第九章 关系数据库查询优化

9.1关系数据库系统的查询优化

9.2 逻辑优化

9.3 物理优化

1 查询优化概述

- 查询优化的必要性
- 查询优化的可能性

2 查询优化的方法

查询优化极大地影响RDBMS的性能。

1 查询优化的必要性（续）

例：求选修了课程C2的学生姓名

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

1 查询优化的必要性(续)

- 假设1：外存：
  - Student:1000条,SC:10000条，选修2号课程:50条
  - 一个硬盘块放10个student或者100个SC
- 假设2：一次I/O交换元组:10个Student，或100个SC，内存中一次可以存放：5块Student元组（即50个Student），1块SC元组（即100个SC）和若干块连接结果元组
- 假设3：读写速度：20块/秒
- 假设4：连接方法：基于数据块的嵌套循环法
- 假设5：查询执行的方法采用物化模型

1 查询优化的必要性(续)

1 Q1 = $\Pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='C2'}(Student \times SC))$

① Student×SC

读取总块数= 读Student表块数 + 读SC表遍数*每遍块数

(读SC表遍数=Student表的总元组数/在内存中的元组数)

=1000/10+(1000/(10×5)) ×(10000/100)

=100+20×100=2100

读数据时间=2100/20=105秒

1 查询优化的必要性(续)

中间结果大小 = 1000*10000 = 10^7　　(1千万条元组)

写中间结果时间 = 10000000/10/20 = 50000秒

②б

读数据时间 = 50000秒

③П

总时间 =105 + 50000 + 50000秒 = 100105秒

= 27.8小时

1 查询优化的必要性(续)

2. $Q2 = \Pi_{S\ name}(\sigma_{SC.Cno='C2'}(Student \bowtie SC))$

① $\bowtie$

读取总块数= 2100块（Q1的结果）读数据时间=2100/20=105秒

因为只有SC只有10000条元组，故等值连接的结果,即:中间结果大小=10000 （减少1000倍）

写中间结果时间=10000/10/20=50秒

②σ

读数据时间=50秒

③Π

总时间 = 105 + 50 + 50秒 = 205秒=3.4分（减少了中间结果）

1 查询优化的必要性(续)

3. Q3 = Π_{Sname}(Student ⋈ б$_{SC.Cno='C2'}$ (SC))

①б

读SC表总块数= 10000/100=100块

读数据时间=100/20=5秒

中间结果大小=50条 不必写入外存

② ⋈

读Student表总块数= 1000/10=100块

读数据时间=100/20=5秒

③ Π

总时间 = 5 + 5秒 = 10秒 （减少中间结果,且全部在内存）

1 查询优化的必要性(续)

4. $Q2 = \Pi_{S\ name}(Student \bowtie б_{SC.Cno='C2'}(SC))$

假设SC表在Cno上有索引，Student表在Sno上有索引

①б

读SC表索引=（发现C2号课程只有50条元组）

读SC表总块数= 50/100<1块

读数据时间:1/20=0.5

中间结果大小=50条 不必写入外存

1 查询优化的必要性(续)

② ⋈

读Student表索引=(根据索引发现只有50个学生)

读Student表总块数= 50/10=5块

读数据时间： 5/20=0.5秒

③ Π

总时间<1~2秒（不必遍历所有的元组记录）

2 查询优化的可能性

- 用户不必考虑如何最好地表达查询以获得较好的效率。
- 关系数据语言的级别很高，使DBMS可以从关系表达式中分析查询语义。
- 系统可以比用户程序的优化做得更好。

2 查询优化的可能性(续)

Student×SC（做SC×Student）

读取总块数= 读SC表块数 + 读Student表遍数

*每遍块数

(读Student表遍数=SC表的总元组数/在内存中的元组数)

=10000/100+(10000/(100×1)) ×(1000/10)

=100+100×100=10100

读数据时间=10100/20=505秒

2 查询优化的可能性(续)

- 优化器可以从数据字典中获取许多信息(包括统计信息和索引信息)，而用户程序则难以获得这些信息。
- 如果数据库的物理统计信息改变了，系统可以自动对查询重新优化以选择相适应的执行计划。在非关系系统中必须重写程序，而重写程序在实际应用中往往是不太可能的。
- 优化器可以考虑数百种不同的执行计划，而程序员一般只能考虑有限的几种可能性。
- 优化器中包括了很多复杂的优化技术

1 查询优化概述

2 查询优化的方法

- 查询优化的方法
  - 启发式（规则式）
    - 主要利用关系代数等价变换进行表达式重写，又称代数优化
    - 基于规则进行优化
    - 该方法通常会访问数据字典，但是不会检查数据
  - 代价模型
    - 使用数学模型对查询代价进行评估
    - 提出多个等价查询计划，根据模型评估执行代价，选择代价最低的查询计划

- 逻辑计划和物理计划
  - 物理计划是逻辑计划的具体实现
  - 一个逻辑计划存在一个最优的物理计划的执行
  - 物理计划依赖于数据存储的方法：排序，索引等

9.1 关系数据库系统的查询优化

9.2 逻辑优化

9.3 物理优化

9.2.1 代数优化的一般准则

9.2.2 关系代数等价变换规则

9.2.3 关系代数表达式的优化算法

9.2.4 代数优化举例

- 选择运算应尽可能先做
  - 目的：减小中间关系
- 在执行连接操作前对关系适当进行预处理
  - 按连接属性排序
  - 在连接属性上建立索引
- 投影运算和选择运算同时做
  - 目的：避免重复扫描关系
- 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

- 某些选择运算 + 在其前面执行的笛卡尔积

  ===>　连接运算

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

  ⇩

  $Student \bowtie SC$

- 提取公共子表达式

9.2.1 代数优化的一般准则

9.2.2 关系代数等价变换规则

9.2.3 关系代数表达式的优化算法

9.2.4 代数优化举例

- 关系代数表达式等价
  - 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
  - 上面的优化策略大部分都涉及到代数表达式的变换

设E1、E2等是关系代数表达式，F是条件表达式

1. 连接、笛卡尔积交换律

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

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

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

2. 连接、笛卡尔积的结合律

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

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

$$(E1 \overset{F}{\bowtie} E2) \overset{F}{\bowtie} E3 \equiv E1 \overset{F}{\bowtie} (E2 \overset{F}{\bowtie} E3)$$

3. 投影的串接定律

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

假设：

1) E是关系代数表达式

2) $A_i(i=1, 2, \ldots, n)$, $B_j(j=1, 2, \cdots, m)$是属性名

3)$\{A_1, A_2, \cdots, A_n\}$构成$\{B_1, B_2, \cdots, B_m\}$的子集

4. 选择的串接定律

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

- 选择的串接律说明　选择条件可以合并
- 这样一次就可检查全部条件。

5. 选择与投影的交换律

(1)假设: 选择条件F只涉及属性A1，…，An

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

(2)假设: F中有不属于A1, …,An的属性B1,…,Bm

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

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

6. 选择与笛卡尔积的交换律

(1) 假设：F中涉及的属性都是E1中的属性

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

(2) 假设：$F=F1\wedge F2$，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：

$$б_F(E1\times E2)\equiv б_{F1}(E1)\times б_{F2}(E2)$$

(3) 假设： $F=F1 \wedge F2$,

F1只涉及E1中的属性,

F2涉及E1和E2两者的属性

$$б_F(E1 \times E2) \equiv б_{F2}(б_{F1}(E1) \times E2)$$

它使部分选择在笛卡尔积前先做

7. 选择与并的交换

假设：E=E1∪E2，E1，E2有相同的属性名

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

8. 选择与差运算的交换

假设：E1与E2有相同的属性名

$б_F(E1-E2) \equiv б_F(E1) - б_F(E2)$

9. 选择对自然连接的分配律

假设：E1和E2是两个关系表达式，

б（E1⋈E2)≡б（E1)⋈б(E2)

10. 投影与笛卡尔积的交换

假设：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

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

11. 投影与并的交换

假设：E1和E2 有相同的属性名

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

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

9.2.1 代数优化的一般准则

9.2.2 关系代数等价变换规则

9.2.3 关系代数表达式的优化算法

9.2.4 代数优化举例

算法：关系表达式的优化

输入：一个关系表达式的语法树。

输出：计算该表达式的程序。

方法：

（1）分解选择运算

利用规则4把形如$\sigma_{F1 \wedge F2 \wedge \cdots \wedge Fn}(E)$变换为

$\sigma_{F1}(\sigma_{F2}(\cdots(\sigma_{Fn}(E))\cdots))$

（2）通过交换选择运算，将其尽可能移到叶端

对每一个选择，利用规则4～9尽可能把它移到树的叶端。

（3）通过交换投影运算，将其尽可能移到叶端

对每一个投影利用规则3，10，l1，5中的一般形式尽可能把它移向树的叶端。

（4）合并串接的选择和投影，以便能同时执行或在一次扫描中完成

- 利用规则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
- 尽管这种变换似乎违背“投影尽可能早做”的原则，但这样做效率更高。

(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。

9.2.1 代数优化的一般准则

9.2.2 关系代数等价变换规则

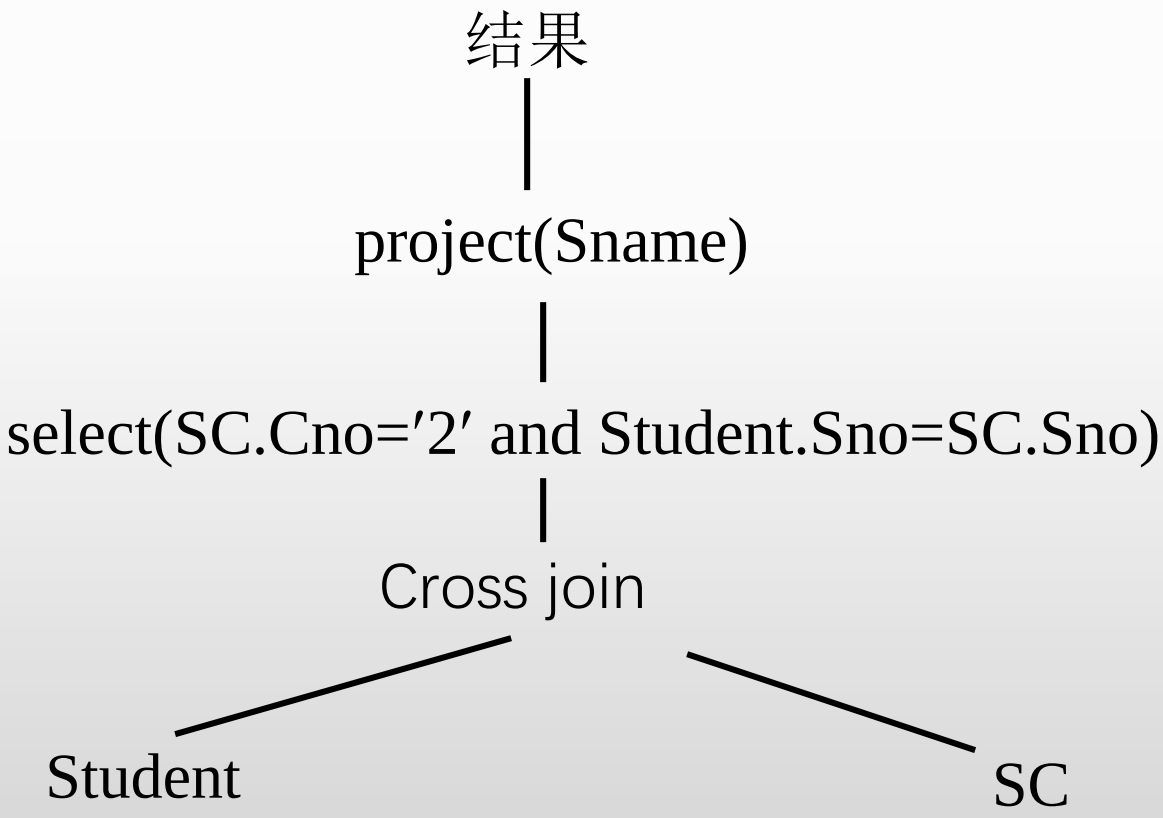
9.2.3 关系代数表达式的优化算法

9.2.4 代数优化举例

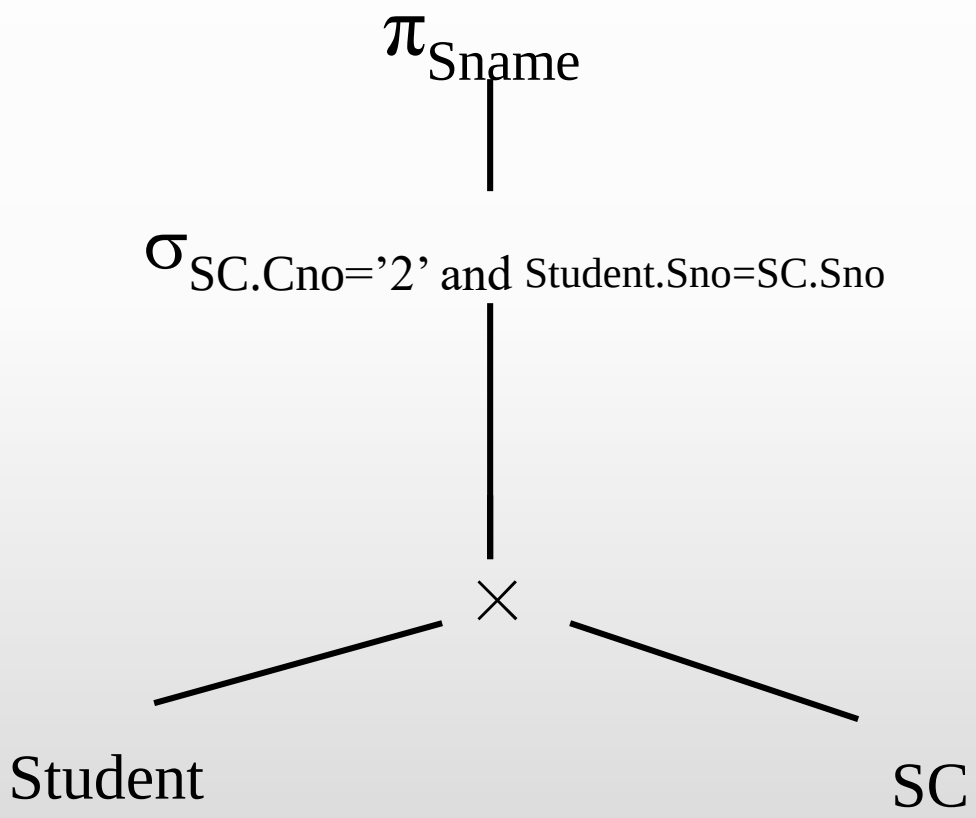
例：求选修了课程C2的学生姓名

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

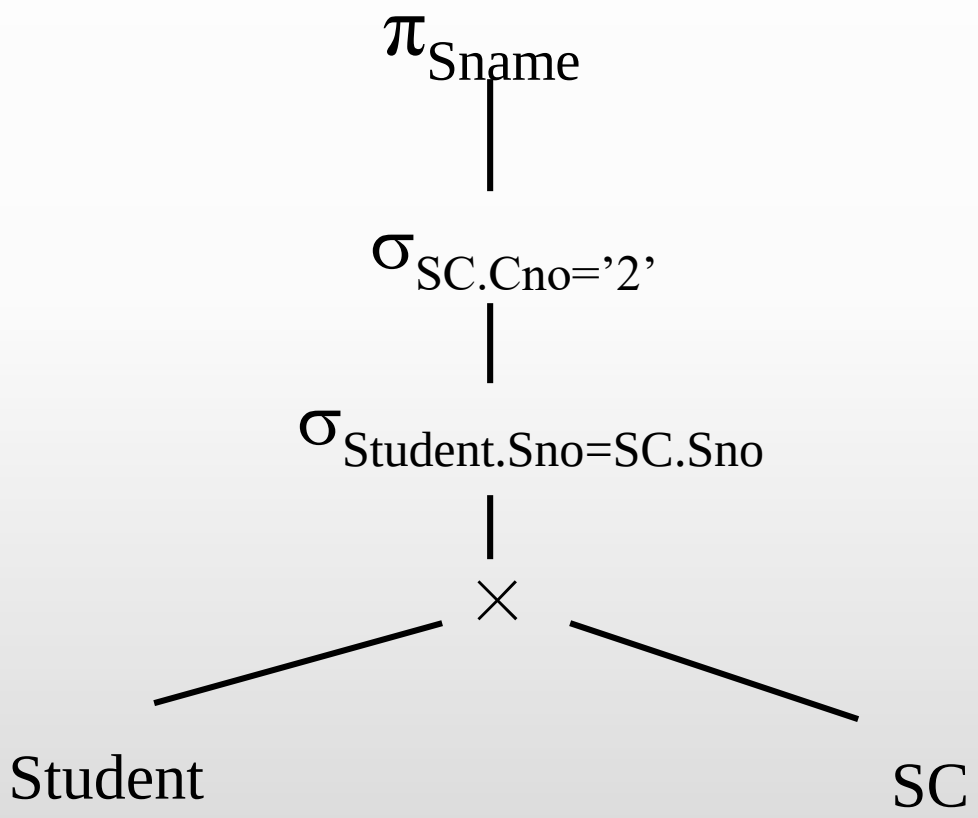
（1）把查询转换成某种内部表示



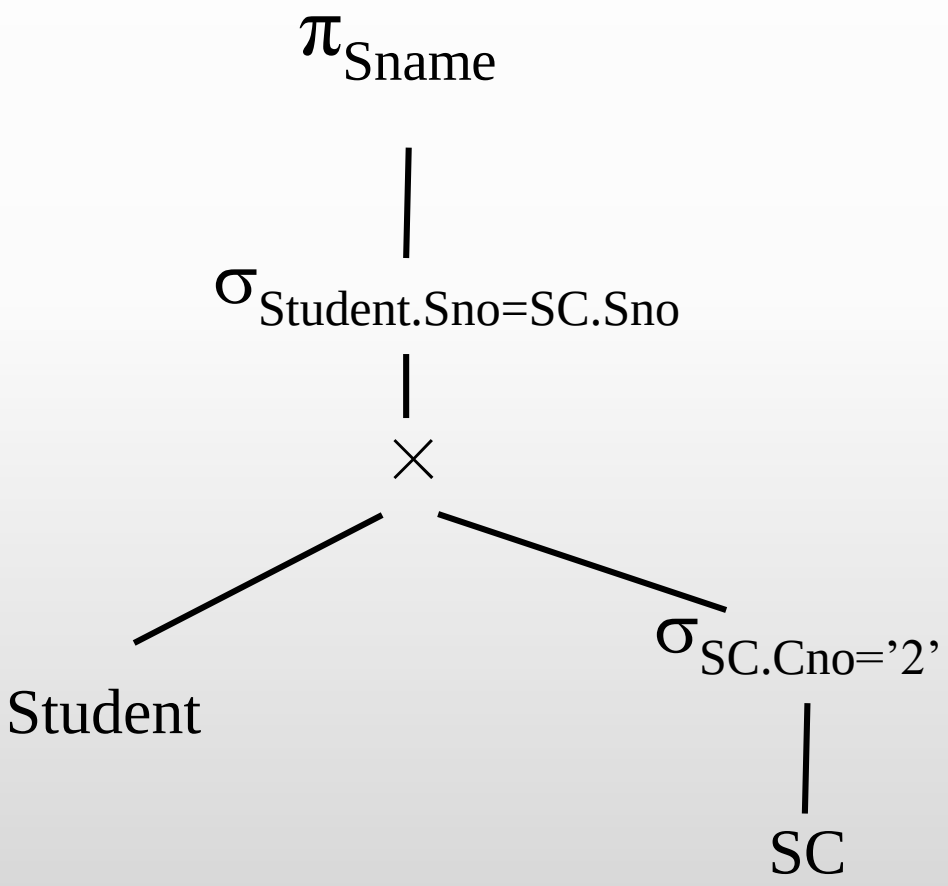
（2）转换为关系代数表达式



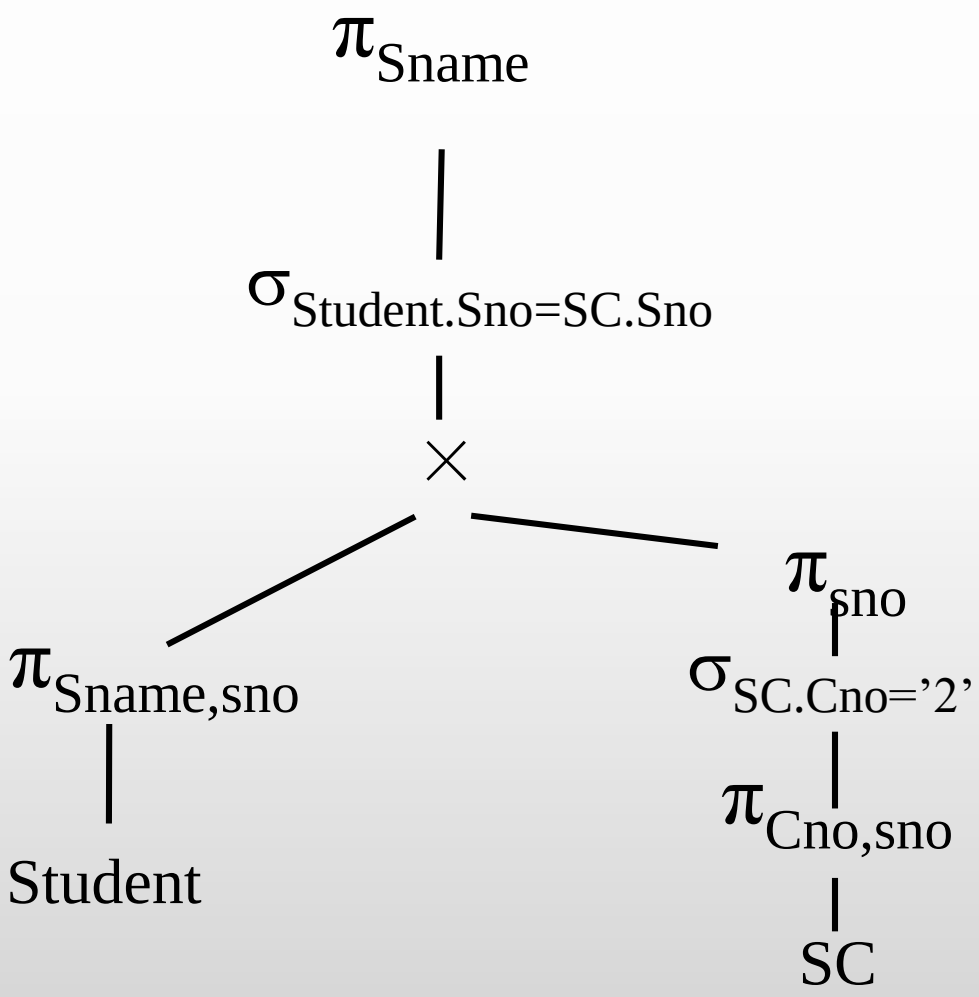
（3）分解选择（规则4）



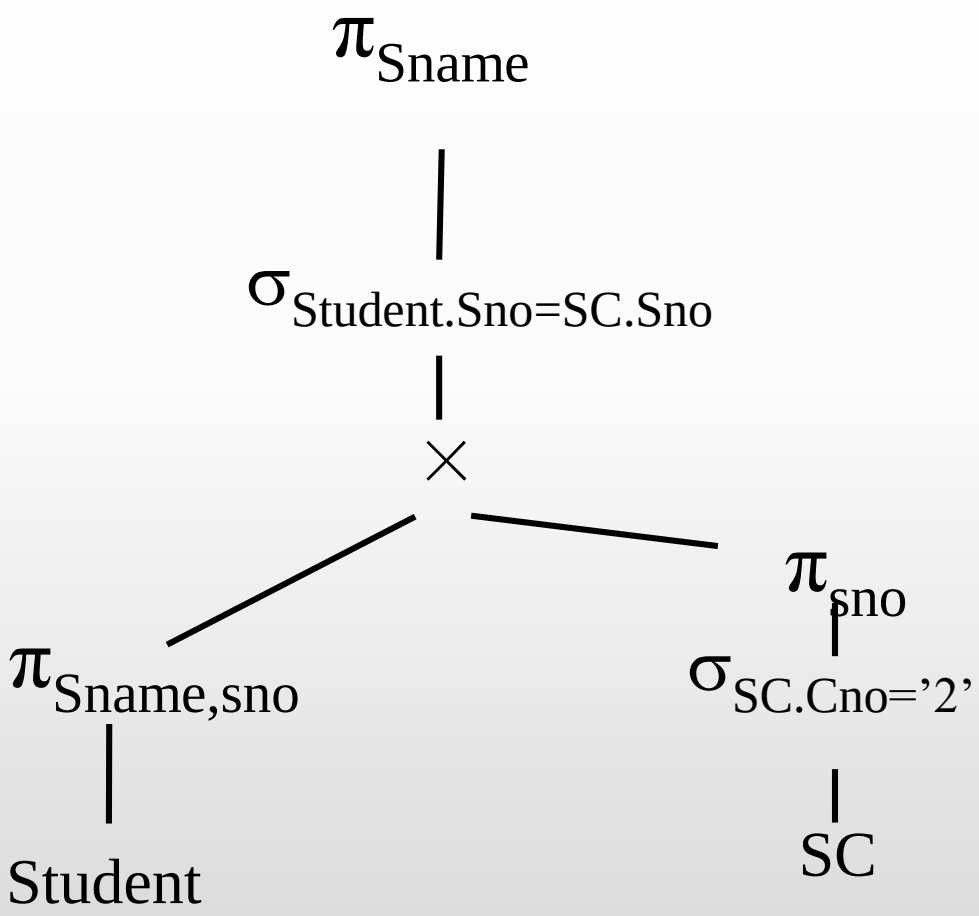
（4）选择下移



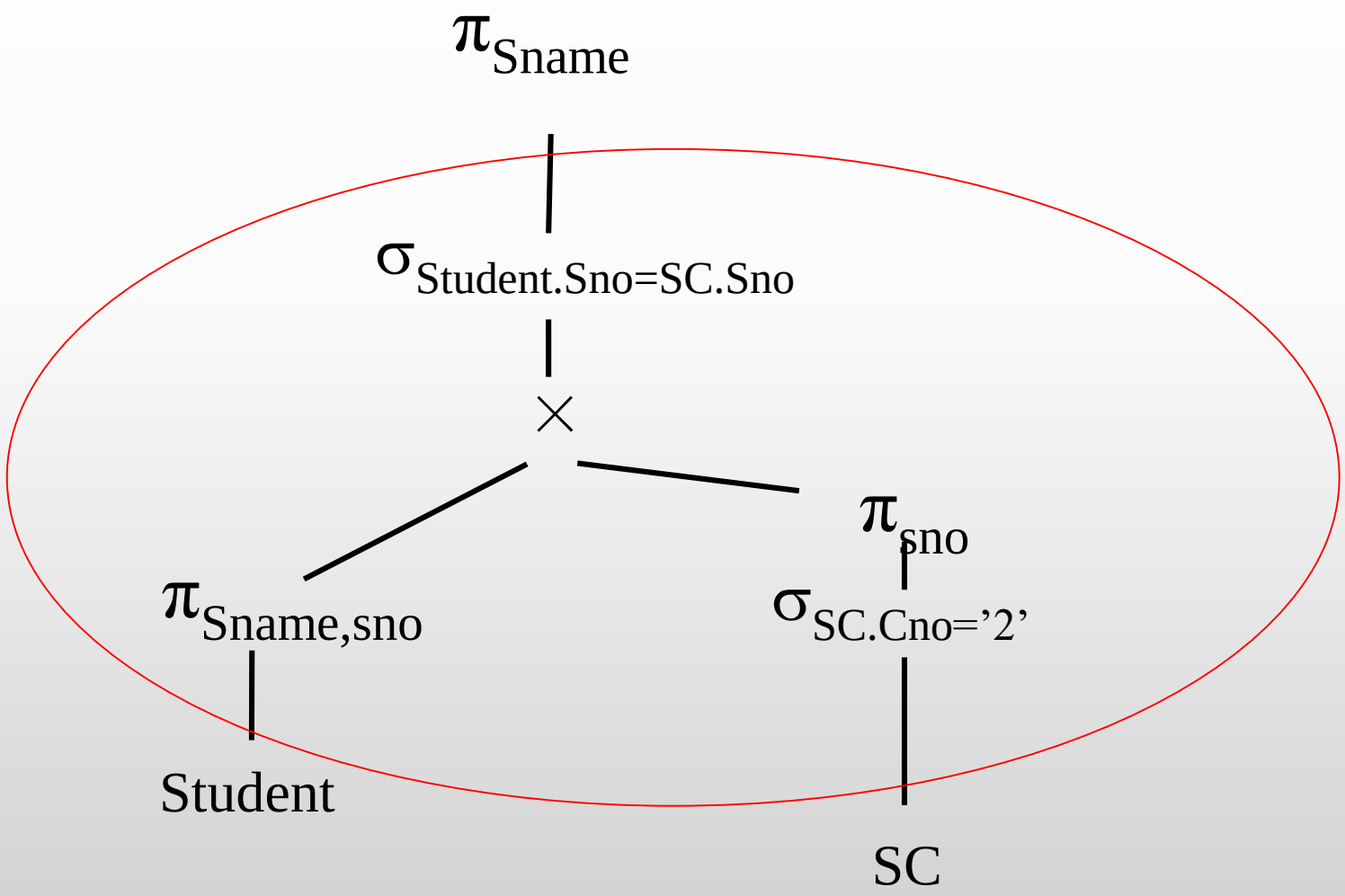
(5) 投影下移


πSname
σStudent.Sno=SC.Sno
×
πSname,sno
Student
πsno
σSC.Cno='2'
πCno,sno
SC

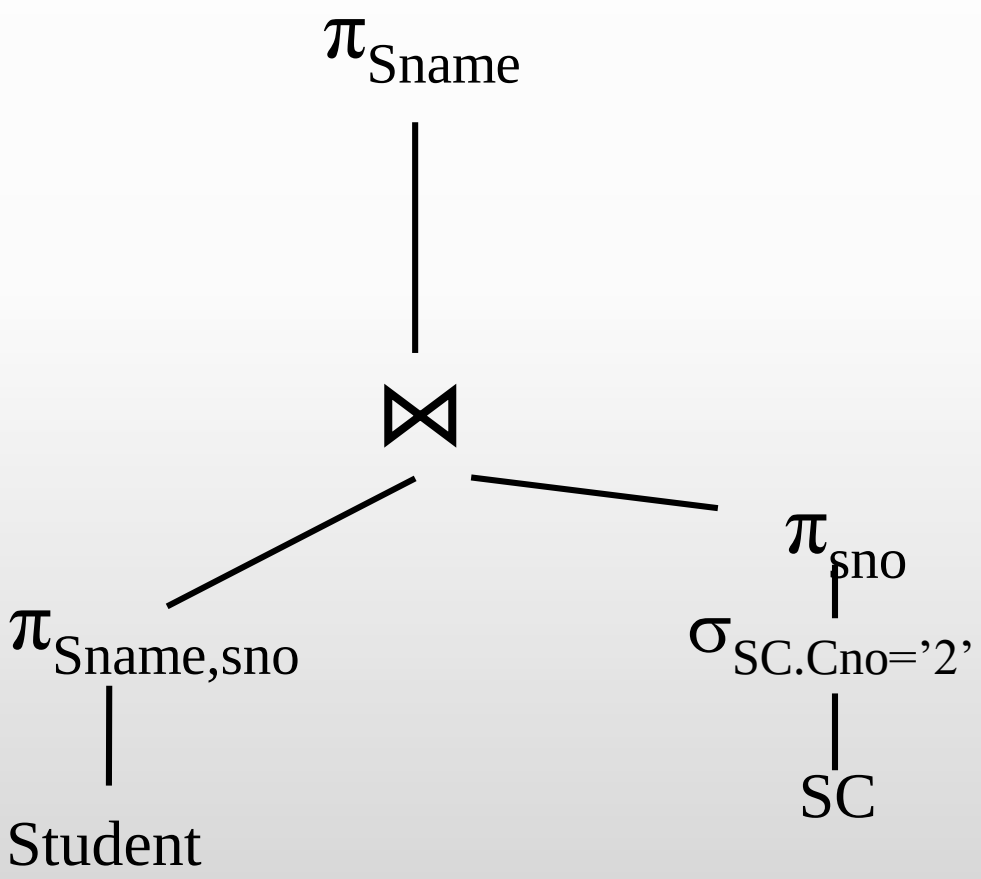
(6) 串接合并


πSname
σStudent.Sno=SC.Sno
×
πSname,sno
Student
πsno
σSC.Cno='2'
SC

（6）分组，恰好笛卡尔积能和其前面的选择运算组成自然连接



（7）代数优化后结果



9.1 关系数据库系统的查询优化

9.2 逻辑优化

9.3 物理优化

9.3.1 代价估计

9.3.2 计划枚举

- 代价估计
  - 基于代价模型对于一个特定查询计划的执行代价进行估计。
- 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

- 基于磁盘DBMS代价模型
  - 磁盘IO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IO或随机IO
  - 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

- 统计信息
  - 代价估计依赖统计信息
  - 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
  - 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

- 重要统计信息
  - Br(块数)，fr（块因子）
  - N_R:关系R的元组的数量
  - V(A,R):关系R中属性A的distinct的元组数量
    - 如学生关系S中，V('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。

    $SC(A,R)= N_R /V(A,R)$
    - 选择基数的表明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。

- 复合谓词选择率估计
  - 选择率（selectivity）： 谓词P的选择率指的是复合谓词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

假设V(age,people)具有5个不同的值（18-22）且N_R=5。

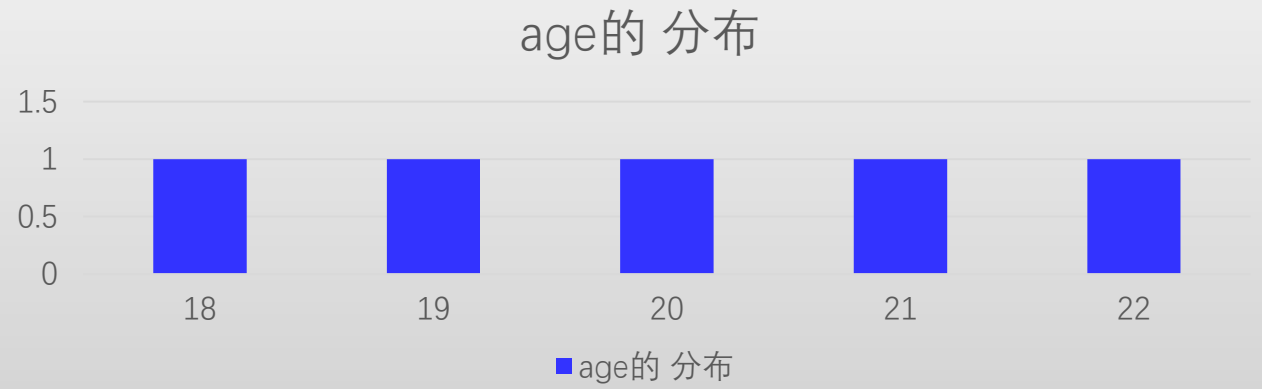
- 相等谓词

查询：Select * from People where age=20

相等谓词：A=constant

相等谓词的选择率：Sel(A=constant)=SC(P)/N_R

Sel(age=20)=1/5

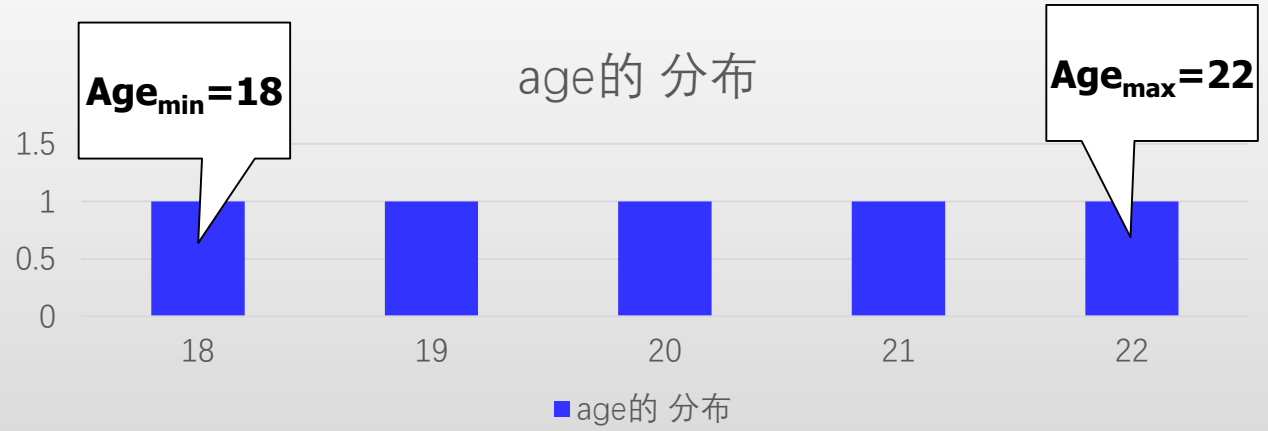


- 范围谓词

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

查询：Select * from People where age>=20

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

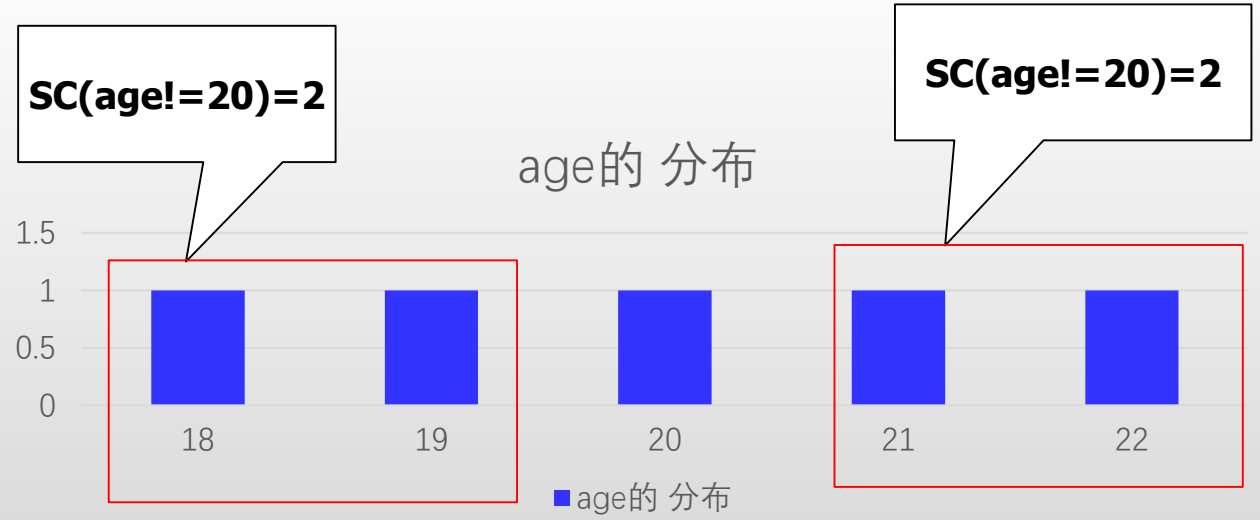


- 非谓词

Sel(!P)=1-sel(P)

查询：Select * from People where age!=20

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



- 与谓词

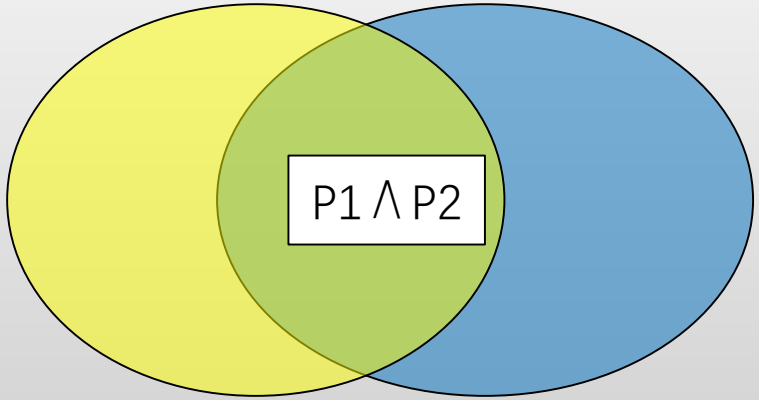
  当谓词之间的选择独立时

  Sel(P1∧P2)=sel(P1)×sel(P2)

  查询:

  Select * from People

  where age=20 and name like ' 王%'



- 或谓词

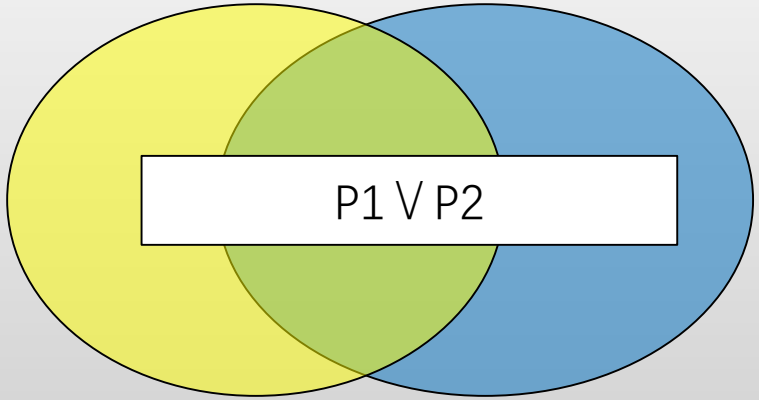
当谓词之间的选择独立时

Sel(P1 ∨ P2)=sel(P1)+sel(P2)-Sel(P1 ∧ P2)

= sel(P1)+sel(P2)-Sel(P1)×Sel(P2)

查询：Select * from People

where age=20 or name like ’王%’



- 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

    假设 $R_{cols} \cap S_{cols}=\{A\}$
  - 假设R中的每一个元组在S中至少能找到一个匹配

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

    $estSize = N_S \times N_R / V(A,R)$

- 假设1：数据均匀分布
- 假设2：谓词的选择性是独立
- 假设3：内表的每个元组能在外表中找到匹配

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：Sel(P1∧P2)=sel(P1)×sel(P2)

- ~~相等谓词~~

  ~~相等谓词的选择率：Sel(A=constant)=SC(P)/N_R~~

- ~~错误的选择率~~

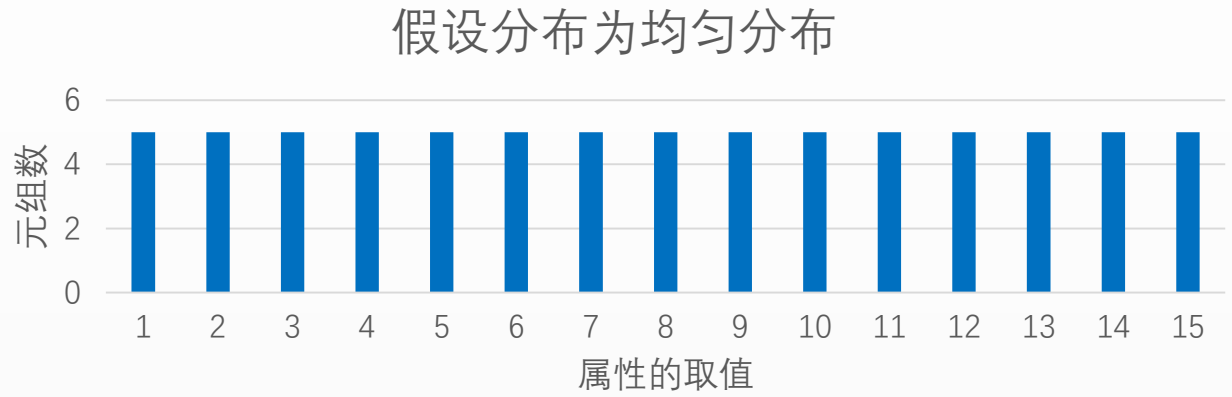
  ~~Sel（Make=‘Honda’ and model =‘Accord’）=1/10×1/100=0.001~~

- 正确的选择率

  Sel（Make=‘Honda’ and model =‘Accord’）=Sel(model =‘Accord’)=1/100

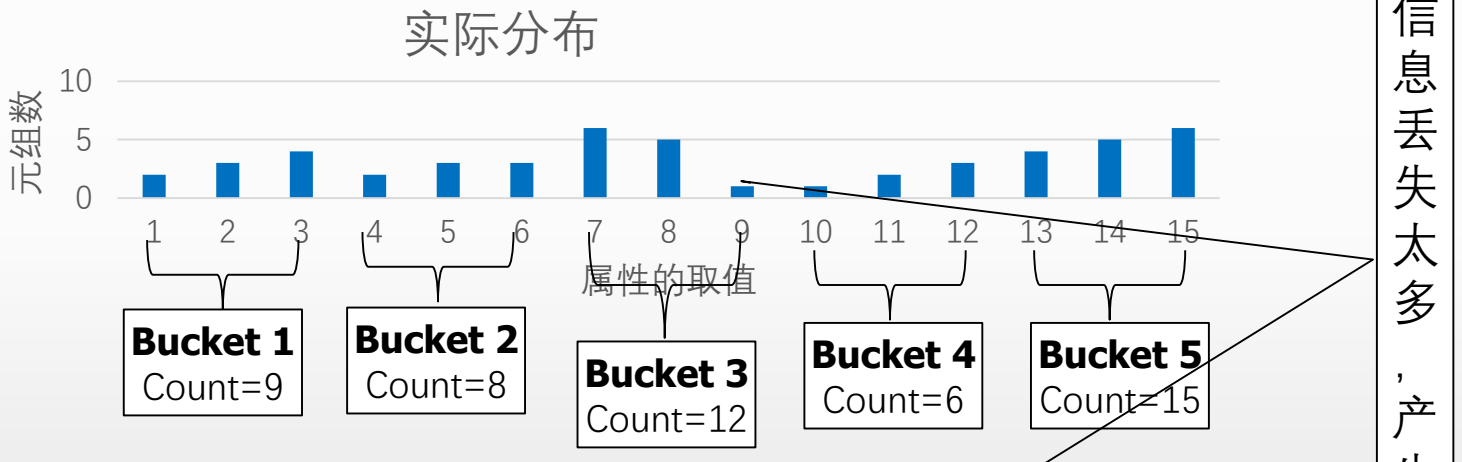
- 原因：属性相关,破坏了独立性假设，修改公式

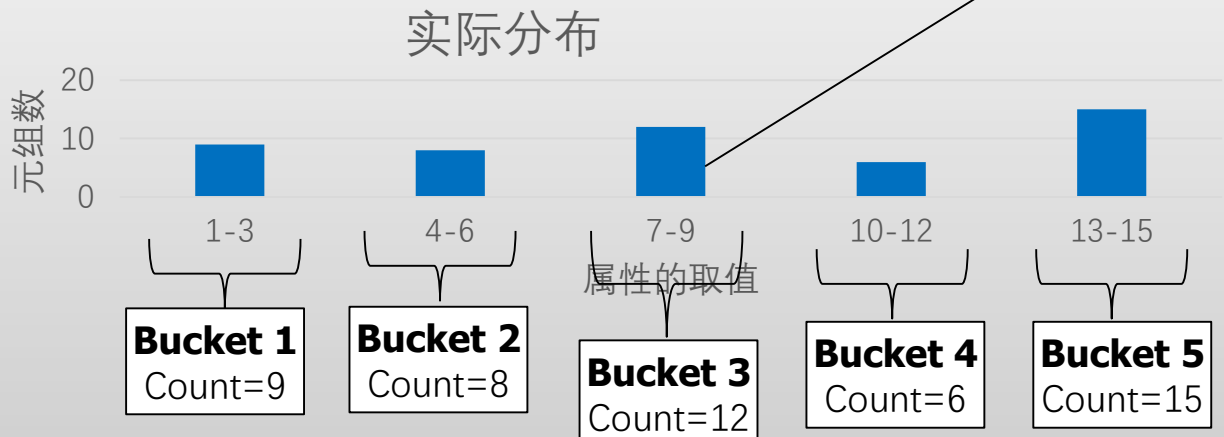
➢关于均匀分布假设



直接存储：15×32bits=60Bytes，如果1500？46KB

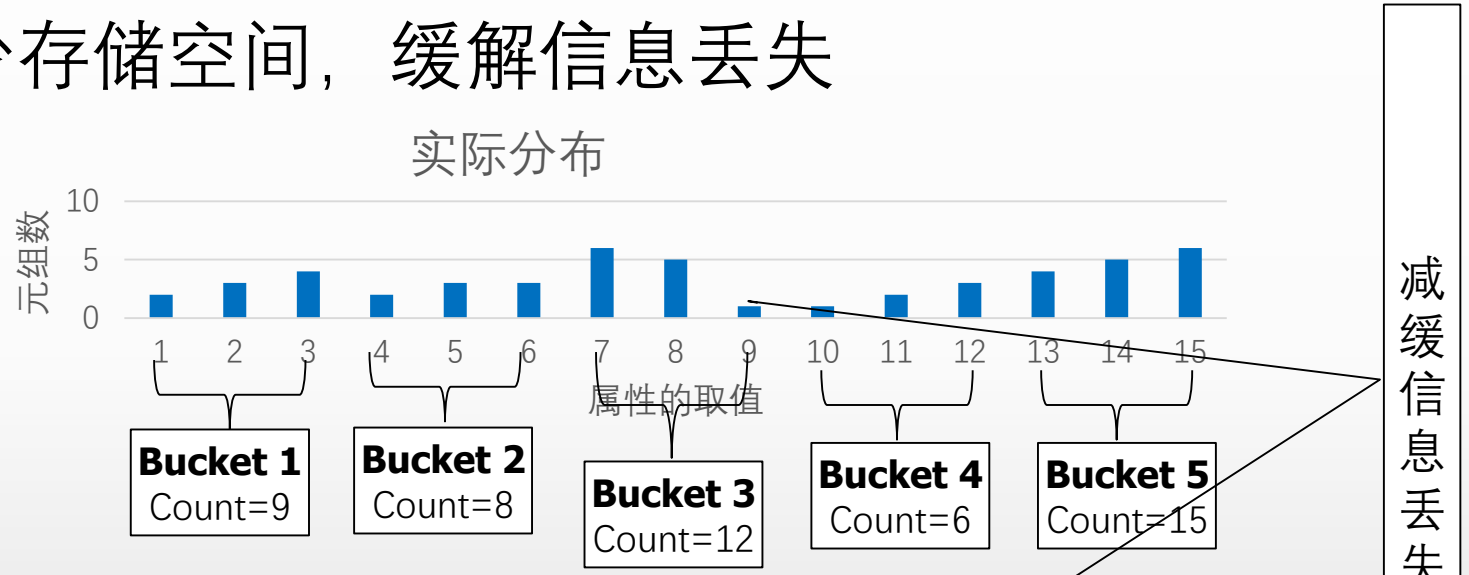
- 等宽直方图
  - 减少存储空间

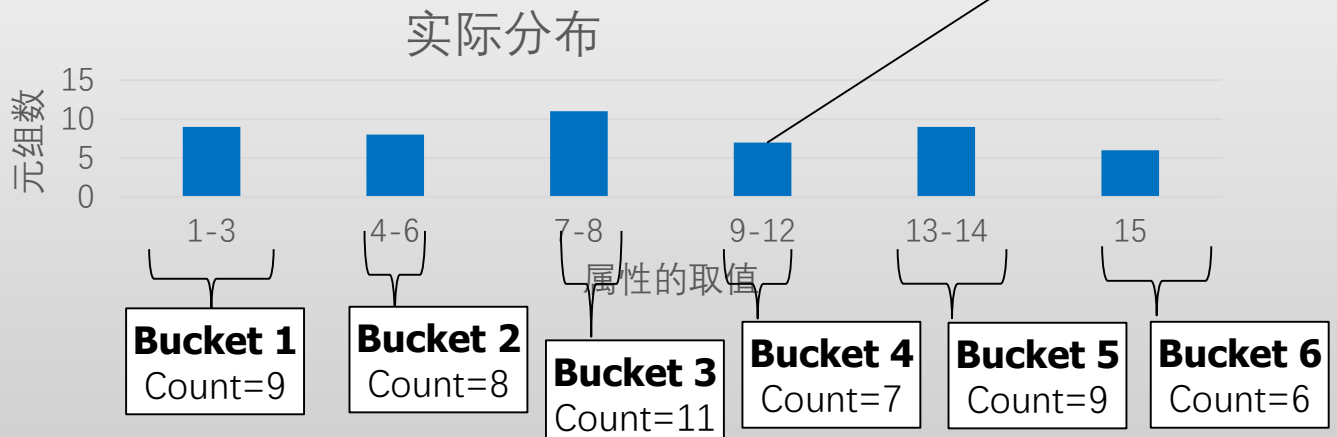




信息丢失太多，产生误差

- 等深直方图
  - 减少存储空间，缓解信息丢失





减缓信息丢失

- Schetch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。

-- Count Min Schetch（1988）

-- HyperLogLog（2007）用于redis

- 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

Sel(age>=50)
=1/3

采样表

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

id	name	age	staus
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

9.3.1 代价估计

9.3.2 计划枚举

- 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

- 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

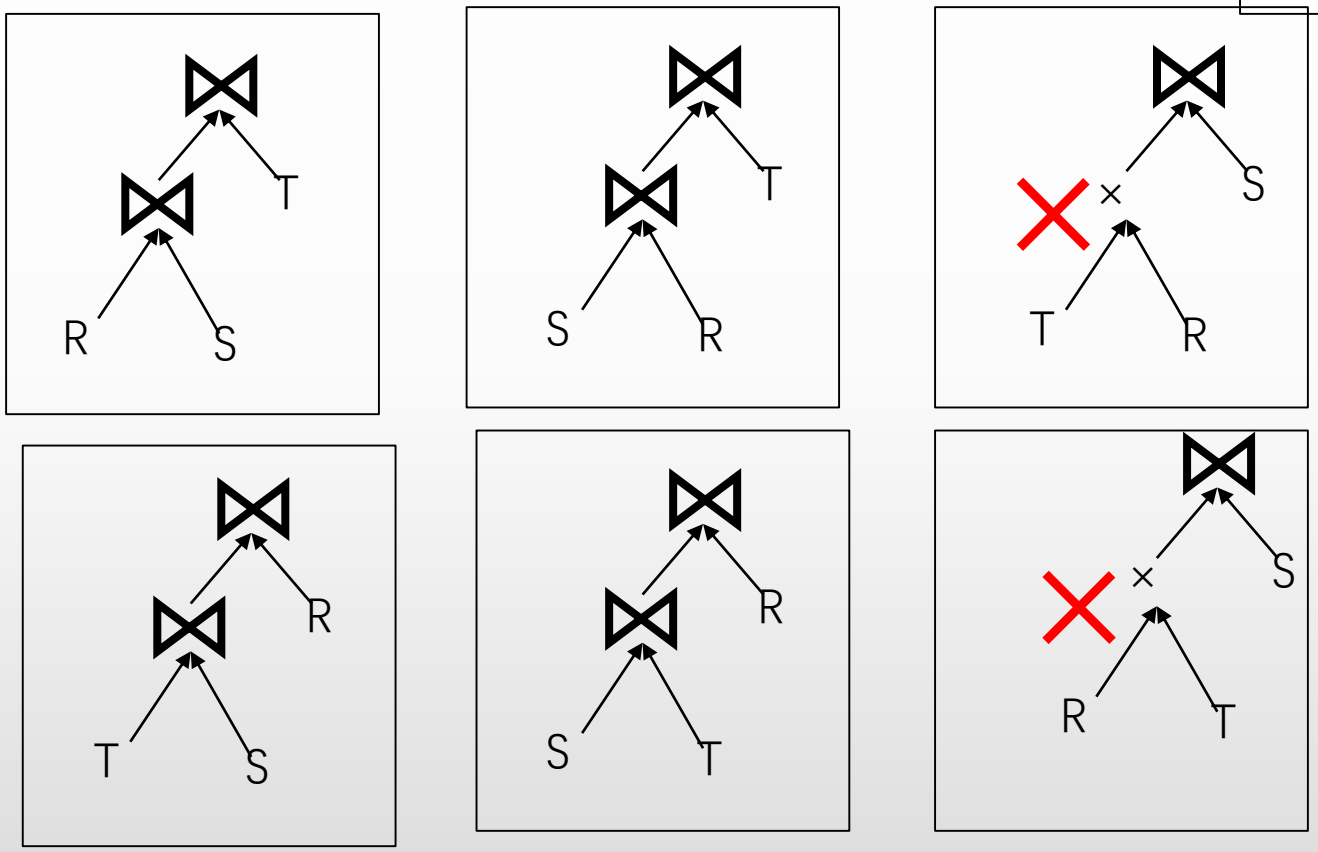
- 多表查询计划
  - 不同连接顺序代价不同
  - 连接的表越多，枚举的查询计划呈阶乘级上升

- 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

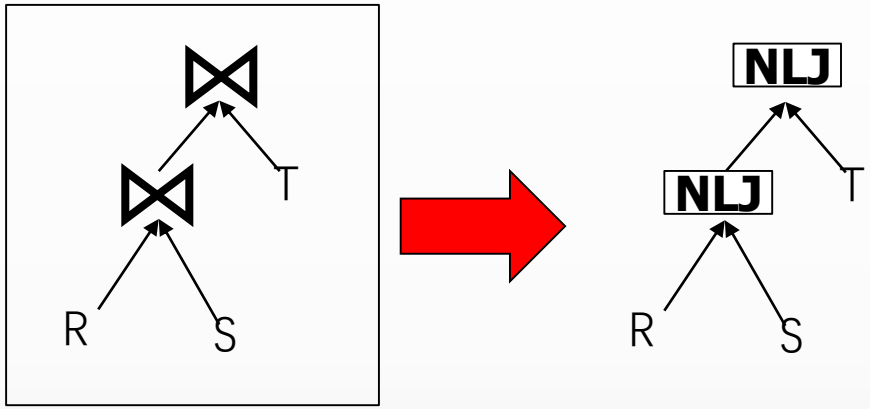
1. 枚举所有可能的连接顺序

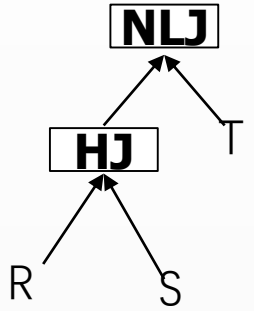
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

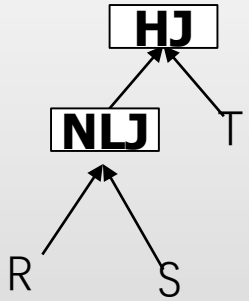


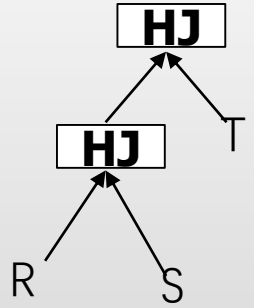
2. 枚举所有可能的连接算法

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



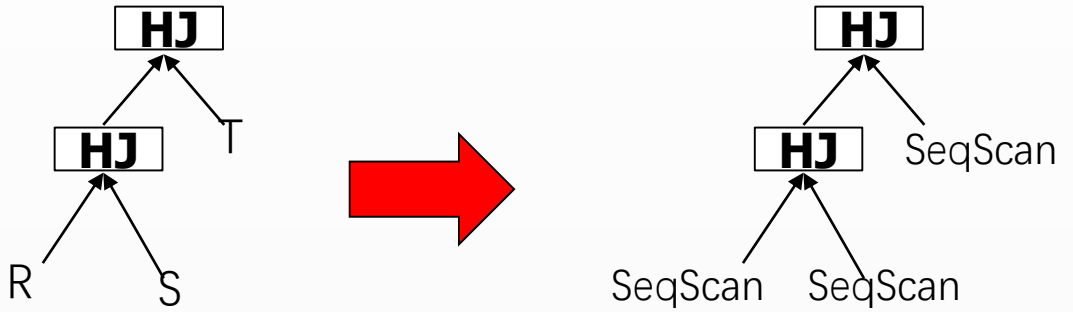


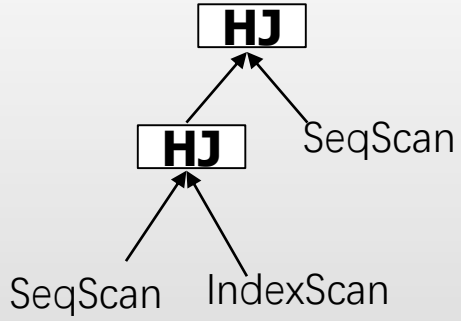




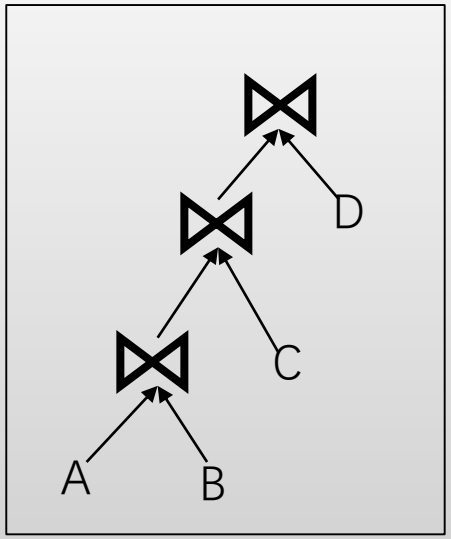
3.枚举所有可能的数据存取方法

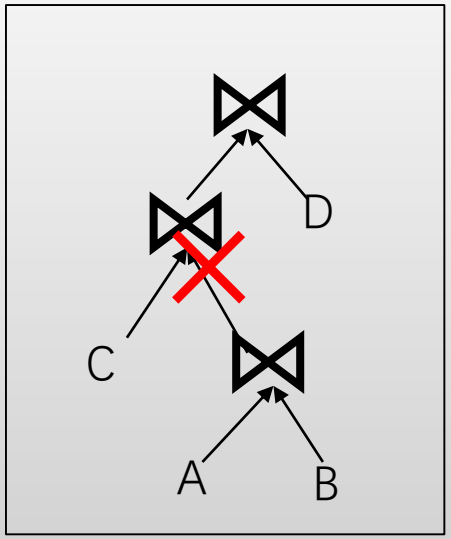
Select * from R,S,T
Where R.a=S.a
And S.b=T.b

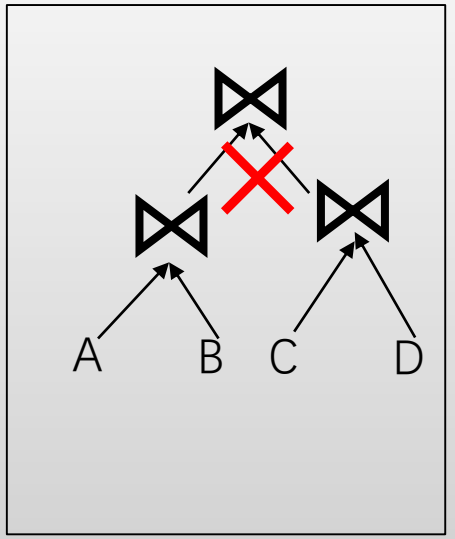




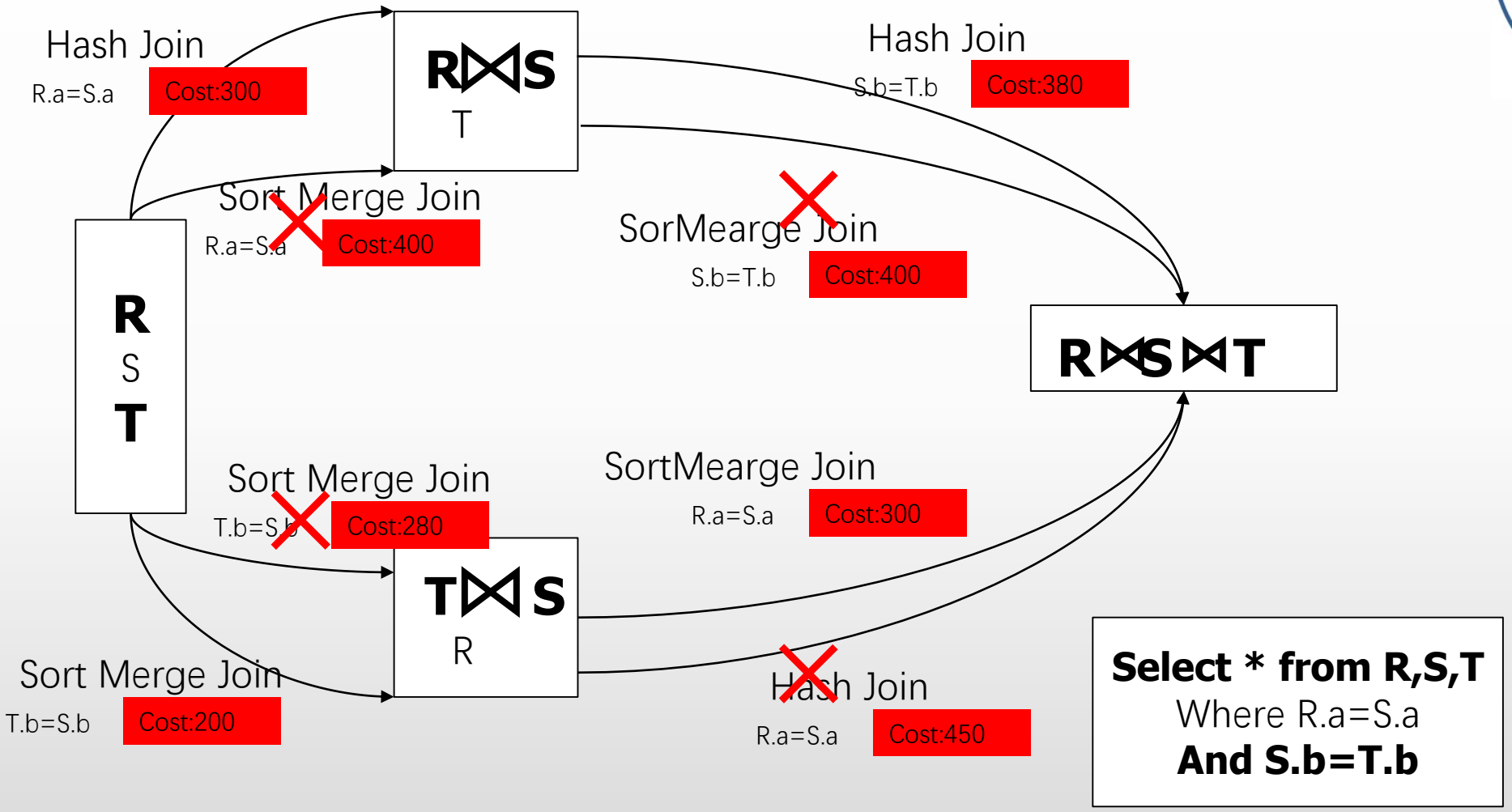
- 多表查询计划(续)
  - System R简化这一问题，只考虑左深树(Left Deep Join Tree)
    - System R认为非左深树不能流水线计算，不符合火山模型

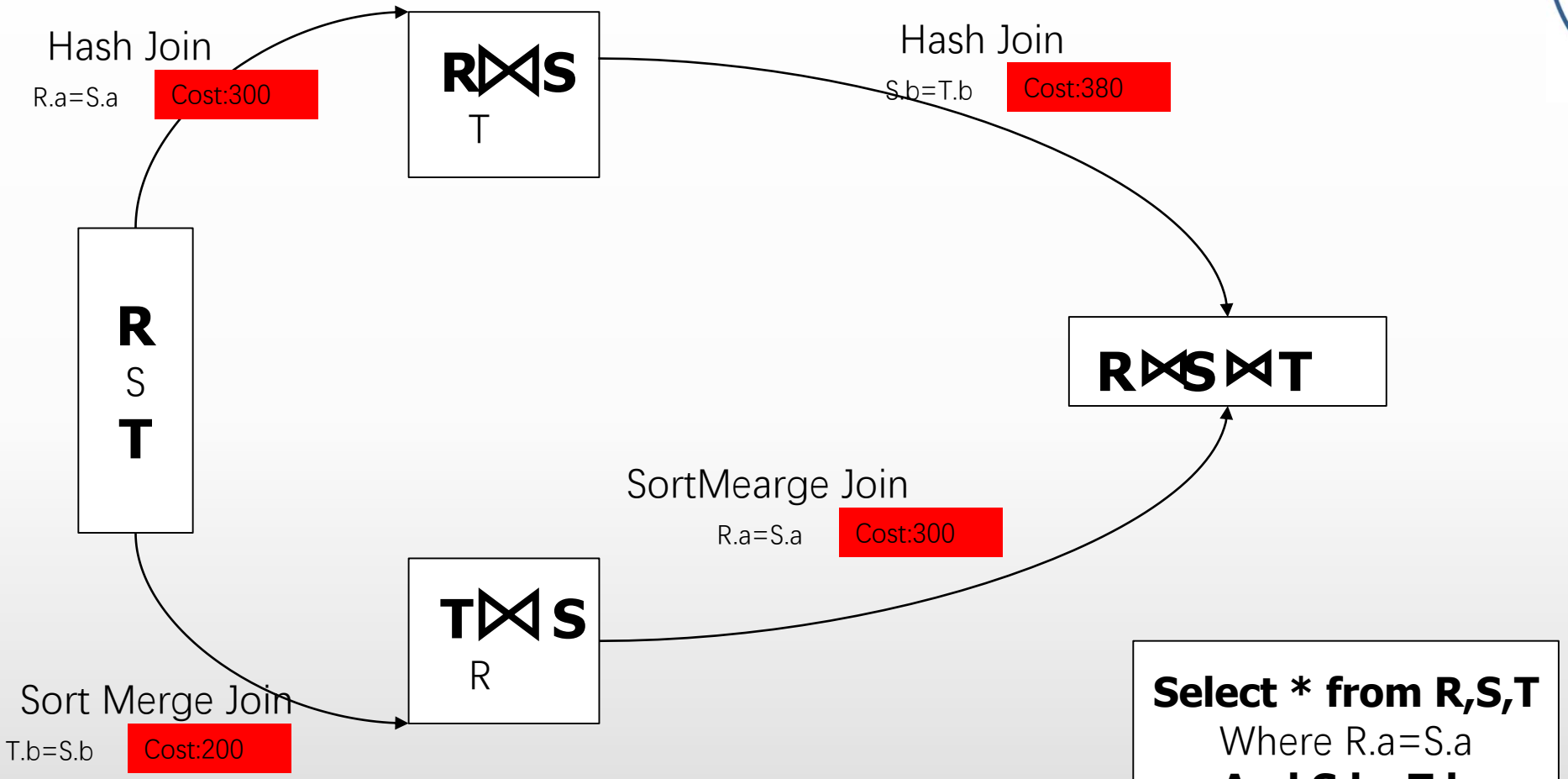


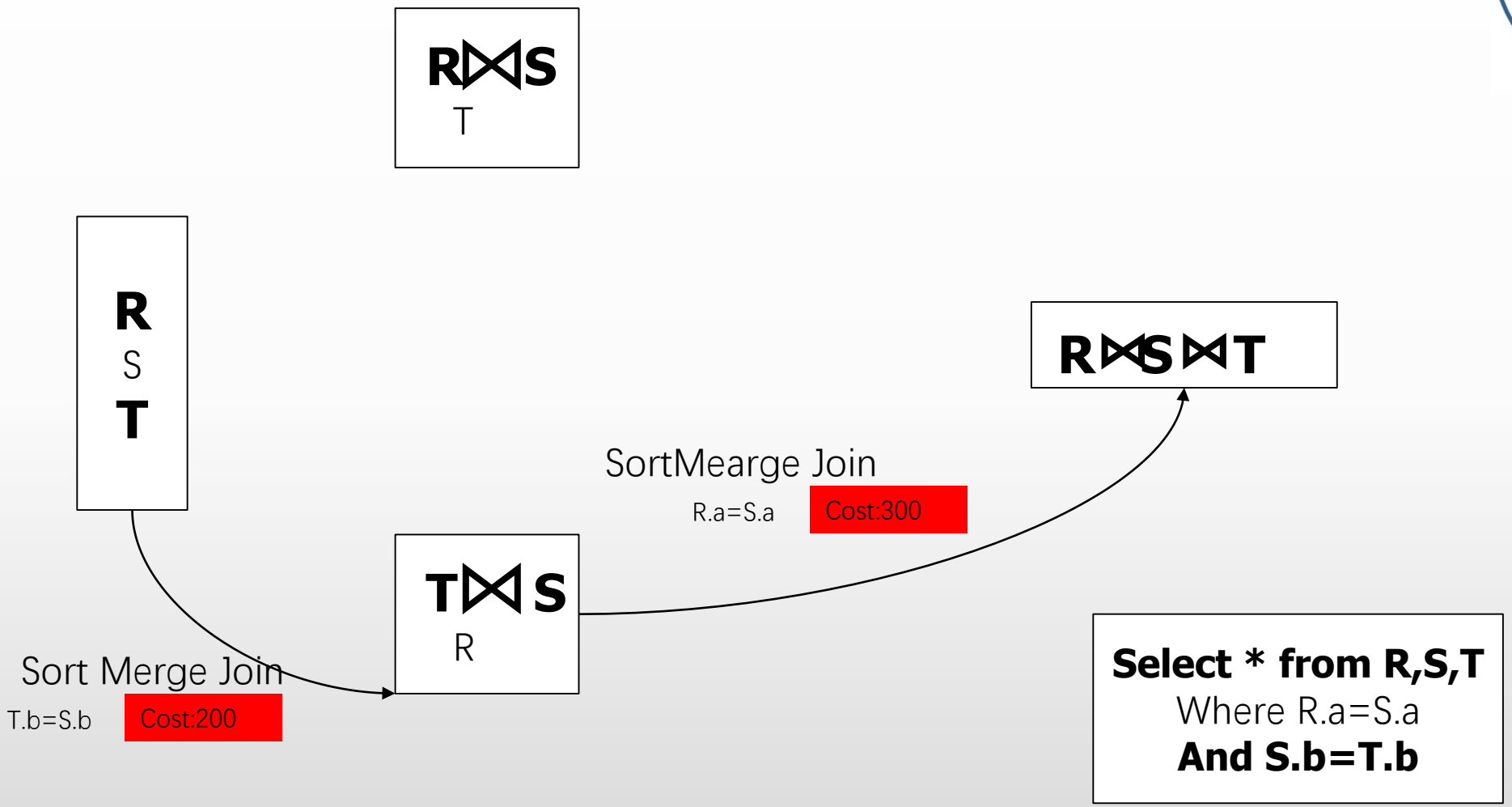




- 基于左深树的物理优化
  - 枚举所有的连接顺序
  - 每个join算子的执行方案
    - Hash join; Sort Merge; Nested Loop …
  - 每个表的访问方法
    - Index Scan; Seq Scan;….
  - 枚举空间巨大
    - 可以采用动态规划算法减少搜索空间


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


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


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

- 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

- 其他优化思路
  - 嵌套子查询的优化
  - 物化视图
  - Top-k优化
  - 连接极小化
  - 多查询优化和共享式扫描
  - 参数化查询优化
  - 。。。

sql语句优化

- 目标

  有利于DBMS选择代价最小的查询执行计划

- 依据

  DBMS优化器支持的优化策略

sql语句优化

Sql语句一般优化策略

- 充分利用索引
- 尽量避免表搜索
- 减少不必要的运算

sql语句优化

1. 搜索参数化

带有=、＜、＞、＞=、＜=等操作符的条件查询就可以直接使用索引。

如:

id = "T0001",salary > 30000,a = 1 and c = 7。

下列则不是搜索参数:

salary = commission,dept != 10,salary *12 > = 30000,age<>21

sql语句优化

在查询中可以提供一些冗余的搜索参数,使优化器有更多的选择余地。

如title和titleauthor两张表是一对多的关系,同样的查询条件我们有以下三种表现方法:

- SELECT title_id, title FROM titles, titleauthor WHERE title.title_id = titleauthor.title_id AND titleauthor.title_id = 'T81002'
- SELECT title_id, title FROM titles, titleauthor WHERE title.title_id = titleauthor.title_id AND title.title_id = 'T81002'
- SELECT title_id, title FROM titles, titleauthor WHERE title.title_id = titleauthor.title_id AND title.title_id = 'T81002' AND titleauthor.title_id = 'T81002'

三种方法一种比一种要好,因为后者为优化器提供了更多的选择机会。

sql语句优化

2.避免使用不兼容的数据类型

SELECT name FROM employee WHERE salary > 60000

3. IS NULL 与 IS NOT NULL

在where子句中使用is null或is not null的语句优化器是不允许使用索引的

4.在海量查询时尽量少用格式转换

where cast(salary as char(2)) = '90'

5.避免不同数据类型间的连接运算

sql语句优化

6.避免where子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

where Left(xsbh,2) = ‘01’

where xsbh = like ’01%’

SQL优化实例

——oracle

例1：RLWT(STCD，TM，LV)——水位表

查询当前时间前一个小时的时刻的水位记录

```
SELECT   *   FROM    RLWT
WHERE    TM+1 = SYSDATE;

SELECT   *   FROM    RLWT
WHERE    TM= SYSDATE   − 1;


SELECT    *   FROM    RLWT
WHERE    TM = TO_DATE(‘2006-04-26 7:00:00’,’YY-MM-DD
HH24:MI:SS’)

SELECT   STCD，TM，LV    FROM    RLWT
WHERE    TM = TO_DATE(‘2006-04-26 7:00:00’,’YY-MM-DD
HH24:MI:SS’)
```

例2：查选修了‘01’号课程的学生姓名。

```
SELECT   SNAME
FROM   STUDENT   WHERE  ‘01’  IN (SELECT CNO
    FROM SC
    WHERE SNO=STUDENT.SNO);

SELECT SNAME
FROM   STUDENT,  SC
WHERE   SC. SNO = STUDENT. SNO
AND   SC.CNO = ’01’
```

例3：SELECT CNO，AVG(GRADE)
FROM SC
GROUP BY CNO
HAVING CNO != ‘001'
AND CNO != ‘002'

SELECT CNO，AVG(GRADE)
FROM SC
WHERE CNO != ‘001'
AND CNO != ‘002'
GROUP BY CNO

例4：SELECT TAB_NAME
FROM TABLES
WHERE TAB_NAME = (SELECT TAB_NAME
FROM TAB_COLUMNS
WHERE VERSION = 604)
AND DB_VER= (SELECT DB_VER
FROM TAB_COLUMNS
WHERE VERSION = 604)

SELECT TAB_NAME
FROM TABLES
WHERE (TAB_NAME, DB_VER)
= (SELECT TAB_NAME, DB_VER
FROM TAB_COLUMNS
WHERE VERSION = 604)

例5：UPDATE EMP
SET EMP_CAT = (SELECT MAX(CATEGORY) FROM
EMP_CATEGORIES),
SAL_RANGE = (SELECT MAX(SAL_RANGE) FROM
EMP_CATEGORIES)
WHERE EMP_DEPT = 0020;

```
UPDATE    EMP
SET (EMP_CAT, SAL_RANGE)
= (SELECT   MAX(CATEGORY) ,     MAX(SAL_RANGE)
    FROM      EMP_CATEGORIES    )
WHERE EMP_DEPT = 0020;
```

例6：SELECT * FROM SC
WHERE CNO >3

SELECT * FROM SC
WHERE CNO>=4

例7: SELECT ACCOUNT_NAME
FROM TRANSACTION
WHERE AMOUNT !=0; 不使用索引

SELECT ACCOUNT_NAME
FROM TRANSACTION
WHERE AMOUNT >0; 使用索引:

索引只能告诉你什么存在于表中, 而不能告诉你什么不存在于表中。