第三篇　系统篇

- 讨论数据库管理系统中查询处理和事务管理的基本概念和基础知识
  - 关系查询处理和查询优化
  - 数据库恢复
  - 并发控制

数据库系统概论

An Introduction to Database System

第九章 关系系统及其查询优化

中国人民大学信息学院 陈红

第九章　关系系统及其查询优化

9.1 关系数据库系统的查询处理

9.2 关系数据库系统的查询优化

9.3 代数优化

9.4 物理优化

9.5 查询执行

9.6 小　结

9.4 物理优化

- 代数优化改变查询语句中操作的次序和组合，不涉及底层的存取路径
- 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，求得优化的查询计划
  - 具有交换律和结合律的操作符的操作顺序
  - 为操作符选取操作算法

物理优化（续）

- 物理优化方法
  - 基于规则的启发式优化
    - 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

9.4 物理优化

- 9.4.1 基于启发式规则的存取路径选择优化
- 9.4.2 基于代价的优化

9.4.1 基于启发式规则的存取路径选择优化

- 一、 选择操作的启发式规则
- 二、 连接操作的启发式规则

基于启发式规则的存取路径选择优化（续）

- 一、 选择操作的启发式规则

1. 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：

2. 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

（一般的 RDBMS 会自动建立主码索引）

基于启发式规则的存取路径选择优化（续）

3. 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

基于启发式规则的存取路径选择优化（续）

4. 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

基于启发式规则的存取路径选择优化（续）

5. 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
   - 如果某些属性上有一般的索引，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

6. 对于用 OR 连接的析取选择条件，一般使用全表顺序扫描

基于启发式规则的存取路径选择优化（续）

- 二、 连接操作的启发式规则：

1. 如果 2 个表都已经按照连接属性排序
   - 选用排序 - 合并方法
2. 如果一个表在连接属性上有索引
   - 选用索引连接方法
3. 如果上面 2 个规则都不适用，其中一个表较小
   - 选用 Hash join 方法

基于启发式规则的存取路径选择优化（续）

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 分别占用的块数为 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+(Br/K-1)Bs
- 显然应该选块数小的表作为外表

9.4 物理优化（续）

- 9.4.1 基于启发式规则的存取路径选择优化
- 9.4.2 基于代价的优化

9.4.2 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

基于代价的优化（续）

- 一、 统计信息
- 二、 代价估算示例
- 三、优化方法

基于代价的优化（续）

一、 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

- 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)

基于代价的优化（续）

2. 对基表的每个列

- 该列不同值的个数 (m)
- 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，则每个值的选择率＝具有该值的元组数 /N
- 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)

基于代价的优化（续）

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)

基于代价的优化（续）

二、 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

基于代价的优化（续）

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
   - 如果选择条件是　**非码属性＝值**
     - 若为 B+ 树索引， S 是索引的选择基数 (有 S 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
     - (最坏的情况)cost=L+S

基于代价的优化（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

基于代价的优化（续）

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+Br Bs/(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。

基于代价的优化（续）

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

    $(2*B)+(2*B*\log_2 B)$

三、物理优化方法

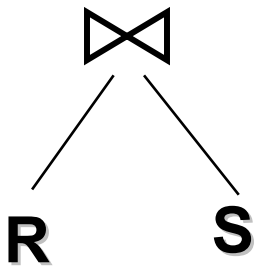
- 物理优化的搜索空间
- 搜索物理计划的方法

物理优化的搜索空间

- 物理优化的搜索空间非常庞大
- 以连接为例：当连接有两个以上的关系时，可能的连接树的数量会迅速增长。

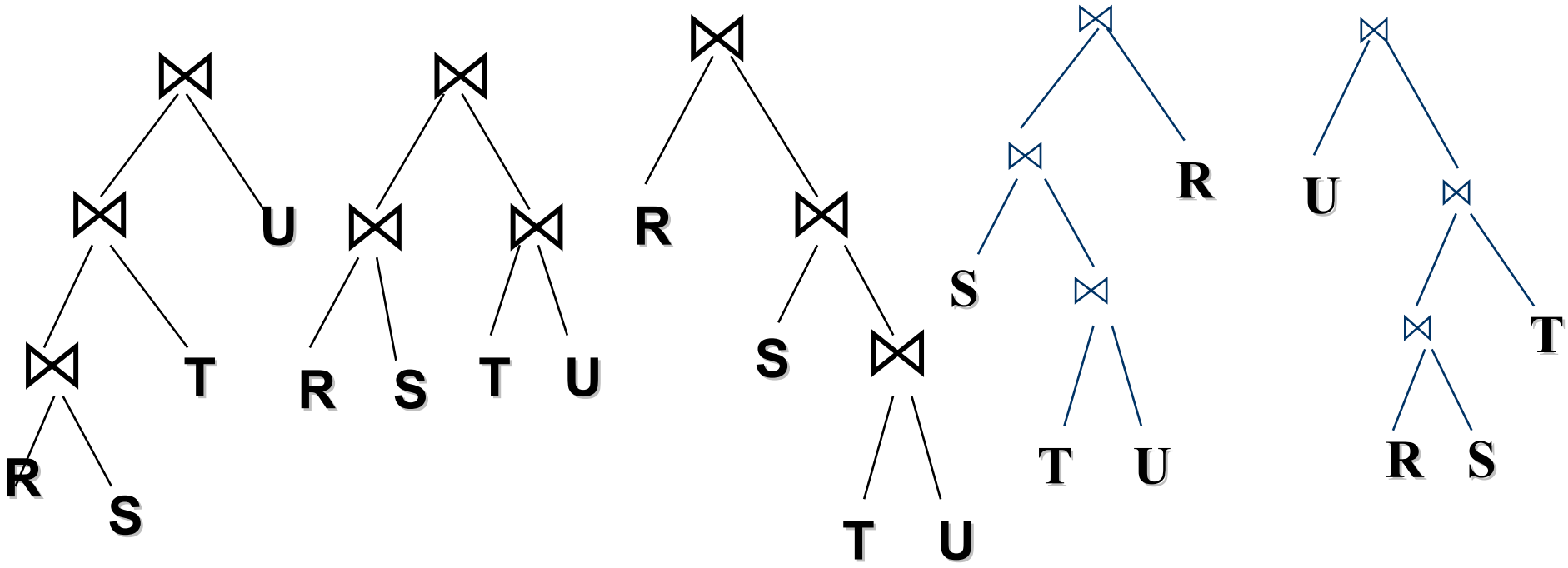
例：两表连接的连接树

- 树形：只有一种



例：四表连接

- 可能的树形状有 5 种



四表连接

- 对于每一种树形状，四个关系又可以有 4! 即 24 种排列次序
- 于是四表连接的查询树共有 5*24 即 120 种形式。

- 对于每个连接操作结点，假设可能有 4 种算法可供选择，因此可以得到 4*4*120=1920 种查询执行计划。

N 表连接

- 树形状的数目 T(n) 可如下递归计算出来：

$$\begin{cases} T(1) = 1 \\ T(n) = \sum_{i=1}^{n} T(i) * T(n-i) \end{cases}$$

例： T(1)=1， T(2)=1， T(3)=2， T(4)=5，
T(5)=14， T(6)=42，等等。

N 表连接

- 对于 n 个关系，每种树形可以有 *n！* 种方法来分配关系。
- 因此当允许所有树型时，查询计划树的可能数目为 *T(n)*n！* 。
- 对于每个操作符结点，又可能有 *m* 种算法可供选择，因此可以得到 *nm* T(n)*n！* 种查询执行计划。
- 为每一种查询执行计划计算其代价，并从中选出最小代价的计划，当 n 比较大时，其代价将是非常大的。

n	m	T(n)	n!	查询计划数目
2	4	1	2	16
3	4	2	6	144
4	4	5	24	1920
5	4	14	120	33600
6	4	42	720	725760

搜索物理计划的方法

- 穷举法
- 启发式选择
- 分支界定计划枚举
- 爬山法
- 动态规划
- Selinger 风格优化

1. 穷举法

- 方法
  - 对所有可能的选择 (连接的次序、操作符的物理实现等) 加以组合；
  - 对每个可能的物理计划估算代价；
  - 选择具有最小代价的一个计划。
- 优点：针对给定的逻辑查询计划，可以为之找到最优的物理计划。
- 缺点：代价高昂。

2. 启发式选择

- 基本思想：用启发式规则裁剪部分可能不太好的物理计划。
- 典型算法：贪心法
- 优点：物理优化的搜索空间比较显著地减小
- 缺点：可能会丢失最优计划

例：优化连接树的启发式规则

- 分为两大类
  - 第一类是限制查询计划树形状的规则
  - 第二类是选取操作算法的启发式规则
- 限制查询计划树形状的规则，就是将查询计划树限制为某一类固定形状，例如左深连接树。
  - 对于 n 个关系，如果只有一种左深连接树的形状，当每个操作符 m 种算法时，搜索空间由 $nm*T(n)*n!$ 种可能的查询执行计划降低为 nm n！种可能的计划。

n	m	T(n)	n!	查询计划数目
2	4	1	2	16
3	4	2	6	144
4	4	5	24	1920
5	4	14	120	33600
6	4	4[illegible]	720	725760

将树形限制为左深树

n	m	T(n)	n!	查询计划数目
2	4	1	2	16
3	4	1	6	72
4	4	1	24	384
5	4	1	120	2400
6	4	1	720	17280

- 选取操作算法的启发式规则
  - 如果参与连接的一个关系在连接属性上有索引，则采用索引连接，且该关系作内表。
  - 如果参与连接的一个关系是按连接属性排好序的，则采用排序 - 合并连接比用散列连接好，但不一定比索引连接好。

3. 分支界定计划枚举

- 基本思想
  - 首先通过启发式规则为整个逻辑查询计划找到一个好的物理计划，令其代价为 C
  - 然后当考虑子计划的其他计划时，去除那些代价大于 C 的计划
  - 如果代价 C 较小，则终止搜索并获得目前为止最优的计划。因为找到更好的计划所花费的代价可能大于所获收益。

4. 爬山法

- 基本思想
  - 依据启发式规则选定一个较好的查询计划，以此作为开始
  - 通过对计划作小的修改，找到具有较低代价的“邻近”计划
  - 当小的修改已经找不到更好的计划时，搜索结束。

5. 动态规划

- 基本思想
  - 自底向上搜索，但对于每个子表达式，仅保留最小代价的计划。
  - 当自底向上对计划树进行处理时，在假定每个子表达式使用了最佳计划的前提下，对当前结点的可能实现加以考虑。

6. Selinger 风格优化

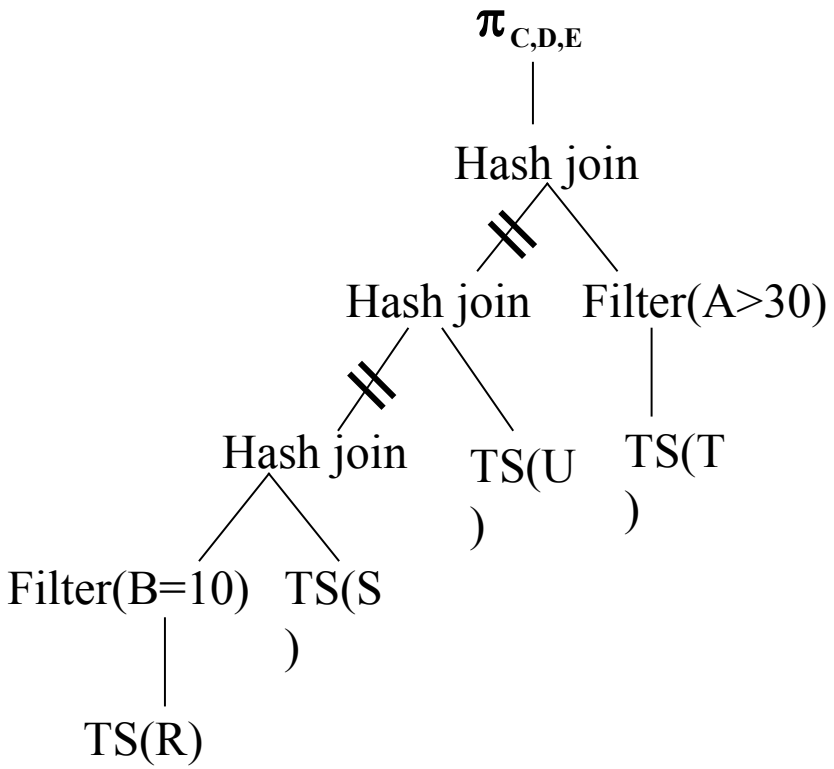
- 基本思想
  - 是动态规划方法的改进。
  - 不仅记录了每个子表达式的最小代价的计划，而且也记录了那些具有较高代价但所产生结果的顺序对表达式树中较高层很有用的计划。

第九章　关系系统及其查询优化

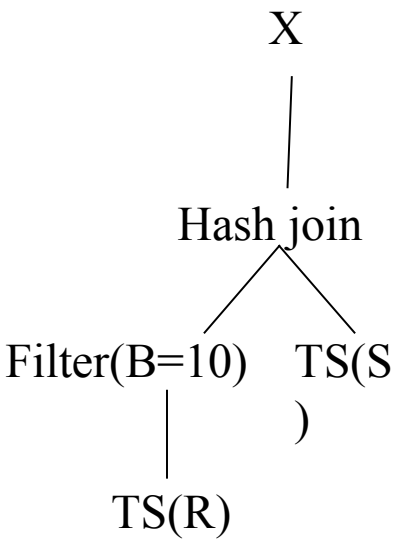
9.5 查询计划的执行

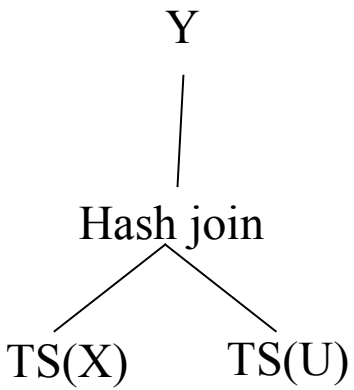
1. 在表示物化的边上将树分解成子树。子树将一次一个地被执行。
2. 执行整棵树的前序遍历，对子树进行排序。
3. 对每个子树，采用流水线方式执行
   - 需求驱动
   - 生产者驱动

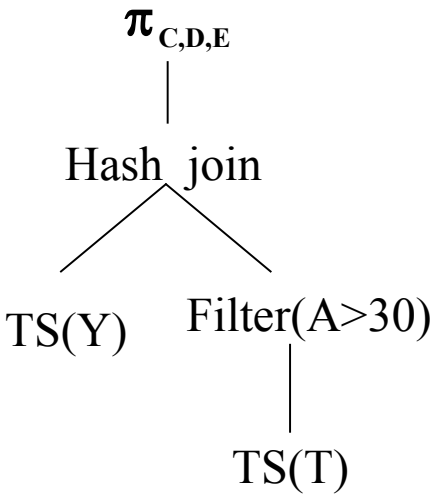
例：



- 对于左图的查询计划，假设各个表都比较大，无法放入内存，需要采用两趟 Hash Join 算法进行连接
  - ✓ 第一趟将要连接的外表和内表按照连接属性的 hash 值分成 n 个可独立连接对
  - ✓ 然后在第二趟分别对 n 个可独立连接对进行连接
- 这时各个连接操作符之间必须以实体化的方式执行。
- 不严格地说，该查询计划被划分成三棵子树，查询执行器将按从左至右的顺序执行这些子树，而每一棵子树的执行均采用流水线方式。

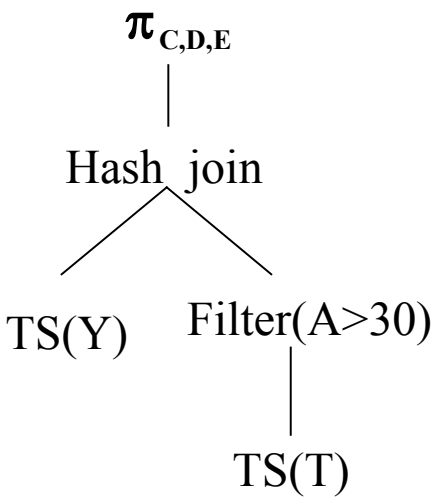

X
Hash join
Filter(B=10)
TS(S
)
TS(R)


Y
Hash join
TS(X)
TS(U)


π C,D,E
Hash join
TS(Y)
Filter(A>30)
TS(T)

需求驱动的流水线执行方式

- 需求驱动流水线执行方式是一种自顶向下的执行方式。
- 系统反复地向流水线顶端的操作符发出需求元组的请求。每当一个操作符收到需要元组的请求时，就计算下一个（几个）元组并返回这些元组。
- 若该操作符的输入不是来自流水线，则下一个（几个）元组可以由输入关系中计算得到，同时系统记载目前为止已返回了哪些元组。
- 若该操作符的某些输入来自流水线，那么该操作符也发出请求以获得来自流水线输入的元组。使用来自流水线输入的元组时，子操作符计算输出元组，然后把它们传给父层。



以第 3 个子树为例

- 采用需求驱动的流水线执行方式时，系统将反复向根结点的投影操作发出需求元组的请求。
- 初始时，投影操作尚未收到输入，于是它向下层的连接操作符发出需求元组的请求。
- 连接操作符向选择操作符和 Y 的全表扫描操作符发出需求元组的请求。
- 选择操作符向 T 的全表扫描操作符发元组请求。
- 选择操作符的输入来自关系 T 的全表扫描。
- 选择操作符获得输入后，将输出提供给连接操作符
- 连接操作符获得左右子结点的输入后，将结果提供给投影操作符
- 投影操作符将输出元组返回给系统。
- 系统接收到返回数据后，再次向根结点投影操作发出需求元组的请求，重复上述过程，直到全部数据处理完。
- 就这样，通过系统反复向根结点的投影操作发

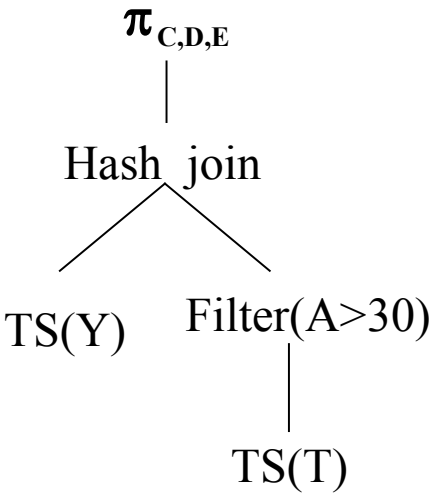
需求驱动的流水线执行方式

- **DBMS 负责操作符的调度：迭代器 + 缓冲区**
  - 对于需求驱动的流水线法，使用迭代器来执行每一棵子树的所有节点。这时，一棵子树中所有节点被同时执行，并通过调用 **GetNext()** 函数同步各操作符的执行顺序。
  - 使用缓冲区
- 防抖动
  - 填充因子

生产者驱动的流水线执行方式

- 生产者驱动流水线是一种自底向上的执行方式，各操作符并不等待元组请求，而是积极主动地产生元组。
- 流水线底部的每个操作符不断地产生元组并将它们放在输出缓冲区中，直到缓冲区满为止。
- 流水线中任何其他层的操作符只要获得较低层的输入元组就产生输出元组，直到其输出缓冲区满为止。
- 一个操作符使用流水线输入的一条元组后，将其从输入缓冲区中删除
- 一旦输出缓冲区已满，操作符必须等待，直到其父操作符将元组从该缓冲区取走而为更多的元组腾出空间。此时，该操作符接着产生更多的元组，直到缓冲区再次满了为止
- 这个过程不断重复，直到该操作符产生所有的输出元组为止

例：



生产者驱动流水线执行方式

- 操作符的调度
  - DBMS 自己调度：依据缓冲区的填充情况
    - 只有当一个输出缓冲区已满，或一个输入缓冲区已空，需更多的输入元组用于产生输出元组时，系统才需在各操作符之间切换。
  - 交给 OS 调度： OS 依据分时机制来调度，依据缓冲区填充情况进行调整
- 防抖动
  - 填充因子

两类流水线比较

- 执行方式： Push 与 Pull
  - 使用生产者驱动的流水线方法可以看成将数据从一棵操作符树的底层推（ push ）上去的过程；而使用需求驱动的流水线方法可看成是从树顶将数据拉（ pull ）上来的过程。
- 产生元组： Active 与 Lazy
  - 在生产者驱动流水线中，元组的产生是积极的；而在需求驱动流水线中，元组消极地（ lazily ）按需求产生。

第九章　关系系统及其查询优化

9.1 关系数据库系统的查询处理

9.2 关系数据库系统的查询优化

9.3 代数优化

9.4 物理优化

9.5 查询执行

9.6 小结

9.6 小　结

- 查询处理是 RDBMS 的[illegible]化技术是查询处理的关键技术
- 本章主要内容
  - 查询处理过程
  - 查询优化
    - 代数优化
    - 物理优化
  - 查询执行

查询分析
查询检查
查询优化
查询执行

9.6 小　结

- 查询处理是 RDBMS 的核心，查询优化技术是查询处理的关键技术
- 本章主要内容
  - 查询处理过程
  - 查询优化
    - 代数优化
    - 物理优化
  - 查询执行

启发式代数优化

9.6 小　结

- 查询处理是 RDBMS 的核心，查询优化技术是查询处理的关键技术
- 本章主要内容
  - 查询处理过程
  - 查询优化
    - 代数优化
    - 物理优化
  - 查询执行

基于规则的存取路径优化
基于代价的优化

9.6 小　结

- 查询处理是 RDBMS 的核心，查询优化技术是查询处理的关键技术
- 本章主要内容
  - 查询处理过程
  - 查询优化
    - 代数优化
    - 物理优化
  - 查询执行

需求驱动流水线
生产者驱动流水线

小　　结（续）

- 比较复杂的查询，尤其是涉及连接和嵌套的查询
  - 不要把优化的任务全部放在 RDBMS 上
  - 应该找出 RDBMS 的优化规律，以写出适合 RDBMS 自动优化的 SQL 语句
- 对于 RDBMS 不能优化的查询需要重写查询语句，进行手工调整以优化性能

作业及实验

- 作业
  - P275-276　2（笔头）
  - P275-276　1,3,4（自己找到答案）
- 实验
  - P276 实验 9


2009/2/23


2009/2/23