的值
  - 使得查询代价最小(实际上是较小)

一个实例（续）

- 系统可以用多种等价的关系代数表达式来完成这一查询

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

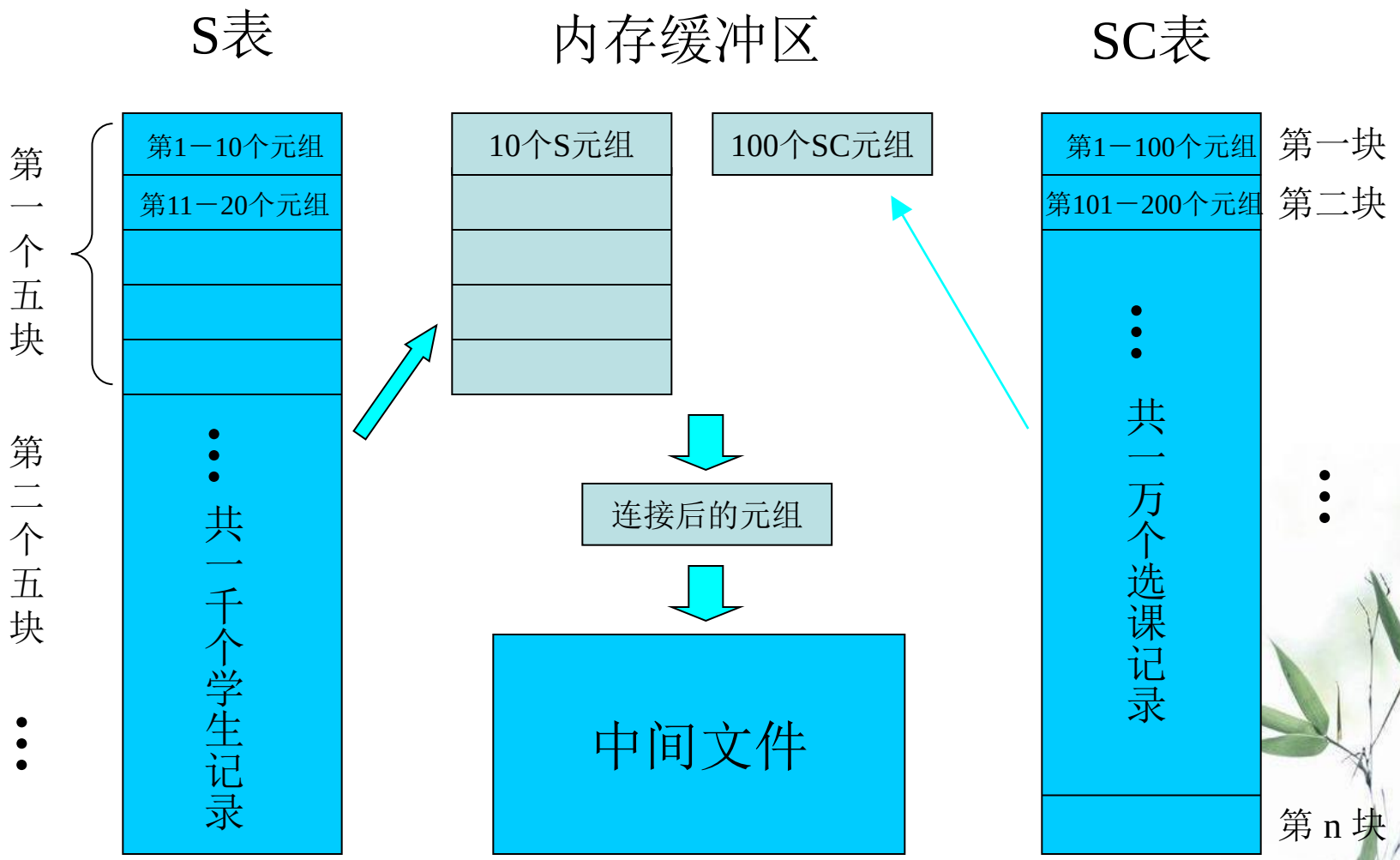
$Q_2=\pi_{Sname}(\sigma_{Sc.Cno='2'}(S\bowtie SC))$

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

系统假设:

- 元组数：student 1000个, SC 10000个，其中有关2号课程的50个
- 内存数：5个内存块用于装student, 1个内存块用于装SC
- 块大小：一个内存块可装10个student元组(或10个student与SC笛卡尔积元组)，或100个SC元组
- 读写速度: 20块/秒 (读写速度一样)
- 连接方法：基于数据块的嵌套循环法
- 连接后的元组装满一块后就写到中间文件上

- 读取S和SC表的策略


S表
内存缓冲区
SC表
第一个五块
第二个五块
第1－10个元组
第11－20个元组
共一千个学生记录
10个S元组
100个SC元组
连接后的元组
中间文件
第1－100个元组
第101－200个元组
共一万个选课记录
第一块
第二块
第 n 块

Q1: $\Pi_{Sname}(\sigma_{S.sno=SC.sno \wedge SC.Cno='2'}(S \times SC))$

S(1000行)

学号 Sno	姓名 Sname
05001	李勇
05002	刘晨

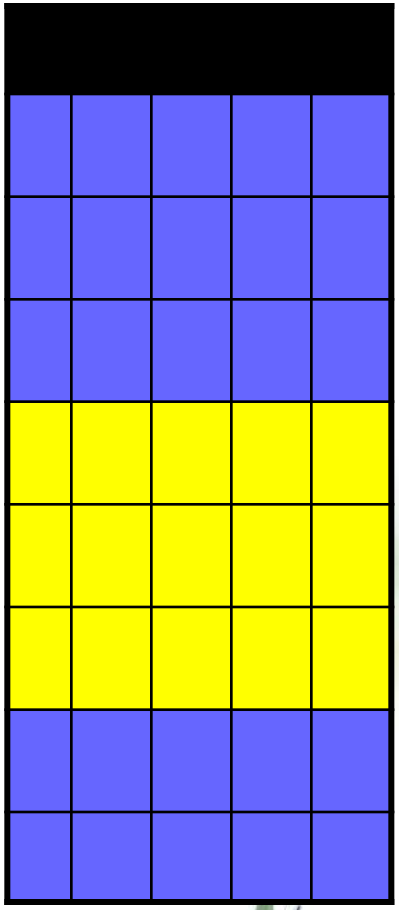
×

SC(10000行)

学号 Sno	课程号 Cno	成绩 Grade
05001	1	92
05001	2	85
05001	3	88
05002	2	90

=

(10000000行)



Q1: $\Pi_{Sname}(\sigma_{S.sno=SC.sno \wedge SC.Cno='2'}(S \times SC))$

1.计算笛卡尔积 S×SC

- 读一遍S表需读的总块数 = 1000/10 = 100块
- 读一遍SC表需读的总块数 = 10000/100 = 100块
- S表要分几次读入内存? 1000/50 = 20 次
- S表的每一部分读入内存，SC表都需读一遍,SC共需读入的块数
  = 100 ×20 = 2000 块
- 笛卡尔积运算读取的总块数 = 100 + 2000 = 2100块
- 读取时间T1 = 2100 /20 = 105 秒
- 中间结果大小 = 1000 * 10000
- 写中间结果的时间T2 = (1000 * 10000) / 10 /20 = 50000 秒

Q1(续):

2.作选择运算$\sigma_{S.sno=SC.sno \wedge SC.Cno='2'}(\ldots)$

– 只需将中间结果读取一遍，并依次判断，选取满足条件的元组(处理时间不计，只计读取时间)

– 读取数据的总时间T3 = 50000 秒

3.作投影运算

– 忽略不计

Q1总时间 = T1 + T2 + T3

= 105 秒 + 50000 秒 + 50000 秒 = 100105 秒

= 27.8 小时

Q2: $\Pi_{Sname}(\sigma_{SC.Cno='2'}(SC \bowtie S))$

S(1000行)

学号 Sno	姓名 Sname
05001	李勇
05002	刘晨

$\bowtie$

SC(10000行)

学号 Sno	课程号 Cno	成绩 Grade
05001	1	92
05001	2	85
05001	3	88
05002	2	90

= **10000行**

Q2: $\Pi_{Sname}(\sigma_{SC.Cno='2'}(SC \bowtie S))$

1.计算自然连接 $SC \bowtie S$

– 读取时间T1 = 2100 /20 = 105 秒(与Q1一样)

– 中间结果大小 = 10000

– 写中间结果的时间T2 = 10000 / 10 /20 = 50 秒

2.作选择运算$\sigma_{SC.Cno='2'}$()

– 读中间数据的时间T3 = 50 秒

3. 投影(时间忽略不计)

Q2总时间 = T1 + T2 + T3

= 105 秒 + 50 秒 + 50 秒 = 205 秒

Q3: $\Pi_{Sname}(S \bowtie \sigma_{SC.Cno='2'}(SC))$

S(1000行)

学号 Sno	姓名 Sname
05001	李勇
05002	刘晨

$\bowtie$

σ …(SC) (50行)

学号 Sno	课程号 Cno	成绩 Grade
05001	2	85
05002	2	90

= 50行

Q3: $\Pi_{Sname}(S \bowtie \sigma_{SC.Cno='2'}(SC))$

1.计算SC1 $\sigma_{SC.Cno='2'}(SC)$:读SC并选择

- 读SC时间T1 = 10000/100/20 = 5 秒
- 中间结果大小 = 50，不必写入外存，
- 写中间结果的时间忽略不计

2.计算S $\bowtie$SC1

- 读S时间T2 = 1000/10/20 = 5 秒
- 注：虽然S需分20次读入，且每读一部分都要遍历一遍SC1,但因SC1只有50个元组，全在内存，无I/O操作，这部分时间忽略不计。
- 计算的结果仍只有50个元组，不计I/O时间。

Q3(续):

3. 投影(时间忽略不计)

Q3总时间 = T1 + T2 = 5 秒 + 5 秒 = 10 秒

Q4: 假定SC在cno上建有索引，S表在sno上建有索引，如仍用Q3方法，则第1步选择时，根本不用读SC表全部，只需读取cno='2'的那些元组(50个),第2步也不需读取所有S表元组，只需读出相关的50个，这样总的时间可进一步大大减少(<10秒)。

对比

- Q1: 27.8小时
- Q2: 205秒
- Q3: 10秒
- Q4: <10秒

- 把代数表达式Q1变换为Q2、 Q3
  - 有选择和连接操作时，先做选择操作，可大大减少连接的元组，这是代数优化
- 在Q3/Q4中
  - SC表的选择操作算法有全表扫描和索引扫描2种方法，索引扫描方法较优
  - 对于Student和SC表的连接，利用Student表上的索引，采用索引连接代价较小,这就是物理优化

查询优化的任务

- 查询优化就是从许多策略中找出最高效的查询执行计划的处理过程。
- 优化可以发生在关系代数层面，即对于给定的表达式，系统尝试找出一个与之等价但执行效率更高的关系代数表达式。**（代数优化）**
- 优化也可以发生在操作执行层面，即对于给定的操作，系统为其选择一个高效的执行策略。**（物理优化）**

代 数 优 化

- 关系代数表达式等价变换规则
- 查询树的启发式优化

关系代数表达式等价变换规则

- 代数优化策略：通过对关系代数表达式的等价变换来提高查询效率

- 关系代数表达式的等价：指用相同的关系代替两个表达式中相应的关系所得到的结果是相同的

- 两个关系表达式E_1和E_2是等价的，可记为$E_1 \equiv E_2$

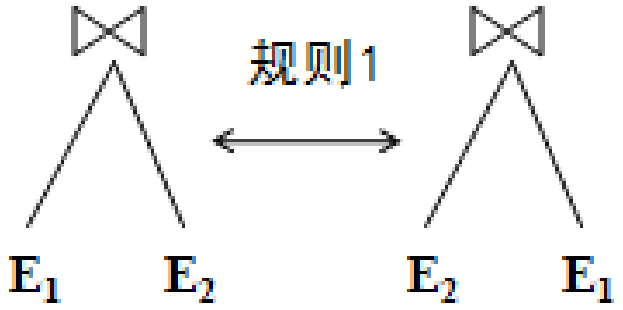
等价规则

- 规则1：连接、笛卡尔积的交换律

$$E_1 \times E_2 \equiv E_2 \times E_1$$

$$E_1 \bowtie E_2 = E_2 \bowtie E_1$$

$$E_1 \underset{F}{\bowtie} E_2 = E_2 \underset{F}{\bowtie} E_1$$



E：关系代数表达式　　F：连接条件

等价规则

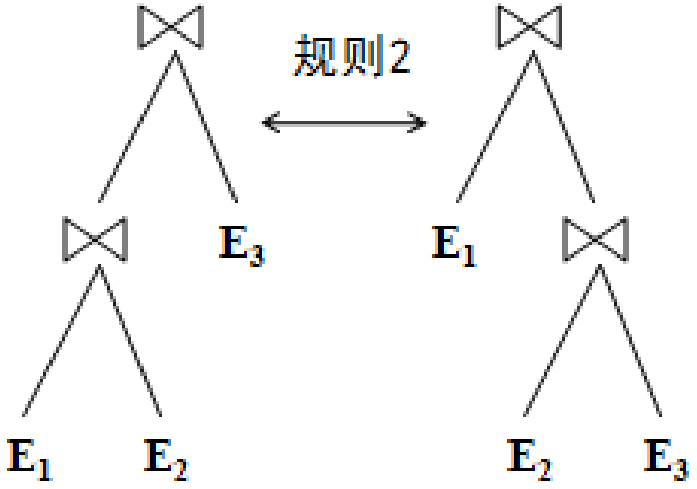
Student(sno,sname,sage….)
Course(cno,cname,….)
Sc(sno,cno,grade)

- **规则2：连接、笛卡尔积的结合律**

$$(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$$

$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

$$(E_1 \underset{F}{\bowtie} E_2) \underset{F}{\bowtie} E_3 \equiv E_1 \underset{F}{\bowtie} (E_2 \underset{F}{\bowtie} E_3)$$



例：已知学生数1000，课程数100，选课数10000，学生记录长度>课程记录长度，试对 (课程⋈学生)⋈选课 做等价变换，使中间结果集最小。

(课程⋈学生)⋈选课 ≡ (学生⋈课程)⋈选课 ≡ 学生⋈(课程⋈选课)

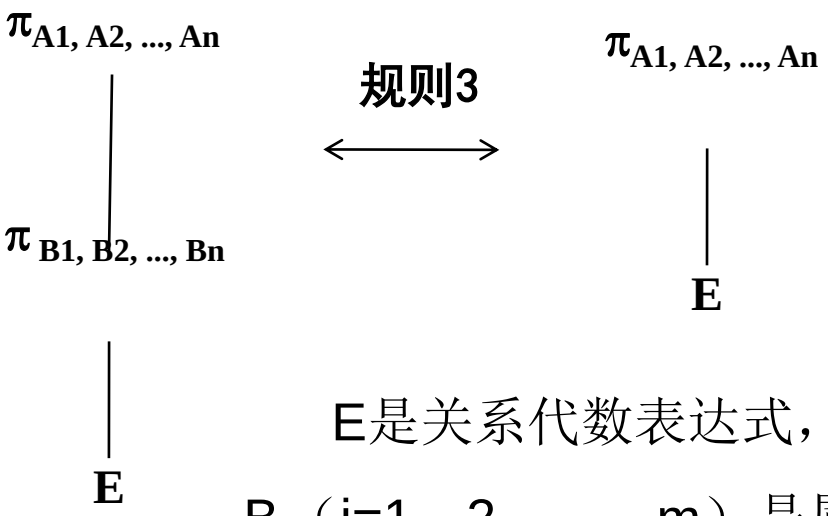
等价规则

Student(sno,sname,sage……)

$$\pi_{Sname}(\pi_{Sname,Sage}(S)) \equiv \pi_{Sname}(S)$$

- 规则3：投影的串接定律

$$\pi_{A1, A2, ..., An}(\pi_{B1, B2, ..., Bn}(E)) \equiv \pi_{A1, A2, ..., An}(E)$$



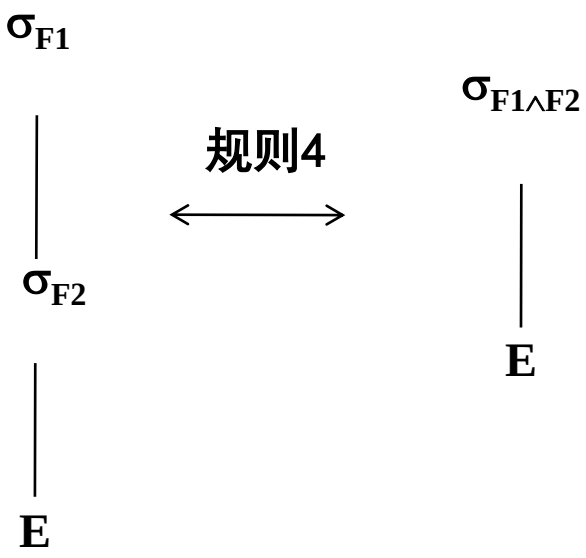
E是关系代数表达式，A_i（i=1，2，…，n），B_j（j=1，2，…，m）是属性名且{A_1，A_2，…A_n}是{B_1，B_2，…，B_m}的子集。

等价规则

- 规则4:选择的串接定律

$$\delta_{Sdept='IS'}(\delta_{Sage>19}(S)) \equiv \delta_{Sdept='IS' \wedge Sage>19}(S)$$

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

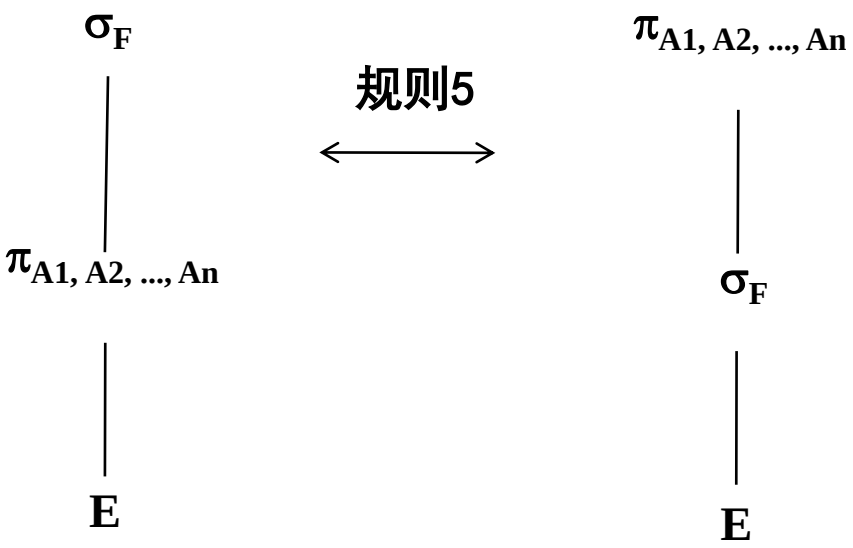


等价规则

$$\delta_{Sage>19}(\pi_{Sname,Sage}(S)) \equiv \pi_{Sname,Sage}(\delta_{Sage>19}(E))$$

- 规则5：选择和投影的交换律

$\sigma_F(\pi_{A1,A2,...,An}(E)) = \pi_{A1,A2,...,An}(\sigma_F(E))$　　此时，条件F只涉及属性组A。

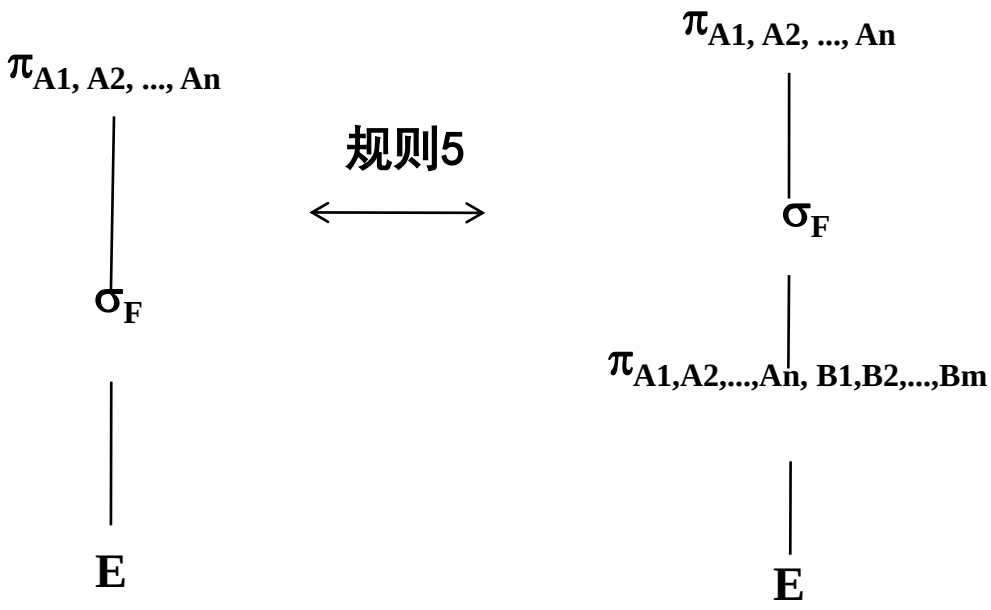


等价规则

$$\pi_{Sname}(\delta_{Sage>19}(S)) \equiv \pi_{Sname}(\delta_{Sage>19}(\pi_{Sname,Sage}(E)))$$

- 规则5：选择和投影的交换律（特殊情况）若条件中有不属于A的属性组B

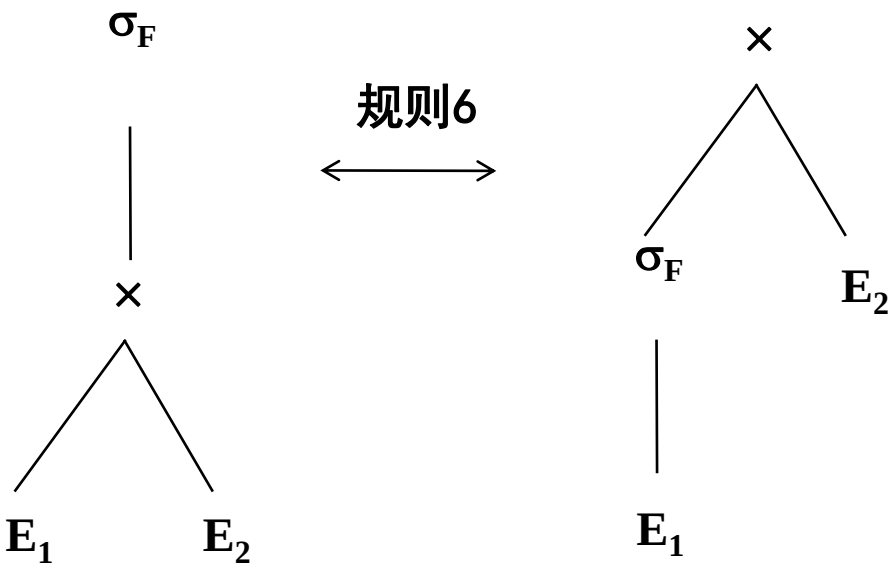
$$\pi_{A1,A2,...,An}(\sigma_F(E)) \equiv \pi_{A1,A2,...,An}(\sigma_F(\pi_{A1,A2,...,An,\ B1,B2,...,Bm}(E)))$$



等价规则

- 规则6:选择与笛卡尔积的交换律（续）

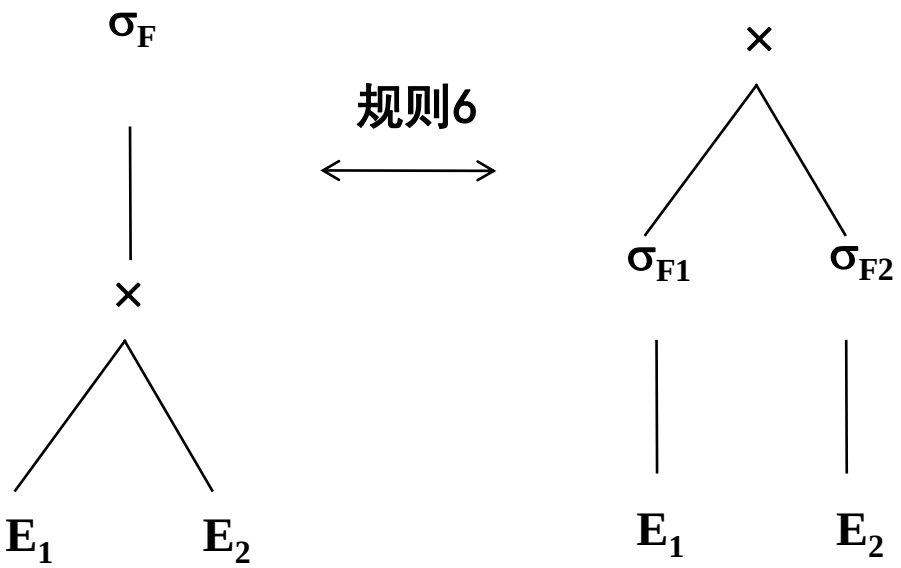
$$\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2 \qquad F\text{仅和}E_1\text{有关}$$



等价规则

- 规则6:选择与笛卡尔积的交换律（续）

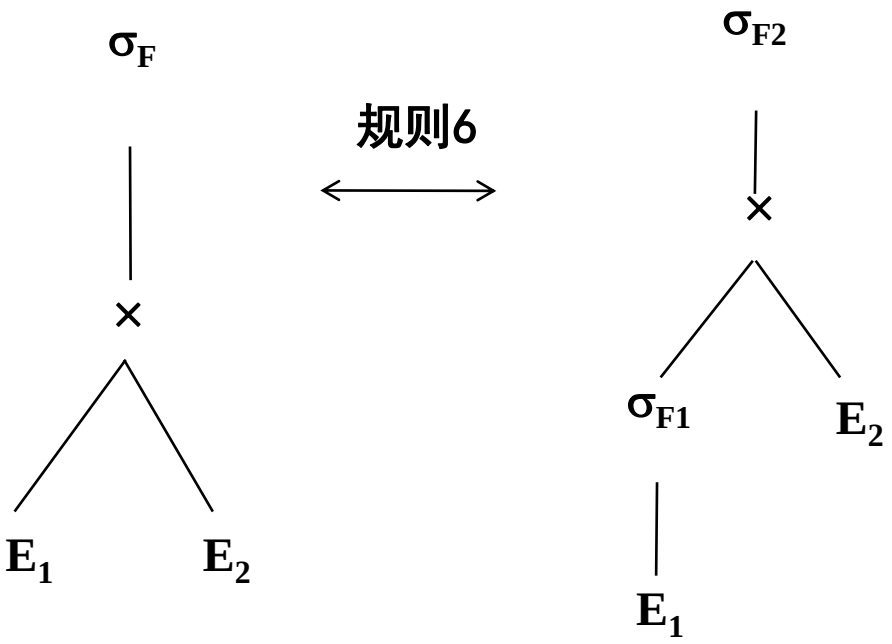
$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F1}(E_1) \times \sigma_{F2}(E_2) \qquad F = F1 \wedge F2$$



等价规则

- 规则6:选择与笛卡尔积的交换律（续）

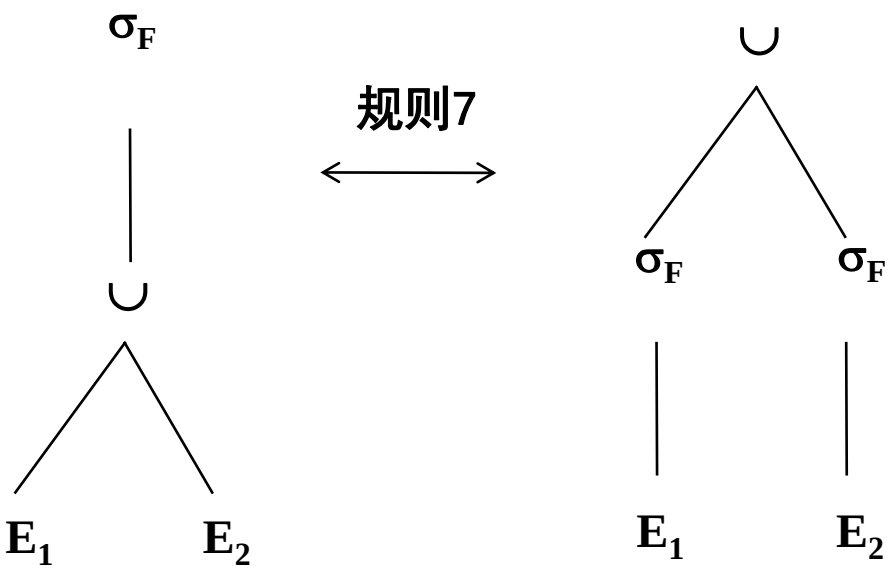
$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F2}(\sigma_{F1}(E_1) \times E_2) \quad F1\text{--}E1 \quad F2\text{--}E1E2$$



等价规则

- 规则7：选择与并的分配律

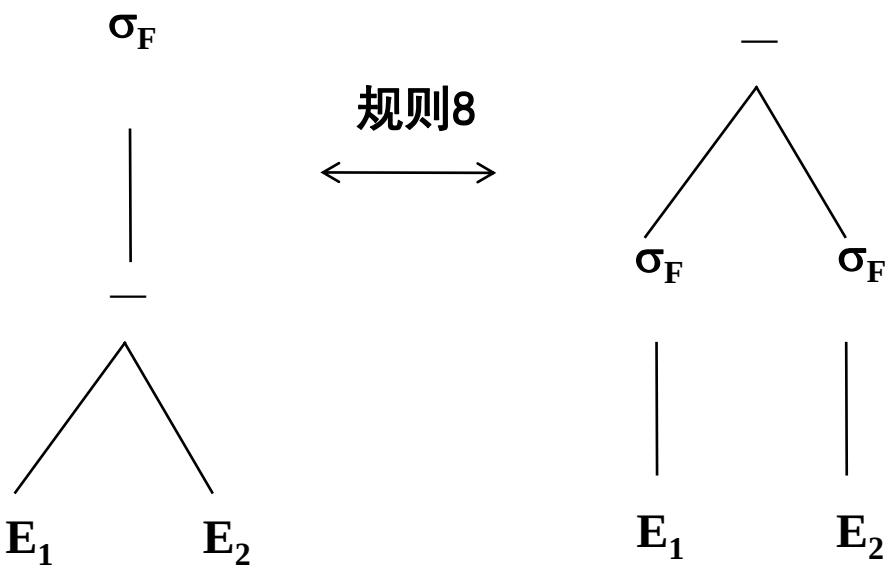
$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$$



等价规则

- 规则8：选择与差的分配律

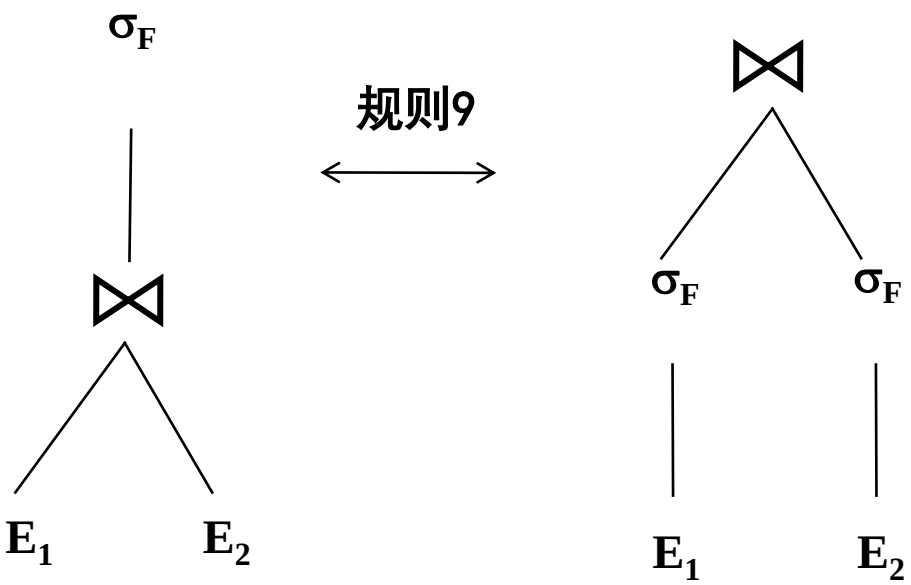
$$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$$



等价规则

- 规则9：选择对自然连接的分配律

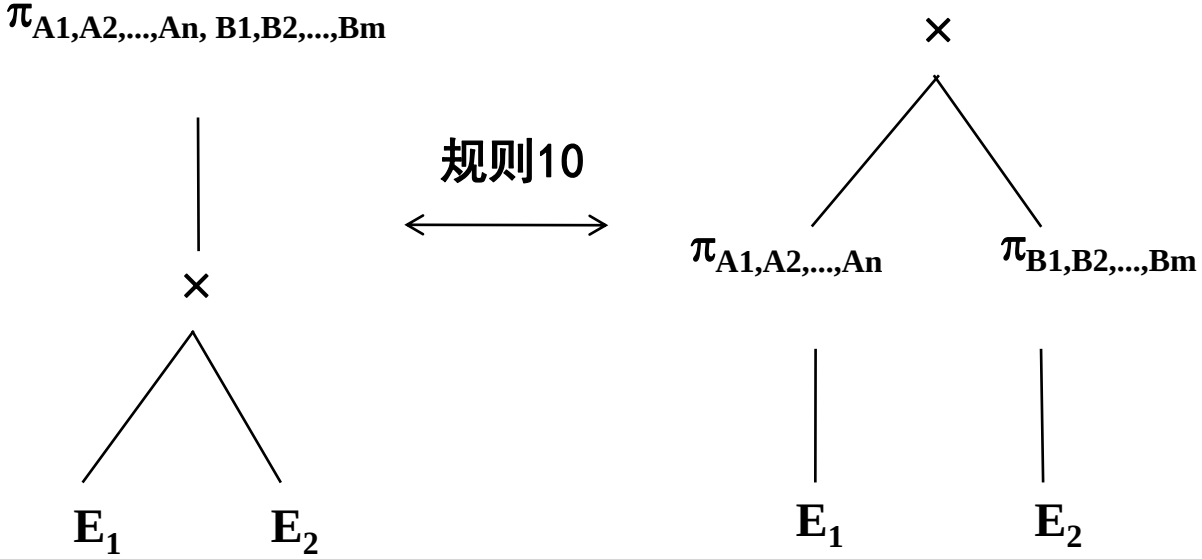
$$\sigma_F(E_1 \bowtie E_2) \equiv \sigma_F(E_1) \bowtie \sigma_F(E_2)$$



等价规则

- 规则10:投影与笛卡尔积的分配律

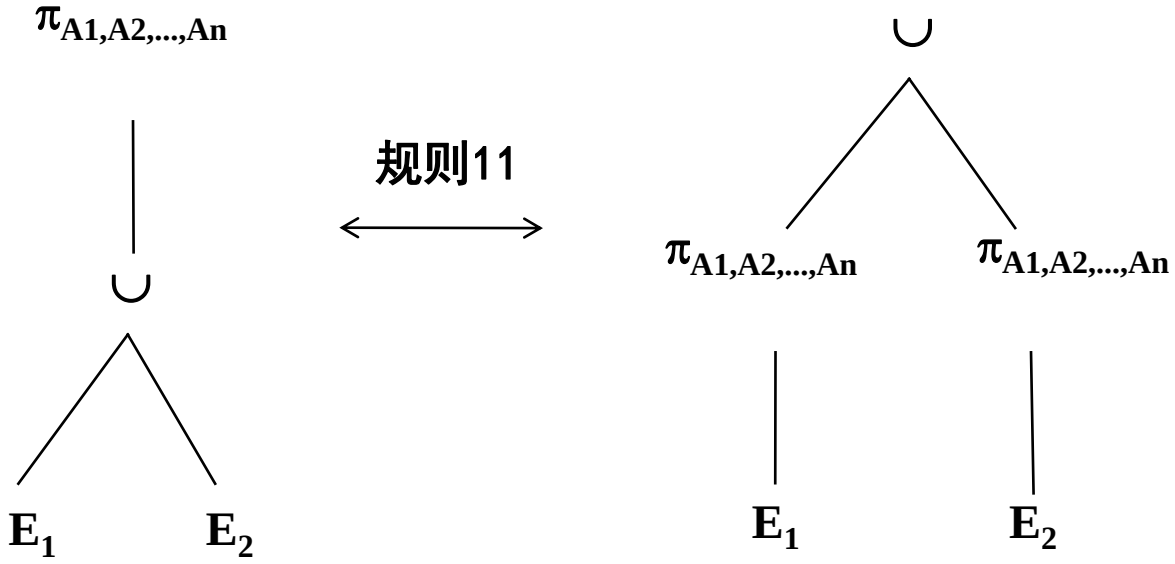
$$\pi_{A1,A2,...,An,\ B1,B2,...,Bm}(E_1 \times E_2) \equiv \pi_{A1,A2,...,An}(E_1) \times \pi_{B1,B2,...,Bm}(E_2)$$



等价规则

- 规则11：投影与并的分配律

$$\pi_{A1,A2,...,An}(E_1 \cup E_2) \equiv \pi_{A1,A2,...,An}(E_1) \cup \pi_{A1,A2,...,An}(E_2)$$



小结

1-2: 连接、笛卡尔积的交换律、结合律

3： 合并或分解投影运算

4： 合并或分解选择运算

5-9： 选择运算与其他运算交换

5，10，11： 投影运算与其他运算交换

代 数 优 化

- 关系代数表达式等价变换规则
- 查询树的启发式优化

查询树的启发式优化

查询优化的基本规则有

1.选择运算应尽可能先做

目的：减小中间关系

2.投影运算和选择运算同时做

目的：避免重复扫描关系

查询树的启发式优化

3.将投影运算与其前面或后面的双目运算结合

目的：减少扫描关系的遍数

4.某些选择运算＋在其前面的笛卡尔积 ==> 连接运算

例：$\sigma_{Student.Sno=SC.Sno}(Student \times SC)$

⇩

$Student \bowtie SC$

5.提取公共子表达式

当查询的是视图时，定义视图的表达式就是公共子表达式的情况。

查询树的启发式优化

- 遵循这些启发式规则，应用等价变换公式来优化关系表达式的算法。

  算法：关系表达式的优化

  输入：一个关系表达式的查询树

  输出：优化的查询树

  方法：

  (1) 利用等价变换规则4把形如$\sigma_{F1\wedge F2\wedge\ldots\wedge Fn}(E)$变换为$\sigma_{F1}(\sigma_{F2}(\ldots(\sigma_{Fn}(E))\ldots))$。

  (2) 对每一个选择，利用等价变换规则4～9尽可能把它移到树的叶端。

查询树的启发式优化

(3) 对每一个投影利用等价变换规则3，5，10，11中的一般形式尽可能把它移向树的叶端。

– 注意：

➢等价变换规则3使一些投影消失

➢规则5把一个投影分裂为两个，其中一个有可能被移向树的叶端

(4) 利用等价变换规则3～5把选择和投影的串接合并成单个选择、单个投影或一个选择后跟一个投影。使多个选择或投影能同时执行，或在一次扫描中全部完成

查询树的启发式优化

（5）对内结点分组

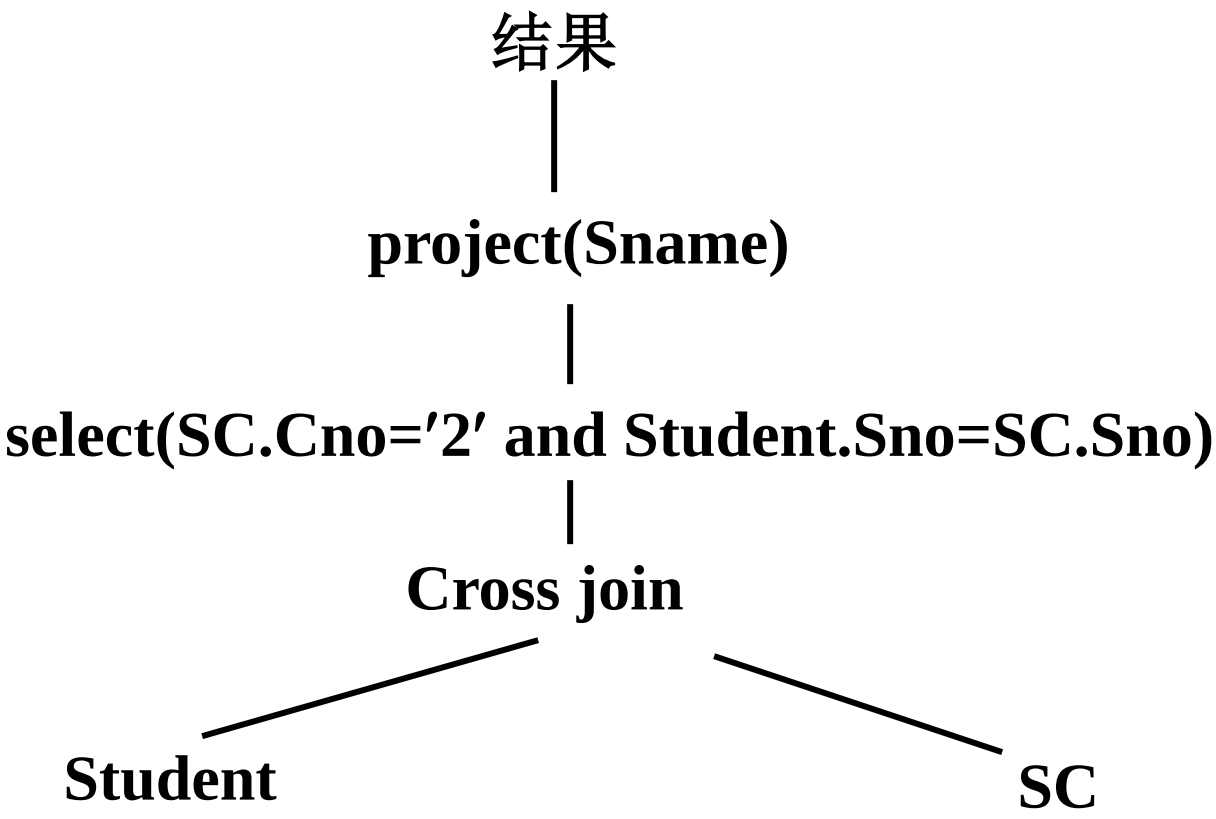
- 把上述得到的语法树的内节点分组。
- 每一双目运算(×,⋈，∪，－)和它所有的直接祖先为一组(这些直接祖先是б，π运算)。
- 如果其后代直到叶子全是单目运算，则也将它们并入该组，但当双目运算是笛卡尔积(×)，而且其后的选择不能与它结合为等值连接时除外。把这些单目运算单独分为一组。

（6）生成程序

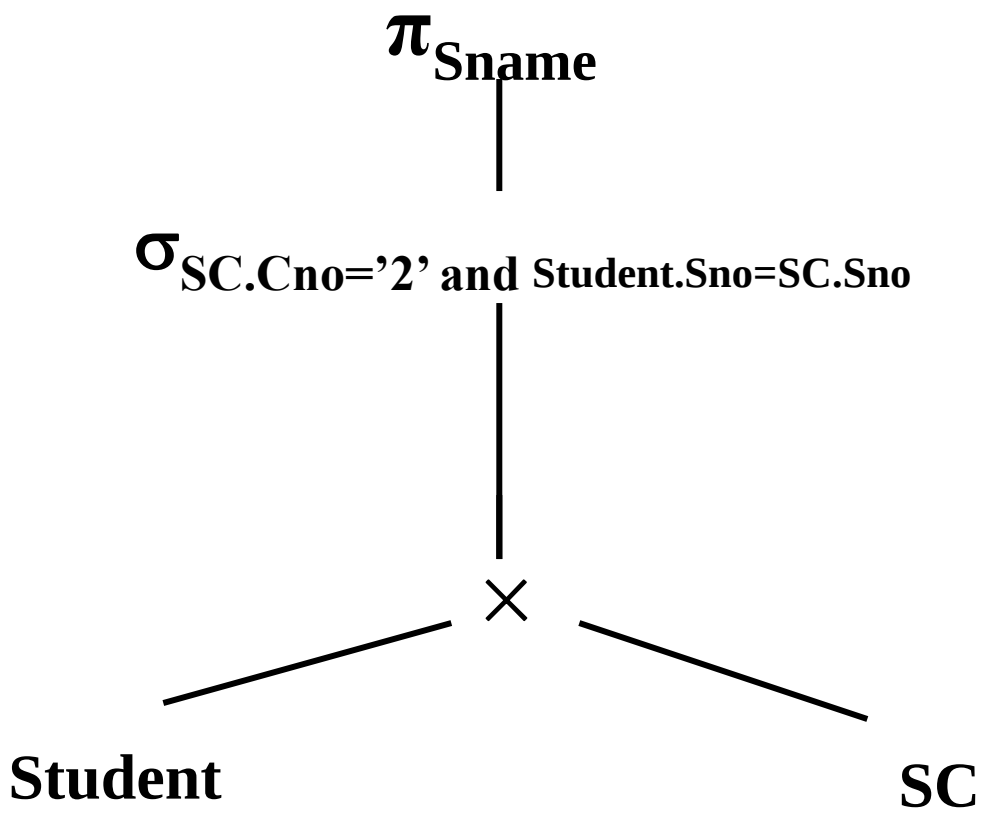
- 生成一个程序，每组结点的计算是程序中的一步。
- 各步的顺序是任意的，只要保证任何一组的计算不会在它的后代组之前计算。

Select sname from student,sc where student.sno=sc.sno and cno= ′2′

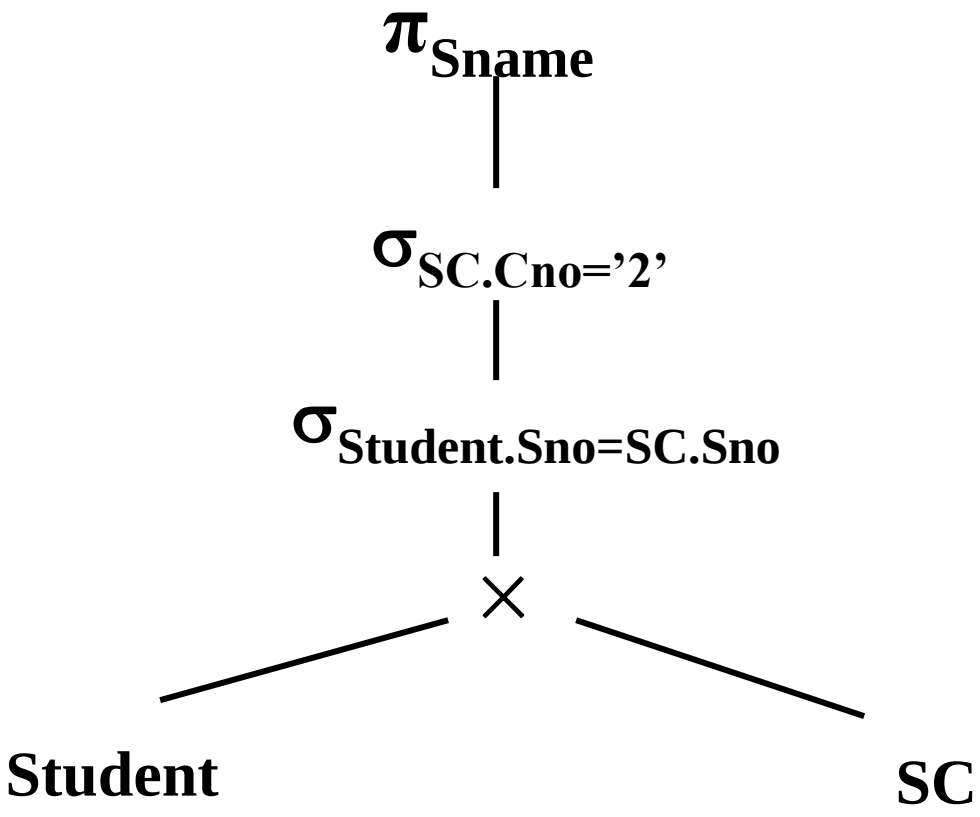
（1）查询树



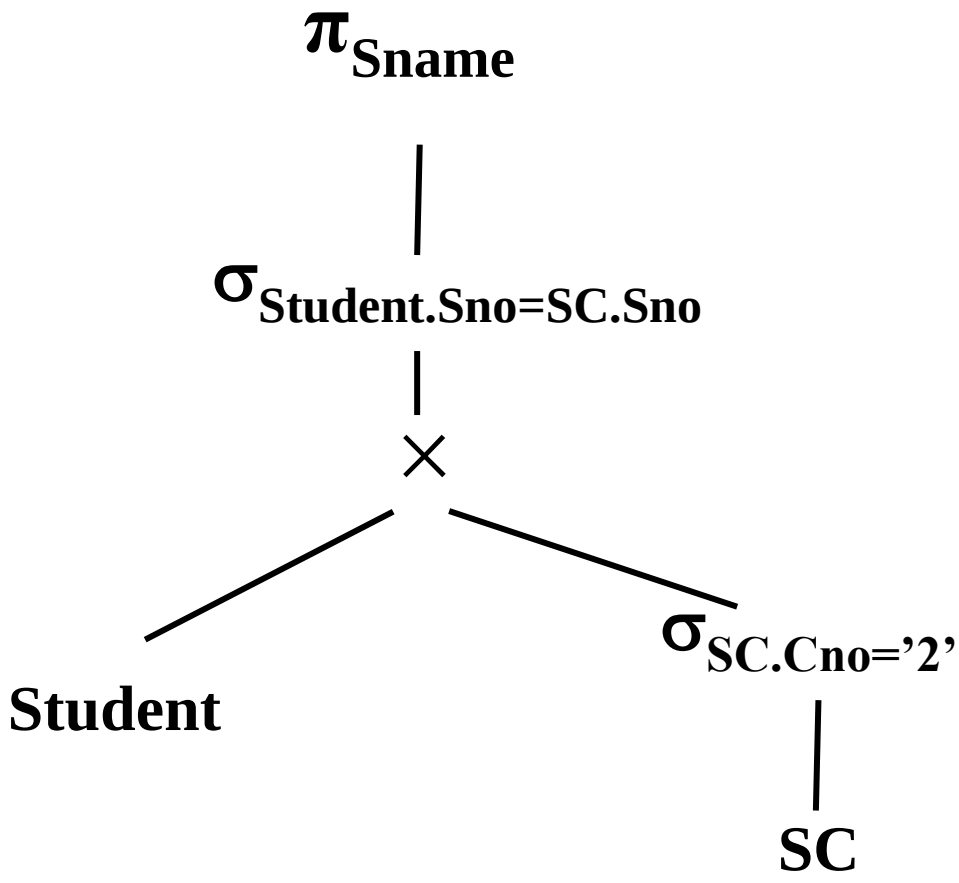
（2）关系代数语法树



（3）分解选择运算

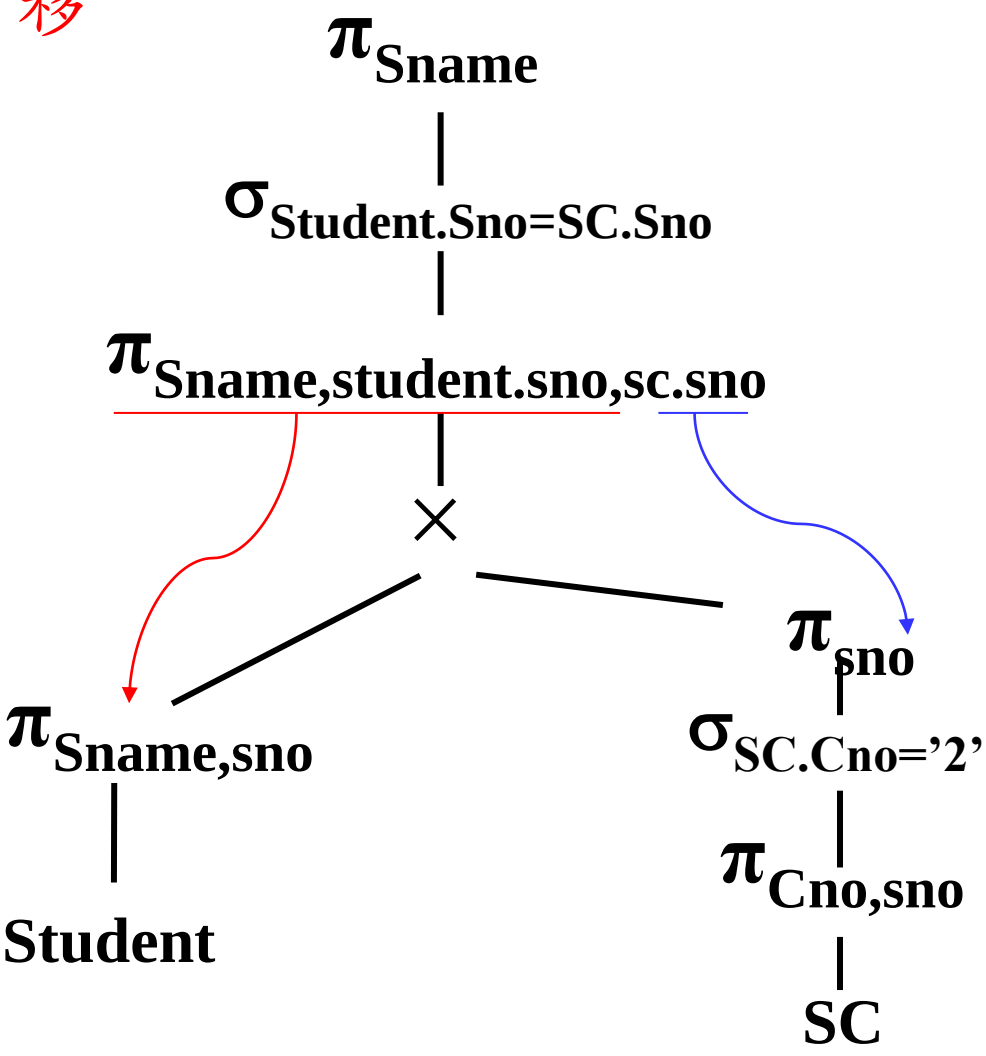


（4）选择下移



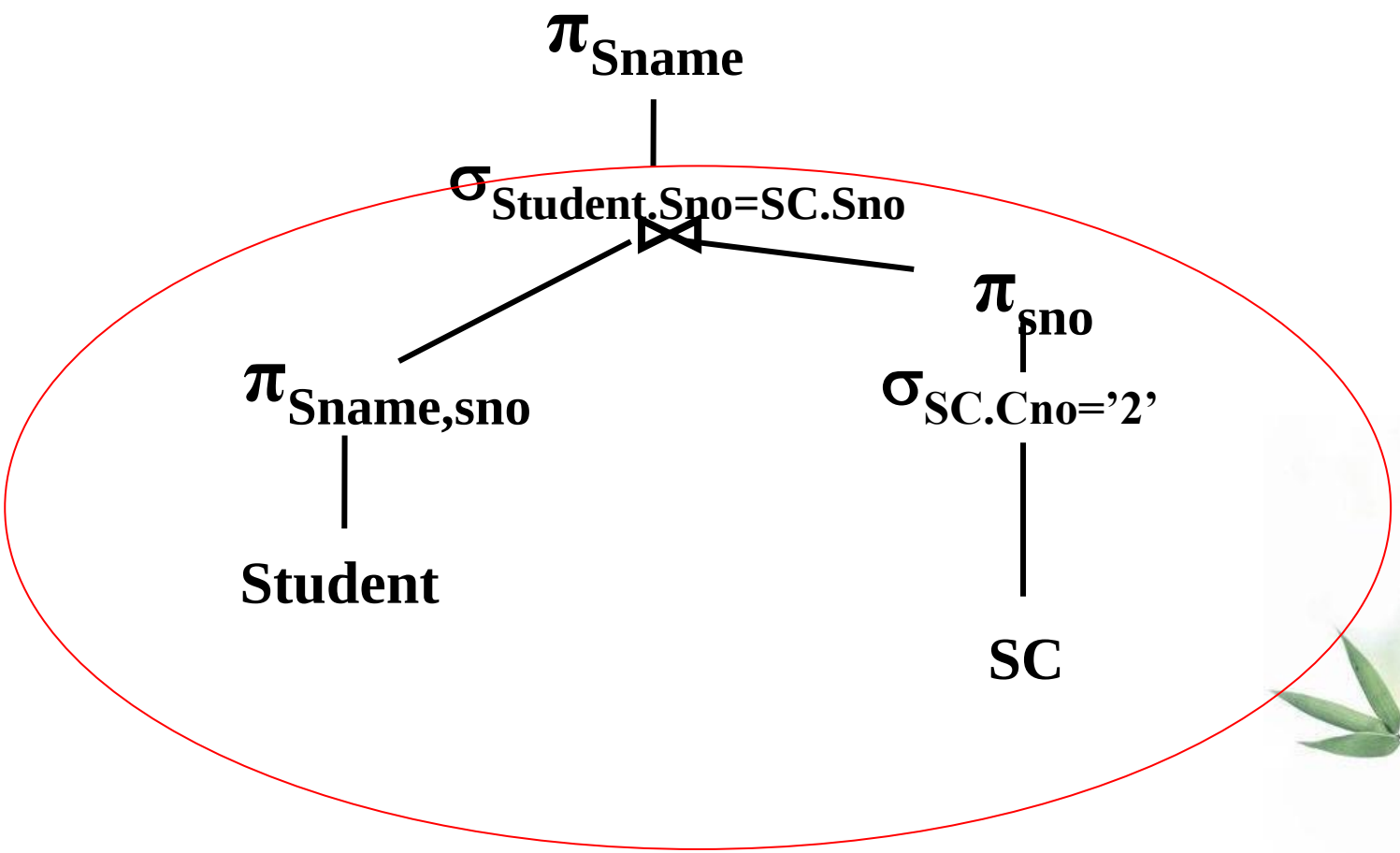
考虑投影的下推细节

（5）投影下移



考虑投影的下推细节

（6）串接合并与分组



优化示例

例：对如下关系代数表达式进行优化。

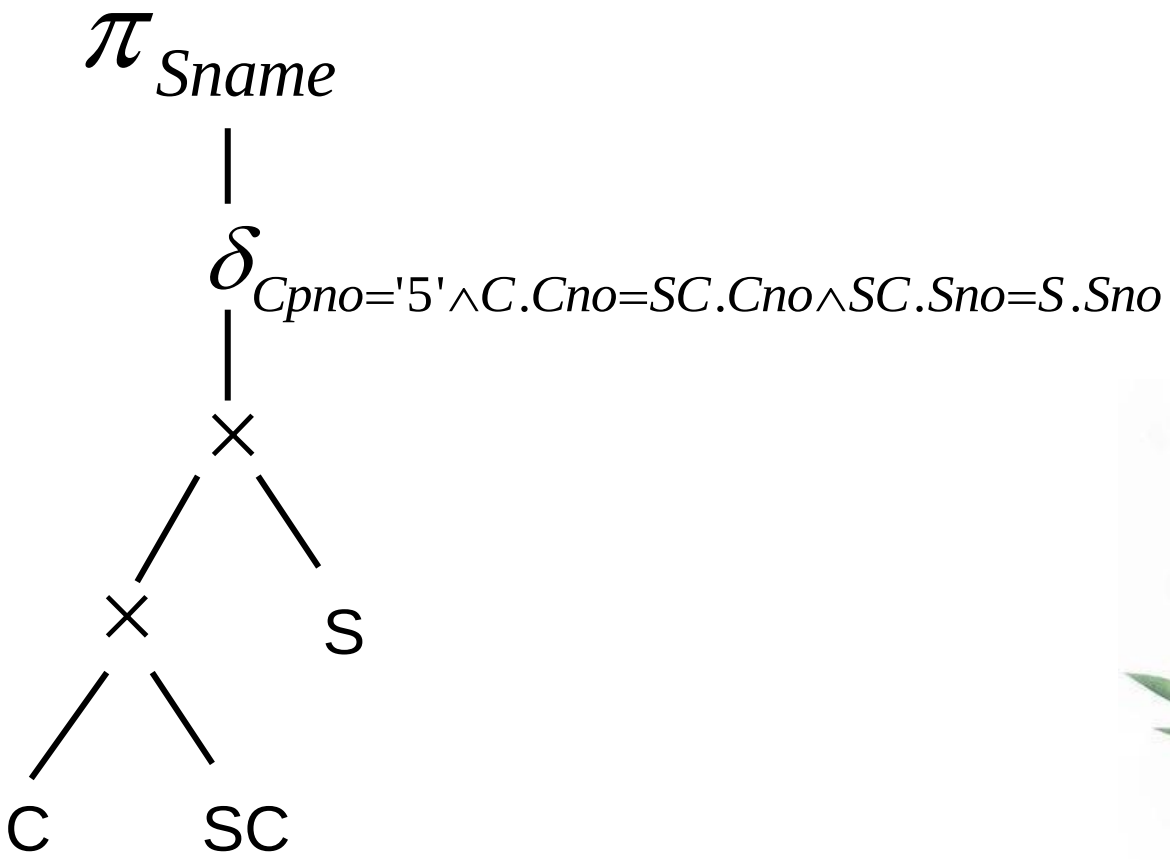
$$\pi_{Sname}(\delta_{Cpno='5'}(C \triangleright\!\triangleleft SC \triangleright\!\triangleleft S))$$

其中，C是课程表，S是学生表，SC是学生选课表。

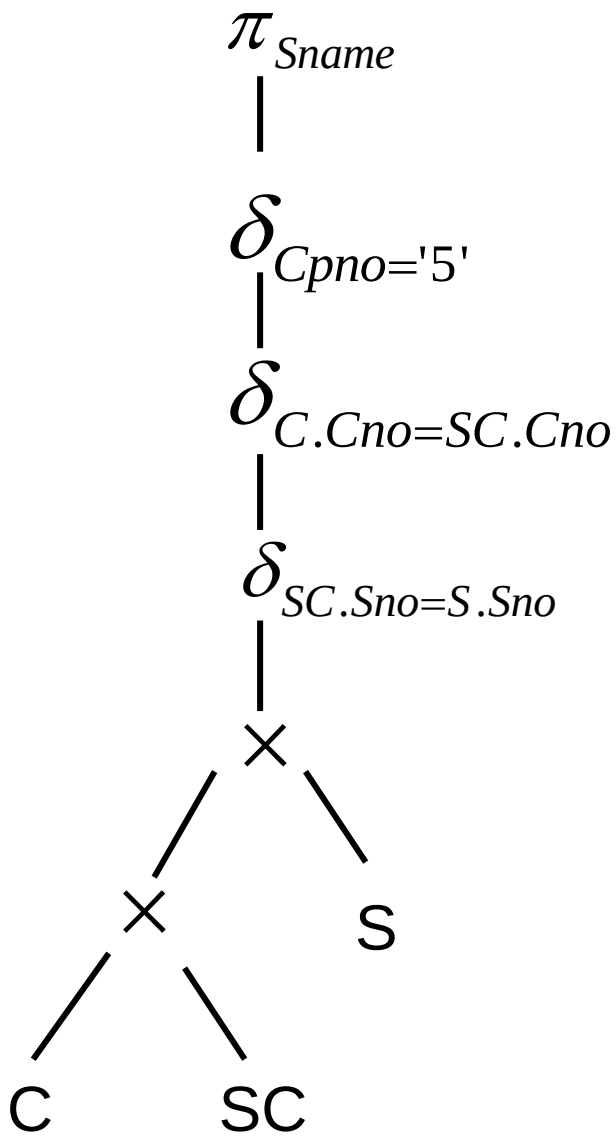
先把自然连接分解为笛卡儿积和选择。

分解后的关系代数表达式

$$\pi_{Sname}(\delta_{Cpno='5'\wedge C.Cno=SC.Cno\wedge SC.Sno=S.Sno}(C\times SC\times S))$$



第一步：利用规则4分解选择运算

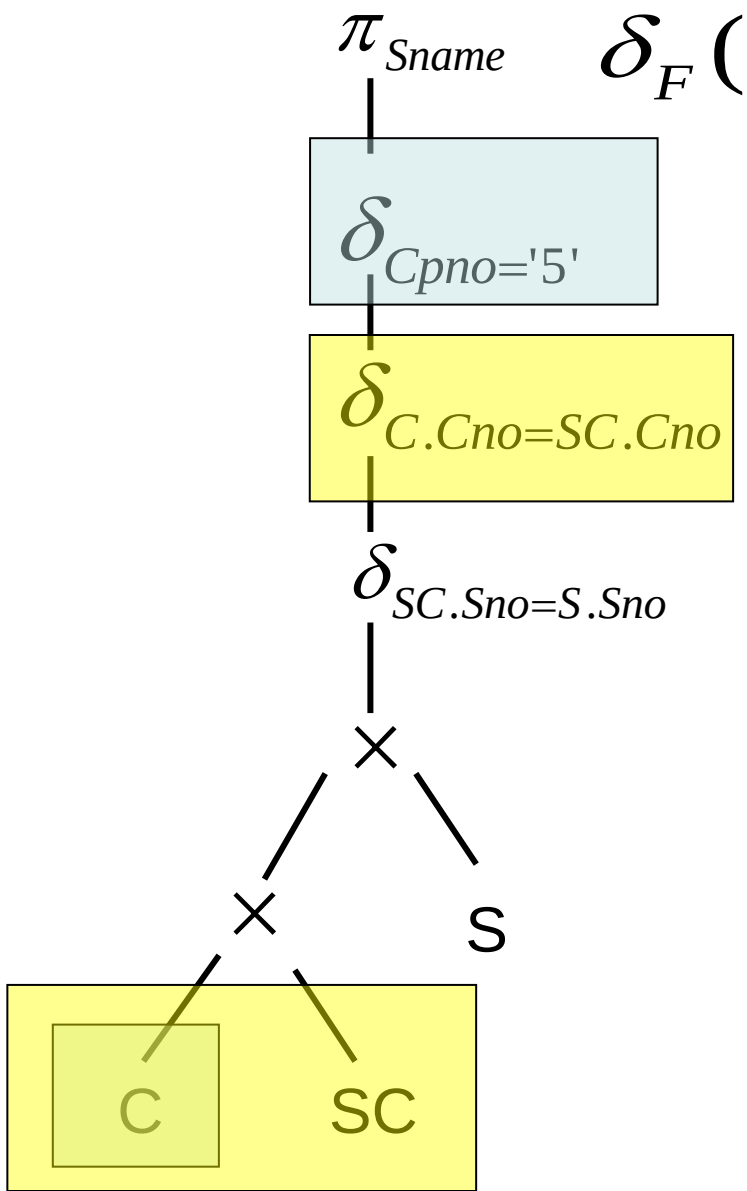


$$\delta_{F_1}(\delta_{F_2}(E)) \equiv \delta_{F_1 \wedge F_2}(E)$$

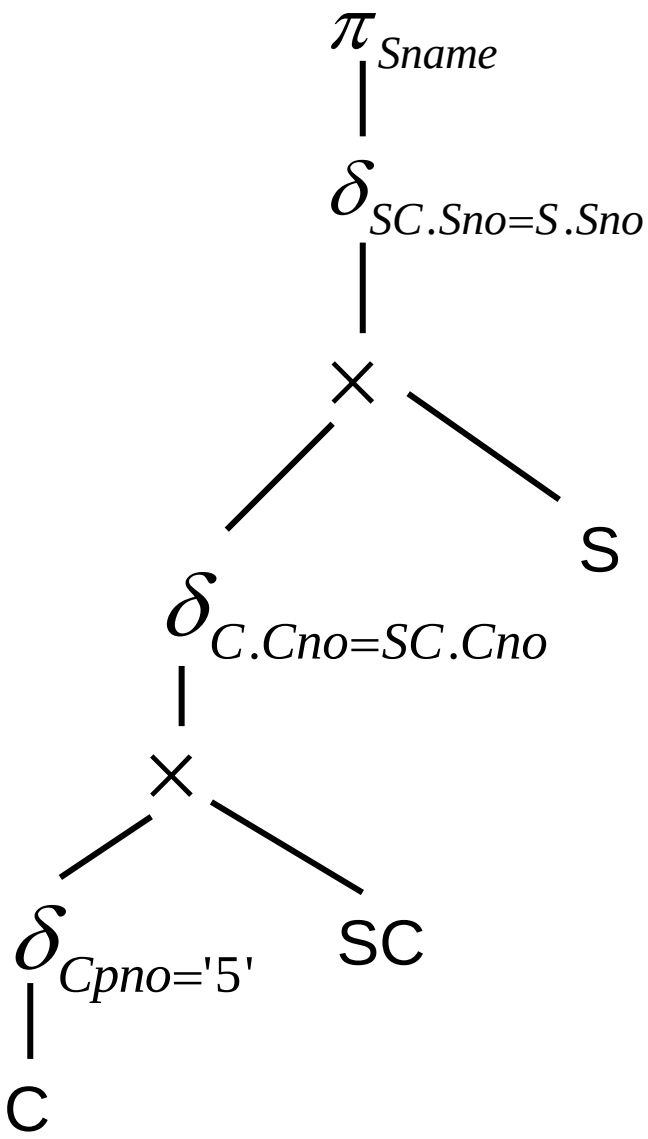
$$\delta_{F_1 \wedge F_2}(E) \equiv \delta_{F_1}(\delta_{F_2}(E))$$

第二步：尽量下放选择运算

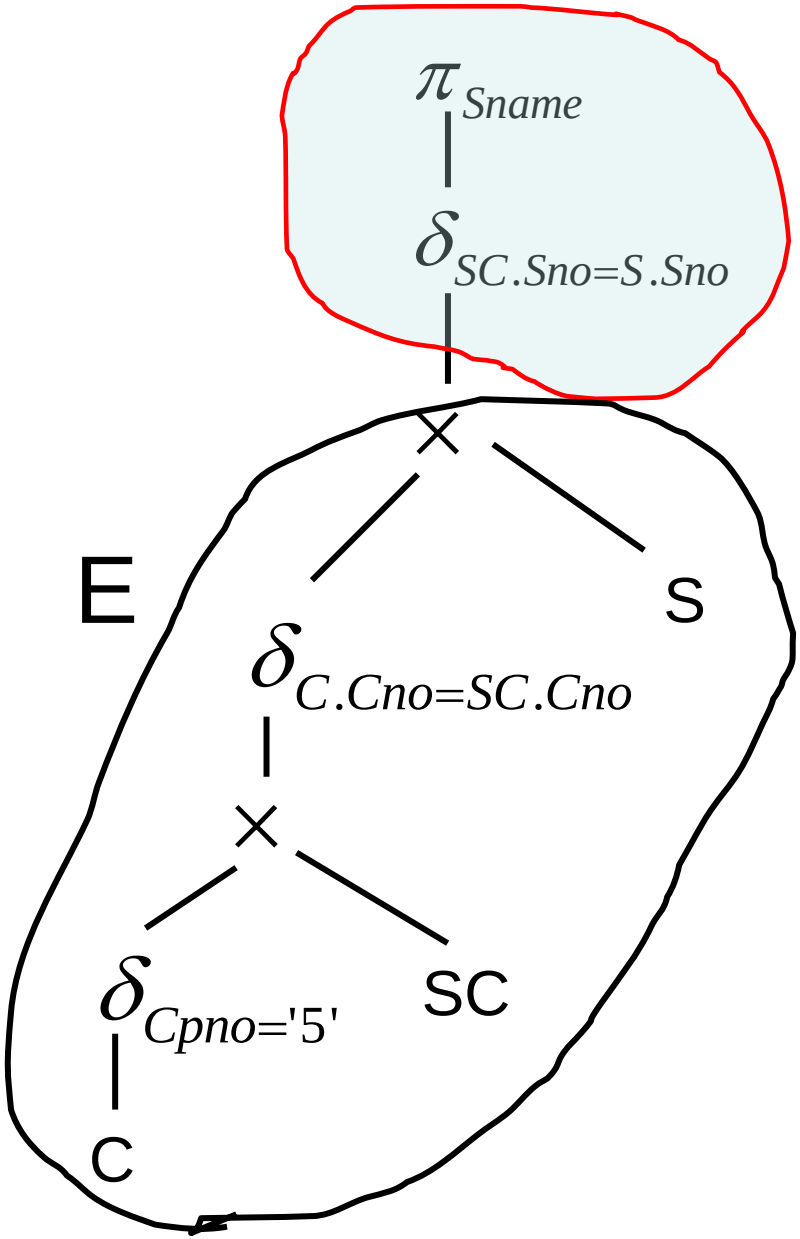
$$\delta_F(E_1 \times E_2) \equiv \delta_F(E_1) \times E_2$$



第二步（2）：下放完成后：



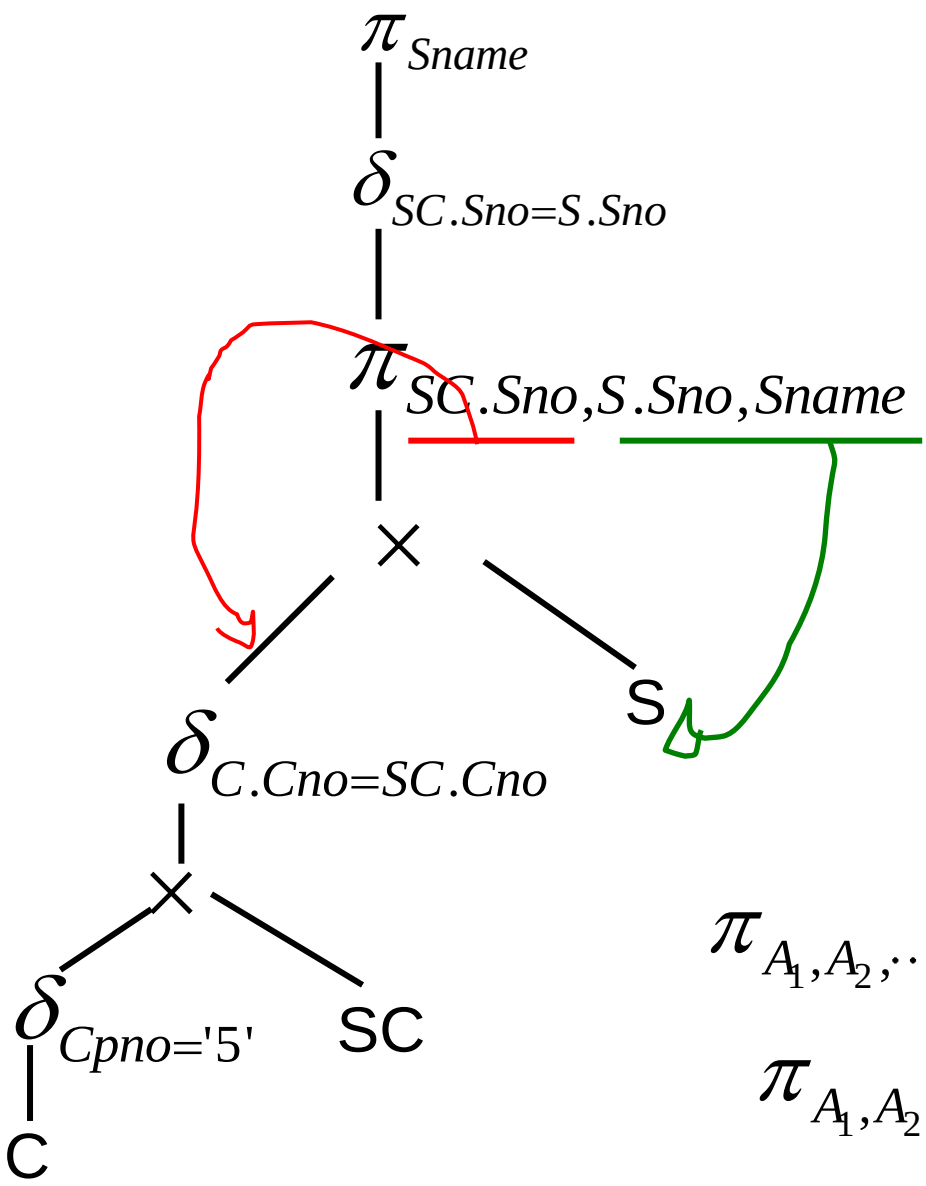
第三步：尽量下放投影运算



$$\pi_{Sname}(\delta_{SC.Sno=S.Sno}(E) \equiv$$

$$\pi_{Sname}(\delta_{SC.Sno=S.Sno}(\pi_{SC.Sno,S.Sno,Sname}(E))$$

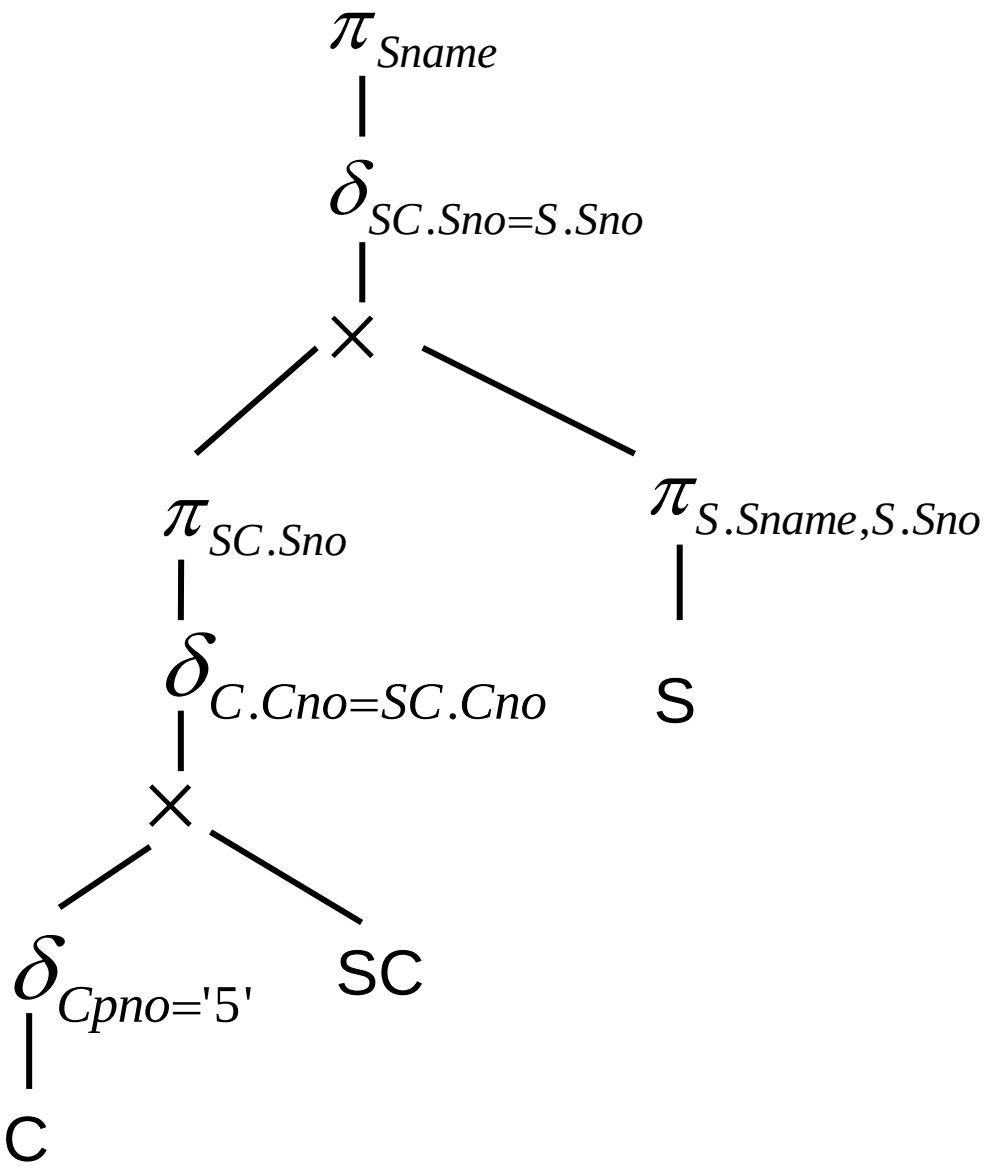
第三步：尽量下放投影运算



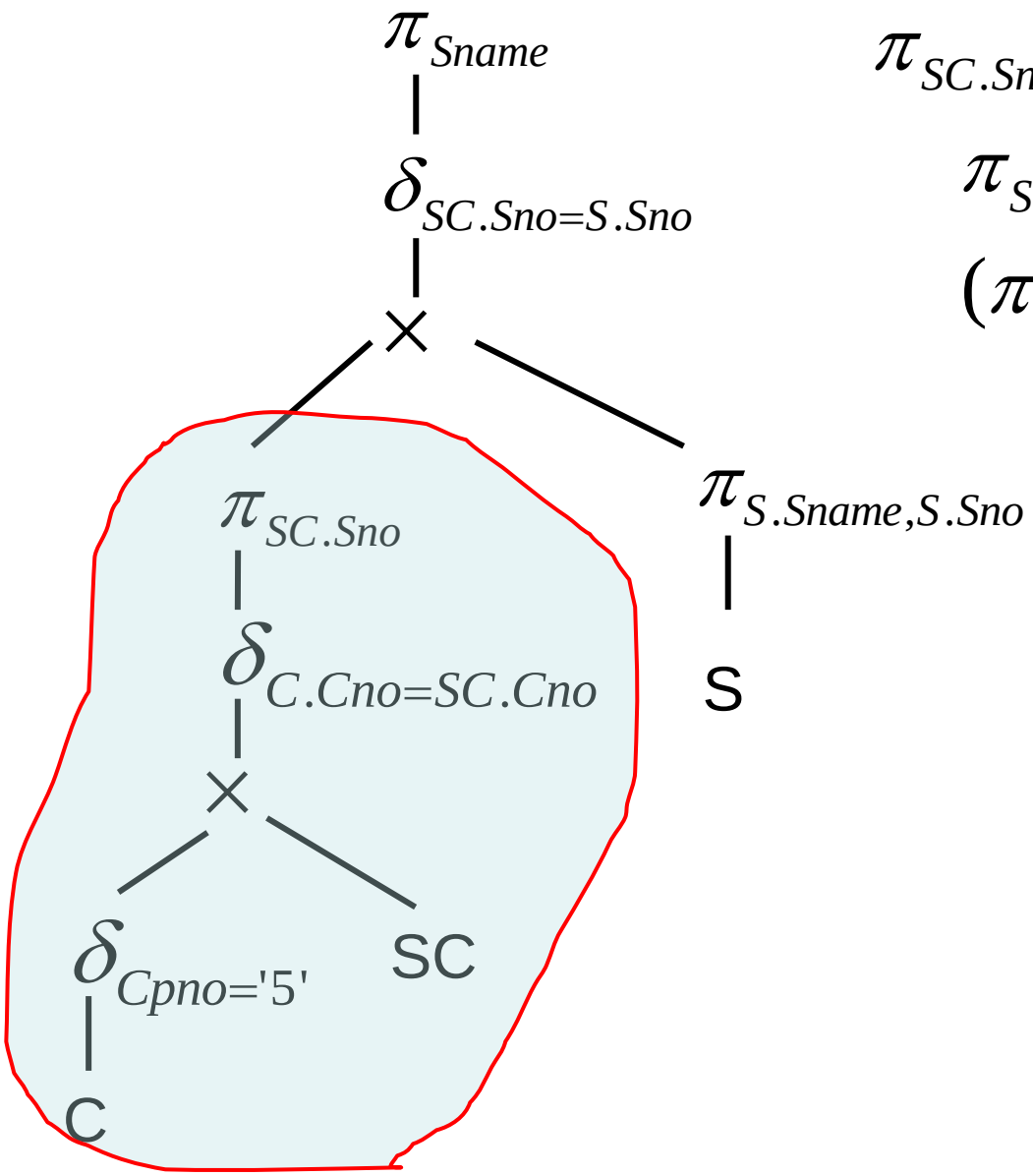
$$\pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E_1 \times E_2) \equiv$$

$$\pi_{A_1,A_2,\cdots,A_n}(E_1) \times \pi_{B_1,B_2,\cdots,B_m}(E_2)$$

第三步（2）：第一次下放后：

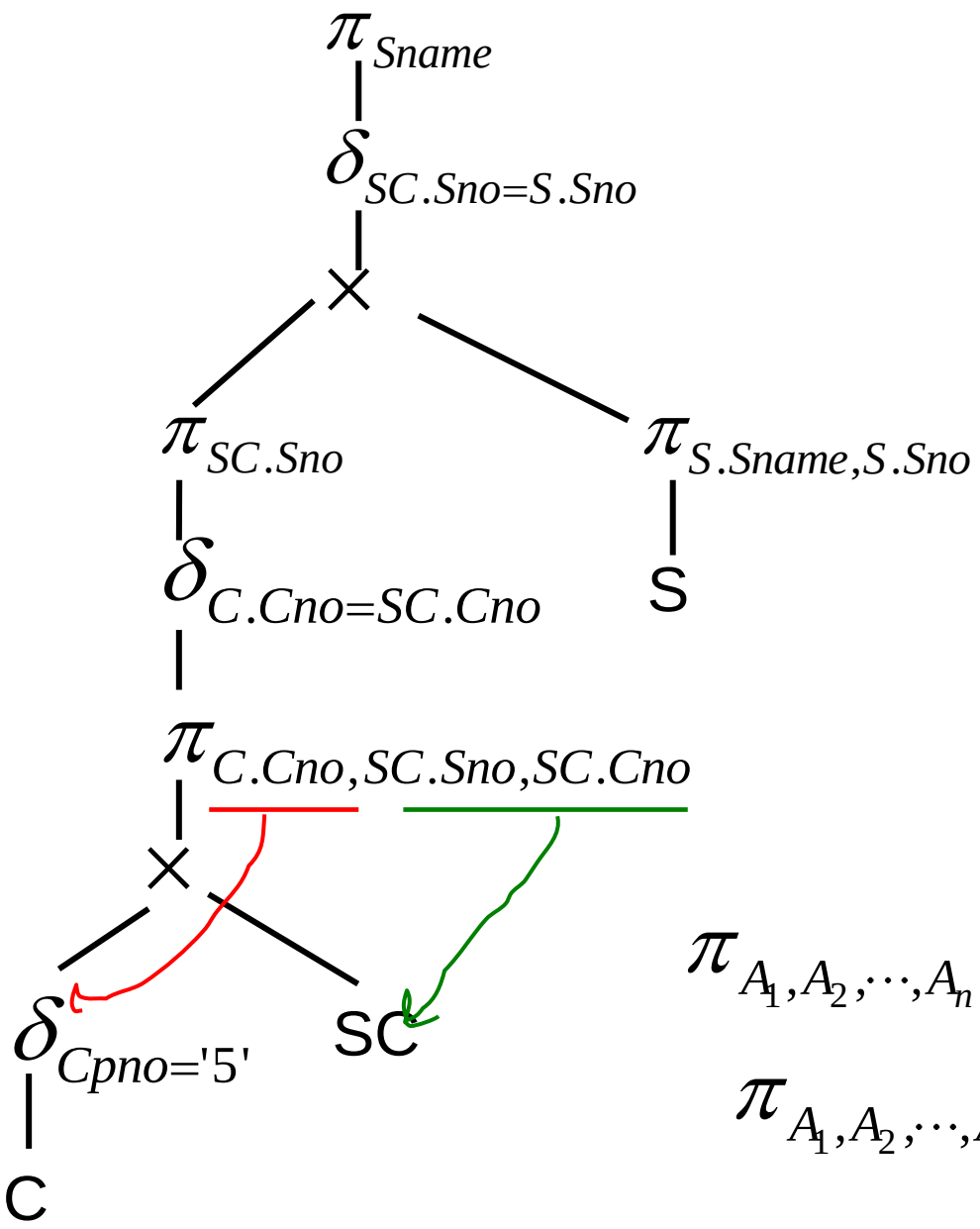

π_{Sname}
$\delta_{SC.Sno=S.Sno}$
×
$\pi_{SC.Sno}$
$\pi_{S.Sname,S.Sno}$
$\delta_{C.Cno=SC.Cno}$
S
×
$\delta_{Cpno='5'}$
SC
C

第三步（3）：第二次下放：



$$\pi_{SC.Sno}(\delta_{C.Cno=SC.Cno}(E) \equiv \pi_{SC.Sno}(\delta_{C.Cno=SC.Cno}((\pi_{C.Cno,SC.Sno,SC.Cno}(E)))$$

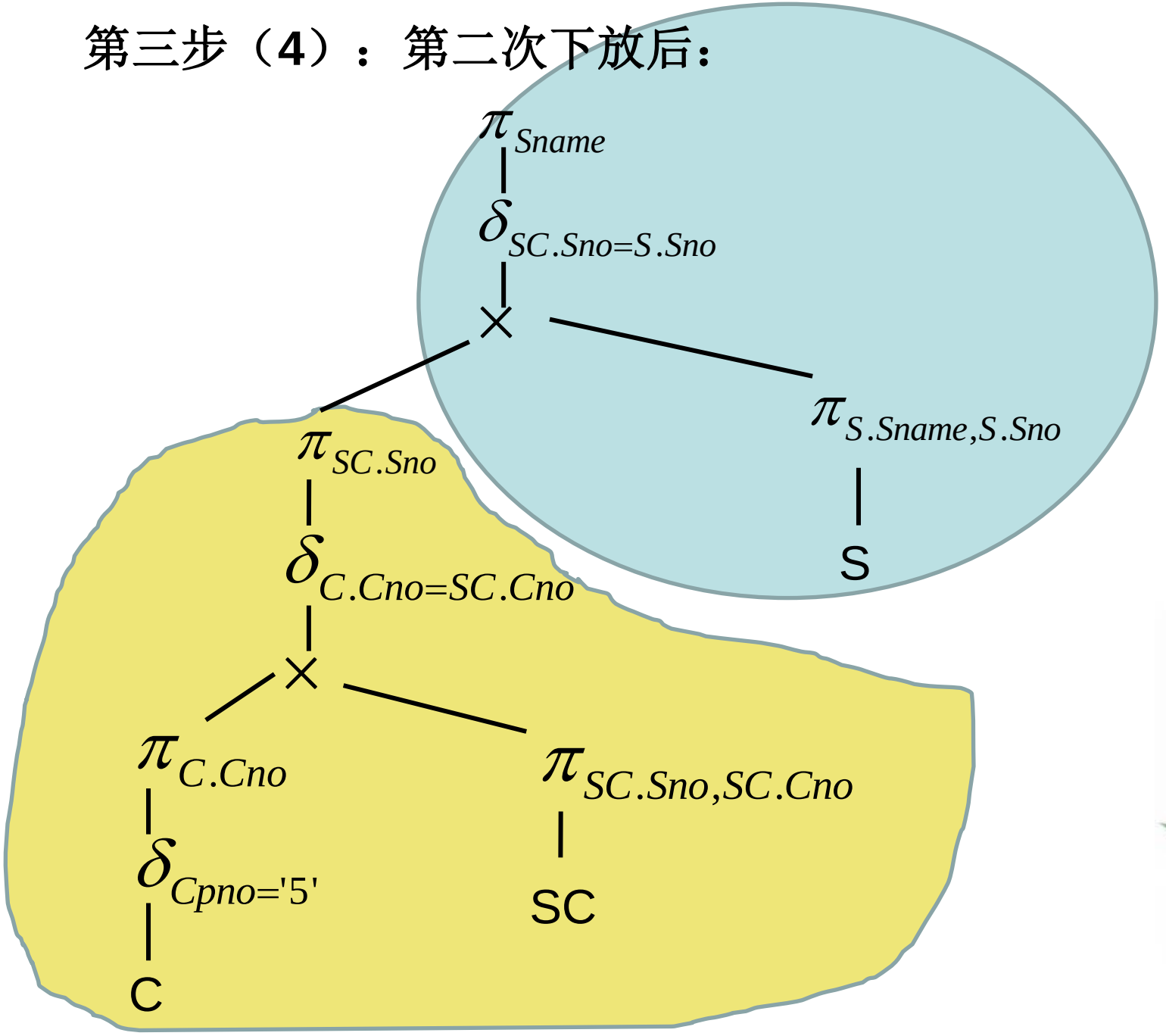
第三步（3）：第二次下放：

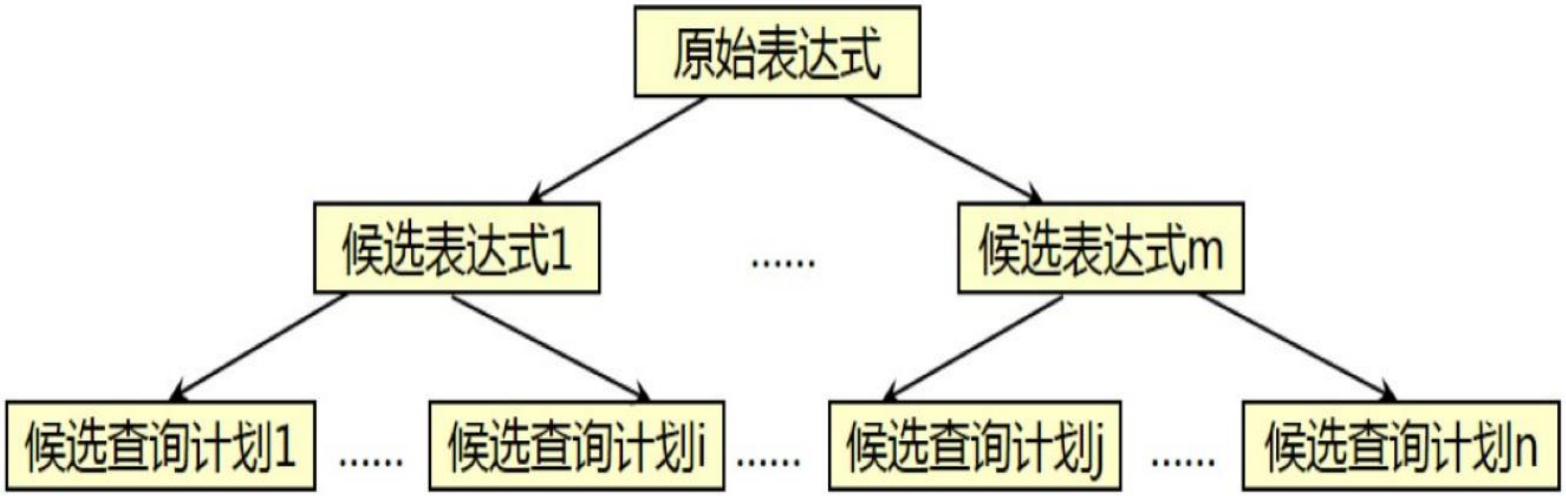


$$\pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E_1 \times E_2) \equiv$$

$$\pi_{A_1,A_2,\cdots,A_n}(E_1) \times \pi_{B_1,B_2,\cdots,B_m}(E_2)$$

第三步（4）：第二次下放后：


π_{Sname}
$\delta_{SC.Sno=S.Sno}$
×
$\pi_{S.Sname,S.Sno}$
S
$\pi_{SC.Sno}$
$\delta_{C.Cno=SC.Cno}$
×
$\pi_{C.Cno}$
$\delta_{Cpno='5'}$
C
$\pi_{SC.Sno,SC.Cno}$
SC



- **基于规则的优化**——根据一系列启发式规则生成最终的查询计划。
- **基于代价的优化**——先估算候选查询计划的代价，再从中选择代价最小的。
- **两者结合的优化方法**——先根据启发式规则产生若干较优的候选查询计划，再从中选择代价最小的。

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
- 对于一个查询语句有许多存取方案，它们的执行效率不同， 仅仅进行代数优化是不够的
- 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

物理优化（续）

优化的方法

- 基于规则的启发式优化
- 基于代价估算的优化
- 两者结合的优化方法

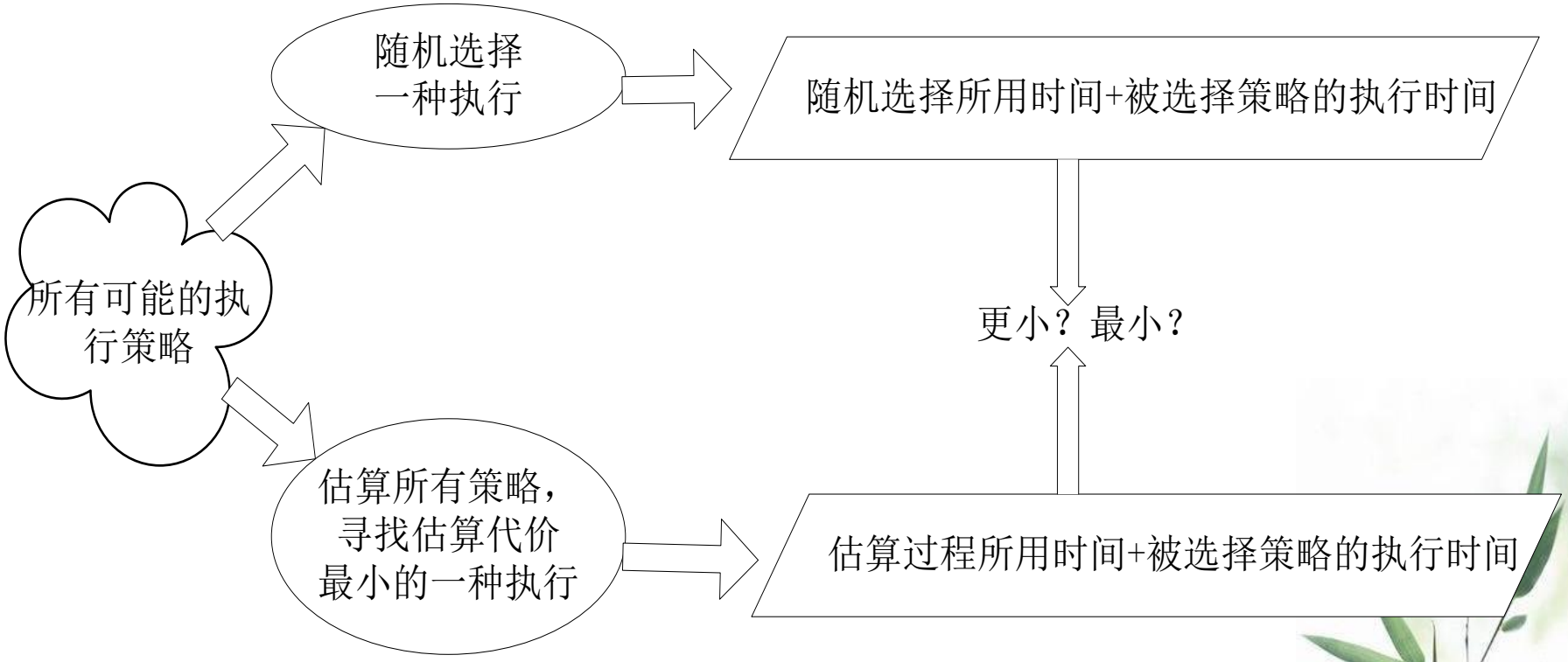
9.4.1 基于启发式规则的存取路径选择优化

一、 选择操作的启发式规则

二、 连接操作的启发式规则

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。



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。

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。

基于启发式规则的存取路径选择优化(续)

一、 选择操作的启发式规则

对于小关系，使用全表顺序扫描。

对于大关系，启发式规则有：

(1)对于选择条件是主码＝值的查询

- 查询结果最多是一个元组，选择主码索引
- 一般的RDBMS会自动建立主码索引

(2)对于选择条件是非主属性＝值的查询，并且选择列上有索引

- 估算查询结果的元组数目
  - 比例较小(<10%)可以使用索引扫描方法
  - 否则使用全表顺序扫描

基于启发式规则的存取路径选择优化(续)

(3)对于选择条件是属性上的非等值查询或者范围查询，并且选择列上有索引

- 估算查询结果的元组数目
  - 比例较小(<10%)可以使用索引扫描方法
  - 否则使用全表顺序扫描

基于启发式规则的存取路径选择优化(续)

(4)对于用AND连接的合取选择条件

- 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的索引
  - 则可以用其他索引扫描方法
  - 否则使用全表顺序扫描

基于启发式规则的存取路径选择优化(续)

- 选择率低，基于索引的选择算法优于全表扫描算法
- 选择率高或要查找的元组均匀地分布在查找的表中，采用全表扫描较好。因为除了对表的扫描操作，还要加上对B+树索引的扫描操作，对每个检索码，从B+树根节点到叶子节点路径上的每个节点都要执行一次I/O操作

(5)对于用OR连接的析取选择条件，一般使用全表顺序扫描

基于启发式规则的存取路径选择优化(续)

二、 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，并选择其中较小的表，确切地讲是占用的块数(b)较少的表，作为外表(外循环的表)。

9.4.2 基于代价的优化

启发式规则优化是定性的选择，适合解释执行的系统

– 解释执行的系统，优化开销包含在查询总开销之中

编译执行的系统中查询优化和查询执行是分开的

– 采用基于代价的优化方法

代价模型

代价估算与数据库的状态密切相关，需要在数据字典中存储优化器所要的统计信息。

常见统计信息

1.基本表

元组总数、元组长度、占用块数、溢出块数

2.列

不同值的个数、选择率selectivity、最大值、最小值、是否有索引、索引类型

3.索引

索引层数、不同索引值个数、索引选择基数（同索引值的情况）、叶结点个数

代价估算公式

B：表的块数； L：索引深度； S：索引选择基数； Y：索引叶结点数； Frs:连接选择性； Mrs:连接结果单块记录数； Nr关系R元组数； Ns关系S元组数。

selectivity

1.全表扫描

cost=B，对于单值搜索，cost=B/2

2.索引扫描

cost=L+1	码=值
cost=L+S	非码属性=值
cost=L+Y/2+B/2	>、>=、<、<=

3.nested loop join

cost=Br+Br*Bs/(K-1)+(Frs*Nr*Ns)/Mrs

4.merge join

cost=Br+Bs+(Frs*Nr*Ns)/Mrs

计划枚举

基于规则的计划重写后，DBMS就可以枚举其物理执行计划并评估其代价。

- 单关系查询
- 多关系查询
- 嵌套查询

计划枚举（续）

- 单关系查询计划
  - 仅仅只考虑表访问的方法就足够了
    - 顺序扫描(sequential scan)
    - 二分查找（聚簇索引）
    - 索引扫描
- 多表查询计划
  - 不同连接顺序代价不同
  - 连接的表越多，枚举的查询计划呈指数级上升

计划枚举（续）

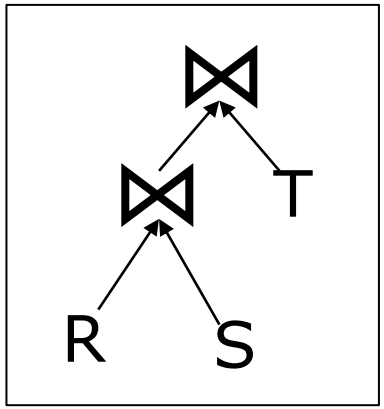
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

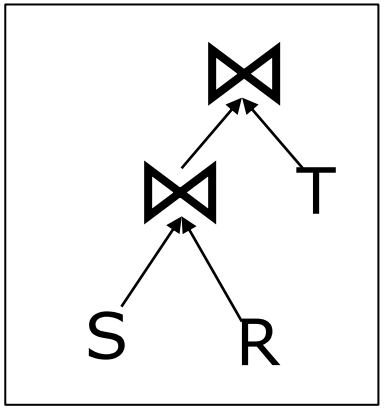
- 一般情况下的计划枚举
  - 枚举所有的连接顺序
  - 每个join算子的执行方案
    - Hash join; Sort Merge; Nested Loop …
  - 每个表的访问方法
    - Index Scan; Seq Scan;….
  - 枚举空间巨大

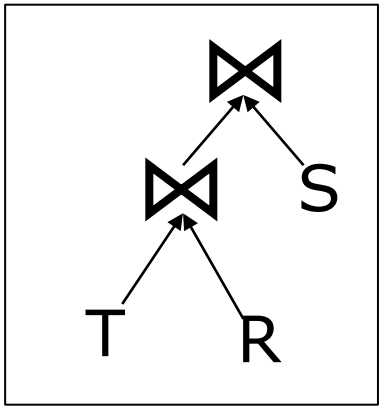
计划枚举（续）

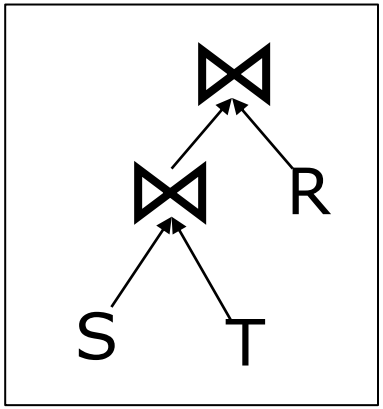
1. 枚举所有可能的连接顺序

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







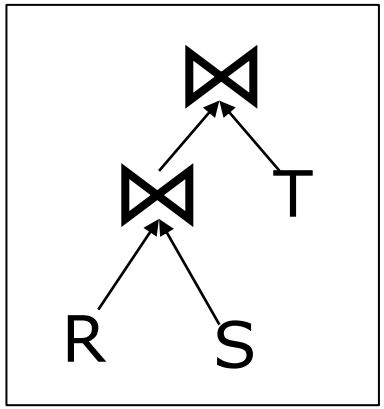


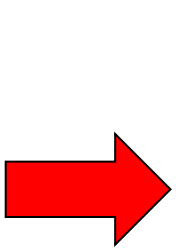
……

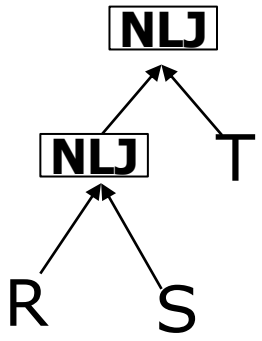
计划枚举（续）

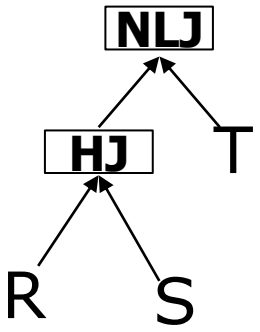
2. 枚举所有可能的连接算法

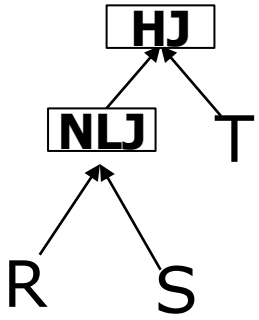
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

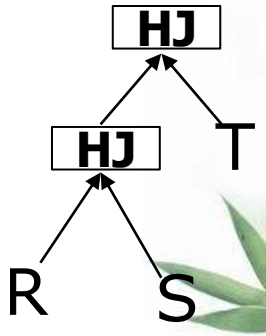








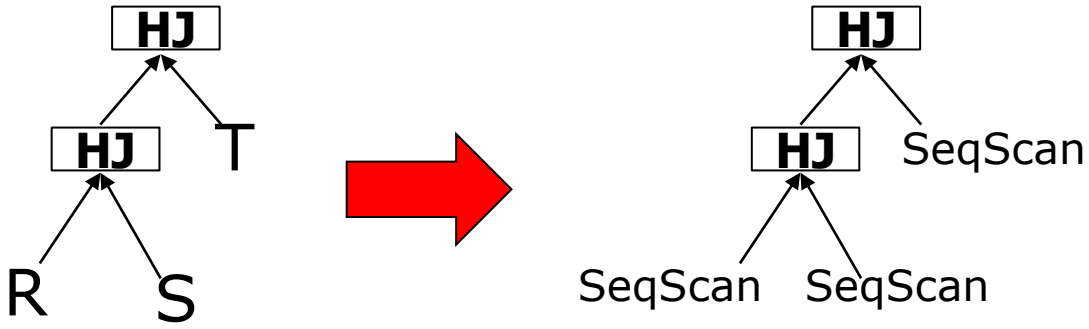


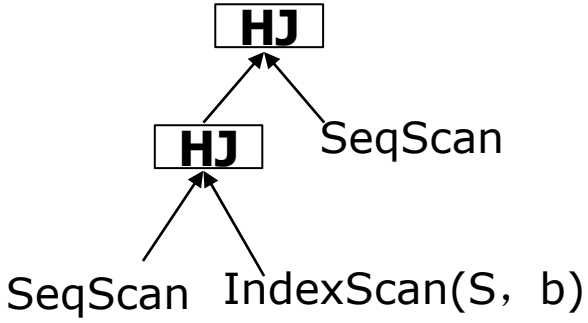


计划枚举（续）

3.枚举所有可能的数据存取方法

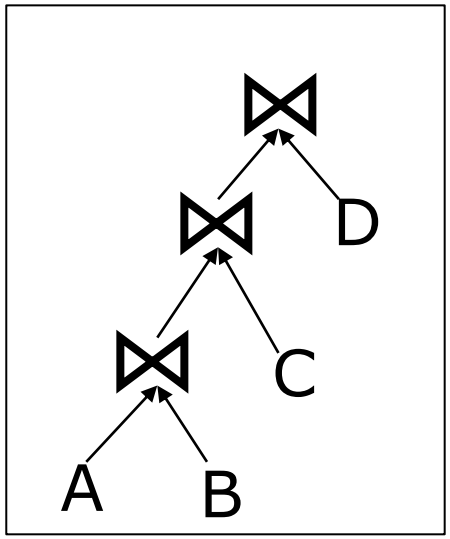
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

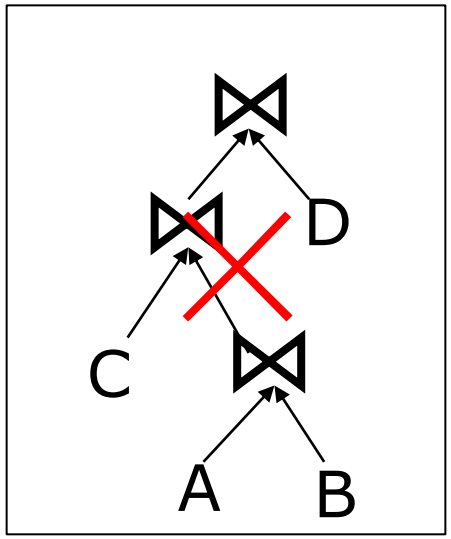


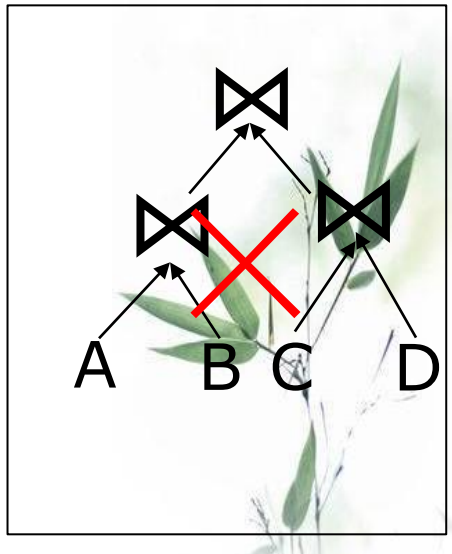


计划枚举（续）

- 多表查询计划(续)
  - 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



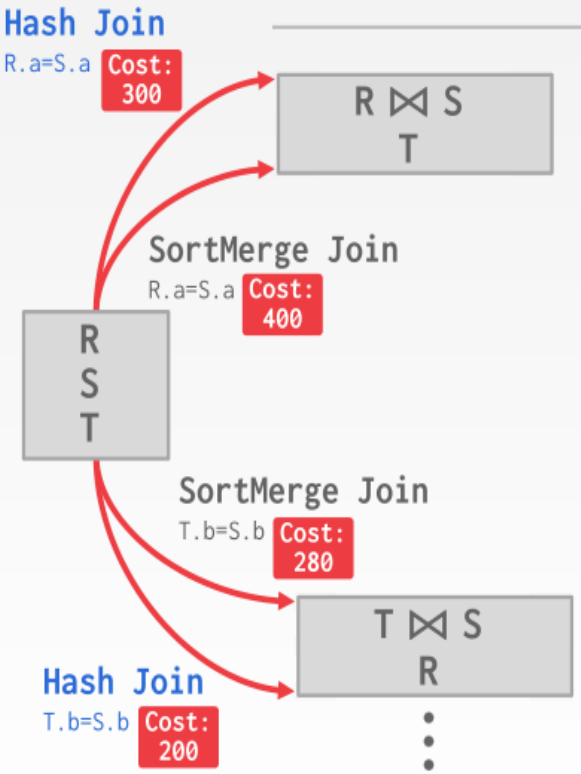




计划枚举（续）

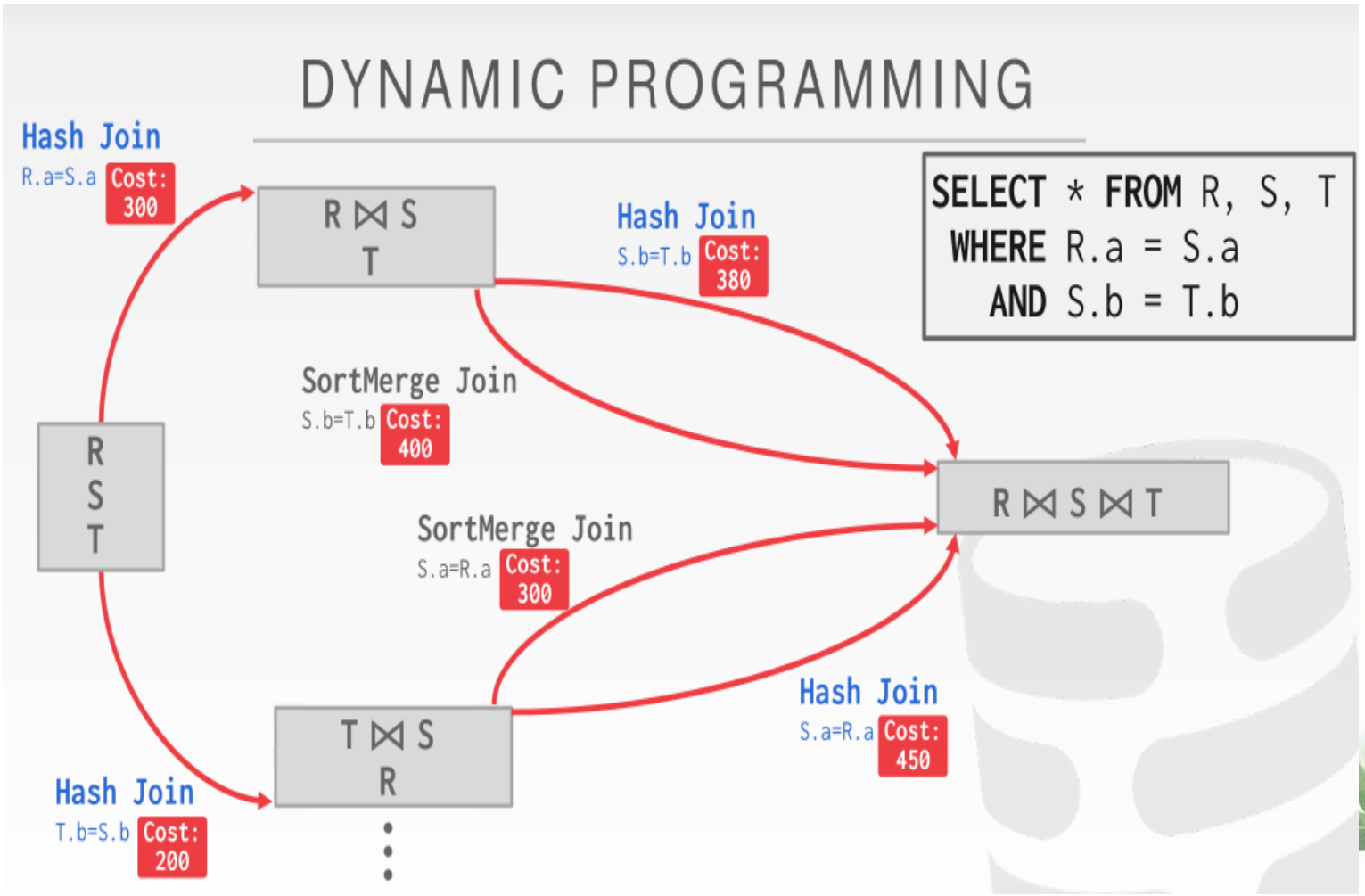
- 基于左深树的物理优化
  - 枚举所有的连接顺序
  - 每个join算子的执行方案
    - Hash join; Sort Merge; Nested Loop …
  - 每个表的访问方法
    - Index Scan; Seq Scan;….
  - 枚举空间巨大
    - 可以采用动态规划算法减少搜索空间

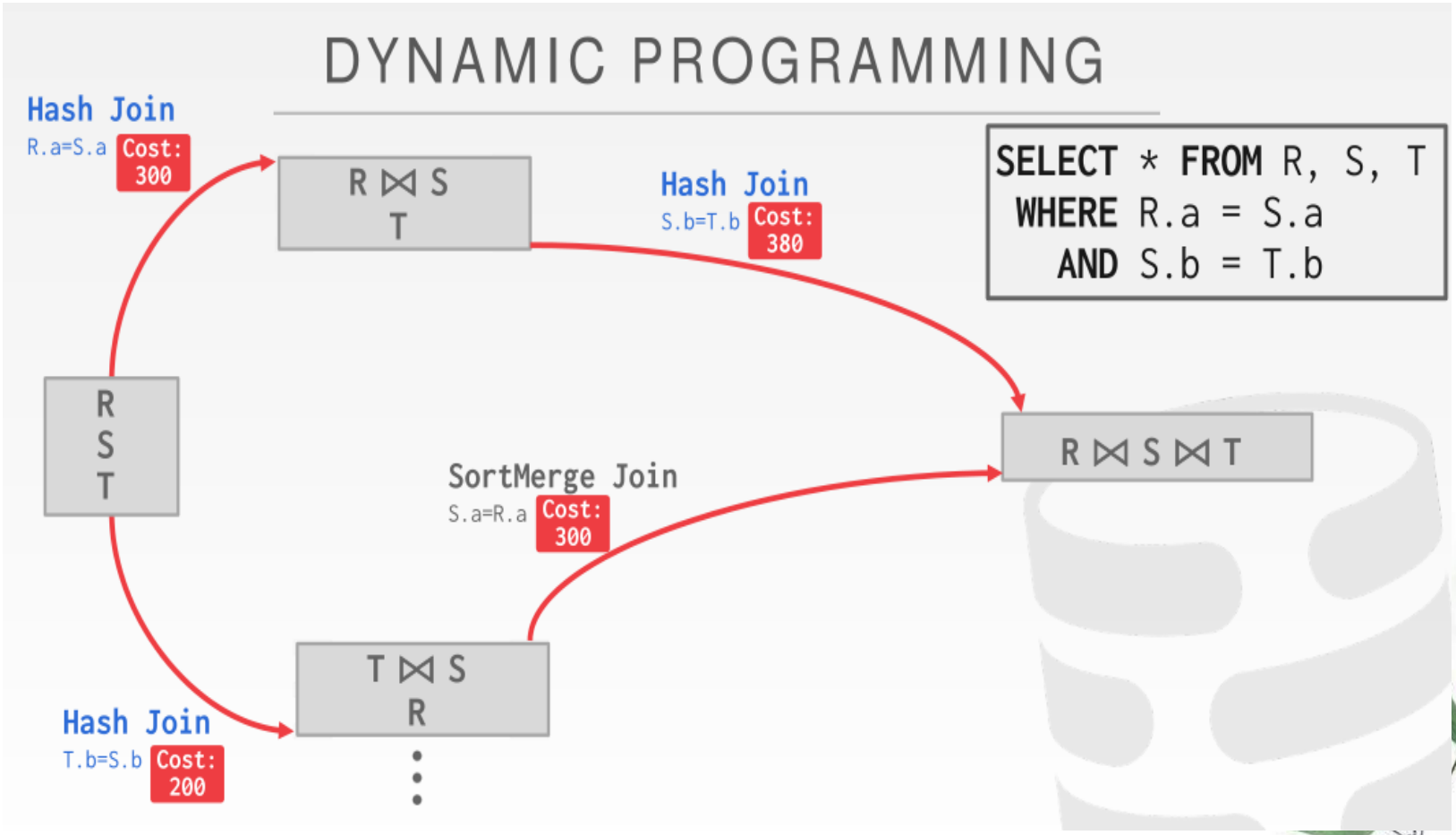
DYNAMIC PROGRAMMING



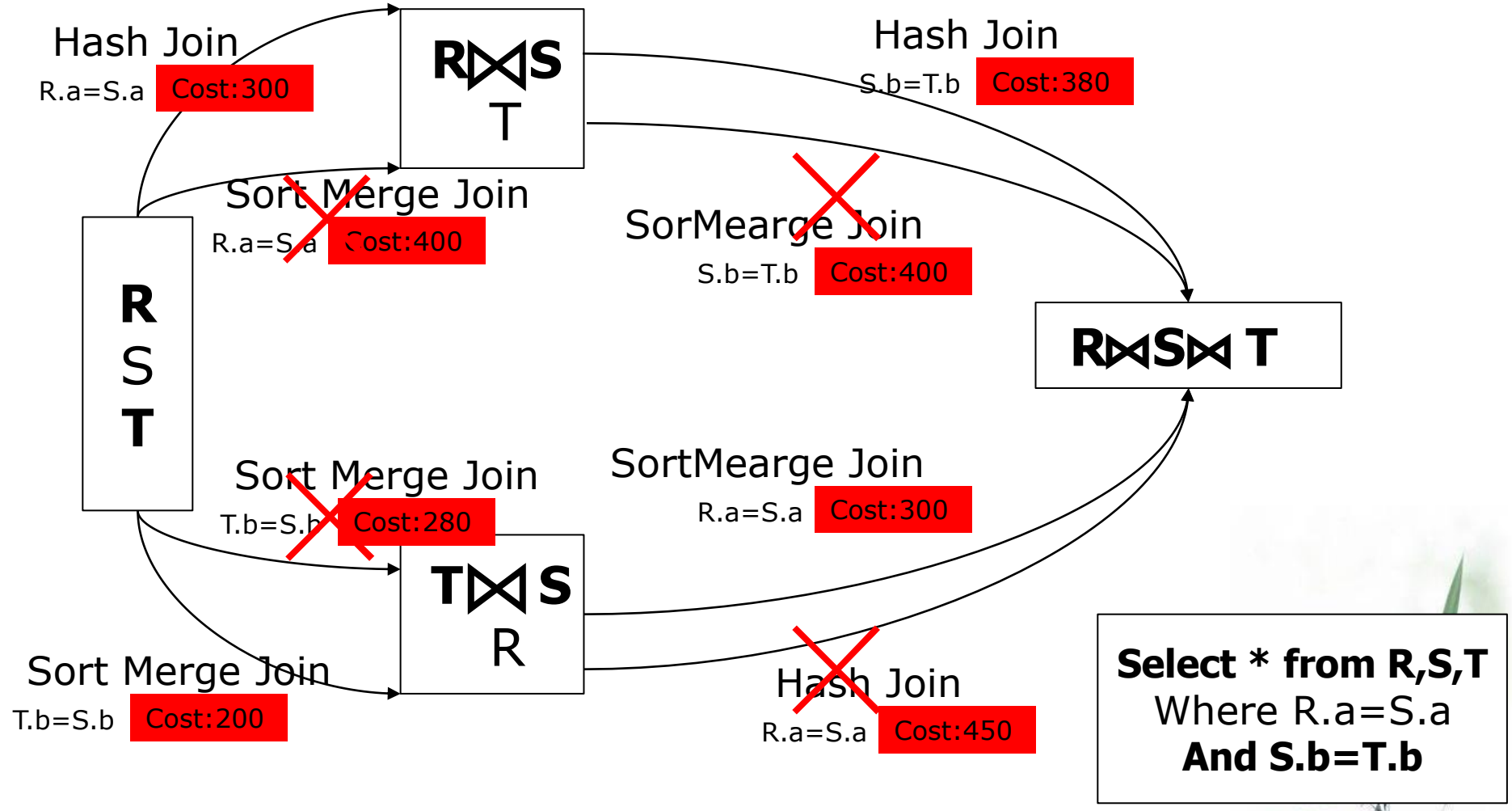

```
SELECT * FROM R, S, T
 WHERE R.a = S.a
   AND S.b = T.b
```

R ⋈ S ⋈ T

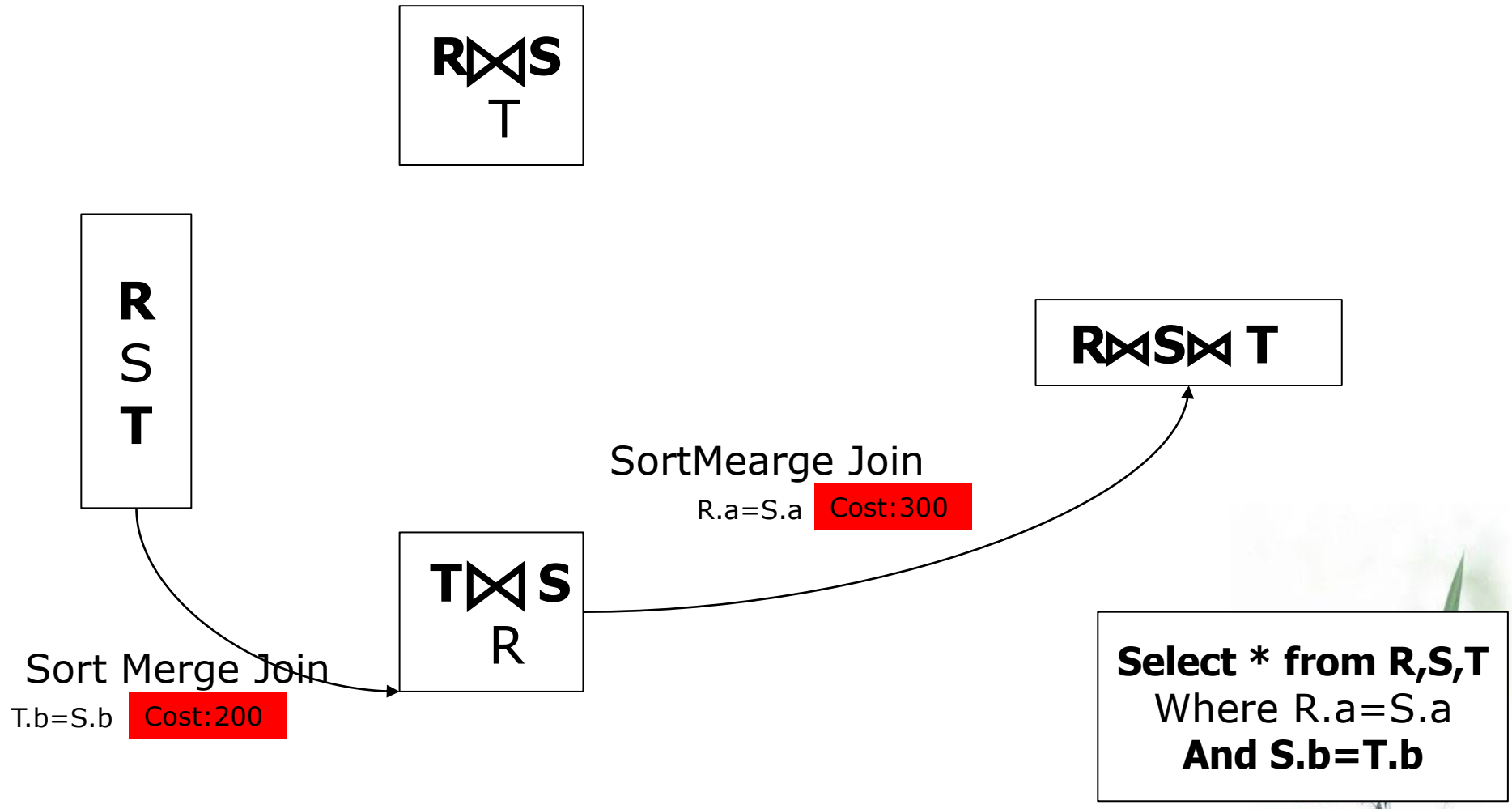

DYNAMIC PROGRAMMING
Hash Join
R.a=S.a Cost: 300
R ⋈ S
T
Hash Join
S.b=T.b Cost: 380
SELECT * FROM R, S, T
WHERE R.a = S.a
AND S.b = T.b
SortMerge Join
S.b=T.b Cost: 400
R
S
T
SortMerge Join
S.a=R.a Cost: 300
R ⋈ S ⋈ T
Hash Join
S.a=R.a Cost: 450
T ⋈ S
R
Hash Join
T.b=S.b Cost: 200


DYNAMIC PROGRAMMING
Hash Join
R.a=S.a
Cost: 300
R ⋈ S
T
Hash Join
S.b=T.b
Cost: 380
SELECT * FROM R, S, T
WHERE R.a = S.a
AND S.b = T.b
R
S
T
SortMerge Join
S.a=R.a
Cost: 300
R ⋈ S ⋈ T
T ⋈ S
R
Hash Join
T.b=S.b
Cost: 200

计划枚举（续）


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
SorMeargeJoin
S.b=T.b
Cost:400
R
S
T
R⋈S⋈ T
Sort Merge Join
T.b=S.b
Cost:280
SortMeargeJoin
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
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

计划枚举（续）


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

小结

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