第7章 查询处理与查询优化

- 7.1 引言
- 7.2 代数优化
- 7.3 物理优化

7.1 引言

- 查询是数据库系统中最基本、最常见和最复杂的操作
- 从用户给出的查询请求开始，直到得到查询结果，这一过程称为查询处理
- 对于一个给定的查询，通常会有许多种可能的执行策略，查询优化就是从众多策略中找出高效执行策略的处理过程
- 查询处理和优化是DBMS实现的关键技术，对系统性能有很大影响

查询处理

- 查询处理是指从数据库中查询数据的一个处理过程
- 这一过程包括将用户的查询语句转变为数据库的查询计划，并且执行计划获得查询结果

查询处理的步骤

- 实际系统对查询优化的具体实现一般可以归纳为四个步骤：

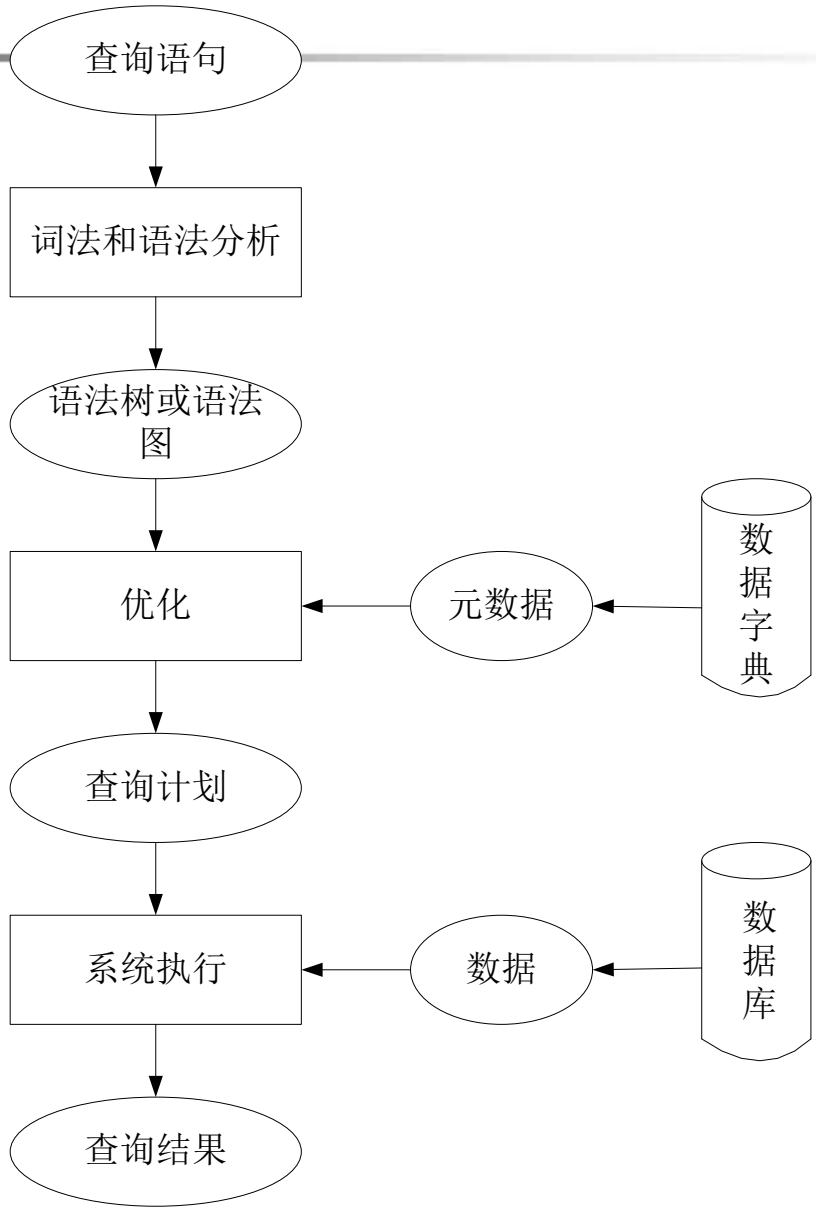
① 将查询转换成某种内部表示，通常是语法树

② 根据一定的等价变换规则把语法树转换成标准（优化）形式

③ 选择低层的操作算法。对于语法树中的每一个操作需要根据存取路径、数据的存储分布、存储数据的聚簇等信息来选择具体的执行算法

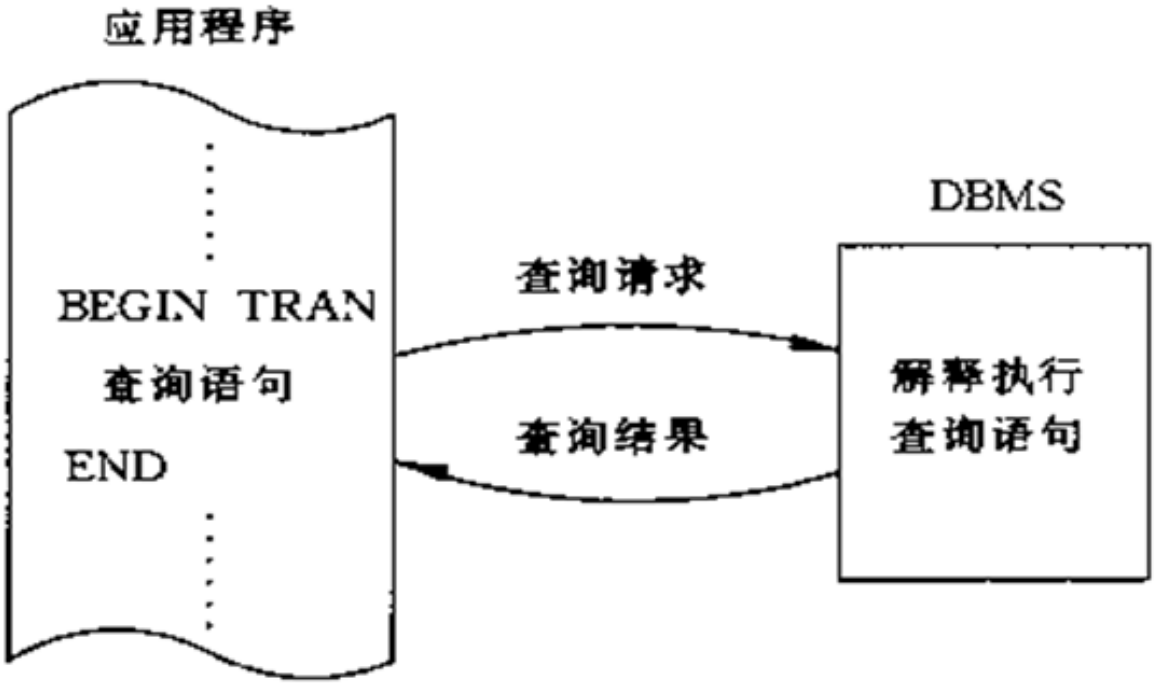
④ 生成查询计划。查询计划也称查询执行方案，由一系列内部操作组成。这些内部操作按一定的次序构成查询的一个执行方案。通常这样的执行方案有多个，需要对每个执行计划计算代价，从中选择代价最小的一个

查询处理的步骤



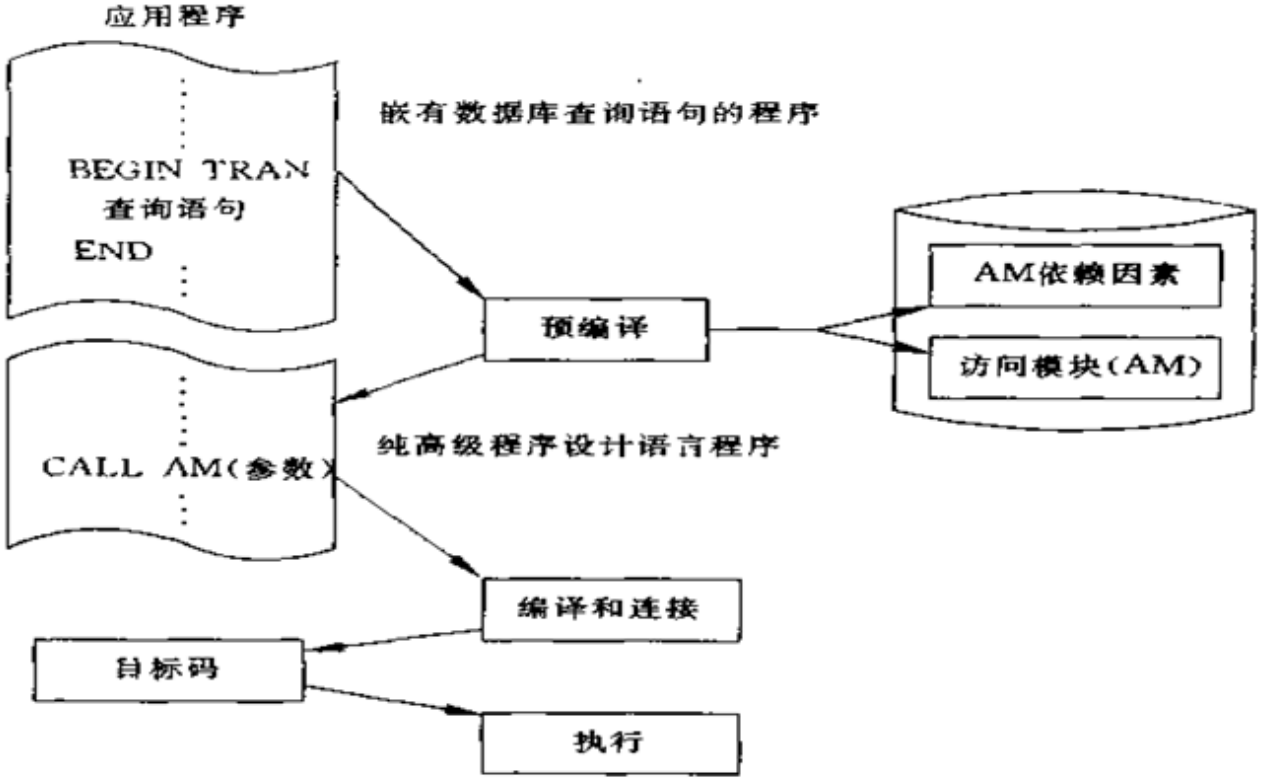
查询处理的实现方式

- 解释：DBMS接到查询语句，创建相应的事务，解释执行查询语句，且在事务完成后返回查询结果，一般不保留可执行代码



查询处理的实现方式

编译：应用程序经预编译处理，将嵌入的查询语句分出，进行语法、词法分析和查询优化，生成一个可执行的访问模块（AM），存于磁盘中。



查询优化

- 关系数据库系统的查询语言一般是“非过程语言”，它减轻了用户选择存取路径的负担。
- 即用户只要提出“做什么?”(What)，不必指出“怎么做”(How)。用户不必关心查询的具体执行过程
- 由DBMS确定合理的、有效的执行策略这方面的功能称为查询优化
- 对于使用非过程查询语言的RDBMS，查询优化是查询处理中非常重要的一环

查询优化的必要性

- 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
- 这是因为：
  1) 优化器可以从数据目录中获取许多统计信息
  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进行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) 优化器可以考虑数百种不同的执行计划，而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

查询优化的途径

1) 对查询语句进行等价变换（如改变基本操作的顺序）使查询执行起来更加有效。这种优化只涉及查询语句本身，而不涉及存取路径，故称为独立于存取路径的优化，或代数优化
2) 根据系统提供的查询路径，选择合理的存取策略（例如是选择顺序扫描还是选择索引），这称为依赖于存取路径的优化，或称物理优化
3) 有些查询可根据启发式规则选择执行策略，这称为规则优化
4) 根据可供选择的执行策略进行代价估算，并从中选择代价最小的执行策略，这称为代价估算优化
5) 此外，还可以通过应用数据库的语义信息对查询进行优化，这称为语义优化

查询处理的代价

- 在集中式数据库中，查询的执行代价主要包括
  - 总代价 = I/O代价 + CPU代价
- 在多用户环境下:
  - 总代价 = I/O代价 + CPU代价 + 内存代价
- 在网络分布环境下:
  - 总代价 = I/O代价 + CPU代价+网络传输代价

实例分析

- 例7-1：给出选修课程”CS-08” 的学生姓名。
- 用SQL语言表达：

```
SELECT S.Sname
FROM Student S，SC
WHERE S.S#=SC.S# AND C#='CS-08';
```

- 假定学生-课程数据库中有500个学生记录,l0 000个选课记录，其中选修课程” CS-08”的选课记录为50个。
- 系统可以用多种等价的关系代数表达式来完成这一查询

Q1= $\pi_{Sname}(\sigma_{Student.s\#=sc.s\#\wedge c\#='CS\text{-}08'}(Student \times SC))$

Q2= $\pi_{Sname}(\sigma_{c\#='CS\text{-}08'}(Student \bowtie SC))$

Q3= $\pi_{Sname}(Student \bowtie \sigma_{c\#='CS\text{-}08'}(SC))$

实例分析

- 查询执行策略Q1代价分析

1) 计算广义笛卡尔积的代价

- 把S和SC的每个元组连接起来。一般连接的做法是：在内存中尽可能多地装入某个表(如Student表)的若干块元组，留出一块存放另一个表(如SC表)的元组。然后把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，再从SC中读入一块和内存中的S元组连接，直到SC表处理完。这时再一次读入若干块S元组，读入一块SC元组，重复上述处理过程，直到把S表处理完
- 设一个块能装l0个Student元组或l00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：

  500/10+500/(10 X 5) X (10 000/100)=1050块
- 其中读Student表50块。读SC表10遍，每遍l00块。若每秒读写20块，则总计要花52.5(秒)
- 连接后的元组数为500X10 000。设每块能装l0个元组，则写出这些块要花500 000/20=25 000(秒)

实例分析

- 查询执行策略Q1代价分析(续)

2) 选择操作的代价

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。这一步读取中间文件花费的时间(同写中间文件一样)需25 000 秒。满足条件的元组假设仅50个，均可放在内存。

3) 投影操作的代价

把第2步的结果在Sname上作投影输出,得到最终结果

Q1查询的总时间约为52.5+2X25000秒

这里，所有内存处理时间均忽略不计

实例分析

- 查询执行策略Q2代价分析

1) 计算自然连接的代价

为了执行自然连接，读取Student和SC表的策略不变,总的读取块数仍为1050块，花费52.5秒。但自然连接的结果比第一种情况大大减少，为10000个。因此写出这些元组时间为10 000 /10/20=50秒。仅为第一种情况的千分之二

2) 读取中间文件块，执行选择运算，花费时间也为50秒

3) 将第2步结果投影输出

Q2总的执行时间≈52.5+50+50＝152.5秒

实例分析

- 查询执行策略Q3代价分析

1) 先对SC表作选择运算，只需读一遍SC表，存取l00块花费时间为5秒，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50块花费时间为2.5秒
3) 把连接结果投影输出

Q3总的执行时间≈5+2.5＝7.5秒

实例分析

- 假如SC表的C#字段上有索引，第1步就不必读取所有的SC元组而只需读取C#=‘CS-08’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2～3块
- 若STUDENT表在S#上也有索引，则第2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，因此读取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

实例分析

- 这个简单的例子充分说明了查询优化的必要性，同时也给了我们一些查询优化方法的初步概念。
- 如当有选择和连接操作时，应当先做选择操作，这样参加连接的元组就可以大大减少。

第7章 查询处理与查询优化

- 7.1 引言
- 7.2 代数优化
- 7.3 物理优化

7.2 代数优化

- 代数优化使用等价变换规则来优化关系代数表达式，使优化后的表达式满足基本原则
- 然后执行一系列已经证明能够优化的策略，而不必考虑实际数据的值以及数据库的存取路径，从而减少执行的开销

代数优化的基本原则

- 代数优化的基本原则是尽量减小查询处理的中间结果大小
- 投影、选择等一元操作对关系进行水平或垂直分割，减小了关系的大小
- 而并、连接等二元操作对两个关系按条件组合，操作费时且结果集比较大
- 在变换查询时，尽量先安排执行投影、选择，后进行连接、笛卡儿积

关系代数等价变换规则

- 关系代数表达式的优化是查询优化的基本课题
- 上面的代数优化原则大部分都涉及到代数表达式等价的变换
- 因此，研究关系代数表达式的优化应当从研究关系表达式的等价变换规则开始

关系代数等价变换规则

- 两个关系表达式El 和E2 是等价的，可记为E1≡E2
- 常用的等价变换规则有：

(1)选择的串接

$$\sigma_{F1}(\sigma_{F2}(R))\equiv\sigma_{F1\wedge F2}(R)$$

(2) 选择的交换

$$\sigma_{F1}(\sigma_{F2}(R))\equiv\sigma_{F2}(\sigma_{F1}(R))$$

关系代数等价变换规则

(3)投影的串接

$$\Pi_{L1}(\Pi_{L2}(\ldots\Pi_{Ln}(R)\ldots)) \equiv \Pi_{L1}(R)$$

条件：$L_1 \subseteq L_2 \subseteq \cdots \subseteq L_n$

(4)选择与投影的交换

$$\Pi_L(\sigma_F(R)) \equiv \sigma_F(\Pi_L(R))$$

条件：$Attr(F) \subseteq L$

(5)连接／笛卡儿积的交换律

$$R \bowtie S \equiv S \bowtie R$$

$$R \times S \equiv S \times R$$

关系代数等价变换规则

(6)选择对连接／笛卡儿积的分配律

$\sigma_F(R \bowtie S) \equiv (\sigma_F(R)) \bowtie S$

条件：$Attr(F) \cap Attr(S) = NULL$

$\sigma_{F1 \wedge F2}(R \bowtie S) \equiv \sigma_{F1}(R) \bowtie \sigma_{F2}(S)$

条件：$Attr(F_1) \cap Attr(S) = NULL$, $Attr(F_2) \cap Attr(R) = NULL$

(7)投影对连接／笛卡儿积的分配律

$\Pi_{L1 \cup L2}(R \underset{F}{\bowtie} S) \equiv \Pi_{L1}(R) \underset{F}{\bowtie} \Pi_{L2}(S)$

条件：$Attr(L_1) \cap Attr(S) = NULL$, $Attr(L_2) \cap Attr(R) = NULL$

$Attr(F) \subseteq Attr(L_1 \cup L_2)$

关系代数等价变换规则

(8)选择对∪、∩、－的分配律

$\sigma_F(R \cup S) \equiv \sigma_F(R) \cup \sigma_F(S)$

$\sigma_F(R \cap S) \equiv \sigma_F(R) \cap \sigma_F(S)$

$\sigma_F(R - S) \equiv \sigma_F(R) - \sigma_F(S)$

(9)投影对∪的分配律

$\Pi_L(R \cup S) \equiv \Pi_L(R) \cup \Pi_L(S)$

(10) ⋈ 、×、∪、∩的结合律

$(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$

代数优化的策略

(1)尽可能早地执行选择操作

(2)投影和选择操作同时进行

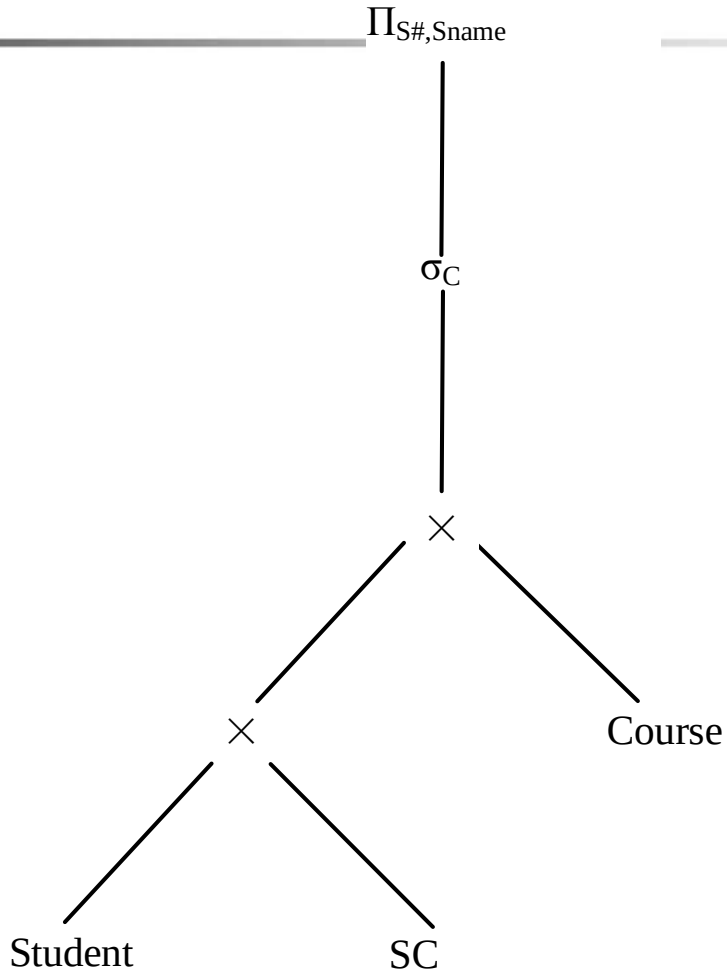
(3)选择和其前执行的笛卡儿积合并成一个连接操作

(4)找出表达式中公共子表达式

(5)投影和其前后的二元运算结合起来

(6)适当的预处理

语法树



C=(Sex='男'∧Cname='数据库系统'∧Grade='优'∧
Student.S# = SC.S#∧Course.C#=SC.C#)

关系代数表达式优化算法

算法：关系表达式的优化。
输入：一个关系表达式的语法树。
输出：计算该表达式的程序。

步骤：

1) 利用规则把形如 $\sigma_{F1\wedge F2\ldots\wedge Fn}(E)$的表达式变换为 $\sigma_{F1}(\sigma_{F2}(\ldots(\sigma_{Fn}(E))\ldots))$
2) 对每一个选择，利用规则尽可能把它移到树的叶端
3) 对每一个投影利用规则中的一般形式尽可能把它移向树的叶端。

关系代数表达式优化算法

4) 利用规则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，尽管这种变换以乎违背‘投影尽可能早做’的原则，但这样做效率更高

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(包括σ , Π运算)为一组。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(×)，而且其后的选择不能与它结合为等值连接时除外。把这些单目运算单独分为一组

6) 生成一个程序，每组结点的计算是程序中的一步。各步的顺序是任意的，只要保证任何一组的计算不会在它的后代组之前计算

实例分析

- 例7-2 设有Student（学生），Course（课程），SC（学生选课）三个关系，查询语句是“找出选修“数据库系统”课程并且成绩为优的男生学号和姓名”，用SQL语句表达如下：

```
SELECT   Student.S#,Sname
FROM     Student, Course, SC
WHERE Sex='男' AND Cname='数据库系统'
        AND Grade='优'  AND Student.S# = SC.S#
        AND  Course.C#=SC.C#
```

实例分析

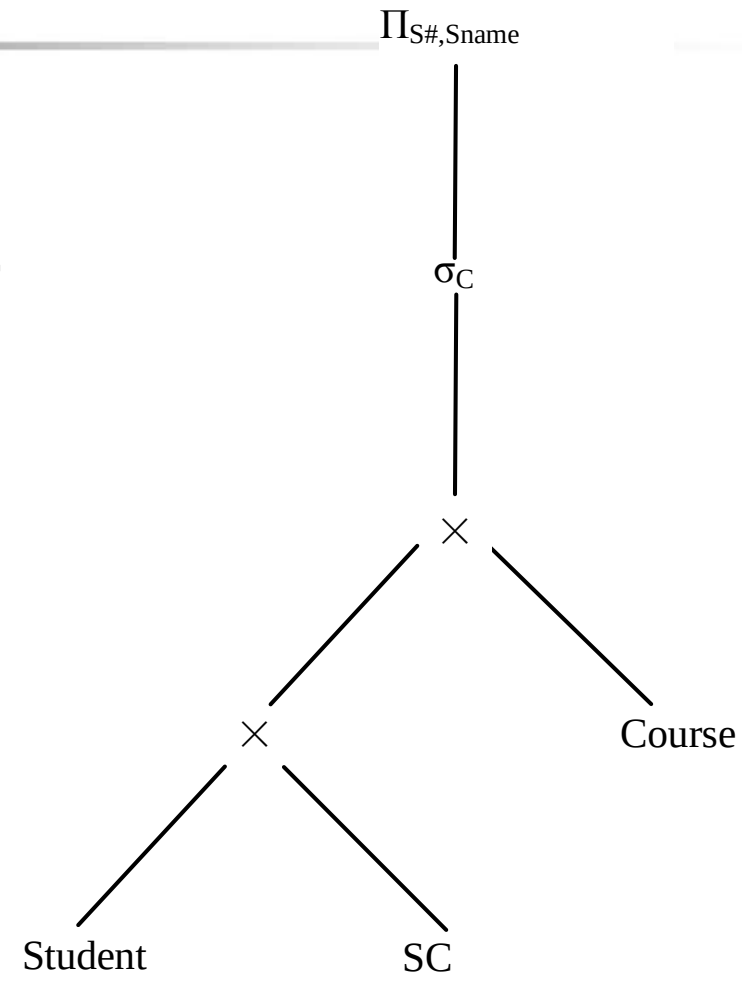
- 假设该查询变换成的关系代数表达式如下：

$$\Pi_{Student.S\#,Sname}(\sigma_{Sex='男' \wedge Cname='数据库系统' \wedge Grade='优' \wedge Student.S\# = SC.S\# \wedge Course.C\#=SC.C\#}(Student \times SC \times Course))$$

- 该表达式可转换为图7-2所示的原始查询树表示。以SELECT子句对应投影操作，FROM子句对应笛卡儿积操作，WHERE子句对应选择操作，生成原始查询树。

实例分析

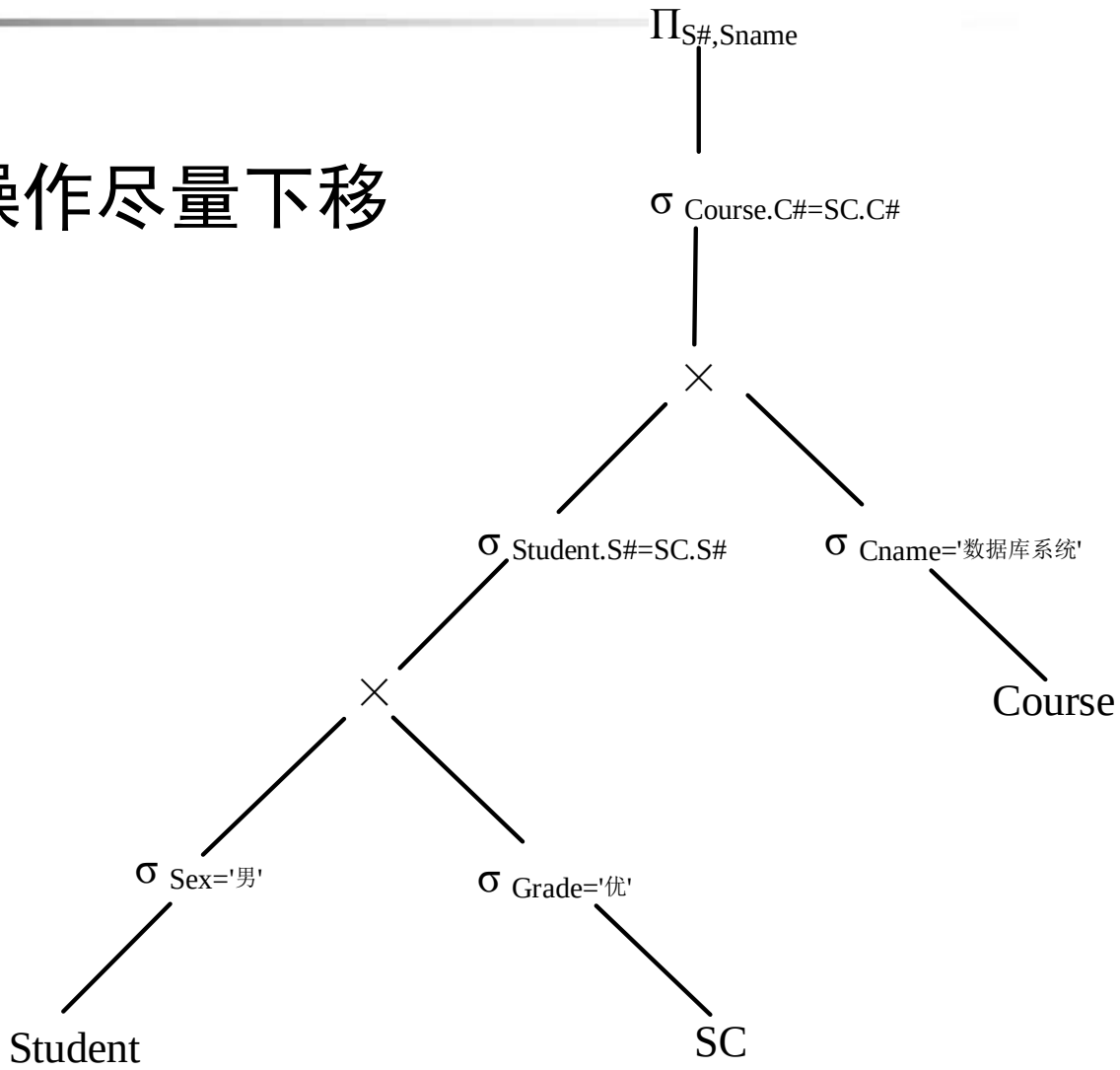
图7-2　原始查询树



C=(Sex='男'∧Cname='数据库系统'∧Grade='优'∧
Student.S# ＝ SC.S#∧Course.C#=SC.C#)

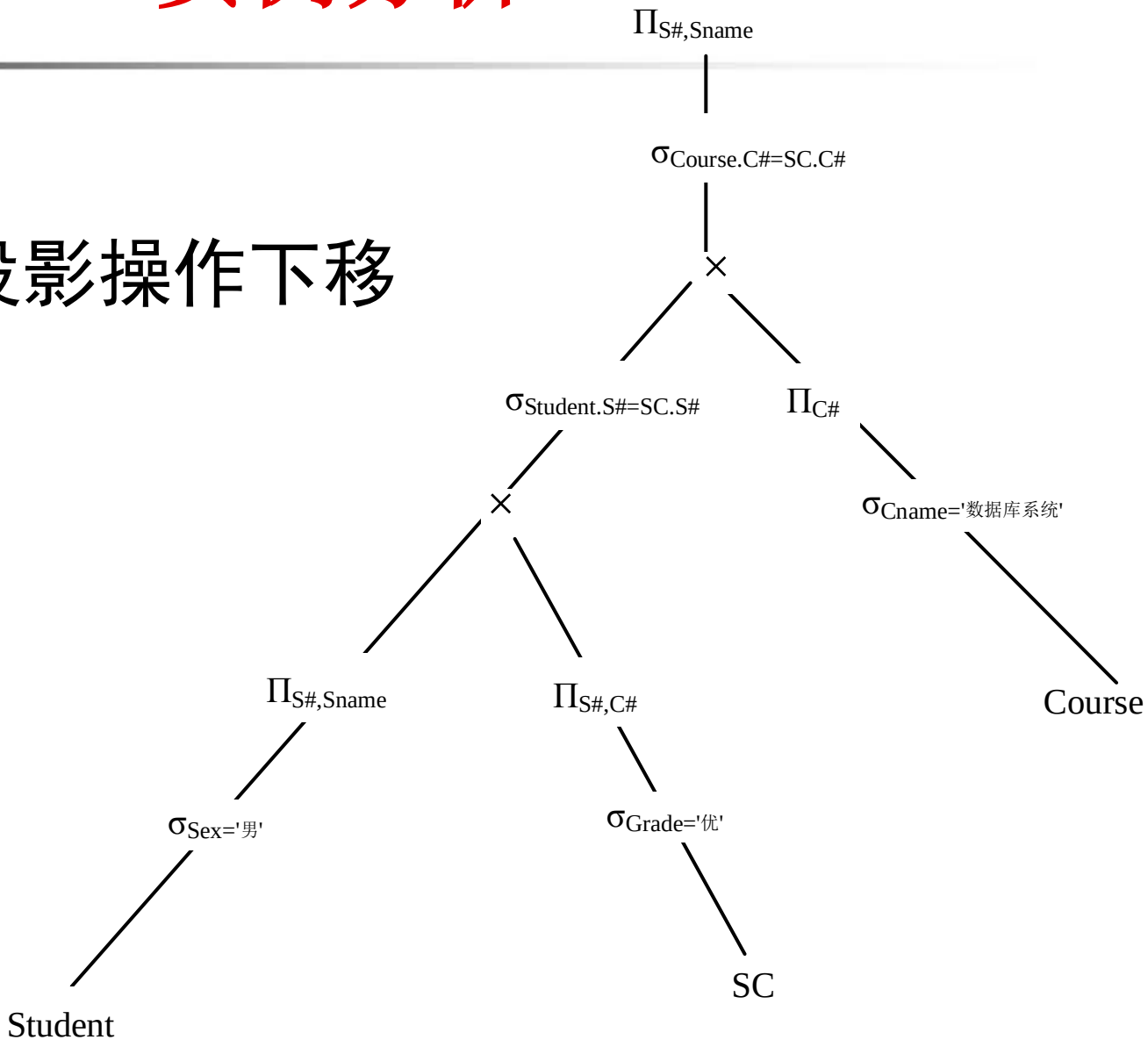
实例分析

图7-3　选择操作尽量下移



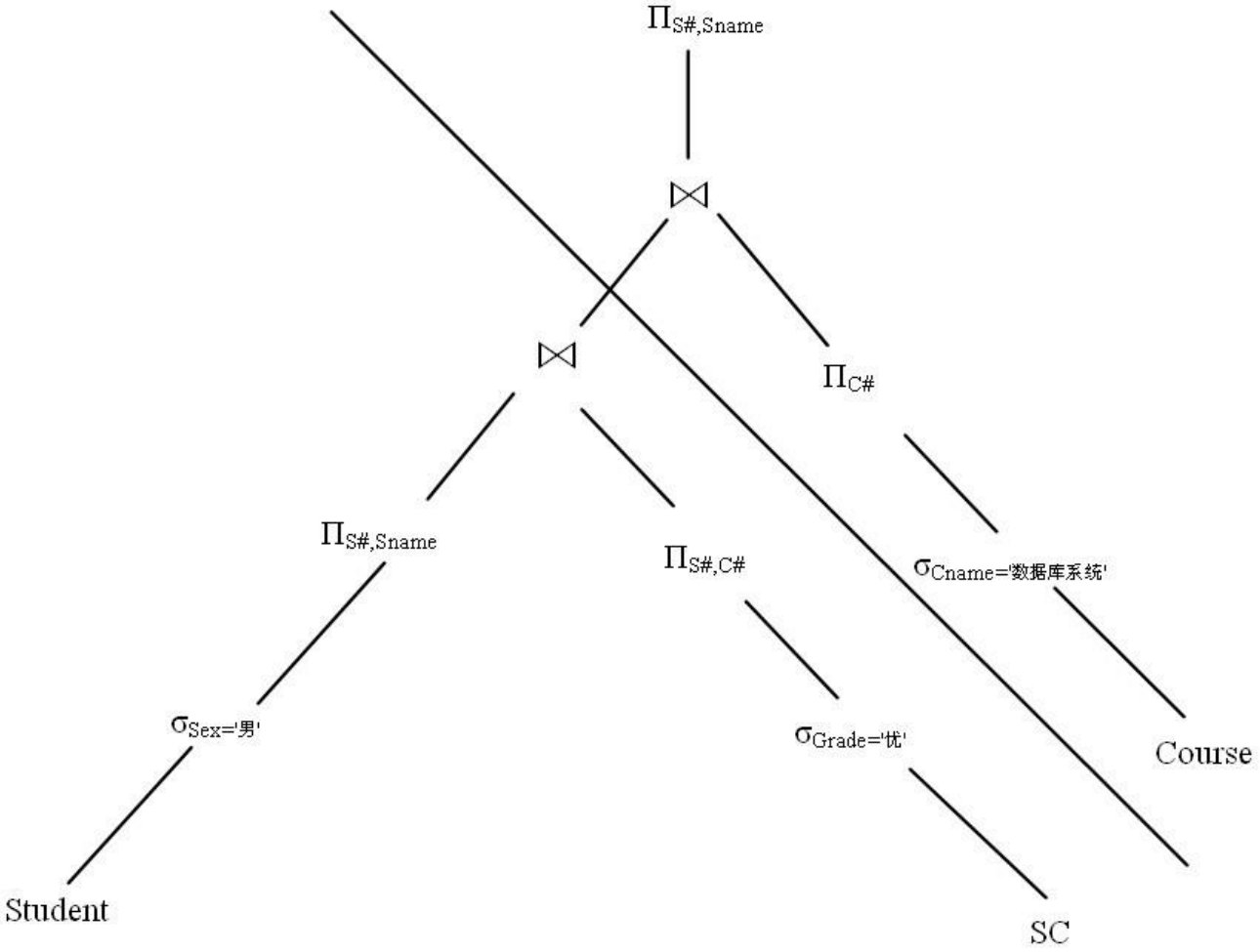
实例分析

图7-4 投影操作下移



实例分析

图7-5　连接条件和笛卡儿积组合成连接操作



第7章 查询处理与查询优化

- 7.1 引言
- 7.2 代数优化
- 7.3 物理优化

7.3 物理优化

- 根据系统所提供的存取路径，选择合理的存取策略(例如选用顺序搜索或索引进行查询)或称为物理优化
- 物理优化又称为依赖于存取路径的优化，而代数优化则属于不依赖于存取路径的优化

选择操作的实现和优化

- 选择操作是从关系中选择满足给定条件的元组，从水平方向减小关系的大小，查询优化中的一条基本原则就是选择尽量先做
- 其执行策略，与选择条件、可用的存取路径以及选取的元组数在整个关系中所占的比例有关，分成下列几种情况来实现

选择操作的实现和优化

- 选择条件可分为：
  - 等值(=)
  - 范围(<,<=,>,>=,Between)
  - 集合(IN)等
- 复合选择条件由简单选择条件通过NOT, AND、OR连接而成

选择操作的实现和优化

- 选择操作的实现方法包括：

  (1) 顺序扫描：适用于“小”的关系，满足条件的元组比例较大或无其他存取路径。

  (2) 利用各种存取路径：包括索引（B+树），动态散列

选择操作的启发式规则

(1) 对于小关系，不必考虑其他存取路径，直接用顺序扫描

(2) 如果无索引或散列等存取路径可用，或估计中选的元组数在关系中占有较大的比例 (例如大于15%)，且有关属性上无索引，则用顺序扫描

(3) 对于主键的等值条件查询，最多只有一个元组可以满足此条件，应优先采用主键的索引或散列

(4) 对于非主键的等值条件查询，要估计中选的元组数在关系中所占的比例。如果比例较小(例如小于15%)，可用索引，否则还是使用顺序扫描

选择操作的启发式规则

(5) 对于范围条件．一般先通过索引找到范围的边界，再通过索引的顺序集沿相应方向 搜索

(6) 对于用AND连接的合取选择条件，若有相应的多属性索引，则优先采用多属性索引

(7) 对于用OR连接的析取选择条件，只能按其中各个条件分别选出 一个元组集，再求这些元组集的并。众所周知，并是开销大的操作，而且在OR连接的诸条件中，只要有一个条件无合适的存取路径，就不得不采用顺序扫描来处理这种查询。因此，在查询 语句中，应尽量避免采用析取选择条件

(8) 有些选择操作只要访问索引就可得到结果，例如查询索引属性的最大值、最小值、平 均值等。在此情况下，应优先利用索引，避免访问数据

连接操作的实现和优化

- 连接是从两个关系的笛卡儿积中选择满足连接条件的元组，操作本身开销大，并且可能产生大的中间结果
- 为了更深入的了解连接操作的优化技术，下面对主要的几个算法分别进行简单描述，并分析各种算法的代价

连接操作的实现和优化

- 实现连接操作一般有4种方法：
  1) 嵌套循环法(nested loop)
  2) 利用索引或散列寻找匹配元组法
  3) 排序归并(sort-merge)法
  4) 散列连接法(hash join)

连接操作的实现和优化

1) 嵌套循环法(nested loop)

- 顺序扫描外关系的每一个元组，然后与内关系的每一个元组进行匹配。
- 设b_R，b_S分别表示R和S的物理块数，n_B为可用的内存缓冲块数，并以其中n_B-1块存放外关系，剩余的1块存放内关系。
  - ①若以R为外关系，S为内关系，用嵌套循环法进行连接需要访问的物理块数为：

    $$b_R+[b_R/(n_B-1)] \times b_S$$

  - ②若以S为外关系，R为内关系，用嵌套循环法进行连接需要访问的物理块数为：

    $$b_S+[b_S/(n_B-1)] \times b_R$$

- 比较上面2个式子，可以看出选择占用物理块少的关系作为外关系 。

连接操作的实现和优化

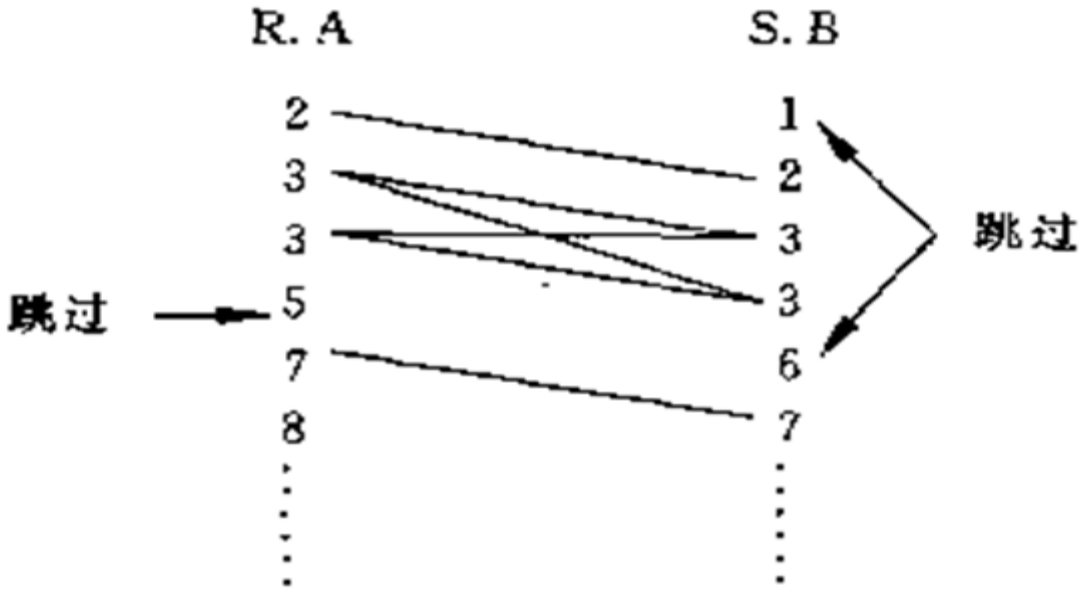
2) 利用索引或散列寻找匹配元组法

- 如果内关系有合适的存取路径（例如索引或散列），可以考虑使用这些存取路径代替顺序扫描。
- 可有效减少I/O次数

连接操作的实现和优化

3) 排序归并(sort-merge)法

- 首先按连接属性对关系排序，然后进行归并连接



连接操作的实现和优化

4) 散列连接法(hash join)

- 首先用散列函数将连接属性散列至文件中，然后对散列到同一个桶中的元组进行匹配。

连接的启发式规则

(1) 如果两个关系都已按连接属性排序，则优先用排序归并法。如果两个关系中已有一个关系按连接属性排序，另一个关系很小，可考虑对此关系按连接属性排序，再用排序归并法连接。

(2) 如果两个关系中有一个关系在连接属性上有索引(特别是顺序索引)或散列，则可令另一关系为外关系，顺序扫描，并利用内关系上的索引或散列寻找其匹配元组，以代替多遍扫描。

(3) 如果应用上述两规则的条件都不具备，且两个关系都比较小，可以应用嵌套循环法 。

(4) 如果(1)、(2)、(3)规则都不适用，可用散列连接法。

投影操作的实现

- 投影操作选取关系的某些列，从垂直的方向减小关系的大小
- 投影操作一般可与选择、连接等操作同时进行，不再需要附加的I/O开销。
- 但是，投影结果可能会出现重复元组，应消除这些完全相同的元组

投影操作的实现

- 如果关系已经经过排序，等值的元组相邻存储，很容易只留一个元组副本去掉多余的副本
- 也可使用散列方法来实现消除重复元组，整个关系按建立在某一属性或多个属性上的散列函数进行划分

集合操作的实现

- 对于笛卡儿积(×)一般可采用嵌套循环法；
- 对于∪、∩、－等操作需要发现共同元组 ；