

关系数据库的查询处理优化

讲解人：李鸿岐

查询处理

```
SELECT *  
FROM Student  
WHERE dNo='01' AND (age<20 or age>23);
```

❖ 关系数据库管理系统查询处理阶段：

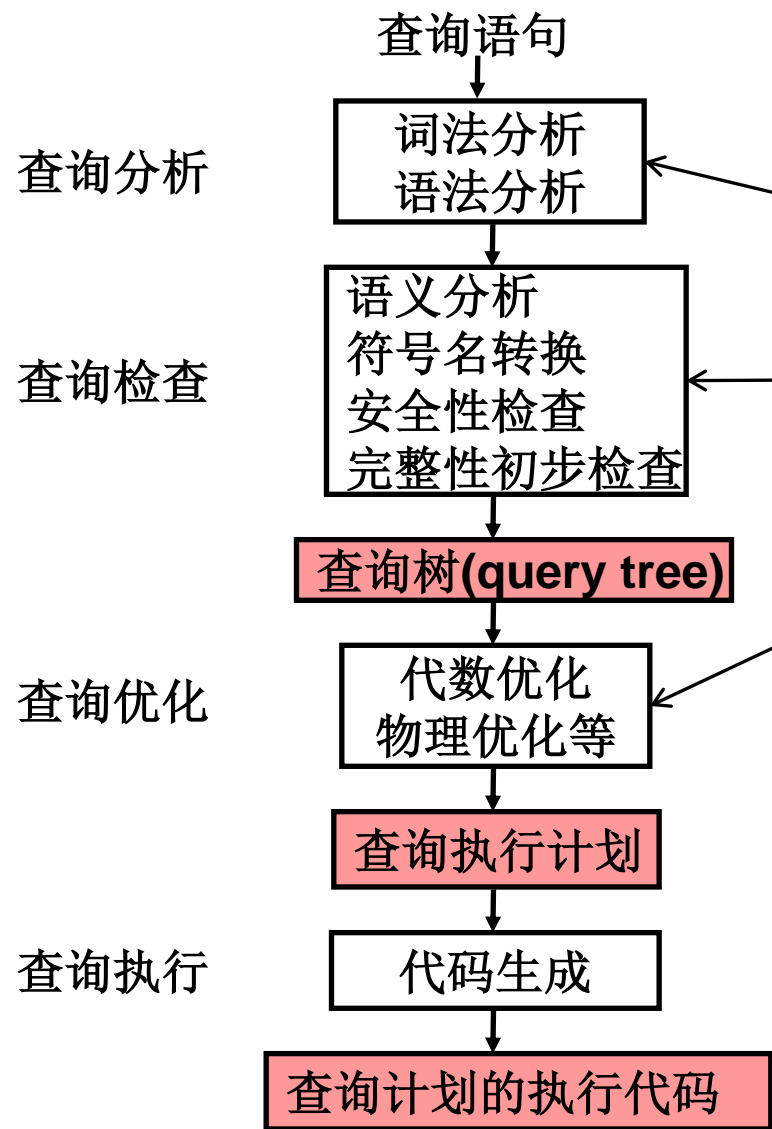
1. 查询分析
2. 查询检查
3. 查询优化
4. 查询执行

查询处理

❖ 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 **SELECT** **WHERE**

■ **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



❖ 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

查询处理

□ 查询检查

- ❖ 初步的、静态的检查
- ❖ 动态的、参照完整性检查 ✖

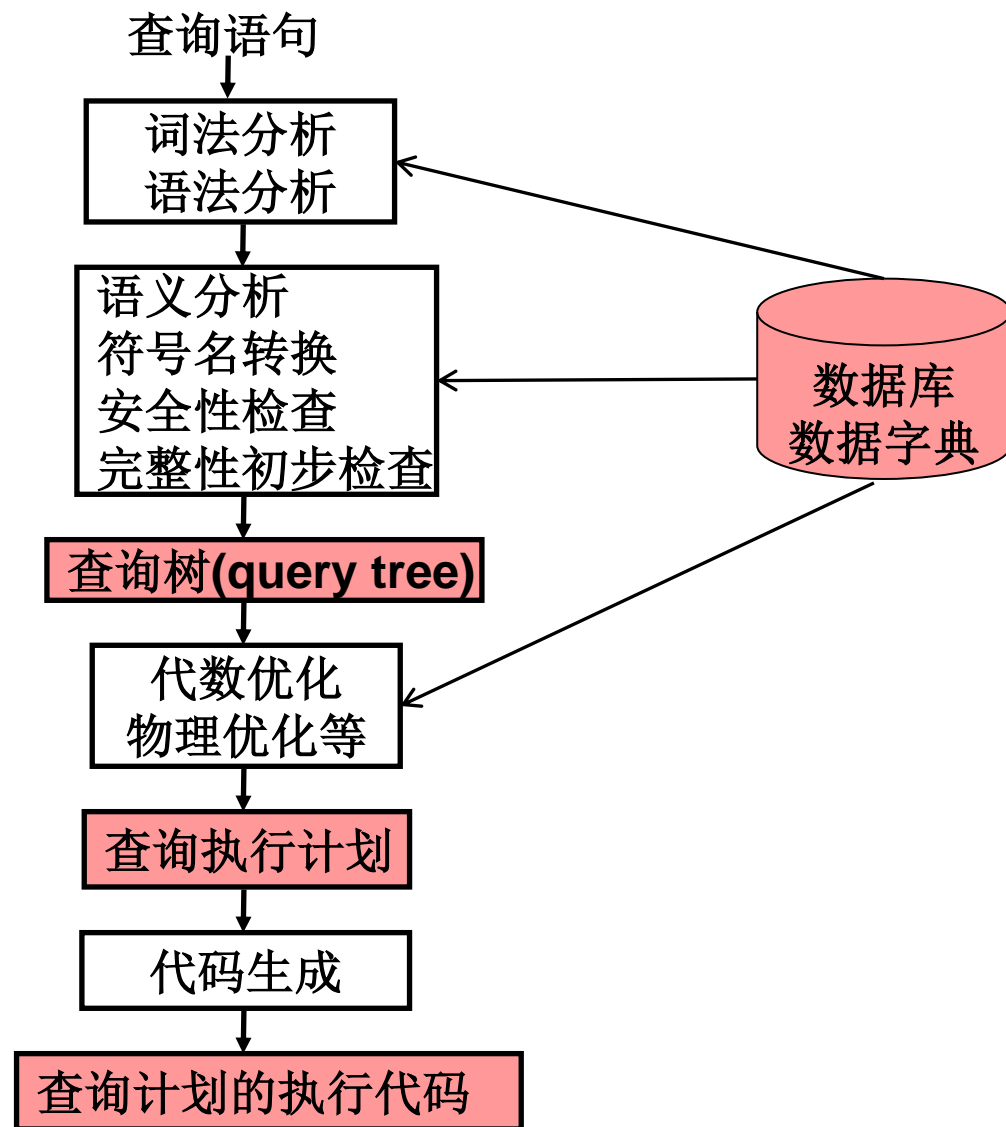
- ❖ 根据数据字典中的用户权限和完整性约束定义（增、删、改）对**用户**的**存取权限**进行检查
- ❖ 检查通过后把**SQL**查询语句转换成内部表示，即等价的**关系代数表达式**。
- ❖ 关系数据库管理系统一般都用查询树，也称为**语法分析树**来表示扩展的关系代数表达式。

查询分析

查询检查

查询优化

查询执行



查询处理

□ 查询优化

❖ 选择一个高效执行的查询处理策略

❖ 分类

❖ 启发式规则：
选择操作下推，优先执行

■ 代数优化/逻辑优化：指关系代数表达式的优化（操作次序）

■ 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)

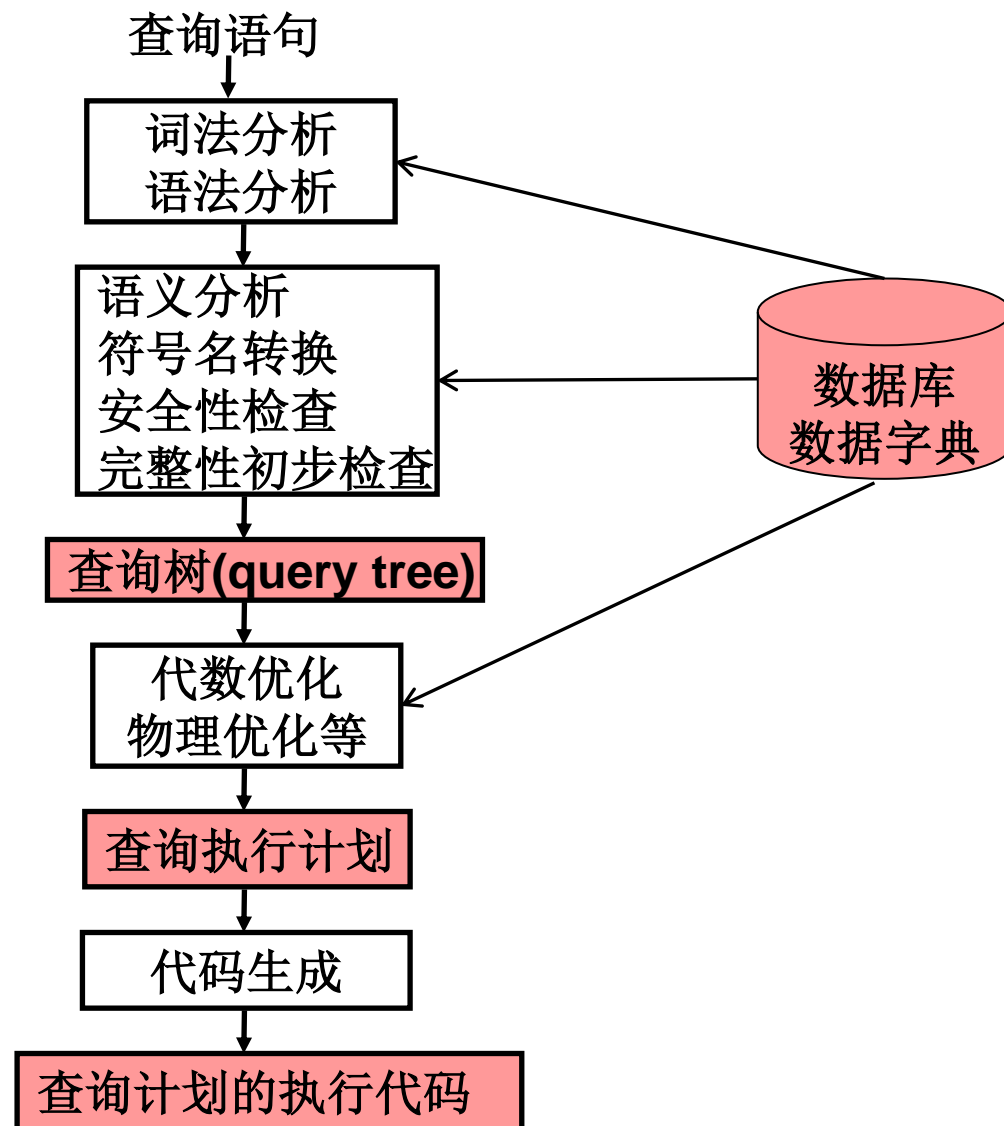
■ 基于语义(semantic based)

查询分析

查询检查

查询优化

查询执行

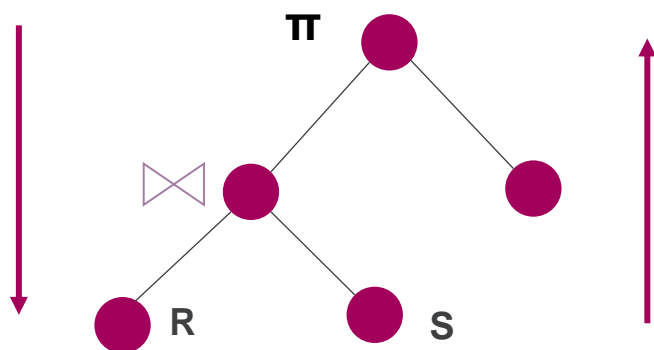


❖ 如没有结果返回时，可直接通过语义返回输出 6

查询处理

❑ 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
- ❖ 两种执行方法
 - 自顶向下（迭代器）
 - 自底向上（数据驱动）



底层向上

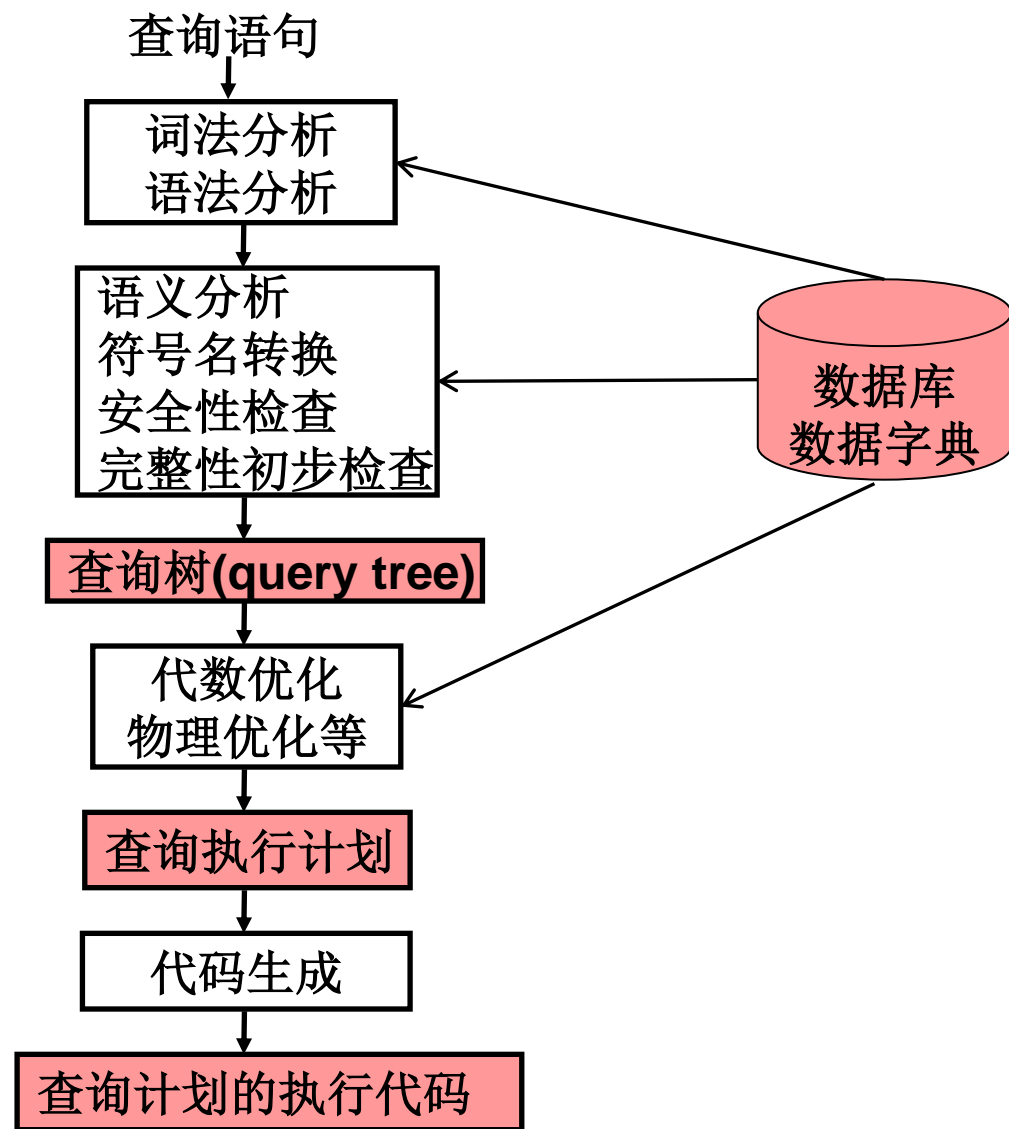
触发方式

查询分析

查询检查

查询优化

查询执行



□ 算法示例

1. 选择操作的实现
2. 连接操作的实现

❖ 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
- 适合小表，不适合大表

(2) 索引扫描方法 (Index Scan)

- 适合于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

□ 算法示例

❖ [案例] **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;

❖ 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

❖ [例C4]

假设**Sdept**和**Sage** 上都有索引

■ 算法2:

- 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。

□ 算法示例

1. 选择操作的实现

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

❖ 只讨论等值连接(或自然连接)最常用的实现算法

❖ [案例] **SELECT ***

FROM Student, SC

WHERE Student.Sno = SC.Sno;

(1) 嵌套循环算法(nested loop join)

(2) 排序-合并算法(sort-merge join 或merge join)

(3) 索引连接(index join)算法

(4) Hash Join算法

查询处理

□ 算法示例

1. 选择操作的实现
2. 连接操作的实现

等值连接 $R \bowtie S$ 的结果如下

$R.B = S.B$

R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a3	b4	12
a4		

S

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

A	R.B	C	S.B	E
a1	b1	5	b1	3
a1	b2	6	b2	7
a2	b3	8	b3	10
a2	b3	8	b3	2

(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。

1. 逐一扫描R表的每一行

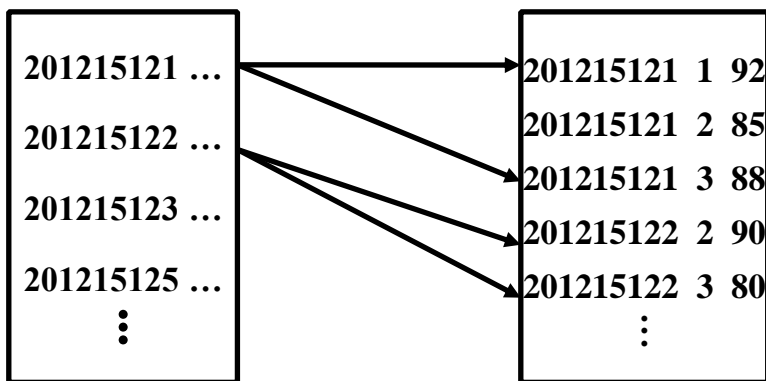
2. 为R表中的每一行读整个S表一次

查询处理

□ 算法示例

1. 选择操作的实现

2. 连接操作的实现



排序-合并连接方法示意图

(2) 排序-合并算法(sort-merge join 或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

- ❖ **Student**表和**SC**表都只要扫描一遍
- ❖ 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**
- ❖ 对于大表，先排序后使用排序-合并连接算法执行连接，总的时间一般仍会减少

查询处理

□ 算法示例

1. 选择操作的实现

2. 连接操作的实现

等值连接 $R \bowtie S$ 的结果如下

$R.B = S.B$

1. 按连接属性B排序：假设 $b1 < b2 < b3 < b4 < b5$

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a3	b4	12
a4		

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

A	R.B	C	S.B	E
a1	b1	5	b1	3
a1	b2	6	b2	7
a2	b3	8	b3	10
a2	b3	8	b3	2

2. 逐一扫描R表的每一行

3. 为R表中的每一顺序扫描S表，直到遇到大于R表连接字段的元组

(2) 排序-合并算法(sort-merge join 或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

□ 算法示例

1. 选择操作的实现
2. 连接操作的实现

(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**表中的元组处理完为止

□ 算法示例

1. 选择操作的实现

2. 连接操作的实现

(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。
- 划分阶段(building phase, 也称为partitioning 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
- 试探阶段(probing phase, 也称为连接阶段join 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

❖ 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

查询优化

查询优化

- ❖ 查询优化在**RDBS**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
- 查询优化的必要性
- 查询优化概述

查询优化

□ 查询优化的必要性

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

[案例] 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
- 选修**2**号课程的选课记录为**50**个

查询优化

❖ 可以用多种等价的关系代数表达式来完成这一查询

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

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

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

查询优化

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

(1) 计算广义笛卡尔积

❖ 算法:

- 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

查询优化

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

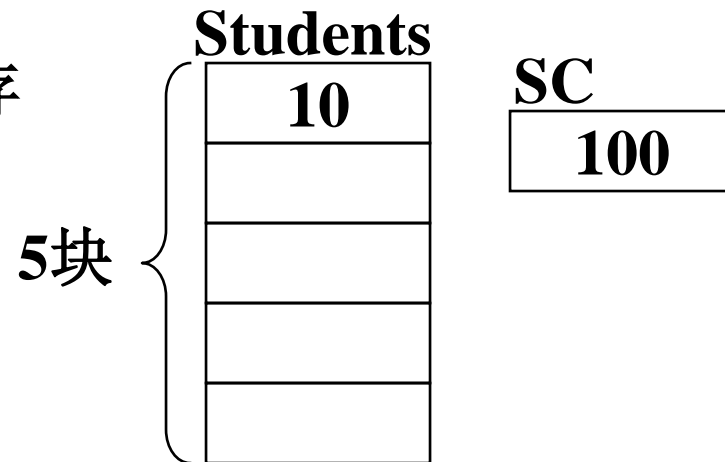
❖ 设一个块能装10个Student元组或100个SC元组，在内存中存放5块Student元组和1块SC元组，则读取总块数为

Student
块数

Student 读入内存的次数

SC块数

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



■ 读Student表100块，需20遍，读SC表20遍，每遍100块，则总计要读取2100数据块。

❖ 若每秒读写20块，则总计要花 $2100/20=105s$

■ 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 10^6 块。

❖ 写出这些块，则总计要花 $10^6/20=5 \times 10^4s \approx 13.89h$

查询优化

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

(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。
 - ❖ 读入这些块，则总计要花 $10^6/20=5*10^4s \approx 13.89h$
- 若满足条件的元组假设仅 **50** 个，均可放在内存。

每秒读20块

(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 = $2100 + 10^6 + 10^6$

❖ 执行查询的总时间约 $105s + 13.89h + 13.89h \approx 27.78h$

查询优化

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

(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** 个元组（**10000**条选课记录）
- 写出数据块 = **10^3** 块（假设每块装**10**个元组）

(2) 读取中间块

- 执行选择运算，读取的数据库为 **10^3** 块

(3) 把第（2）步的结果在**Sname**上作投影输出，得到最终结果

- ❖ 第二种情况下执行查询的总读写数据块 = $2100 + 10^3 + 10^3$
- ❖ 执行查询的总时间约 $105s + 10^3/20s + 10^3/20s \approx 205s \approx 3.4 \text{ min}$ 约为第一种**的488分之一**

查询优化

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

(1) 对**SC**表作选择运算

■ 选修2号课程的选课记录为**50**个

■ 只需读一遍**SC**表，存取**100**块，因为满足条件的元组仅**50**个，

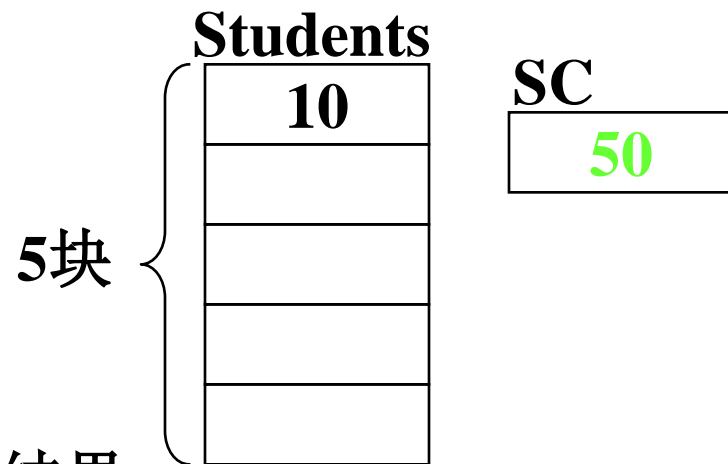
■ 不必使用中间文件

(2) 读取**Student**表

■ 把读入的**Student**元组和内存中的**SC**元组作连接

■ 只需读一遍**Student**表，共**100** 块

(3) 把第(2)步的结果在**Sname**上作投影输出，得到最终结果



❖ 第二种情况下执行查询的总读写数据块 = $100 + 100$

❖ 执行查询的总时间约 $100/20s + 100/20s \approx 10s$ 约为第二种的20分之一，为第一种的五分之一

查询优化

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

❖ 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

■ 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

查询优化

❖ 把代数表达式Q1变换为Q2、 Q3

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



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

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

- 连接后的元组可以先不立即写出，而是和后续的选择操作结合，省去读、写开销
- 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

查询优化

❖ 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。

查询优化

□ 查询优化概述

用户只要提出“干什么”，而不必指出“怎么干”

❖ 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又是关系系统的优点所在
- 减轻了用户选择存取路径的负担

❖ 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

□ 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

➤ 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。

➤ 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

□ 查询优化概述

❖ 关系数据库管理系统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)

- 处理机时间(CPU代价)

- 查询的内存开销

- I/O代价是最主要的

❖ 磁盘I/O操作涉及机械动作，所需时间较内存操作高几个数量级

■ 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

□ 查询优化概述

❖ 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)
- 查询优化的搜索空间有时非常大，实际选择的策略不一定是最优的，而是较优的



查询优化

❖ 代数优化

❖ 物理优化

关于本讲内容



祝各位学习愉快!

感谢观看！

讲解人：李鸿岐