

- **关系查询：用户只需提出“干什么”，不必指出“怎么干”**
- **存取路径的选择以及SQL的操作过程由系统自动完成**

数据库查询操作很重要的就是性能问题，即：如何快速并且资源最省的将结果从庞大的数据库中检索出来，这需要DBMS对用户书写的非过程查询语句进行优化，以选择最优的方案予以执行。

# 第十一章 关系查询处理和查询优化

# **第十一章 关系查询处理和查询优化**

## **11.1 关系数据库系统的查询处理**

## **11.2 关系数据库系统的查询优化**

## **11.3 代数优化**

## **11.4 物理优化**

## **11.5 小结**

# 第十一章 关系查询处理和查询优化

## 11.1 关系数据库系统的查询处理

## 11.2 关系数据库系统的查询优化

## 11.3 代数优化

## 11.4 物理优化

## 11.5 小结

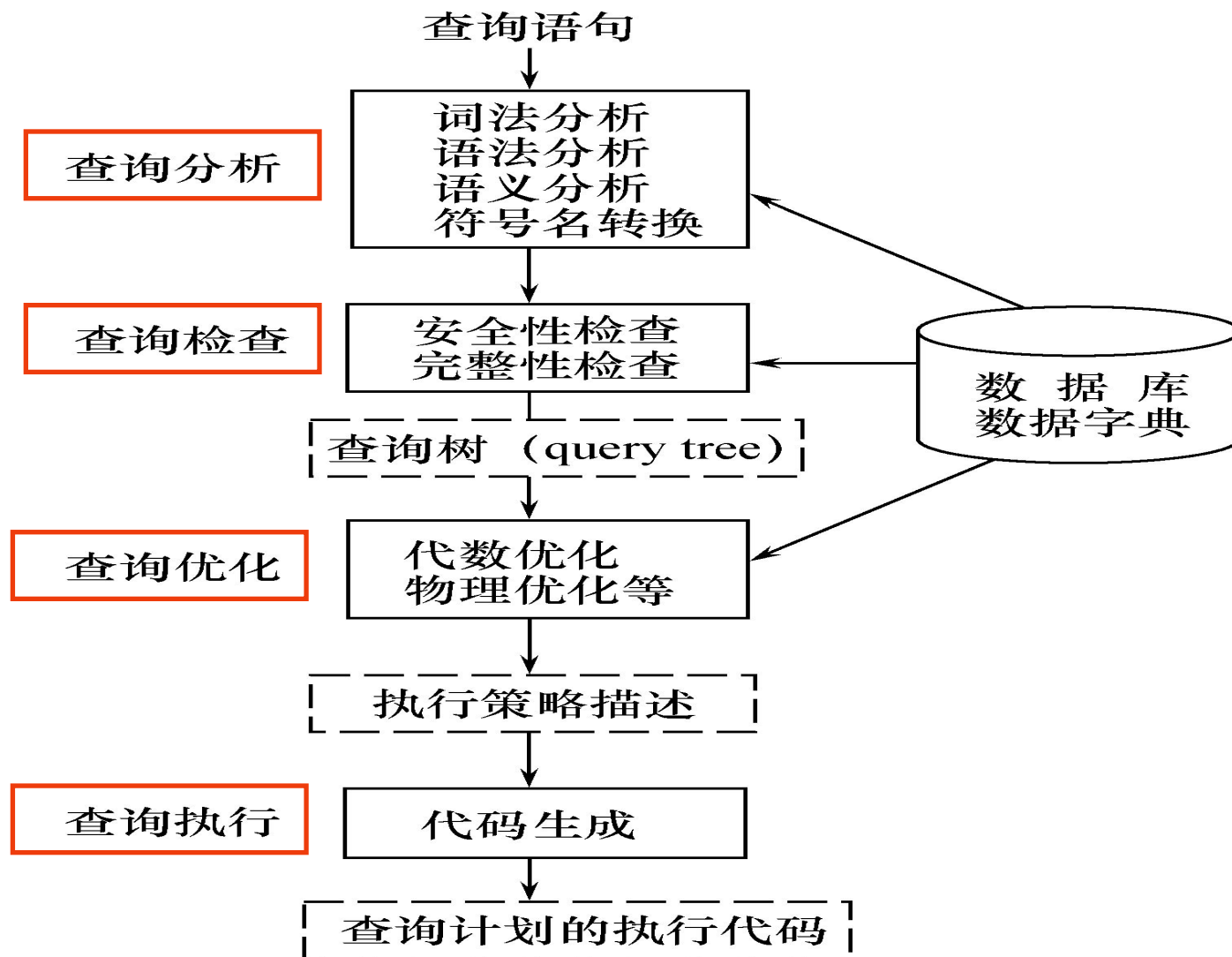
# 11.1 关系数据库系统的查询处理

## ■ 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

# 查询处理步骤



# 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**
- **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
- 把数据库对象的外部名称转换为内部表示



### 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(code generator)生成执行查询计划的代码



**如：检索学过数据结构课程的所有同学的学号、姓名及成绩。**

**Select S#, Sname, Score**

**From Student, SC, Course**

**Where Cname = ‘数据结构’ and Student.S#  
= SC.S# and Course.C# = SC.C# ;**

**上述SQL语句可以按照一种标准方式，转换成如下形式的关系代数语句。**

$\pi_{S\#, Sname, Score}(\sigma_{Cname="数据结构" \text{ and } Student.S\# = SC.S\# \text{ and } Course.C\# = SC.C\#}(Student \times SC \times Course))$

**因此有两个问题：**

- (1) 是否能按照上式表征的操作顺序来执行？按上式顺序执行，效率如何？是否有其他更快速的方式？**
- (2) 每一个操作，即关系代数的并、差、积、选择和投影、连接等操作如何实现？**

**对前一个问题，我们可以有多条关系代数语句来表达与上述操作顺序同样的检索需求，但哪一个更好呢？**

$\pi_{S\#, Sname, Score}(\sigma_{Cname=\text{“数据结构”}}(Student \bowtie SC \bowtie Course))$

$\pi_{S\#, Sname, Score}(Student \bowtie (SC \bowtie (\sigma_{Cname=\text{“数据结构”}}(Course))))$

$\pi_{S\#, Sname, Score}(\pi_{S\#, Sname}(Student) \bowtie (SC \bowtie (\pi_{C\#}(\sigma_{Cname=\text{“数据结构”}}(Course)))))$

**前述示例说明用户给出了检索需求后，我们可以从语法上予以优化，从而选择出最优的操作顺序。关系代数是解决查询优化问题的一个很重要的方法。**

**对后一个问题，即当语法优化进行完毕后，如何执行每一个具体的操作呢？我们说，对每一个具体操作也有多种执行方案。**

**例如： $\sigma_{Cname='数据结构'}(Course)$ 可以有以下一些查询实现方案：**

### **方案1：表空间扫描方法**

**直接对Course表进行扫描，从第一条检索到最后一条，对每一条进行条件比较，将满足条件的记录找出。**

### **方案2：利用Course上的Cname排序索引的方法**

**利用排序索引可以进行诸如二分查找等快速检索，找到满足条件的相应索引项，再据索引指针将满足条件的记录找出。**

**当条件更复杂时，可选择的方案还会更多。**



**再例如：**  $R \bowtie S$

**方案1：**先将R全部读入内存(或尽可能多地读入)，然后反复访问磁盘上的S，进行相应记录的连接操作。

**方案2：**先将S全部读入内存(或尽可能多地读入)，然后反复访问磁盘上的R，进行相应记录的连接操作。

**方案3：**将内存分成较均匀的两部分，同时读入R, S的一部分，然后反复访问磁盘，读取R, S的剩余部分；进行相应记录的连接操作。

**再例如：**  $R \bowtie S$

**当考虑文件组织方式和索引结构时，可选择的方案还会更多，比如是否预先排序，是否使用索引(没有索引是否先建立临时索引)，... ..。**

**嵌套循环法**

**排序-合并法**

**索引连接法**

**Hash Join法**

- **DBMS如何衡量这些方案的优劣呢？**
  - **衡量CPU的占用时间**
  - **衡量I/O访问次数**
  - **内存使用代价(与缓冲区数目与大小的匹配)**
  - **中间结果存储代价**
  - **计算量(如搜索记录、合并记录、排序记录、字段值的计算等)**
  - **网络通信量**
  - **... ..**

- **依据什么信息来计算这些方案的上述各种指标呢？可以依据数据库本身的一些统计信息(Catalog Information)**
  - **nr: 关系r的元组数目**
  - **br: r占用的数据块的数目**
  - **sr: r的元组的大小**
  - **fr: r的块因子，即一块中能够存储的r的元组数目**
  - **$V(A, r)$ : r中属性A出现不同值的数目，即  $\Pi A(r)$  的数目.**
  - **$SC(A, r)$ : r中属性A的选择基数，满足A上等价性的平均记录数**
  - **... ..**

## ■ 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)

# 11.1 关系数据库系统的查询处理

## ■ 实现查询操作的算法示例

### □ 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;

# 11.1 关系数据库系统的查询处理


## ■ 实现查询操作的算法示例

### □ 选择操作的实现

#### ■ 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

Students



A vertical arrow points downwards from the text '适合小表，不适合大表' to a box containing a list of student IDs. This illustrates the full table scan method where every row is checked sequentially.

2011001, ...
2011002, ...
2011003, ...
2011004, ...
2011005, ...
...

# 11.1 关系数据库系统的查询处理

## ■ 实现查询操作的算法示例

### □ 选择操作的实现

#### ■ 索引(或散列)扫描方法

- 适合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



# 11.1 关系数据库系统的查询处理

## ■ 实现查询操作的算法示例

### □ 选择操作的实现

#### ■ 索引(或散列)扫描方法

**[ 例1-C2 ] 以C2为例 , Sno = '200215121' , 并且Sno上有索引(或Sno是散列码)**

- 使用索引(或散列)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

# 11.1 关系数据库系统的查询处理

- 实现查询操作的算法示例

- 选择操作的实现

- 索引(或散列)扫描方法

**[ 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。

# 11.1 关系数据库系统的查询处理

## ■ 实现查询操作的算法示例

### □ 选择操作的实现

#### ■ 索引(或散列)扫描方法

[ 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

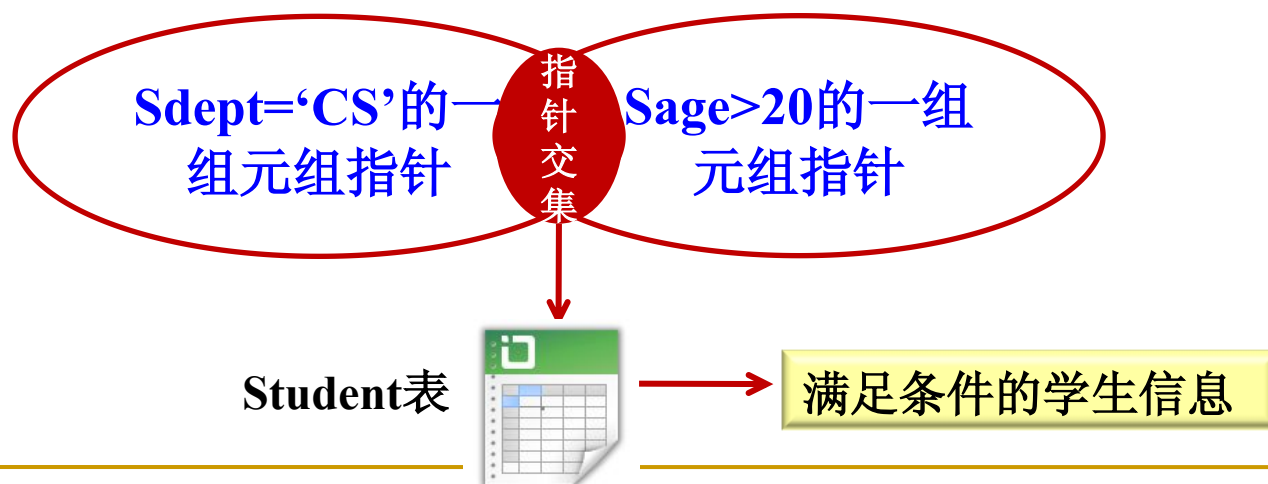
# 11.1 关系数据库系统的查询处理

## ■ 实现查询操作的算法示例

### □ 选择操作的实现

#### ■ 索引(或散列)扫描方法

[ 例1-C4 ] 以C4为例 , Sdept = 'CS' AND Sage > 20 , 如果Sdept和Sage上都有索引 :



# 11.1 关系数据库系统的查询处理

## ■ 实现查询操作的算法示例

### □ 选择操作的实现

#### ■ 索引(或散列)扫描方法

[ 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。

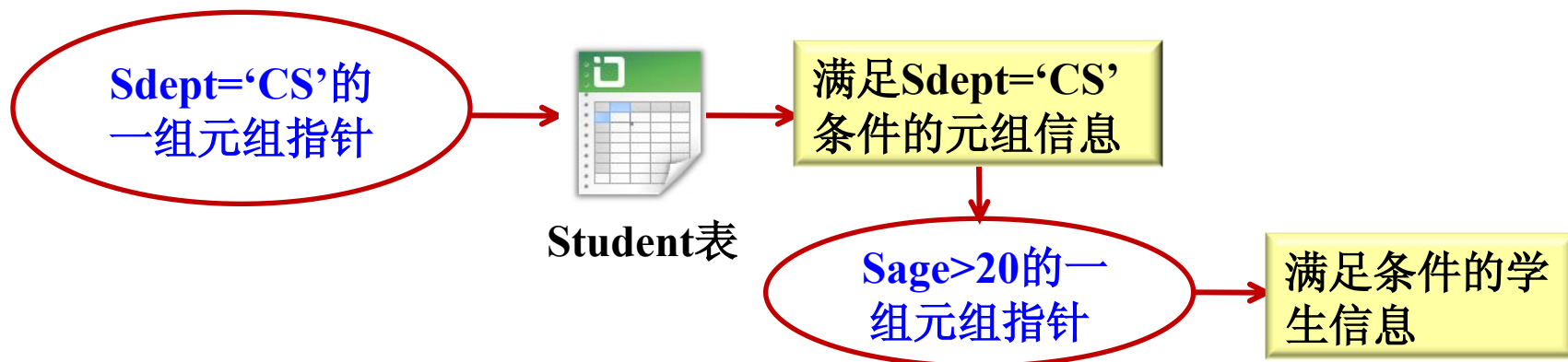
# 11.1 关系数据库系统的查询处理

## ■ 实现查询操作的算法示例

### ■ 选择操作的实现

#### ■ 索引(或散列)扫描方法

[ 例1-C4 ] 以C4为例, Sdept = 'CS' AND Sage > 20, 如果Sdept和Sage上都有索引:



# 11.1 关系数据库系统的查询处理

## ■ 实现查询操作的算法示例

### ■ 连接操作的实现

- 嵌套循环方法(*nested loop*)
- 排序-合并方法(*sort-merge join* 或 *merge join*)
- 索引连接(*index join*)方法
- *Hash Join*方法

# 11.1 关系数据库系统的查询处理

## 例 连接操作的实现

Select \*

From Student,SC

Where Student.Sno=SC.Sno

### ① 嵌套循环方法

Student

学号 Sno	姓名 Sname	性别 Ssex	年龄 Sage	所在系 Sdept
95002	李勇	男	20	CS
95001	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

SC

学号 Sno	课程号 Cno	成绩 Grade
95001	1	92
95002	2	90
95001	3	88
95002	2	85
95002	3	80

对Student中的每一个元组，检查SC中的每一个元组，若在连接属性上相等，则连接输出，直到Student中的元组处理完为止。



# 11.1 关系数据库系统的查询处理

## ② 排序-合并方法

SC

Student

学号 Sno	姓名 Sname	性别 Ssex	年龄 Sage	所在系 Sdept
95001	李勇	男	20	CS
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

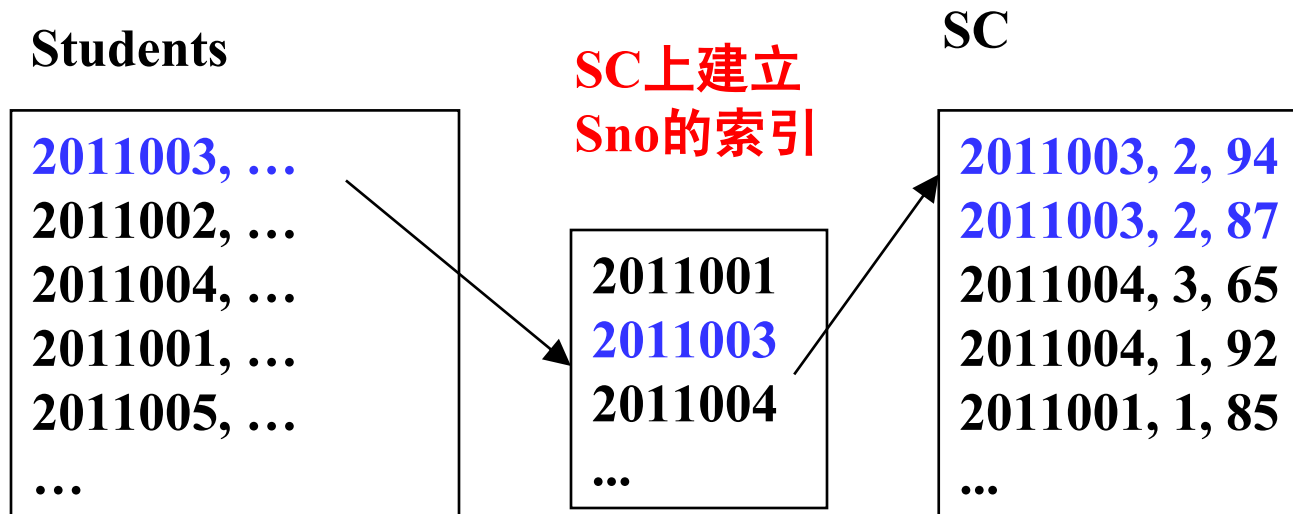
学号 Sno	课程号 Cno	成绩 Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

取Student表中第一个Sno，依次扫描SC表中具有相同Sno的元组，把他们连接起来；

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。

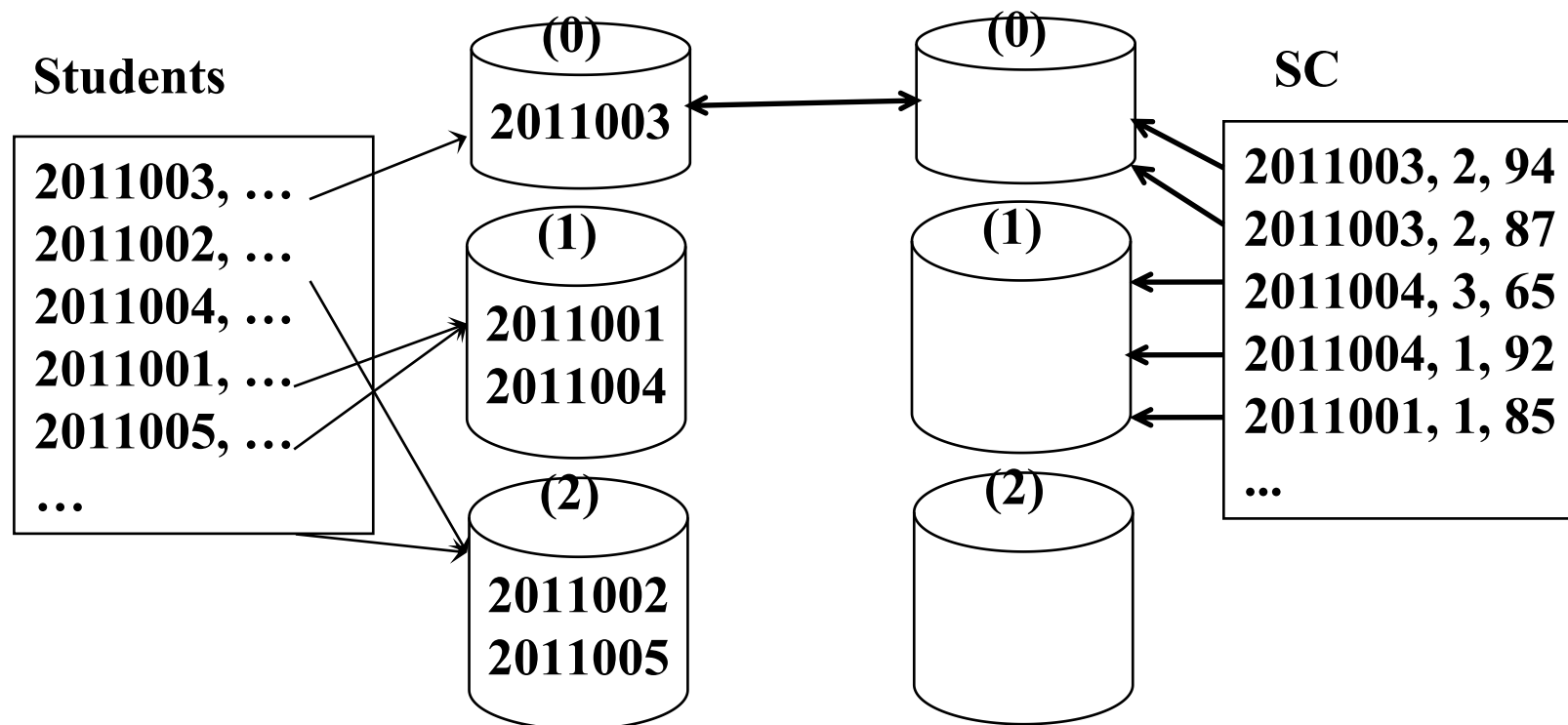
# 11.1 关系数据库系统的查询处理

## ②索引连接方法



# 11.1 关系数据库系统的查询处理

## ②Hash连接方法



# **第十一章 关系查询处理和查询优化**

## **11.1 关系数据库系统的查询处理**

## **11.2 关系数据库系统的查询优化**

## **11.3 代数优化**

## **11.4 物理优化**

## **11.5 小结**

# 第十一章 关系查询处理和查询优化

## 11.1 关系数据库系统的查询处理

## 11.2 关系数据库系统的查询优化

## 11.3 代数优化

## 11.4 物理优化

## 11.5 小结

# 第十一章 关系查询处理和查询优化

- **关系系统的查询优化既是RDBMS实现的关键技术又是关系系统的优点所在。**
- **它减轻了用户选择存取路径的负担，用户只需提出“干什么”，不必指出“怎么干”。**
- **查询优化的优点不仅在于用户不必考虑如何最好地表达查询以获得较好的效率，而且在于系统可以比用户程序的“优化”做得更好**

# 第十一章 关系查询处理和查询优化

**(1) 优化器可以从数据字典中获取许多统计信息，而用户程序则难以获得这些信息**

**(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。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。**

# 第十一章 关系查询处理和查询优化

**(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**



## 11.2 关系数据库系统的查询优化

**[例2] 求选修了2号课程的学生姓名。用SQL表达：**

```
SELECT Student.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**

## 11.2 关系数据库系统的查询优化

- 系统可以用多种等价的关系代数表达式来完成这一查询

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

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

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

# 一、第一种情况

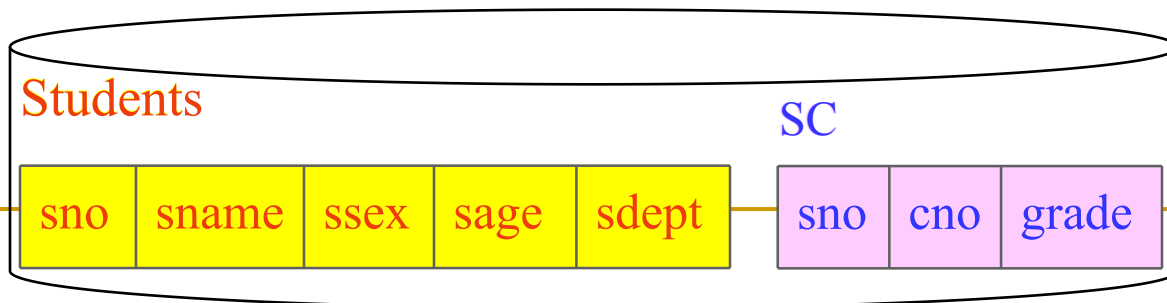
$$Q_1 = \pi_{\text{Sname}}(\sigma_{\text{Student.Sno}=\text{SC.Sno} \wedge \text{Sc.Cno}='2'} \text{Student} \times \text{SC})$$

## 1. 计算广义笛卡尔积

内存  
(DBMS驻留)

- ❑ 硬盘的数据以何种形式被加载到内存中？
- ❑ 内存是否足够大？

硬盘/外存



内存  
(DBMS驻留)

## 第1步：加载数据

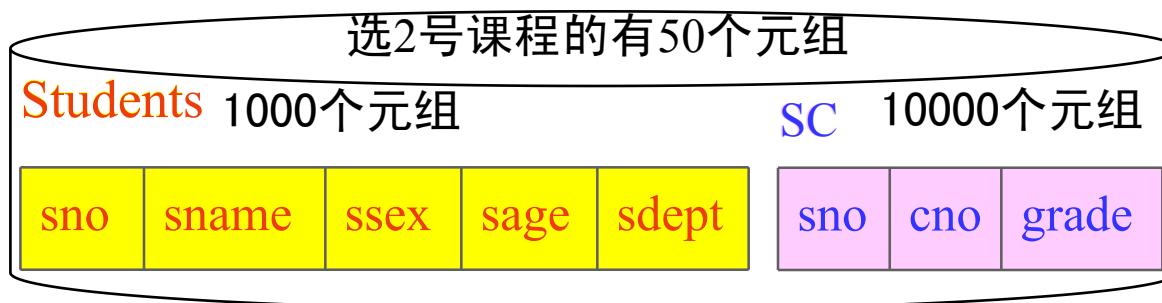
10个Students元组	10个Students元组	10个Students元组
10个Students元组	10个Students元组	100个SC元组

## 第2步：内存中进行关系操作

## 第3步：输出最终结果;或 产生临时结果, 写回硬盘, 重复1-2步

主要考虑  
I/O代价

硬盘  
/外存



# 一、第一种情况

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

## 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
  - 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、第一种情况

### 1. 计算广义笛卡尔积

- 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

10个Students元组	10个Students元组	10个Students元组
10个Students元组	10个Students元组	100个SC元组

- 则读取总块数为

$$\frac{1000}{10} + \frac{1000}{10 \times 5} \times \frac{10000}{100} = 2100$$

- 其中，读Student表100块。读SC表20遍，每遍100块
- 设每秒读写20块，则总计要花105s

# 一、第一种情况

## 1. 计算广义笛卡尔积

- 连接后的元组数为  $10^3 \times 10^4 = 10^7$ 。设每块能装10个元组，则写出这些块要用  $10^6/20 = 5 \times 10^4$ s

# 一、第一种情况

## 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 $5 \times 10^4 \text{s}$
- 满足条件的元组假设仅50个，均可放在内存



# 一、第一种情况

## 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  
 $\approx 105 + 2 \times 5 \times 10^4 \approx 10^5 \text{s} \approx 28 \text{小时}$
- ❑ 所有内存处理时间均忽略不计

## 二、第二种情况

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

### 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104个
- 写出这些元组时间为 $104/10/20=50\text{s}$ ，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 $\approx 105 + 50 + 50 \approx 205\text{s} \approx 3.4\text{分钟}$

### 三、第三种情况

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

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 $\approx 5 + 5 \approx 10s$

- **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总共3~4块**
- **若Student表在Sno上也有索引**
  - **第二步也不必读取所有的Student元组**
  - **因为满足条件的SC记录仅50个，涉及最多50个Student记录**
  - **读取Student表的块数也可大大减少**
- **总的存取时间将进一步减少到数秒**

## 11.2 关系数据库系统的查询优化

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

# **第十一章 关系查询处理和查询优化**

## **11.1 关系数据库系统的查询处理**

## **11.2 关系数据库系统的查询优化**

## **11.3 代数优化**

## **11.4 物理优化**

## **11.5 小结**

# 第十一章 关系查询处理和查询优化

## 11.1 关系数据库系统的查询处理

## 11.2 关系数据库系统的查询优化

## 11.3 代数优化

## 11.4 物理优化

## 11.5 小结

## 11.3 代数优化

- **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 $E_1 \equiv E_2$**



## 11.3 代数优化-关系代数表达式等价变换规则

### ■ 常用的等价变换规则：

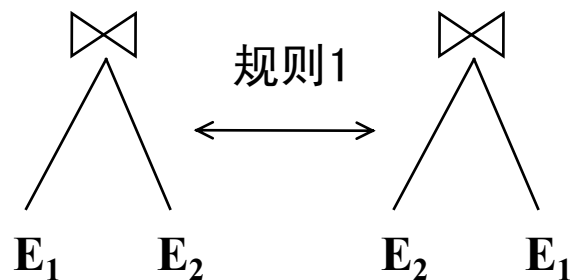
#### 1. 连接、笛卡尔积交换律

设 $E_1$ 和 $E_2$ 是关系代数表达式， $F$ 是连接运算的条件，则有

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

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

$$E_1 \bowtie_F E_2 \equiv E_2 \bowtie_F E_1$$



# 11.1 代数优化-关系代数表达式等价变换规则

## ■ 常用的等价变换规则：

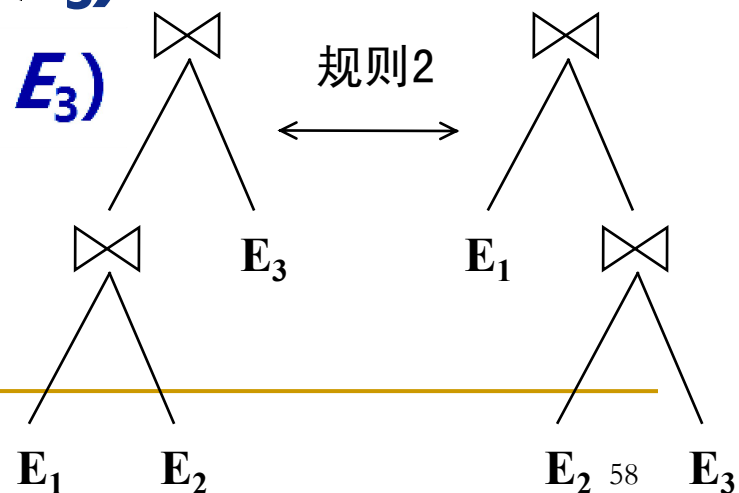
### 2. 连接、笛卡尔积的结合律

设  $E_1, E_2, E_3$  是关系代数表达式， $F_1$  和  $F_2$  是连接运算的条件，则有

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

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

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



# 11.1 代数优化-关系代数表达式等价变换规则

## ■ 常用的等价变换规则：

### 3. 投影的串接定律

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

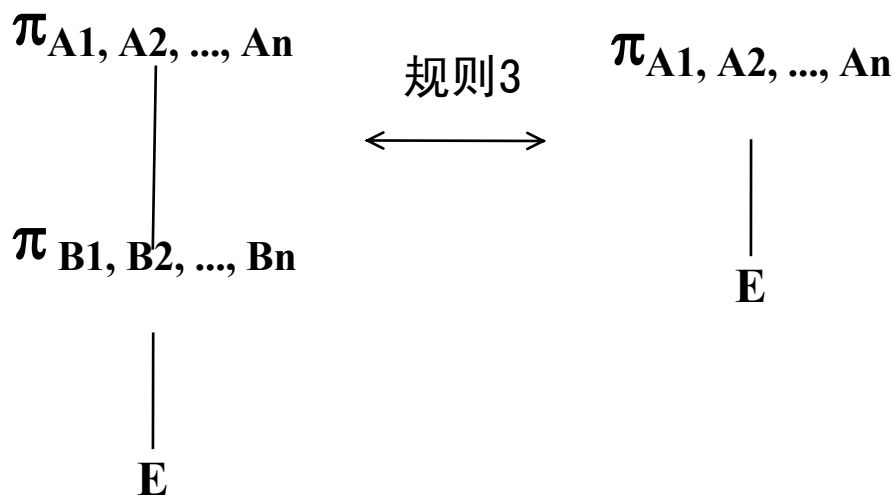
这里， $E$ 是关系代数表达式， $A_i(i=1, 2, \dots, n)$ ， $B_j(j=1, 2, \dots, m)$ 是属性名且 $\{A_1, A_2, \dots, A_n\}$ 构成 $\{B_1, B_2, \dots, B_m\}$ 的子集。

# 11.1 代数优化-关系代数表达式等价变换规则

## ■ 常用的等价变换规则：

### 3. 投影的串接定律

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



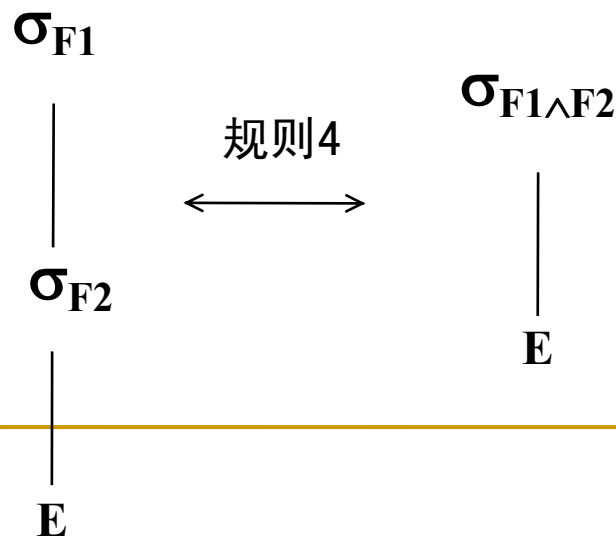
# 11.1 代数优化-关系代数表达式等价变换规则

## 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， $F_1$ 、 $F_2$ 是选择条件。

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。

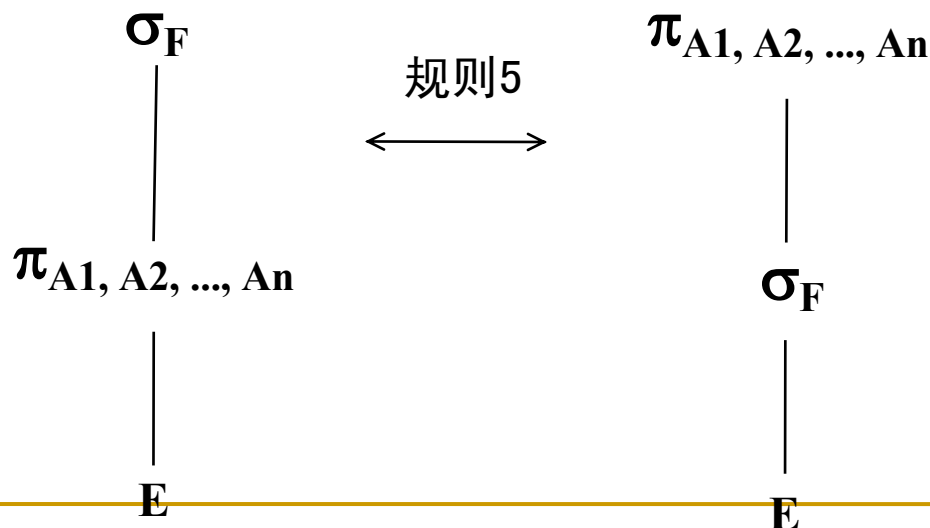


# 11.1 代数优化-关系代数表达式等价变换规则

## 5. 选择与投影操作的交换律

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

选择条件F只涉及属性 $A_1, \dots, A_n$

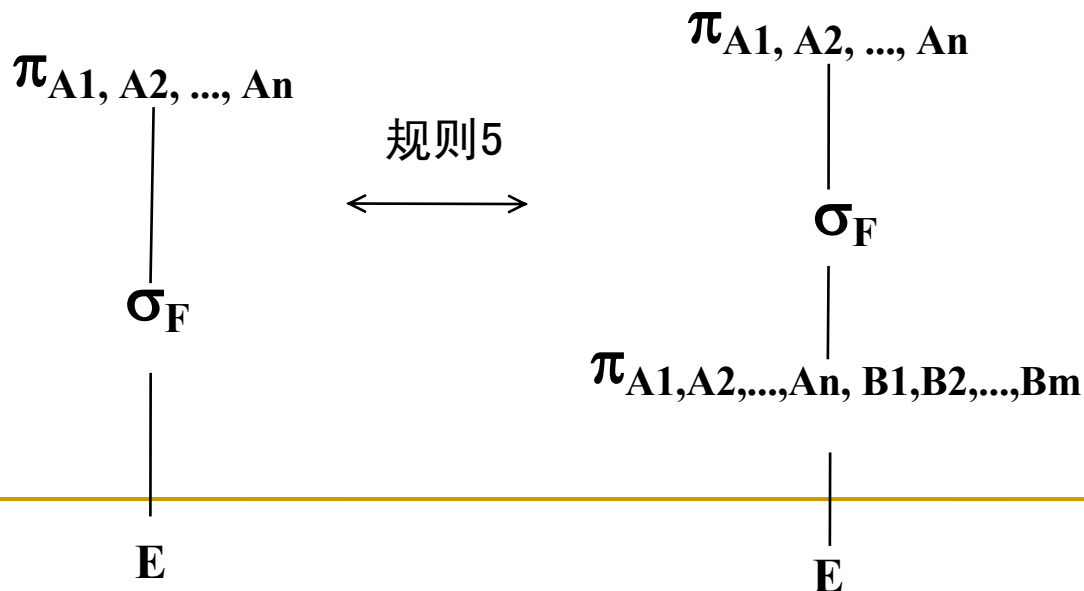


# 11.1 代数优化-关系代数表达式等价变换规则

## 5. 选择与投影操作的交换律

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

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

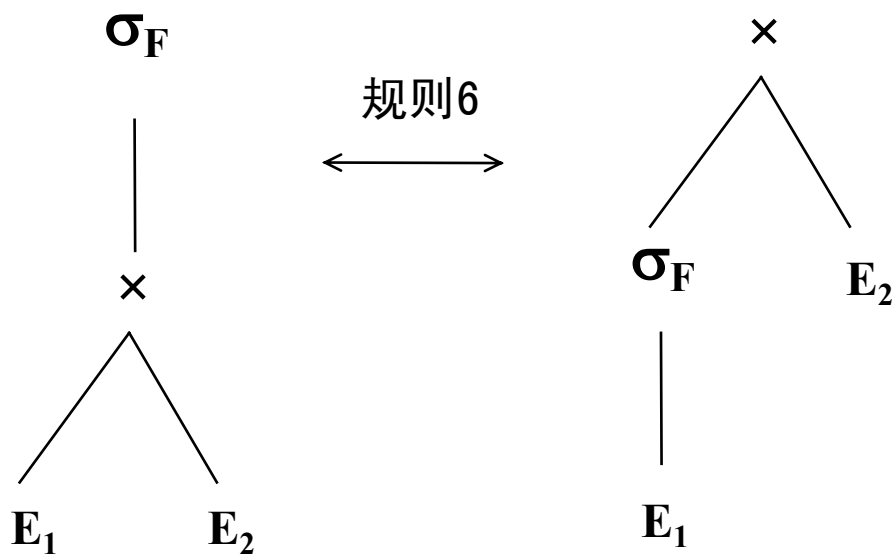


# 11.1 代数优化-关系代数表达式等价变换规则

## 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

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



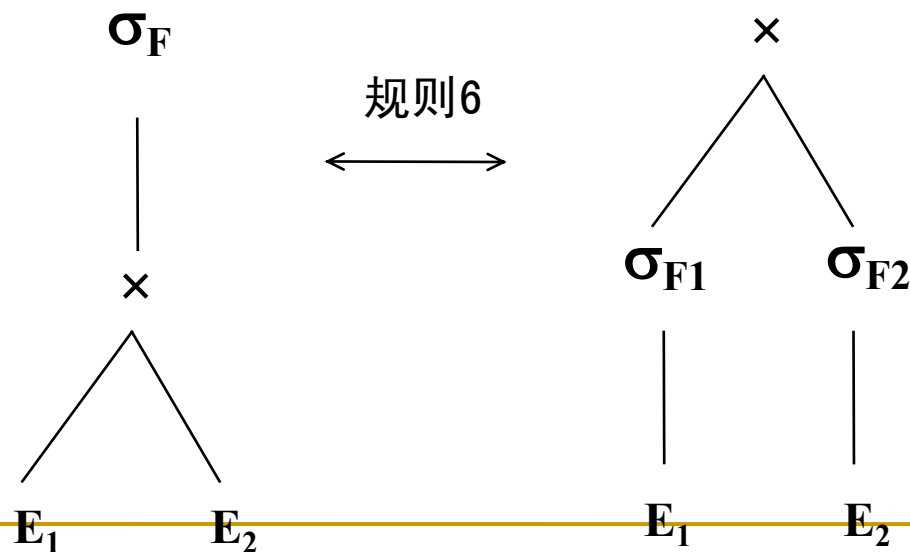


# 11.1 代数优化-关系代数表达式等价变换规则

## 6. 选择与笛卡尔积的交换律

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：

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



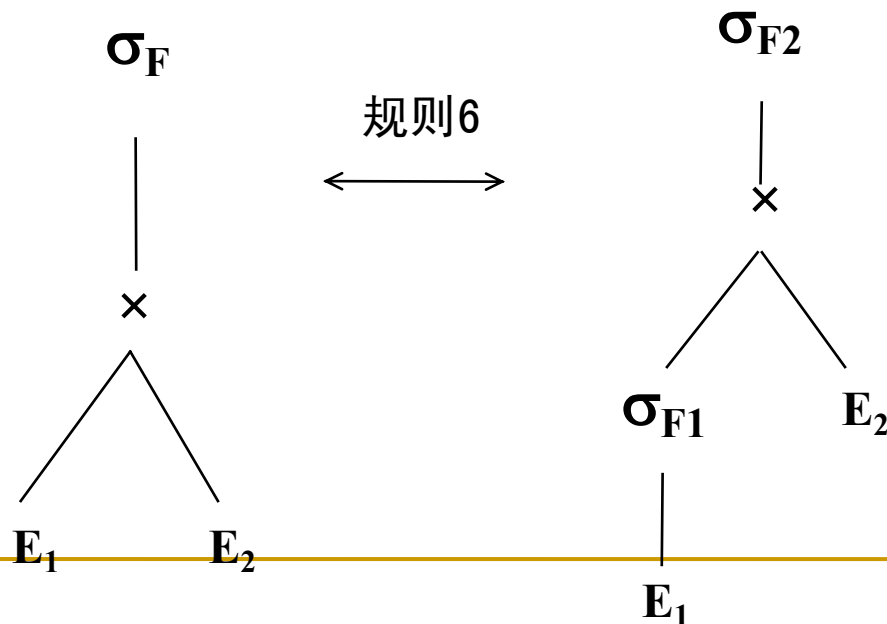
# 11.1 代数优化-关系代数表达式等价变换规则

## 6. 选择与笛卡尔积的交换律

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

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

规则6使部分选择在笛卡尔积前先做。

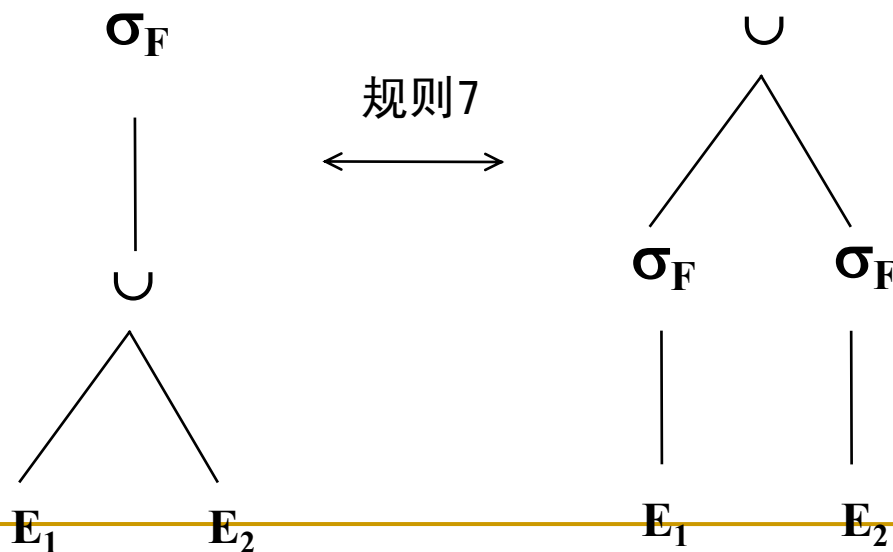


# 11.1 代数优化-关系代数表达式等价变换规则

## 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

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

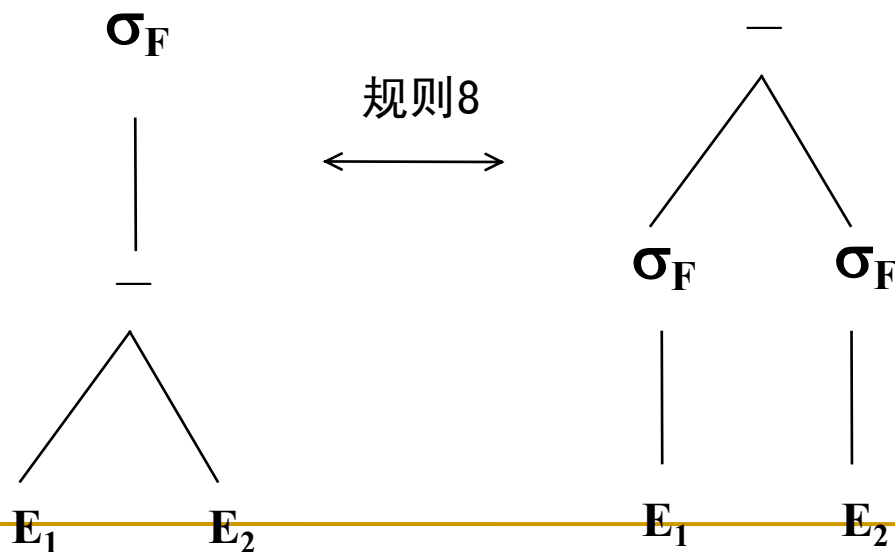


# 11.1 代数优化-关系代数表达式等价变换规则

## 8. 选择与差运算的分配律

若 $E_1$ 与 $E_2$ 有相同的属性名，则

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

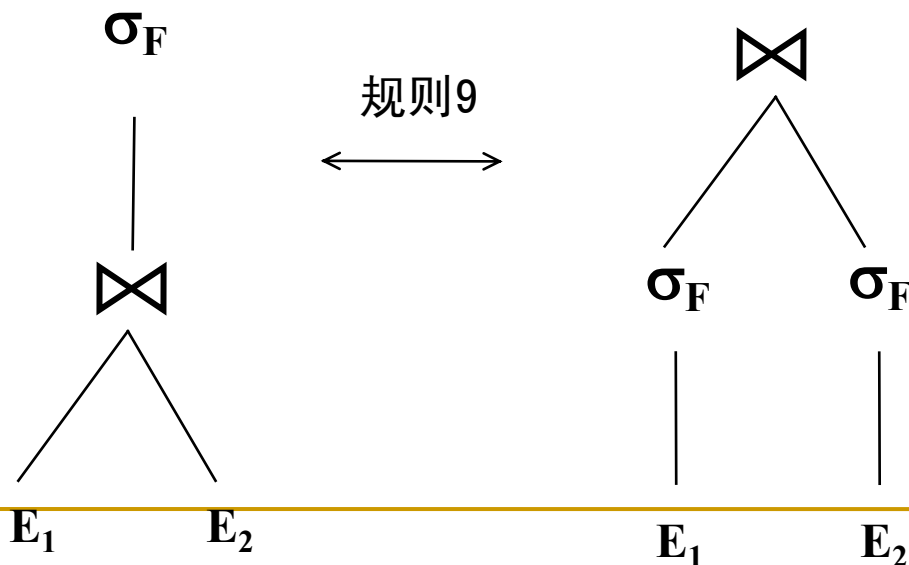


# 11.1 代数优化-关系代数表达式等价变换规则

## 9. 选择对自然连接的分配律

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

F只涉及 $E_1$ 与 $E_2$ 的公共属性



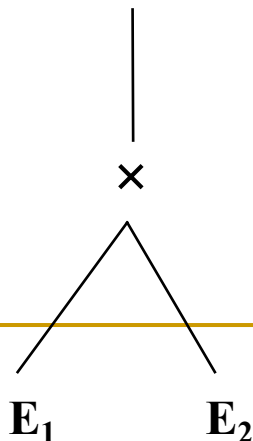
# 11.1 代数优化-关系代数表达式等价变换规则

## 10. 投影与笛卡尔积的分配律

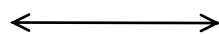
设 $E_1$ 和 $E_2$ 是两个关系表达式， $A_1, \dots, A_n$ 是 $E_1$ 的属性， $B_1, \dots, B_m$ 是 $E_2$ 的属性，则

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

$\pi_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}$



规则10



×

$\pi_{A_1, A_2, \dots, A_n}$

$\pi_{B_1, B_2, \dots, B_m}$

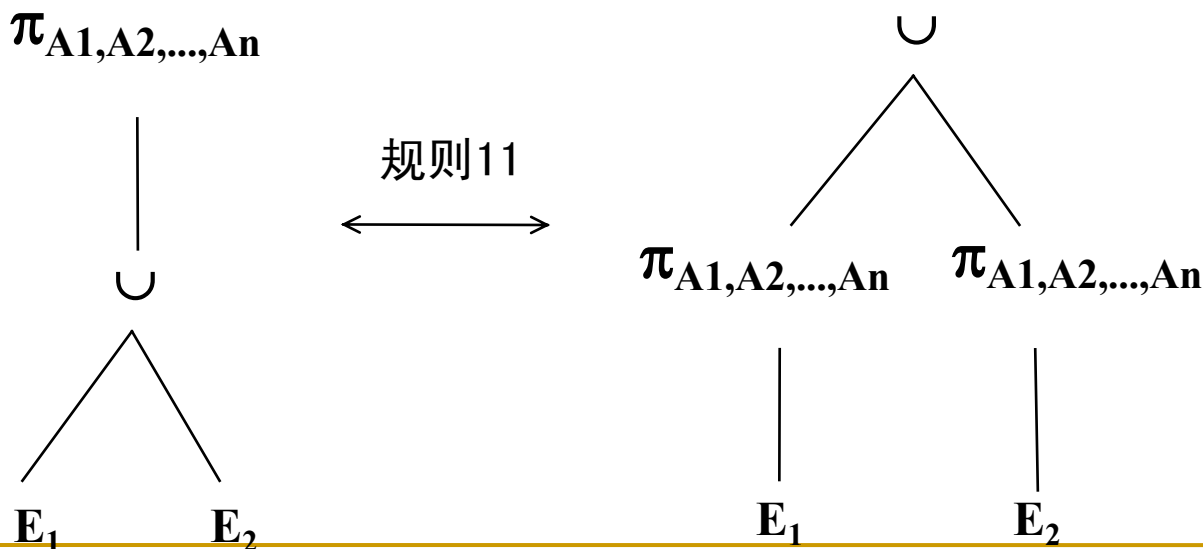


# 11.1 代数优化-关系代数表达式等价变换规则

## 11. 投影与并的分配律

设 $E_1$ 和 $E_2$ 有相同的属性名，则

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



# 11.1 代数优化-查询树的启发式优化

## ■ 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. 投影运算尽可能先做。



# 11.1 代数优化-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**

原始语法树：

select--投影

from--笛卡尔积

where--选择

## 11.1 代数优化-查询树的启发式优化

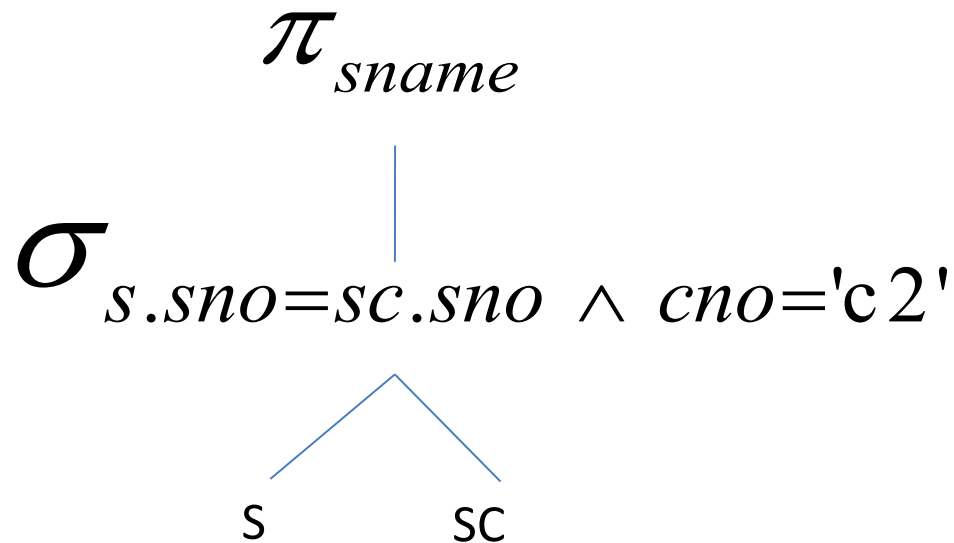
**方法：**

- (1)将合取选择分解为单个选择运算系列**
- (2)把选择运算尽可能移到查询树的下面。**
- (3)将产生最小关系的连接运算先做**
- (4)将选择运算和连接运算结合在一起做**
- (5)把投影运算可能移到查询树的下面，如有必要，引入新的投影运算。**

**select sname from s,sc**

**where s.sno=sc.sno and sc.cno='c2'**

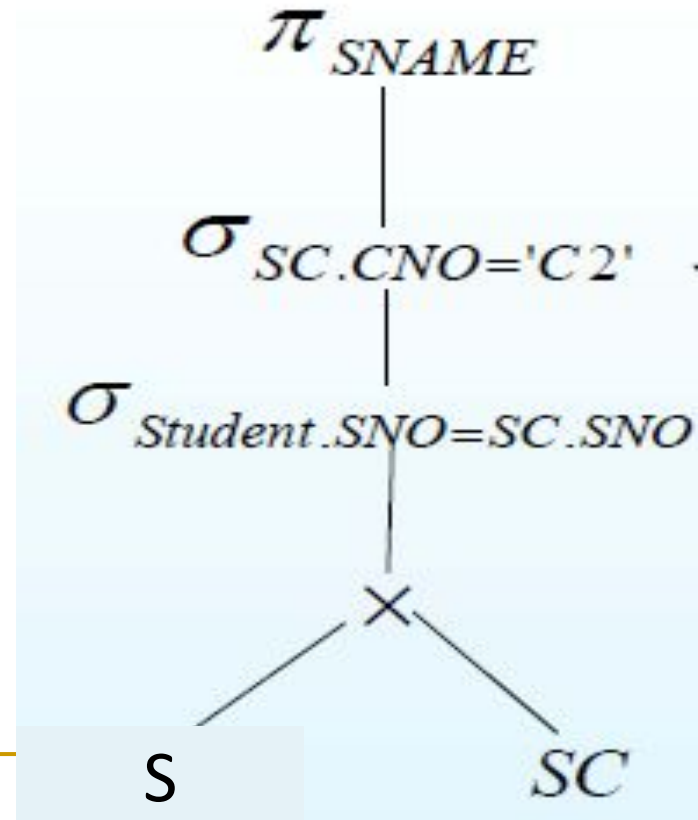
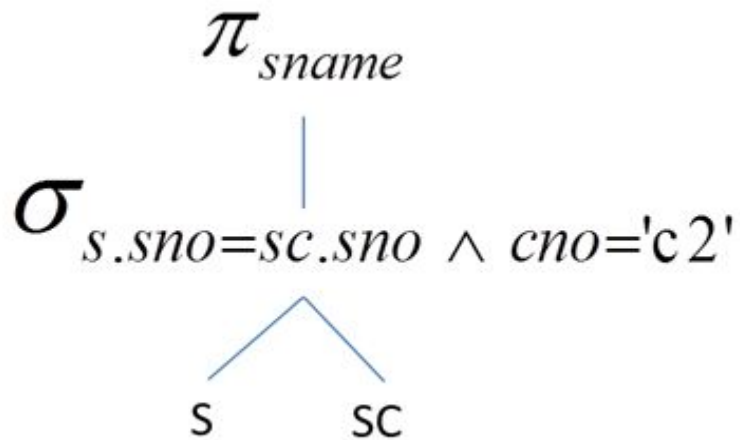
$$Q1 = \pi_{SNAME} (\sigma_{Student.SNO=SC.SNO \wedge SC.CNO='c2'} (Student \times SC))$$



# 将合取选择分解为单个选择运算系列

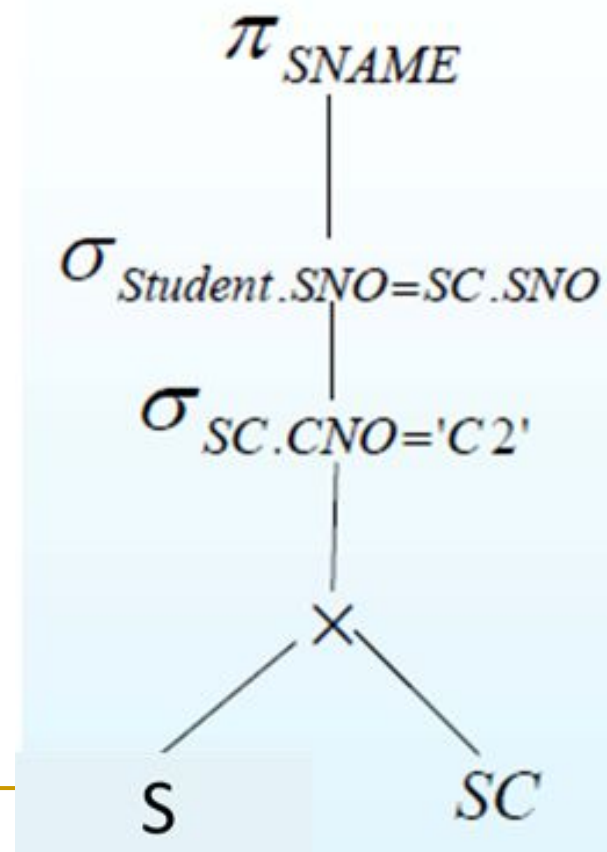
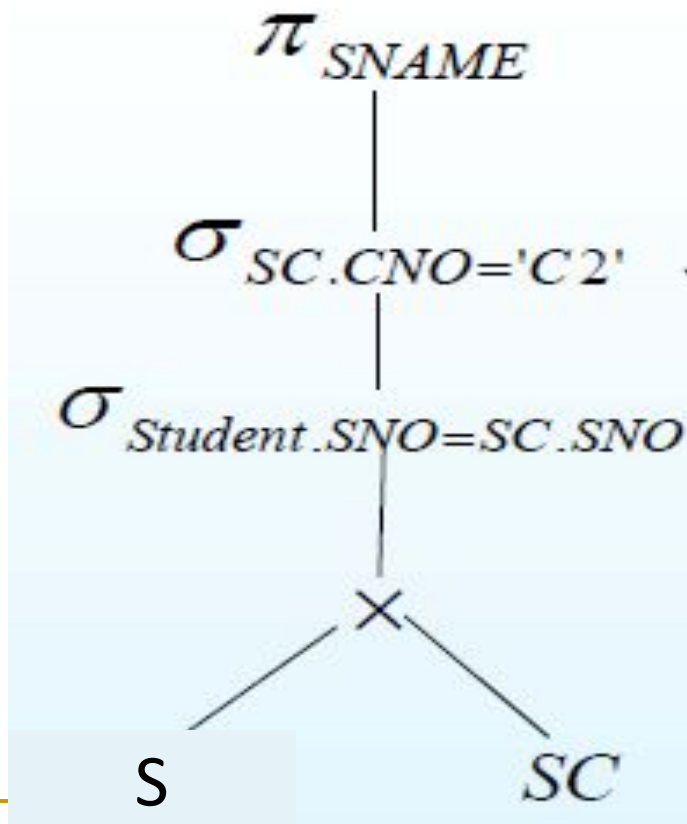
## 应用规则4选择的串接定律

$$\sigma_{s.sno=sc.sno \wedge cno='c2'}(E) \equiv \sigma_{cno='c2'}(\sigma_{s.sno=sc.sno}(E))$$



## 把选择运算尽可能移到查询树的下面

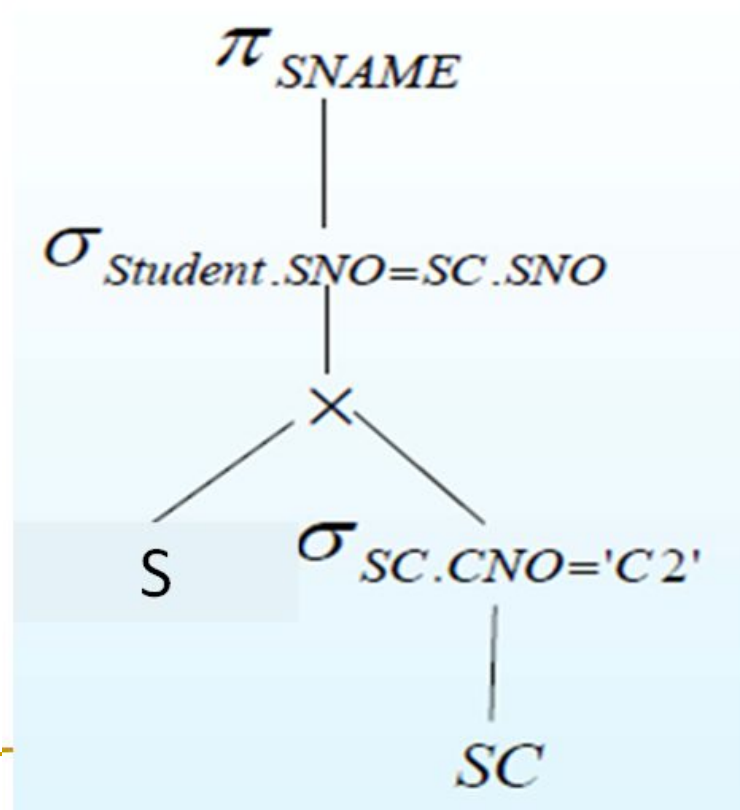
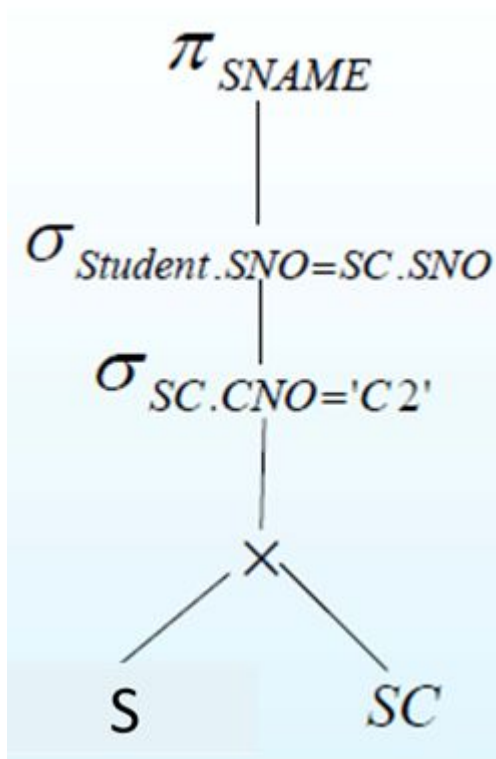
$$\sigma_{cno='c2'}(\sigma_{s.sno=sc.sno}(E)) \equiv \sigma_{s.sno=sc.sno}(\sigma_{cno='c2'}(E))$$



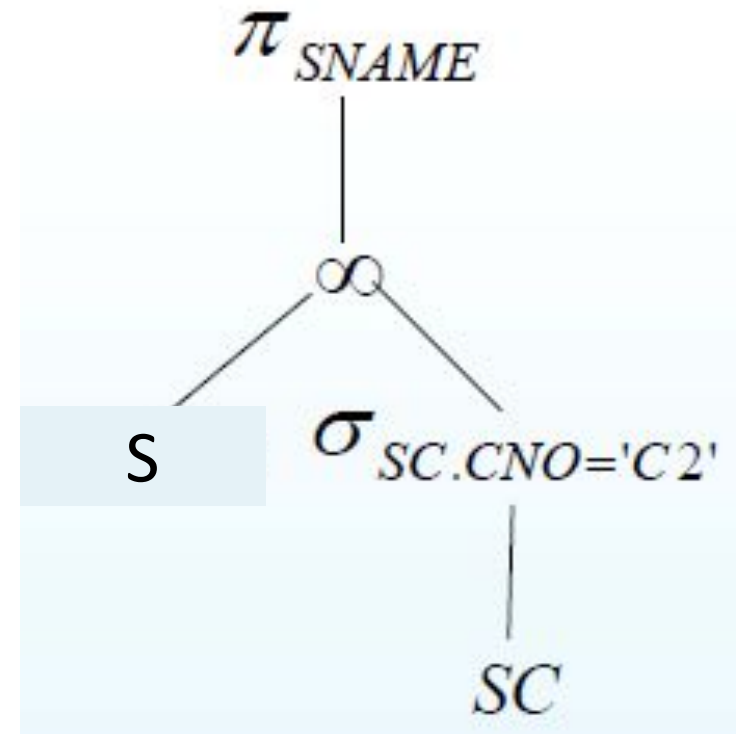
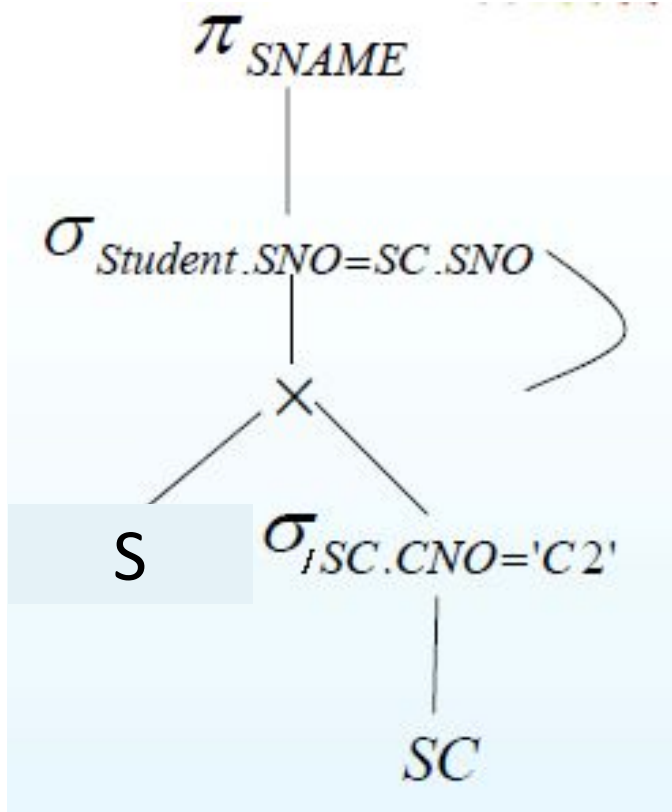
## 把选择运算尽可能移到查询树的下面

### 应用规则6选择与笛卡尔积的交换律

$$\sigma_{cno='c2'}(S \times SC) = S \times (\sigma_{cno='c2'}(SC))$$



## 将选择运算和连接运算结合在一起做



把投影运算可能移到查询树的下面，如有必要，引入新的投影运算。

### 应用规则3投影的串接定律

$$\pi_A(\pi_B(E)) \equiv \pi_A(E)$$

$$A \subseteq B$$

$\pi_{SNAME}$



S

$\sigma_{SC.CNO='C2'}$

SC

$\pi_{sname}$

$\pi_{sname, s.sno, sc.sno}$



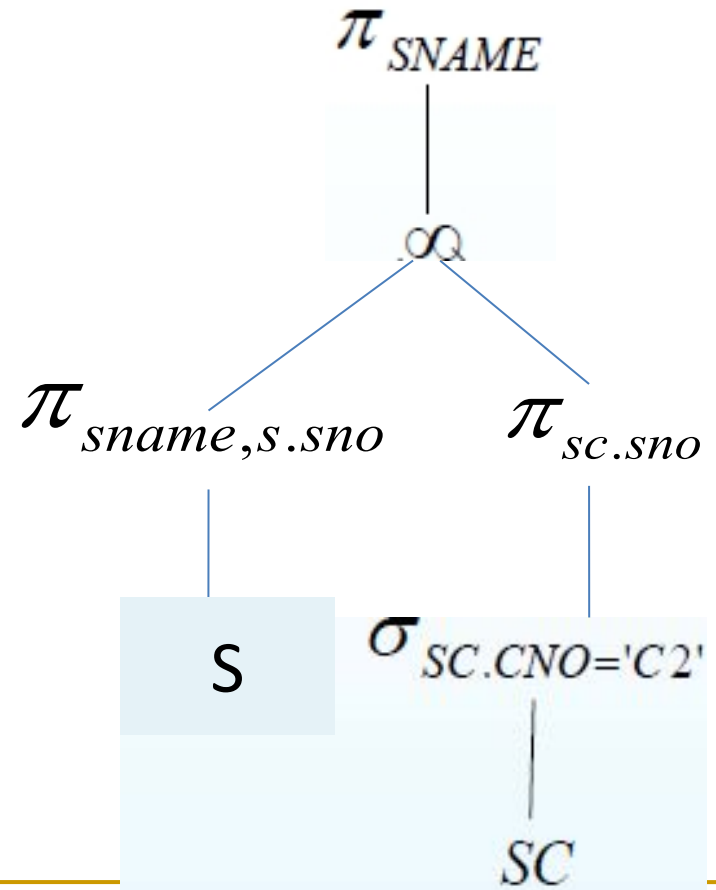
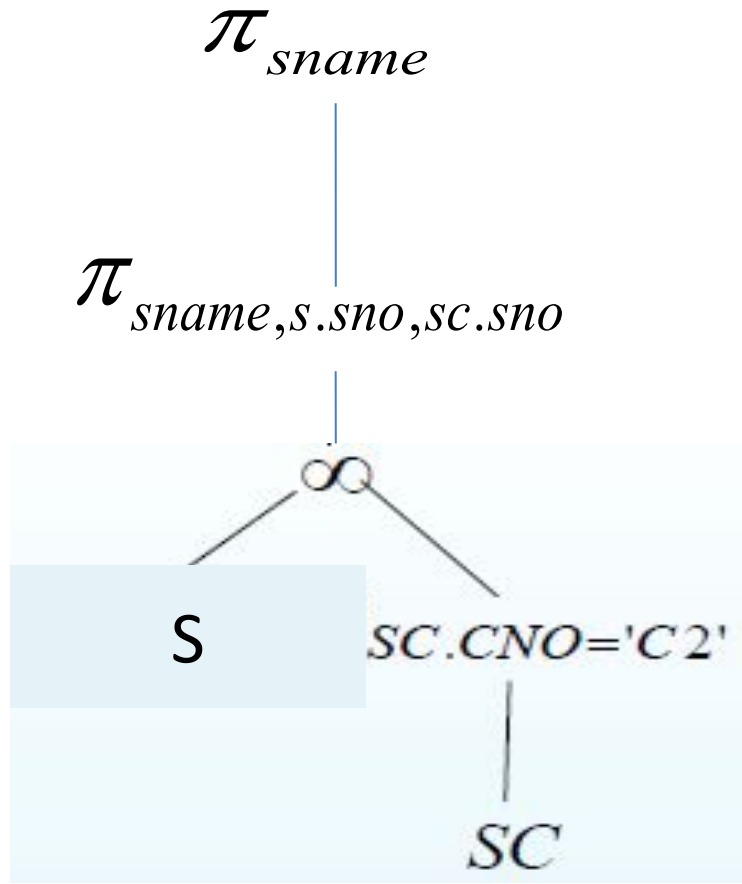
S

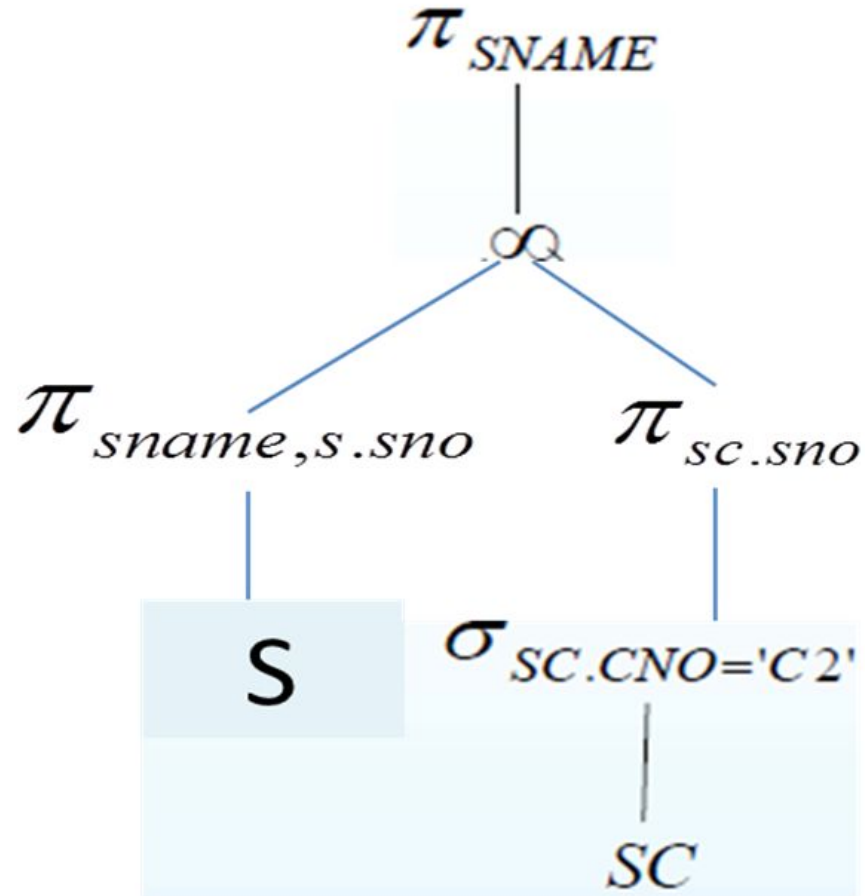
$SC.CNO='C2'$

SC



把投影运算可能移到查询树的下面，如有必要，引入新的投影运算。





$$\pi_{sname}(\pi_{sname,s.sno}(S) \Join \pi_{sc.sno}(\sigma_{cno='c2'}(SC)))$$

# **第十一章 关系查询处理和查询优化**

## **11.1 关系数据库系统的查询处理**

## **11.2 关系数据库系统的查询优化**

## **11.3 代数优化**

## **11.4 物理优化**

## **11.5 小结**

# 第十一章 关系查询处理和查询优化

## 11.1 关系数据库系统的查询处理

## 11.2 关系数据库系统的查询优化

## 11.3 代数优化

## 11.4 物理优化

## 11.5 小结

## 11.4 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**

## 11.4 物理优化

- **选择的方法：**
  - **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**

## 11.4 物理优化-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:

**1. 对于小关系，使用全表顺序扫描，即使选择列上有索引**

对于大关系，启发式规则有：

**2. 对于选择条件是主码 = 值的查询**

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

## 11.4 物理优化-基于启发式规则的存取路径选择优化

### 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



## 11.4 物理优化-基于启发式规则的存取路径选择优化

### 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

## **11.4 物理优化-基于启发式规则的存取路径选择优化**

**5. 对于用OR连接的析取选择条件，一般使用全表顺序扫描**

## 11.4 物理优化-基于启发式规则的存取路径选择优化

### 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

## 11.4 物理优化-基于启发式规则的存取路径选择优化

4. 最后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)。

理由：

- 设连接表R与S分别占用的块数为 $B_r$ 与 $B_s$
  - 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 $B_r + (B_r/K - 1)B_s$
- 
- 显然应该选块数小的表作为外表

## 11.4 物理优化-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**
  - *可以采用精细复杂一些的基于代价的优化方法*

## 11.4 物理优化-基于代价的优化

- 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
- 数据字典中存储的优化器需要的统计信息：
  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( $BO$ )

## 11.4 物理优化-基于代价的优化

### 2. 对基表的每个列

- 该列不同值的个数( $m$ )
- 选择率( $f$ )
- 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)

## 11.4 物理优化-基于代价的优化

### 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)



## 11.5 小 结

- **查询处理是RDBMS的核心，查询优化技术是查询处理的关键技术**
- **本章讲解的优化方法**
  - **启发式代数优化**
  - **基于规则的存取路径优化**
  - **基于代价的优化**
- **本章的目的：掌握查询优化方法的概念和技术**

## 小 结（续）

- **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**