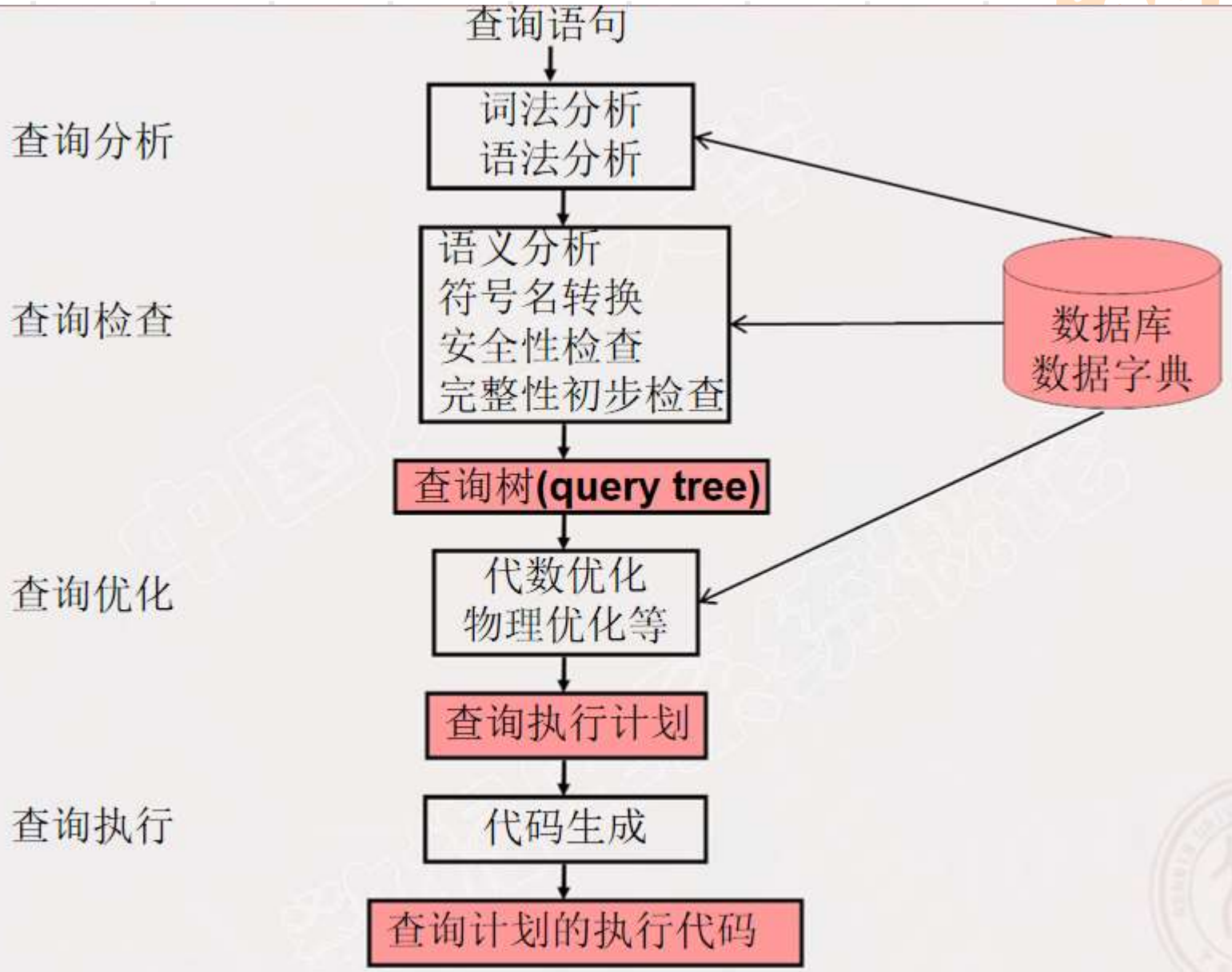
第7章 查询处理与优化

- 7.1 概述
- 7.2 代数优化
- 7.3 物理优化
- 7.4 代价估算优化*

7.1 概述

- 查询(Query)是数据库系统中最基本、最常见和最复杂的操作。
- 从用户给出查询请求开始，到得出查询结果的过程称为查询处理。
- 查询优化是从众多策略中找出高效执行策略的过程。
- 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。

查询处理的步骤


查询语句
查询分析
词法分析
语法分析
查询检查
语义分析
符号名转换
安全性检查
完整性初步检查
数据库
数据字典
查询树(query tree)
查询优化
代数优化
物理优化等
查询执行计划
查询执行
代码生成
查询计划的执行代码

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。
- 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/逻辑优化：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。
- 两种执行方法
  - 自顶向下（拉）
  - 自底向上（推）

查询处理的实现方式

- 解释执行：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。

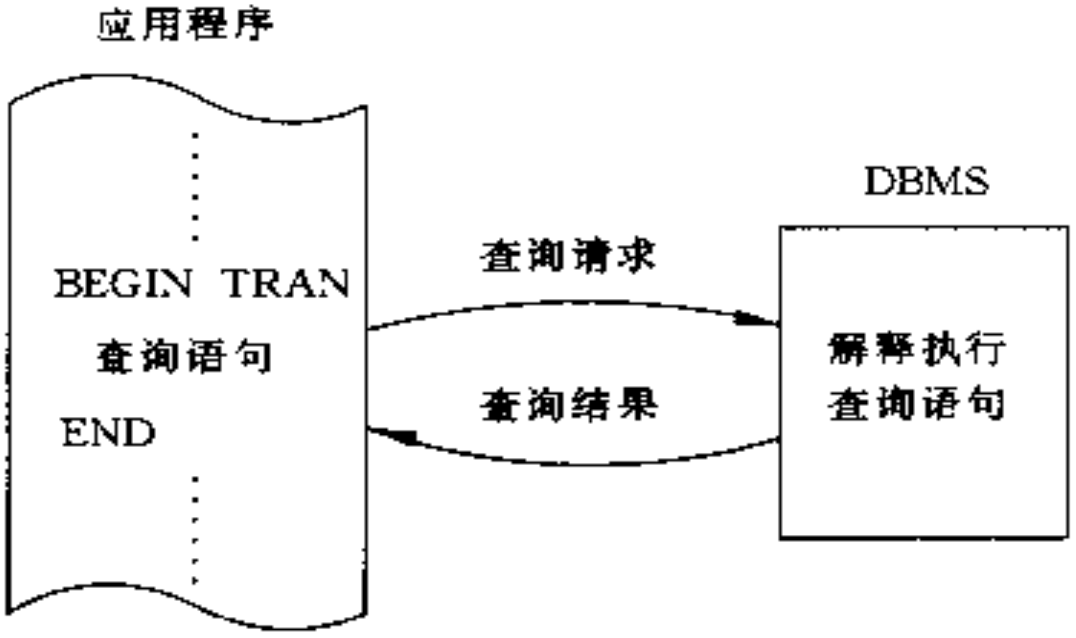


图 6-1　查询语句的解释执行

查询处理的实现方式

- 预编译：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。

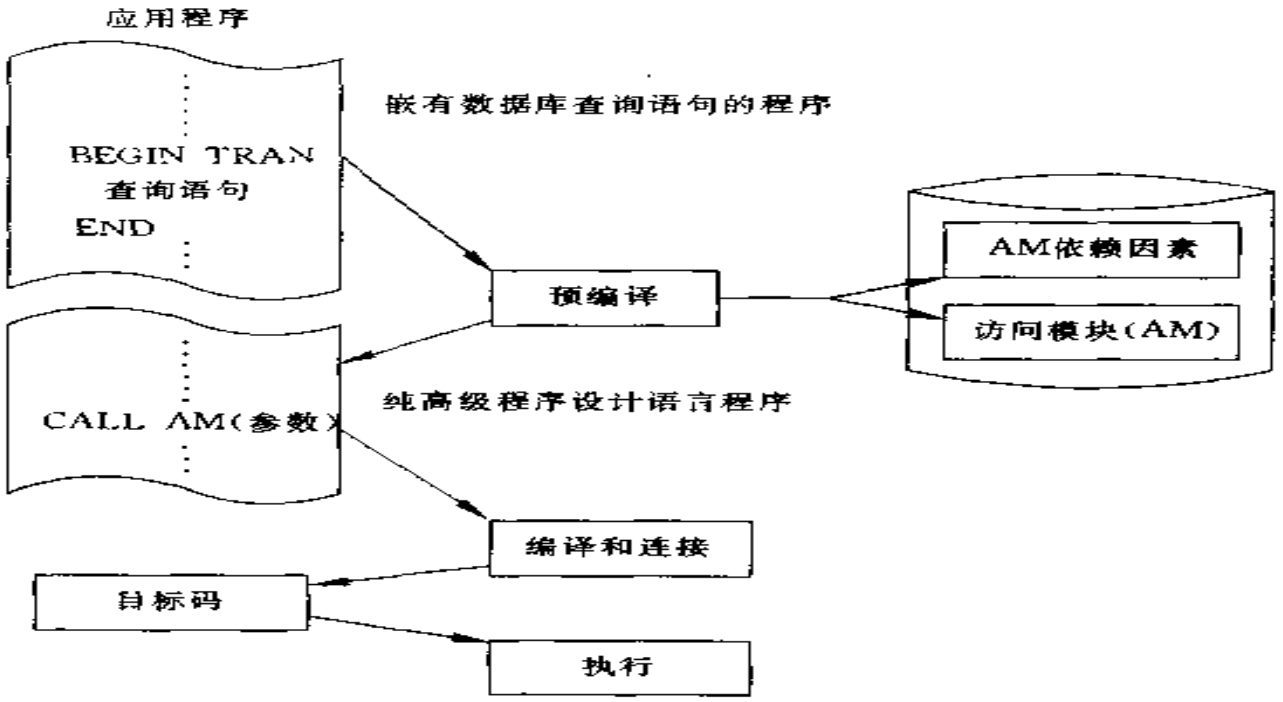


图 6-2 查询语句的编译

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。

关系数据库系统的查询优化

- 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。
- 其优点是极大减轻了用户选择存取路径的负担。

关系数据库系统的查询优化

- 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。

关系数据库系统的查询优化

- 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，以及操作的序列由用户决定。
  - 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，系统无法对此加以改进。

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) 优化器可以从数据字典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：求选修了2号课程的学生姓名。用SQL语言表达：

```
SELECT S.Sname
FROM Student S，SC
WHERE S.SNO=SC.SNO AND SC.CNO='2'；
```

- 假定学生-课程数据库中有l000个学生记录,l0000个选课记录，其中选修2号课程的选课记录为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= $\pi_{Sname}(\sigma_{Student.sno=sc.sno \wedge sc.cno='2'}(S \times SC))$

Q2= $\pi_{Sname}(\sigma_{sc.cno='2'}(S \mid\!><\!\mid SC))$

Q3= $\pi_{Sname}(S \mid\!><\!\mid \sigma_{sc.cno='2'}(SC))$

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人某个表(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，量复上述处理过程，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：
- 1000/10+1000/(10 X 5) X (10000/100)=2100块
- 其中读Student表l00块。读SC表20遍，每遍l00块。若每秒读写20块，则总计要花105(秒)
- 连接后的元组数为1000X10000, 设每块能装l0个元组，则写出这些块要花1000000/20=50000(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50000 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105+2X5X10000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2100块，花费l05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 /10/20=50秒,仅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50秒

3)将第2步结果投影输出

Q2总的执行时间≈105+50+50＝205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l00块花费时间为5秒
3) 把连接结果投影输出

Q3总的执行时间≈5+5＝10秒

实例分析

- 假如SC表的Cno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，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。

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原来的查询
- 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

- 如果两个关系表达式El 和E2在任何数据库实例上的求值结果都是相同的，则称El 和E2相互等价，可记为：E1≡E2
- 常用的等价变换规则有：

(1)选择的串接律：

$$\sigma_{C1}(\sigma_{C2}(R)) \equiv \sigma_{C1 \wedge C2}(R)$$

(2) 选择的交换律：

$$\sigma_{C1}(\sigma_{C2}(R)) \equiv \sigma_{C2}(\sigma_{C1}(R))$$

关系代数等价变换规则

(3)投影的串接律：

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

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

(4)选择与投影的交换律：

$\Pi_L(\sigma_C(R)) \equiv \sigma_C(\Pi_L(R))$

条件：$Attr(C) \subseteq L$

(5)连接／笛卡儿积的交换律：

$R |><| S \equiv S |><| R$

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

关系代数等价变换规则

(6)选择对连接／笛卡儿积的分配律：

$\sigma_C(R|><|S) \equiv (\sigma_C(R))|><|S$

条件：Attr(C)∩Attr(S)=NULL

$\sigma_{C1 \wedge C2}(R|><|S) \equiv \sigma_{C1}(R)|><|\sigma_{C2}(S)$

条件： $Attr(C_1) \cap Attr(S)=NULL$, $Attr(C_2) \cap Attr(R)=NULL$

(7)投影对连接／笛卡儿积的分配律：

$\Pi_{L1 \cup L2}(R|><|_C S) \equiv \Pi_{L1}(R)|><|_C \Pi_{L2}(S)$

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

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

关系代数等价变换规则

(8)选择对∪、∩、－的分配律：

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

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

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

(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：

$(R |><| S) |><| T \equiv R |><| (S |><| T)$

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；

关系代数表达式优化算法

- 算法7-1：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(1)把形如 $\sigma_{C1 \wedge C2 \ldots \wedge Cn}(E)$ 的表达式变换为 $\sigma_{C1}(\sigma_{C2}(\ldots(\sigma_{Cn}(E))\ldots))$
2) 对每一个选择，利用规则(4)－(8)尽可能把它移到树的叶端(下推σ)

关系代数表达式优化算法

3) 对每一个投影利用规则(3)，(9)，(l0)，(5)中的一般形式尽可能把它移向树的叶端。

4) 利用规则(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，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Shop(商店)，Customer(顾客)，SC(购物关系)三个关系，查询语句是“找出西安销售给顾客“张立”彩电的商店编号和名称”，用SQL语句表达如下：

```
SELECT  S#, Sname
FROM    Shop S, Customer C, SC
WHERE S.Address='西安'  AND Cname='张立'
AND  Item='彩电'   AND  S.S# = SC.S#  AND
C.C#=SC.C#;
```

实例分析

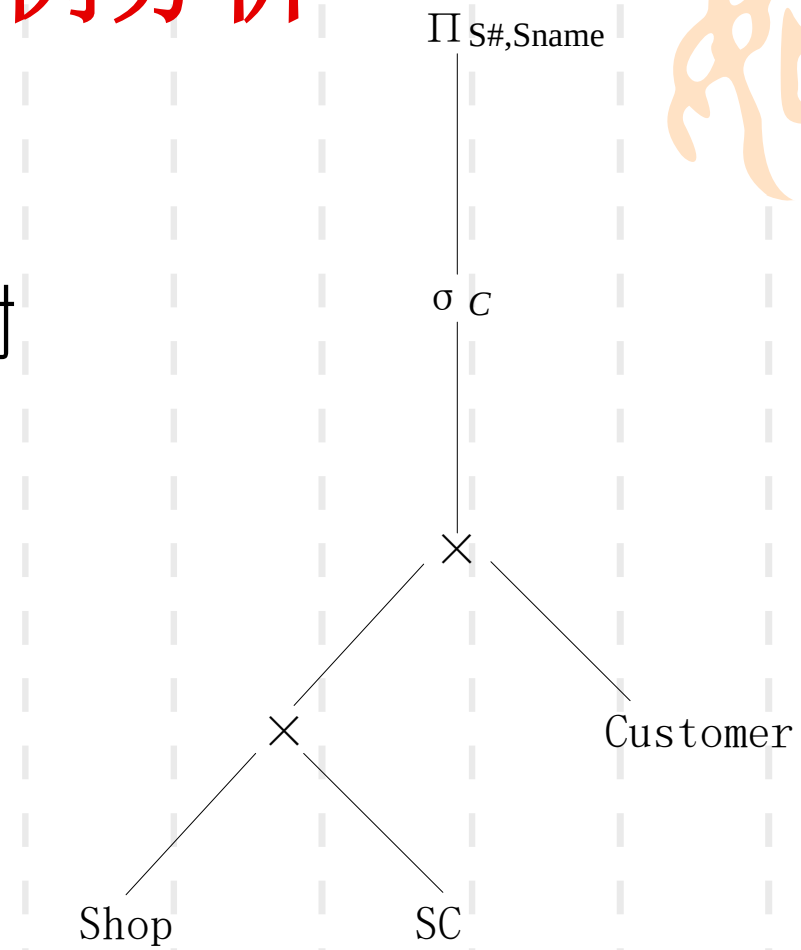
- 假设该查询变换成的关系代数表达式如下：

$$\Pi_{S\#,Sname}(\sigma_{Shop.Address='西安' \wedge Cname='张立' \wedge Item='彩电' \wedge Shop.S\# = SC.S\# \wedge Customer.C\#=SC.C\#}(Shop \times SC \times Customer))$$

- 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

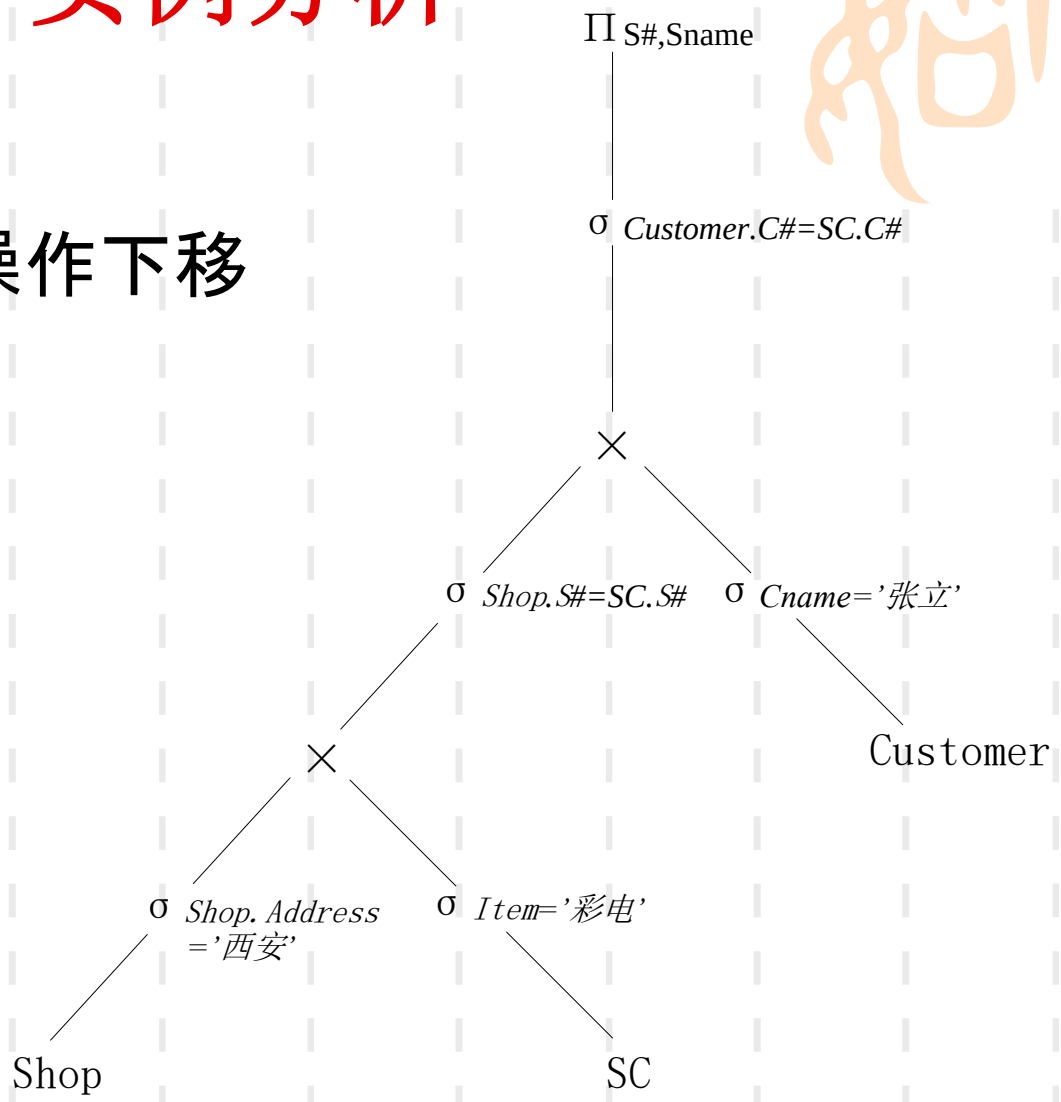
图7-2　原始查询树



C=*Shop.Address*='西安' ∧ *Cname*='张立' ∧ *Item*='彩电' ∧ *Shop.S#* = *SC.S#* ∧ *Customer.C#=SC.C#*

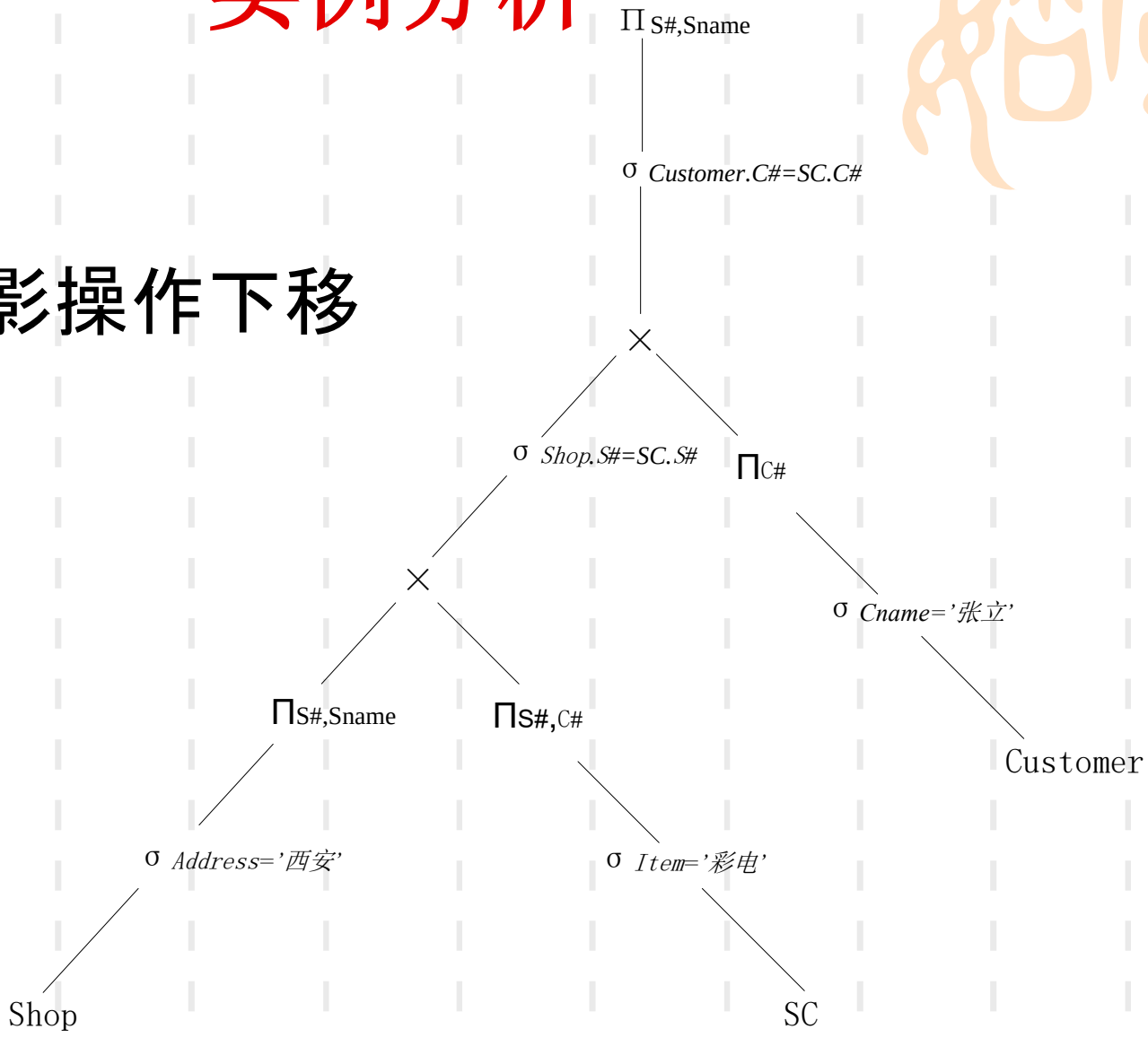
实例分析

图7-3　选择操作下移



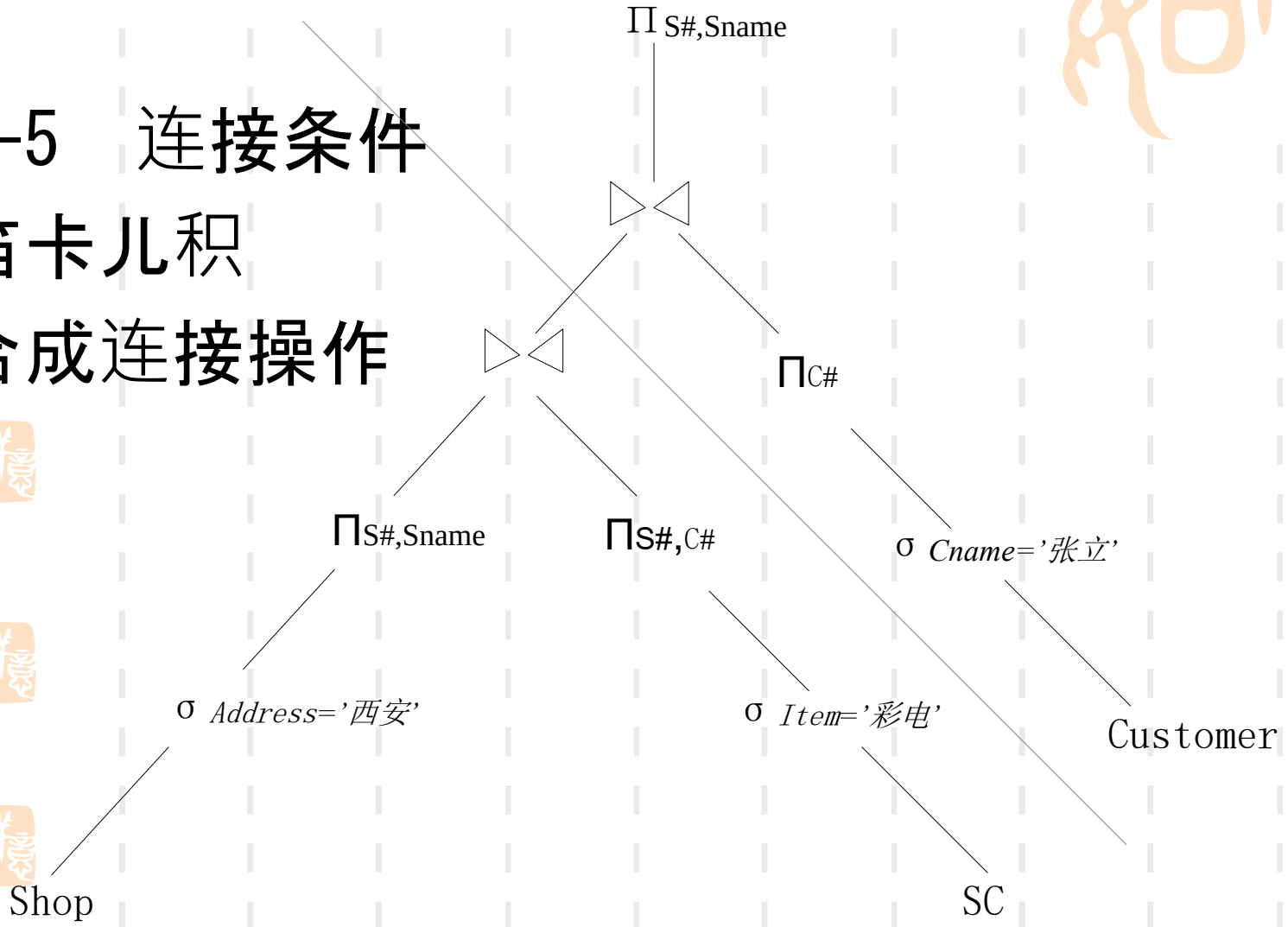
实例分析

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



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)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%)，且有关属性上无族集索引，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%)，可用无序索引，否则只能用族集索引或顺序扫描。

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(nested loop)

② 利用索引或散列寻找匹配元组法

③ 排序归并(sort-merge)

④ 散列连接(hash join)

连接操作的实现和优化

1) 嵌套循环(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，可以看出应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)

- 首先按连接属性对关系排序，然后进行归并连接

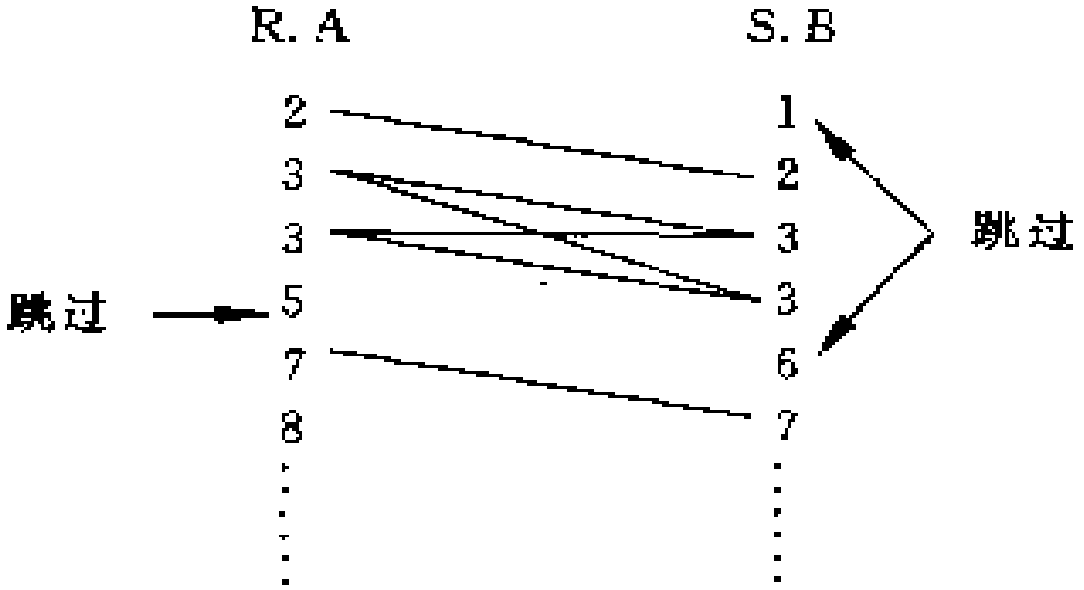


图 6-5　排序归并法示意图

连接操作的实现和优化

1) 散列连接(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(特别是簇集索引)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。
- 也可使用和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；

7.4 代价估算优化

- 代价估算优化：对可供选择的执行策略进行代价估算，从中选用代价最小的执行策略。
- 在小型和解释执行的DBMS中，规则优化用得比较多。因为在解释执行的数据库系统中，优化时间包含在事务的执行时间中，不宜采用开销太大的优化方法。
- 在编译实现的数据库系统中，一次编译，可供多次执行，编译时间不包括在事务的执行时间中，值得采用较复杂的优化方法。
- 一般先用规则优化，选择几个可取的执行策略，然后对它们进行代价比较，从中择优。

查询执行代价的组成与代价统计参数

- 查询执行代价主要包括3个方面：
  1) I/O代价(*)
  2)CPU代价
  3)通信代价
- 访问磁盘1次所需的代价可表示为：

$$C_{I/O}=D0 + x * D1$$

- 其中：x :存取数据的大小，以字节表示

  D0:与x无关的I/O代价，包括寻道时间和等待时间

  D1:每个字节所需的传输时间

  一般D0>> XD1 故: I/O代价=I/O次数 X D0

查询执行代价的组成与代价统计参数

- 下面给出进行代价估算时将要用到的统计参数:

n_R：R中的元组数；

P_R：R块因子，即每个物理块中包含的元组数；

N_A：属性A在一个关系中出现的不同值的个数；

F_A：属性A的选择因子，即属性A为某一个值的概率，一般假定属性值均匀分布，$F_A=1/N_A$；

M_A：属性A的值域大小|DOM(A)|；

Li：索引i的级数；

选择操作的代价估算

(1) 顺序扫描

- 最多选取一个元组的I/O代价：

$$Csa=0.5(n_R/p)=0.5\ b_R$$

- 选取多个元组的I/O代价：

$$Csa = b_R$$

(2) 利用主键上的索引或散列进行等值查询

- 通过索引访问的I/O代价：$C_{Ik} = L+1$
- 通过散列访问的I/O代价：$C_{hk} = 1$ (假定散列没有溢出)

选择操作的代价估算

(3) 利用非主键上的无序索引进行等值查询

- 分析表明几乎每取一个元组都需要访问一个物理块，故:

$$C_{INK} = L + s$$

其中 s 为满足选择条件的元组数

(4) 利用聚簇索引进行等值查询

- $C_{CI} = L + [s/p]$

(5) 利用聚簇索引进行范围查询

- $C_{CIR} = L + [b/2]$

选择操作的代价估算

- 例：设有关系STUDENT，其统计数据及存取路径如下：

  Sn=10000 Sb=2000 即 p=5

- 在属性DEPT上建有聚簇索引：N_{DEPT} = 25， L=2
- 在属性SNO上建有主键索引：N_{SNO} = 10000， L=4
- 在属性DNO上建有辅助索引：N_{DNO} = 20， L=2
- 在属性AGE上建有辅助索引：N_{AGE} = 15， L=2
- 设有下列查询：

  Q1：$\sigma_{SNO='992311'}(STUDENT)$

  Q2：$\sigma_{DEPT='CS'}(STUDENT)$

  Q3：$\sigma_{AGE>=20}(STUDENT)$

  Q4：$\sigma_{DEPT='CS'\ AND\ DNO='108'\ AND\ AGE>=20}(STUDENT)$

- 试用代价估算优化选取存取策略，并估算其执行代价

选择操作的代价估算

解： **Q1：$\sigma_{SNO='992311'}(STUDENT)$**

由于SNO上建有主索引，应优先采用主索引，其执行代价可估算为：

$$C_{Q1}=L+1=4+1=5$$

Q2：$\sigma_{DEPT='CS'}(STUDENT)$

由于DEPT上建有聚簇索引，故可不考虑顺序扫描，满足Q2的元组数s估计为：

$$s=10000/25=400$$

由于STUDENT关系对DEPT是聚簇的，故I/O代价可估算为：

$$C_{Q2}=L+[s/p]=2+[400/5]=82$$

选择操作的代价估算

解：Q3：$\sigma_{AGE>=20}(STUDENT)$

是范围查询，虽然在AGE上建有辅助索引，但不是聚簇索引，如果取一半元组，则使用索引还不如使用顺序扫描，故：

$$C_{Q3} = b = 2000$$

Q4：$\sigma_{DEPT='CS'AND\ DNO='108'\ AND\ AGE>=20}(STUDENT)$

查询条件是合取式，由于没有适当的多属性索引可用，只有2种可能的策略：

选择操作的代价估算

策略1：预查询法

- 满足条件 DNO='108' 的元组数估算为： $n/N_{DNO} = 10000/20 = 500$。设顺序集每块可容纳200个tid，则从DNO辅助索引的顺序集中取500个tid所需的I/O代价为：$C_{DNO} = L + [500/200\ ] = 3+3 =6$
- 满足条件 AGE>= 20 的元组数估算为：$n/2 = 10000/2 = 5000$则从AGE辅助索引的顺序集中取5000个tid所需的I/O代价为：$C_{AGE} = L + [5000/200\ ] = 2+25 = 27$
- 2个tid集的交集的大小应小于或等于500。由于取500个随机存放的元组一般需要500次I/O，故预查询法的I/O代价估算为：$Ca = C_{DNO} + C_{AGE} + 500 = 533$

选择操作的代价估算

策略2：用其中代价最小的一个条件选出元组，再用其他条件对这些元组进行筛选

- 应从3个条件中选出I/O代价最小的条件：

$\sigma_{DEPT='CS'}$：同Q2，CQ2 = 82

$\sigma_{DNO='108'}$：s = 10000/20= 500

$C_{DNO='108'}$：C_{DNO}+500 = 6 + 500=506

$\sigma_{AGE>=20}$：同Q3，CQ3 = 2000

- 在3个条件中，以$\sigma_{DEPT='CS'}$的代价最小，故可先按此条件选取出满足条件DEPT='CS'的学生的元组并同时检查每个元组是否满足其他2个条件，其I/O代价为Cb = 82
- 由于Ca>Cb，故C_{Q4}=Cb=82

连接操作的代价估算

(1)连接结果大小的估算

- 为估算连接操作的代价，首先需要估算连接结果的大小。为此，引入连接选择因子(join selectivity)

  $js = |R|><|_C S| / |R \times S| = |R|><|_C S| / |R| \times |S|$

- 如果无连接条件C，则js=1
- 如果没有元组满足连接条件，则js=0
- 一般 0<=js<=1
- 如果连接属性A为R的键，则js<= 1/ |R|
- 如果连接属性A为S的键，则js<= 1/ |S|
- 经过分析可知，一般情况下，连接结果的元组数为：

  $|R|><|_C S| = (|R| \times |S|)/M$

- M为连接属性A的值域大小

连接操作的代价估算

(2)嵌套循环法代价估算

- 假设$b_R < b_s$，故选R为外关系，S为内关系。设共有n_B块缓冲，则用嵌套循环法做连接所需的代价为：

$$C_{NLJ}=b_R+\lceil b_R/(n_B-1)\rceil \times b_s+(js\times|R|\times|S|)/p_{RS}$$

- $(js\times|R|\times|S|)/p_{RS}$存储连接结果所需的代价。
- p_{RS}为连接结果的块因子。

连接操作的代价估算

(3) 利用索引或散列寻找匹配元组法代价估算

- 若用无序索引

$$C_{NSJ} = b_R + (|R| \times (LB + |S| / N_B))$$

- 若用簇集索引

$$C_{SJ} = b_R + (|R| \times (LB + |S| / (N_B \times pS)))$$

- 若用散列(设在S关系的B属性L建立散列文件)

$$C_{SHJ} = bR + |R| \times h$$

- h>=1表示每次散列平均访问的块数，如果溢出很少，h接近于1

连接操作的代价估算

(4) 排序归并法代价估算

- 如果R、S已对A、B属性分别排序，则用排序归并法连接的代价为

$$C_{SMJ} = b_R + b_S$$

- 如果R、S中有未对A或B排序的，须加上排序的代价。排序的代价与排序的算法有关。若用两路归并外排序，共需扫描 $\lceil \log_2 b \rceil$ 次，即需

$$b \times \lceil \log_2 b \rceil$$

次I / O

连接操作的代价估算

(5)散列连接法代价估算

- 设散列文件存于内存巾，其中不放整个元组，只放tid及其连接属性值，则建立散列文件 的代价为

$$C_H = bR + bS$$

- 在连接时，以内存中的桶为单位，按连接条件R.A＝S.B进行tid配对

R中的匹配元组数＝|R|×（NB／M)＝js×|R|×NB

S中的匹配元组数＝ |S|×（NA／M)＝js×|S|×NA

- 取R、S中的匹配元组的I／O代价可估算为：

$$C_J = js \times (|R| \times NB + \times |S| \times N_A)$$

- 在估算中，假设每个不同的匹配元组处于不同的块中，这是偏高的估算，故散列连接的代 价可估算为：

$$C_{HJ} = CH + CJ = bR + bS + js \times (|R| \times N_B + \times |S| \times N_A)$$

作业

- P288
  - 9-2
  - 9-3
  - 9-5