

第9章 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。

数据处理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”，处理过程说到底还是有先后顺序和过程的，

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

基本概念1

查询分析（词法、语法、语义、符号名）

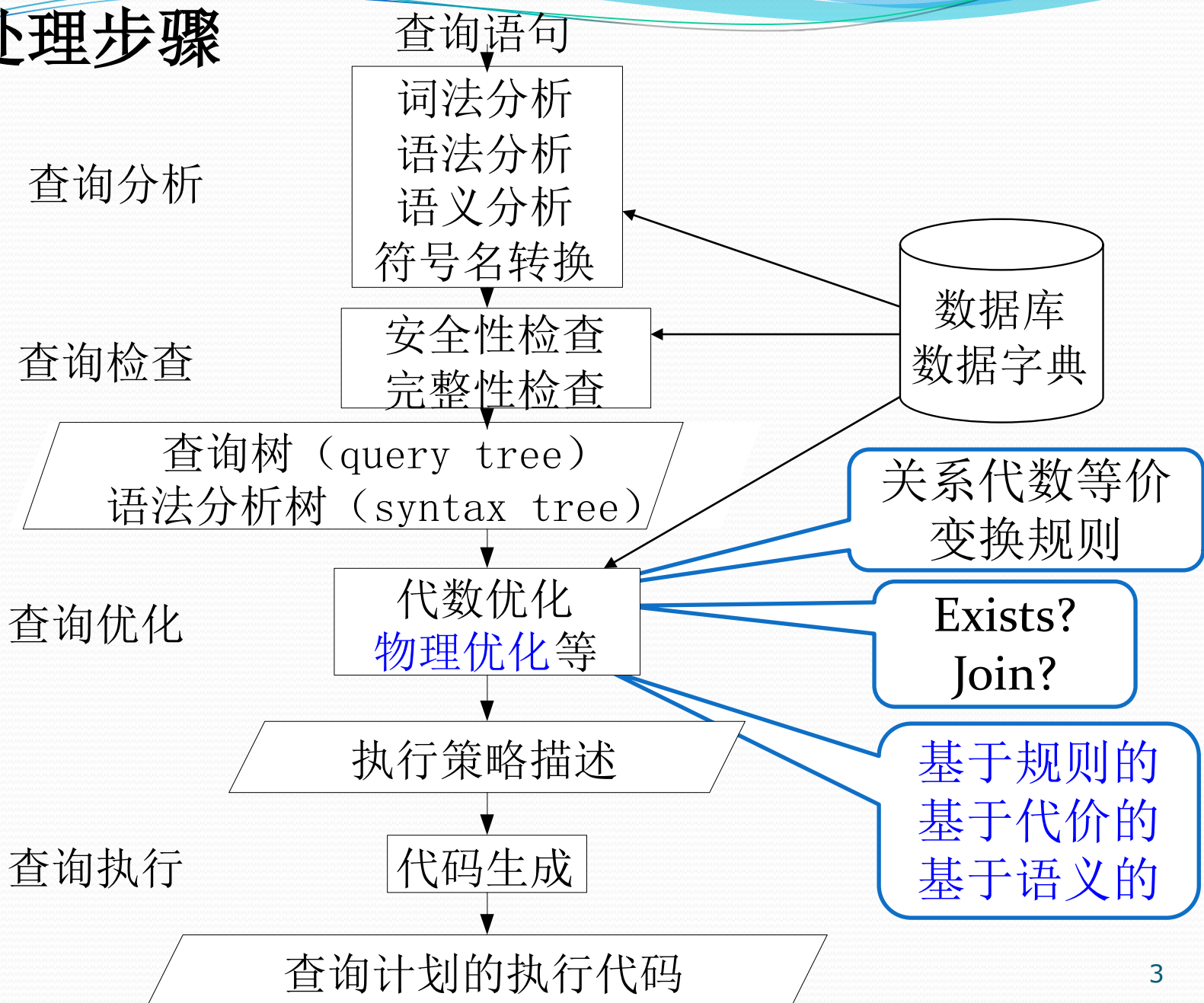
查询树（**query tree**），语法分析树（**syntax tree**）

关系代数语法树

执行树

执行计划

查询处理步骤



代数优化——优化关系代数表达式

基于规则的

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）。

选择存取路径和底层操作算法

Cluster

语义上组合条件，例如：

```
WHERE value BETWEEN 1 AND 100  
OR value BETWEEN 50 AND 150;
```

SQL语句（语法成分）



SQL语法树（**structure**明确）



易于设计从明确的
structure到关系代数
表达式树状结构
的算法



代数语法树（初始状态）



实现查询的**多解性**：
一个查询可能存在
多种实现途径（**逻辑
顺序不同、算法
不同、存取方法不
同**）。



代数语法树结构可
变形（**逻辑层面**），
节点的执行算法可
变换（**含物理层
面**），如何变？依
据什么变？

查询优化的任务和原理

可变项太多：

语法树结构可变形
（逻辑层面），节
点的执行算法可变
换（含物理层面）。



优化机制要在一个策略空间（可能很庞大）里找最优解，穷举法不一定可行，最优解未必保证能找到。

（即使理论上最优，实际中也未必最优）



启发式规则

（代数的、
物理操作的）；

代价模型

基本概念2

选择运算的实现方法（参见课件第8章）：

全表扫描

索引扫描

教材9.1.2节算法具体实现过程，掌握原理

连接运算的实现方法：

嵌套循环连接（nested loop）

排序—合并连接（sort-merge join）

Hash连接（hash join）

索引连接（index join）

两个关系是否只扫描一次？

如果两个关系的连接属性均存在重复值又会如何？

HASH连接

嵌套循环连接算法 (nested loop)

For 外层循环对应关系R的每个元组r :

for 内存循环对应关系S中的每一个元组s:

if r和s满足连接条件: 将r和s连接结果添加至临时结果集合 result;

Return result;

排序——合并连接 (sort-merge join)

HUST-CS

PANPENG

- 1 If 参与连接的关系R和S没有排好序： 先按照连接属性对它们排序（升序）；
- 2 初始化两个扫描指针 CUR_R 和 CUR_S 分别指向R和S的第一个元组；
- 3 If CUR_S 的连接属性A小于 CUR_R 的连接属性A： 将扫描指针 CUR_S 向后移动直至指向的S元组满足和 CUR_R 的连接条件；
- 4 将当前 CUR_R 和 CUR_S 的元组连接后保存至临时结果集合result；
- 5 While R未扫描完{
- 6 while CUR_S 满足与 CUR_R 的连接条件{
- 7 将当前 CUR_R 和 CUR_S 的元组连接后保存至临时结果集合result；
- 8 CUR_S 向后移动；
- 9 }
 CUR_R 向后移动；
- 10 }
- 10 Return result;

小孩牵手组合

Hash连接 (hash join)

将连接属性作为hash码。

第一步，划分（**building phase**），对包含较少元组的关系R进行一遍处理，将其元组分散到hash桶中；

第二步，试探（**probing phase**），对另一个关系S进行一遍处理，将其元组按照同一个hash函数寻找所属的hash桶里的R元组，补充相应的连接结果，直至S的全部元组处理完成后，输出所有的连接结果。

索引连接 (index join)

在连接关系S的连接属性上已经建立了索引。

For 关系R的每个元组r:

通过S的索引查找相应的满足连接条件的元组并补充连接结果;

9.2 查询优化的任务：提高速度（DBMS）

具体目标：

- 1、减少中间关系规模
- 2、减少I/O

例：设有如下关系：

学生（学号，姓名，性别，出生日期，所在系），

课程（课号，名称，学分）， **成绩**（学号，课号，成绩）

要求查询选修了2号课程的学生姓名。

查询的SQL语句:

```
SELECT  Student.Sname
FROM    Student, SC
WHERE   Student.Sno=SC.Sno
AND     SC.Cno= 'C2' ;
```

可用如下等价的代数表达式来完成这一查询:

$$Q1 = \pi_{\text{姓名}} \left(\sigma_{\text{学生.学号}=\text{成绩.学号} \wedge \text{课号}='2'} (\text{学生} \times \text{成绩}) \right)$$
$$Q2 = \pi_{\text{姓名}} \left(\sigma_{\text{课号}='2'} (\text{学生} \bowtie \text{成绩}) \right)$$
$$Q3 = \pi_{\text{姓名}} \left(\text{学生} \bowtie \sigma_{\text{课号}='2'} (\text{成绩}) \right)$$

由于查询执行的策略不同, 查询时间相差很大。

统计量：

学生记录——1000条；

成绩记录——10000条；

选修了2号课程的记录——50条。

一个物理块（页面）能容纳：

10条学生记录 $1000/10=100$ 块

或100条成绩记录 $10000/100=100$ 块。

内存仅提供7个存储页面，其中5页存放学生记录，1页存放成绩记录，1页存放中间结果。

磁盘每秒钟读/写20块数据记录。

采用嵌套循环实现方式。

查询执行方法采用物化模型。

外表：学生表
内表：成绩表

第一种情况：

$Q1 = \pi_{\text{姓名}} (\sigma_{\text{学生.学号}=\text{成绩.学号} \wedge \text{课号}='2'} (\text{学生} \times \text{成绩}))$

1. 计算：学生 \times 成绩

读取的总块数为：

学生表
批量读次数

成绩表块数

学生表
块数

$$1000/10 + 1000/(10 \times 5) \times 10000/100 = 2100 (\text{块})$$

所需时 $T1 = 2100/20 = 105$ (秒)

笛卡儿积的元组个数为 $10^3 \times 10^4 = 10^7$ ，设每块能装10个元组，则写出这些块所需的时间 $T2 = 10^7/10/20 = 5 \times 10^4$ (秒)

注意：笛卡尔积写出！

2. 作选择 σ

依次读入笛卡尔积连串后的结果，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 T_3 与 T_2 相等，即 $T_3=5 \times 10^4$ （秒），满足条件的记录仅有50条，结果直接驻留内存。

3. 作投影 π

将内存中的结果在“姓名”上作投影，得最终结果，因此第一种情况下执行查询的总时间为： $T=T_1+T_2+T_3 \approx 10^5$ （秒）

（总时间约28小时）

第二种情况 $Q2 = \pi_{\text{姓名}} (\sigma_{\text{课号}='2'} (\text{学生} \bowtie \text{成绩}))$

1. 计算自然连接

读取学生表和成绩表的策略不变，总的读取时间仍105秒，但自然连接的结果比第一种情况大大减少，为 10^4 条，因此，写出这些元组所需时间为 $10^4/10/20=50$ （秒）。

10000条成绩记录

2. 作选择

读取中间结果所需的时间仍为50（秒），符合条件的记录为50条。

3. 作投影

将中间结果投影输出。

第二种情况总的执行时间为： $105+50+50=205$ （秒）

第三种情况 $Q3 = \pi_{\text{姓名}} (\text{学生} \bowtie \sigma_{\text{课号}='2'} (\text{成绩}))$

1. 先对成绩表作选择运算，只读取一遍成绩表，存取花费时间为5秒，因满足条件的记录为50条，不必使用中间文件。
2. 读取学生表并与内存中的成绩记录作连接，花费时间5秒。
3. 输出投影结果。

第三种情况总的执行时间为10秒。

1000/10/20

上例充分说明查询优化的必要性，同时给出一些查询优化方法的基本思想。

避免笛卡儿积；
尽量让选择运算在连接运算之前执行

第四种情况 $Q4 = \pi_{\text{姓名}} (\text{学生} \bowtie \sigma_{\text{课号}='2'} (\text{成绩}))$

假设成绩表在课号属性上建有索引，学生表在学号属性上建有索引。

1. 先对成绩表作索引扫描，获取课号为2的成绩元组指针，关联获取50条成绩记录关联的学生元组指针，50条记录的规模可能只需读取一个数据块（聚簇索引），读取数据时间1/20秒。因满足条件的记录为50条，不必使用中间文件。
2. 对学生表进行索引扫描并与内存中的成绩记录作连接，50个学生理想状况下可能只需读取5个数据块（聚簇索引，每页10条学生记录），花费时间5/20秒。
3. 输出投影结果。

第四种情况总的执行时间约为(1+5)/20秒。

说明关系数据库中索引的重要性

9.3 代数优化

9.3.1 关系代数表达式的等价变换规则

1 连接/笛卡尔积的交换律

$$E1 \times E2 \equiv E2 \times E1$$

$$E1 \bowtie E2 \equiv E2 \bowtie E1$$

$$E1 \bowtie_F E2 \equiv E2 \bowtie_F E1$$

2 连接/笛卡尔积的结合律

$$(E1 \times E2) \times E3 \equiv E1 \times (E2 \times E3)$$

$$(E1 \bowtie E2) \bowtie E3 \equiv E1 \bowtie (E2 \bowtie E3)$$

$$(E1 \bowtie_F E2) \bowtie_F E3 \equiv E1 \bowtie_F (E2 \bowtie_F E3)$$

3 投影的串接律

$$\pi_{A_1, A_2, \dots, A_n}(\pi_{B_1, B_2, \dots, B_m}(E)) \equiv \pi_{A_1, A_2, \dots, A_n}(E)$$

A_1, A_2, \dots, A_n

B_1, B_2, \dots, B_m

假设:

- 1) E 是关系代数表达式
- 2) $A_i (i=1, 2, \dots, n), B_j (j=1, 2, \dots, m)$ 是属性名
- 3) $\{A_1, A_2, \dots, A_n\}$ 构成 $\{B_1, B_2, \dots, B_m\}$ 的子集

4 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，一次可检查全部条件。

5 选择与投影的交换律

(1) 假设: 选择条件 **F** 只涉及属性 **A₁, ..., A_n**

$$\sigma_F(\pi_{A_1, A_2, \dots, A_n}(E)) \equiv \pi_{A_1, A_2, \dots, A_n}(\sigma_F(E))$$

注意选择条件是否超出投影范围，若超出，则应用更一般的规则，**先扩充投影属性**

(2) 假设: **F** 中有不属于 **A₁, ..., A_n** 的属性 **B₁, ..., B_m**

$$\pi_{A_1, A_2, \dots, A_n}(\sigma_F(E)) \equiv \pi_{A_1, A_2, \dots, A_n}(\sigma_F(\pi_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(E)))$$

6 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 $F = F1 \wedge F2$ ，并且 $F1$ 只涉及 $E1$ 中的属性， $F2$ 只涉及 $E2$ 中的属性，则由上面的等价变换规则1，4，6可推出：

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

(3) 若 $F = F1 \wedge F2$ ， $F1$ 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。

7 选择与并的分配率

假设： $E = E1 \cup E2$, $E1$, $E2$ 有相同的属性名

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

8 选择与差的分配率

假设： $E1$ 与 $E2$ 有相同的属性名

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

9 选择与自然连接的分配率

假设： $E1$ 和 $E2$ 是两个关系表达式，

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

10 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：

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

11 投影与并的分配率

设E1和E2 有相同的属性名

$$\pi_{A_1, A_2, \dots, A_n} (E_1 \cup E_2) \equiv \pi_{A_1, A_2, \dots, A_n} (E_1) \cup \pi_{A_1, A_2, \dots, A_n} (E_2)$$

注意：没有投影和交的分配率

主要是代数优化

9.3.2 一般策略（**查询树的启发式**优化）

1、“选择”尽可能提前执行；

这是最基本一条，因为“选择”使中间结果变小。

2、索引和排序

特别是对连接运算，连接前“先排序”或“建立”索引，提高速度。

例：对贷款者表Borrower 与贷款记录表Loans进行自然连接

① 对贷款记录loans 按贷款者身份证号ID建立索引；

② 对贷款者表Borrower 满足条件的元组的ID值：

➤ 通过loans 索引查元组

➤ Borrower 元组与相应元组连接起来

无需反复扫描loans表本身

3、“投影”和“选择”同时进行（避免多次扫描关系，即避免投影和选择各扫描一次关系）。

前提是两种运算对同一关系实施才成立。

4、投影与其前后的其它双目运算同时进行，避免重复扫描关系（没有必要为了去掉某些字段而扫描一遍关系）。

5、（某些）选择与选择前的笛卡尔积结合

扫描得到的元组立即与参与计算的另一元组做匹配条件过滤，将这种笛卡尔积转变为连接运算。

连接运算（特别是等值连接运算）比笛卡尔积快。

6、提取公共子表达式。

① 计算公共子表达式结果

② 结果存入外存

③ 需要从外存调入内存使用（无需重计算）

④ 前提：外存调入内存的时间远远少于计算公共表达式时间

关系表达式代数优化算法

输入：关系代数表达式的查询树（语法树）

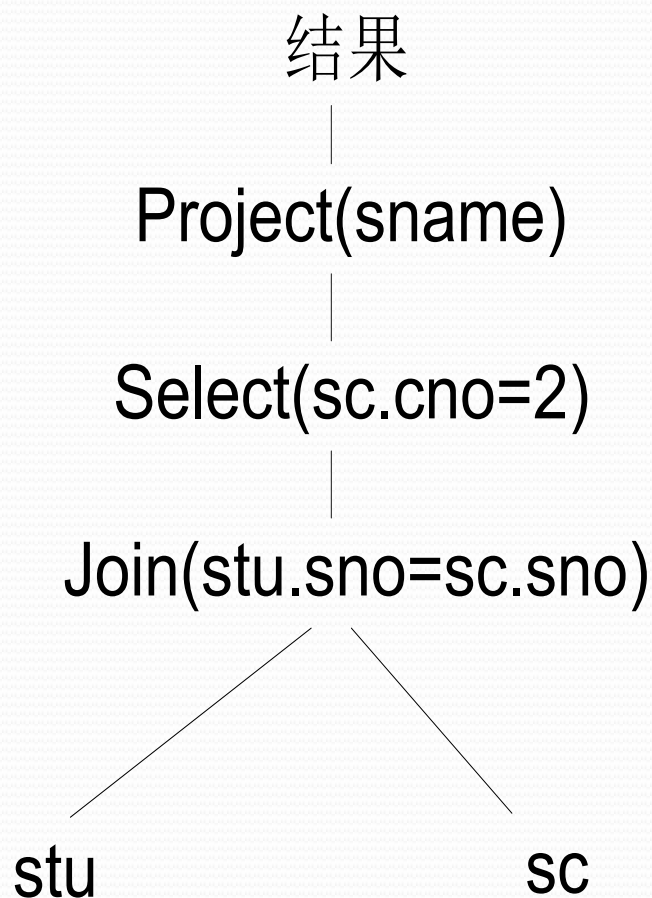
输出：优化后的查询树（语法树）

- 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（选择、投影运算）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。

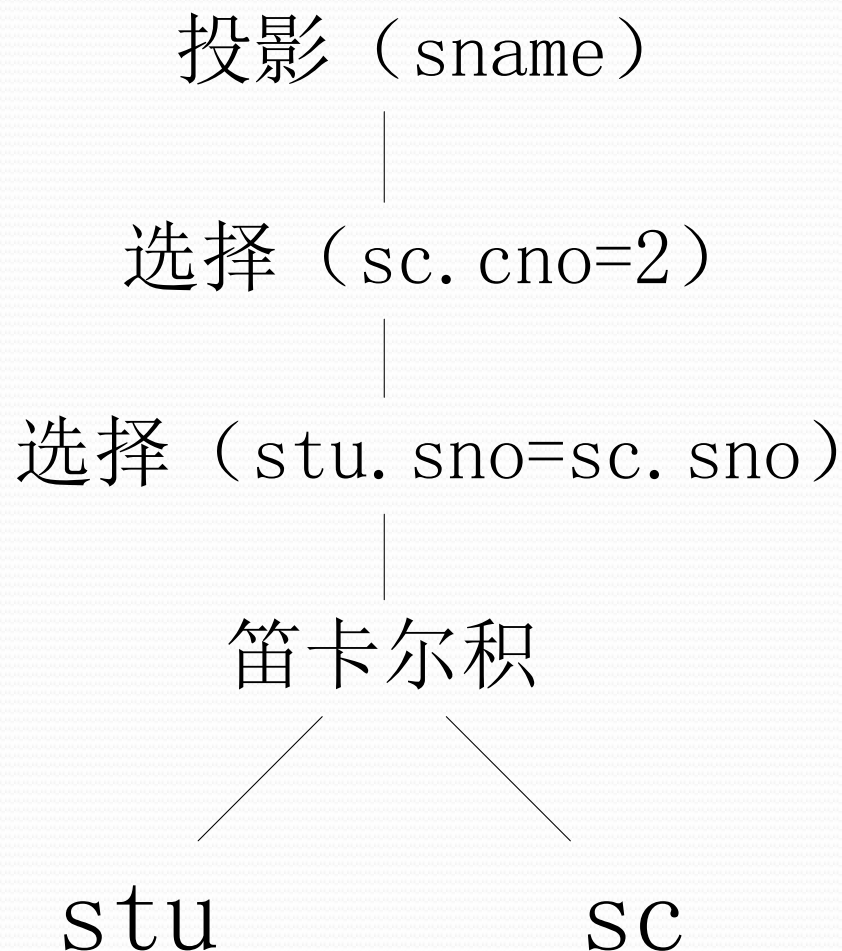
SQL查询的代数处理过程（课本图9.3~9.5）

查询树

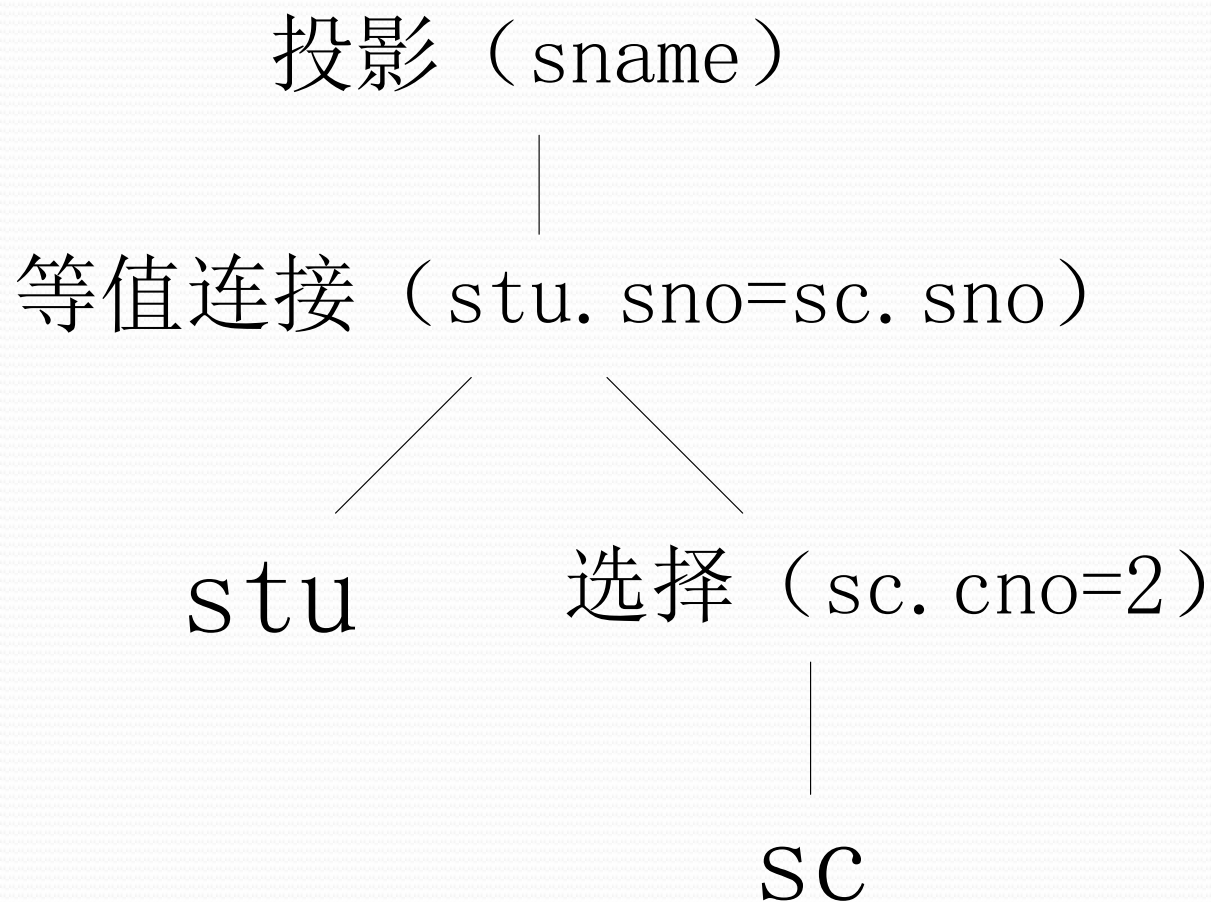
`select` sname `from` stu, sc `where` stu.sno=sc.sno and sc.cno=2



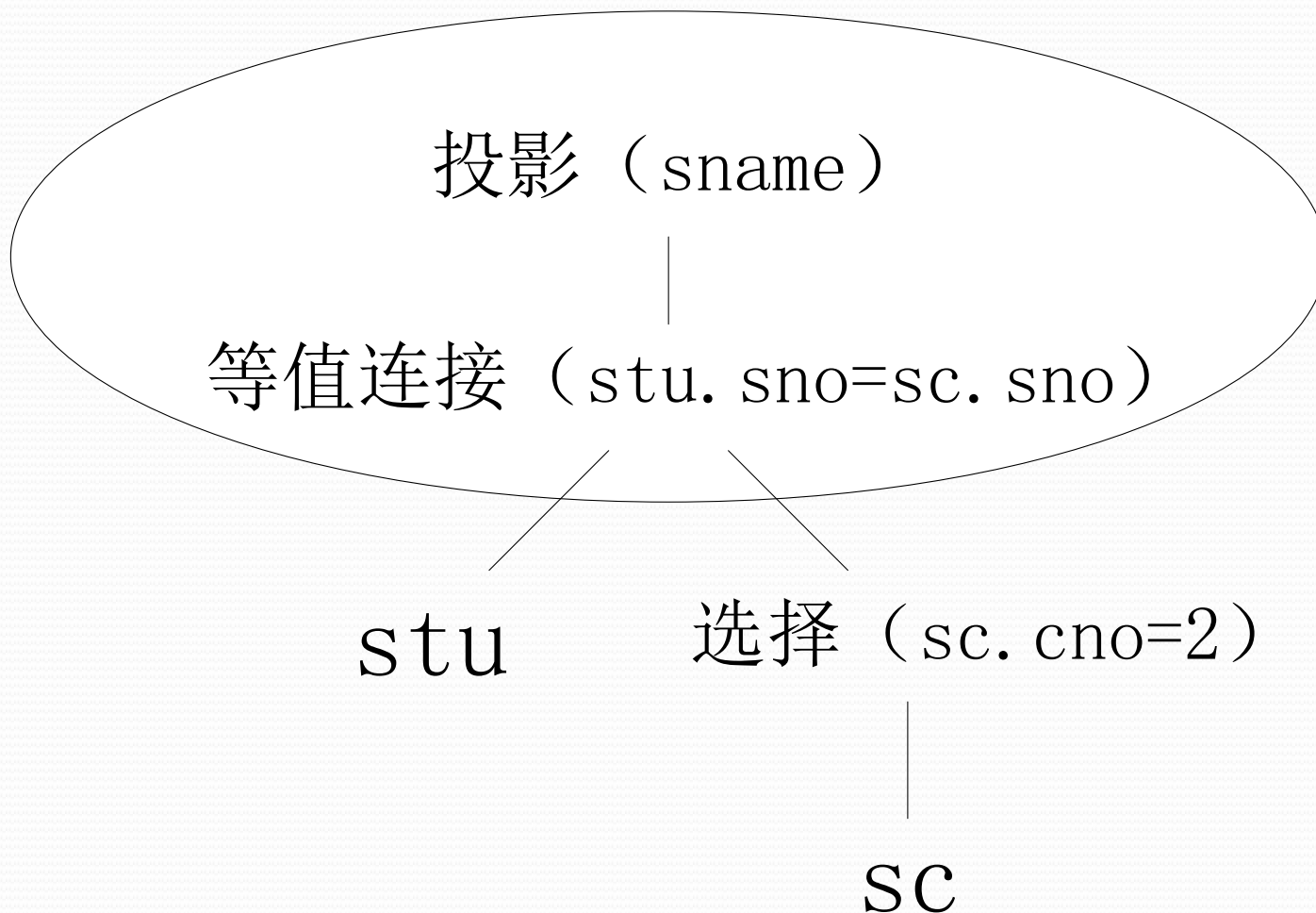
关系代数语法树



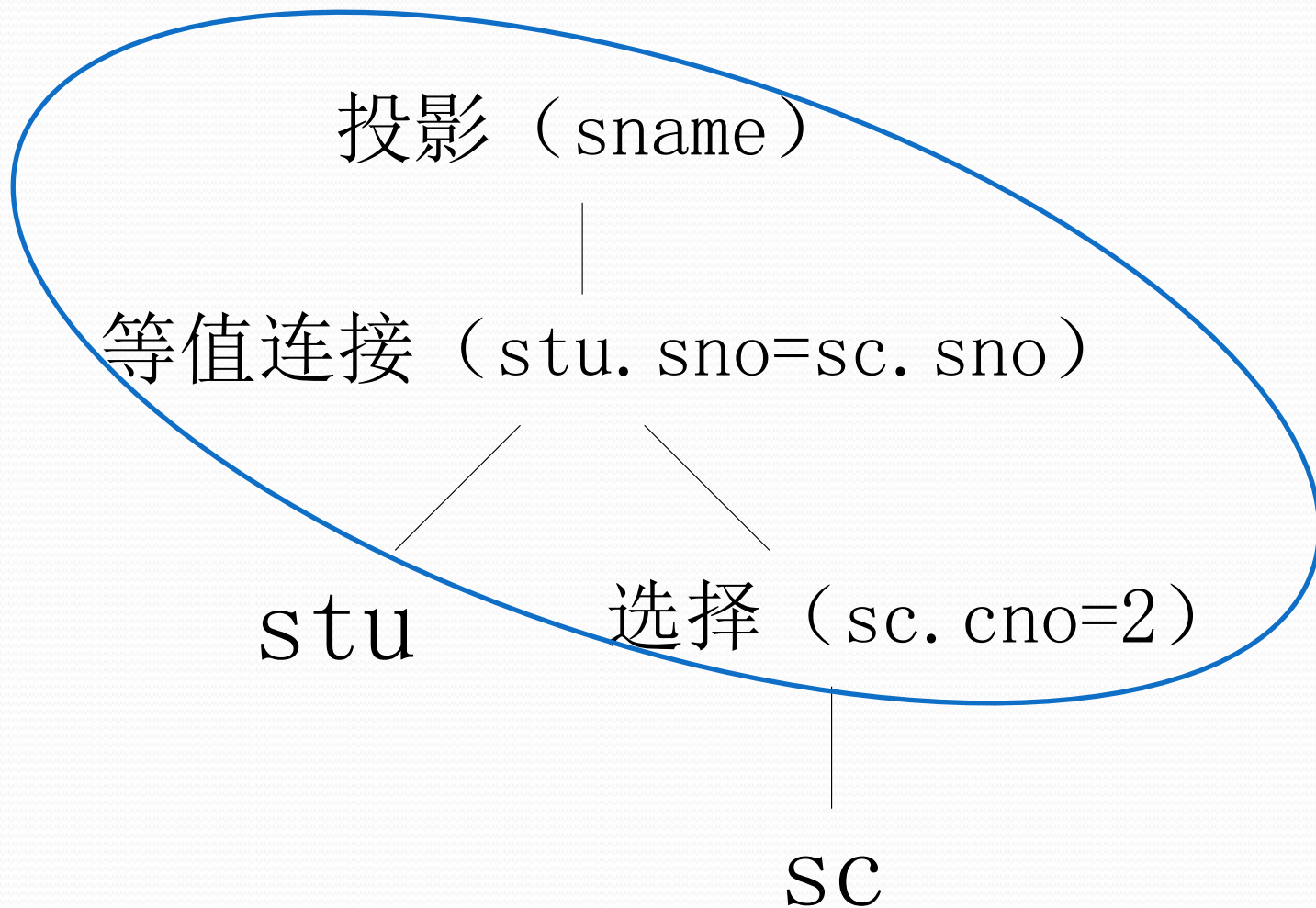
代数优化后的语法树1



代数优化后的语法树2

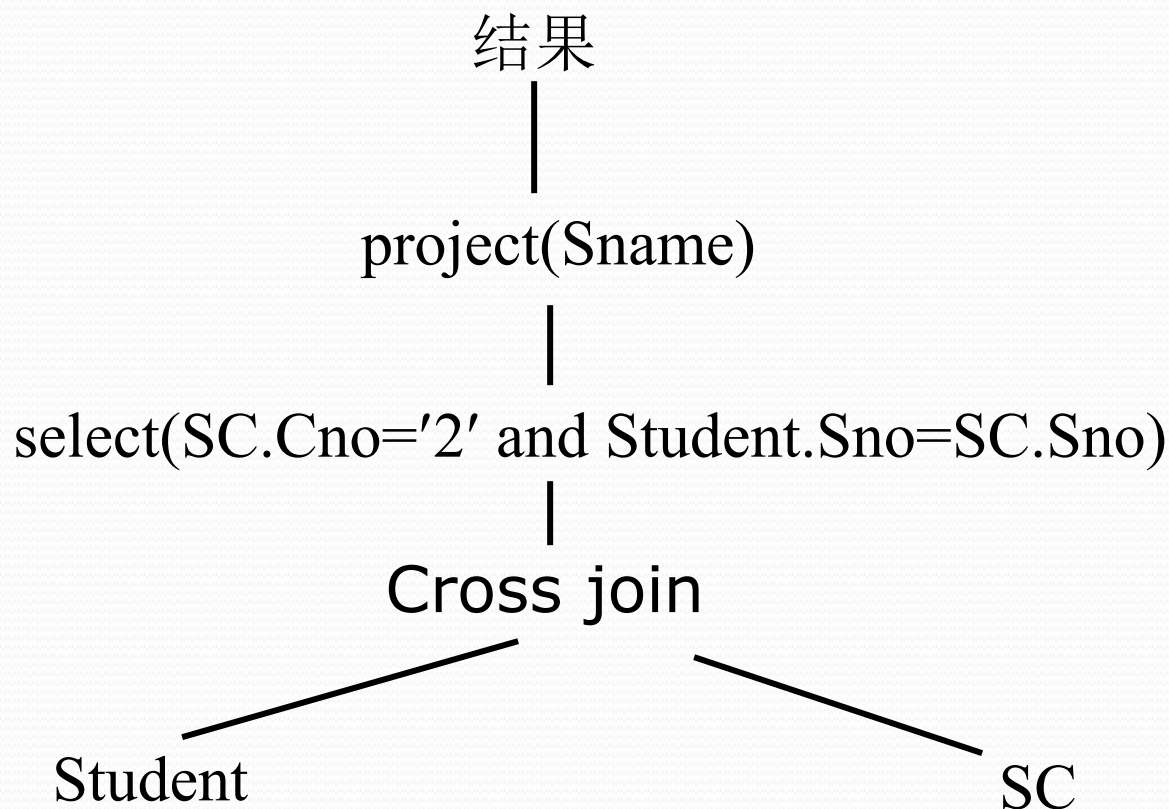


代数优化后的语法树3



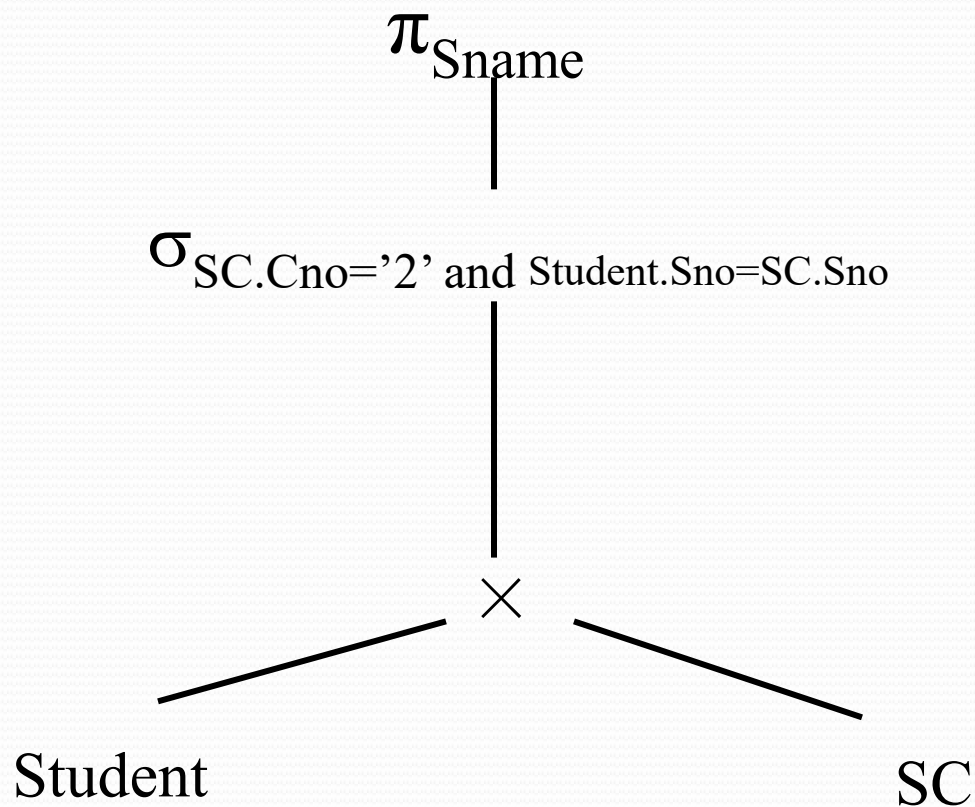
考虑投影的下推细节

(1) 查询树



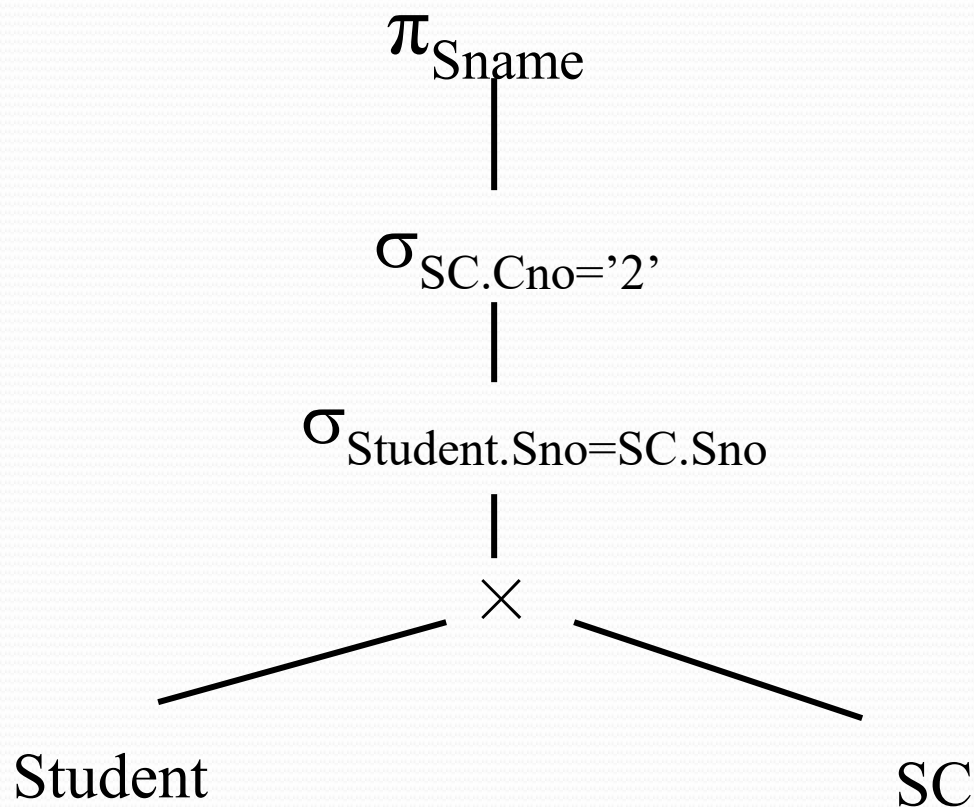
考虑投影的下推细节

(2) 关系代数语法树



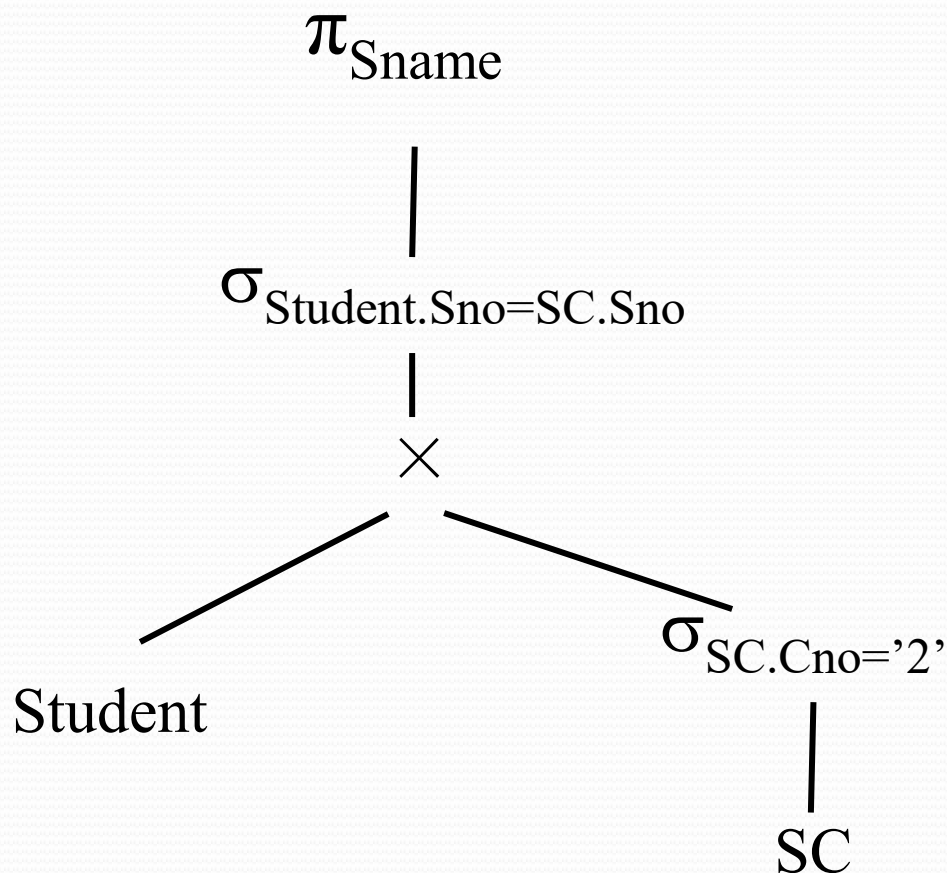
考虑投影的下推细节

(3) 分解选择运算



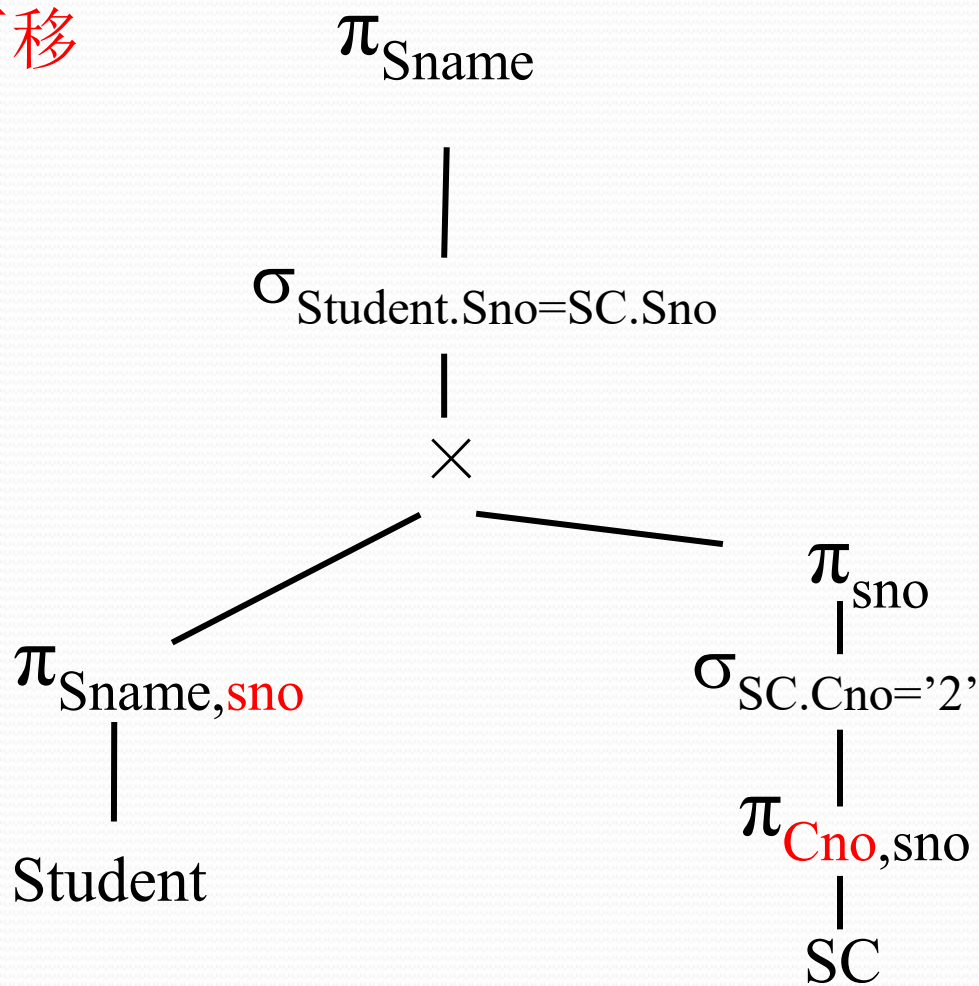
考虑投影的下推细节

(4) 选择下移



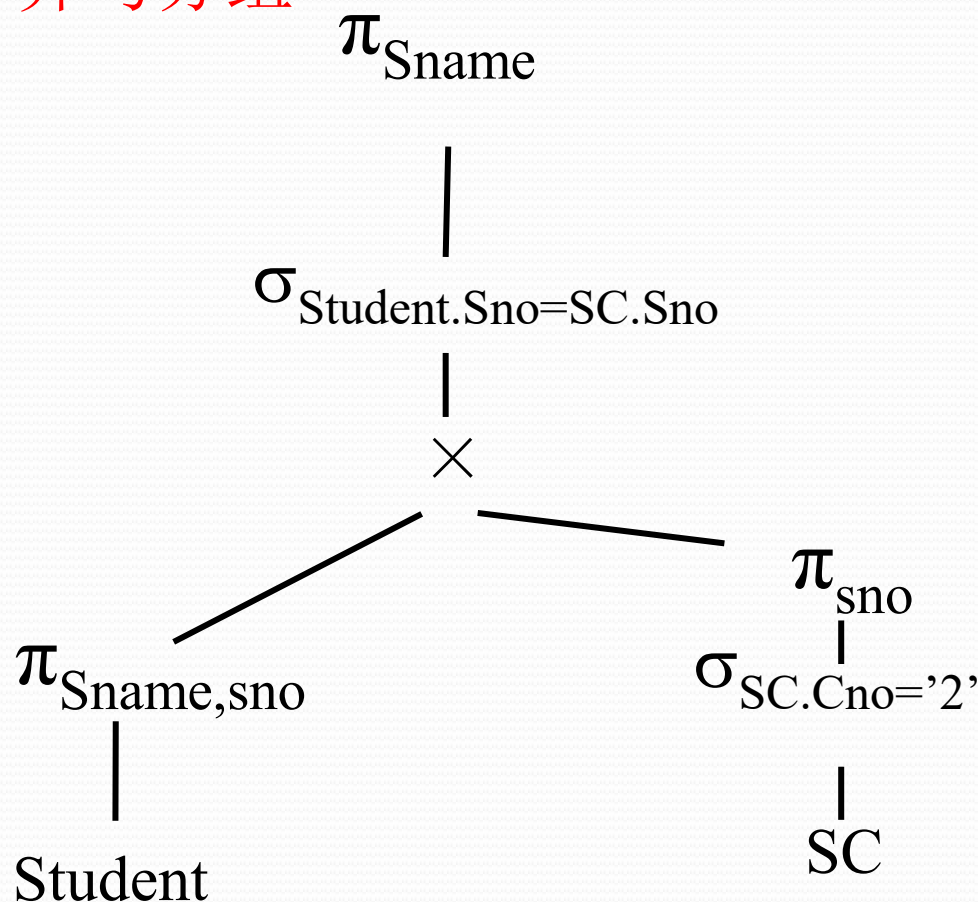
考虑投影的下推细节

(5) 投影下移



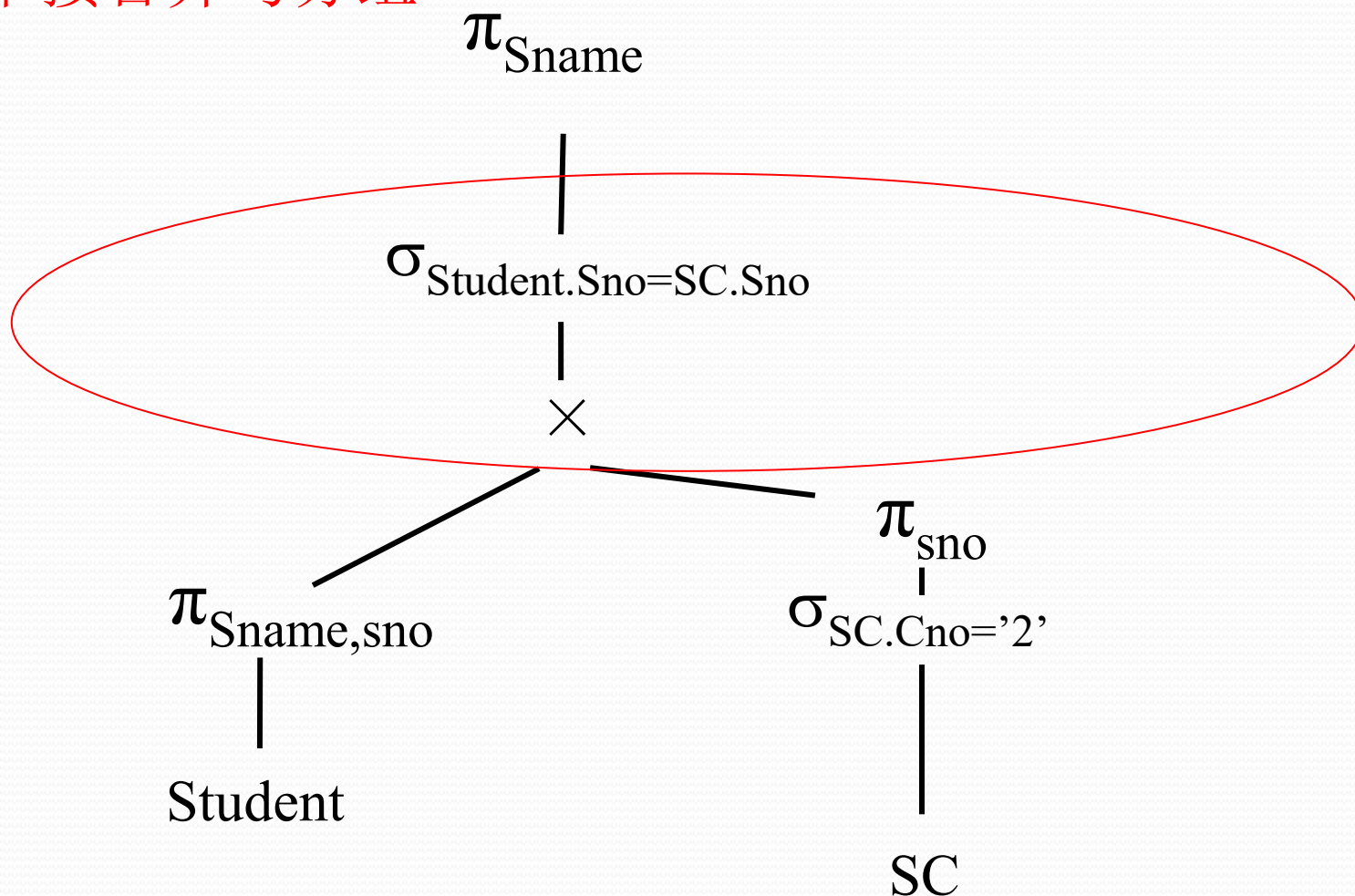
考虑投影的下推细节

(6) 串接合并与分组



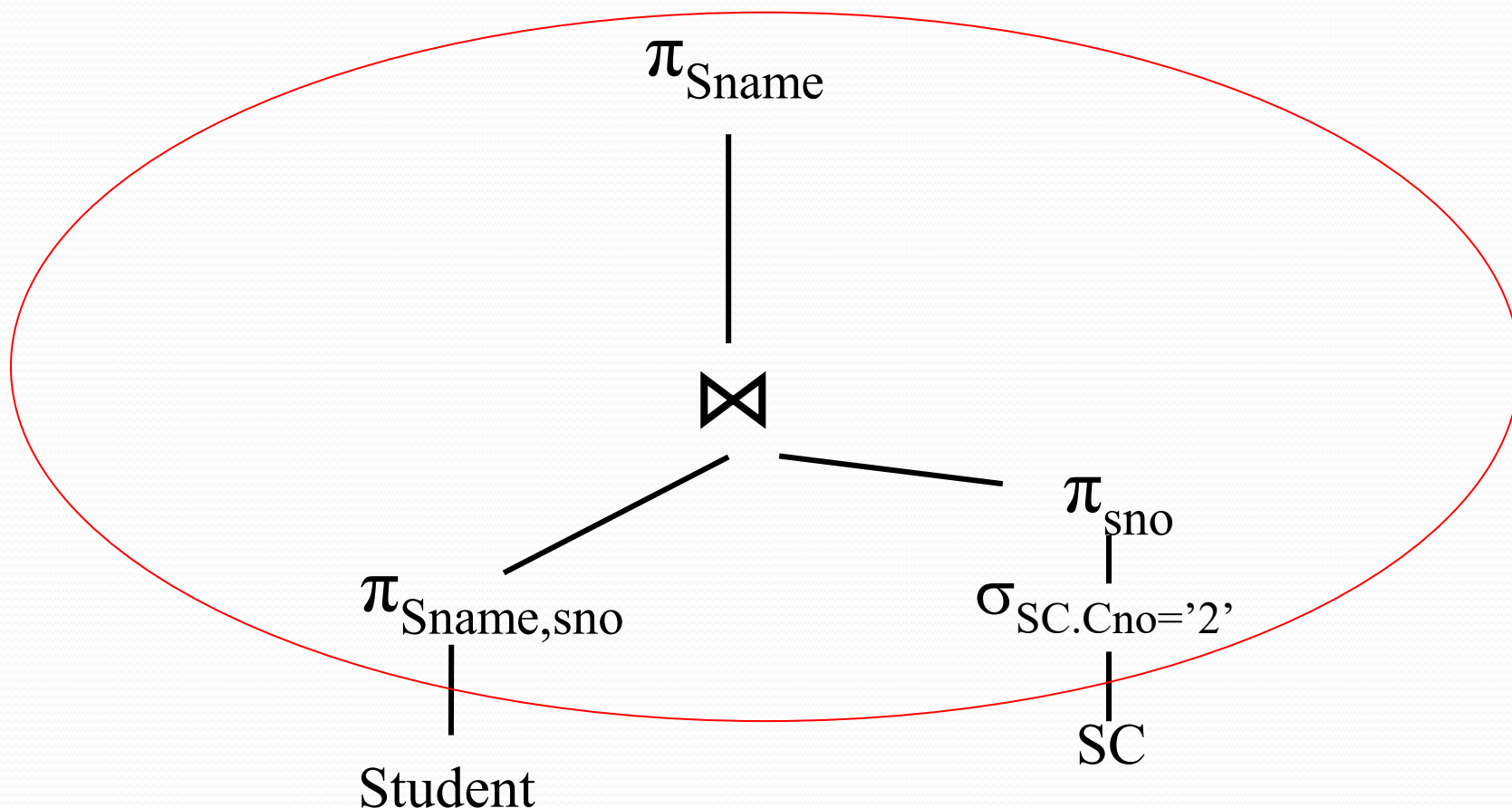
考虑投影的下推细节

(6) 串接合并与分组



考虑投影的下推细节

(7) 优化后的查询树



9.4 物理优化

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。

总代价 = I/O代价 + CPU代价 + 内存代价 + 通信代价

代价模型

- 物理代价
CPU, I/O, 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

代价模型（续）

- 基于磁盘DBMS代价模型
 - 磁盘I/O占代价模型的主要部分
 - CPU代价可以忽略
 - 要考虑数据的读取方式：顺序I/O或随机I/O
 - 依赖于缓冲区管理方式
- 分布式DBMS代价模型
 - 要考虑通信代价

代价模型（续）

统计信息

- 代价估算与数据库的状态密切相关，需要存储优化器所需要的统计信息。
- 统计信息包括表、属性和索引的信息
- 通常存放于数据字典中
- 状态需要及时更新，不同的系统更新时机不同

不同系统统计信息的更新命令：

- PostGres: ANALYSE
- Oracle/MySQL: ANALYSE TABLE
- SQL Server: UPDATE STATISTICS
- DB2: RUNSTATS

代价模型（续）

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（需要考虑同索引值的情况）、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。

1. 全表扫描

$\text{cost} = B$, 对于单值搜索, $\text{cost} = B/2$

2. 索引扫描

$\text{cost} = L + 1$

码 = 值

$\text{cost} = L + S$

非码属性 = 值

$\text{cost} = L + Y/2 + B/2$

>、>=、<、<=

3. nested loop join

$\text{cost} = Br + Br * Bs / (K - 1) + (Frs * Nr * Ns) / Mrs$

4. merge join

$\text{cost} = Br + Bs + (Frs * Nr * Ns) / Mrs$

代价模型（续）

- 统计信息

- N_R : 关系R的元组的数量

- $V(A, R)$: 关系R中属性A的distinct的值数量

如学生关系S中, $V(\text{'Gender'}, S) = 2$

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

$$SC(A, R) = N_R / V(A, R)$$

选择基数表明该属性作为选择属性能选择多少元组，查询优化**优先选择** $SC(A, R)$ **较小的属性**。但是这个参数成立的前提假设是数据均匀分布。

代价模型（续）

- 复合谓词选择率估计
 - 复合谓词P的选择率（selectivity）：指的是复合谓词P的元组占有率。
 - 选择率依赖于谓词P的类型
 - 相等
 - 范围
 - 非
 - 与
 - 或

代价模型 (续)

假设 $V(\text{age, people})$ 具有5个不同的值 (18-22) 且 $N_R=5$ 。

- 相等谓词

查询: `Select * from People where age=20`

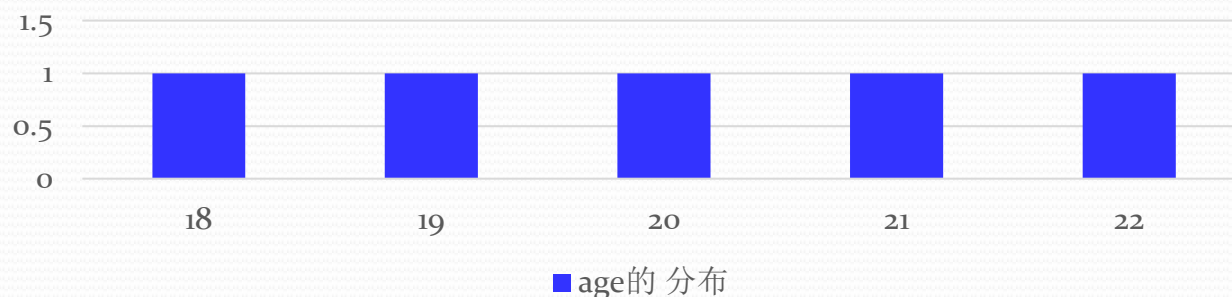
相等谓词: $A=\text{constant}$

相等谓词的选择率: $\text{Sel}(A=\text{constant}) = \text{SC}(P) / N_R$

$\text{Sel}(\text{age}=20) = 1/5$

SC(P): 满足条件P的元组数

age的分布



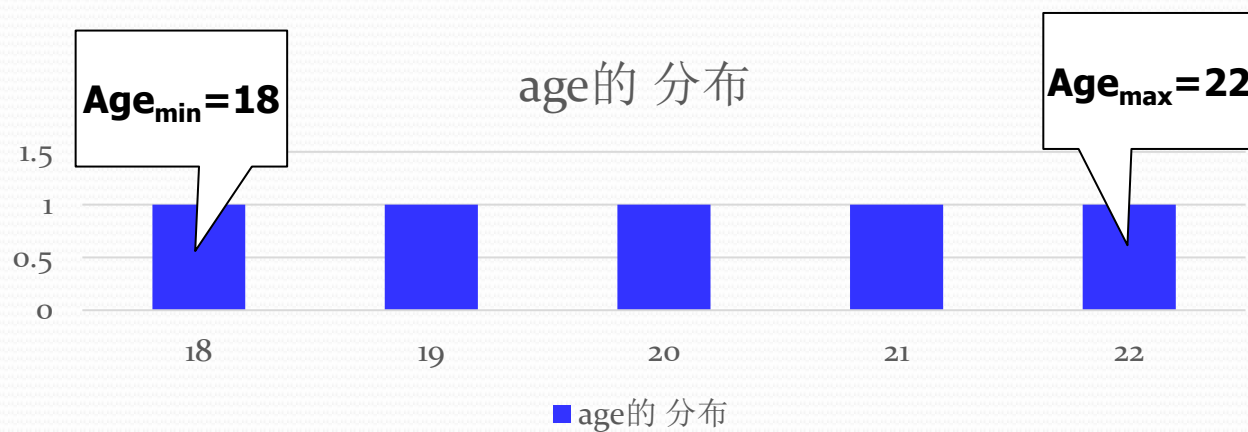
代价模型（续）

- 范围谓词

$$\text{Sel}(A \geq a) = (A_{\max} - a + 1) / ((A_{\max} - A_{\min} + 1))$$

查询: Select * from People where age ≥ 20

$$\text{Sel}(\text{age} \geq 20) = (22 - 20 + 1) / (22 - 18 + 1) = 3/5$$



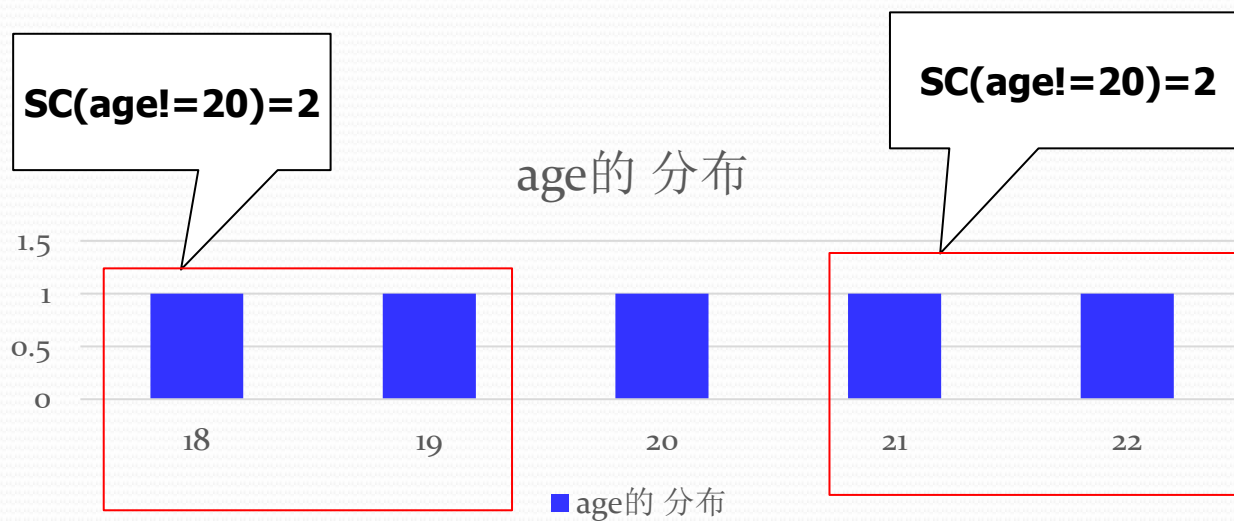
代价模型 (续)

- 非谓词

$$\text{Sel}(!P) = 1 - \text{sel}(P)$$

查询: `Select * from People where age!=20`

$$\text{Sel}(\text{age} \neq 20) = 1 - 1/5 = 4/5$$



代价模型（续）

- 与谓词

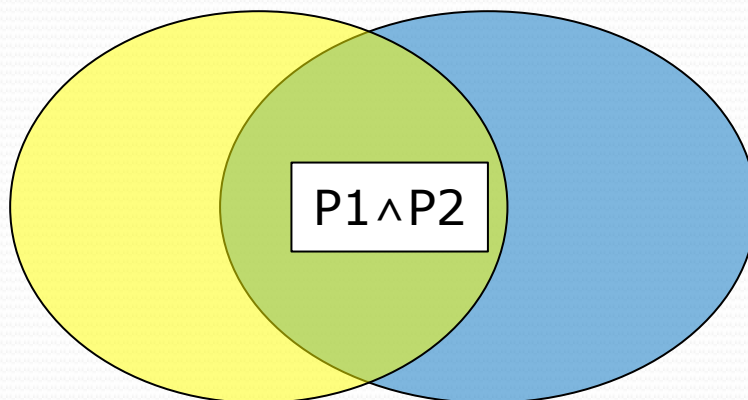
当谓词之间的选择独立时

$$\text{Sel}(P1 \wedge P2) = \text{sel}(P1) \times \text{sel}(P2)$$

查询：

Select * from People

where age=20 and name like ' 王%'



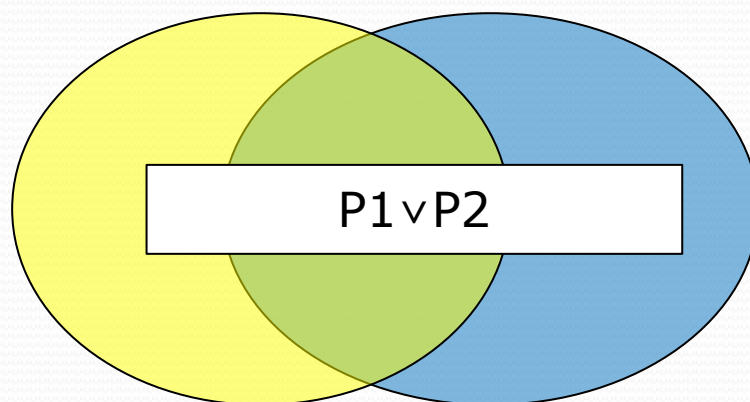
代价模型 (续)

- 或谓词

当谓词之间的选择独立时

$$\begin{aligned}\text{Sel}(P1 \vee P2) &= \text{sel}(P1) + \text{sel}(P2) - \text{Sel}(P1 \wedge P2) \\ &= \text{sel}(P1) + \text{sel}(P2) - \text{Sel}(P1) \times \text{Sel}(P2)\end{aligned}$$

查询: `Select * from People`
`where age=20 or name like ' 王%'`



代价模型（续）

选择率的假设

假设1：数据均匀分布

假设2：谓词的选择性是独立的

假设3：join时内表的每个元组能在外表中找到匹配

代价模型（续）

例：假设制造商的数量=10，汽车型号的数量=100，查询谓词
Make= 'Honda' and model = 'Accord'

- 与谓词

与谓词选择率： $\text{Sel}(P1 \wedge P2) = \text{sel}(P1) \times \text{sel}(P2)$

错误！

- 正确的选择率

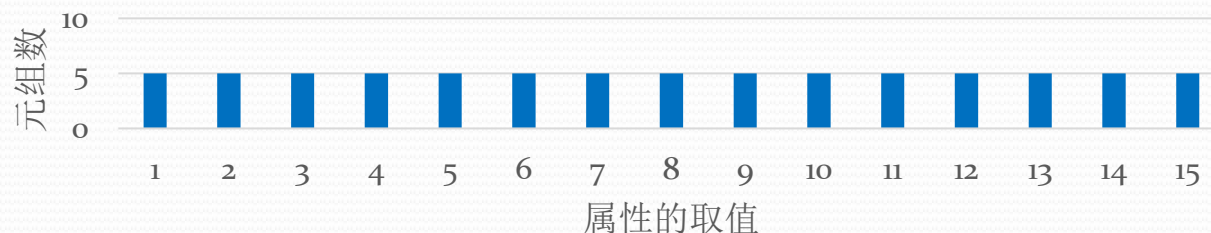
$\text{Sel}(\text{Make} = \text{'Honda'} \text{ and model} = \text{'Accord'})$
 $= \text{Sel}(\text{model} = \text{'Accord'}) = 1/100$

- 原因：属性相关，破坏了独立性假设，需要修改公式

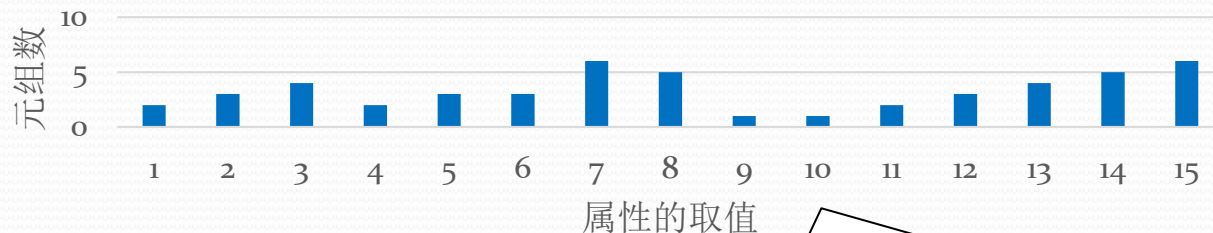
代价模型（续）

- 关于均匀分布假设

假设分布为均匀分布



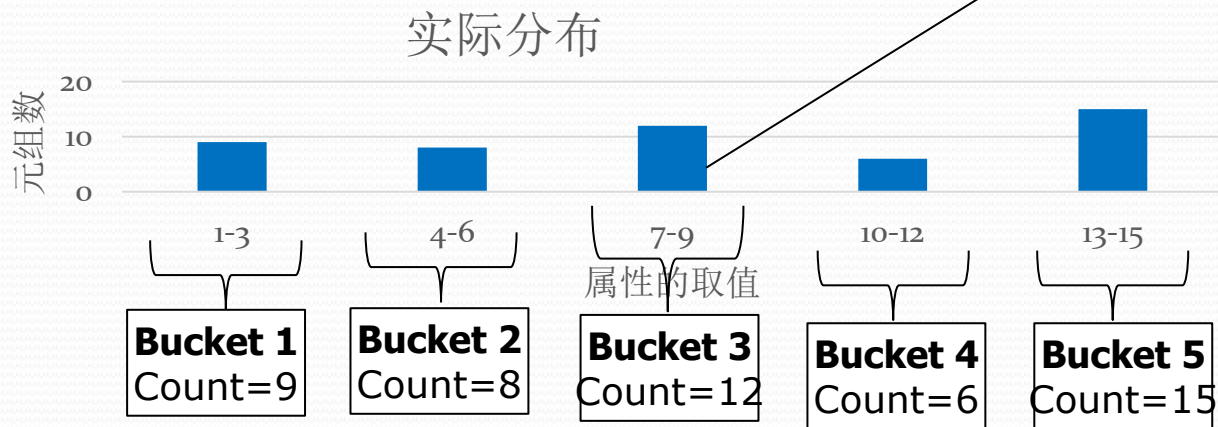
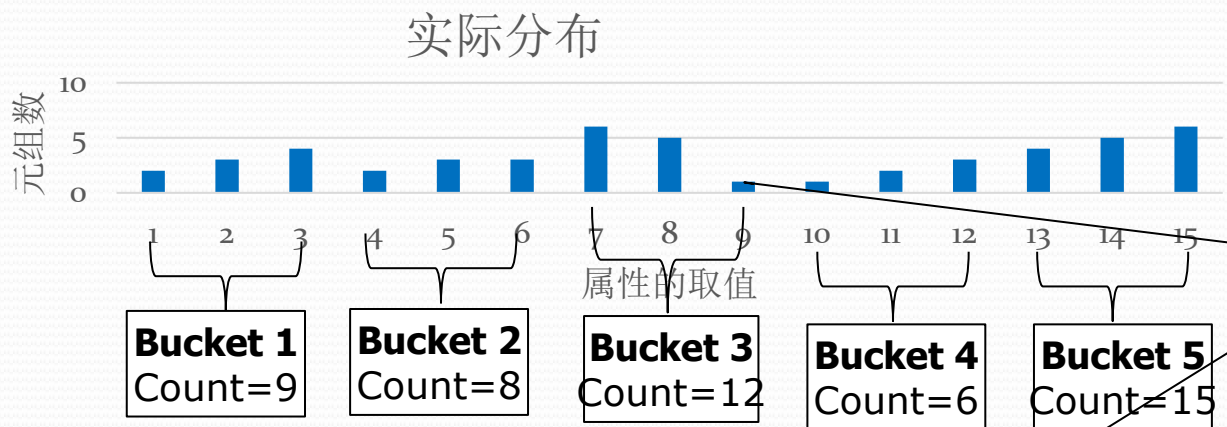
实际分布



直接存储： $15 \times 32\text{bits} = 60\text{Bytes}$ ，如果1500种取值？6KB

代价模型（续）

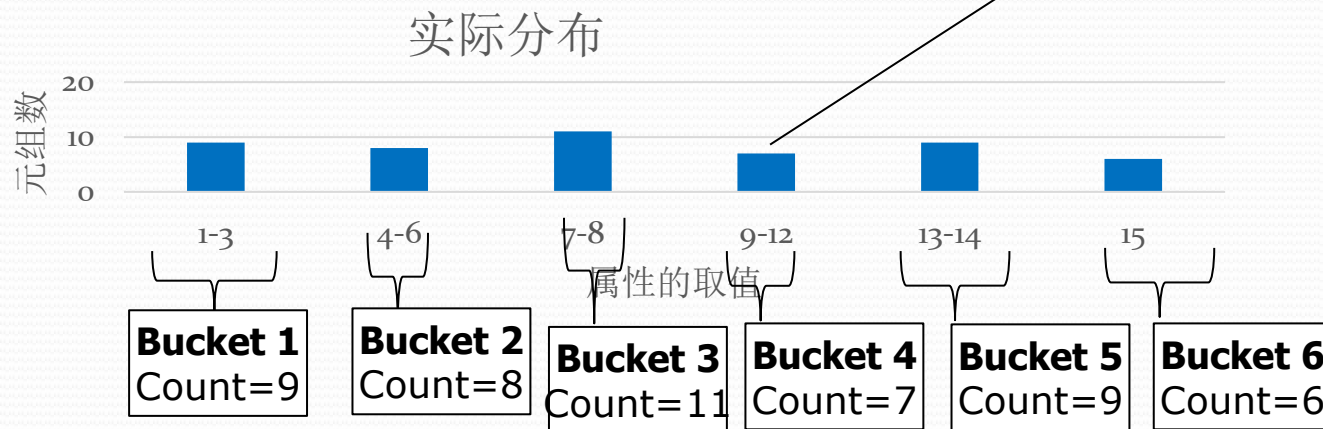
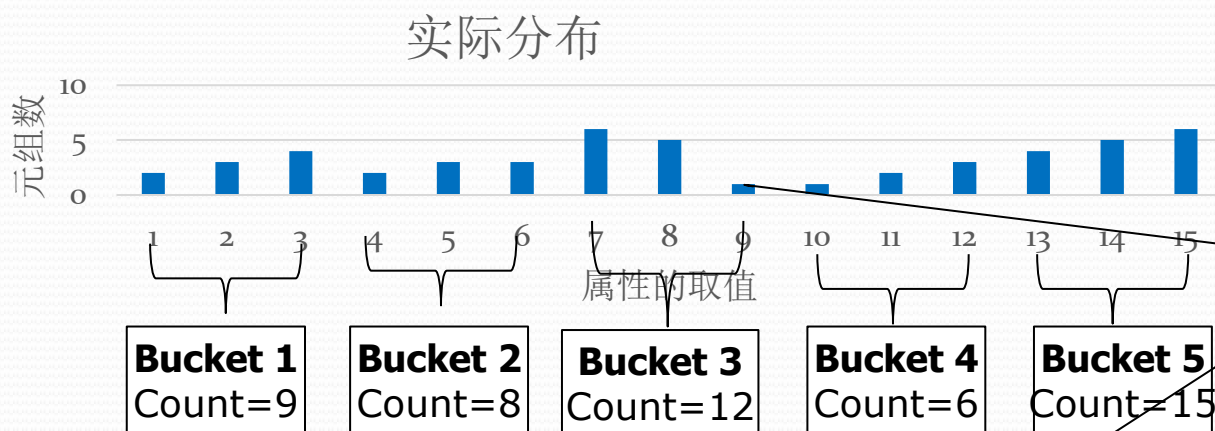
- 等宽直方图
减少存储空间



信息丢失太多，产生误差

代价模型（续）

- 等深直方图
 - 减少存储空间，缓解信息丢失



减缓信息丢失

代价模型（续）

Sketch技术

用概率推断来估计数据的统计特性，牺牲了准确性，但是代价变得很低。

典型例子：

- Count Min Sketch (1988)
- HyperLogLog (2007) 用于redis

代价模型 (续)

- 采样估计
 - 从大表中进行采样，通过采样表估计选择率。

Select AVG(age)
From people
Where age>50

采样表

id	name	age	status
1001	Zhao	59	退休
1003	Sun	25	旷工
1005	Zhou	39	休假

$\text{Sel}(\text{age} \geq 50)$
 $= 1/3$

id	name	age	status
1001	Zhao	59	退休
1002	Qian	41	正常
1003	Sun	25	旷工
1004	Li	26	离职
1005	Zhou	39	休假
1006	Wu	57	正常

...10亿个元组

9.4 物理优化 (续)

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)

- 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; ...



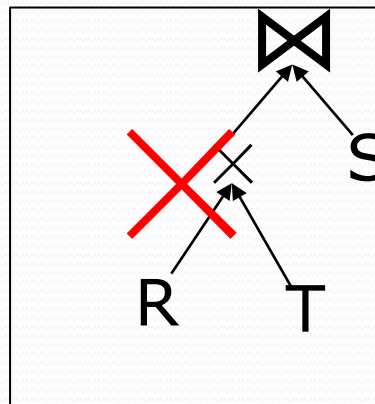
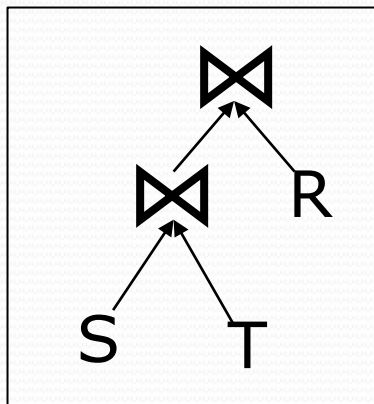
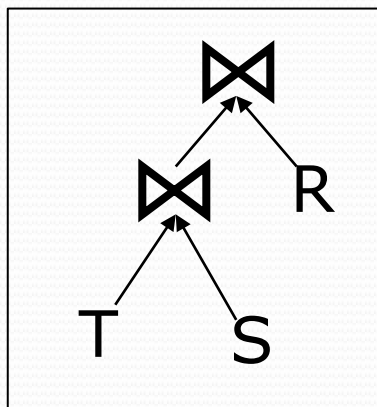
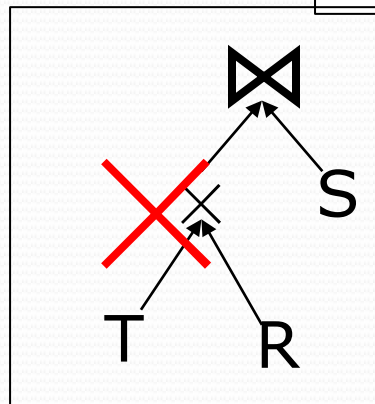
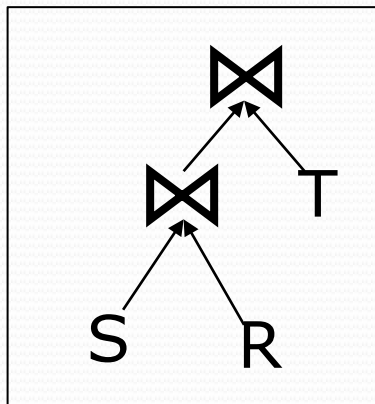
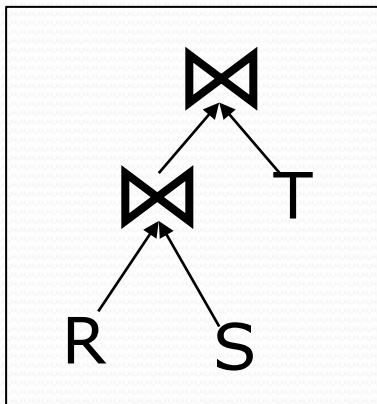
枚举空间巨大

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


计划枚举 (续)

1. 枚举所有可能的连接顺序

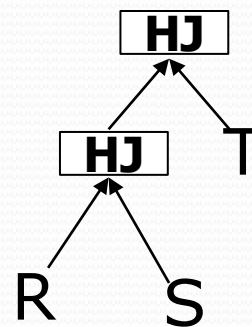
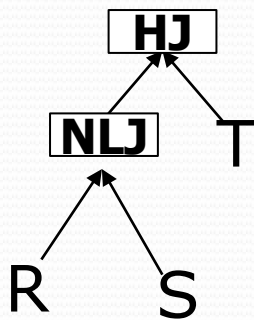
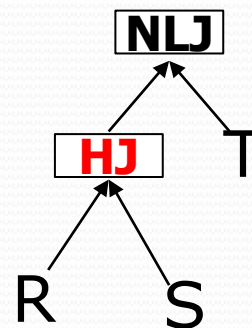
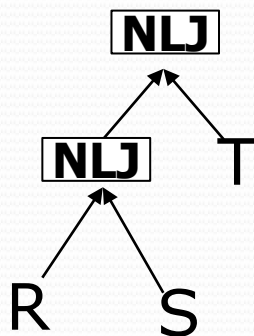
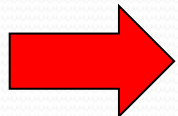
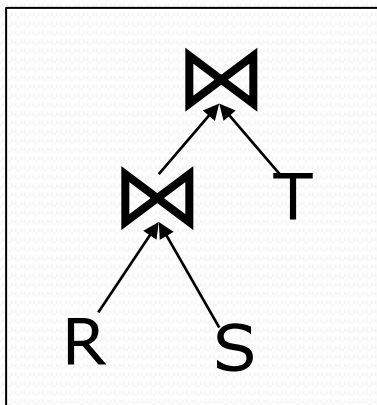
Select * from R,S,T
Where R.a=S.a
And S.b=T.b



计划枚举 (续)

2. 枚举所有可能的连接算法

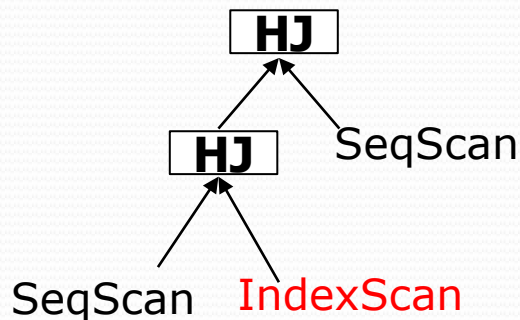
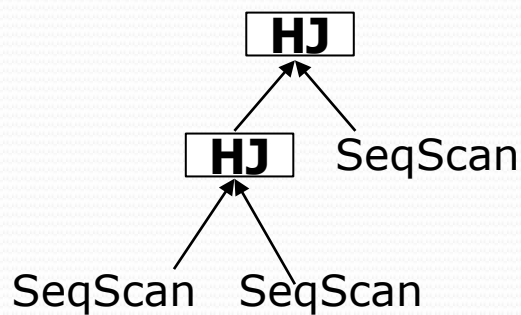
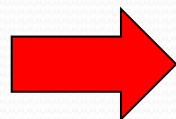
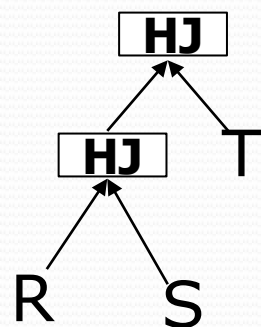
Select * from R,S,T
Where R.a=S.a
And S.b=T.b



计划枚举 (续)

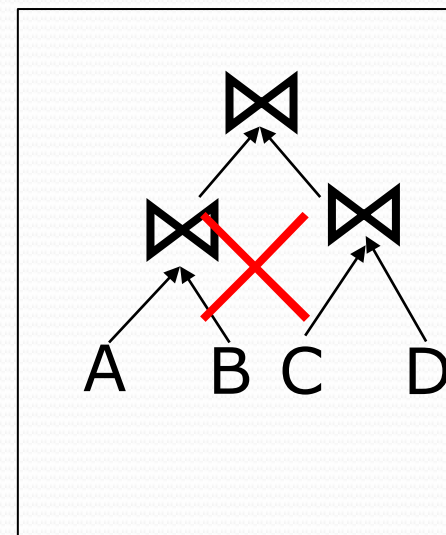
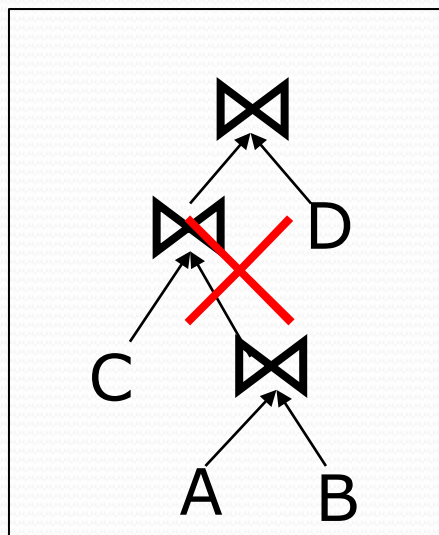
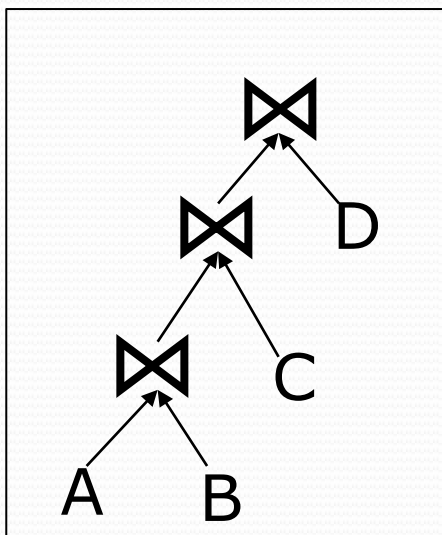
3. 枚举所有可能的数据存取方法

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



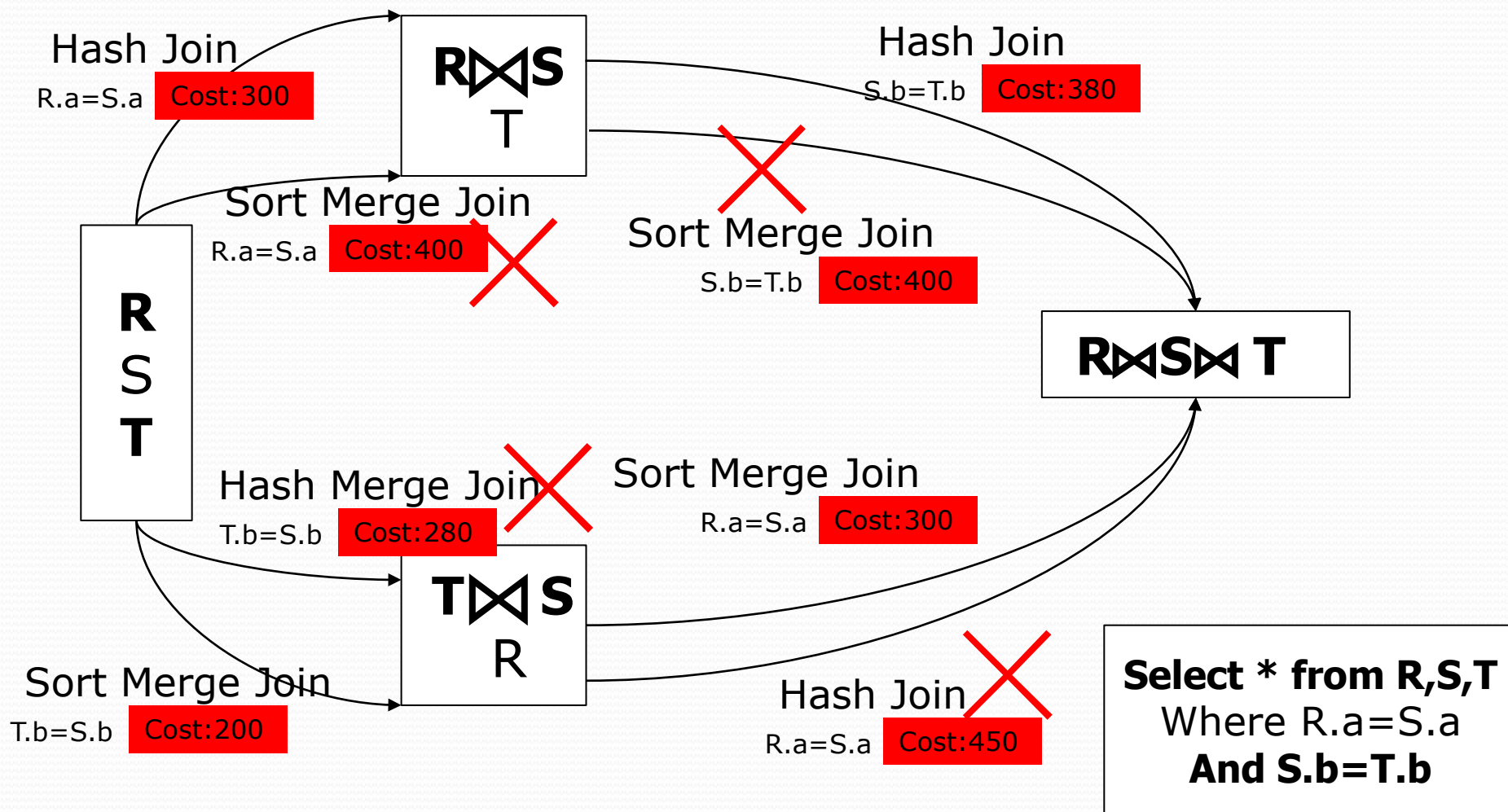
计划枚举（续）

- 基于左深树的物理优化
 - 枚举所有的连接顺序（左深）
 - 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;....

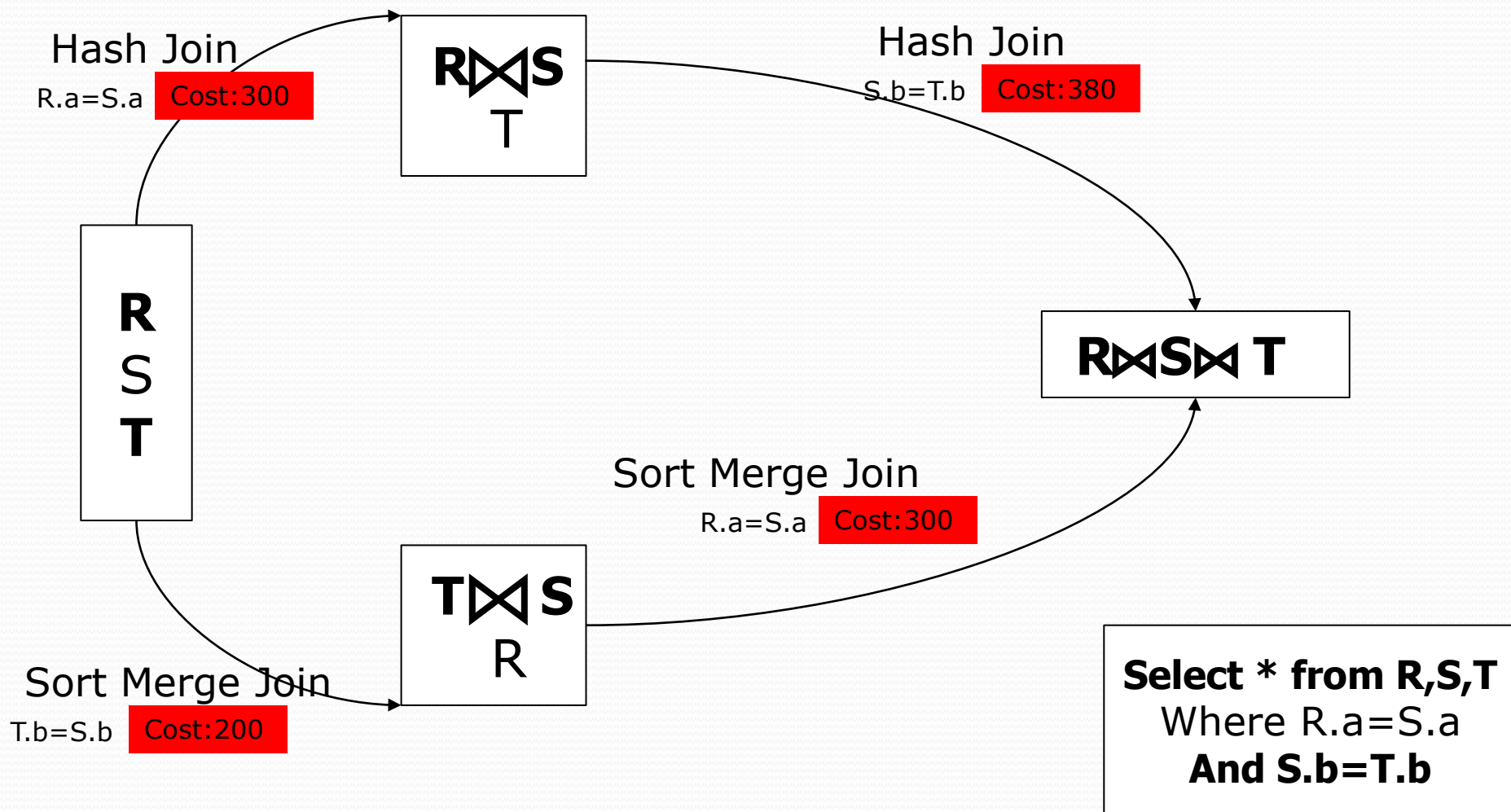
枚举空间巨大

- 可以采用动态规划算法减少搜索空间

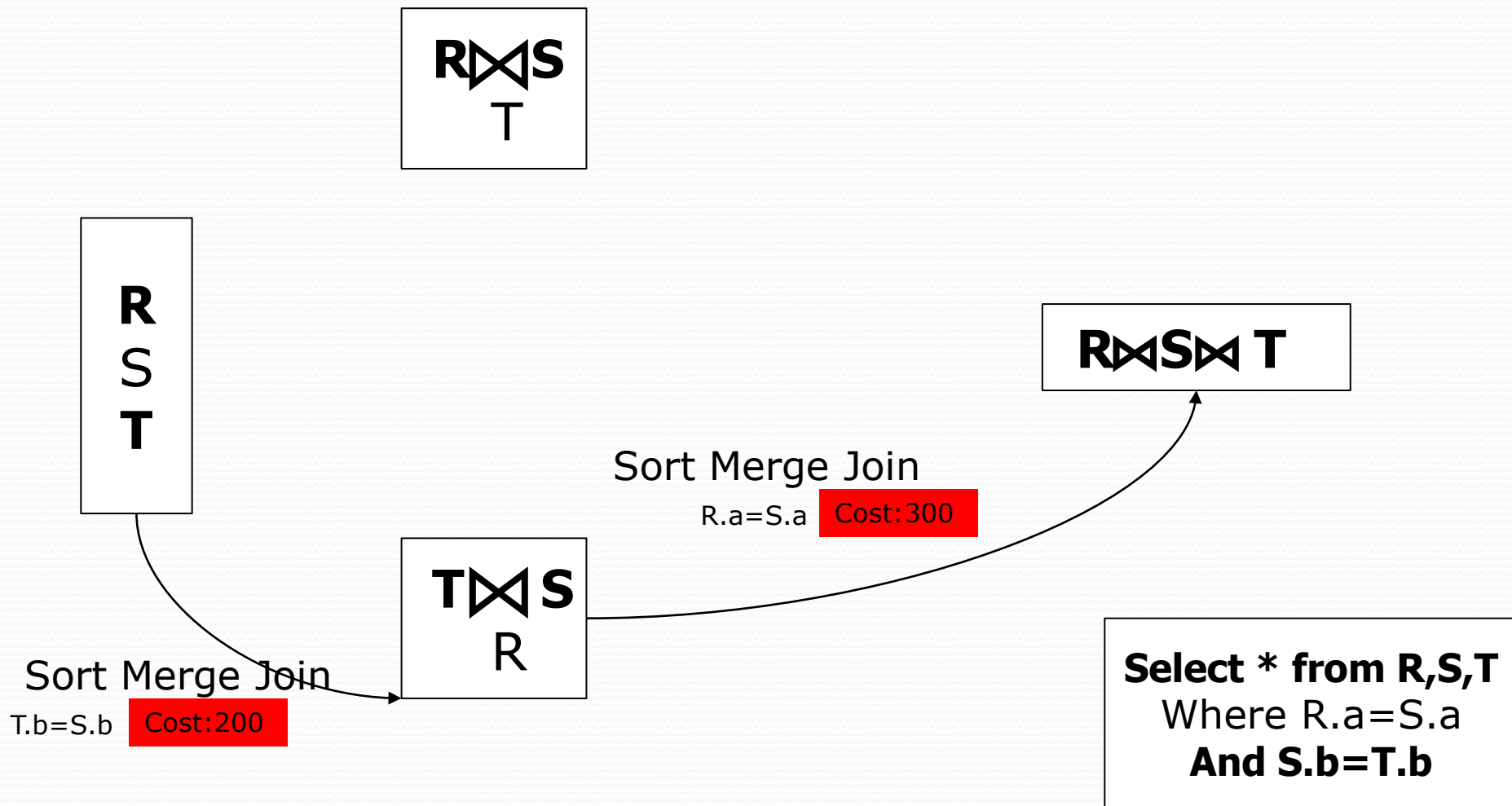
计划枚举 (续)



计划枚举 (续)



计划枚举 (续)



计划枚举 (续)

- POSTGRES优化器
 - 传统的动态规划优化器
 - 基于遗传算法的优化器(GEQO)
 - 当连接表的数量小于12时, 采用动态规划算法
 - 当连接表的数量大于等于12时, 采用GEQO

9.4 物理优化（续）

启发式规则

选择操作的启发式规则

- 1) 对于小关系，使用全表顺序扫描，即使选择列上有索引。
- 2) 对于选择条件是“主码=值”的查询，可以选择主码索引。
(一般的DBMS会自动的建立主码索引)

10%?, 30%?

- 3) 对于选择条件是“非主属性=值”的查询，并且选择列上有索引，**则估算查询结果元组数目，如果比例较小，可以使用索引扫描算法，否则还是使用全表顺序扫描。（DBA监控）**

4) 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，**则估算查询结果的元组数目，如果比例较小，可以使用索引扫描，否则使用全表顺序扫描。**

selectivity

5) 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，则**优先采用组合索引**；如果有多个一般索引，可以用**索引扫描并求交集**的方法；否则采用全表顺序扫描。

6) 对于**用OR连接的析取选择条件**，一般使用**全表顺序扫描**。

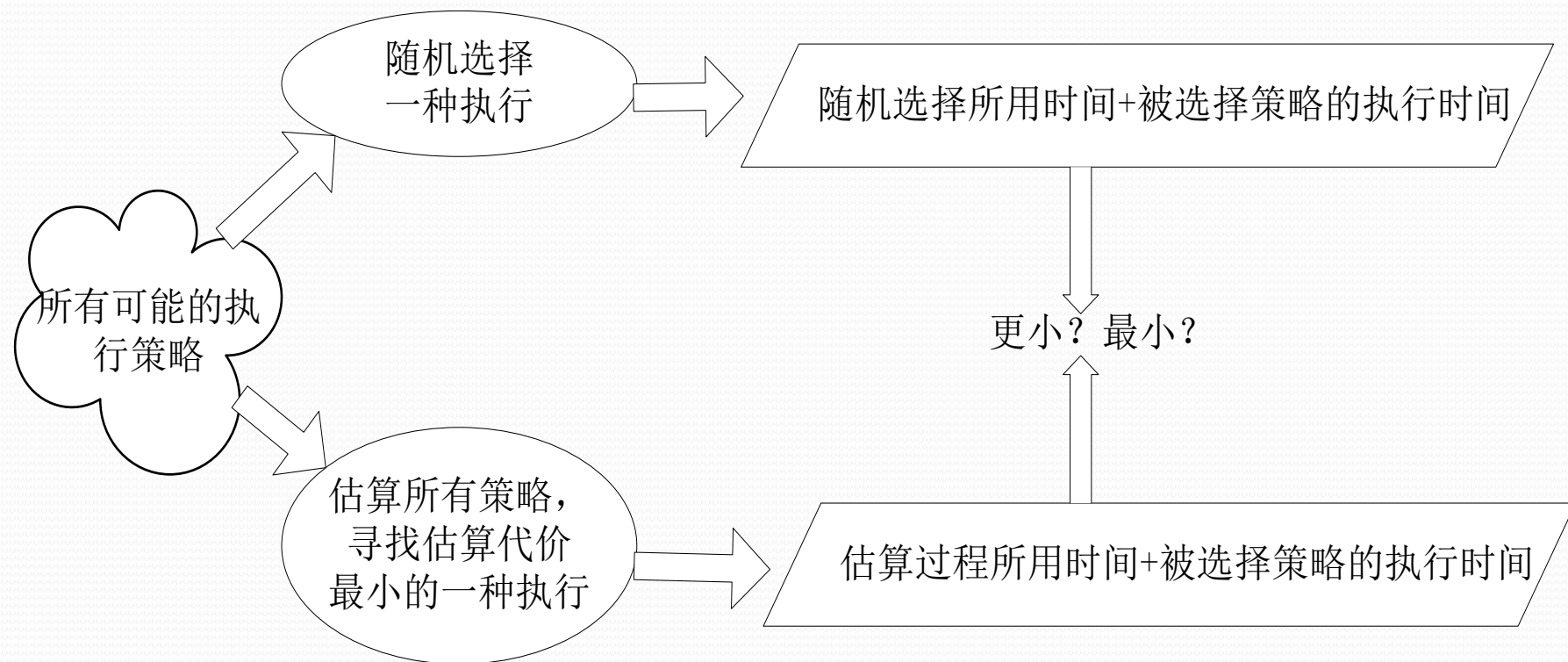
连接操作的启发式规则

- 1) 如果2个表都已按照连接属性排序，则选用merge-sort方法。
- 2) 如果1个表在连接属性上有索引，则可选用索引连接方法。
- 3) 如果上述2个规则均不适用，且其中1个表较小，可选用hash join方法。
- 4) 对于nested loop join方法，选择占用块数较小的表作外表。

问题：内存块的分配？外表的选择？

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。

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。

9.5 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。
- 理解优化的问题需要先了解关系操作的执行过程。


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

数据库行列存储机制在解决社会性突发问题起到的作用

openGauss 行存&列存

Cust_no	Seat_id	Birth_date
1	I_3A	2003-05-01
2	I_3B	2002-02-01
3	I_3C	2002-05-01

行存表

1	I_3A	2003-05-01
2	I_3B	2002-02-01
3	I_3C	2002-05-01

列存表

1	2	3
I_3A	I_3B	I_3C
2003-05-01	2002-02-01	2002-05-01

行存（Row Store）

容易修改记录

读取数据代价高，可能读入不需要的列

不同记录重复率低，压缩率低

列存（Column Store）

修改记录代价高，写一条记录时需要访问多次IO

可以只读取相关的数据

同一列重复值高，压缩率高

慕课讨论题

- 查询优化基本的思路是什么？

关系数据库的查询有其内在执行过程，要考虑逻辑、物理等多个层面的因素，在这其中查询优化的基本思路是什么？

- 在哪些情况下利用索引可以取得较好的优化效果？

关系数据库可以为关系建立索引，但是却不一定所有的关系都要建立索引，在哪些情况下利用索引可以取得较好的优化效果？