数据库系统概论

An Introduction to Database System

第九章 关系查询处理和查询优化

关系系统及其查询优化（续）

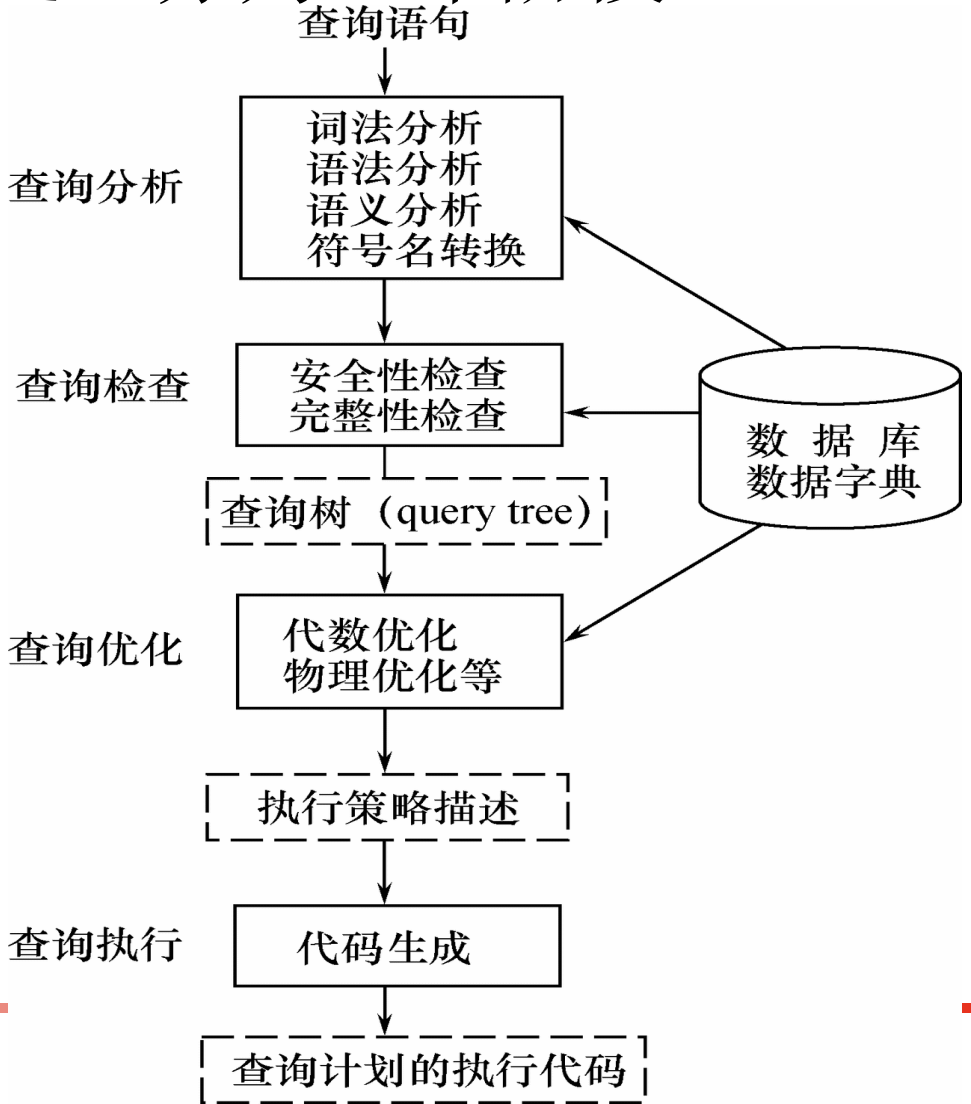
- 本章目的：
  - **RDBMS** 的查询处理步骤
  - 查询优化的概念
  - 基本方法和技术

- 查询优化分类 ：
  - 代数优化：关系代数表达式的优化
  - 物理优化：存取路径和底层操作算法的选择

9.1.1 查询处理步骤

- RDBMS 查询处理分为 4 个阶段 ：

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
- 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

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**
    - 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 (或散列) 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

选择操作的实现（续）

- ［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。

选择操作的实现（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 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
  - **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

连接操作的实现（续）

1. 嵌套循环方法 (nested loop)

- SELECT * FROM Student，SC
  WHERE Student.Sno=SC.Sno；

  - 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)

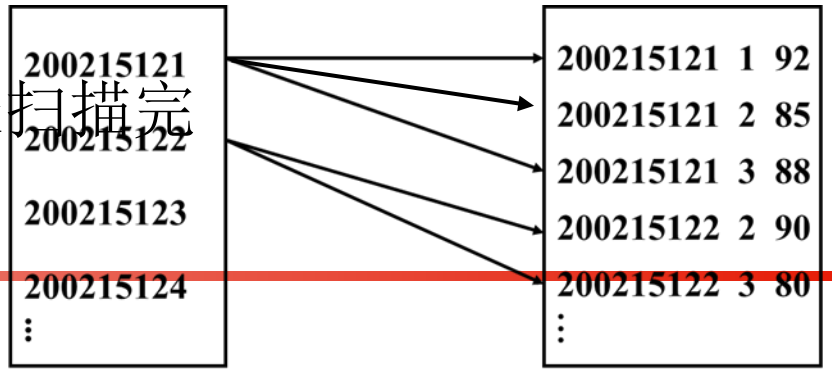
①如果连接的表没有排好序，先对 Student 表和 SC 表按连接属性 Sno 排序 ；

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

- **SELECT * FROM Student, SC**
  WHERE Student.Sno=SC.Sno；



连接操作的实现（续）

- Student 表和 SC 表都只要扫描一遍
- 如果 2 个表原来无序，虽然执行时间要加上对两个表的排序时间，但对于 2 个大表，先排序后使用排序 - 合并方法执行连接，总的时间一般仍会大大减少

连接操作的实现（续）

3. 索引连接 (index join) 方法

① 在 SC 表上建立属性 Sno 的索引，如果原来没有的话。

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

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

9.2.1 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）

- 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。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
- 求选修了 2 号课程的学生姓名

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

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

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

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

$$\frac{1000}{10} \quad \frac{1000}{10\times 5}\times\frac{10000}{100}$$

=100+20×100=2100 块

- 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（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（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（续）

- 二、 第二种情况

$Q_2=\pi_{Sname}(\sigma_{Sc.Cno='2'}$ (Student ⋈ 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 105 s
   - 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 $10^4/10/20=50$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≈ 105+50+50≈205s

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（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
- 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）

- 该例充分说明查询优化的必要性，同时给出了查询优化方法的初步概念：
  - 既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
  - 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

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）

- 常用的等价变换规则（教材 P269 ）

1. 连接、笛卡尔积交换律
2. 连接、笛卡尔积的结合律
3. 投影的串接定律
4. 选择的串接定律
5. 选择与投影操作的交换律
6. 选择与笛卡尔积的交换律

关系代数表达式等价变换规则（续）

7. 选择与并的分配律

8. 选择与差运算的分配律

9. 选择对自然连接的分配律

10. 投影与笛卡尔积的分配律

11. 投影与并的分配律

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，则可以在扫描该关系的同时完成所有的这些运算以避免重复扫描关系

3. 把投影同其前或其后的双目运算结合起来。没有必要为去掉某些字段而扫描一遍关系。

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。连接，特别是等值连接运算要比同样关系上的笛卡尔积省很多时间。

5. 找出公共子表达式。重复出现的子表达式结果，写入中间文件。

9.3.2 查询树的启发式优化

- 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：优化的关系树
- 方法：

(1) 利用等价变换规则 4 把形如 $\sigma_{F_1 \wedge F_2 \wedge \cdots \wedge F_n}(E)$ 变换为 $\sigma_{F_1}(\sigma_{F_2}(\cdots(\sigma_{F_n}(E))\cdots))$。

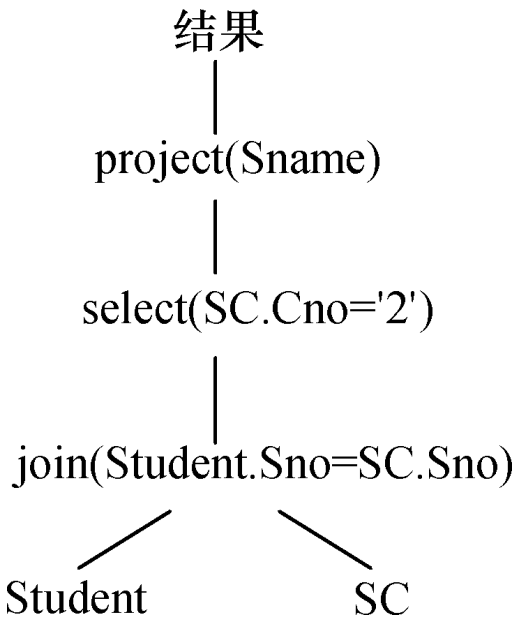
(2) 对每一个选择，利用等价变换规则 4～9 尽可能把它移到树的叶端。

(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。

查询树的启发式优化（续）

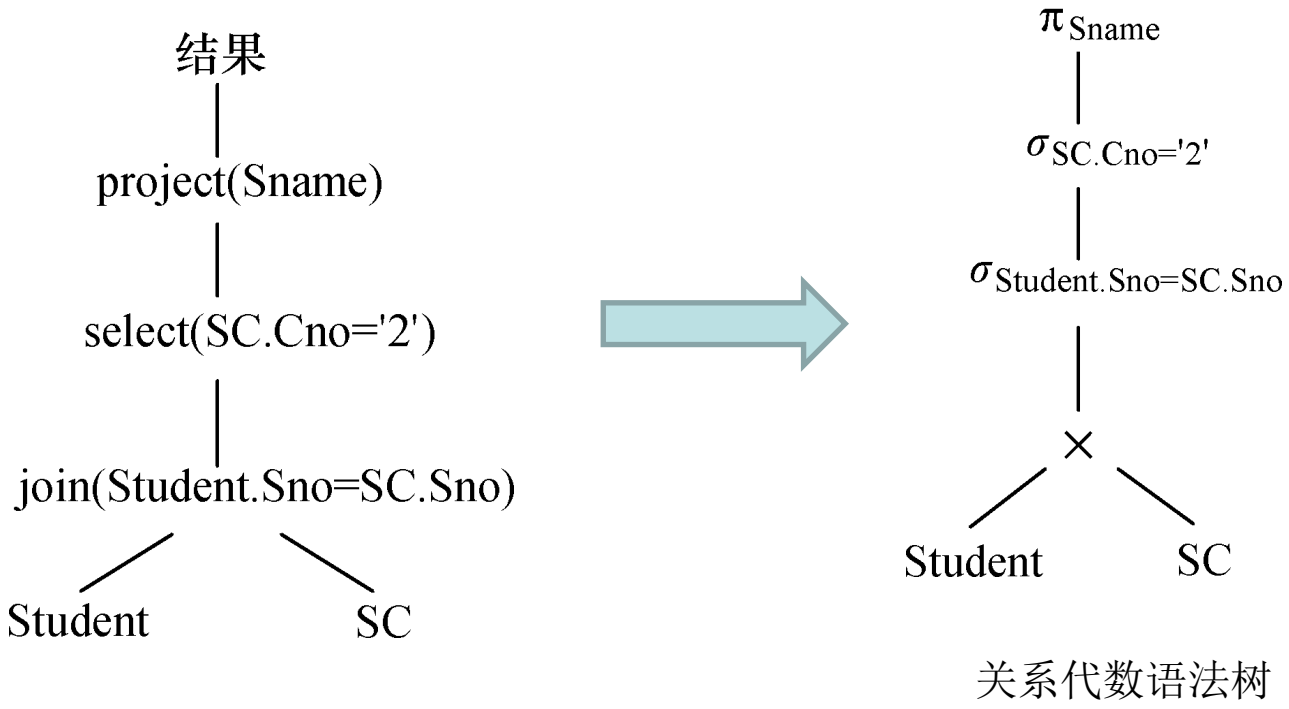
［例 4 ］　下面给出［例 3 ］中 SQL 语句的代数优化示例。

(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。

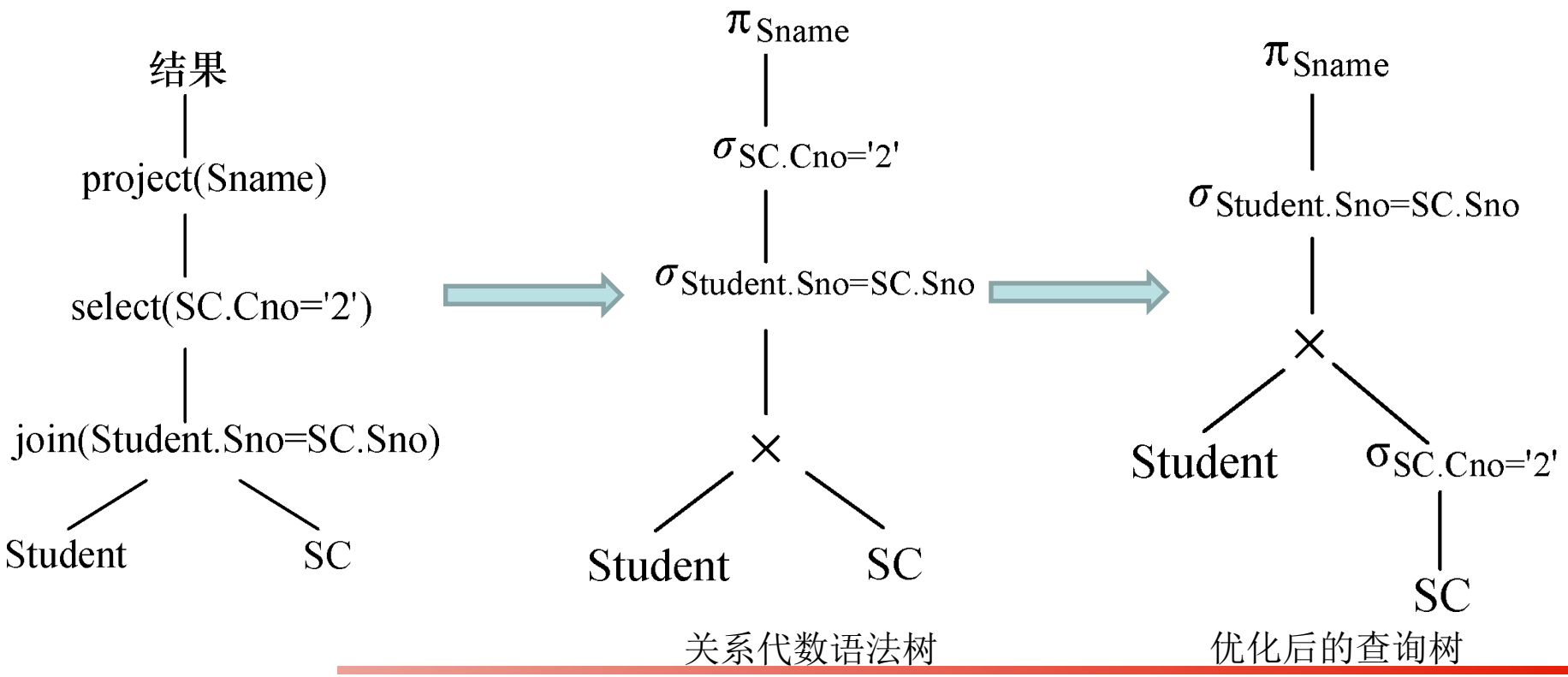


关系代数语法树

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



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）

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

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有：
- 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：

1. 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 RDBMS 会自动建立主码索引）。
2. 对于选择条件是非主码 = 值的查询，并且选择列上有索引，则要估算查询结果的元组数目，如果比例较小（少于 10% ）可以使用索引扫描方法，否则使用全表顺序扫描。

9.4.1 基于启发式规则的存取路径选择优化

3. 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，同样要估算查询结果的元组数目，如果比例较小（少于10%）可以使用索引扫描方法，否则使用全表顺序扫描。

4. 对于 AND 连接的选择条件，如果有涉及这些属性的组合索引，则优先采用组合索引的扫描方法。

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 。

了解和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