第九章

查询处理和查询优化

教学内容：

查询处理的基本步骤

查询优化的概念

优化的基本方法和技术

要求掌握：

1、DBMS处理查询的步骤

2、查询优化的原理和方法

3、如何将语法树转化为优化的语法树

教学重点及难点：

查询优化、语法树转化

第九章 查询优化

9.1 关系数据库系统的查询处理

9.2 关系数据库系统的查询优化

9.3 代数优化

9.4 物理优化

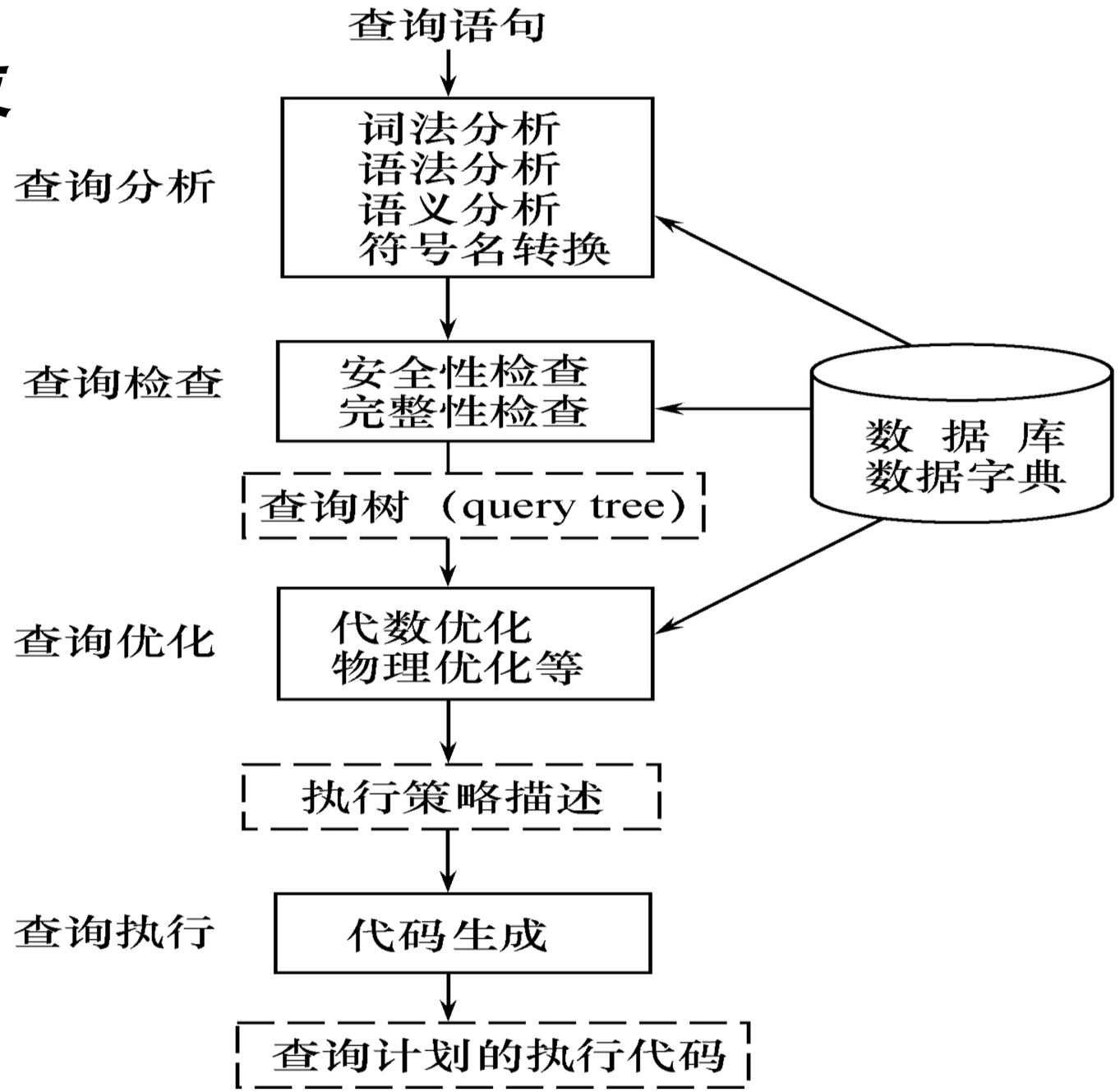
9.5 SQL调优

9.1 关系数据库系统的查询处理

- 9.1.1 查询处理步骤
- 9.1.2 实现查询操作的算法示例

9.1.1 查询处理步骤

- **RDBMS查询处理阶段**
  1. **查询分析**
  2. **查询检查**
  3. **查询优化**
  4. **查询执行**



1. 查询分析

- 对查询语句进行扫描、词法分析和语法分析
- 从查询语句中识别出语言符号，如SQL关键字、关系名和属性名等
- 进行语法检查和语法分析

2. 查询检查

- 根据数据字典对合法的查询语句进行语义检查
- 根据数据字典中的用户权限和完整性约束定义对用户的存取权限进行检查
- 检查通过后把SQL查询语句转换成等价的关系代数表达式
- RDBMS一般都用查询树(语法分析树)来表示扩展的关系代数表达式

3. 查询优化

- 查询优化：选择一个高效执行的查询处理策略
- 查询优化分类：
  - 代数优化：指关系代数表达式的优化
  - 物理优化：指存取路径和底层操作算法的选择

- 查询优化方法选择的依据：
  - 基于规则 (rule based)
  - 基于代价 (cost based)
  - 基于语义 (semantic based)

4. 查询执行

- 依据优化器得到的执行策略生成查询计划
- 代码生成器生成执行查询计划的代码

9.1 关系数据库系统的查询处理

- 9.1.1 查询处理步骤
- 9.1.2 实现查询操作的算法示例

9.1.2 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 (table scan)

- 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
- 运行只需要很少的内存，适合小表，不适合大表

2. 索引扫描方法 (index scan)

- 选择条件中的属性上有索引
- 通过索引先找到满足条件的元组指针，再通过元组指针直接在查询的基本表中找到元组

索引扫描算法举例

［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

索引扫描算法举例（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：分别用上面两种方法分别找到Sdept＝'CS'的一组元组指针和Sage>20的另一组元组指针
  - 求这两组指针的交集
  - 到student表中检索
  - 得到计算机系年龄大于20的学生

索引扫描算法举例（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(s)，检索内层循环(SC)中的每一个元组(sc)
- 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

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按连接属性排好序的情况
- 排序-合并连接方法的步骤：

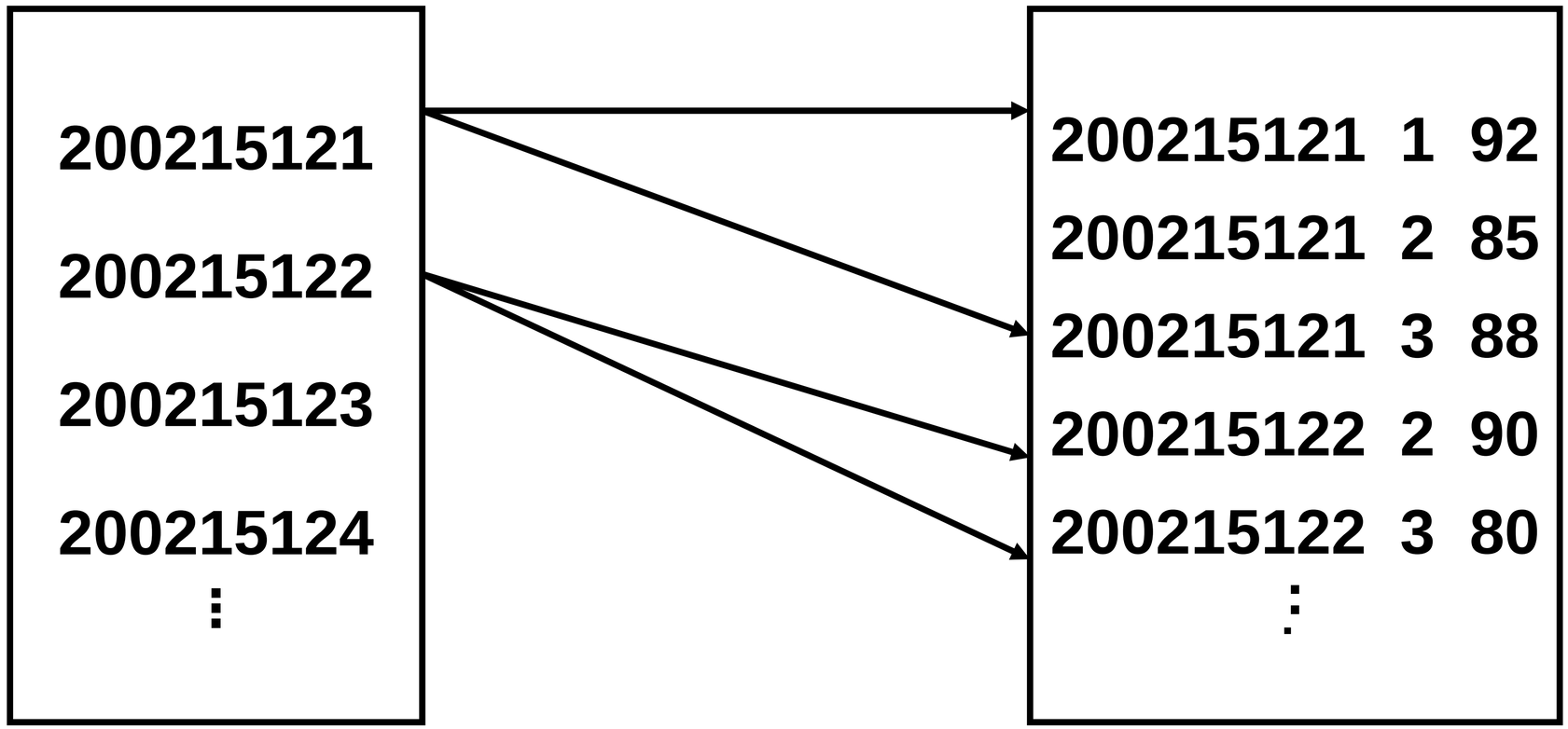
① 如果连接的表没有排好序，先对Student表和SC表按连接属性Sno排序

② 取Student表中第一个Sno，依次扫描SC表中具有相同Sno的元组

③ 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

④ 重复上述步骤直到Student 表扫描完

连接操作的实现（续）


200215121
200215122
200215123
200215124
⋮
200215121 1 92
200215121 2 85
200215121 3 88
200215122 2 90
200215122 3 80
⋮

连接操作的实现（续）

- Student表和SC表都只要扫描一遍
- 如果两个表原来无序，执行时间要加上对两个表的排序时间
- 对于两个大表，先排序后使用排序-合并连接算法执行连接，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（如果原来没有该索引）

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

连接操作的实现（续）

4. Hash Join方法

- 把连接属性作为hash码，用同一个hash函数把Student和SC中的元组散列到同一个hash文件中
- 步骤：

  划分阶段(partitioning phase)：
  - 对包含较少元组的表（比如Student）进行一遍处理
  - 把它的元组按hash函数分散到hash表的桶中

  试探阶段(probing phase)，也称为连接阶段(join phase)：
  - 对另一个表(SC)进行一遍处理
  - 把SC的元组散列到适当的hash桶中
  - 把元组与桶中所有来自Student并与之相匹配的元组连接起来

连接操作的实现（续）

- 上面hash join算法前提：假设两个表中较小的表在第一阶段后可以完全放入内存的hash桶中

- 以上的算法思想可以推广到更加一般的多个表的连接算法上

9.2 关系数据库系统的查询优化

- 查询优化在关系数据库系统中有着非常重要的地位
- 关系查询优化是影响RDBMS性能的关键因素

9.2.1 查询优化概述

- 用户不必考虑如何更好地表达查询以获得较高的效率
- DBMS系统提供的查询优化可以比用户程序的“优化”做得更好，这是因为：

(1) 优化器可以从数据字典中获取许多统计信息，做出正确的估算，而用户程序难以获得这些信息；

(2) 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。不需要重写程序；

9.2.1 查询优化概述

(3) 优化器可以考虑数百种不同的执行计划，程序员一般只能考虑有限的几种可能性；

(4) 优化器中包括了很多复杂的优化技术，这些优化技术往往只有最好的程序员才能掌握。系统的自动优化相当于使得所有人都拥有这些优化技术。

查询代价

- 在集中式数据库中，查询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）以及查询的内存开销

总代价=I/O代价+CPU代价+内存代价

- 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）

9.2.2 一个实例

[例3] 求选修了2号课程的学生姓名。

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

- 假定学生-课程数据库中有1000个学生记录，10000个选课记录
- 其中选修2号课程的选课记录为50个

9.2.2 一个实例

- 系统可以用多种等价的关系代数表达式来完成这一查询
  - $Q1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='2'}(Student \times SC))$
  - $Q2 = \pi_{Sname}(\sigma_{SC.Cno='2'}(Student \bowtie SC))$
  - $Q3 = \pi_{Sname}(Student \bowtie \sigma_{SC.Cno='2'}(SC))$

一个实例：第一种情况

$Q1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='2'}(Student \times SC))$

1. 计算广义笛卡尔积

- 把Student和SC的每个元组连接起来的做法：
  - 在内存中尽可能多地装入某个表(如Student表)的若干块，留出一块存放另一个表(如SC表)的元组
  - 把SC中的每个元组和Student中每个元组连接，连接后的元组装满一块后就写到中间文件上
  - 再从SC中读入一块和内存中的Student元组连接，直到SC表处理完
  - 再读入若干块Student元组，读入一块SC元组
  - 重复上述处理过程，直到把Student表处理完

一个实例：第一种情况

- 设一个块能装10个Student元组或100个SC元组，在内存中存放5块Student元组和1块SC元组，则读取总块数为

$$\frac{1000}{10}+\frac{1000}{10\times5}\times\frac{10000}{100}=100+20\times100=2100\text{块}$$

n(Stu) n(Stu) M(SC)

block(Stu) blockd(Stu) blockd(SC)
xn(blockstu) xn(blockSC)
100 20 100

- 其中，读Student表100块。读SC表20遍，每遍100块。若每秒读写20块，则总计要花105s。
- 连接后的元组数为$10^3\times10^4=10^7$。设每块能装10个元组，则写出这些块要用$10^6/20=5\times10^4$s。

一个实例：第一种情况

$$Q1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='2'}(Student \times SC))$$

2. 作选择操作

- 依次读入连接后的元组，按照选择条件选取满足要求的记录
- 假定内存处理时间忽略。读取中间文件花费的时间(同写中间文件一样)需5×10^4s
- 满足条件的元组假设仅50个，均可放在内存

一个实例：第一种情况

$Q1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='2'}(Student \times SC))$

3. 作投影操作

- 把第2步的结果在Sname上作投影输出，得到最终结果
- 第一种情况下执行查询的总时间 $\approx 105+2\times5\times10^4\approx10^5$s
- 所有内存处理时间均忽略不计

一个实例：第二种情况

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

1. 计算自然连接
   - 执行自然连接，读取Student和SC表的策略不变，总的读取块数仍为2100块花费105s
   - 自然连接的结果比第一种情况大大减少，为10^4个
   - 写出这些元组时间为10^4/10/20=50s（每块放10个元组，每秒读写20块），为第一种情况的千分之一
2. 读取中间文件块，执行选择运算，花费时间也为50s
3. 把第2步结果投影输出

第二种情况总的执行时间≈105+50+50≈205s

一个实例：第三种情况

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

1. 先对SC表作选择运算，只需读一遍SC表，存取100块花费时间为5s，因为满足条件的元组仅50个，不必使用中间文件
2. 读取Student表，把读入的Student元组和内存中的SC元组作连接，也只需读一遍Student表共100块，花费时间为5s
3. 把连接结果投影输出

第三种情况总的执行时间≈5+5≈10s

一个实例：第三种情况

- 假如SC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Q1变换为Q2、 Q3
  - 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
- 在Q3中
  - SC表的选择操作算法有全表扫描和索引扫描两种方法，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Student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

- 9.3.1 关系代数表达式等价变换规则
- 9.3.2 查询树的启发式优化

9.3.1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$E_1 \equiv E_2$

关系代数表达式等价变换规则（续）

1. 连接、笛卡尔积的交换律

设E_1、E_2是关系代数表达式，F是连接运算条件，则有

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

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

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

2. 连接、笛卡尔积的结合律

设E_1、E_2、E_3是关系代数表达式，F_1、F_2是连接运算条件，则有

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

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

$$(E_1 \underset{F_1}{\bowtie} E_2) \underset{F_2}{\bowtie} E_3 \equiv E_1 \underset{F_1}{\bowtie} (E_2 \underset{F_2}{\bowtie} E_3)$$

关系代数表达式等价变换规则（续）

3. 投影的串接定律

$$\prod_{A_1,A_2,\ldots,A_n}(\prod_{B_1,B_2,\ldots,B_m}(E)) \equiv \prod_{A_1,A_2,\ldots,A_n}(E)$$

这里，E是关系代数表达式，A_i和B_j是属性名，且$\{A_1,A_2,\ldots,A_n\}$构成$\{B_1,B_2,\ldots,B_m\}$的子集。

4. 选择的串接定律

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

这里，E是关系代数表达式，F_1、F_2是连接运算条件。选择的串接律说明**选择条件可以合并，这样一次就可以检查全部条件**。

关系代数表达式等价变换规则（续）

5. 选择与投影操作的交换律

$$\sigma_F(\prod_{A_1,A_2,\ldots,A_n}(E)) \equiv \prod_{A_1,A_2,\ldots,A_n}(\sigma_F(E))$$

这里，选择条件F只涉及属性$A_1,\ldots,A_n$。

若F中有不属于$A_1,\ldots,A_n$的属性$B_1,\ldots,B_m$，则有更一般的规则：

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

关系代数表达式等价变换规则（续）

6. 选择与笛卡尔积的交换律

如果F中涉及的属性都是E_1中的属性，则

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

如果$F = F_1 \wedge F_2$，并且F_1只涉及E_1中的属性，F_2只涉及E_2中的属性，则由上面的等价变换规则1、4、6可推出

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2)$$

若F_1只涉及E_1中的属性，F_2涉及E_1和E_2两者的属性，则仍有

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \times E_2)$$

它使部分选择在笛卡尔积前先做。

关系代数表达式等价变换规则（续）

7. 选择与并的分配律

设$E = E_1 \cup E_2$，E_1、E_2有相同的属性名，则

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

8. 选择与差运算的分配律

若E_1、E_2有相同的属性名，则

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

9. 选择对自然连接的分配律

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

F只涉及E_1与E_2的公共属性。

关系代数表达式等价变换规则（续）

10. 投影与笛卡尔积的分配律

设E_1与E_2是两个关系表达式，$A_1,\ldots,A_n$是E_1的属性，$B_1,\ldots,B_m$是E_2的属性，则

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

11. 投影与并的分配率

若E_1、E_2有相同的属性名，则

$$\prod_{A_1,A_2,\ldots,A_n}(E_1 \cup E_2) \equiv \prod_{A_1,A_2,\ldots,A_n}(E_1) \cup \prod_{A_1,A_2,\ldots,A_n}(E_2)$$

9.3 代数优化

- 9.3.1 关系代数表达式等价变换规则
- 9.3.2 查询树的启发式优化

9.3.2 查询树的启发式优化

- 如何从SQL语句生成原始的查询树？

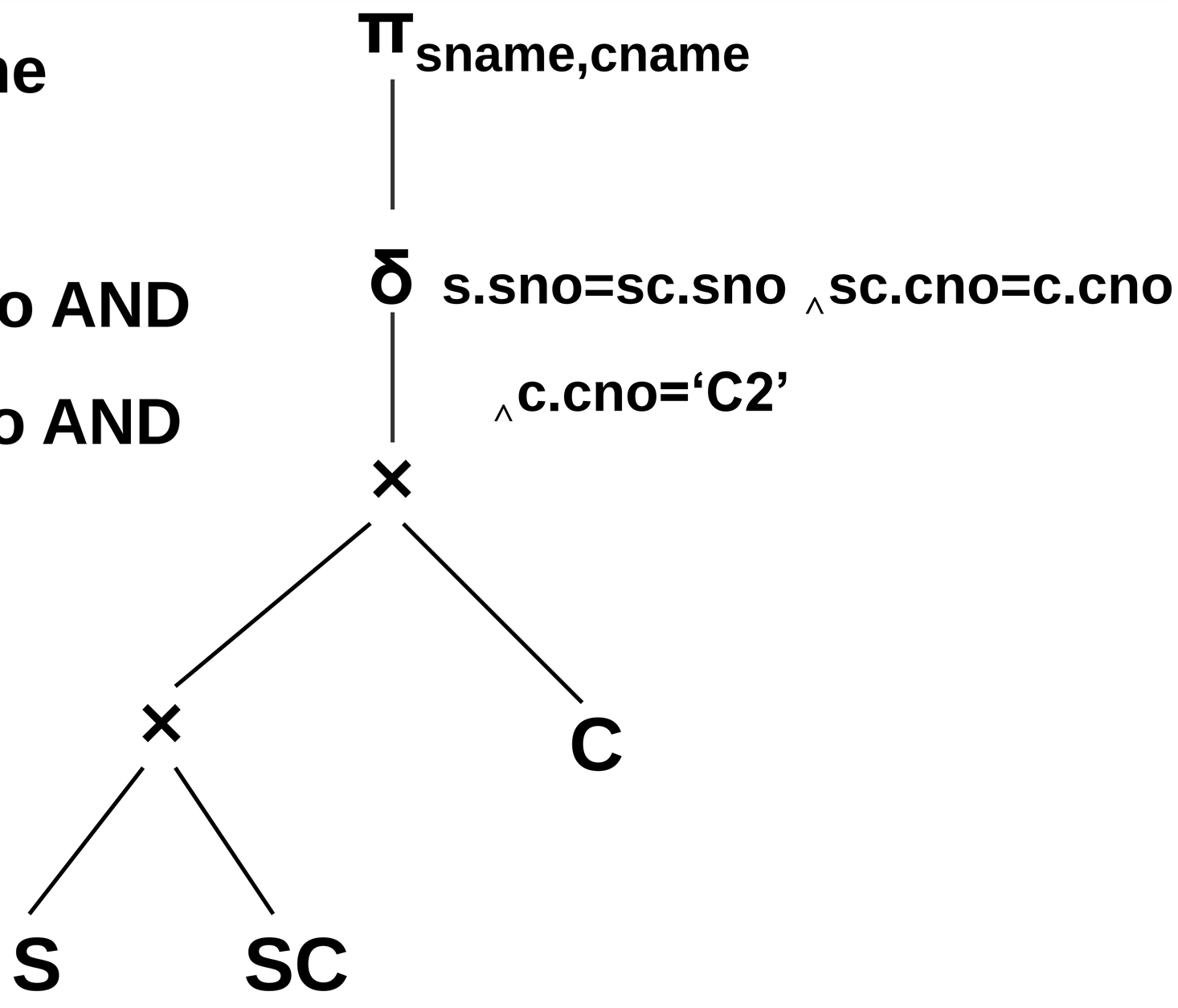
  (1) 以**SELECT**语句对应**投影**操作

  (2) 以**FROM**语句对应**笛卡尔积**操作

  (3) 以**WHERE**语句对应**选择**操作

例子

```
SELECT sname,cname
FROM   s,sc,c
WHERE  s.sno=sc.sno AND
       sc.cno=c.cno AND
       c.cno='C2'
```



典型的启发式规则

1. **选择运算应尽可能先做**，在优化策略中这是最重要、最基本的一条

2. 把投影运算和选择运算同时进行

- 如有若干投影和选择运算，并且它们都对同一个关系操作，则可以在扫描此关系的同时完成所有的这些运算以避免重复扫描关系

典型的启发式规则（续）

3. 把投影同其前或其后的双目运算结合起来
4. 把某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算
5. 找出公共子表达式

- 如果这种重复出现的子表达式的结果不是很大的关系并且从外存中读入这个关系比计算该子表达式的时间少得多，则先计算一次公共子表达式并把结果写入中间文件是合算的

查询树的启发式优化（续）

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$\sigma_{F_1 \wedge F_2 \wedge \ldots \wedge F_n}(E)$的表达式变换为$\sigma_{F_1}(\sigma_{F_2}(\ldots(\sigma_{F_n}(E))\ \ldots))$。

（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（续）

（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：

- 规则3使一些投影消失

$$\prod_{A_1,A_2,\ldots,A_n}(\prod_{B_1,B_2,\ldots,B_m}(E)) \equiv \prod_{A_1,A_2,\ldots,A_n}(E)$$

- 规则5把一个投影分裂成两个，其中一个有可能被移向叶端

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

查询树的启发式优化（续）

（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（续）

（5）把上述得到的语法树的内结点分组。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σ，$\prod$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，而且后面不是与它组成等值连接的选择时，则不能把选择与这个双目运算组成同一组。把这些单目运算单独分为一组。

检索至少学习LIU老师所授一门课的女同学的学号和姓名，用SQL表达为：

```
SELECT s#,sn
FROM s,sc,c
WHERE t='LIU' and sex='女' and
s.s#=sc.s# and sc.c#=c.c#
```

优化的过程：

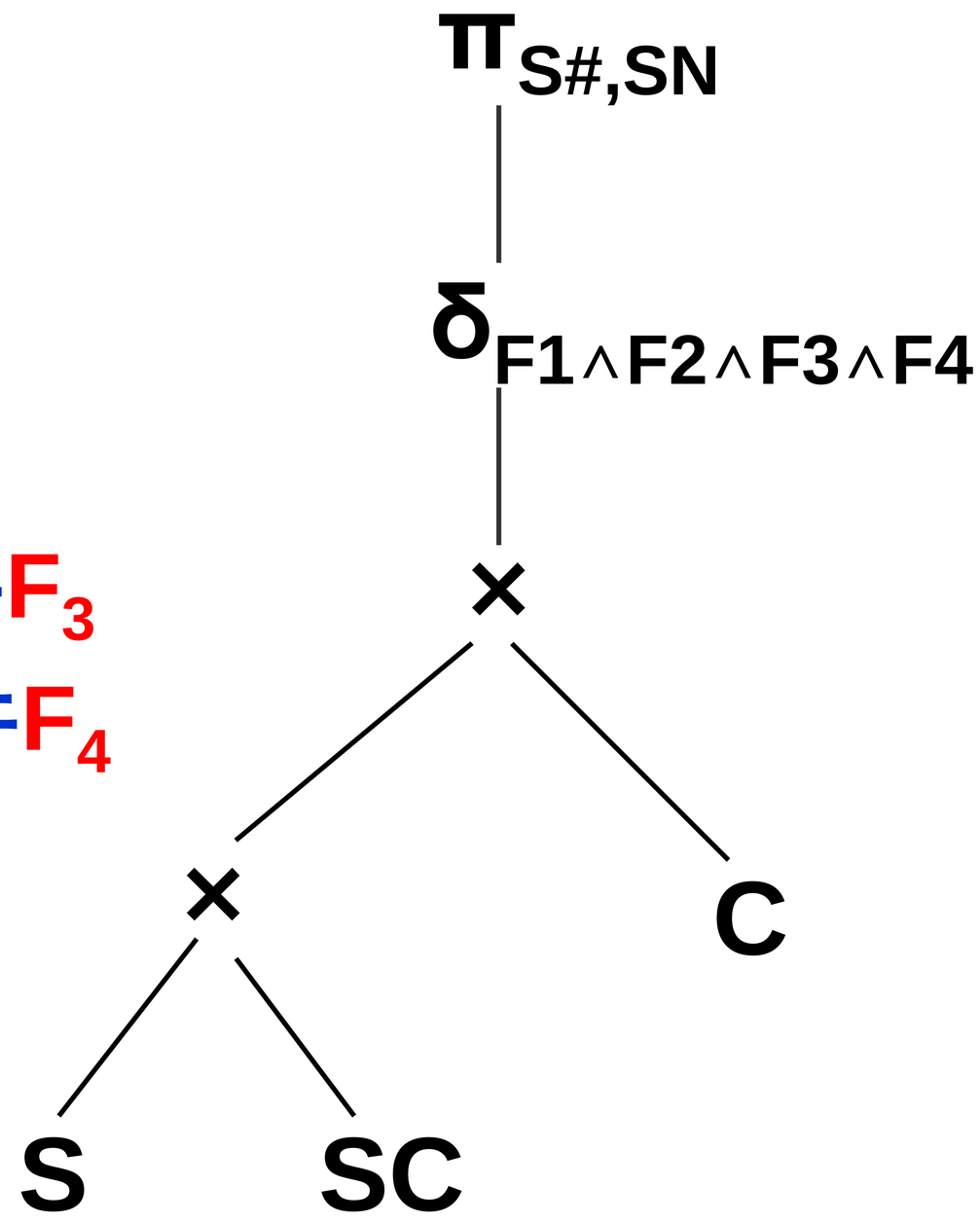
(1) 画出原始的查询树

令T=‘LIU’为条件F_1

SEX=‘女’为条件F_2

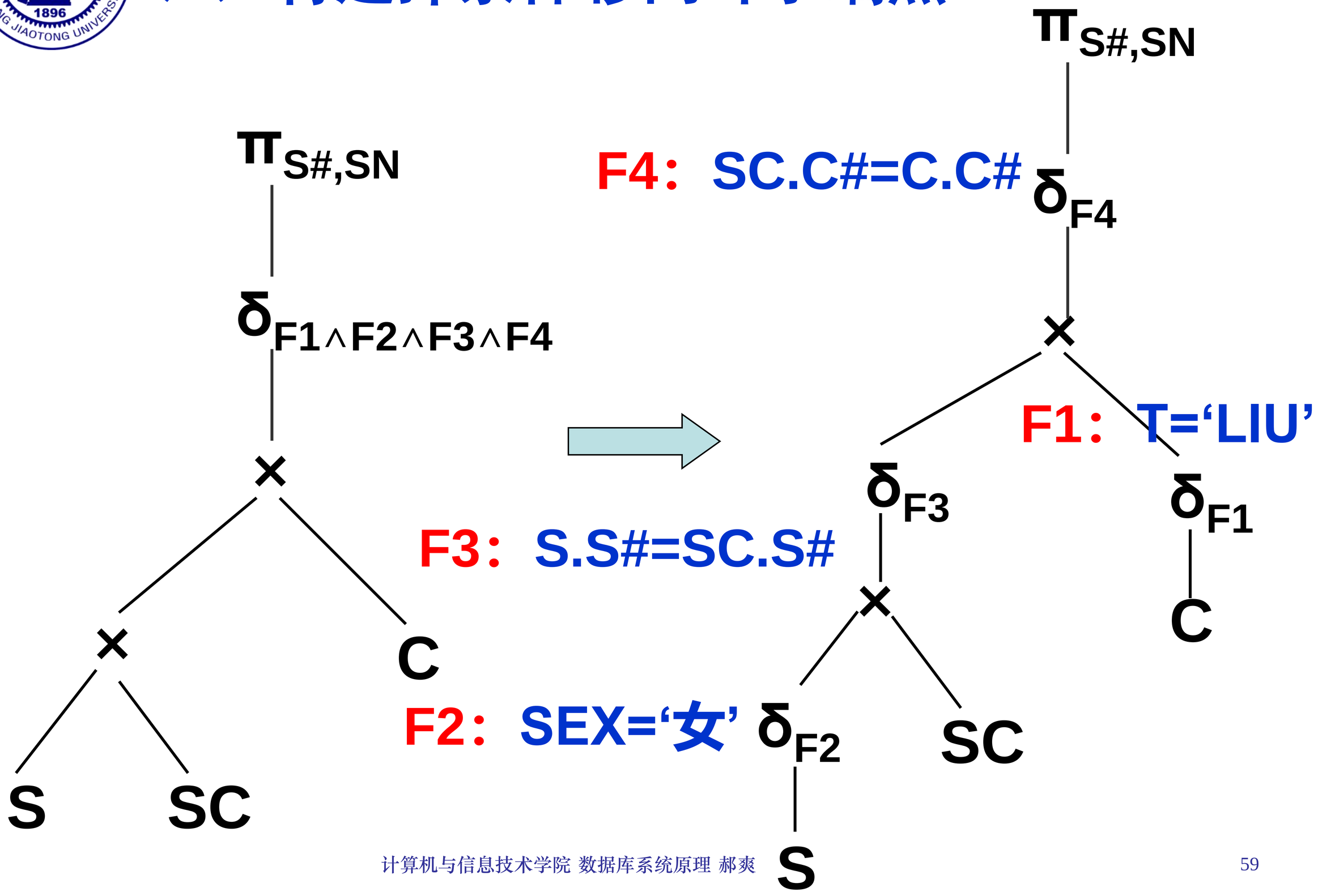
S.S#=SC.S#为条件F_3

SC.C#=C.C#为条件F_4

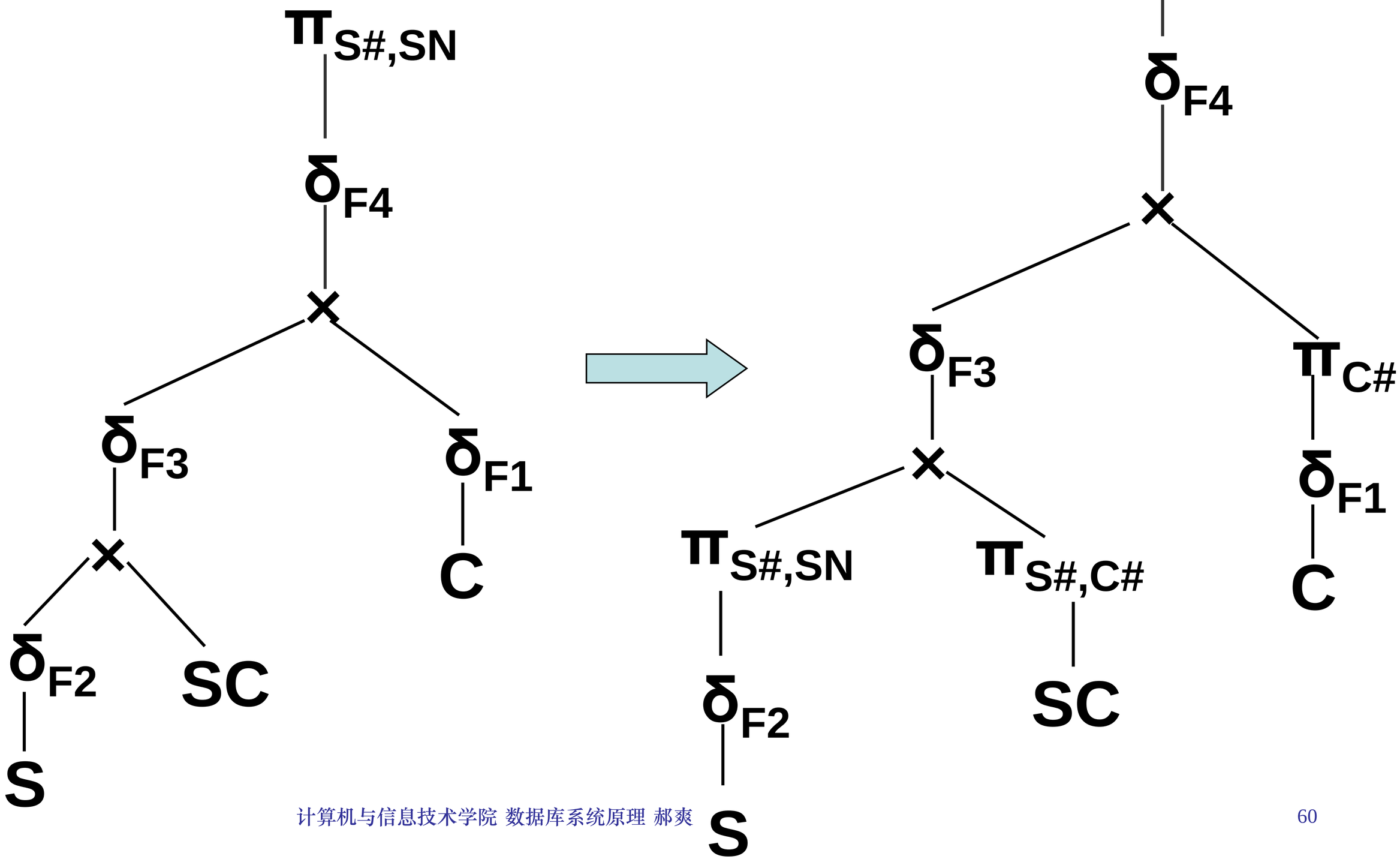


$$\pi_{S\#,SN}(\delta_{F1\wedge F2\wedge F3\wedge F4}(S\times SC\times C))$$

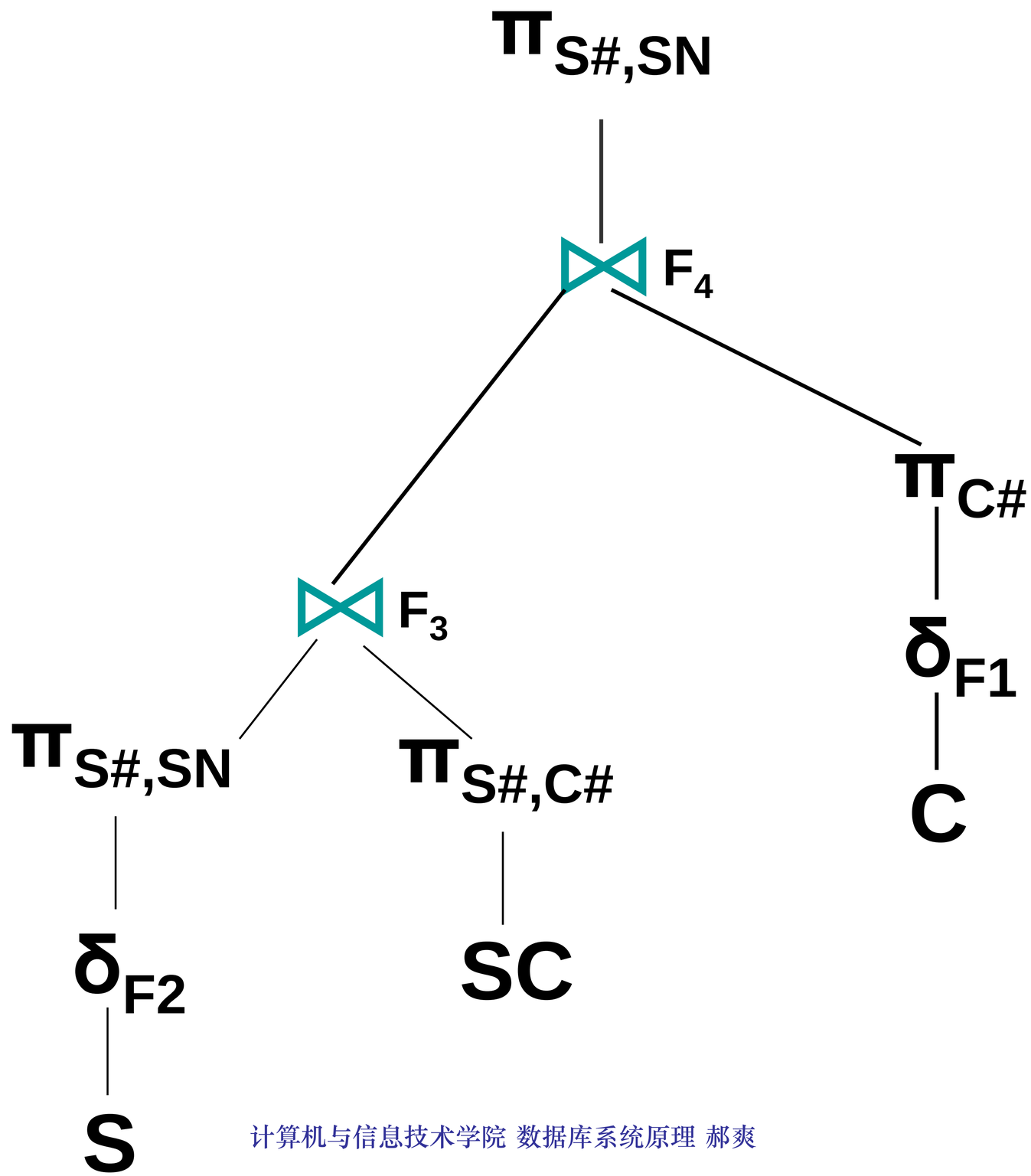
（2）将选择条件移向叶子端点



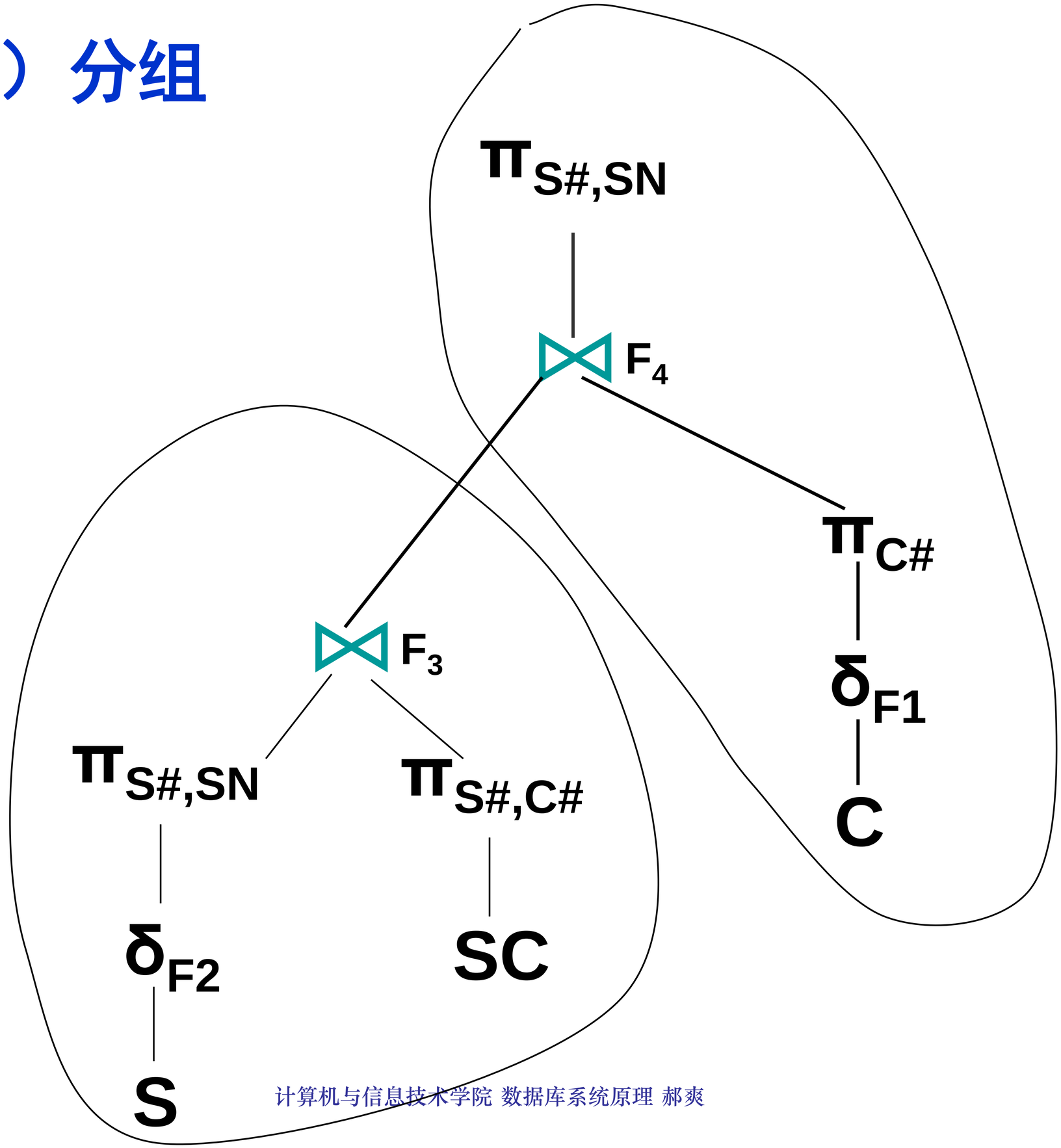
（3）在连接前先做投影操作将无关的属性去掉



（4）用连接操作代替笛卡尔积



（5）分组



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

9.4 物理优化

- 9.4.1 基于启发式规则的存取路径选择优化
- 9.4.2 基于代价的优化

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

一、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. 对于大关系，若选择条件是主码＝值的查询
   - 查询结果最多是一个元组，可以选择主码索引

选择操作的启发式规则(续)

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

   要估算查询结果的元组数目

   - 如果比例较小(<10%)可以使用索引扫描方法
   - 否则还是使用全表顺序扫描

选择操作的启发式规则(续)

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

   要估算查询结果的元组数目

   - 如果比例较小(<10%)可以使用索引扫描方法
   - 否则还是使用全表顺序扫描

选择操作的启发式规则(续)

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，优先采用组合索引扫描方法

```
CREATE INDEX idx
ON table(col2,col3,col5)
```

   - 如果某些属性上有一般的索引，则用［例1-C4］中介绍的索引扫描方法，否则使用全表顺序扫描。
6. 对于用OR连接的析取选择条件，一般使用全表顺序扫描

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

连接操作的启发式规则(续)

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较少的表，作为外表(外循环的表) 。

理由：

- 设连接表R与S分别占用的块数为Br与Bs
- 连接操作使用的内存缓冲区块数为K
- 分配K-1块给外表
- 如果R为外表，则嵌套循环法存取的块数为Br+BsBr/K-1
- 显然应该选块数小的表作为外表

9.4.2 基于代价的优化

一、 统计信息

二、 代价估算示例

一、统计信息

- 基于代价的优化方法要**计算各种操作算法的执行代价**，与数据库的状态密切相关
- **数据字典**中存储优化器需要的统计信息：

统计信息

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)

统计信息（续）

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)

统计信息（续）

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)

二、代价估算示例

1、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，那么平均搜索代价

  cost=B/2

9.4.2 基于代价的优化

2. 索引扫描算法的代价估算公式

- 如果选择条件是码＝值
  - 如［例1-C2］，则采用该表的主索引
  - 若为B+树，层数为L，需要存取B+树中从根结点到叶结点L块，再加上基本表中该元组所在的那一块，所以cost=L+1
- 如果选择条件涉及非码属性
  - 如［例1-C3］，若为B+树索引，选择条件是相等比较，S是索引的选择基数(有S个元组满足条件)
  - 最坏的情况下，满足条件的元组可能会保存在不同的块上，此时，cost=L+S

9.4.2 基于代价的优化

2. 索引扫描算法的代价估算公式

- 如果比较条件是>，>=，<，<=操作
  - 假设有一半的元组满足条件就要存取一半的叶结点
  - 通过索引访问一半的表存储块cost=L+Y/2+B/2
  - 如果可以获得更准确的选择基数，可以进一步修正Y/2与B/2

9.4.2 基于代价的优化

3. 嵌套循环连接算法的代价估算公式

- **9.4.1中已经讨论过了嵌套循环连接算法的代价 cost＝Br+(Br/(K-1))*Bs**
- **如果需要把连接结果写回磁盘，cost＝ Br+(Bs/(K-1))* Br +(Frs*Nr*Ns)/Mrs**
  - **其中Frs为连接选择性(join selectivity)，表示连接结果元组数的比例**
  - **Mrs是存放连接结果的块因子，表示每块中可以存放的结果元组数目**

9.4.2 基于代价的优化

4. 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 $(2*B)+(2*B*\log_2 B)$

9.5 SQL调优

- 9.5.1 SQL调优与查询优化器
- 9.5.2 SQL调优导则

9.5.1 SQL调优与查询优化器

- 数据库性能调优一般从发现、分析和解决SQL语句执行中的问题着手，该过程统称为SQL调优（SQL Tuning)
- SQL语句的执行效率依赖于两个方面：（1）执行环境 （2）执行计划
- SQL调优是针对执行计划的

9.5.1 SQL调优与查询优化器

SQL语句的执行计划虽然是由DBMS的查询优化器拟订，但用户仍然可以通过两个途径影响执行计划的拟订。

1. 摸透查询优化器的“脾气”，“投其所好”、“避其所忌”
2. 用提示语句影响查询优化器拟订执行计划

摸透查询优化器的“脾气”

- 如有些查询优化器对于连接操作，一般按连接对象在FROM子句中出现的先后次序进行连接
  - 将小表放在前面，大表放在后面，可以尽快淘汰无用的中间结果

- 又如在有些查询优化器中，凡是查询条件用OR连接的，就一概不用索引
  - 设法改写，使用UNION ALL

摸透查询优化器的“脾气”

```
SELECT *
FROM STUDENT
WHERE YEAR(BDATE)=1980  OR HEIGHT＞1.80;
```

改写成：

```
SELECT *
FROM STUDENT
WHERE YEAR(BDATE)=1980
UNION ALL
SELECT *
FROM STUDENT
WHERE HEIGHT＞=1.80;
```

摸透查询优化器的“脾气”

- 摸透查询优化器的“脾气”并不难，可以用EXPLAIN PLAN语句获得查询优化器所拟订的执行计划，还可以用跟踪语句进一步查得计划的详细执行情况。

```
Execution Plan
----------------------------------------------------
0   SELECT STATEMENT Optimizer = HINT:RULE
1   0   TABLE ACCESS (BY INDEX ROWID)OF 'EMP'
2   1   INDEX (RANGE SCAN)OF'IX_DEPTNO' (NON-UNIQUE)
```

用提示语句影响查询优化器

- 提示是用户影响查询优化器决策的一种手段
  - 优化目标—例如提示以响应时间或吞吐率为优化目标
  - 查询语句变换—例如提示将嵌套查询变换成非嵌套查询等
  - 存取途径选择—例如用某一索引或全表扫描等
  - 连接次序方法—例如提示连接的先后次序，采用嵌套循环或排序归并法等
  - 其他提示

提示举例一

```
SELECT  /*+FIRST  ROW（3）
        优化目标是尽快返回前3行 */
      SNAME，HEIGHT
FROM   STUDENT S
WHERE  SEX=‘男’；
```

提示举例二

```
SELECT  /*+INDEX（EMP EMP_DeptNO_IX）
        采用EMP 表上的索引EMP_DeptNO_IX */
       ENO，ENAME
FROM   EMP
WHERE  DeptNO=100；
```

提示举例三

```
SELECT /*+ USE_NL_WITH_INDEX（EMP_ENO_IX）
        连接采用嵌套循环法，并用索引
        EMP_ENO_IX寻找匹配元组 */
     D.DNO，D.MNO，E.ENAME
FROM  DEPT D，EMP E
WHERE  D.MNO=E.ENO;
```

9.5.2 SQL调优导则

- SQL是非过程语言，同样的功能可有多种不同的表达，这就存在一个如何表达才能提高SQL语句的执行效率问题。
- SQL语句能否有效执行，不但与语句本身有关，还和DBMS及执行环境有关。下面的一些规则在一般情况下是可用的，但不能绝对化。

9.5.2 SQL调优导则

1、尽可能避免排序操作。

对下列语法成分或操作，必须慎用：

- DISTINCT
- 集合操作UNION，INTERSECTION，EXCEPT（MINUS）
- GROUP BY，ORDER BY子句
- 排序归并连接法（除非参与连接的表已经按连接属性排序）

9.5.2 SQL调优导则

2、利用查询条件，尽可能早地消除无用元组，以缩小中间结果：

（1）能用WHERE子句表示的查询条件，不要放在HAVING子句中。

（2）当心在语句中出现笛卡尔乘积（即无连接条件的连接），慎用外连接操作、非等连接（即连接条件中出现非等号比较符）等操作。

（3）在多元连接时，须将选择性高、元组少的表列在FROM子句的前两位，其他表也要按选择性高和元组数少优先原则依次排序，以缩小中间结果。

如查询计算机系所开课程的平均成绩和最高成绩:

```
SELECT  C.CNO，AVG(SC.GRADE)，MAX(SC.GRADE)
FROM  SC，C
WHERE  C.CNO=SC.CNO
GROUP  BY C.CNO
HAVING  C.CNO='CS%';
```

改写成

```
SELECT  C.CNO，AVG(SC.GRADE)，MAX(SC.GRADE)
FROM  SC，C
WHERE  C.CNO='CS%' AND C.CNO=SC.CNO
GROUP BY  C.CNO
```

9.5.2 SQL调优导则

3、大量加载数据时，不要用INSERT语句

- 应利用加载工具，例如Oracle中的SQL*Loader

4、EXISTS，IN的应用

- 慎用关联嵌套子查询，在多数情况，IN比EXISTS较为有利

9.5.2 SQL调优导则

5、不滥用视图

- 视图是按其定义临时生成的中间结果，没有索引之类的存取路径可用。
- 虽然允许视图的定义中再引用其他视图，但就性能而言，这种多层嵌套的视图不值得采用。如果不是出于安全、保密等的考虑，可将视图定义合并到FROM子句中。

6、当心查询优化器选择全表扫描

- 如果查询条件中出现比较符“＜＞”或“IS NULL”等，则查询优化器一般选择全表扫描

9.5.2 SQL调优导则

总之，在编写SQL语句时，设计者都得想一想：查询优化器可能为所设计的语句选用哪样的执行计划，以免出现意外的耗费资源的“大户”，这样做是有益的。

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
  - 基于代价的存取路径优化

小结

- 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

- 对于RDBMS不能优化的查询需要重写查询语句，进行手工调整以优化性能

作业

第五版：

P.290

2(假设内存有k块可用),3,4,5

作业

第五版 P.290 3

```
select cname
from s, c, sc
where s.sno=sc.sno and sc.cno=c.cno and s.sdept='IS'
```

第五版 P.290 5

```
select tname
from teacher, department, work
where teacher.tno=work.tno and
        department.dno=work.dno and dname='CS'
        and salary>5000
```