



哈爾濱工業大學

Harbin Institute of Technology

数据库系统

万晓珑 博士
大数据计算研究中心

wxl@hit.edu.cn



哈爾濱工業大學

Harbin Institute of Technology

数据库系统

万晓珑 博士
大数据计算研究中心

wxl@hit.edu.cn



哈爾濱工業大學

Harbin Institute of Technology

数据库系统

第九章 关系查询处理和查询优化

第三篇 系统篇

❖ 讨论数据库管理系统中查询处理和事务管理的基本概念和基础知识

- 第9章 关系查询处理和查询优化
- 第10章 数据库恢复技术
- 第11章 并发控制
- *第12章 数据库管理系统

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 查询计划的执行

9.6 小 结

关系查询处理和查询优化（续）

❖ 本章内容：

- 关系数据库管理系统的查询处理步骤
- 查询优化的概念
- 基本方法和技术

❖ 查询优化分类：

- 代数优化：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

9.1 关系数据库系统的查询处理

9.1.1 查询处理步骤

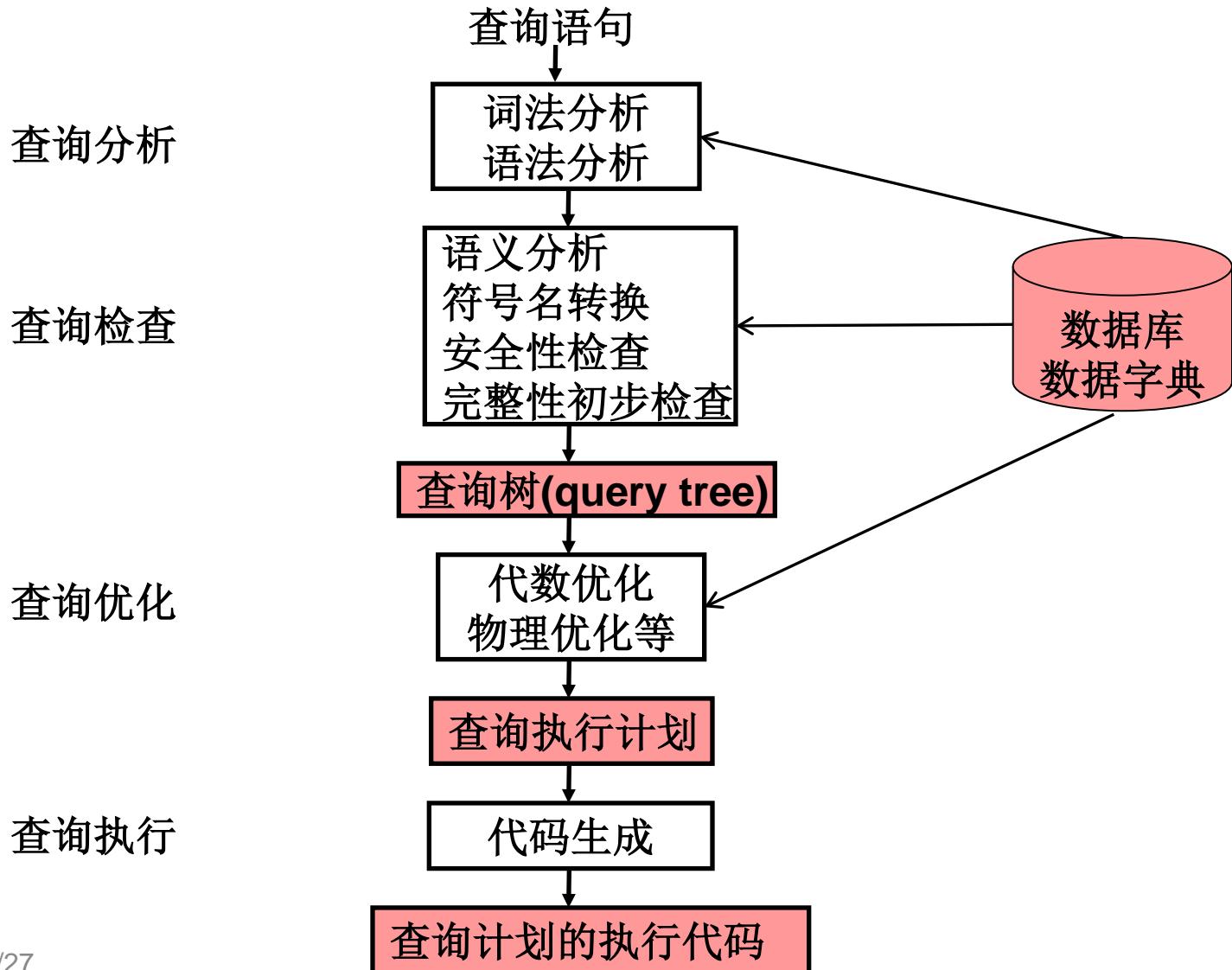
9.1.2 实现查询操作的算法示例

9.1.1 查询处理步骤

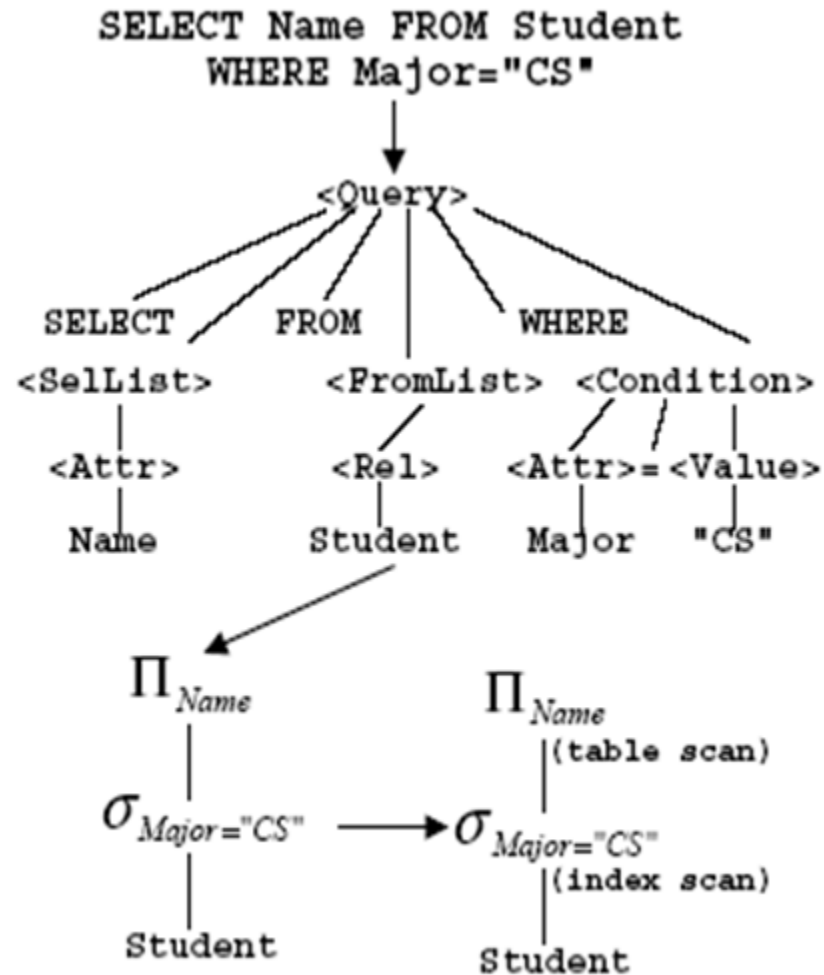
❖ 关系数据库管理系统查询处理阶段：

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）



查询处理步骤（续）



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

❖ 查询检查的任务

- 合法权检查、视图转换、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
 - 无权限或违反完整性约束，将拒绝执行该查询
 - 此时的完整性检查，是初步的、静态的检查
- ❖ 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：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

1. 选择操作的实现
2. 连接操作的实现

1.选择操作的实现

❖ 选择操作典型实现方法:

(1) 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
- 定长元组 **vs.** 变长元组
- 适合小表，不适合大表；或者大多数元组都需要考虑的情况

1.选择操作的实现

❖ 选择操作典型实现方法:

(2) 索引扫描方法 (Index Scan)

- 适合于选择条件中的属性上有索引
- 索引示例: **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
- 单维索引 **vs.** 多维索引

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

选择操作的实现（续）

- 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。

选择操作的实现（续）

- 算法三：在**Sdept**和**Sage**上具有二维索引，
 - 直接查找满足条件**Sdept='CS' AND Sage>20**的元组指针
 - 到**Student**表中检索
 - 得到计算机系年龄大于**20**的学生

2.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) 嵌套循环算法(nested loop join)**
- (2) 排序-合并算法(sort-merge join)**
- (3) 索引连接(index join)算法**
- (4) Hash Join算法**

连接操作的实现（续）

（1）嵌套循环算法(nested loop join)

- 对外层循环(**Student**表)的每一个元组(**s**), 检索内层循环(**SC**表)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-合并算法(sort-merge join)

- 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
- 取**Student**表中第一个**Sno**，依次扫描**SC**表中具有相同**Sno**的元组
- 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
- 重复上述步骤直到**Student** 表扫描完

连接操作的实现（续）

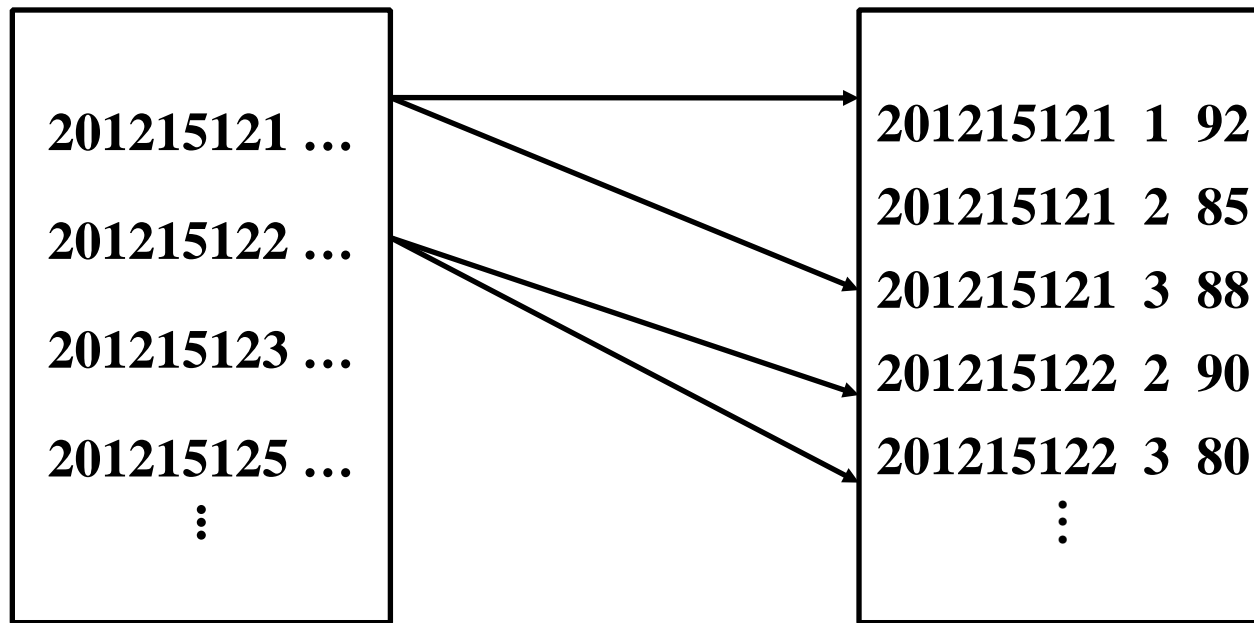


图9.2 排序-合并连接方法示意图

连接操作的实现（续）

- ❖ 排序-合并连接算法中，**Student**表和**SC**表至少扫描一遍
- ❖ 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
- ❖ 对于两个表都比较大的情况，先排序后使用排序-合并连接算法执行连接，总的时间相比较嵌套循环算法一般仍会减少

连接操作的实现（续）

（3）索引连接(index join)算法

■ 步骤：

- ① 在**SC**表上已经建立属性**Sno**的索引。
 - 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**表的元组处理完为止

连接操作的实现（续）

（4）Hash Join算法

- 把连接属性作为hash码，用同一个hash函数把Student表和SC表中的元组散列到hash表中
- 包括划分和试探两个阶段
- 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

连接操作的实现（续）

（4）Hash Join算法

■ 划分阶段(building phase)

- 对包含较少元组的表(如**Student**表)进行一遍处理
- 把它的元组按**hash**函数分散到**hash**表的桶中

连接操作的实现（续）

（4）Hash Join算法

■ 试探阶段(probing phase)

- 对另一个表(**SC表**)进行一遍处理
- 把**SC表**的元组也按同一个hash函数（hash码是连接属性）进行散列
- 把**SC**元组与桶中来自**Student**表并与之相匹配的元组连接起来

连接操作的实现（续）

- ❖ **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
- ❖ 该前提不满足的情况
 - 两个关系的较小者也无法放入内存
 - 采用基于磁盘的划分阶段和试探阶段
 - 同样考虑**Student**和**SC**的连接

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 查询计划的执行**

9.6 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关系数据库管理系统性能的关键因素
- ❖ 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

❖ 关系系统的查询优化

- 是关系数据库管理系统实现的关键技术，也是关系系统的优点所在
- 减轻了用户选择存取路径的负担

查询优化概述（续）

❖ 非关系系统

- 用户使用过程化的语言表达查询要求，执行何种记录级的操作，以及操作的序列是由用户来决定的
- 用户必须了解存取路径，系统要提供用户选择存取路径的手段，查询效率由用户的存取策略决定
- 如果用户做了不当的选择，系统是无法对此加以改进的

查询优化概述

❖ 查询优化的优点

- 用户不必考虑如何最好地表达查询以获得较好的效率
- 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）

❖ 关系**DBMS**通过某种**代价模型**计算出各种查询执行策略的执行代价，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
- 扇区或磁盘块（物理和逻辑最小存储单元）
- 读取一个磁盘块的典型时间 **vs.** **CPU**处理速度

查询优化概述（续）

- ❖ 关系**DBMS**通过某种**代价模型**计算出各种查询执行策略的执行代价，选取代价最小的执行方案
 - 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，为什么？优化和实际执行时间的折衷)

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

- ❖ 一个关系查询可以对应不同的执行方案，其效率可能相差非常大。
- ❖ **[例9.3]** 求选修了**2**号课程的学生姓名。

SELECT Student.Sname

FROM Student, SC

WHERE Student.Sno=SC.Sno AND SC.Cno='2'

- 假定有**1000**个学生记录，**10000**个选课记录
- 选修**2**号课程的选课记录为**50**个

一个实例（续）

❖ 可以用多种等价的关系代数表达式来完成这一查询

- $Q_1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='2'}(Student \times SC))$
- $Q_2 = \pi_{Sname}(\sigma_{SC.Cno='2'}(Student \bowtie SC))$
- $Q_3 = \pi_{Sname}(Student \bowtie \sigma_{SC.Cno='2'}(SC))$

SELECT Student.Sname

FROM Student, SC

WHERE Student.Sno=SC.Sno AND SC.Cno='2'

一个实例（续）

1. 第一种情况

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

一个实例（续）

（1） 计算广义笛卡尔积

- 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。
- 依次从**SC**读入一块和内存中的**Student**元组做笛卡儿积，直到**SC**表处理完。**SC**的每个元组和**Student**内存中的每个元组都需要连接，笛卡儿积元组装满一块后就写到中间文件。
- 再读入若干块**Student**元组，并读取**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**，内存中存放**5块Student元组**和**1块SC元组**，读取总块数（**1000个学生记录**，**10000个选课记录**）

$$\frac{1000}{10} + \frac{1000}{10 \times 5} \times \frac{10000}{100} = 100 + 20 \times 100 = 2100 \text{ 块}$$

- 读**Student表100块**，读**SC表20遍**，每遍**100块**，则总计要读取**2100数据块**。
- 连接后的笛卡尔积元组数为 **$10^3 \times 10^4 = 10^7$** 。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。读取中间文件花费的时间(同写中间文件一样)需读入 10^6 块。
- 已知满足条件的元组仅**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

❖ 执行查询的总读写数据块= $2100+10^6+10^6$

一个实例（续）

2. 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(嵌套循环连接)
- 自然连接的结果比第一种情况大大减少，为 **10^4** 个元组
- 写出数据块= **10^3** 块

一个实例（续）

2.第二种情况（续）

（2）读取中间文件块，执行选择运算，读取的数据块 = 10^3 块

（3）把第2步结果投影输出。

- 执行查询的总读写数据块 = $2100 + 10^3 + 10^3$
- 其执行代价大约是第一种情况的488分之一

一个实例（续）

3.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，读一遍**SC**表，存取**100**块，满足条件的元组**50**个，不使用中间文件。

（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。

一个实例（续）

3.第三种情况（续）

- 第三种情况总的读写数据块=**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

一个实例（续）

- ❖ 若**Student**表在**Sno**上也有索引
 - 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Q1变换为Q2、 Q3

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



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

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

- 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

实例: 小结

❖ 在 Q_3 中

- **SC**表的选择操作算法有全表扫描或索引扫描，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，采用索引连接代价也较小
- 这就是物理优化

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

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 查询计划的执行**

9.6 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
- ❖ 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：

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

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

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

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

关系代数表达式等价变换规则（续）

❖ 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

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

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

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

关系代数表达式等价变换规则（续）

3. 投影的串接定律

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

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

关系代数表达式等价变换规则（续）

4.选择的串接定律

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

- 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

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

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

- 选择条件F只涉及属性 A_1, \dots, A_n

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

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

- 若F中不属于 A_1, \dots, A_n 的属性 B_1, \dots, B_m

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**E₁**中的属性，则

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

- 如果**F=F1** \wedge **F2**，**F1**只涉及**E1**中的属性，**F2**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：

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

- 若**F1**只涉及**E1**中的属性，**F2**涉及**E1**和**E2**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, \dots, A_n 是 E_1 的属性， B_1, \dots, B_m 是 E_2 的属性，则

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

11. 投影与并的分配律

设 E_1 和 E_2 有相同的属性名，则

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

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

❖ 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，没有必要为了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。

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。

规则4：选择的串接定律

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

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：

规则5-9：选择运算与其他运算交换

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

（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**4**：合并或分解选择运算

规则**5**：投影运算与选择运算交换

- 等价变换规则**3**使一些投影消失或使一些投影出现
- 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） 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**3**：合并或分解投影运算

规则**5,10,11**：投影运算与其他运算交换

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 (分组意味着全局优化和局部优化的关系，作为整体参与全局优化策略)。

- 每一双目运算(\times , \bowtie , \cup , $-$)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(\times), 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

查询树的启发式优化（续）

❖ [例9.4] 下面给出[例9.3]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

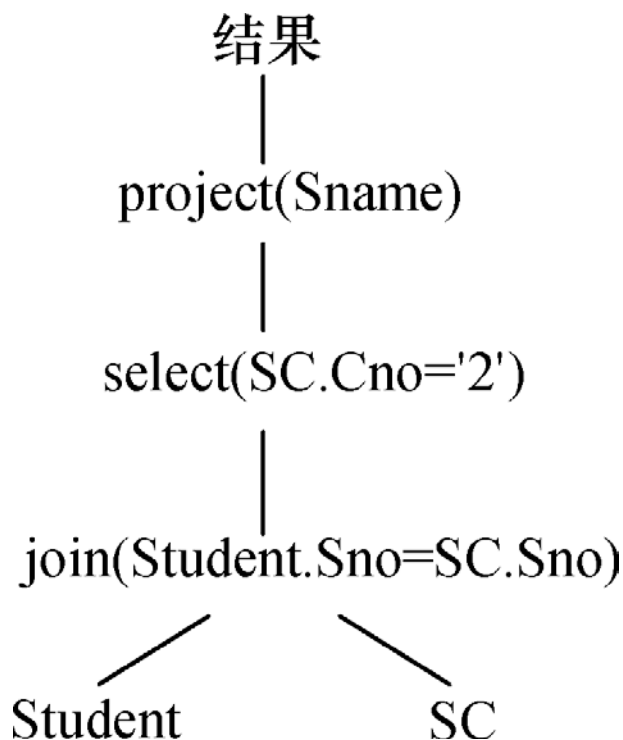


图9.3 查询树图

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，则上面的查询树如图9.4所示。

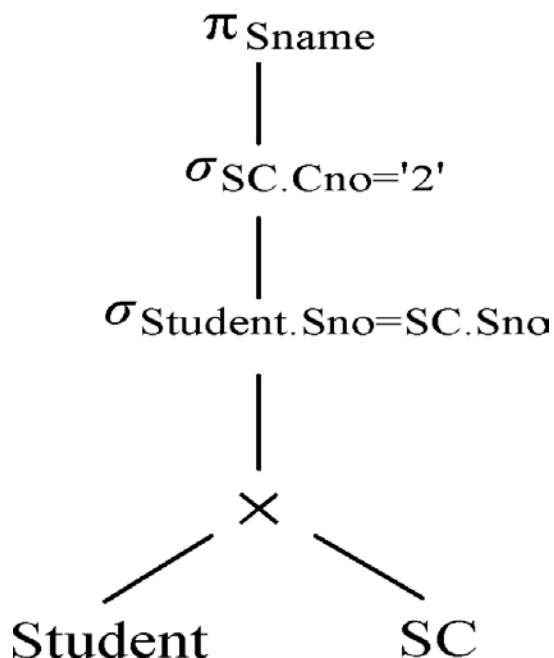
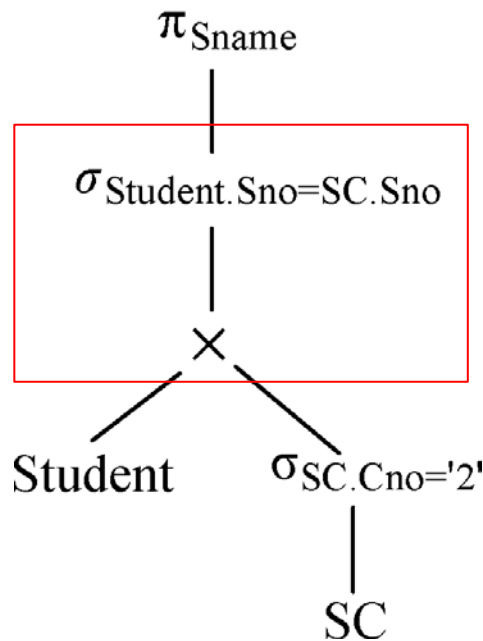


图9.4 关系代数语法树图

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，图9.4查询树便转换成下图优化的查询树。这就是9.2.2节中Q3的查询树表示。



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 查询计划的执行**

9.6 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：

- 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

1.选择操作的启发式规则

2.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）

1. 选择操作的启发式规则

- 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：
 - （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）

（2）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）

（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

- 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（多维索引）
 - 优先采用组合索引扫描方法
- 如果某些属性上有单维索引，可以用索引扫描方法
 - 通过分别查找满足每个条件的指针，求指针的交集
 - 通过索引查找满足部分条件的元组，然后在扫描这些元组时判断是否满足剩余条件
- 其他情况：使用全表顺序扫描

（5）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）

2. 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-合并算法

(2) 如果一个表在连接属性上有索引

- 选用索引连接算法

(3) 如果上面2个规则都不适用，其中一个表较小

- 选用Hash join算法

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，即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 $B_r + B_r * B_s / (K-1)$
- 显然应该选块数小的表作为外表

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

9.4.2 基于代价的优化

- ❖ 启发式规则优化是定性选择，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）

1.统计信息

2.代价估算示例

3.优化方法

基于代价的优化（续）

1.统计信息

- 基于代价的优化方法要计算查询的各种不同执行方案的执行代价，它与数据库的状态密切相关
- 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(**I**)
 - 占用的块数(**B**)

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)
- 列最大值和最小值
- 列上是否已经建立了索引
- 哪种索引(**B+**树索引、**Hash**索引、聚集索引)
- 可以计算选择率(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, 则要计算每个值的选择率, $f=\text{具有该值的元组数}/N$

基于代价的优化（续）

（3）对索引

- 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）

2. 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）

（2）索引扫描算法的代价估算公式

- 如果选择条件是“码=值”
 - ✓ 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 $\text{cost} = L + 1$

基于代价的优化（续）

（2）索引扫描算法的代价估算公式（续）

- 如果选择条件涉及非码属性

- ✓ 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)

- ✓ 满足条件的元组可能会保存在不同的块上，所以(最坏的情况)**cost=L+S**

基于代价的优化（续）

（2）索引扫描算法的代价估算公式（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
 - ✓ **$\text{cost} = L + Y/2 + B/2$**
 - ✓ 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

- 嵌套循环连接算法的代价

$$\text{cost} = Br + Br * Bs / (K - 1)$$

- 如果需要把连接结果写回磁盘

$$\text{cost} = Br + Br * Bs / (K - 1) + (Frs * Nr * Ns) / Mrs$$

- 其中**Frs**为连接选择性，表示连接结果元组数的比例
- **Nr**和**Ns**分别表示关系**R**和关系**S**中的元组数
- **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

$$\text{cost} = B_r + B_s + (F_{rs} * N_r * N_s) / M_{rs}$$

- 如果必须对文件排序

- ✓ 还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

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

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 查询计划的执行**

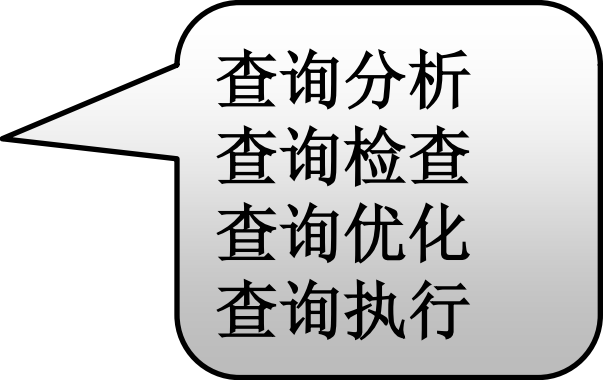
9.6 小 结

9.6 小 结

❖ 查询处理是关系数据库管理系统的核心，查询优化技术是查询处理的关键技术

❖ 本章主要内容

- 查询处理过程
- 查询优化
 - 代数优化
 - 物理优化
- 查询执行



查询分析
查询检查
查询优化
查询执行

9.6 小 结

❖ 查询处理是关系数据库管理系统的核心，查询优化技术是查询处理的关键技术

❖ 本章主要内容

- 查询处理过程

- 查询优化

 - 代数优化

 - 物理优化

- 查询执行



启发式代数优化

9.6 小 结

❖ 查询处理是关系数据库管理系统的核心，查询优化技术是查询处理的关键技术

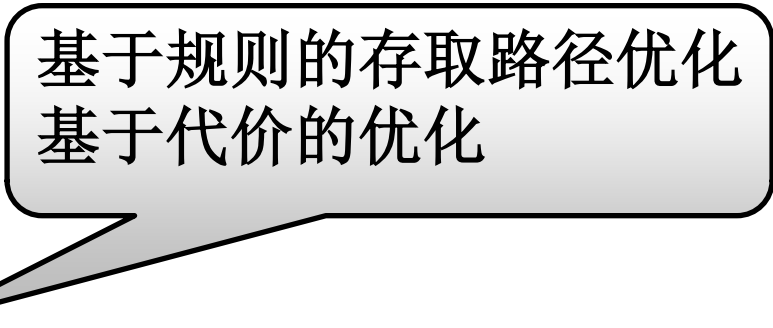
❖ 本章主要内容

- 查询处理过程

- 查询优化

 - 代数优化

 - 物理优化



基于规则的存取路径优化
基于代价的优化

- 查询执行

9.6 小 结

- ❖ 查询处理是关系数据库管理系统的核心，查询优化技术是查询处理的关键技术
- ❖ 本章主要内容
 - 查询处理过程
 - 查询优化
 - 代数优化
 - 物理优化
 - 查询执行

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关系数据库管理系统
 - 应该找出关系数据库管理系统的优化规律，以写出适合关系数据库管理系统自动优化的**SQL**语句
- ❖ 对于关系数据库管理系统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