第八章 查询处理与优化

- 8.1 查询处理算法
- 8.2 查询优化
- 8.3 小结

8.1 查询处理算法

- 关系的逻辑存储结构是二维表，表的元素是元组。不同的DBMS使用不同的物理存储结构存放关系。
- 一般地讲，DBMS向操作系统申请若干个文件，把这些文件占用的磁盘空间作为一个整体进行段页式管理，一个页面又叫做一块（Block）。不同系统的块的大小也不一样，块是DBMS的I/O单位。一个关系的元组被存储在一个或多个块中。

8.1 查询处理算法

- 为便于描述和讨论，首先引入一些记号。用B(R)表示关系R占用的块数，用T(R)表示R中元组的数目。
- 在查询处理中，需要内存和磁盘操作，由于磁盘I/O操作涉及机械动作，需要的时间与内存操作相比要高几个数量级。因此，在评估查询处理算法时，一般用算法读写的I/O块数作为衡量单位。

8.1.1 外部排序

- 排序是数据库中最基本的操作之一。很多语句中执行DISTINCT、GROUP BY和ORDER BY子句等都要使用排序操作。在数据库环境下，排序操作对象的数量十分巨大，例如，对一个有几百万元组的关系进行排序就要使用外部排序算法。

8.1.1 外部排序

- 一个典型的外部排序算法分为内部排序阶段和归并阶段。其核心思想是根据内存的大小，将存放在磁盘上待排序的关系逻辑上分为若干个段（run），一个段的大小以可使用的内存大小为上限。在内部排序阶段，从磁盘上把一个段中的全部元组读入内存，使用我们熟悉的内部排序算法，如快速排序，将这些元组排序，然后把它们写到磁盘临时存放。处理完所有的段后，进入归并排序阶段，采用多路归并排序算法，经过1趟或2趟归并排序后，就完成对关系的排序。

8.1.1 外部排序

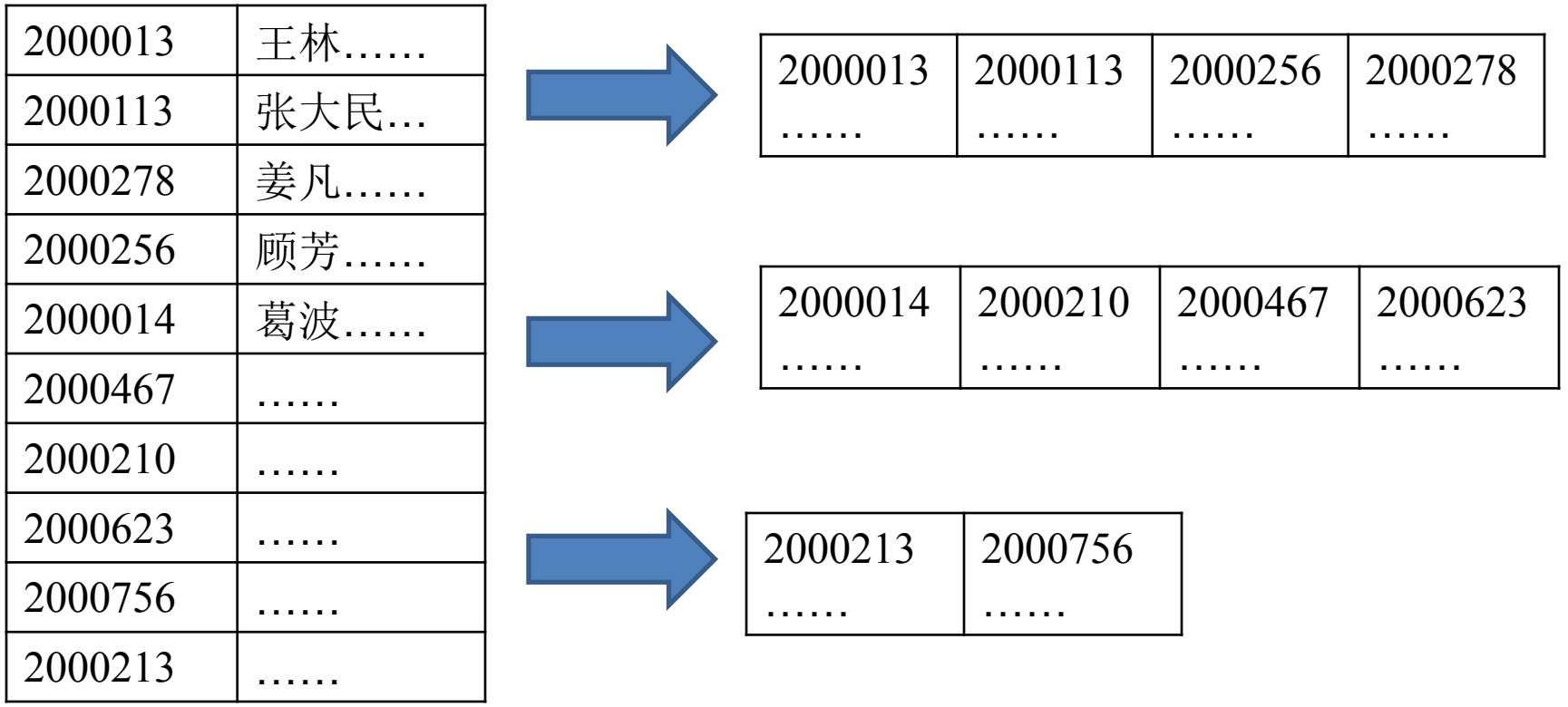

图8.1 对Student关系的内部排序阶段

8.1.1 外部排序

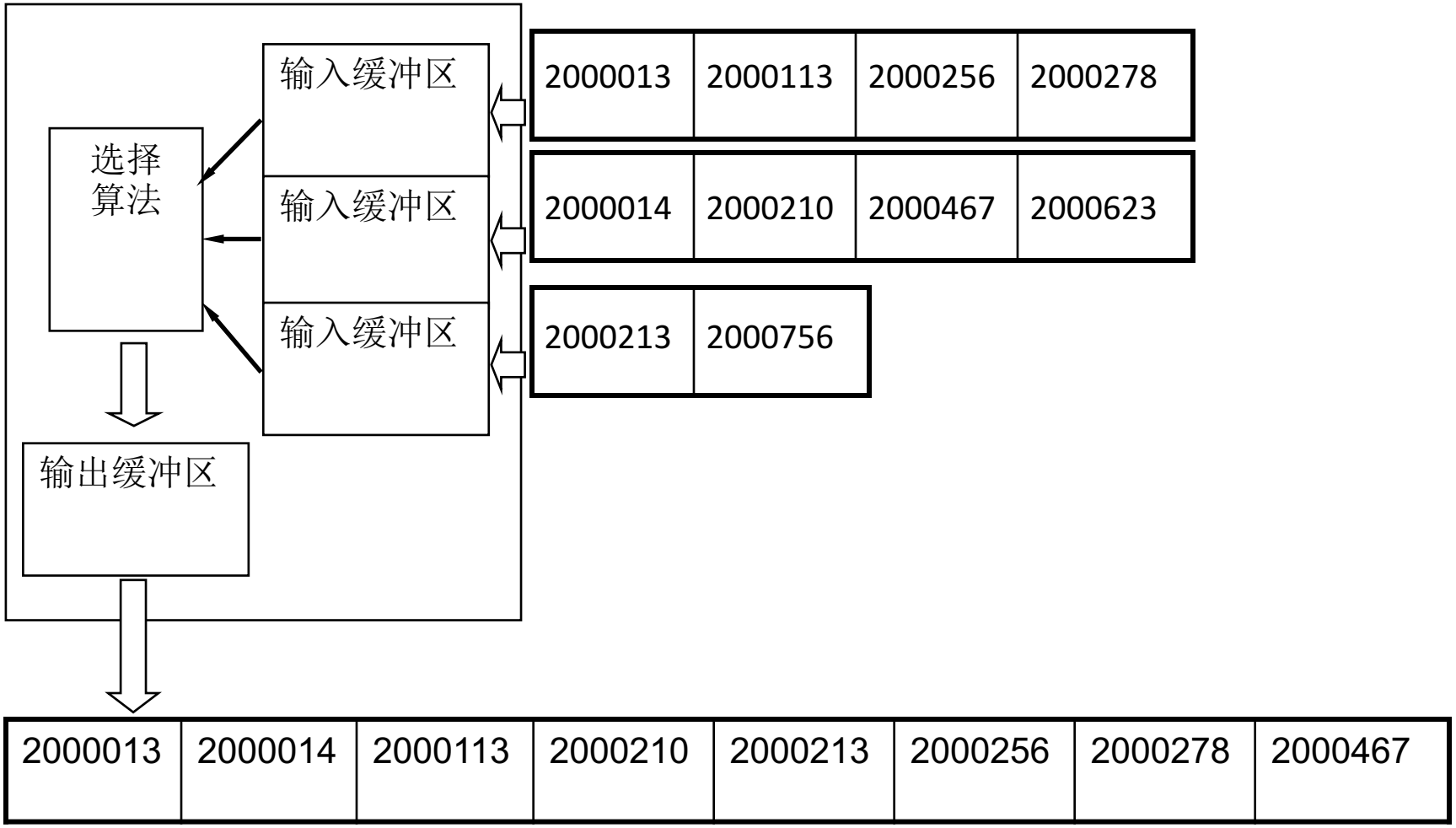

图8.2 对Student关系的多路归并阶段

8.1.1 外部排序

- 图8.1是内部排序示意图，学号是排序属性，假设可用内存大小为4个元组，10个元组分3次读入内存，排序后在磁盘上有3个有序的段。
- 图8.2是3路归并示意图，选择算法从输入缓冲区中选择最小的学号，把该学号所在的元组放到输出缓冲区，待缓冲区满后，将缓冲区中的元组在输出到磁盘。图中为了节省空间，只给出了每个元组的学号属性。一个输入缓冲区和输出缓冲区是一个或多个块，选择算法一般使用堆。

8.1.1 外部排序

- 如果可用内存为M块，则外部排序的I/O次数大约为：

$$2B(R)\log_{(M-1)}B(R)$$

- 所有的DBMS都对外部排序算法进行了最大限度的优化，理论计算表明，对绝大数应用而言，多路归并阶段只需要一趟即可。

思考：是否有更好的方法来减少I/O次数？

8.1.2 集合操作算法

- 在SQL中，集合有两种语义，即传统的集合语义和包语义，二者的差别在于是否允许出现重复的元素。
- 我们首先给出包语义的并、交和差运算的定义。为了叙述方便，用记号t^m表示元组t在集合中的出现次数，m>0表示t重复出现了m次，m=0，表示t没有出现在集合中。

8.1.2 集合操作算法

- 包并：两个集合R和S包并的结果要包含R和S中的所有元组，用公式表示为：
  - $R \cup_B S = \{ t^{m+n} \mid t^m \in R \wedge t^n \in S \}$
- 图8.2中关系R和S中都有元组（a2，b2，c1），这个元组在结果中重复出现了2次。

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_1

$\cup_B$

A	B	C
a_1	b_2	c_2
a_1	b_3	c_2
a_2	b_2	c_1

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_1
a_1	b_2	c_2
a_1	b_3	c_2
a_2	b_2	c_1

图8.2 包并运算

8.1.2 集合操作算法

- 包交：在传统的集合中，如果$t \in R \cap S$，则$t \in R$并且$t \in S$，即如果t出现在交的结果中，t在R和S中至少各出现一次。包中允许元组重复出现，因此，包交的定义如下：
  - $R \cap_B S = \{ t^k \mid t^m \in R \wedge t^n \in S, k=min(m,n) \}$

8.1.2 集合操作算法

R

A	B	C
a_2	b_2	c_1
a_1	b_1	c_1
a_2	b_2	c_1
a_1	b_2	c_2
a_2	b_2	c_1

$\cap_B$

S

A	B	C
a_1	b_2	c_2
a_2	b_2	c_1
a_1	b_3	c_2
a_2	b_2	c_1

$R \cap S$

A	B	C
a_1	b_2	c_2
a_2	b_2	c_1
a_2	b_2	c_1

图8.4 包交运算

8.1.2 集合操作算法

- 包差：差运算类似于并运算，包差的定义

$$R -_B S = \{ t^k | t^m \in R \wedge t^n \in S, k=max(0,m-n)\}$$

R

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_1	b_1	c_1
a_2	b_2	c_1

$-_B$

S

A	B	C
a_1	b_2	c_2
a_1	b_3	c_2
a_2	b_2	c_1

R-S

A	B	C
a_1	b_1	c_1
a_1	b_1	c_1

图8.5 包差运算

8.1.2 集合操作算法

- 采用的语义不同，实现集合操作的算法也不同。下面给出不同语义下集合并、交和差操作的算法。假设可用的内存为M块，参与操作的两个关系是R和S，并且S为两个关系中占用存储空间较少的关系。
- 一趟算法
  - 如果满足B(S)≤M – 1的条件，则集合操作只需要读关系R和S各一次，写结果关系一次，总的I/O次数等于2(B(R) + B(S))，这样的算法叫做一趟算法。包语义下的操作不需要去除重复元组，实现起来相对简单

8.1.2 集合操作算法

- 集合并
  - 将 S 读入内存中的M－ 1 个缓冲区
  - 建立一个查找结构，例如，二叉查找树
  - Do
    - 把 R 的一个块读到第M个缓冲区，对于这个缓冲区中的每个元组t，在查找结构中查找是否有与t相同的元组，如果没有，输出t，否则，就不输出。
  - while R中还有其它的块
  - 输出S的所有元组

思考：关系S的元组是否需要去重？

8.1.2 集合操作算法

- 集合交
  - 集合交的算法和集合并的算法不同之处在于，对于R的一个元组t，如果能在查找结构中找到它，则输出t，否则不输出它

8.1.2 集合操作算法

- 集合差
  - 集合差是一种不可交换的操作，R-S不同于S-R。假设R为关系中较大的关系。在两种情况下，将S读到M－1个缓冲区中，建立查找结构。
  - 对于R-S，每次读取R的一个块，检查块中的每个元组t。若t在S中，则忽略t，否则输出t
  - 对于S-R，每次读取R的一个块，检查块中的每个元组t。若t在S中，将主存中S的副本中将t删掉，否则不做任何处理。最后，当把R中的所以元组扫描完后，将S中剩余的元组复制到输出

8.1.2 集合操作算法

- 包并
  - 包并的结果是R和S中的所有元组，因此，只要分别读入R和S的元组并将它们输出到操作结果中即可
- 包交
  - 将S读到M—1个缓冲区中，对任意的元组t，只存储它的一个副本，并为它设置一个计数器，计算器的值等于t在S中出现的次数。
  - 读取R的每一块，对块中的每一个元组t，如果t在S中，并且t的计数器的值为正值，则输出t，并将计数器的值减1；如果t在S中，并且计数器的值为0，则不输出t；如果t不在S中则不输出t

8.1.2 集合操作算法

- 包差
  - 对于S-R，将S读到M-1个缓冲区中，对任意的元组t，只存储它的一个副本，并为它设置一个计数器，计数器的值等于t在S中出现的次数；然后，读取R的每一块到第M个缓冲区，对块中的每一个元组t，如果t出现在S中，则将t的计数器的值减1。如果t不在S中，则放弃它；处理完R的所有元组后，输出内存中S的元组。对S的任意一个元组t，如果计数器的值为正值，则重复输出t，重复次数等于它的计数器的值。如果计数器的值等于0，则不输出t

8.1.2 集合操作算法

- 包差
  - 对于R-S，同前面一样处理S中的元组；然后，读取R的每一块，对块中的每一个元组t，如果t不出现在S中，则输出t；如果t出现在S中，并且t的计数器的值等于0，则输出t，否则，不输出t，但是将其计数器减1

8.1.2 集合操作算法

- 二趟算法
  - 如果B(S)>M – 1则需要采用二趟算法，通过排序（也可以采用散列的方法）消除重复元组。
- 使用排序方法实现集合并的伪代码如下：
  - 重复地将R的M块装入内存排序，在磁盘上产生一组有序的段。
  - 对S做相同的工作，产生S的一组有序段。
  - 为R和S的每个段分配一个内存缓冲区，将每个段的第一块读入缓冲区。
  - 重复的在所有缓冲区中查找关键字最小的元组t，输出t，并且从缓冲区中删除t的所有副本。如果某个缓冲区空，则读入段中的下一块。
- 实现集合其它操作的算法和集合并相似，不再赘述。

8.1.3 选择操作算法

- 选择操作只涉及一个关系，一般采用全表扫描或者基于索引的算法
- **全表扫描算法**
  - 全表扫描算法非常简单，假设可以使用的内存为M块，全表扫描的伪代码如下：
    - 按照物理次序读R的M块到内存。
    - 检查内存的每个元组t，如果t满足选择条件，则输出t.
    - 如果R还有其它的块未被处理，重复上述两个步骤。

8.1.3 选择操作算法

- **基于索引的算法**
- 基于索引的伪代码如下：
  - 在B+树中查找满足条件的元组所在的块B1、B2、……、Bn。
  - 消除B1、B2、……、Bn中重复的块，最终块的集合为B1、B2、……、Bm。
  - 从R中逐一把B1、B2、……、Bm读入内存，在块中找到满足条件的元组作进一步的处理（如投影）并输出。

8.1.4 连接操作算法

- **嵌套循环算法**
  - 嵌套循环算法的思想非常简单：对S中的任意一个元组t_S，找出中R所有满足连接条件的元组t_R，输出t_S和t_R的连接结果。

8.1.4 连接操作算法

- 如果采用全表扫描的方法在R中查找所有满足条件的元组，并使用块的形式，则伪代码如下：
  - FOR $B_S \in S$ DO
    - FOR $B_R \in R$ DO
      - {对任意的$t_S \in B_S$，$t_R \in B_R$，如果t_S和t_R满足连接条件，则输出二者的连接结果}
  - 算法的I/O次数为：B(S) + B(S)×B(R) 。

8.1.4 连接操作算法

- **排序合并连接算法**
- 排序合并连接算法的思想是先将关系R和S按照连接属性排序，然后使用合并两个有序线性表的归并算法实现两个关系的连接操作。假设有关系R(X，Y)和S(Y，Z)，有M块内存作为缓冲区，实现自然连接的排序合并连接算法的伪代码如下：
  - (1) 用Y作为关键字，使用8.1.1节介绍的两段多路归并排序对S排序。
  - (2) 类似的对R也进行排序。

8.1.4 连接操作算法

- (3)使用两个缓冲区，分别存放S和R的一个块，归并已经排序过的S和R。设置两个指针P_S和P_R，初始时分别执向S和R在缓冲区的第一个元组，重复下面的操作步骤：
  - (3.1) 取出P_S所指的元组t_S，取出P_R所指的元组t_R。
  - (3.2) 如果$t_S.Y=t_R.Y$，则输出t_S和t_R连接的结果，然后使P_R指向下一个元组，如果P_R指向了R在缓冲区中最后一个元组的后面，则读入R的下一块，令P_R指向第一个元组，如果R中所有的块已经处理完，则算法结束，否则，转向（3.2）。

8.1.4 连接操作算法

- （3.3）如果$t_S.Y>t_R.Y$，则将P_R指向下一个元组，如果P_R指向了R在缓冲区中最后一个元组的后面，则读入R的下一块，令PR指向第一个元组，如果R中所有的块已经处理完，则算法结束，否则，转向（3.2）。
- （3.4）如果$t_S.Y<t_R.Y$，则将PS指向下一个元组，如果P_S指向了S在缓冲区中最后一个元组的后面，则读入S的下一块，令P_S指向第一个元组，如果S中所有的块已经处理完，则算法结束，否则，转向（3.2）。

- 以上算法假设R和S在属性Y上没有重复值。

思考：本节给出了集合的交并差，关系的选择和连接，还缺投影，如何实现投影？

第八章 空间查询处理与优化

- 8.1 查询处理算法
- 8.2 查询优化
  - 8.2.1 概述
  - 8.2.2 一个实例
  - 8.2.3 查询优化的一般准则
  - 8.2.4 关系代数等价变换规则
  - 8.2.5 关系代数表达式的优化算法
  - 8.2.6 优化的一般步骤
- 8.3 小结

8.2 查询优化

- 查询优化在关系数据库管理系统中有着非常重要的地位。关系数据库管理系统和非过程化的SQL语言能够取得巨大的成功，关键是得益于查询优化技术的发展。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，使关系数据库管理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。这就为关系数据库管理系统在性能上接近甚至超过非关系系数据库统提供了机遇。

8.2.1 概述

- 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，例如关系中的元组数、关系中每个属性值的分布情况等。优化器可以根据这些信息选择有效的执行计划，而用户程序则难以获得这些信息。

8.2.1 概述

- 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。

- 关系数据库查询优化的总目标是：选择有效的策略，求得给定关系表达式的值。

8.2.1 概述

- 查询优化一般来说，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。

8.2.1 概述

- 4、生成查询计划。查询计划也称查询执行方案，是由一系列内部操作组成的。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。在集中式关系数据库中，计算代价时主要考虑磁盘读写的I/O次数，也有一些系统还考虑了CPU的处理时间。

- 步骤（3）和步骤（4）实际上没有清晰的界限，有些系统是作为一个步骤处理的。

8.2.1 概述

- 目前的商品化RDBMS大都采用基于代价的优化算法。在集中式数据库中，查询的执行开销主要包括：

  总代价 = I/O代价 + CPU代价

- 在多用户环境下，内存在多个用户间的分配情况会明显地影响这些用户查询执行的总体性能。因此，多用户数据库还应考虑查询的内存开销，即：

  总代价 = I/O代价 + CPU代价 + 内存代价

8.2.2 一个实例

- [例1] 求选修了1024号课程的学生姓名。用SQL语言表达：

```
SELECT  Sname
FROM    Student, SC
WHERE  Student.Sno=SC.Sno AND
SC.Cno='1024';
```

- 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1024号课程的选课记录为50个。

8.2.2 一个实例

- 系统可以用多种等价的关系代数表达式来完成这一查询

  $Q1=\pi_{Sname}(\sigma_{Student.sno=Sc.Sno \wedge sc.cno='1024'}(Student \times SC))$

  $Q2=\pi_{Sname}(\sigma_{sc.cno='1024'}(Student \bowtie SC))$

  $Q3=\pi_{Sname}(Student \bowtie \sigma_{sc.cno='1024'}(SC))$

- 还可以写出几种等价的关系代数表达式，但分析这三种就足以说明问题了。后面将看到由于查询执行的策略不同，查询时间相差很大。

思考：Q1、Q2和Q3，你认为哪一种效率最高？

8.2.2 一个实例

$Q1=\pi_{Sname}(\sigma_{Student.sno=Sc.Sno \wedge sc.cno='1024'}(Student \times SC))$

- 计算广义笛卡尔积
  - 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。

8.2.2 一个实例

$Q1=\pi_{Sname}(\sigma_{Student.sno=Sc.Sno \wedge sc.cno='1024'}(Student \times SC))$

- 计算广义笛卡尔积
  - 假设一个块能装入10个学生记录或100个选课记录，在内存中存放5块学生记录和1块选课记录，则读取总块数:

    1000/10 + (1000/(10*5)) * (10000/100) = 2100块
  - 其中读学生表100块，读选课记录表20编，每编读100块。若每秒读写20块，则总用时105秒
  - 连接后的元组数目为$10^3 \times 10^4 = 10^7$，设每块能装入10个元组，则写出这些块需要$10^6/20 = 5 \times 10^4$秒

8.2.2 一个实例

$Q1=\pi_{Sname}(\sigma_{Student.sno=Sc.Sno\wedge sc.cno='1024'}(Student\times SC))$

- 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。这一步读取中间文件花费的时间(同写中间文件一样)需5×10^4 s。满足条件的元组假设仅50个，均可放在内存。
- 作投影
  - 把第2步的结果在Sname上作投影输出，得到最终结果。因此第一种情况下执行查询的总时间≈ $l05+2\times5\times10^4\approx l0^5$ s。这里，所有内存处理时间均忽略不计。

8.2.2 一个实例

$Q2=\pi_{Sname}(\sigma_{sc.cno='1024'}(Student \bowtie SC))$

- 计算自然连接
  - 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l05s。但自然连接的结果比第一种情况大大减少，为10^4 个。因此写出这些元组时间为$10^4/10/20=50$ s，仅为第一种情况的千分之一
- 读取中间文件块，执行选择运算，花费时间也为50 s
- 把第2步结果投影输出
- 总的执行时间≈105+50+50≈205 s

8.2.2 一个实例

$Q3=\pi_{Sname}(Student \bowtie \sigma_{sc.cno='1024'}(SC))$

- 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5s，因为满足条件的元组仅50个，不必使用中间文件。
- 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5s。
- 3. 把连接结果投影输出。
- 总的执行时间≈5+5≈10s。

8.2.2 一个实例

- 假如SC表的Cno字段上、Student表Sno字段上分别有索引，则总的存取时间将进一步减少到数秒。
- 这个简单的例子充分说明了查询优化的必要性，同时也给出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。

8.2.3 查询优化的一般准则

- 下面的优化策略一般能提高查询效率，但不一定是所有策略中最优的。
- 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。它常常可使执行时节约几个数量级，因为选择运算一般使计算的中间结果大大变小。
- 2. 在执行连接前对关系适当地预处理。预处理方法主要有两种，在连接属性上建立索引和对关系排序，然后执行连接。

8.2.3 查询优化的一般准则

- 3. 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。
- 4. 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。

8.2.3 查询优化的一般准则

- 5. 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(如8.1.2节中的实例)。
- 8. 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。

8.2.4 关系代数等价变换规则

- 研究关系代数表达式的优化最好从研究关系表达式的等价变换规则开始。所谓关系代数表达式的等价是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。

8.2.4 关系代数等价变换规则

- 两个关系表达式EI和E2是等价的，可记为E1≡E2。常用的等价变换规则有以下几条：
- 1、 连接、笛卡尔积交换律
- 设E1和E2是关系代数表达式，F是连接运算的条件，则有：
  - E1×E2≡E2×E1
  - E1⋈E2≡E2⋈E1
  - E1⋈E2≡E2⋈E1

8.2.4 关系代数等价变换规则

- 2、连接、笛卡尔积的结合律
- 设E1，E2，E3是关系代数表达式，F1和F2是连接运算的条件，则有：
  - (E1×E2)×E3 ≡E1 ×(E2×E3)
  - (E1 E2) ⋈E3 ≡E1⋈(E2 ⋈E3)
  - (E1 E2) ⋈E3 ≡E1⋈(E2 ⋈E3)

8.2.4 关系代数等价变换规则

- 3. 投影的串接定律

$$\pi_{A1,A2,\cdots An}(\pi_{B1,B2,\cdots Bm}(E)) \equiv \pi_{A1,A2,\cdots An}(E)$$

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

8.2.4 关系代数等价变换规则

- 4. 选择的串接定律

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

这里，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。

8.2.4 关系代数等价变换规则

- 5. 选择与投影的交换律

$$\sigma_F(\pi_{A1,A2,\cdots An}(E)) \equiv \pi_{A1,A2,\cdots An}(\sigma_F(E))$$

这里，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，则有更一般的规则：

$$\pi_{A1,\ A2,\ \cdots An}(\sigma_F(E)) \equiv \pi_{A1,A2,\cdots An}(\sigma_F(\pi_{A1,A2\ ,\cdots AnB1,B2,\cdots Bm}(E)))$$

8.2.4 关系代数等价变换规则

- 8. 选择与笛卡尔积的交换律

  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：

  $$\sigma_F(E_1 \times E_2) \equiv \sigma_{F1}(E_1) \times \sigma_{F2}(E_2)$$

  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

  $$\sigma_F(E_1 \times E_2) \equiv \sigma_{F2}(\sigma_{F1}(E_1) \times E_2)$$

  它使部分选择在笛卡尔积前先做。

8.2.4 关系代数等价变换规则

- 7. 选择与并的交换

  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. 选择与差运算的交换

  若E_1与E_2有相同的属性名，则

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

8.2.4 关系代数等价变换规则

- 9. 投影与笛卡尔积的交换

  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

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

- l0. 投影与并的交换

  设E_1和E_2 有相同的属性名，则

  $$(E_1 \cup E_2) \equiv (E_1) \cup (E_2)$$

8.2.5 关系代数表达式的优化算法

- 应用上面的变换法则来优化关系表达式，使优化后的表达式能遵循8.1.3节中的一般原则。下面给出关系表达式的优化算法。
  - 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。

8.2.5 关系代数表达式的优化算法

方法：

（1）利用规则4把形如$\sigma_{F1\wedge F2\wedge\cdots\wedge Fn}(E)$变换为 $\sigma_{F1}(\sigma_{F2}(\ldots(\sigma_{Fn}(E))\ldots))$。

（2）对每一个选择，利用规则4～8尽可能把它移到树的叶端。

（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。

8.2.5 关系代数表达式的优化算法

（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，尽管这种变换似乎违背“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结点分组。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。

8.2.5 关系代数表达式的优化算法

（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。

8.2.6 优化的一般步骤

- 各个关系数据库管理系统的优化方法不尽相同，大致的步骤可以归纳如下:
  - （1）把查询转换成某种内部表示
    - 关系代数语法树
  - （2）把语法树转换成标准(优化)形式
  - （3）选择低层的存取路径
    - 考虑索引和数据的存储分布
  - （4）生成查询计划，选择代价最小的

8.2.6 优化的一般步骤

- 生成查询计划，选择代价最小的
  - 在做连接运算时，若表R1和R2均无序，连接属性上也没有索引，则可以有以下几种查询计划
    - 对两个表做属性预排序
    - 对R1在连接属性上建立索引
    - 对R2在连接属性上建立索引
    - 在R1和R2的连接属性上均建立索引
  - 计算代价主要考虑磁盘读写的I/O次数，内存CPU处理时间在粗略计算时可不考虑

8.3 小结

- 在这一章中，我们介绍了外部排序算法、集合操作算法、选择和连接算法，这些算法是实现SQL查询的基础。然后通过一个实例简要描述了查询优化的步骤和方法。了解这些算法和查询优化的原理可以帮助我们更好的理解SQL语句的执行过程，书写出更好的SQL语句。

8.3 小结

- 1、DBMS以块作为I/O单位，将磁盘上的块有机的组织成一个整体，一个关系被存储在若干块中。一般以I/O次数衡量算法的性能。
- 2、外部排序是实现其它数据库操作的基础，多采用两阶段多路归并排序，DBMS对其进行了充分的优化句。
- 3、由于集合中不能有相同的元素，消除重复元素需要排序或散列，代价较高，而投影操作等可能产生重复元素，因此，SQL中采用包语义，特别是包并操作十分简单。

8.3 小结

- 4、连接操作是关系模型中必不可少的操作，研究工作者对此进行了深入的研究。嵌套循环连接算法是最简单实用的算法，对于理解连接操作很有帮助。DBMS对连接操作做了充分的优化。
- 5、通过索引在选择操作和连接操作中的应用，我们可以理解建立一个适当的索引的重要性。现代的关系数据库一般都提供了自动选择索引的实用工具，减少了建立索引的盲目性。
- 6、查询优化有代数操作优化和物理操作优化两个层面。代数操作优化是通过代数式的等价变换而实现的，我们可以再次体会到理论研究对实际应用的促进作用。