数 据 库 系 统

Database System

主讲: 张仲楠 教授
Email：zhongnan_zhang@xmu.edu.cn
Office：海韵A416

数据库系统

Database System

第九章　关系查询处理和查询优化

本章内容

- 关系数据库管理系统的查询处理步骤
- 查询优化技术：
  - 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

第九章 关系查询处理和查询优化

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 小 结

9.1 关系数据库系统的查询处理

- 查询处理: DBMS执行查询语句的过程
- 任务: 把用户提交的查询语句转换为**高效的查询执行计划**

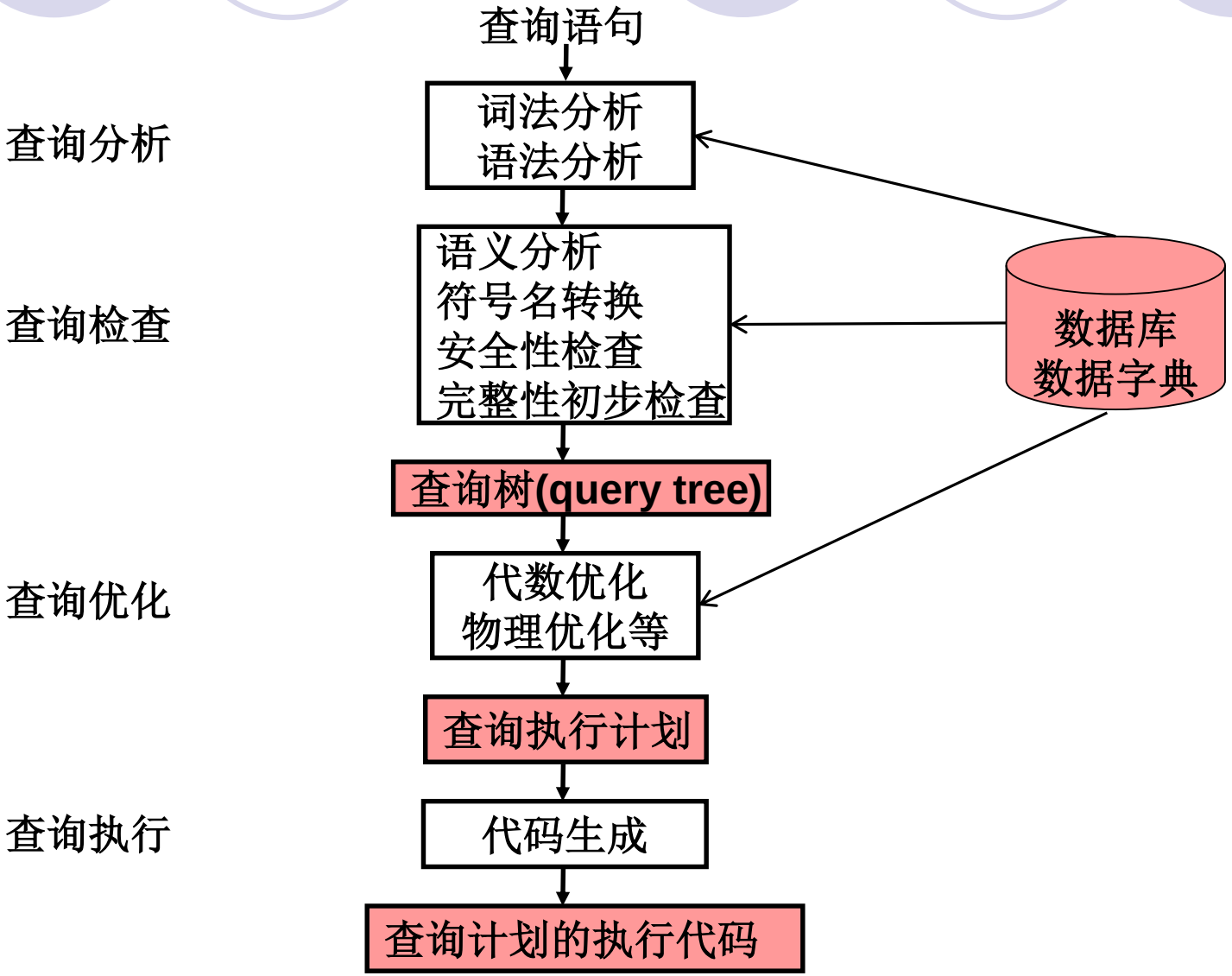
9.1 关系数据库系统的查询处理

- **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

查询处理步骤（续）


查询语句
查询分析
词法分析
语法分析
查询检查
语义分析
符号名转换
安全性检查
完整性初步检查
数据库
数据字典
查询树(query tree)
查询优化
代数优化
物理优化等
查询执行计划
查询执行
代码生成
查询计划的执行代码

1. 查询分析

- 查询分析的任务：对查询语句进行扫描、词法分析和语法分析
  - 词法分析：从查询语句中识别出**正确的语言符号**
  - 语法分析：进行语法检查

2. 查询检查

- 对合法的查询语句进行语义检查
  - 语义分析
  - 符号名转换
  - 安全性检查
  - 完整性初步检查
- 根据数据字典中有关的模式定义检查语句中的数据库对象，如关系名、属性名是否存在和有效
- 如果是对视图的操作，则要用视图消解方法把对视图的操作转换成对基本表的操作

2. 查询检查

- 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查询语句转换成内部表示，即等价的关系代数表达式。
- 关系数据库管理系统一般都用查询树，也称为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**/逻辑优化：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**
- 查询优化的选择依据
  - 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执行计划
- 代码生成器(code generator)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

- **选择操作典型实现方法：**
  - **1. 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
  - **2. 索引扫描方法**
    - 适合选择条件中的属性上有索引(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，再通过元组指针直接在查询的基本表中找到元组

选择操作的实现（续）

- ［例9.1］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＝'201215121'；

C3：Sage>20；

C4：Sdept＝'CS' AND Sage>20；

选择操作的实现（续）

- 全表扫描算法
  - 假设可以使用的内存为M块，全表扫描算法思想：

① 按照**物理次序**读Student的M块到内存

② 检查内存的每个元组t，如果满足选择条件，则输出t

③ 如果student还有其他块未被处理，重复①和②

选择操作的实现（续）

- 索引扫描算法
- [例9.1-C2] SELECT *

  FROM Student

  WHERE Sno='201215121'
  - 假设Sno上有索引(或Sno是散列码)
  - 算法：
    - 使用索引(或散列)得到Sno为'201215121'元组的指针
    - 通过元组指针在Student表中检索到该学生

选择操作的实现（续）

- [例9.1-C3] SELECT *

  FROM Student

  WHERE Sage>20

  - 假设Sage 上有B+树索引
  - 算法：
    - 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。

选择操作的实现（续）

- [例9.1-C4] SELECT *

  FROM Student

  WHERE Sdept='CS' AND Sage>20;

  - 假设Sdept和Sage上都有索引
  - 算法一：分别用Index Scan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

选择操作的实现（续）

- 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
  - 并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9.2］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）

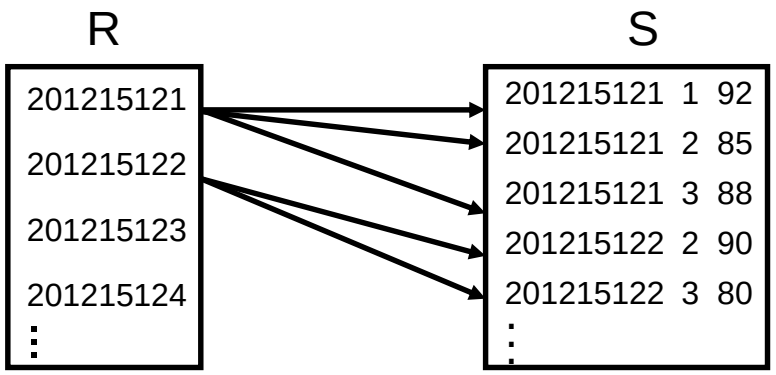
2. 排序-合并方法(sort-merge join 或merge join)

- 适合连接的诸表已经排好序的情况
- 如果2个表原来无序，执行时间要加上对两个表的排序时间
- 对于2个大表，先排序后使用sort-merge join方法执行连接，总的时间一般仍会大大减少

连接操作的实现（续）

- 排序-合并法算法：

1. 对于关系模式R和S，r和s为对应的关系。
2. JoinAttrs表示关系r和关系s连接时所用的连接属性。
3. r和s已经按照JoinAttrs分别排序。



排序-合并连接方法示意图

```
pr := r的第一个元组的地址；
ps := s的第一个元组的地址；
while (ps ≠ null and pr ≠ null) do
    begin
      t_s := ps所指向的元组；
      S_s := {t_s}；
      让ps指向关系s的下一个元组；
      done := false;
      while (not done and ps ≠ null) do
          begin
            t_s' := ps所指向的元组；
            if (t_s'[JoinAttrs] = t_s[JoinAttrs])
              then begin
                     S_s := S_s ∪ {t_s'};
                     让ps指向关系s的下一个元组；
                   end
              else done := true;
          end
      t_r := pr所指向的元组；
      while (pr ≠ null and t_r[JoinAttrs] < t_s[JoinAttrs]) do
          begin
            让pr指向关系r的下一个元组；
            t_r := pr所指向的元组；
          end
      while (pr ≠ null and t_r[JoinAttrs] = t_s[JoinAttrs]) do
          begin
            for each t_s in S_s do
              begin
                将 t_s ⋈ t_r加入结果中；
              end
            让pr指向关系r的下一个元组；
            t_r := pr所指向的元组；
          end
    end
```

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R和S中的元组散列到hash桶中
- **步骤：**
  - 划分阶段(partitioning phase)：
    - 对包含较少元组的表(比如R)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(S)进行一遍处理
    - 把S的元组散列到适当的hash桶中
    - 把元组与桶中所有来自R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

第九章关系查询处理和查询优化

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 小　结

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
- 由于关系表达式的语义级别很高，使关系系统可以从关系表达式中**分析查询语义**，提供了执行查询优化的可能性

9.2 关系数据库系统的查询优化

- **9.2.1 查询优化概述**
- **9.2.2 一个实例**

9.2.1 查询优化概述

- 查询优化是关系系统的优点所在
  - 减轻了用户选择存取路径的负担
  - 用户只需要提出“干什么”
  - 不要求用户具有较高的技术水平

9.2.1 查询优化概述

- 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
  - (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。

查询优化概述（续）

- (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

查询优化概述（续）

- RDBMS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
    - 执行开销主要包括：
      - 磁盘存取块数(I/O代价)
      - 处理器时间(CPU代价)
      - 查询的内存开销
    - **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**

查询优化概述（续）

- 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 关系数据库系统的查询优化

- **9.2.1 查询优化概述**
- **9.2.2 一个实例**

9.2.2 一个实例

[例9.3] 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

一个实例（续）

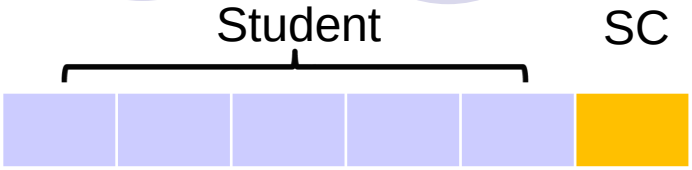
- 系统可以用多种等价的关系代数表达式来完成这一查询

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

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

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

一个实例（续）



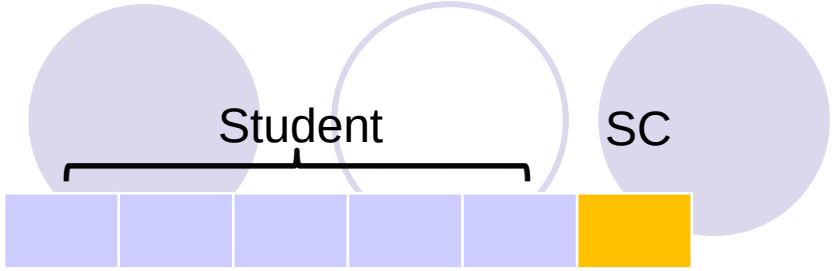
一、第一种情况

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

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
  - 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（续）



- 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

读Student　　读SC

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

- 其中，读Student表100块。读SC表20遍，每遍100块。
- 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10^6块

一个实例（续）

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。读取中间文件同样需10^6块
- 满足条件的元组假设仅50个，均可放在内存

一个实例（续）

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读写数据块

  $= 2100+10^6 +10^6$
- 所有内存处理时间均忽略不计

一个实例（续）

二、 第二种情况

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

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
   - 自然连接的结果比第一种情况大大减少，为**10^4**个
   - 写出数据块为$10^4/10=10^3$，**为第一种情况的千分之一**
2. 读取中间文件块，读取的块数也为10^3。
3. 把第2步结果投影输出。

- 第二种情况总的块数=2100+ 10^3 +10^3
- 其执行代价大约是第一种情况的488分之一

一个实例（续）

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，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。
3. 把连接结果投影输出

- 第三种情况总的读写数据库=100+100
- 其执行代价大约是第一种情况的万分之一，是第二种情况的20分之一

一个实例（续）

- 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
- 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

一个实例（续）

- 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**

第九章 关系查询处理和查询优化

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 小　结

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
- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on every legal database instance(两个关系代数表达式被认为是**等价**的，如果对于每个合法的数据库实例它们均产生一组相同的元组)
- 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）

- **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

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

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

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

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

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

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

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

关系代数表达式等价变换规则（续）

3. 投影的串接定律

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

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

从另一个角度，**选择条件也可以分开检查**。

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

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

选择条件F只涉及属性A_1，…，A_n。

若F中有不属于A_1，…，A_n的属性B_1，…，B_m则有更一般的规则：

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

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

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

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和E_2是两个关系表达式，A_1，…，A_n是E_1的属性，B_1，…，B_m是E_2的属性，则

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

11. 投影与并的分配律

设E_1和E_2有相同的属性名，则

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

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

- 启发式算法（**heuristic algorithm)**是相对于最优化算法提出的。
- 一个基于直观或经验构造的算法，对优化问题的实例能在可接受的计算成本（计算时间、占用空间等）内，给出一个近似最优解，该近似解于真实最优解的偏离程度不一定可以事先预计

9.3.2 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. 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**

查询树的启发式优化（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,尤其是等值连接

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

查询树的启发式优化（续）

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
- 当查询的是视图时，定义视图的表达式就是公共子表达式的情况

查询树的启发式优化（续）

- 遵循这些启发式规则，应用9.3.1的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

  方法：

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

  (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**。

规则4：选择的串接定律

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

规则4：　合并或分解选择运算
规则5-9：　选择运算与其他运算交换

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：$\pi_{A_1,A_2,\cdots A_n}(\pi_{B_1,B_2,\cdots B_m}(E)) \equiv \pi_{A_1,A_2,\cdots A_n}(E)$
  - 等价变换规则3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 $\pi_{A_1,A_2,\cdots A_n}(\sigma_F(E)) \equiv \pi_{A_1,A_2,\cdots A_n}(\sigma_F(\pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E)))$

(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**

规则4：合并或分解选择运算

查询树的启发式优化（续）

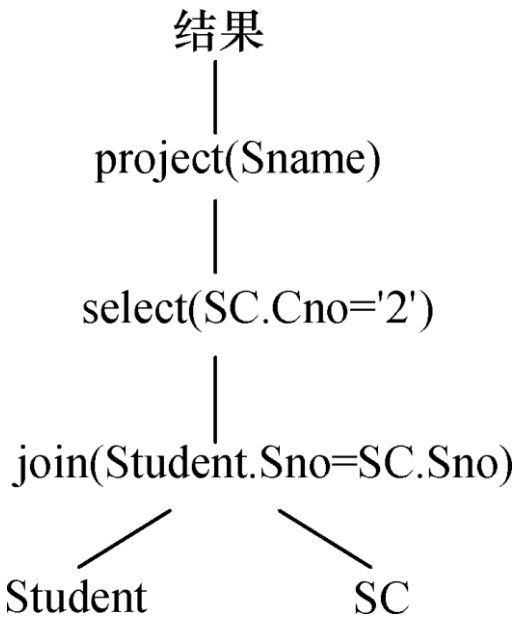
(5) 把上述得到的语法树的**内节点分组**。每一双目运算(×，⋈，∪，–)和它所有的**直接祖先**为一组(这些直接祖先是(σ，π运算)。

- 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

查询树的启发式优化（续）

［例4］ 下面给出［例3］中 SQL语句的代数优化示例。

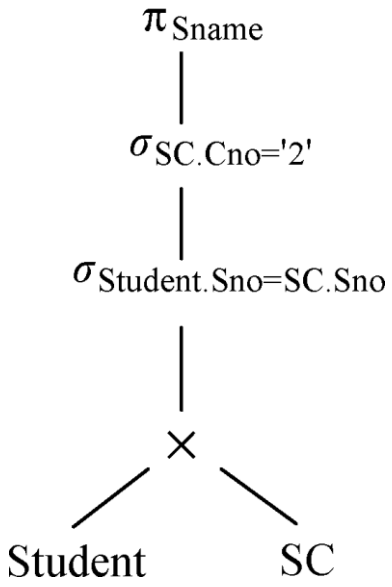
(1) 把SQL语句转换成查询树，如下图所示



查询树

查询树的启发式优化（续）

为了使用关系代数表达式的优化法，假设内部表示是关系代数语法树，则上面的查询树如下图所示。

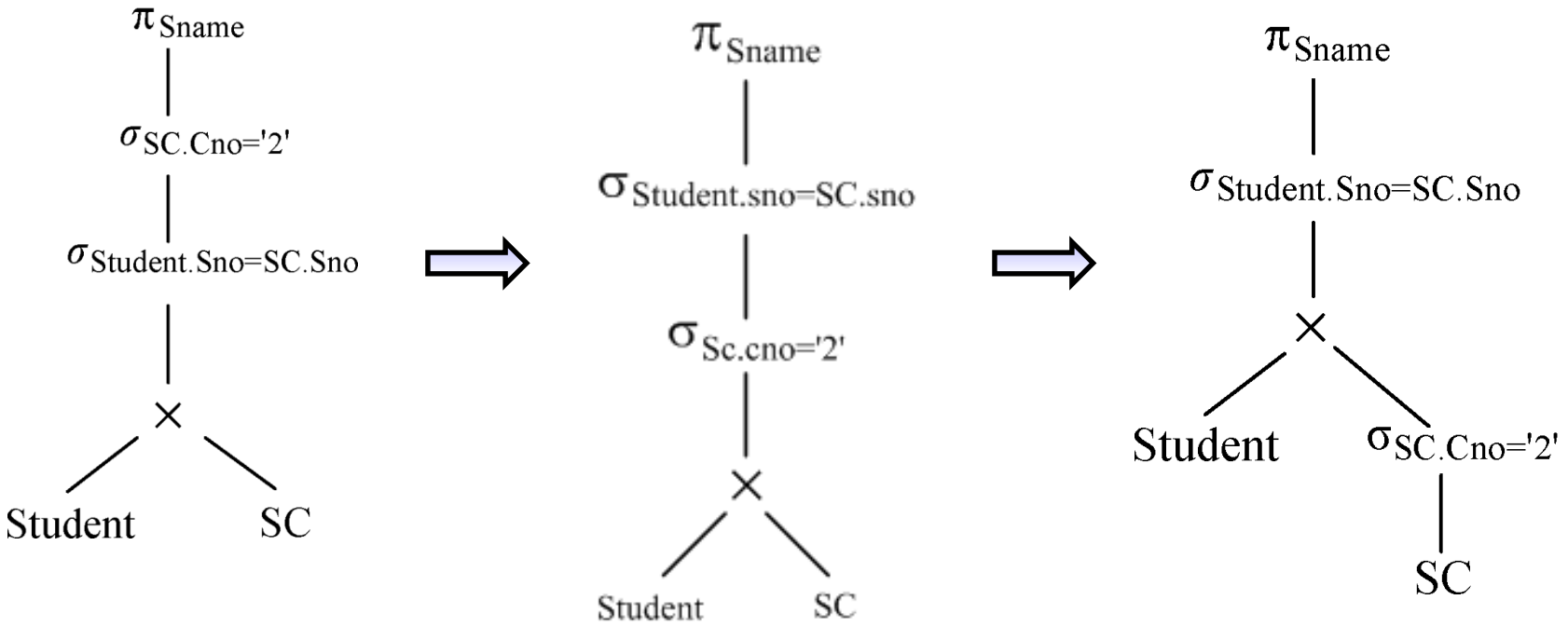


关系代数语法树

查询树的启发式优化（续）

(2) 对查询树进行优化

利用规则4、6把选择$\sigma_{SC.Cno}$='2'移到叶端，查询树便转换成下图所示的优化的查询树。这就是9.2.2节中Q_3的查询树表示

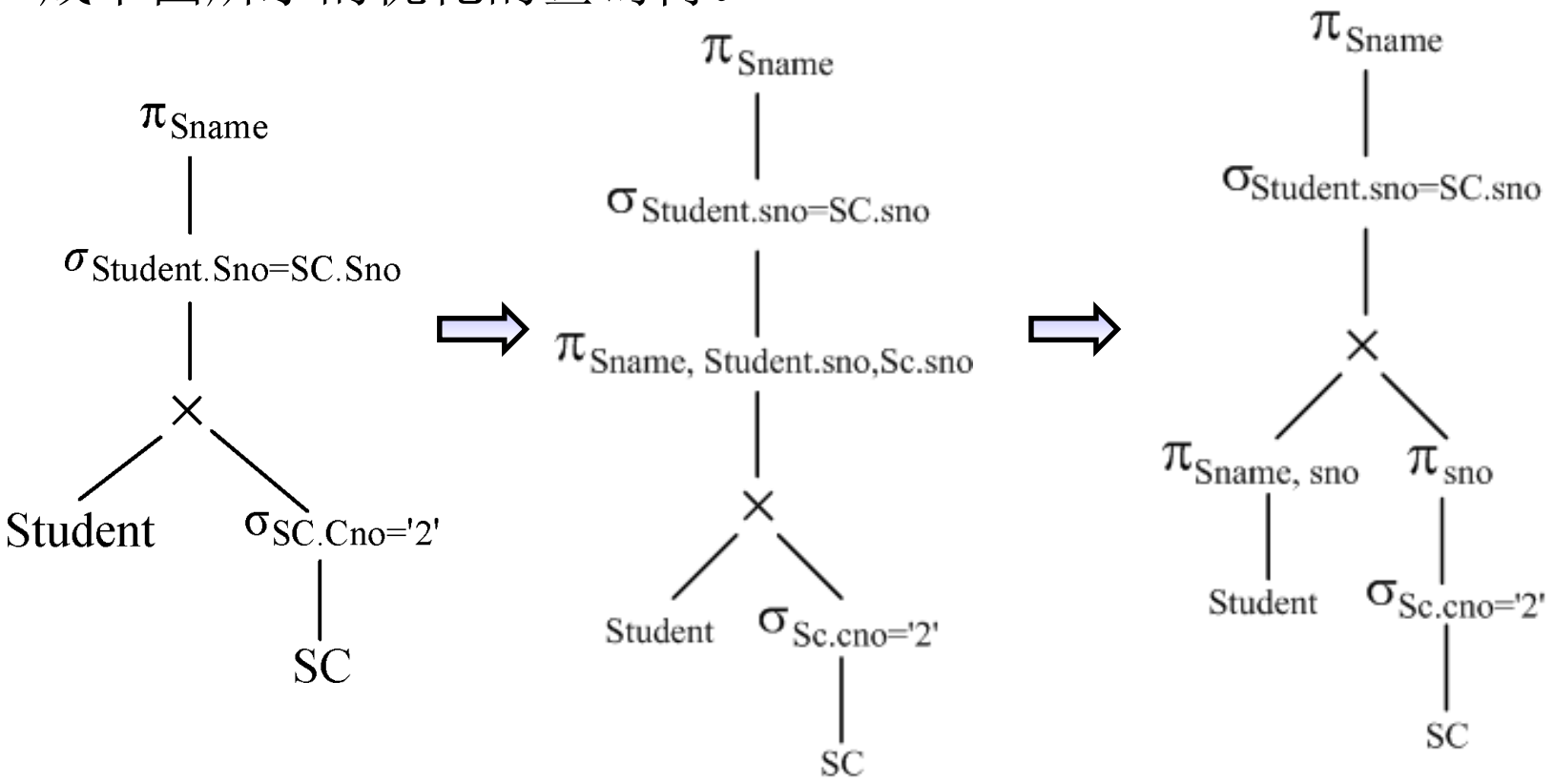


规则4: $\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$　规则6: $\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$

查询树的启发式优化（续）

(3) 对查询树进行优化

利用规则5、10把投影π_{Sname}向叶端移动，查询树便转换成下图所示的优化的查询树。



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

第九章 关系查询处理和查询优化

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 小　结

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

物理优化（续）

- 物理优化方法
  - 基于规则的启发式优化
    - **启发式规则**是指那些在大多数情况下都适用，但不是在每种情况下都是适用的规则。
  - 基于代价估算的优化
    - 优化器估算不同执行策略的代价，并选出具有最小代价的执行计划。

物理优化（续）

- 物理优化方法（续）
  - 两者结合的优化方法：
    - 可能的执行策略很多，穷尽所有策略进行代价估算是不可行的
    - 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
    - 然后分别计算这些候选方案的执行代价，较快地选出最终的优化方案

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

基于启发式规则的存取路径选择优化(续)

- **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。

基于启发式规则的存取路径选择优化(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

基于启发式规则的存取路径选择优化(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

基于启发式规则的存取路径选择优化(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
- 如果某些属性上有一般的索引
  - 则可以用［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

基于启发式规则的存取路径选择优化(续)

- **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

基于启发式规则的存取路径选择优化(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(b)较少的表，作为**外表**(外循环的表) 。

理由：

外表只会读一遍

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

内表要读多遍

9.4 物理优化（续）

- **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**

9.4.2 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

基于代价的优化（续）

- 一、 统计信息
- 二、 代价估算示例

基于代价的优化（续）

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)

数据库中记录的保存方式有一种是顺序文件组织。方便查询和修改。但是进行插入和删除就比较麻烦。 为了减少移动，就在每个记录的后面添加上一条指针，指向下一条记录的位置。 对于插入操作，如果这条记录所在的块儿中有位置，就可以插进去。 否则，就需要将新记录插入到一个新的块儿中，这个新块儿，就叫做**溢出块**。

基于代价的优化（续）

2. 对基表的每个列

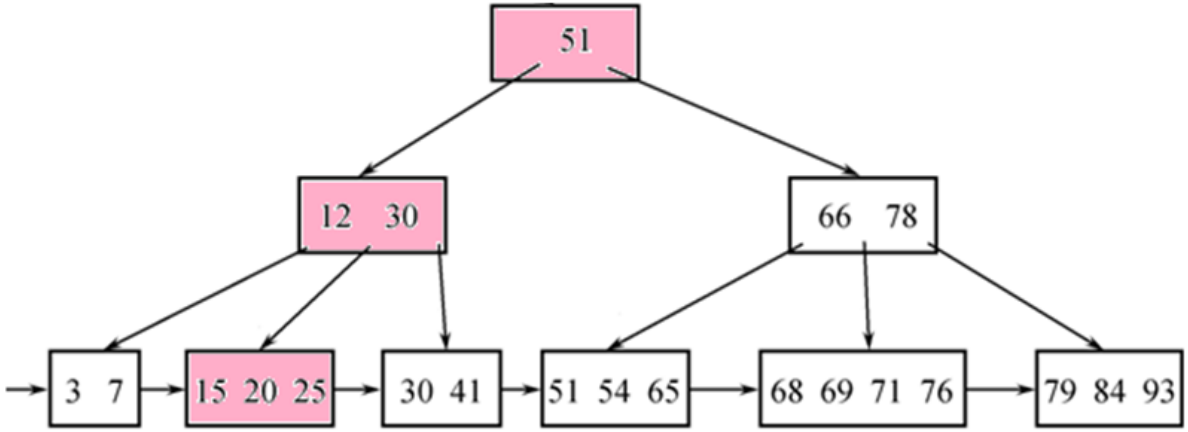
- 该列不同值的个数(m)
- 选择率(f)
  - 如果不同值的分布是均匀的，f＝1/m
  - 如果不同值的分布不均匀，则每个值的选择率＝具有该值的元组数/N
- 该列最大值
- 该列最小值
- 该列上是否已经建立了索引
- 索引类型(B+树索引、Hash索引、聚集索引)

也称为聚簇索引

基于代价的优化（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)



基于代价的优化（续）

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

基于代价的优化（续）

```
SELECT *
FROM Student
WHERE Sage>20;
```

```
SELECT *
FROM Student
WHERE Sdept='CS'AND Sage>20;
```

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

基于代价的优化（续）

- 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

基于代价的优化（续）

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+Bs/(K-1) Br

  - 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。

基于代价的优化（续）

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

注意：外排序

第九章 关系查询处理和查询优化

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 小　结

9.5 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
  - 基于代价估算的优化
- 本章的目的：希望读者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**
- 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