第6章　查询处理和优化

2012. 03

目录 Contents

- 6.1 概述
- 6.2 代数优化
- 6.3 依赖于存取路径的规则优化
- 6.4 代价估算优化

6.1 概述

- 一、查询与查询处理
  - 查询(Query)：是数据库的最基本、最常用、最复杂的操作。包括：
    - 直接的SELECT：
      - **e.g. SELECT…FROM…WHERE…(SELECT…)…;**
    - 间接的SELECT：
      - **e.g. INSERT INTO…SELECT…;**
      - **DELETE FROM…WHERE…(SELECT…)…;**
      - **UPDATE…SET…WHERE…(SELECT…)…;**
      - **CREATE VIEW…AS SELECT…;**

6.1 概述

- 一、查询与查询处理(cont.)
  - 查询处理(Query Processing)：从DBMS接受一个查询请求到返回查询结果的处理过程。包括以下步骤：
    - 词法、语法分析
      - **SELECT操作转换为语法树；**
    - 权限检查
      - **检查用户是否对有关模式对象有相应的访问权限；**
    - 语义分析与查询优化
      - **形成高效／优化的执行计划(execution plan)；**
    - 执行查询并返回结果
      - **按执行计划执行查询，并返回结果。**
  - 可见，查询处理的关键／核心步骤是查询优化。

6.1 概述

- 二、查询优化及其方法
  - 查询优化(Query Optimization): 为一个查询确定一个效率（较）高的执行计划（即：操作的先后顺序，表数据的I/O方法，等）。

6.1 概述

- 二、查询优化及其方法 (cont.)
  - DBMS为何要进行查询优化?
    - 为了提高数据库的性能
      - **虽然影响数据库性能的因素有很多（如：操作的执行效率、DB的设计质量、通讯开销、硬件性能，…），但在相同环境下，执行效率是关键因素。**
    - 为了方便用户
      - **DBMS有了优化机制后，用户可随意表达查询要求，而不必考虑效率问题。这是使用高度非过程化、说明性语言（SQL）的关系数据库环境所必须的。**
    - 由系统来“优化”比由用户来“优化”更有效
      - **DBMS可充分利用数据字典（DD）中保存的各种参数进行优化，这比网状 / 层次数据库系统中由用户来写“优化的”DML语句更为有效。**

6.1 概述

- 二、查询优化及其方法 (cont.)
  - 查询优化的方法
    - **两个层次**
      - 代数层：将查询所对应的关系代数表达式以抽象的语法树来表示，然后等价变换之，以期减小查询过程中的中间结果大小。
      - 物理层：选择高效的存取路径存取表数据，以期提高磁盘I/O的效率。
    - **两种策略**
      - 基于规则（Rule）：运用启发式规则为查询确定一个理论上认为“最优的”执行计划。
      - 基于代价(Cost)估算：为查询确定几个理论上认为“较优的”执行计划，然后根据数据字典中的参数分别估算它们的执行代价(时间开销)，选择代价最小者作为“最优的”执行计划。

6.1 概述

- 二、查询优化及其方法 (cont.)
  - 查询优化的方法(cont.)
    - 实际系统中往往综合运用以上两种策略，在两个层次上进行优化。
    - DBMS中承担优化工作的部分称优化器(Optimizer)。
    - 一般地，优化器的工作对用户是透明的。
    - 但有的系统允许用户询问优化器为某个查询选择的执行计划(e.g. Oracle中通过EXPLAIN PLAN命令)；还允许用户建议优化器如何优化一个查询(e.g. Oracle中通过在SELECT中插入优化提示)。

目录 Contents

- 6.1 概述
- 6.2 代数优化
- 6.3 依赖于存取路径的规则优化
- 6.4 代价估算优化

6.2 代数优化

- **目标**
  - 代数优化是对查询进行等效变换，尽量减小查询中间结果的大小。
  - 关系代数表达式等价：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。
- **方法**
  - 用语法树表示查询，
  - 基于代数变换规则，
  - 等价变换关系操作的次序。

6.2 代数优化

- **策略**
  - **尽可能缩减查询过程中的中间结果**
  - 先做选择、投影，后做连接、并；
  - 先做小关系间的连接 / 笛卡尔积，后做大关系间的连接 / 笛卡尔积；
  - 将“笛卡尔积＋选择”合并为“连接”；
  - 对原始关系加必要的投影，以消除对查询无用的属性。

6.2 代数优化

- 常用等价变换规则
  - $\sigma_{F1}(\sigma_{F2}(R)) \equiv \sigma_{F1 \wedge F2}(R)$
  - $\sigma_{F1}(\sigma_{F2}(R)) \equiv \sigma_{F2}(\sigma_{F1}(R))$
  - $\Pi_{list1}(\Pi_{list2}(R)) \equiv \Pi_{list1}(R)$
    - list1⊆list2, list1和list2为属性集
  - $\Pi_{list}(\sigma_{F}(R)) \equiv \sigma_{F}(\Pi_{list}(R))$
    - 如果选择条件F所使用的属性全在属性集list中
  - $R \bowtie S \equiv S \bowtie R$
  - $\sigma_{F}(R \bowtie S) \equiv (\sigma_{F}(R)) \bowtie S$
    - 如果F中所涉及的属性都是 R中的属性，即 Attr.(F) ⊆ Attr.(R)

6.2 代数优化

- 常用等价变换规则(续)
  - $\sigma_{F1 \wedge F2}(R \bowtie S) \equiv \sigma_{F1}(R) \bowtie \sigma_{F2}(S)$
    - 如果F1中所涉及的属性都是R中的属性，F2中所涉及的属性都是S中的属性，即Attr.(F1) $\subseteq$ Attr.(R), Attr.(F2) $\subseteq$ Attr.(S)
  - $\Pi_L(R \bowtie_f S) \equiv (\Pi_{A1,\ldots An}(R)) \bowtie_f (\Pi_{B1,\ldots Bm}(S))$
    - 属性集L=$\{A_1,\ldots,A_n,B_1,\ldots,B_m\}$，其中 $\{A_1,\ldots,A_n\} \subseteq$Attr.(R), $\{B_1,\ldots,B_m\} \subseteq$ Attr.(S), Attr.(f) $\subseteq$L。
    - 含义是如果将投影操作推到连接前执行，必须将参与连接的关系中与连接条件有关的属性保留。若属性集L没有完全包括这些属性，则在交换投影与连接操作时，必须把这些属性补充到相应的关系的投影属性集中。

6.2 代数优化

- 常用等价变换规则(续)
  - $\sigma_F(R\ \theta\ S) \equiv (\sigma_F(R))\ \theta\ (\sigma_F(S))$
    - 集合运算$\theta \in \{\cup, \cap, -\}$
  - $\Pi_L(R \cup S) \equiv \Pi_L(R) \cup \Pi_L(S)$
  - 设运算$\theta \in \{\bowtie, \times, \cup, \cap\}$,

    则$R\ \theta\ (S\ \theta\ T) \equiv (R\ \theta\ S)\ \theta\ T$

6.2 代数优化

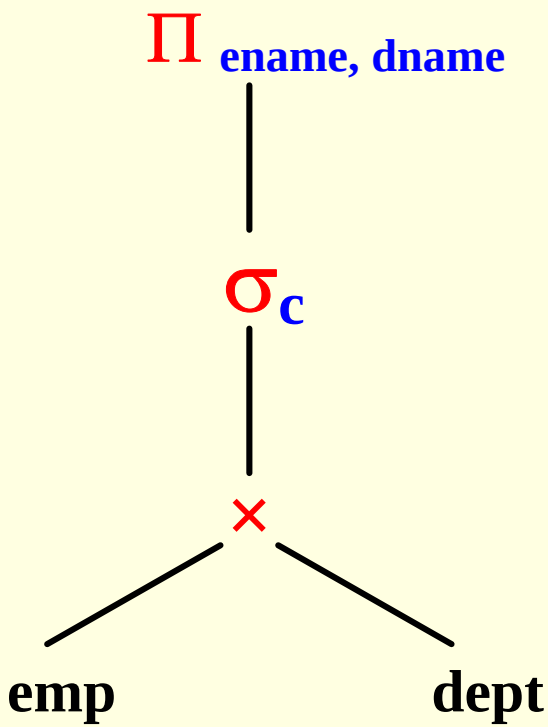
- 例子
  - SELECT ename, dname FROM emp, dept
    WHERE emp.deptno=dept.deptno AND job='clerk' AND
    loc='Xian' ;

- $Q1= \Pi_{ename,\ dname}(\sigma_{emp.deptno=dept.deptno\ AND\ job='clerk'\ AND\ loc='Xian'}(dept \times emp))$
- $Q2= \Pi_{ename,\ dname}(\sigma_{emp.deptno=dept.deptno}(\sigma_{loc='Xian'}(dept) \times \sigma_{job='clerk'}(emp)))$
- $Q3= \Pi_{ename,\ dname}(\sigma_{loc='Xian'}(dept) \bowtie \sigma_{job='clerk'}(emp))$
- $Q4= \Pi_{ename,\ dname}(\Pi_{deptno,\ dname}(\sigma_{loc='Xian'}(dept)) \bowtie \Pi_{ename,\ deptno}(\sigma_{job='clerk'}(emp)))$

6.2 代数优化

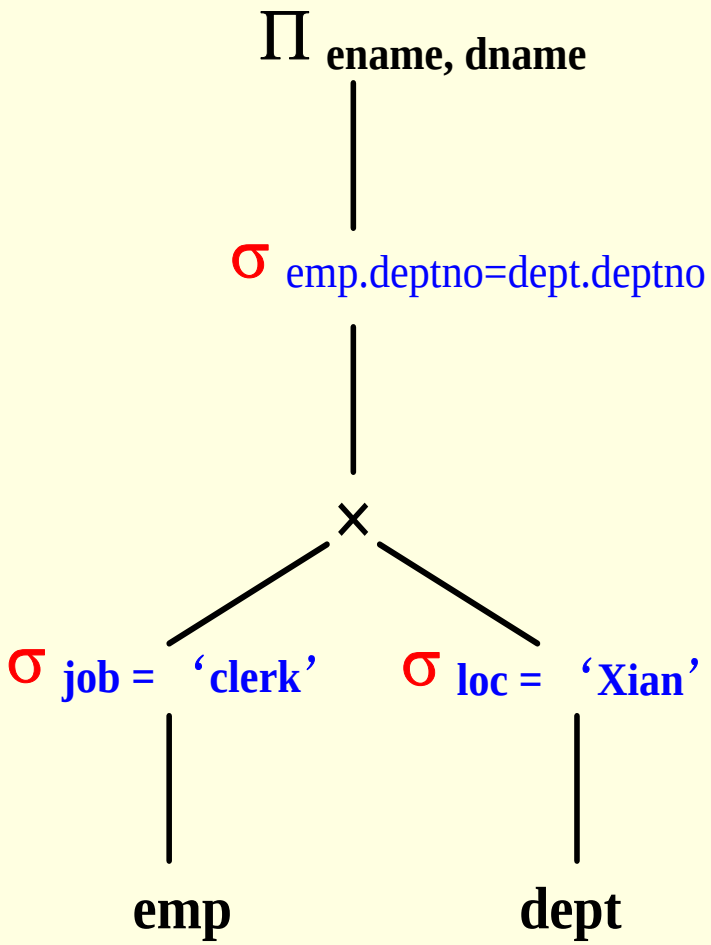
- 代数优化的基本步骤和规则
  - 以SELECT子句对应投影操作，以FROM子句对应笛卡儿乘积，以WHERE子句对应选择操作，生成原始查询树。
  - 应用变换原则，尽可能将选择条件移向树叶方向；
  - 先做小关系间的连接 / 笛卡尔积，后做大关系间的连接 / 笛卡尔积；
  - 将“笛卡尔积＋选择”合并为“连接”；
  - 对每个叶节点加必要的投影，以消除对查询无用的属性。

6.2 代数优化



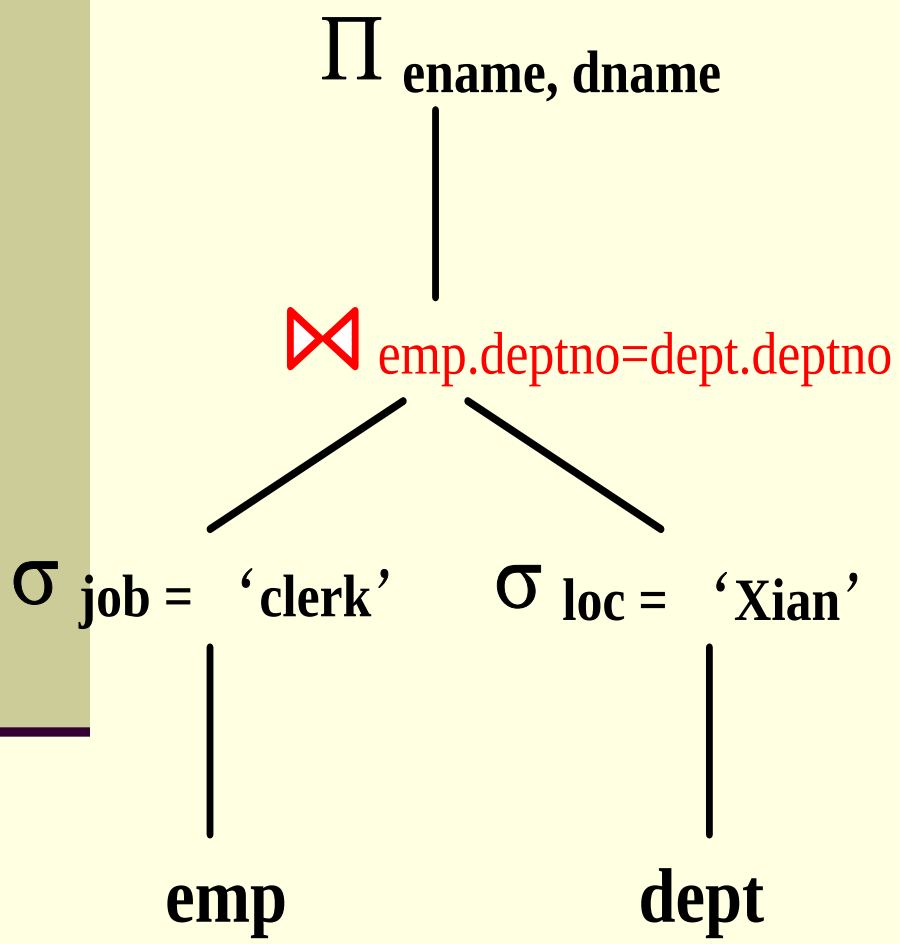
C = emp.deptno=dept.deptno AND
job=’clerk’ AND loc=’Xian’

1

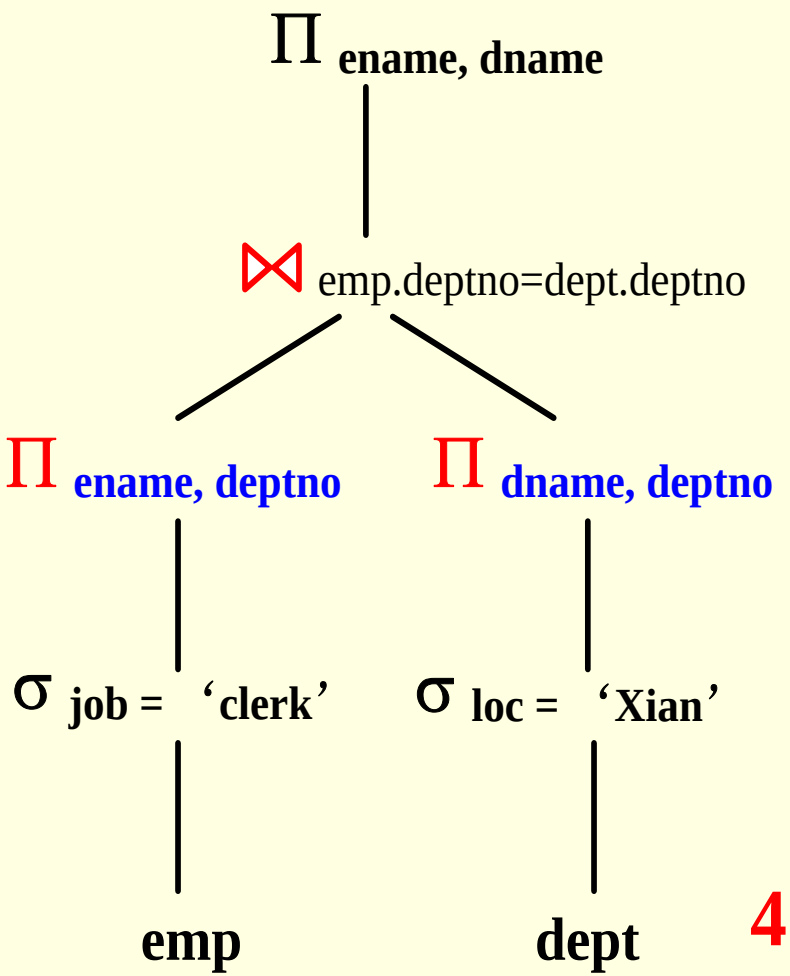


2

6.2 代数优化


Π ename, dname
⋈ emp.deptno=dept.deptno
σ job = ‘clerk’
σ loc = ‘Xian’
emp
dept

3


Π ename, dname
⋈ emp.deptno=dept.deptno
Π ename, deptno
Π dname, deptno
σ job = ‘clerk’
σ loc = ‘Xian’
emp
dept

4

目录 Contents

- 6.1 概述
- 6.2 代数优化
- 6.3 依赖于存取路径的规则优化
- 6.4 代价估算优化

6.3 依赖于存取路径的规则优化

- 一、目标
  - 尽量提高磁盘I/O的效率。
- 二、方法
  - 在存取表数据时，基于存取路径选择规则，选择效率最高的存取路径。
  - 存取路径：e.g. Oracle 中预定义了15种可能的表数据存取路径：

ACCESS PATH	RANK
用**ROWID**存取单行	**1**
用簇集连接存取单行	**2**
…	…
在索引列上的有限范围查找	**10**
…	…
全表扫描	**15**

6.3 依赖于存取路径的规则优化

- 三、策略
  - 对具体的表存取，根据系统预定义的选择条件，先判别可用的所有存取路径；
  - 在所有可用的存取路径中选择RANK值最小者。

6.3 依赖于存取路径的规则优化

- 例子
  - 存取路径“用簇集连接存取单行”的选择条件是：

此路经适合于连接存储于同一簇集中的表，且如下两个条件为“真”：
✓WHERE子句中有条件：“一个表的簇集键列等于另一表的对应列”；
✓WHERE子句中有条件：“连接后只返回单行结果”。

  - 若表emp与dept已在deptno列上建簇集，则下列查询可用以上存取路径：
    - SELECT empno, ename, dname, loc
      FROM emp, dept
      WHERE emp.deptno=dept.deptno AND empno=1001 ;

6.3 依赖于存取路径的规则优化

- Oracle中优化方法与目标的选择
  - Oracle优化器对SQL语句选择何种优化方法与目标，取决于以下因素：
    - OPTIMIZER_MODE初始化参数(RULE / COST)；
    - 数据字典(DD)中的统计数据；
    - ALTER SESSION命令的OPTIMIZER_GOAL参数(CHOOSE / ALL_ROWS / FIRST_ROWS / RULE)；
    - SQL语句中的优化提示(hints)

目录 Contents

- 6.1 概述
- 6.2 代数优化
- 6.3 依赖于存取路径的规则优化
- 6.4 代价估算优化

6.4 代价估算优化

- 一、目标
  - 选择代价最小的执行计划
- 二、方法与策略
  - 针对各种可能的查询操作实现方案（包括实现算法及可用的存取路径），根据代价估算模型分别计算出相应的执行代价，选择代价最小者作为最终执行计划。

6.4 代价估算优化

- 代价估算模型
  - 一个操作的代价：

    $C = C_{I/O}+C_{CPU}+C_{COMM}$

    $\approx C_{I/O}$

    $= (D_0+D_1X) \times$ I/O次数

    $\approx D_0 \times$ I/O次数
  - 一个查询执行计划的代价：

    $C_{EP}=\sum$I/O次数$=\sum$I/O数据块数
  - 所以，比较一个查询操作的多个实现方案的相对代价时，只需比较它们的I/O次数或I/O数据块数。

[注]：

D_0 — 每次I/O的寻道、等待时间

D_1 — 每字节数据的传输时间

X — 每次I/O的字节数

一般地，$D_0 >> D_1X$，且对一个存储设备，可认为D_0为常数。

6.4 代价估算优化

- 统计数据
  - 要计算I/O次数或I/O数据块数，必须依赖一系列有关的存储结构参数，这些参数作为统计数据存储于数据字典（DD）中，DBMS可以访问它们。
  - DBA还可用扩充的SQL命令（e.g. Oracle中ANALYZE命令）来计算指定结构的统计数据，并将计算结果存储于数据字典的相应基表中。
  - e.g. Oracle中关于“索引”的统计数据：
    - 索引B树的深度（精确值）
    - B树叶块的数目
    - 不同索引值的数目
    - 每个索引值的叶块之平均数
    - 每个索引值的数据块之平均数
    - 聚集因子（索引值的行如何排序好）
    - 以上数据存储于DD中关于INDEXS的基表中。

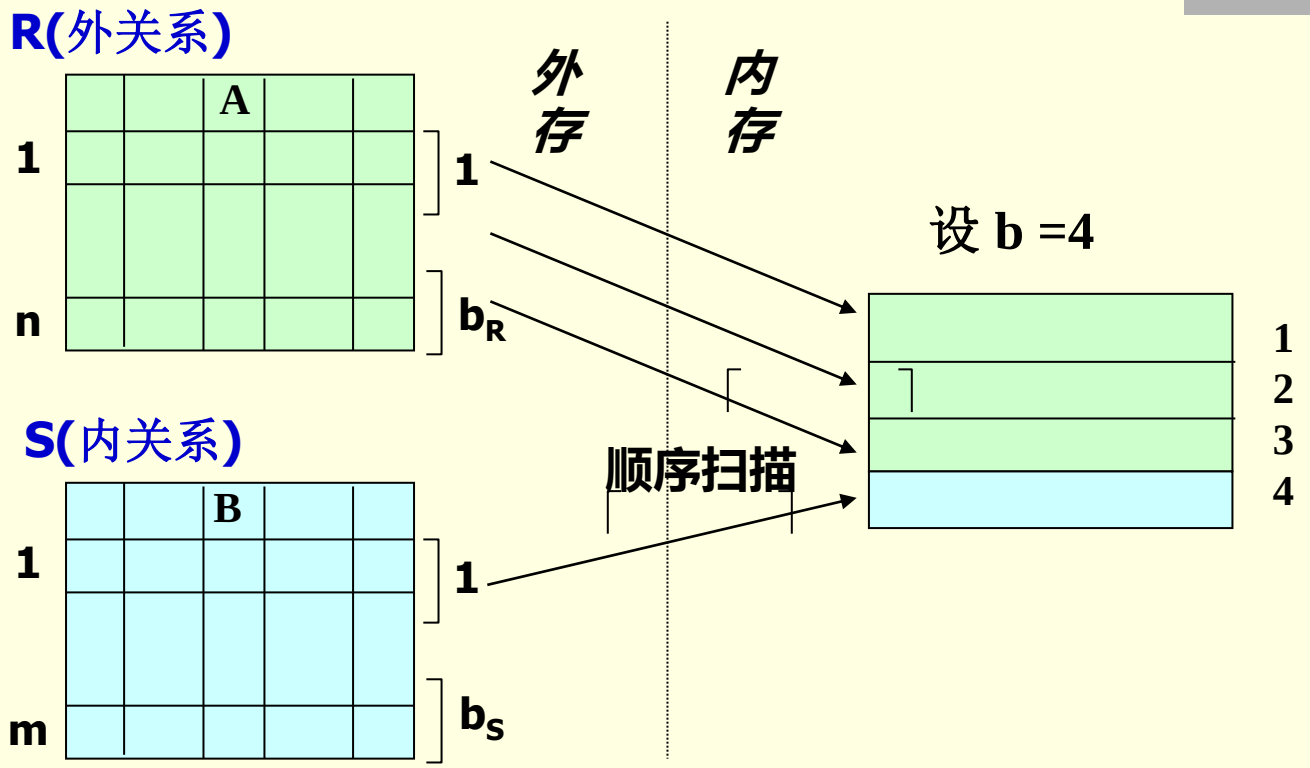
6.4 代价估算优化

- 查询操作的实现算法
  - 各种单个查询操作（如：选择、连接、投影、集合操作）可有多种实现算法，每一种实现算法有相应的存储结构/存取路径选择策略。
  - 一个SQL SELECT查询可看作是各种操作的组合。按组合操作执行查询可省去创建许多临时文件，因而也省去了许多I/O操作。

6.4 代价估算优化

- 例子
  - 连接操作的实现算法与代价估算
  - 连接操作：$R \bowtie_{R.A=S.B} S$ 有四种实现算法
    - ①嵌套循环法(Nested Loop)
    - 实际上，不是以元组(行)为单位I/O的，而是以数据块为单位I/O的。
    - 设b_R—R的物理块数；b_S—S的物理块数；b—内存缓冲区中可用的块数，(其中b-1块用于外关系，1块用于内关系)。

6.4 代价估算优化



则，完成连接所需访问的数据块数（相对代价）：$C = b_R + b_R / (b-1) \times b_S$

若内、外关系交换，则相对代价为：$C = b_S + b_S / (b-1) \times b_R$

故应将物理块少者作为外关系。

6.4 代价估算优化

- ②利用索引或散列寻找匹配元组法
- 若内关系上有索引或散列(簇集)，则可利用这样的存取路径来代替顺序扫描。
- 设关系S的B属性列上有(单表)索引簇集，且：

n_R—关系R的元组数；n_S—关系S的元组数；b_R—关系R的物理块数；b_S—关系S的物理块数；P_S—关系S的块因子（即一块中元组数）；

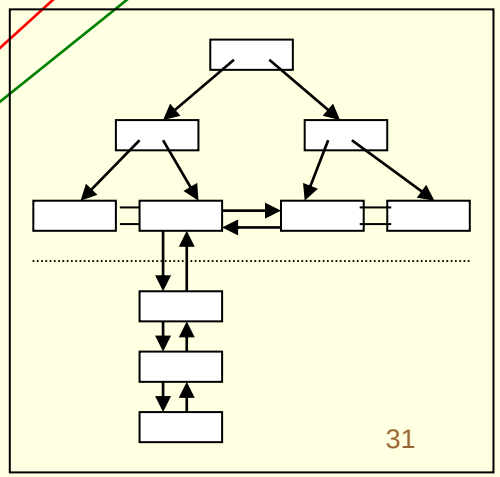
L_B—索引B树的深度；N_B—属性B在关系S中取多少不同的值；

- 完成连接的相对代价：$C = b_R + (n_R \times (L_B + (n_S / N_B)/ P_S))$

属性B每种不同值在关系S中所对应的平均元组数

关系S中满足连接条件(S.B=R.A)的元组所占数据块数

关系S中利用索引簇集每次匹配的I/O块数



6.4 代价估算优化

③排序归并法（Sort-merge）

- 完成连接的相对代价：$C = C_{R排序} + C_{S排序} + b_R + b_S$

④散列连接法（Hash Join）