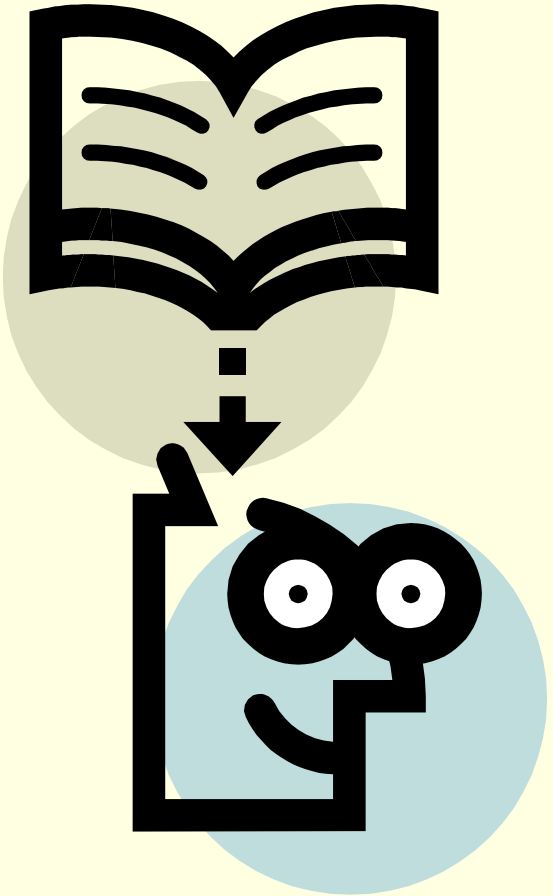
第6章 查询处理和优化
Chapter 6 Query Processing & Optimization

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（简介）



6.1 概述

- 一、查询与查询处理
  - 查询（query）：数据库的基本、常用、复杂的操作。包括：
    - 直接的SELECT操作：
      - **SELECT … FROM … WHERE … (SELECT…) … ;**
    - 间接的SELECT操作：
      - **INSERT INTO … SELECT… ;**
      - **DELETE FROM … WHERE … (SELECT…) … ;**
      - **UPDATE … SET… (SELECT…) … WHERE … (SELECT…) … ;**
      - **CREATE VIEW… AS SELECT … ;**

6.1 概述

- 一、查询与查询处理（续）
  - 查询处理（query processing）：从DBMS接受一个查询请求到返回结果的整个处理过程。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，查询处理的最关键、核心步骤是查询优化。

6.1 概述

Source: Abraham Silberschatz, et al., Database System Concepts, 6th ed., Page 538

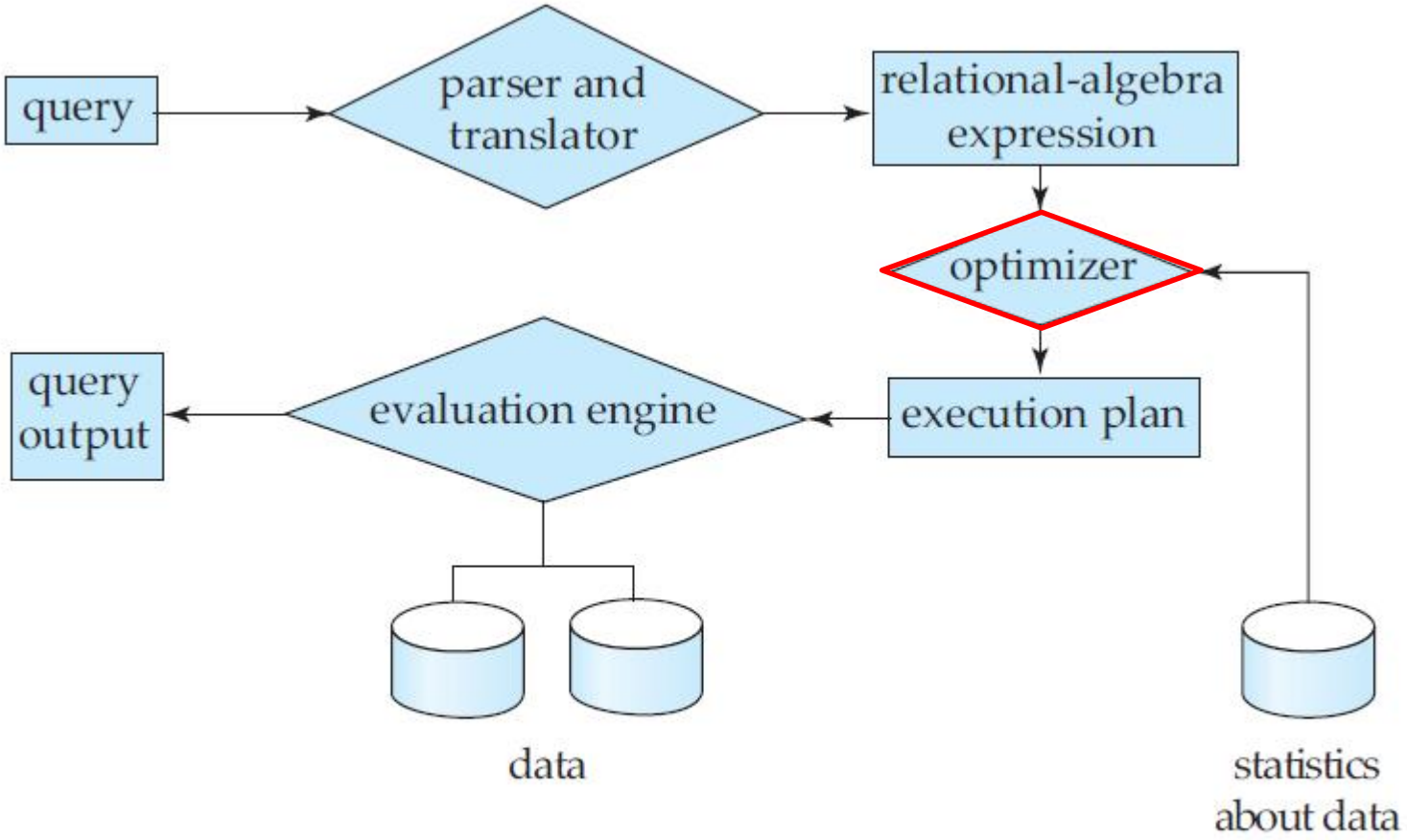


Figure 12.1 Steps in query processing.

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高的执行计划（execution plan）（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（续）
  - 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，而不必考虑查询效率问题。这是使用高度非过程化、声明式语言（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”QL/DML语句更为有效。**

6.1 概述

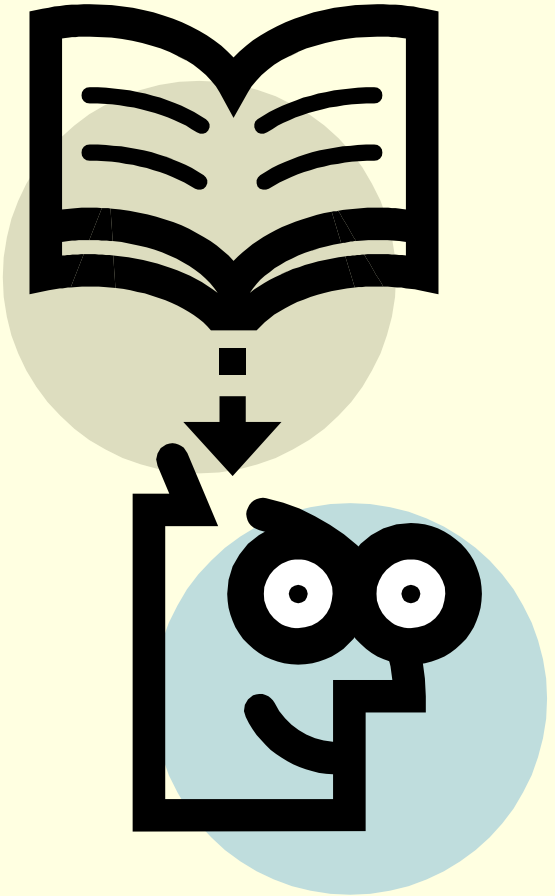
- 二、查询优化及其方法（续）
  - 查询优化的方法
    - **两个层次**
      - 代数层：将查询所对应的关系代数表达式以语法树来表示，然后等价变换之，目的是减小查询过程中的中间结果大小。
      - 物理层：选择高效的存取路径来访问表数据，目的是提高磁盘I/O的效率。
    - **两种策略**
      - 基于规则（rules）：运用启发式规则（heuristic rules）为查询确定一个理论上认为“最优的”执行计划。
      - 基于代价估算（cost estimates）：为查询确定几个理论上认为“较优的”执行计划，然后根据数据字典中的参数分别估算它们的执行代价（时间开销），从中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（续）
  - 查询优化的方法（续）
    - 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中承担优化的模块称优化器（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所选择的执行计划（e.g. Oracle中通过EXPLAIN PLAN命令来实现）；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中插入优化提示来实现）。

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（简介）



6.2 代数优化

- **目标**
  - 尽量减小查询中间结果的大小。
- **方法**
  - 用语法树表示查询【称原始语法树】，
  - 基于变换策略，运用关系代数的等价变换规则（equivalence rules on relational-algebra expressions）（教材第120页中规则(1)~(11)），对语法树中的关系代数操作的次序进行等价变换，形成优化后语法树。

6.2 代数优化

- **变换策略：**
  - 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

- 常用的等价变换规则（教材第120页中规则(1)~(11)）
  - 规则1： $\sigma_{c1\ \mathrm{AND}\ c2\ldots\ \mathrm{AND}\ cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(\ldots\sigma_{cn}(R)\ \ldots))$
  - 规则2： $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$
  - 规则3： $\Pi_{list1}(\Pi_{list2}(\ldots\ \Pi_{listn}(R)\ldots)) \equiv \Pi_{list1}(R)$
    - 如果n个属性集满足list1$\subseteq$ list2 $\subseteq\ldots\subseteq$ list$n$
  - 规则4： $\Pi_{list}(\sigma_{c}(R)) \equiv \sigma_{c}(\Pi_{list}(R))$
    - 如果选择条件c中涉及的属性全部包含在属性集list中
  - 规则5： $R \bowtie S \equiv S \bowtie R$
  - 规则6： $\sigma_{c}(R \bowtie S) \equiv (\sigma_{c}(R)) \bowtie S$
    - 如果选择条件c中涉及的属性都是 R中的属性，即 $\mathrm{Attr}(c) \subseteq \mathrm{Attr}(R)$

6.2 代数优化

- 常用等价变换规则（续）
  - 规则7： $\sigma_{c1\ AND\ c2}(R \bowtie S) \equiv \sigma_{c1}(R) \bowtie \sigma_{c2}(S)$
    - 如果c1中涉及的属性都是R中的属性，c2中涉及的属性都是S中的属性，即：$Attr(c1) \subseteq Attr(R)$, $Attr(c2) \subseteq Attr(S)$
  - 规则8： $\Pi_{list}(R \bowtie_{c} S) \equiv (\Pi_{list1}(R)) \bowtie_{c} (\Pi_{list2}(S))$
    - 如果属性集 $Attr(c) \subseteq list$，且$list = list1 \cup list2$，其中，$list1 \subseteq Attr(R)$，$list2 \subseteq Attr(S)$。
    - 含义：如果将投影操作提前到连接前执行，那么投影操作必须保留参与连接的两个关系中与连接条件有关的属性——若属性集$list = list1 \cup list2$没有完全包含属性$Attr(c)$，则在交换投影与连接操作的顺序时，必须把未包含的属性补充到相应关系的投影属性集list1、list2中。

6.2 代数优化

- 常用等价变换规则（续）
  - 规则9： $\sigma_c(R\ \theta\ S) \equiv (\sigma_c(R))\ \theta\ (\sigma_c(S))$
    - 集合运算$\theta \in \{\cup, \cap, -\}$
  - 规则10： $\Pi_{list}(R \cup S) \equiv \Pi_{list}(R) \cup \Pi_{list}(S)$
  - 规则11： $R\ \theta\ (S\ \theta\ T) \equiv (R\ \theta\ S)\ \theta\ T$
    - 运算$\theta \in \{\bowtie, \times, \cup, \cap\}$

6.2 代数优化

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

下列4个关系代数表达式似乎都等价地表示了上述查询：

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

$$Q2= \Pi_{ename,\ dname}(\sigma_{emp.deptno=dept.deptno}(\sigma_{job='clerk'}(emp) \times \sigma_{loc='Xian'}(dept)))$$

$$Q3= \Pi_{ename,\ dname}(\sigma_{job='clerk'}(emp) \bowtie_{emp.deptno=dept.deptno} \sigma_{loc='Xian'}(dept))$$

$$Q4= \Pi_{ename,\ dname}(\Pi_{ename,\ deptno}(\sigma_{job='clerk'}(emp)) \bowtie_{emp.deptno=dept.deptno} \Pi_{deptno,\ dname}(\sigma_{loc='Xian'}(dept)))$$

到底哪个关系代数表达式更优呢？ ——Q4最优！

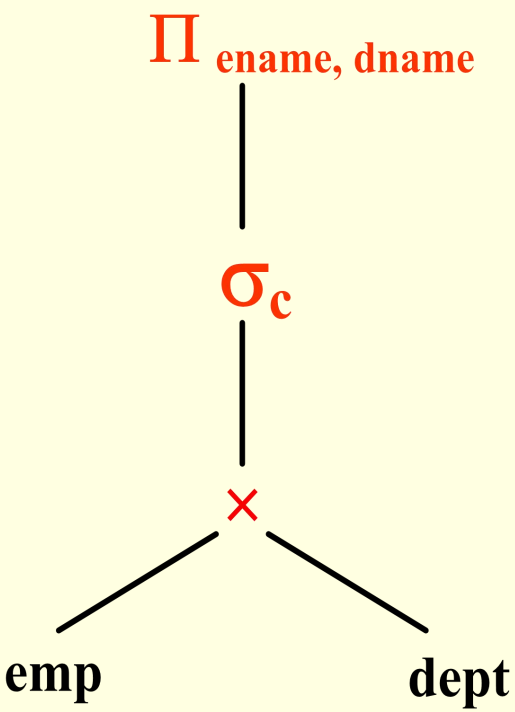
6.2 代数优化

- 代数优化策略与基本步骤：
  - 以SELECT子句对应投影操作，以FROM子句对应笛卡儿积，以WHERE子句对应选择操作，构建原始查询树（语法树）。
  - 运用等价变换规则尽可能将选择条件移向叶节点方向；
  - 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；【具体例子见教材】
  - 将“笛卡尔积＋选择”合并为“连接”；
  - 对每个叶节点加必要投影，以消除对查询无用的属性，最终形成优化后的查询树（语法树）

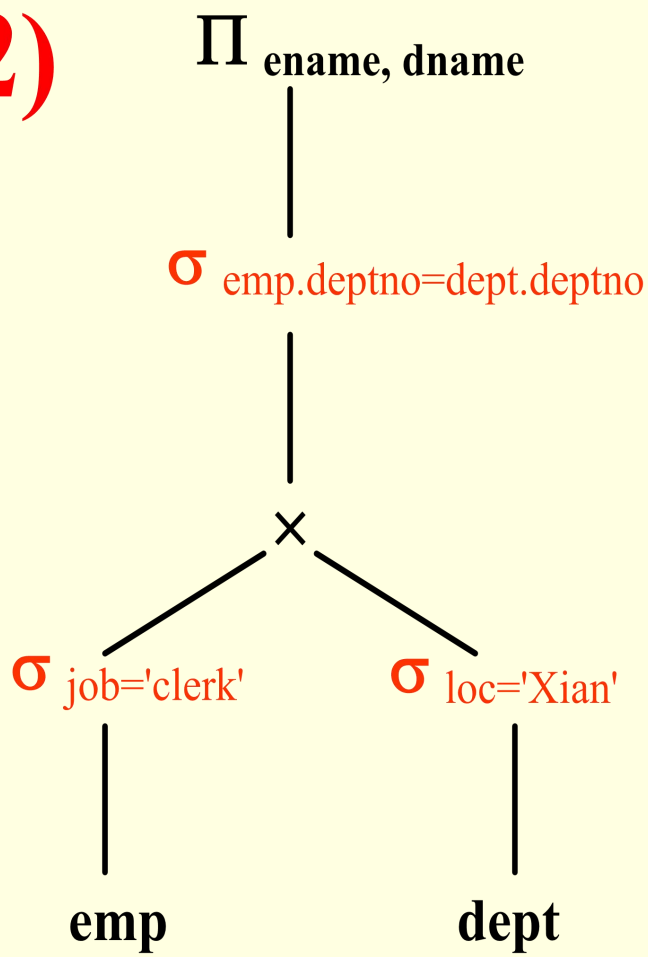
6.2 代数优化

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

(1)



(2)

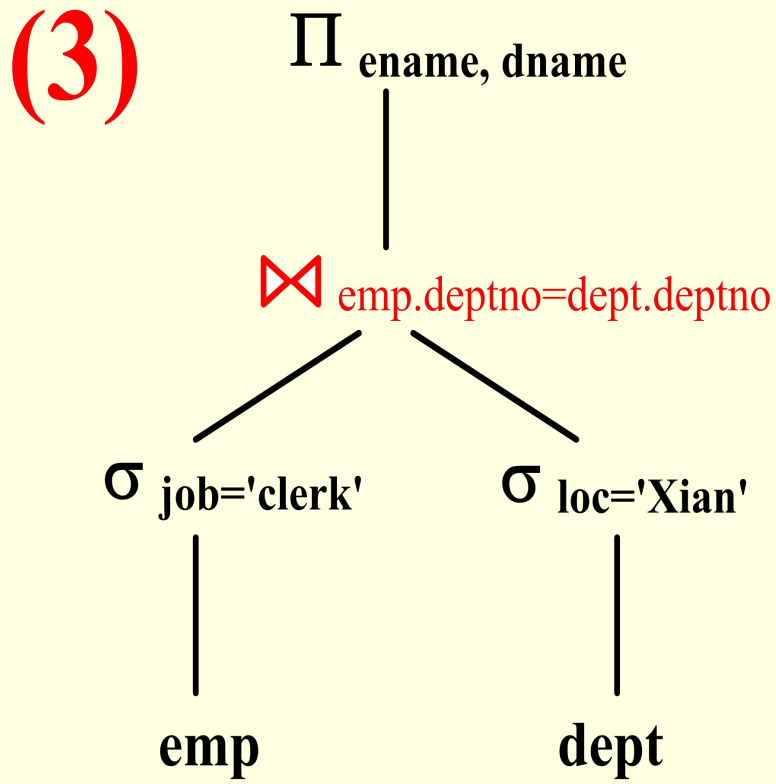


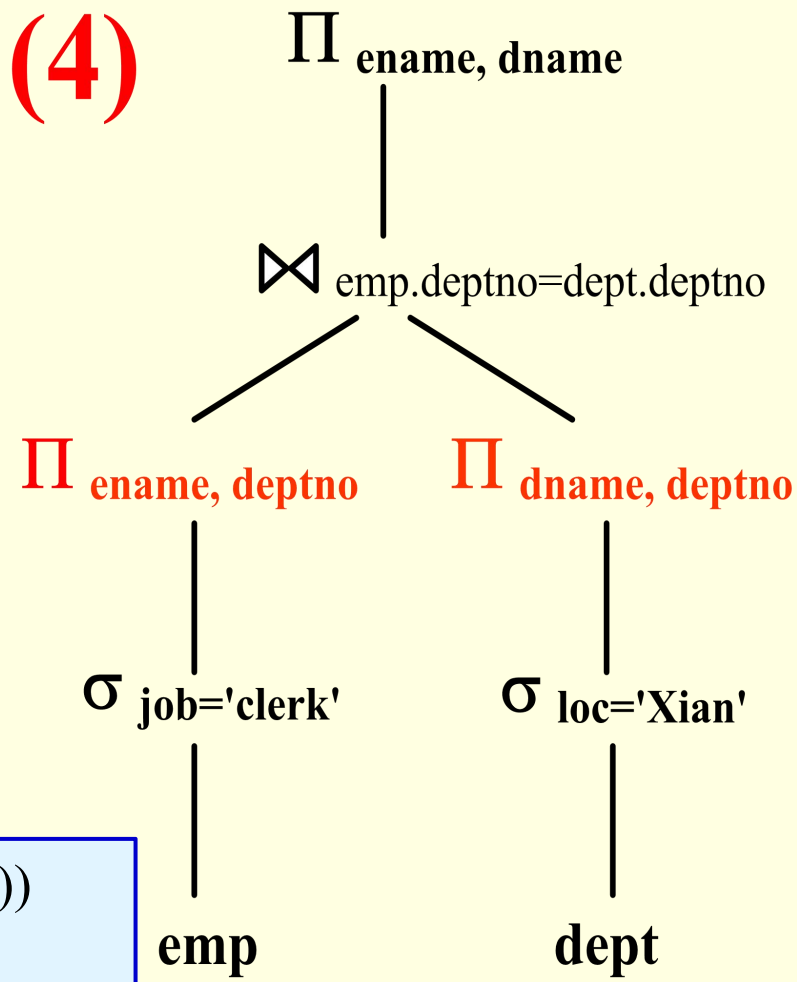
上述选择条件：

c = emp.deptno=dept.deptno AND
job='clerk' AND loc='Xian'

6.2 代数优化

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



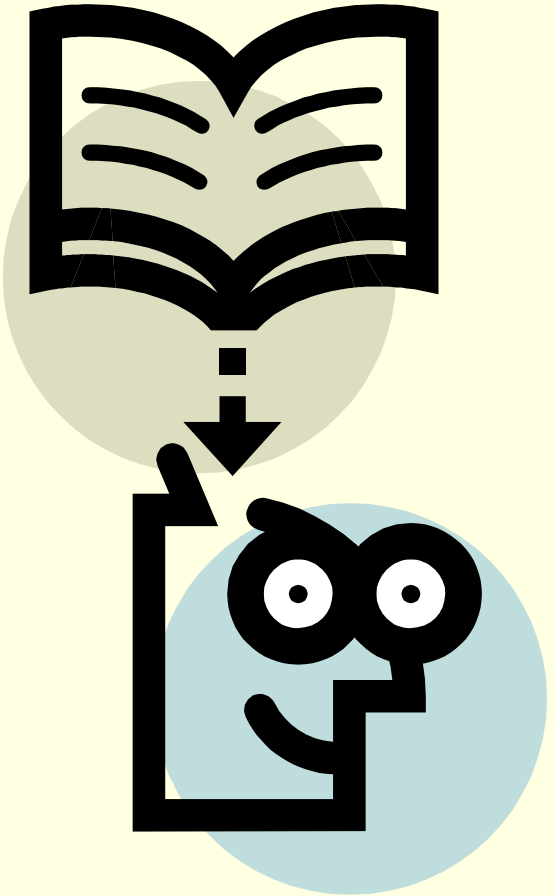


$\Pi_{ename,\ dname}(\Pi_{ename,\ deptno}(\sigma_{job='clerk'}(emp)) \bowtie_{emp.deptno=dept.deptno} \Pi_{dname,\ deptno}(\sigma_{loc='Xian'}(dept)))$

优化后

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（简介）



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，根据预定义的存取路径选择（access path selection）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

快 ↓ 慢

- 三、策略：对具体的表存取，根据系统预定义的存取路径选择条件，先判别所有可用的存取路径，然后在其中选择RANK值最小的存取路径。

6.3 依赖于存取路径的规则优化

- 例：
  - 存取路径“用簇集连接存取单行”的选择条件是：

此路经适用于对存储于同一簇集中的表进行连接，且满足如下两个条件：
✓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：导致“连接后只返回单行结果”。

  - 若两个表emp与dept已在deptno列上创建了簇集，则下列查询可用上述存取路径：

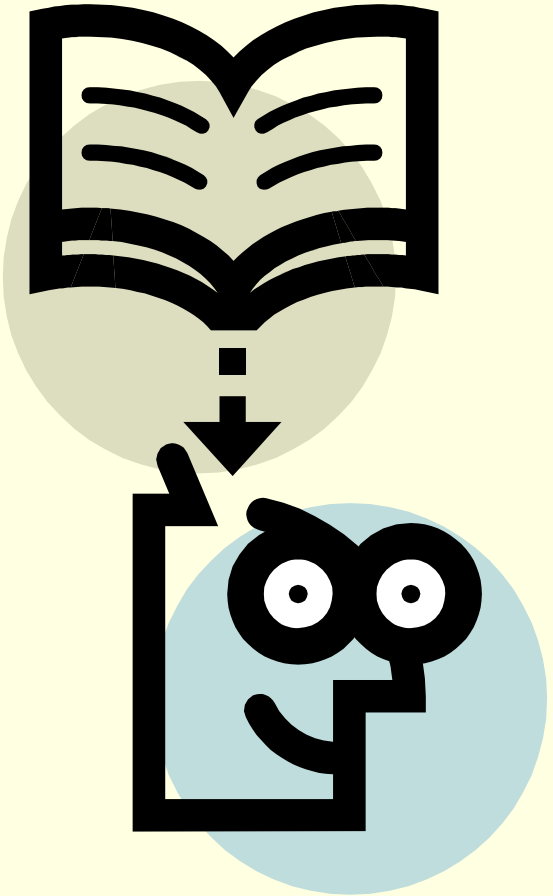
```
SELECT empno, ename, dname, loc
FROM   emp, dept
WHERE emp.deptno=dept.deptno AND empno=1001 ;
```

6.3 依赖于存取路径的规则优化

- 【补充】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；
    - 数据字典中的统计数据；
    - ALTER SESSION命令的OPTIMIZER_GOAL参数：CHOOSE | ALL_ROWS | FTRST_ROWS | RULE；
    - 用户在SQL语句中给DBMS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（简介）



6.4 代价估算优化

- 一、目标
  - 选择（估算的）代价最小的查询执行计划（query-execution plan）。
- 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（cost estimates）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的执行计划。

6.4 代价估算优化

- 代价估算模型
  - 一个查询操作的代价：

$$C = C_{I/O} + C_{CPU} + C_{COMM}$$
$$\approx C_{I/O}$$
$$= (D_0 + D_1 X) \times \text{I/O次数}$$
$$\approx D_0 \times \text{I/O次数}$$

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，$D_0 >> D_1 X$，且对同一个存储设备，可认为D_0为常数。

  - 一个查询执行计划EP的代价：

$$C_{EP} = \sum \text{I/O次数} \quad \text{或者} = \sum \text{I/O数据块数}$$

  - 由此可见，比较一个查询的多个执行计划（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。

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，必须依赖相关的存储结构参数，它们作为统计数据存储于数据字典（DD）中，DBMS可以获得。
  - 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（DD）的相应基表中。 例如：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）

    以上统计数据存储于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。

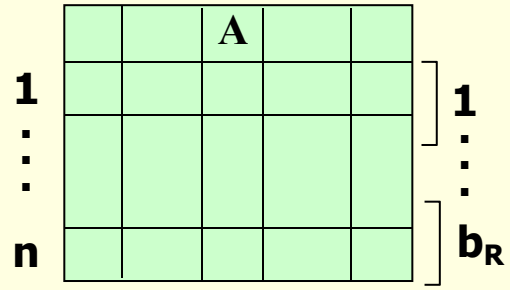
6.4 代价估算优化

- 例：连接操作的实现算法与代价估算 （教材P125-127）

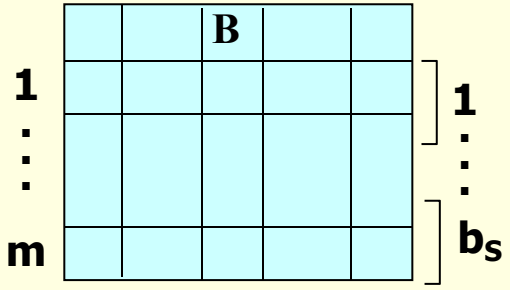
  连接操作 $R \bowtie_{R.A=S.B} S$ 有四种实现算法：

① 嵌套循环法（nested loop）

外关系R：共n行占用b_R块



内关系S：共m行占用b_S块

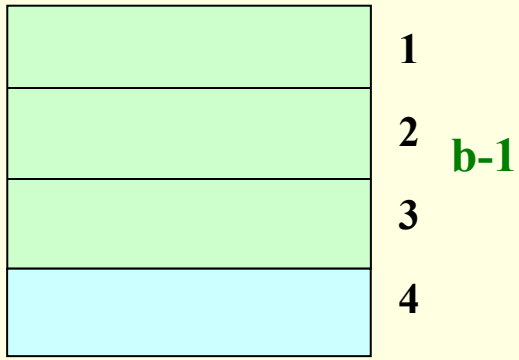


外存（顺序扫描） 内存

扫描一次

扫描多次

设内存缓冲区中可用的块数b = 4（其中b-1块用于外关系，剩余1块用于内关系）



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

若内、外关系交换，则相对代价为：

$C' = b_S + \lceil b_S / (b-1) \rceil \times b_R$

——故应将物理块少者作为外关系

6.4 代价估算优化

嵌套循环法的相对代价：

$$C = b_R + \lceil b_R / (b-1) \rceil \times b_S$$

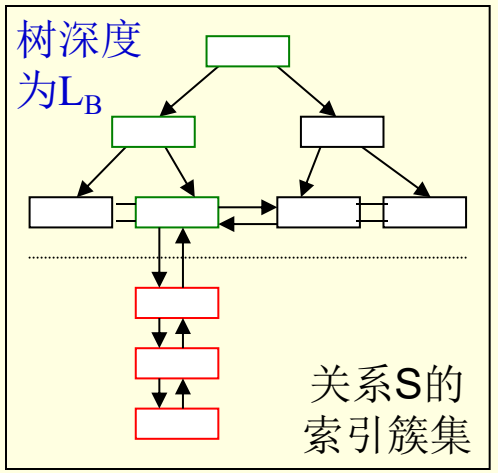
② 利用索引（或散列）寻找匹配元组法

- 若内关系S上有索引或散列（簇集），则可利用这样的（快速）存取路径来代替前面的顺序扫描。
- 设内关系S的属性列B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（1块存储的元组数）；

  L_B—索引B树的深度；N_B—属性B在关系S中取多少种不同的值；
- 则完成连接的相对代价为：

  $$C = b_R + (n_R \times (L_B + (n_S / N_B) / P_S))$$



关系S中属性B的每种不同值所对应的平均元组数

满足连接条件R.A=S.B的关系S中元组所占的数据块数

关系R中每个元组去匹配关系S中元组所需的I/O块数

关系R中所有元组去匹配关系S中元组所需的I/O总块数

6.4 代价估算优化

③ 排序归并法（sort-merge）：代价 $C = C_{R排序} + C_{S排序} + b_R + b_S$

如果 R,S 按连接属性排序，则可按序比较 $R.A$ 和 $S.B$，找出所有匹配的元组。在此方法中，R 和 S 都只需扫描一次。图 6-4 是排序归并法示意图，图中假定 A,B 属性的域为正整数，在比较时首先检查 A,B 中的小者，如 $S.B$ 中的 1，看看有无匹配对象。若有，就组合成连接元组；如果没有，就跳过它。如果 A,B 不是主键，则 $R.A$ 和 $S.B$ 中可能有相同的属性值，如图 6-4中的 3。按照连接运算的要求，$R.A=3$ 的每个元组须与 $S.B=3$ 的所有元组匹配。

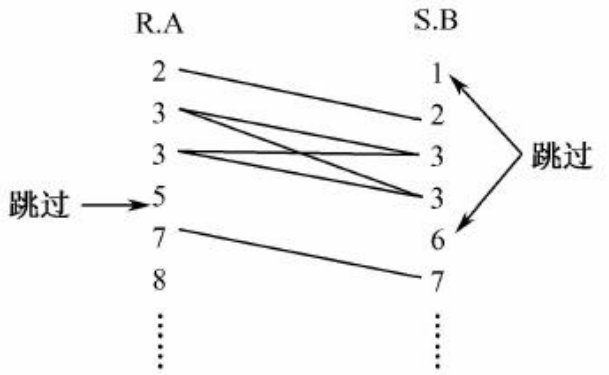


图 6-4　排序归并法示意图

Source: 主教材Page 126

④ 散列连接法（hash join）：略

The End

- 第六章作业：
  - 【补充】将主教材Page 64中例3-4查询语句的查询条件中去掉YEAR(BDATE)=1986，然后用语法树表示该修改后查询语句的代数优化过程。【提示：这是连接3个基表的复杂查询，需要运用“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”的变换策略——请参考主教材中例6-1】
  - 提醒：请在截止时间（10月30日23:59）之前提交答案！