上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

SJTU

数 据 库 技 术

郭 捷

(guojie@sjtu.edu.cn)

饮 水 思 源 • 爱 国 荣 校

第十章 查询处理和优化

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目 录

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

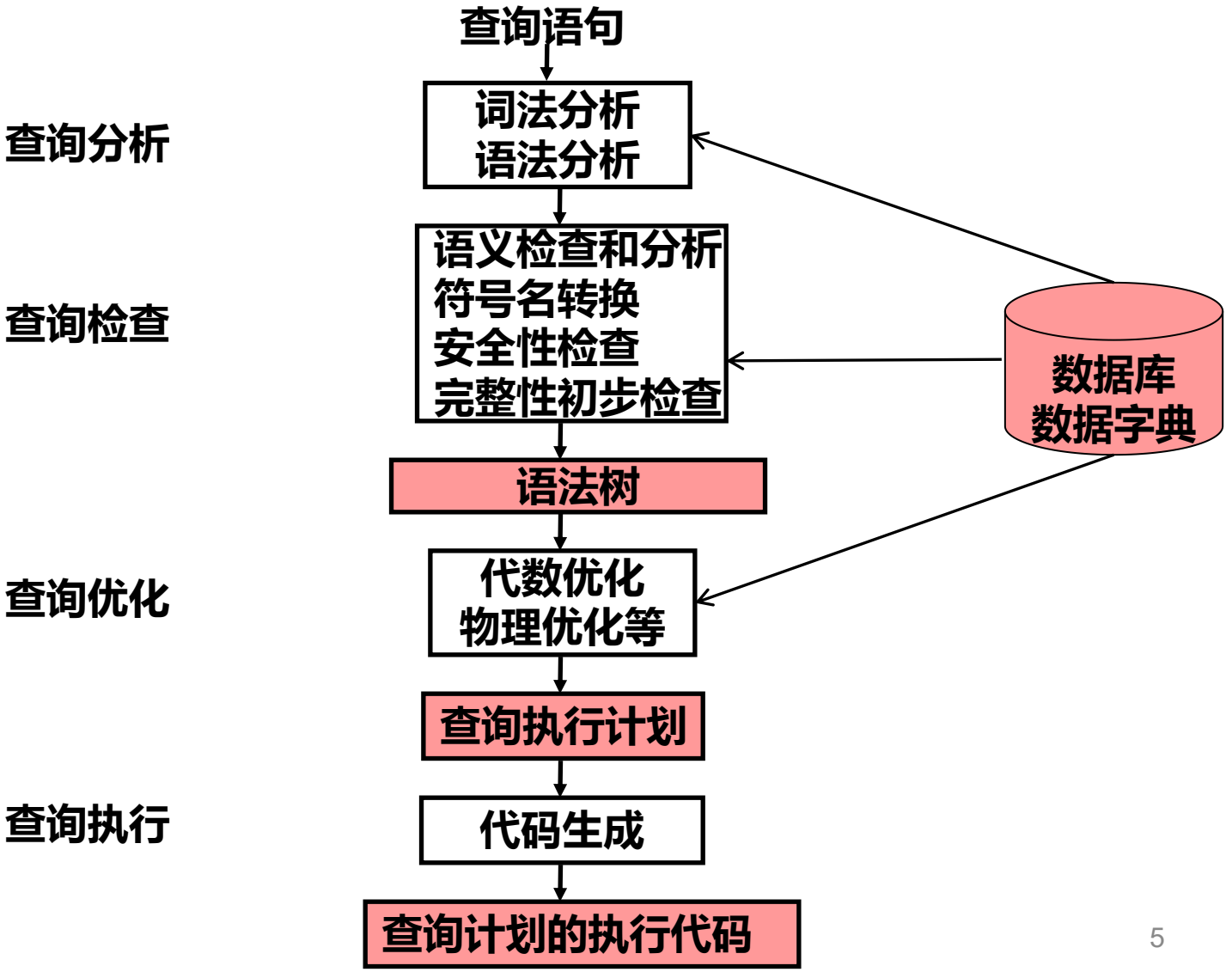
01

查询处理步骤

查询处理步骤

❖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



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，判断查询语句是否符合SQL语法规则。

2. 查询检查

- 查询检查的任务：
  - 语义检查和分析
  - 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语法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；

02

实现查询操作的算法

1.选择操作的实现

选择操作典型实现方法：

（1）全表扫描算法（full 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）索引扫描算法（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。

1.选择操作的实现

[例10.1]　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＝’20180003'；

C3：Sbirthdate>=’2000-1-1’；

C4：Smajor='计算机科学与技术' AND Sbirthdate>=’2000-1-1’；

1.选择操作的实现

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；

1.选择操作的实现

索引扫描算法

[例] SELECT *

FROM Student

WHERE Sno=‘20180003'

- 假设Sno上有索引
- 算法：
  - 使用索引(或散列)得到Sno为‘20180003’ 元组的指针
  - 通过元组指针在Student表中检索到该学生

1.选择操作的实现

[例] SELECT *

FROM Student

WHERE Sbirthdate>=’2000-1-1’

- 假设Sbirthdate上有B+树索引
- 算法：
  - 使用B+树索引找到Sbirthdate>=’2000-1-1’的第一个索引项，以此为入口点在B+树的顺序集上得到符合条件的所有元组指针；
  - 通过这些元组指针到Student表中检索到所有2000 年 1月 1日以后出生的学生。

1.选择操作的实现

[例] SELECT *

FROM Student

WHERE Smajor='计算机科学与技术' AND Sbirthdate>=’2000-1-1’;

- 假设Smajor和Sbirthdate上都有索引
- 算法：用Index Scan找到Smajor='计算机科学与技术'的一组元组指针和Sbirthdate>=’2000-1-1’的另一组元组指针
  - 求这两组指针的交集
  - 到Student表中检索
  - 得到满足条件的学生

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)最常用的实现算法

[例10.2]　针对如下SQL语句：

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

2.连接操作的实现

（1）嵌套循环算法(nested loop join)

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

（3）索引连接算法(index join)

（4）哈希连接算法 (hash join)

2.连接操作的实现

（1）嵌套循环算法(nested loop join)

- 由两层循环组成，其基本思想：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。

2.连接操作的实现

（2）排序-合并算法(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将排好序的 Student表和 SC 表中的数据块读到内存中；

③依次扫描 SC 表中与 Student 表具有相同 SNO 值的元组，把它们连接起来；

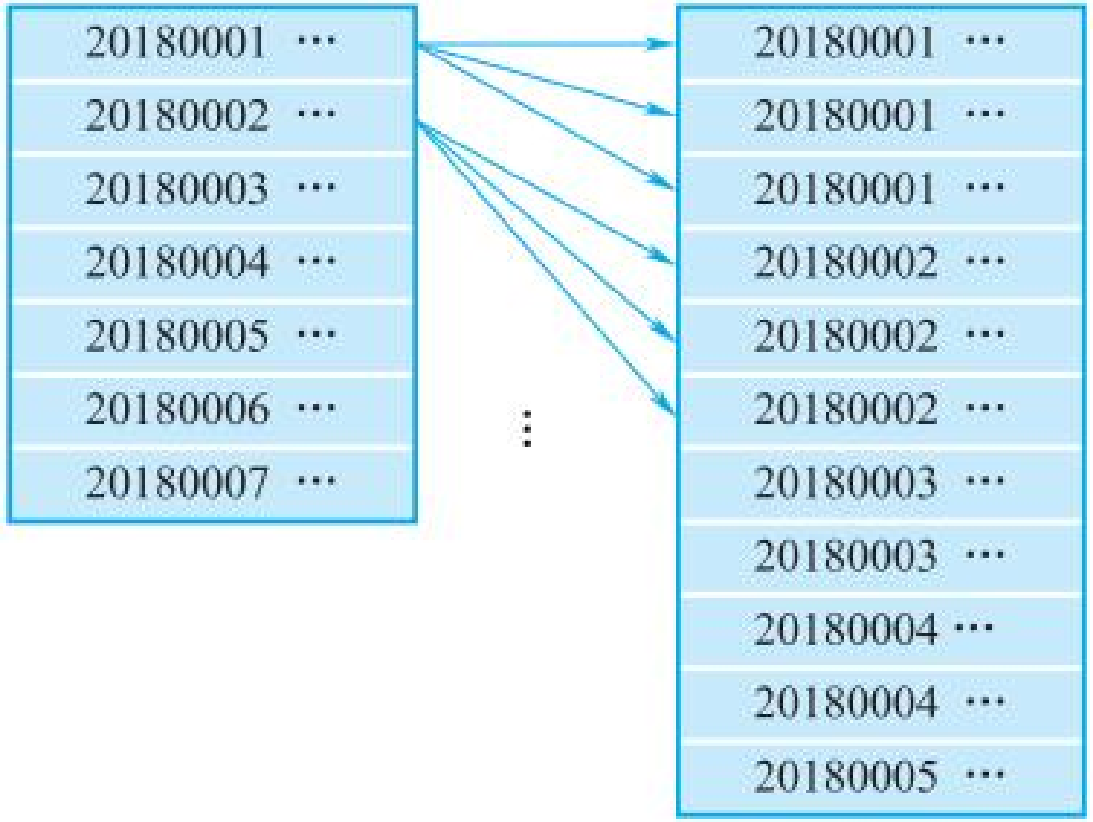
④当扫描到Sno不相同的第一个SC元组时，返回Student表扫描它的下一个元组，再扫描SC表中具有相同Sno的元组，把它们连接起来；

⑤在内存中的 Student 表或 SC 表扫描完后， 继续将相应表余下的数据块读入内存， 重复第③和第④步， 直到 Student 表或 SC 表全部扫描完。

2.连接操作的实现

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

- 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，即使加上排序后使用排序-合并连接算法的总执行时间一般仍会减少；



2.连接操作的实现

（3）索引连接(index join)算法

① 在SC表上建立属性Sno的索引，若已有Sno上的索引，则跳过①

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

2.连接操作的实现

（4）哈希连接（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)
  - 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)
  - 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，映射到相应的Student表哈希桶
  - 把SC元组与桶中来自Student表并与之相匹配的元组连接起来

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

第十章 查询处理和优化

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目 录

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主要因素；
- 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；

01

查询优化概述

SJTU

查询优化概述

- 关系系统的查询优化
  - 关系数据库管理系统实现的关键技术；
  - 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。

    （3）优化器可以考虑成百上千种不同的执行计划，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

- 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
  - 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

- 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
  - 尽量降低查询代价

02

一个实例

SJTU

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10.3] 求选修了81003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='81003'
```

- 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81003号课程的选课记录为50个

一个实例

- 可以用多种等价的关系代数表达式来完成这一查询

  - $Q_1=\pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='81003'}(Student \times SC))$
  - $Q_2=\pi_{Sname}(\sigma_{SC.Cno='81003'}(Student \bowtie SC))$
  - $Q_3=\pi_{Sname}(Student \bowtie \sigma_{SC.Cno='81003'}(SC))$

一个实例

1. 第一种情况

$$Q_1=\pi_{Sname}(\sigma_{Student.Sno=SC.Sno\wedge SC.Cno='81003'}(Student\times SC))$$

（1） 计算广义笛卡儿积

- 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；
- 把读入的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；
- 再从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

- 设一个数据块能装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：

$$\frac{1000}{10} + \frac{1000}{10 \times 5} \times \frac{10000}{100} = 100+20\times100 = 2100\text{块}$$

- 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

1. 第一种情况

$$Q_1=\pi_{Sname}(\sigma_{Student.Sno=SC.Sno\wedge SC.Cno='81003'}(Student\times SC))$$

（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
- 若满足条件的元组假设仅50个，均可放在内存

一个实例

1. 第一种情况

$$Q_1=\pi_{Sname}(\sigma_{Student.Sno=SC.Sno\wedge SC.Cno='81003'}(Student\times SC))$$

（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

一个实例

2. 第二种情况

$$Q_2 = \pi_{Sname}(\sigma_{Sc.Cno='81003'}(Student \bowtie SC))$$

（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
- 写出数据块= 10^3 块

一个实例

2. 第二种情况

$$Q_2 = \pi_{Sname}(\sigma_{Sc.Cno='81003'}(Student \bowtie SC))$$

（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。

- **第二种情况下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

3. 第三种情况

$$Q_3=\pi_{Sname}(Student \bowtie \sigma_{SC.Cno='81003'}(SC))$$

（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

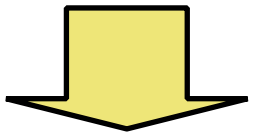
一个实例

- 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='81003'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

一个实例

- 把代数表达式Q1变换为Q2、 Q3

$$Q_1=\pi_{Sname}(\sigma_{Student.Sno=SC.Sno\wedge Sc.Cno='81003'}(Student\times SC))$$



$$Q_2=\pi_{Sname}(\sigma_{Sc.Cno='81003'}(Student \bowtie SC))$$

$$Q_3=\pi_{Sname}(Student \bowtie \sigma_{SC.Cno='81003'}(SC))$$

- 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。

一个实例

- 在Q3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。

第十章 查询处理和优化

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目 录

01

关系代数表达式等价变换规则

SJTU

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$E_1 \equiv E_2$

常用的等价变换规则

1. 连接运算、笛卡儿积运算交换律

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

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

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

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

常用的等价变换规则

2. 连接运算、笛卡儿积运算的结合律

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

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

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

$$(E_1 \underset{F_1}{\rhd\lhd} E_2) \underset{F_2}{\rhd\lhd} E_3 \equiv E_1 \underset{F_1}{\rhd\lhd} (E_2 \underset{F_2}{\rhd\lhd} E_3)$$

常用的等价变换规则

3. 投影运算的串接律

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

- E是关系代数表达式
- A_i (i=1，2，…，n)，B_j (j=1，2，…，m)是属性名
- $\{A_1, A_2, \ldots, A_n\}$ 是 $\{B_1, B_2, \ldots, B_m\}$ 的子集

常用的等价变换规则

4. 选择运算的串接律

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

- 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

常用的等价变换规则

5. 选择运算与投影运算的交换律

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

- 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有更一般规则

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

常用的等价变换规则

6. 选择运算与笛卡儿积运算的交换律

- 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儿积前先做。

常用的等价变换规则

7. 选择与并的分配律

设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

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

常用的等价变换规则

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

10. 投影运算与笛卡儿积运算的分配律

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

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

常用的等价变换规则

11. 投影运算与并运算的分配律

设E_1和E_2有相同的属性名，则

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

02

语法树的启发式优化

SJTU

语法树的启发式优化

- 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。

语法树的启发式优化

- 典型的启发式规则

（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儿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儿积省很多时间。

语法树的启发式优化

典型的启发式规则

（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一种公共子表达式。

语法树的启发式优化

- 遵循这些启发式规则，应用10.3.1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：

规则4：选择的串接定律

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

（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。

语法树的启发式优化

（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：

- 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

规则3：合并或分解投影运算
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

语法树的启发式优化

（5）把经过上述变换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儿积(×)，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

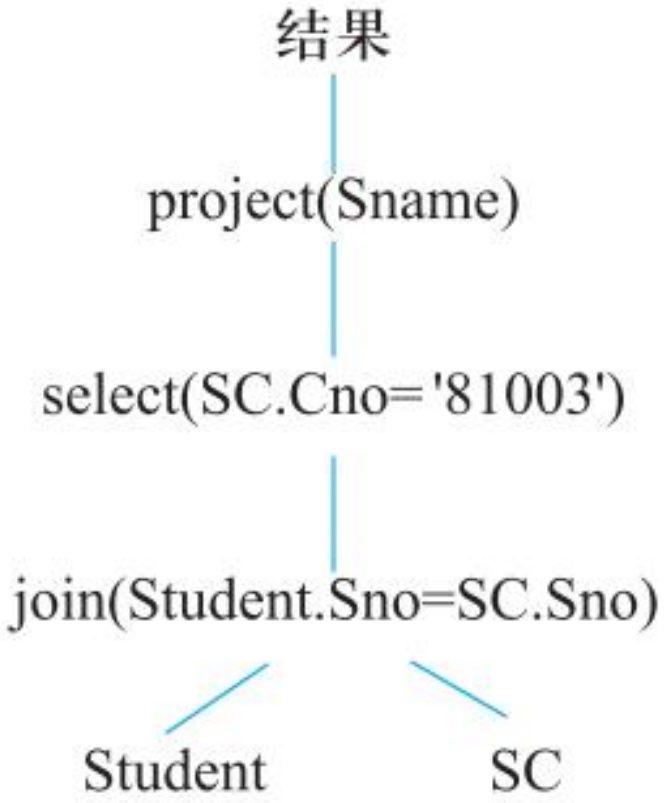
语法树的启发式优化示例

- [例]下面给出[例10.3]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：

[例10.3] 求选修了81003号课程的学生姓名。

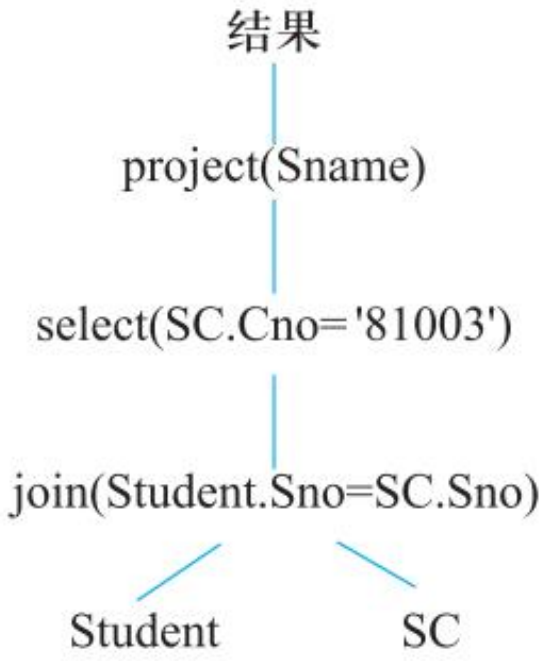
```
SELECT   Student.Sname
FROM     Student, SC
WHERE    Student.Sno=SC.Sno AND
         SC.Cno='81003'
```



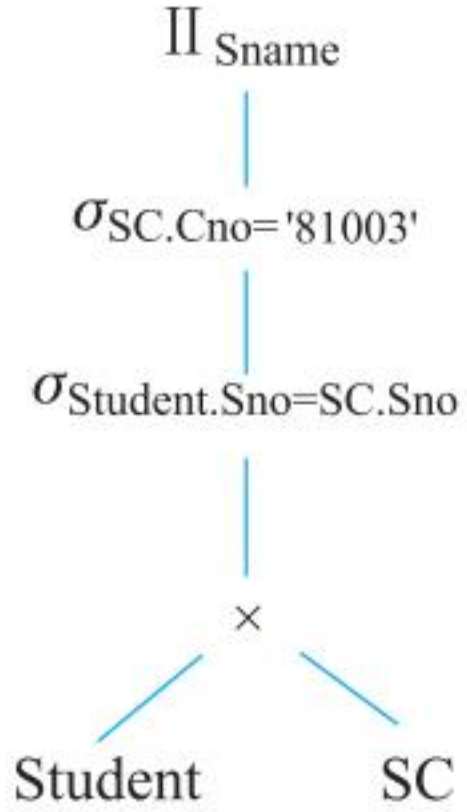
语法树图

语法树的启发式优化示例

- 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，则上面的语法树如图10.4所示。



语法树图

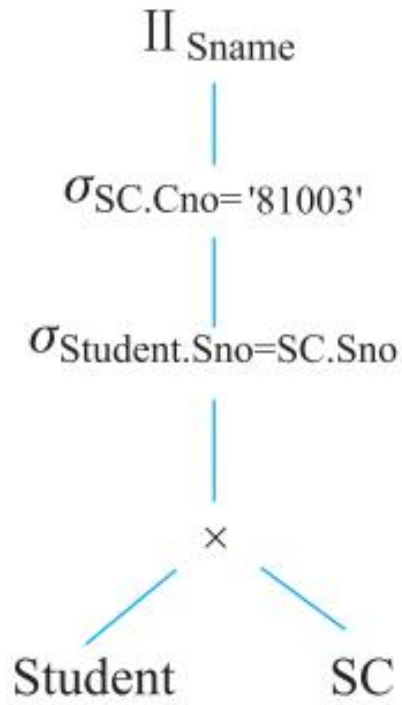


关系代数语法树

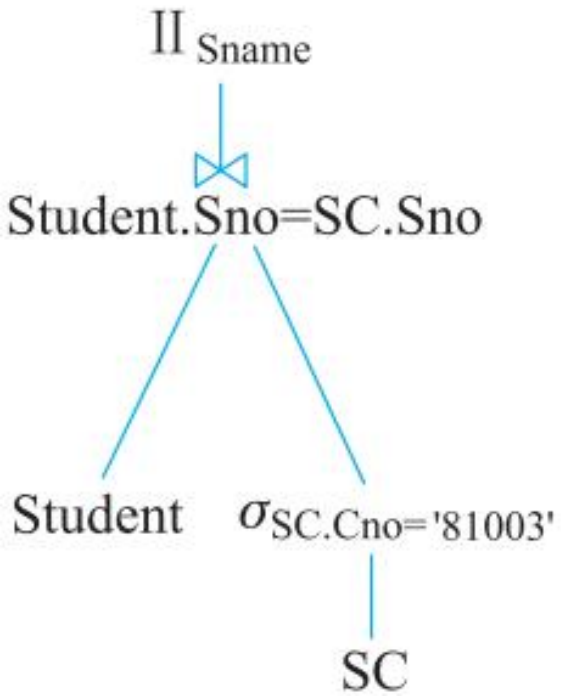
语法树的启发式优化示例

（2）对语法树进行优化

■ 利用规则4、6把选择$\sigma_{SC.Cno='81003'}$移到叶端，关系代数语法树转换成下图优化的语法树。



关系代数语法树



优化后的查询树

第十章 查询处理和优化

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目 录

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，求得优化查询计划；

物理优化

物理优化方法

- 基于启发式规则的启发式优化
  - 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。
- 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；

一、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

- 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，可以选择主码索引扫描
- 一般的关系数据库管理系统会自动建立主码索引

一、基于启发式规则的存取路径选择优化

（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

（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

一、基于启发式规则的存取路径选择优化

（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
  - 优先采用组合索引扫描方法
- 如果某些属性上有单属性索引，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）对于用OR连接的析取选择条件，一般使用全表顺序扫描

一、基于启发式规则的存取路径选择优化

2.连接操作的启发式规则

（1）如果两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

- 选用哈希连接算法

一、基于启发式规则的存取路径选择优化

（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：

- 设连接表R与S占用的块数分别为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Br+BrBs/(K-1)
- 显然应该选块数小的表作为外表

二、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

二、基于代价的优化

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，为此在数据字典中存储了优化器需要的数据库统计信息（database statistics information）
- 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)

二、基于代价的优化

（2）对基表的每个列

- 该列不同值的个数(m)
- 列最大值
- 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=具有该值的元组数/N

二、基于代价的优化

（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)

二、基于代价的优化

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

（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cost=L+1

二、基于代价的优化

（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，N/m是索引的选择基数(有N/m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+ N/m

二、基于代价的优化

（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
  - **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**

二、基于代价的优化

（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

  cost＝Br+BrBs/(K-1)

- 如果需要把连接结果写回磁盘

  cost＝Br+Br Bs/(K-1)+(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

二、基于代价的优化

（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
  - 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 ***4B***

第十章 查询处理和优化

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目 录

查询计划的执行

- 自顶向下
  - 系统反复向查询计划顶端的操作符发出需要查询结果元组的请求，操作符收到请求后，就试图计算下一个（几个）元组并返回这些元组。
  - 在计算时，如果操作符的输入缓冲区为空，它就会向其孩子操作符发送需求元组的请求。
  - 这种需求元组的请求一直传到叶子节点，启动叶子操作符运行，并返回其父操作符一个（几个）元组。
  - 父操作符再计算自己的输入返回给上层操作符，直至顶端操作符。
  - 重复这一过程，直到处理完整个关系。

查询计划的执行

- 自底向上
  - 查询计划从叶子节点开始执行，叶节点操作符不断地产生元组并将它们放入其输出缓冲区中，直到缓冲区填满为止，这时它必须等待其父操作符将元组从该缓冲区取走才能继续执行
  - 然后其父节点操作符开始执行，利用下层的输入元组来产生它自己的输出元组，直到输出缓冲区满为止
  - 这个过程不断重复，直到产生所有的输出元组

查询计划的执行

- 自顶向下的执行方式是一种被动的、需求驱动的执行方式
- 自底向上的执行方式是一种主动的执行方式

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

THANK YOU !

饮水思源 爱国荣校