第5章 查询处理和查询优化

北京理工大学 计算机学院
张文耀
zhwenyao@bit.edu.cn

主要内容

- 查询概述
- 查询处理过程
- 关系操作的基本实现算法
- 查询优化技术
- 代数优化
- 基于存取路径的优化
- 基于代价估算的优化

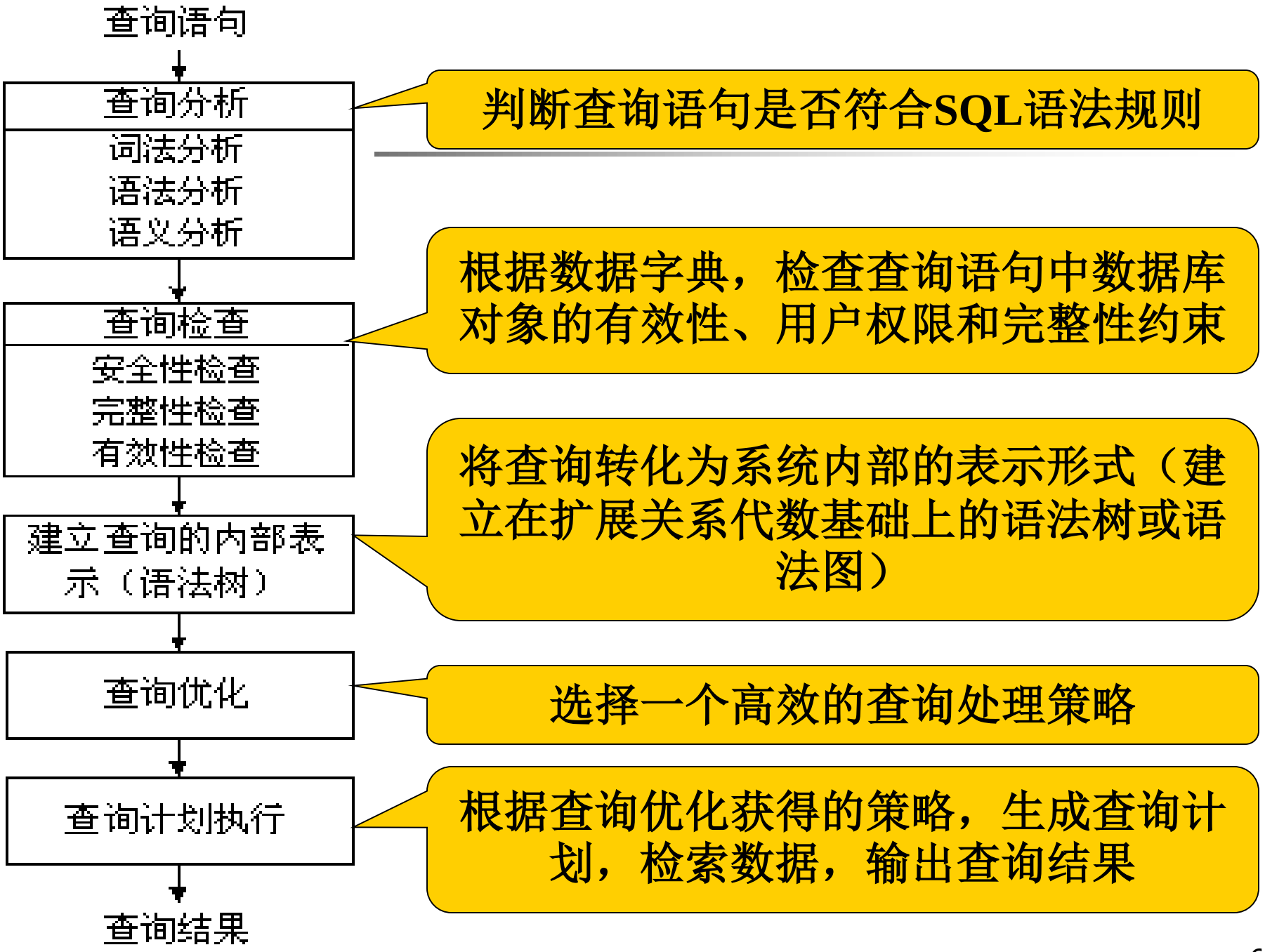
1.查询概述

- 查询是数据库管理系统中使用最频繁、最基本的操作，对系统性能有很大影响。
- 对于同一个**SQL**查询，通常可以有多个等价的关系代数表达式。
- 由于存取路径的不同，每个关系代数表达式的查询代价和效率也是不同的。
- 为了提高查询效率，需要针对具体查询请求，建立一个相对高效的查询计划，即进行查询优化。
- 查询优化技术是关系数据库的关键技术。
- **DBMS**的查询优化器包括了很多查询优化技术。

- 查询语句的执行方式
  - 解释方式
    - **DBMS**不保留可执行代码，每一次都重新解释执行查询语句，事务完成后返回查询结果。
    - 具有灵活、应变性强的特点，但是开销比较大、效率比较低。
    - 主要适用于不重复使用的偶然查询。
  - 编译方式
    - 先进行编译处理，生成可执行代码。运行时，直接执行可执行代码。
    - 当数据库中某些数据发生改变，再重新编译。
    - 主要优点是执行效率高、系统开销小。

2.查询处理过程

- 查询处理是指从数据库中提取数据所涉及的处理过程，包括将用户提交的查询语句转变为数据库的查询计划，并执行这个查询计划得到查询结果。
- 关系数据库的查询处理分为五个阶段
  - 查询分析
  - 查询检查
  - 建立查询的内部表示
  - 查询优化
  - 查询执行


查询语句
查询分析
词法分析
语法分析
语义分析
判断查询语句是否符合SQL语法规则
查询检查
安全性检查
完整性检查
有效性检查
根据数据字典，检查查询语句中数据库对象的有效性、用户权限和完整性约束
建立查询的内部表示（语法树）
将查询转化为系统内部的表示形式（建立在扩展关系代数基础上的语法树或语法图）
查询优化
选择一个高效的查询处理策略
查询计划执行
根据查询优化获得的策略，生成查询计划，检索数据，输出查询结果
查询结果

3.关系操作的基本实现算法

- 为了实现查询处理，需要选择合适的算法实现基本的关系操作

  1）选择操作的实现

  2）连接操作的实现

  3）投影操作的实现

  4）集合运算的实现

1）选择操作的实现

- 选择操作的实现方法有：
  - 顺序扫描法
    - 按照关系中元组的物理顺序扫描每个元组，检查该元组是否满足选择条件，如果满足则输出。
    - 实现简单，不需要特殊的存取路径。
    - 适用于任何关系，尤其适用于被选中的元组数占有较大比例或总的元组数较少的关系。
    - 简单，通用，效率低
  - 二分查找法
    适应于等值比较，高效，
    但要求物理文件按选择字段有序组织。

- 索引扫描法（ 散列扫描法）
  - 要求选择条件涉及的属性上有索引(例如B+树索引或Hash索引)
  - 通过索引先找到满足条件的元组指针，然后通过该指针继续检索满足查询条件的元组。
  - 索引在一定程度上能提高查询效率。

- 【例】

  Select * from student where <条件表达式>,

  其中条件表达式可以有以下几种情况：

  C1：无条件

  C2：Sno = ‘200636’

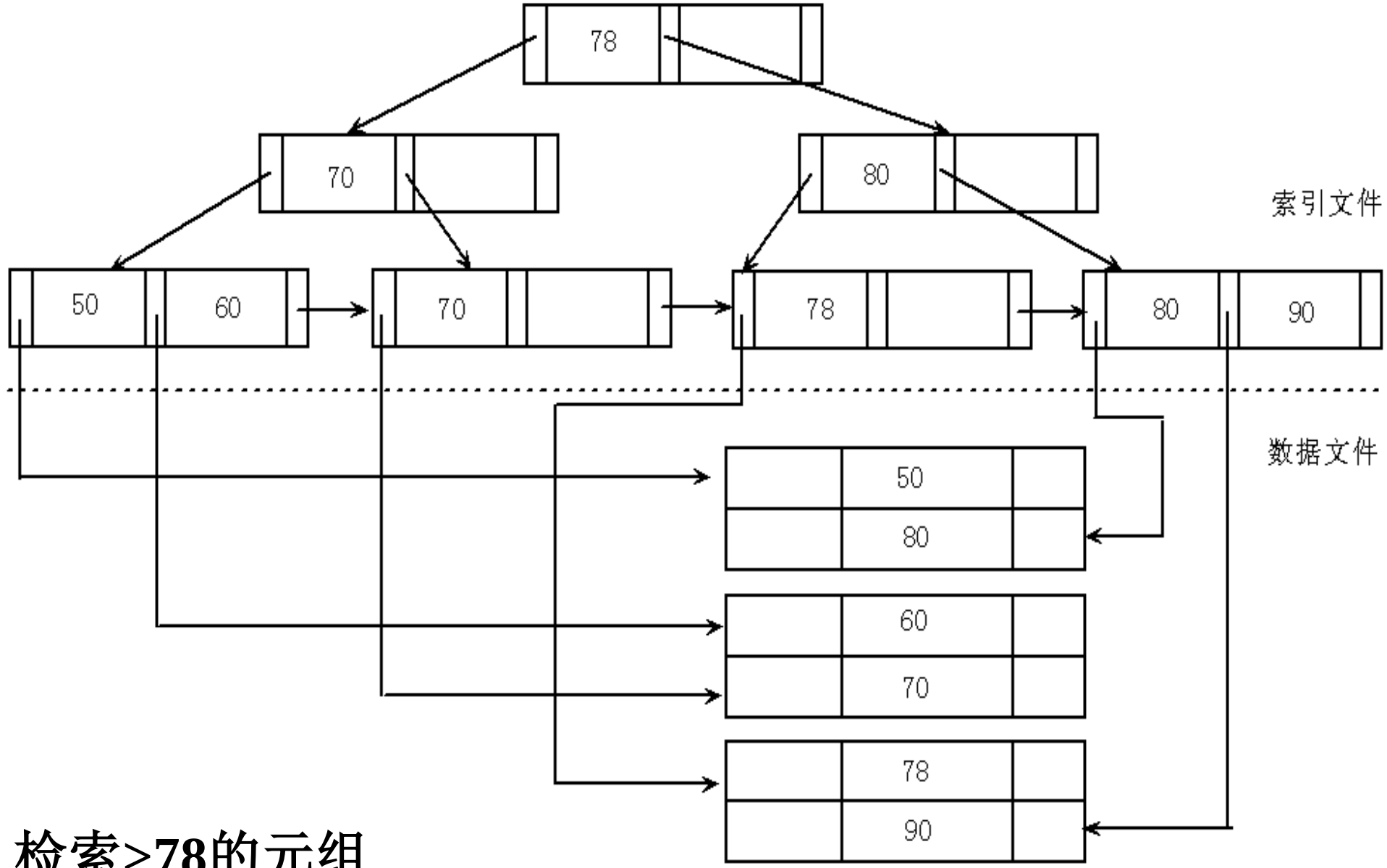
  C3：Sage > 18

  C4：Sdept = ‘计算机’ and Sno = ‘200636’

  C5：Sdept = ‘计算机’ and Sage > 18

  C6：Sage > 18 or Sdept = ‘计算机’

- **C1**：无条件
  顺序扫描法
- **C2**：**Sno = ‘200636’**
  - 字段**Sno**是排序属性，可以用二分查找法实现**C2**中的选择条件
  - 如果是非排序属性，可用顺序扫描法
- **C3**：**Sage > 78**
  - 如果**Sage**上有**B+**树索引，可以使用**B+**树索引找到**Sage=78**的索引项，以此为入口点找到**B+**树的顺序集上所有**Sage>78**的元组指针，进而找到相应元组。



检索>78的元组

B＋树索引示例

复合选择的情况

C4：Sdept = ‘计算机’ and Sno = ‘200636’

C5：Sdept = ‘计算机’ and Sage > 18

C6：Sage > 18 or Sdept = ‘计算机’

- 合取选择条件的实现
  - 使用组合索引实现
    如果在（**Sdept，Sno**）上建立了组合索引，可以通过这个组合索引直接找到满足条件**C4**的元组。
  - 使用单独索引实现
    如果某个属性具有索引，先通过索引找到符合该属性条件的元组，然后进一步判断是否满足其余属性条件。
    例如**C5**，先找到**Sdept = ‘计算机’** 的元组，在判断**Sage > 18**是否成立

- 使用多个索引实现
  - 如果在多个检索属性上存在索引，可以分别检索满足条件的元组，再求交集。
  - 如果还有非索引属性的检索条件，则需进一步判断。
  - 例：**C5: Sdept = ‘计算机’ and Sage > 18**，假设**Sdept**和**Sage**上都有索引。
    - 分别使用索引找到**Sdept=**‘计算机’的一组元组指针和**Sage>18**的另一组元组指针
    - 然后求这两组指针的交集，再到**student**表中检索元组

- 合取选择条件的选择
  - 首先选择选择性小的条件检索元组
  - 选择性（**Selectivity**）：满足条件的元组数占关系中元组总数的比例。
  - 一般地，如果选择条件的选择性为**s**，满足该条件的元组数则可以估计为（元组数× **s** ）。所估计的值越小，就越有必要首先使用这个条件来检索元组
  - 例：查成绩**90**分以上的男生

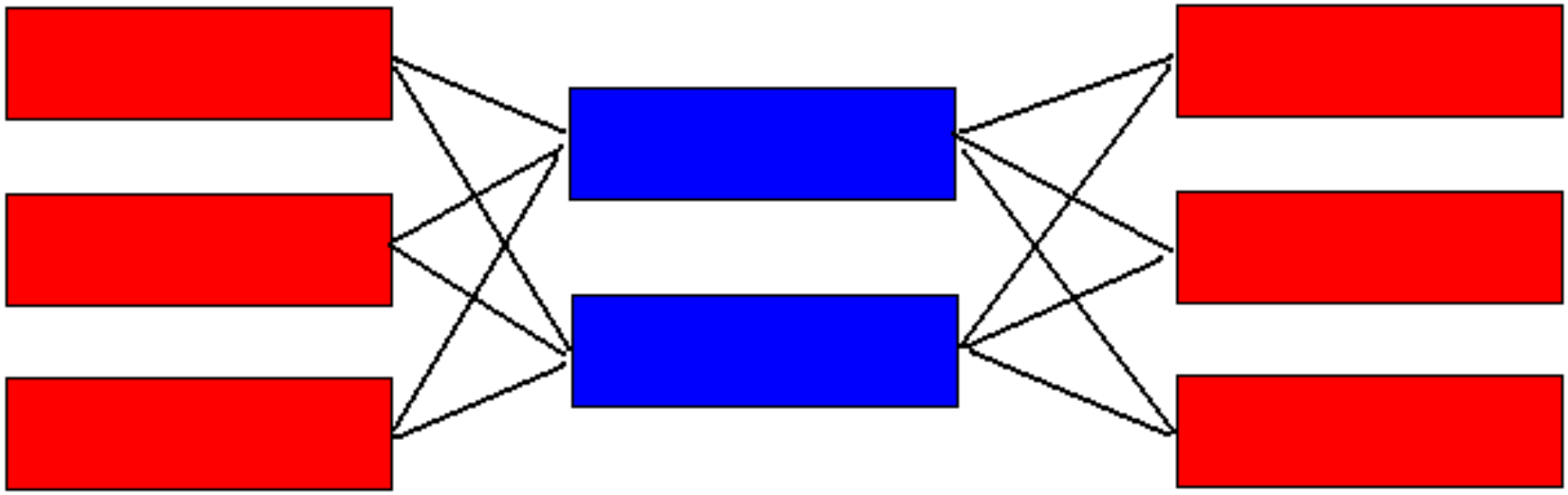
- 析取选择条件的实现
  - 优化处理难度比较大，
    例如 **C6 ： Sage > 18 or Sdept = ‘计算机’**
  - 如果**Sdept**上有索引，而**Sage**上没有索引，基本上无法进行优化。
  - 只要任意一个条件没有索引，就只能使用顺序扫描方法。
  - 对于条件中涉及的属性都具有索引时，才能通过优化检索满足条件的元组，然后再通过合并操作消除重复元组。
  - 尽量避免用 **or** 。

2）连接操作的实现

- 连接操作的特点
  - 查询处理中最耗时的操作之一，
  - 操作本身开销大，
  - 可能产生很大的中间结果。
- 连接操作的实现方法
  - 嵌套循环法
  - 索引嵌套循环法
  - 排序合并法
  - 散列连接（**Hash Join**）法

连接操作的实现—嵌套循环法

- 基本思想：对于关系**R**（外循环）中的每个元组**t**，检索关系**S**（内循环）中的每个元组**s**，判断这两个元组是否满足连接条件**t[A]=s[B]**。如果满足条件，则连接这两个元组作为输出结果。
- 嵌套循环法是最简单、最直接的连接算法，与选择操作中的顺序扫描法类似，不需要特别的存取路径。
- 适用于任何条件的连接。
- 选择哪一个关系用于外循环、哪一个关系用于内循环会给连接的性能带来比较大的差异。 一般使用较少块的文件作为外循环文件，连接代价较小。



3R+6S

3R+6S−2

2S+6R

2S+6R−1

连接操作的实现—索引嵌套循环法

- 在嵌套循环法中，如果两个连接属性中的一个属性上存在索引（或散列），可以使用索引扫描代替顺序扫描。
  - 例如，在关系**S**中的属性**B**上存在索引，则对于**R**中的每个元组，可以通过**S**的索引查找满足**t[B]＝s[A]**的所有元组，而不必扫描**S**中的所有元组，以减少扫描时间。
- 在一般情况下，索引嵌套循环法和嵌套循环法相比，查询的代价可以降低很多。

连接操作的实现—排序合并法

- 适用于等值连接和自然连接
- 前提条件：关系**R**和**S**分别按照连接属性**A**和**B**排序
- 处理过程：按照连接属性同时对关系**R**和**S**进行扫描，匹配**A**和**B**上有相同值的元组，连接起来，作为结果输出。
- 特点：两个关系各扫描一次，而且是同步进行的。
- 示例：下页的等值连接

- 等值连接示例

R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

S

B	E
b1	3
b2	7
b3	10
b3	2
b5	2

$R \bowtie S$

R.B=S.B

A	R.B	C	S.B	E
a_1	b_1	5	b_1	3
a_1	b_2	6	b_2	7
a_2	b_3	8	b_3	10
a_2	b_3	8	b_3	2

- 如果**R**和**S**没有排序，可以进行排序处理，再进行连接操作。
- 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，后使用排序合并法执行连接，总的时间一般仍会大量减少。

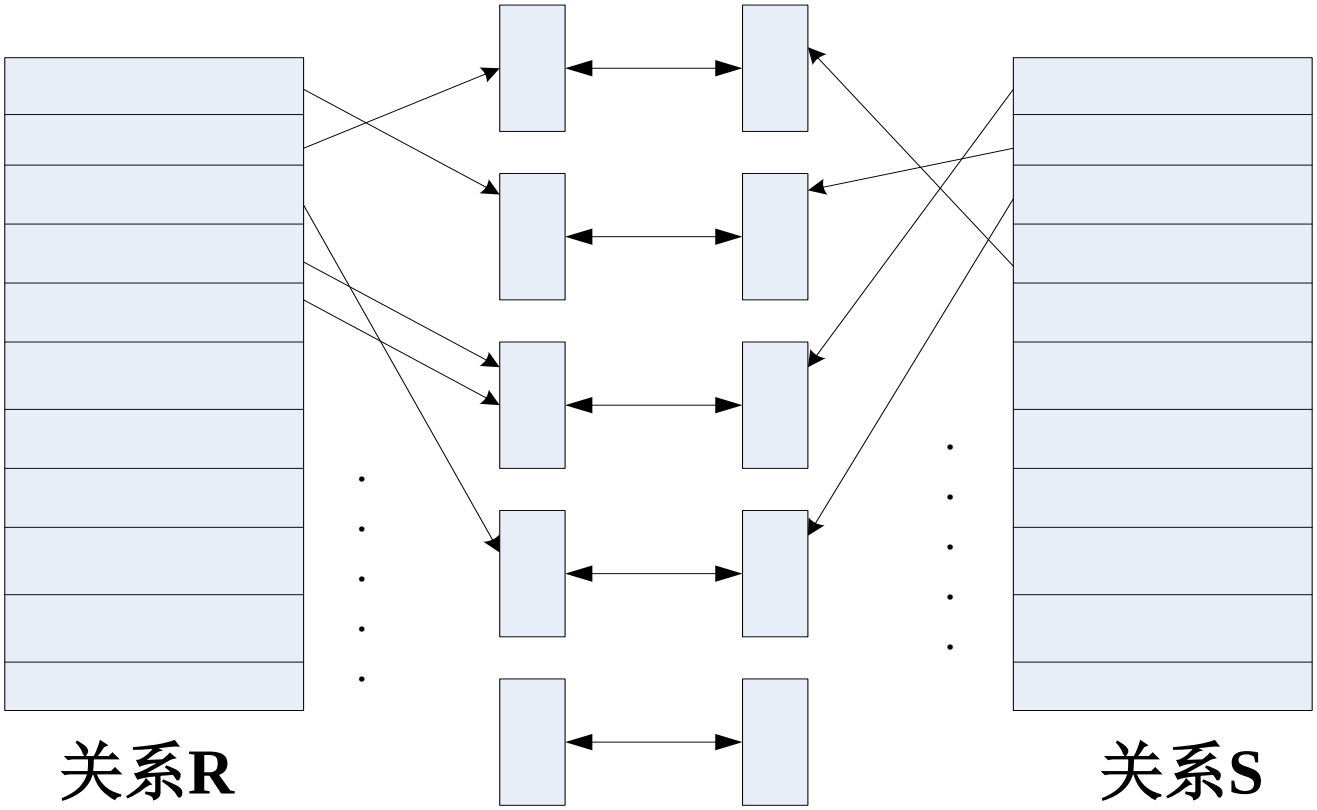
注意：这里的排序不同于索引。

连接操作的实现—散列(Hash)连接法

- 类似于排序合并法，两个关系只需各扫描一次，适用于等值连接和自然连接。
- 基本思想：在关系**R**和**S**的连接属性上，使用同一个**hash**函数，将关系**R**和**S**划分成多个具有相同散列值的元组的集合；然后在各个元组集合内进行连接操作。
- 基本过程
  - 划分阶段：将元组较少的关系散列到**hash**文件桶中
  - 试探（连接）阶段：将另一个关系的元组散列到相应的**hash**桶中，并与该桶中原有关系的元组进行匹配结合。
  - 匹配时可以采用嵌套循环处理的方法。

散列法的依据：如果属性值相等，散列值必定相等；

散列值相等，属性值未必相等。



散列法的适用条件：

划分阶段形成的散列桶最好可以完全存放在内存中。

3）投影操作的实现

- 投影操作选取关系的某些列，从垂直的方向减小关系的大小。
  - 如果投影属性列包括了关系**R**的主键，那么操作可以直接执行，操作的结果将与**R**中的元组个数相同
  - 否则，需要消除重复元组
    - 通常做法：先排序操作结果，再去掉多余的副本
    - 用散列法消除重复元组，
      - 把投影结果的每个元组散列到相应的散列桶中；
      - 然后检查是否与该桶中已存在的元组重复，如果该重复，则不把这个元组插入桶中。

4）集合运算的实现

- 传统的集合运算是二元的，包括并、差、交、笛卡尔积四种运算。
  - 并、差、交运算实现的常用方法类似排序合并法
    - 首先对参加运算的两个关系分别按照主键属性排序；
    - 排序后只需同时对两个关系执行一次扫描就可以生成计算结果。
  - 笛卡尔积的实现通常使用嵌套循环法
    - 由于笛卡尔积的操作结果中包含了**R**和**S**中每个元组的组合，其结果集会比参与运算的关系大得多，操作代价非常高。

4.查询优化技术

- 关系数据库系统的查询优化
  - 对于一个具体的查询请求，可以用几种不同形式的查询语句来表达，而不同的表达会使数据库的响应速度大不相同。因此，在实际应用中需要利用查询优化技术对不同形式的查询语句进行分析，并选择高效合理的查询语句，使执行查询所需要的系统开销降至最低，从而提高数据库系统的性能。
  - 从查询的多个执行策略中选择合适的执行策略来执行，这个选择的过程就是查询优化。
- 查询优化对关系数据库管理系统的性能至关重要。

查询优化示例

- 【例 】查询选修“DataBase”课程的学生成绩。用SQL表达如下

```
SELECT  SC.Grade
    FROM  Course，SC
    WHERE Course.Cno=SC.Cno
                and Course.Cname='DataBase';
```

怎么实现？

$\Pi_{Grade}(\sigma_{Course.Cno=SC.Cno \wedge Course.Cname='DataBase'}(Course \times SC))$

$\Pi_{Grade}(\sigma_{Course.Cname='DataBase'}(Course \bowtie SC))$

$\Pi_{Grade}(SC \bowtie \sigma_{Course.Cname='DataBase'}(Course))$

- 哪种策略最佳？
- 假设：
  - **SC:10000条，Course:100条，满足条件的元组为100个。**
  - 内存被划分为**6**块，每块能装**10**个**Course**元组或**100**个**SC**元组。每次在内存中放**5**块**Course**元组和**1**块**SC**元组。

$\Pi_{Grade}(\sigma_{Course.Cno=SC.Cno \wedge Course.Cname='DataBase'}(Course \times SC))$

- **Course×SC**

1）在内存中尽可能多地装入**Course**表，留出一块存放**SC**的元组。

2）把**SC**中的每个元组和**Course**中的每个元组连接，完成之后，继续读入下一块**SC**的元组，同样和内存中**Course**的每个元组连接，依此类推，直到**SC**表的元组全部处理完毕。

3）把**Course**表中没有装入的元组尽可能多地装入内存，同样逐块装入**SC**表的元组去做元组的连接，直到**Course**表的所有元组全部进行完连接。

- 读取的总块数
  =读Course表的块数+ 读SC表遍数*每遍块数
  =100/10+100/(10*5)*10000/100
  =10+200=210
  读数据的时间=210/20=10.5s （假设每秒读写20块）
- 中间结果的元组数 = 10000*100 = 10^6
  写中间结果的时间=10^5/20=5000s（每块10个元组）

- $\sigma_{Course.Cno=SC.Cno \wedge Course.Cname='DataBase'}$
  将已经连接好的10^6个元组重新读入内存，按照选择条件选取满足条件的元组。满足条件的元组为100个。

- 假设结果全部存放在内存中，忽略内存处理时间。
- 选择操作的时间开销为：
  读数据的时间=5000秒 （与写文件一样）

- Π_{Grade}
  仍为100个元组，可以放在内存中，不需要作I/O操作，忽略内存处理时间。因此这一步操作时间可以忽略。
- 总时间代价
  =10.5＋5000＋5000 ≈ 2.78 （小时）

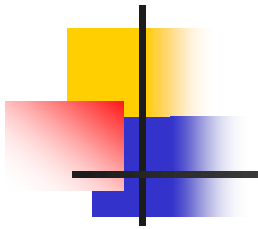
$\Pi_{Grade}(\sigma_{Course.Cname='DataBase'}(Course \bowtie SC))$

- **Course⋈ SC**
  - 读取数据块 210块，读数据的时间=210/20=10.5秒
  - 中间结果元组数=10000 （SC为10000条）
    写中间结果的时间=10000/10/20=50秒
- $\sigma_{Course.Cname='DataBase'}$
  读取上一步已连接好的10000个元组，检查是否满足选择条件，产生一个100个元组的结果集。
  - 读数据的时间= 10000 /20 =50秒
- Π_{Grade}: 不需要作I/O操作
- 总时间＝10.5＋50＋50秒＝110.5秒

$\Pi_{Grade}(SC \bowtie \sigma_{Course.Cname='DataBase'}(Course))$

- $\sigma_{Course.Cname='DataBase'}(Course)$
  先装入Course表元组，再选择满足条件的元组，结果集为一个元组，不必写入外存
  - 读Course表的总块数= 100/10=10块
    读数据的时间=10/20=0.5秒
- $\bowtie$

  将10000个SC的元组依次读入内存，和内存中的唯一1个Course元组作自然连接。只需读一遍SC表。
  - SC表的总块数= 10000/100=100块
    读数据的时间=100/20=5秒
- Π_{Grade}：不需要作I/O操作
- 总时间＝0.5＋5秒＝5.5秒



$\Pi_{Grade}(\sigma_{Course.Cno=SC.Cno \wedge Course.Cname='DataBase'}(Course \times SC))$

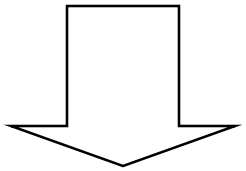
$\Pi_{Grade}(\sigma_{Course.Cname='DataBase'}(Course \bowtie SC))$

$\Pi_{Grade}(SC \bowtie \sigma_{Course.Cname='DataBase'}(Course))$

- 哪种策略最佳？
  - **2.78小时**
  - **110.5秒**
  - **5.5秒**

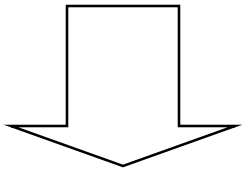
- 原因

$$\Pi_{Grade}(\sigma_{Course.Cno=SC.Cno \wedge Course.Cname='DataBase'}(Course \times SC))$$



选择条件Course.Cno = SC.Cno
与笛卡尔积组合成连接操作

$$\Pi_{Grade}(\sigma_{Course.Cname='DataBase'}(Course \bowtie SC))$$



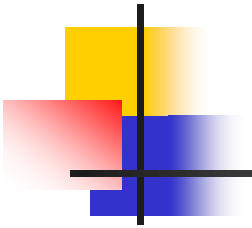
条件Course.Cname = ‘DataBase’
移到连接操作中的关系Course中

$$\Pi_{Grade}(SC \bowtie \sigma_{Course.Cname='DataBase'}(Course))$$

每次变换都使参加运算的元组大大减少！

查询优化的必要性

- 前面的优化实例表明：对于等价的关系代数表达式，其相应的查询效率有着巨大差异，因此需要进行查询优化，通过合理的优化，选择合适的查询计划，以获得较高的查询效率。
- 在关系数据库系统中，由于关系数据库语言的非过程特性，用户和应用程序一般不考虑如何最好地表达查询、如何最好地实现查询。因此，DBMS必须有专门的查询优化模块负责对查询的优化处理。



- 由DBMS进行查询优化的好处
  - 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
  - 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。
  - 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。
  - 优化器中可以包括很多复杂的优化技术。

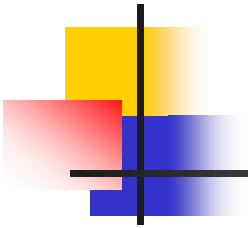
查询优化的目标

- 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
    - 总代价=磁盘存取块数(I/O代价)＋处理机时间(CPU代价)＋查询内存开销（内存代价）
    - 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
    - 通信代价是最主要的

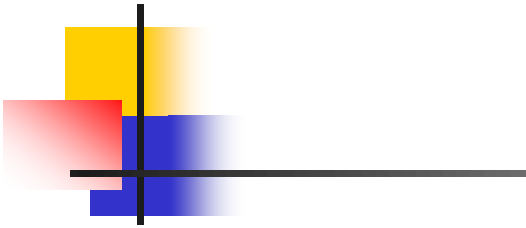
查询优化技术的分类

- 按优化层次分为：
  - 代数优化
    - 是基于关系代数表达式的优化；
    - 按照一定的规则，改变代数表达式中操作的次序和组合，使查询执行更高效；
    - 只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；
    - 是通过关系代数表达式的等价变换规则来完成优化，也称为规则优化（逻辑优化）。

- 基于存取路径的优化
  - 合理选择各种操作的存取路径以获得优化效果；
  - 需要考虑数据的物理组织和访问路径，以及底层操作算法的选择；
  - 涉及数据文件的组织方法、数据值的分布情况等，也称为物理优化。



- 按选择依据分：
  - 基于规则的优化
    - 启发式规则
  - 基于代价估算的优化
    - 对于多个可选的查询策略，通过估算执行策略的代价，从中选择代价最小的作为执行策略。
    - 由于需要计算操作执行的代价，优化本身所需要的开销较大。



- 按优化阶段分：
  - 静态查询优化
    - 在查询执行前对**SQL**语句进行等价变换；
    - 应用启发式规则进行优化。
  - 动态查询优化
    - 运行时根据实际数据调整执行计划；
    - 基于代价估算的优化，根据统计信息估算执行代价

查询优化的一般过程

1. 将查询转换成某种内部表示，通常是语法树；
2. 根据一定的等价变换规则，把语法树转换成标准（优化）形式，即对语法树进行优化；
   （代数优化）
3. 选择低层的操作算法；
   对于语法树中的每一个操作，计算各种执行算法的执行代价，选择代价小的执行算法。
   （物理优化）
4. 生成查询计划，选择代价最小的。
   查询计划（查询执行方案）是由一系列内部操作组成的。

5.代数优化

- 代数优化
  基于关系代数等价变换规则的优化
- 主要依据：关系代数表达式的等价变换规则
- 主要思想：对查询的代数表达式进行适当的等价变换，改变相关操作的先后执行顺序，把初始查询树转换为优化后的查询树，完成代数表达式的优化
- 与具体系统的存储技术无关

- 关系代数表达式的等价
  - 是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**E1**和**E2**是等价的，可记为**E1 ≡ E2**
- 关系代数中常用的等价变换规则

  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

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

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

$(E1 \bowtie_{F1} E2) \bowtie_{F2} E3 \equiv E1 \bowtie_{F1} (E2 \bowtie_{F2} E3)$

3. 投影的串接定律

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

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

4. 选择的串接定律

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

- 选择的串接定律说明：多个选择操作可以合并为一个
- 一次检查全部条件

5. 选择与投影的交换律

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

- 选择条件F只涉及属性$A_1, \ldots, A_n$

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

- F中有不属于$A_1, \ldots, A_n$ 的属性$B_1,\ldots,B_m$

6. 选择与笛卡尔积的交换律

- F中涉及的属性都是E1中的属性

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

- $F=F_1 \wedge F_2$，并且F_1只涉及E1中的属性，F_2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：

  $\sigma_F(E1 \times E2) \equiv \sigma_{F_1}(E1) \times \sigma_{F_2}(E2)$

- $F= F_1 \wedge F_2$， F_1只涉及E1中的属性， F_2涉及E1和E2两者的属性

  $\sigma_F(E1 \times E2) \equiv \sigma_{F_2}(\sigma_{F_1}(E1) \times E2)$

  它使部分选择在笛卡尔积前先做

7. 选择与并的分配律

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

- **E=E1∪E2，E1，E2**有相同的属性名

8. 选择与差运算的分配律

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

- **E1**与**E2**有相同的属性名

9. 选择对自然连接的分配律

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

- **F**只涉及**E1**与**E2**的公共属性

10. 投影与笛卡尔积的分配律

$$\Pi_{A_1,A_2,\ldots,A_n,\ B_1,B_2,\ldots,B_m}(E1 \times E2) \equiv \Pi_{A_1,A_2,\ldots,A_n}(E1) \times \Pi_{B_1,B_2,\ldots,B_m}(E2)$$

- $A_1, \ldots, A_n$是E1的属性，$B_1, \ldots, B_m$是E2的属性

11. 投影与并的分配律

$$\Pi_{A_1,A_2,\ldots,A_n}(E1 \cup E2) \equiv \Pi_{A_1,A_2,\ldots,A_n}(E1) \cup \Pi_{A_1,A_2,\ldots,A_n}(E2)$$

- E1和E2 有相同的属性名

代数优化策略

- 基本原则：减少查询处理的中间结果的大小。
- 典型的启发式规则
  - 1）选择操作尽可能早地执行
    - 目的：减小中间关系
    - 在优化策略中这是最重要、最基本的一条
  - 2）投影运算和选择运算尽量同时进行
    - 目的：避免重复扫描关系
    - 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

- 3）将投影操作与其前或其后的二元操作结合起来同时进行，以减少扫描关系的遍数。
- 4）把某些选择同在它前面的笛卡尔积结合起来，合并为一个连接操作，可以节省时间和空间开销。
- 5）找出公共子表达式，计算一次公共子表达式并把结果写入中间文件，以达到节省时间的目的。
  - 如果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，否则不一定有利。
  - 当查询涉及视图时，定义视图的表达式就是公共子表达式的典型情况。

代数优化算法

- **DBMS**自动完成关系代数表达式的优化，优化前先将关系代数表达式转换为查询树。
- 输入：一个关系表达式的查询树。
- 输出：优化的查询树。
- 方法：

(1)分解选择运算

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

(2)通过交换选择运算，将其尽可能移到叶端

对每一个选择操作，利用等价变换规则**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

对每一个投影操作，利用等价变换规则**3**，**5**，**10**，**11**中的一般形式，尽可能将其移向树的叶端。

注意：

- 等价变换规则**3**会使一些投影消失；
- 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)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
- 注意：这种变换似乎违背“投影尽可能早做”的原则，但是有时这样做的效率可能更高。

(5)把某些选择同在它前面的笛卡尔积合并为连接操作，

- 节省时间和空间开销

(6)对内结点分组

将经过上述处理得到的语法树的内结点分组。

- 每个二元操作(×、⋈、∪、－)结点和它的直接祖先操作(σ、∏)合并为一组。
- 如果其后代直到叶结点全是一元操作，则也将其并入该组；但是，如果二元操作是笛卡尔积(×)，且后面的选择操作不与它构成等值连接，则选择条件要单独分为一组。

(7)生成程序

- 每组结点的计算是程序中的一步；各步的顺序是任意的，只要保证它们之间的依赖关系。

代数优化示例

【例】查询选修了“DataBase”这门课程的计算机学院的学生姓名。

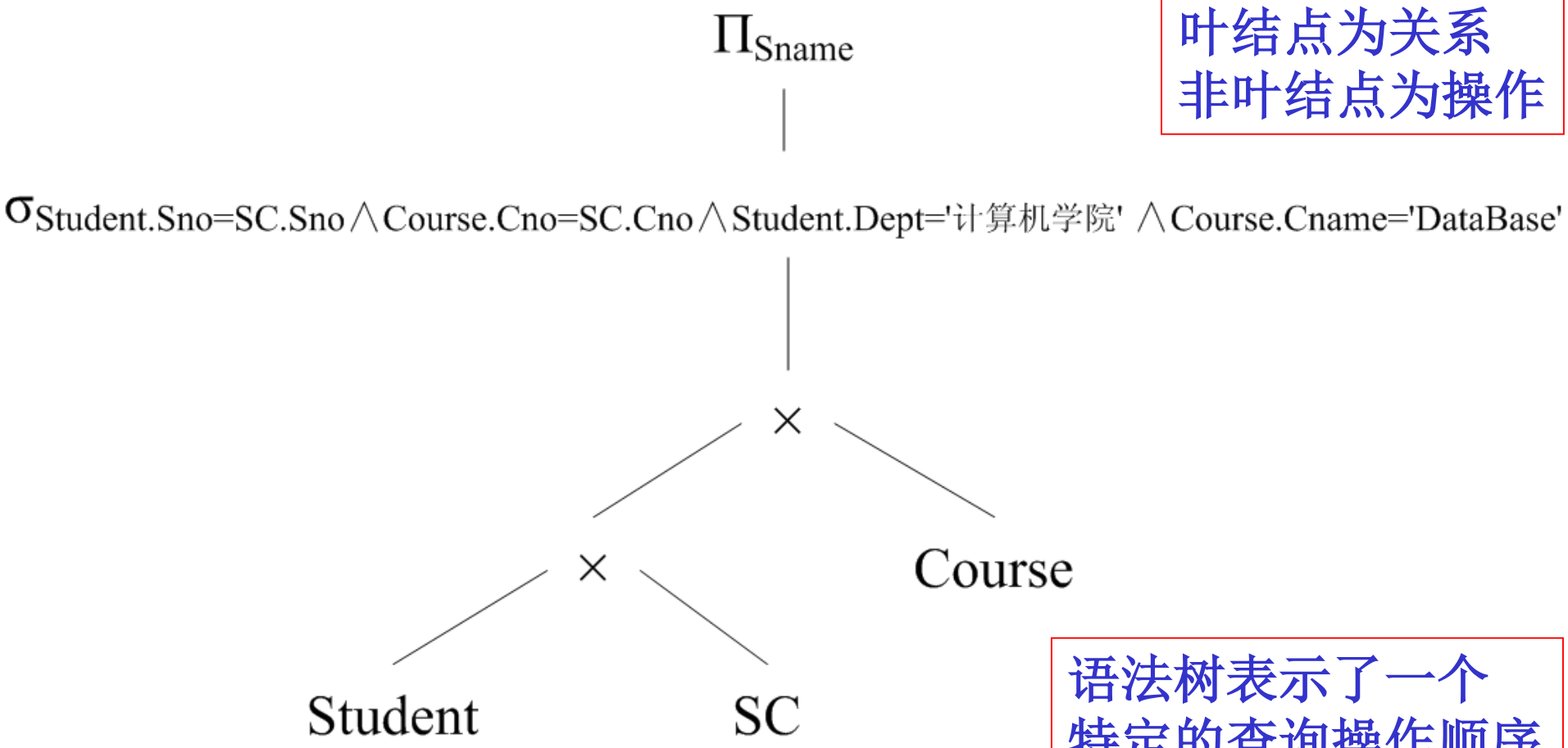
用SQL语句表达如下：

```
SELECT  Student.Sname
  FROM  Student, SC, Course
  WHERE  Student.Sno=SC.Sno and
         Course.Cno= SC.Cno  and
         Student.Dept='计算机学院' and
         Course.Cname='DataBase'
```

相应的关系代数表达式为：

$$\Pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge Course.Cno=SC.Cno \wedge Student.Dept=‘计算机学院’ \wedge Course.Cname='DataBase‘}((Student \times SC) \times Course))$$

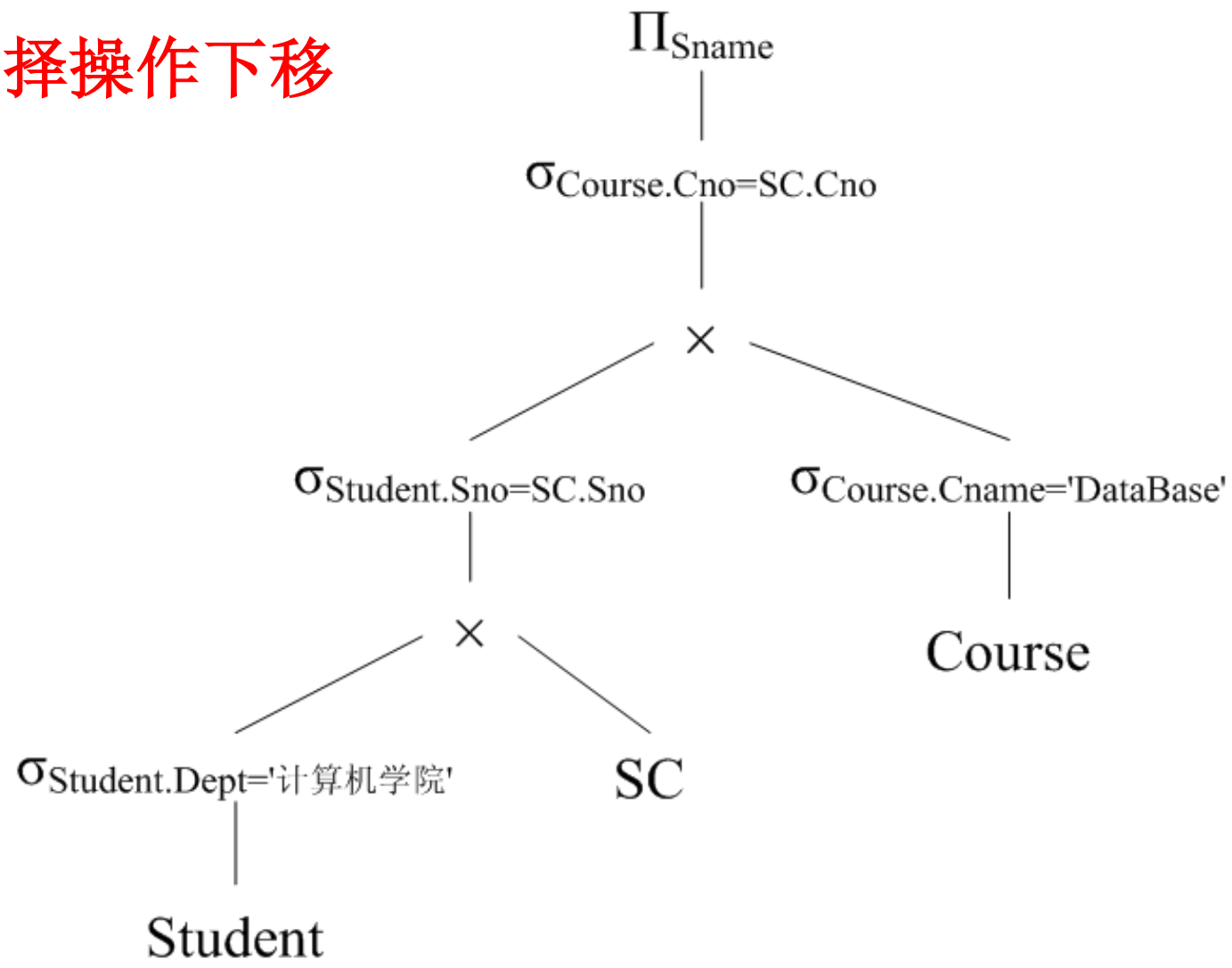
关系代数表达式的语法树为：



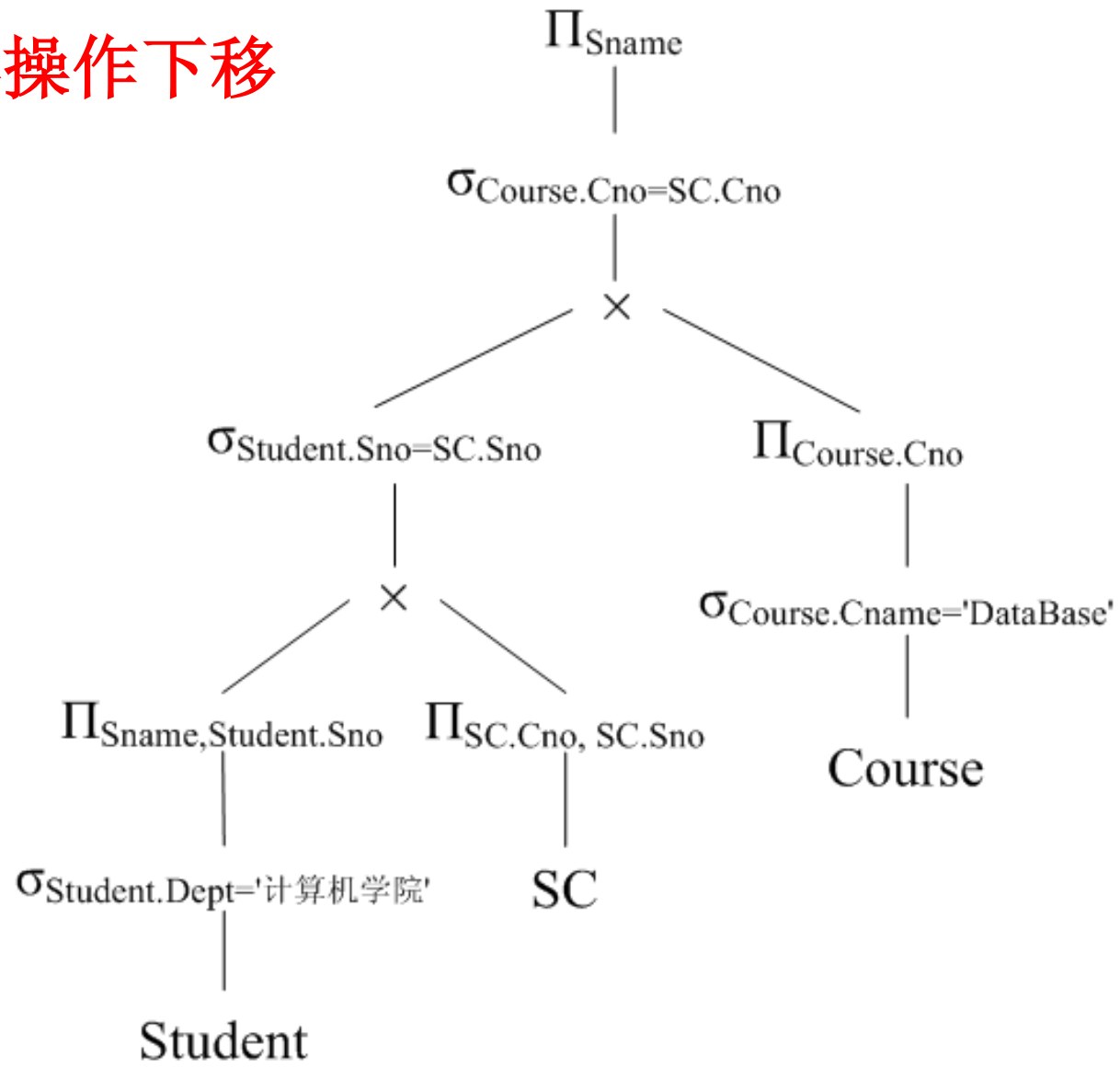
叶结点为关系
非叶结点为操作

语法树表示了一个
特定的查询操作顺序

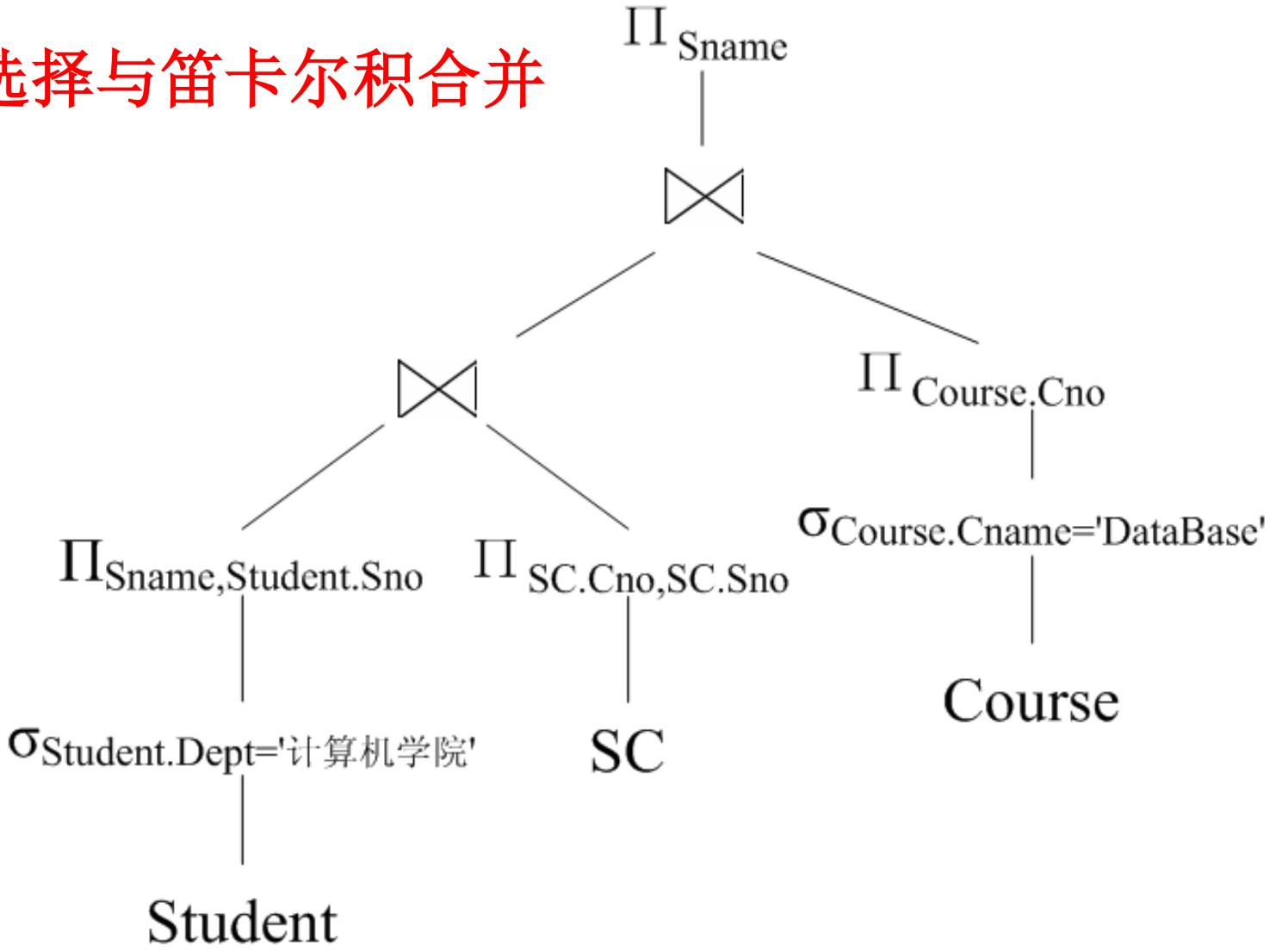
优化1：选择操作下移


Π Sname
σ Course.Cno=SC.Cno
×
σ Student.Sno=SC.Sno
σ Course.Cname='DataBase'
×
Course
σ Student.Dept='计算机学院'
SC
Student

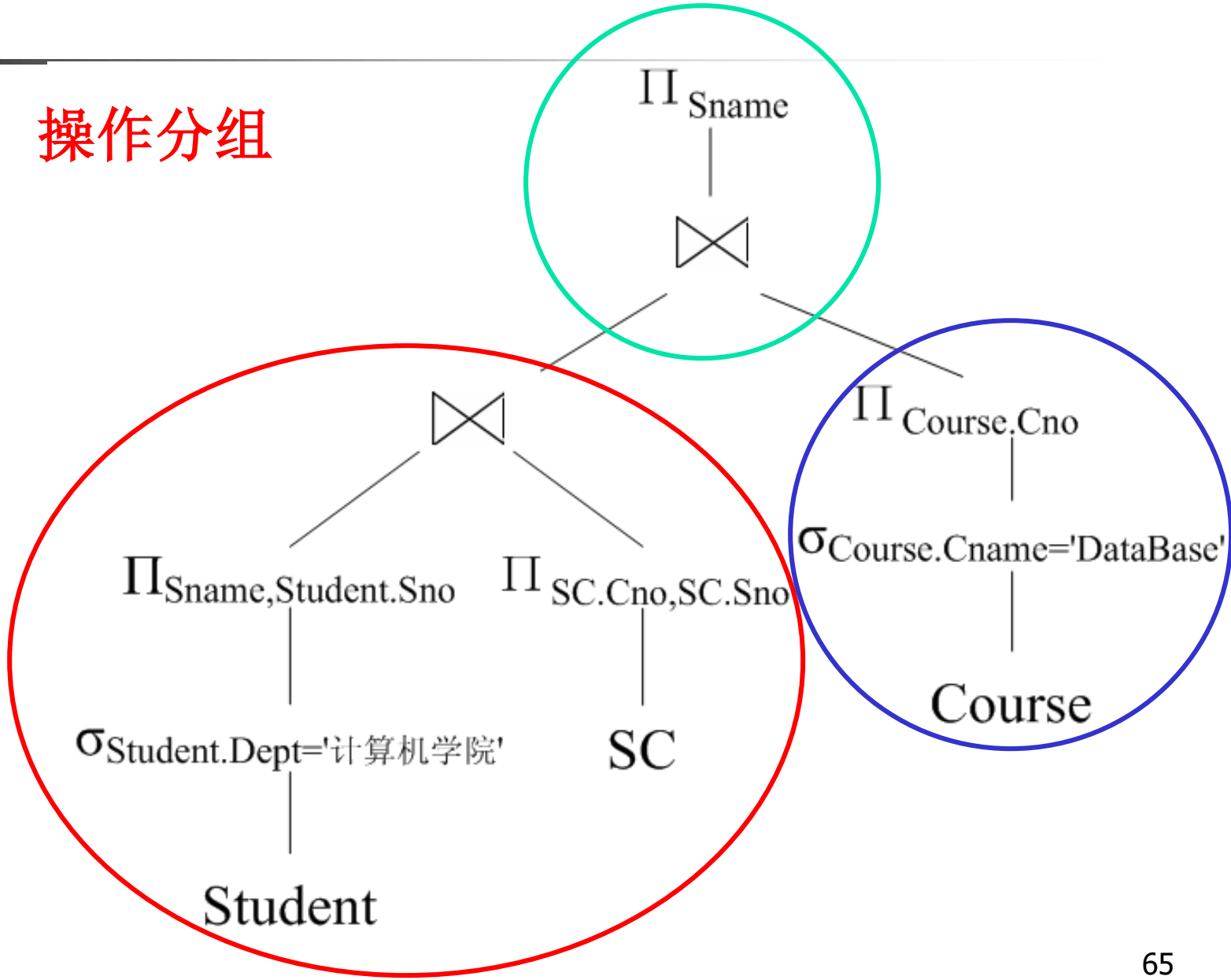
优化2：投影操作下移


Π Sname
σ Course.Cno=SC.Cno
×
σ Student.Sno=SC.Sno
Π Course.Cno
×
σ Course.Cname='DataBase'
Π Sname,Student.Sno
Π SC.Cno, SC.Sno
Course
σ Student.Dept='计算机学院'
SC
Student

优化3：选择与笛卡尔积合并



优化4：操作分组


Π Sname
⋈
⋈
Π Course.Cno
σCourse.Cname='DataBase'
Course
ΠSname,Student.Sno
Π SC.Cno,SC.Sno
σStudent.Dept='计算机学院'
SC
Student

6.基于存取路径的优化

- 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，优化效果有限。
- 在关系数据库中，数据与存取路径分离；每种操作有多种实现算法，有多种存取路径。
- 对于一个查询语句的不同存取方案，它们的执行效率会有很大的差异。
- 选择合适的存取路径，能够收到显著的优化效果。
- 基于存取路径的优化：利用启发式规则确定合适的存取路径并选择合理的操作算法。

选择操作的启发式规则

(1)对于小关系，使用顺序扫描法，即使选择列上有索引。

(2)对于选择条件是“主键=值”的查询，查询结果最多是一个元组，可以选择主键索引。

(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
  - 否则还是使用顺序扫描方法

(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，处理策略与(3)类似。

(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；如果某些属性上有一般的索引，则可以用索引扫描方法，否则使用全表顺序扫描方法。

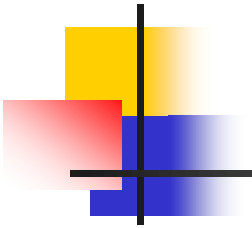
(6) 对于用OR连接的析取选择条件，一般使用全表顺序扫描方法。

连接操作的启发式规则

- 如果两个表都已经按照连接属性排序
  - 选用排序合并法
- 如果一个表在连接属性上有索引
  - 选用索引连接法
- 如果上面两个规则都不适用，其中一个表较小
  - 可以选用散列（**Hash**）连接法
- 其他情况
  - 选用嵌套循环方法，并选择占用存储块较少的表作为外循环表

7. 基于代价估算的优化

- 基于存取路径的优化是依靠启发式规则的定性优化，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
  - 定性优化
- 在编译执行的系统中，查询优化和查询执行是分开的
  - 可以采用精细的较为复杂的基于代价估算的优化方法
  - 定量优化



- 优化思想：查询优化器枚举各种可能的查询策略，估算各种策略的代价，选择代价最低的策略。
- 如何估算代价？
  分析影响查询执行代价的因素
  - 访问存储器的代价
  - 存储中间文件的代价
  - 计算代价（对内存操作的代价）
  - 内存使用代价（内存缓冲区的数目）
  - 通信代价（数据在不同结点间传送的代价）

- 代价估算的优化目标
  - 大型数据库
    使访问二级存储器（磁盘）的代价最小化
  - 小型数据库
    使计算代价最小化
  - 分布式数据库
    使通信代价最小化

集中式数据库

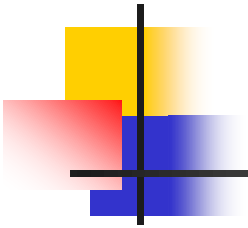
总代价= I/O代价＋CPU代价＋内存代价）

I/O代价是最主要的

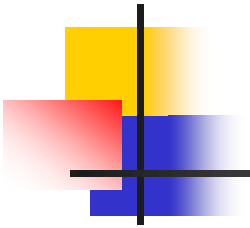
分布式数据库

总代价=I/O代价+CPU代价+内存代价＋通信代价

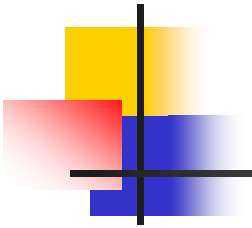
通信代价是最主要的



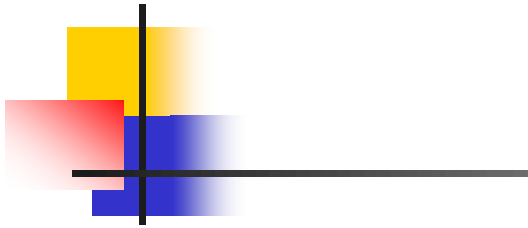
- 具体操作的代价估算
  - 与数据库中数据的状态密切相关。
  - **DBMS**会在数据字典中存储各种统计信息
- 基本表的统计信息
  - 元组数量**(N)**
  - 元组长度**(l)**
  - 占用的块数**(B)**
  - 每块可以存放的元组数目**(Mrs)**
  - 占用的溢出块数**(BO)**



- 基本表每个列的统计信息
  - 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)



- 索引的统计信息
  - 索引的层数(L)
  - 不同索引值的个数
  - 索引的选择基数(S)
    每个索引值所对应的平均元组个数
  - 索引的叶结点数(Y)



- 几种典型的查询操作实现算法的代价估算
  - 选择操作
    - 顺序扫描
    - 二分查找
    - 使用索引（或散列）的扫描
  - 连接操作
    - 嵌套循环
    - 索引嵌套循环
    - 排序合并

选择操作实现方法的代价估算

- 顺序扫描的代价估算：
  - 如果关系R的元组占用的块数为B_R（块是数据在磁盘和内存之间传递的单位），顺序扫描方法的代价 **cost=B_R**。
  - 如果选择条件是主键上的相等比较操作，那么平均搜索一半的文件块才能找到满足条件的元组，因此平均搜索代价**cost=$B_R/2$**。
- 二分查找的代价估算：
  - 二分查找法是针对文件的物理块进行的，平均搜索代价为 $\lceil \log_2 B_R \rceil$

- 索引扫描算法的代价估算：

①如果选择条件是主键属性的相等比较操作，需要存取索引树中从根结点到叶结点 **L** 个块，再加上基本表中该元组所在的那一块，所以**cost=L+1**。

② 如果选择条件涉及非主键属性的相等比较，若为**B+**树索引，如果有**S**个元组满足条件，若每个满足条件的元组可能会保存在不同块上，最坏情况下**cost=L + S**。

③如果比较条件是＞，＞＝，＜，＜＝操作，而且假设有一半的元组满足条件就要存取一半的叶结点，则代价估计

cost=L + Y/2 + B_R/2

连接操作实现方法的代价估算

- 嵌套循环法
  - 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
    $cost= B_R+ B_SB_R/(K-1)$
  - 如果需要把连接结果写回磁盘，则
    $cost=B_R+ B_SB_R/(K-1)+(Frs*N_R*N_s)/Mrs$
    - 其中N是关系的元组总数,
    - 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，即一个块中能够存放的元组数量 。

- 索引嵌套循环法
  - 若内关系S存在索引，针对外关系R的任意一个给定的元组，都可利用索引查找内关系S的相关元组
  - 最坏情况下，内存只能容纳关系R和索引各一个物理块，所需要访问的物理块数为

    $cost=B_R+N_R\times c$
    - c表示利用索引来选择满足连接条件的关系S的元组所需要的代价。
  - 在关系上建立索引进行连接操作与嵌套循环法相比，所花费的代价要降低很多。
  - 如果两个关系都有索引，一般效率较高的方法是把元组较少的关系作为外关系。

- 排序合并法
  - 如果连接表已经按照连接属性排好序，则

    cost＝$B_R + B_s + (Frs*Nr*Ns)/Mrs$
  - 如果必须对文件排序，需要加上排序的代价
    - 对于包含**B**个块的文件，两阶段归并排序代价大约是

      $(2*B) + (2*B*\log_2 B)$
  - 因此代价函数是：

    Cost＝$(2* B_R)+(2* B_R*\log2\ B_R)$ +

    $(2* B_S)+(2*B_s*\log2 B_s)$ +

    $B_R+B_S+(Frs*Nr*Ns)/Mrs$

附：基于机器学习的查询优化

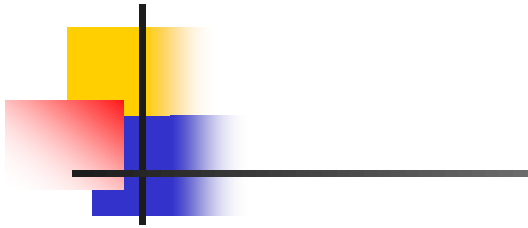
- 利用机器学习算法和技术来改进传统数据库查询优化。
- 从历史查询数据中自动学习模式和规律，从而做出更智能的优化决策。
- 主要体现在三个层面：预测（如执行时间预测）、决策（如连接顺序的选择）和自适应调整（如运行时计划修正）
- 已覆盖多个关键环节：查询代价估计、执行计划选择、查询重写、索引推荐和资源分配等。
- 在顶级数据库会议(VLDB、SIGMOD等)上相关研究成果呈现快速增长趋势。
- 不少数据库系统都已开始集成不同程度的机器学习优化功能。

表：机器学习在查询优化中的典型应用场景

应用场景	技术方法	性能提升	典型案例
索引推荐	监督学习(随机森林)	减少20-40%查询时间	MySQL智能索引优化器
查询重写	深度学习(Seq2Seq)	提升15-30%执行效率	Google Spanner重写引擎
计划选择	强化学习(PPO算法)	超越传统方法50个点	DeepRetrieval系统
代价估计	神经网络	降低30%估计误差	Microsoft SQL Server

本章小结

- 查询处理的基本过程
- 关系操作的基本实现算法
- 查询优化的含义和必要性
- 查询优化的过程
- 查询优化技术
  - 代数优化
  - 基于存取路径的优化
  - 基于代价估算的优化
  - 基于机器学习的优化



- 查询优化的基本过程
  1. 把查询转换成某种内部表示
  2. 代数优化：把语法树转换成标准（优化）形式
  3. 物理优化：选择低层的存取路径
  4. 生成查询计划，选择代价最小的

- 代数优化技术是指关系代数表达式的优化，即按照一定的规则，改变代数表达式中操作的次序和组合，使查询效率更高
- 基于存取路径的优化是指对存取路径和底层操作算法的选择和优化。
- 代价估算优化是对多个查询策略的优化选择。代价估算优化开销较大，产生所有的执行策略是不太可能的，因此将产生的执行策略的数目保持在一定范围内才是比较合理的。
  - 常常先使用启发式规则，选取若干较优的候选方案，减少代价估算的工作量；
  - 然后分别计算候选方案的执行代价，选出最终优化方案。

- 自动优化有一定的局限性；
- 在实际应用中，开发人员或者是使用者，应该有查询优化意识；
- 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DBMS上；
  - 应该找出DBMS的优化规律，以写出适合DBMS自动优化的SQL语句；
  - 对于DBMS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。

子查询：EXISTS vs IN，连接

作业

- **P112（e119）习题6**
- **思考题**
  P112－113，1，2，3，4，7