# Video 14-2：关系算子的实现

# 实现查询操作的算法示例

关系代数是关系数据库的抽象语言，理解如何实现关系代数操作有助于我们理解查询优化的过程。

1 选择操作的实现

2 连接操作的实现

# 1.选择操作的实现

- 选择操作典型实现方法：
  - （1） 全表扫描方法 (Table Scan)
    - 对查询的基本表顺序扫描，逐一检查每个元组是否满足选择条件，把满足条件的元组作为结果输出
    - 适合小表，不适合大表
  - （2）索引扫描方法 (Index Scan)
    - 适合于选择条件中的属性上有索引(例如B+树索引或Hash索引)
    - 通过索引先找到满足条件的元组主码或元组指针，再通过元组指针直接在查询的基本表中找到元组

# 选择操作的实现（续）

❖ [例9.1] SELECT *

FROM Student

WHERE <条件表达式>

考虑<条件表达式>的几种情况：

C1：无条件；

C2：Sno＝'201215121'；

C3：Sage>20；

C4：Sdept＝'CS' AND Sage>20；

# 选择操作的实现（续）

- 全表扫描算法
  - 假设可以使用的内存为M块，全表扫描算法思想：

① 按照物理次序读Student的M块到内存

② 检查内存的每个元组t，如果满足选择条件，则输出t

③ 如果student还有其他块未被处理，重复①和②

# 选择操作的实现（续）

❖ 索引扫描算法

[例9.1-C2] SELECT *

FROM Student

WHERE Sno='201215121'

- 假设Sno上有索引(或Sno是散列码)
- 算法：
  - 使用索引(或散列)得到Sno为‘201215121’元组的指针
  - 通过元组指针在Student表中检索到该学生

# 选择操作的实现（续）

[例9.1-C3] SELECT *

FROM Student

WHERE Sage>20

- 假设Sage 上有B+树索引
- 算法：
  - 使用B+树索引找到Sage=20的索引项，以此为入口点在B+树的顺序集上得到Sage>20的所有元组指针
  - 通过这些元组指针到student表中检索到所有年龄大于20的学生

# 选择操作的实现（续）

❖ [例9.1-C4] SELECT *

FROM Student

WHERE Sdept='CS' AND Sage>20;

- 假设Sdept和Sage上都有索引

# 选择操作的实现（续）

- 算法一：分别用Index Scan找到Sdept＝’CS’的一组元组指针和Sage>20的另一组元组指针
  - 求这两组指针的交集
  - 到Student表中检索
  - 得到计算机系年龄大于20的学生

# 选择操作的实现（续）

- 算法二：找到Sdept='CS'的一组元组指针，
  - 通过这些元组指针到Student表中检索
  - 并对得到的元组检查另一些选择条件(如Sage>20)是否满足
  - 把满足条件的元组作为结果输出。

# 选择操作的实现（小结）

选择算子的处理要考虑到 选择的<条件表达式>具体情况，采用不同的策略：

C1：无条件； 采用全表扫描

C2：Sno＝‘201215121’；结果集小的情况下，利用索引

C3：Sage>20；结果集合太大的情况下，直接全表扫描

C4：Sdept＝‘CS’ AND Sage>20； 多个条件的情况下比较复杂，会分别考虑每个条件再合并结果，也可能逐一顺序考虑这些条件，甚至条件太复杂的时候直接扫描表格。

# 2.连接操作的实现

- 连接操作是查询处理中最耗时的操作之一
- 本节只讨论等值连接(或自然连接)最常用的实现算法
- [例9.2]

```
SELECT *
FROM   Student, SC
WHERE Student.Sno=SC.Sno;
```

# 连接操作的实现（续）

（1）嵌套循环算法(nested loop join)

（2）排序-合并算法(sort-merge join 或merge join)

（3）索引连接(index join)算法

（4）Hash Join算法

# 连接操作的实现（续）

（1）嵌套循环算法(nested loop join)

- 对外层循环(Student表)的每一个元组(s)，检索内层循环(SC表)中的每一个元组(sc)
- 检查这两个元组在连接属性(Sno)上是否相等
- 如果满足连接条件，则串接后作为结果输出，直到外层循环表中的元组处理完为止。

❖参见爱课程网9.1节动画《连接操作的实现(1)--嵌套循环》

# 连接操作的实现（示意）


图 嵌套循环连接方法示意图

# 连接操作的实现（续）

（2）排序-合并算法(sort-merge join 或merge join)

- 如果连接的表没有排好序，先对Student表和SC表按连接属性Sno排序
- 取Student表中第一个Sno，依次扫描SC表中具有相同Sno的元组
- 当扫描到Sno不相同的第一个SC元组时，返回Student表扫描它的下一个元组，再扫描SC表中具有相同Sno的元组，把它们连接起来
- 重复上述步骤直到Student 表扫描完

# 连接操作的实现（示意）

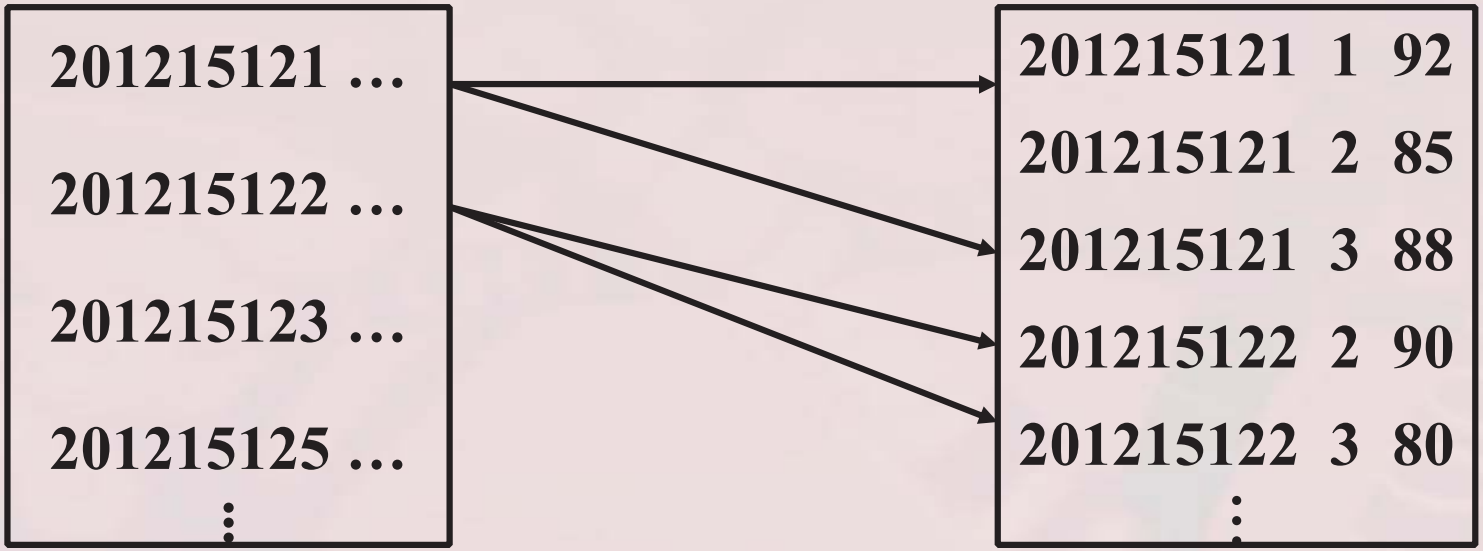


图9.2 排序-合并连接方法示意图

# 连接操作的实现（续）

- Student表和SC表都只要扫描一遍
- 如果两个表原来无序，执行时间要加上对两个表的排序时间
- 对于大表，先排序后使用排序-合并连接算法执行连接，总的时间一般仍会减少
- 参见爱课程网9.1节动画《连接操作的实现（2）--排序合并》

# 连接操作的实现（续）

（3）索引连接(index join)算法

- 步骤：

① 在SC表上已经建立属性Sno的索引。

② 对Student中每一个元组，由Sno值通过SC的索引查找相应的SC元组。

③ 把这些SC元组和Student元组连接起来

循环执行②③，直到Student表中的元组处理完为止

❖ 参见爱课程网9.1节动画《连接操作的实现(4)-- 索引连接 》

# 连接操作的实现（续）

## （4）Hash Join算法

- 把连接属性作为hash码，用同一个hash函数把Student表和SC表中的元组散列到hash表中。
- 划分阶段(building phase, 也称为partitioning phase)
  - 对包含较少元组的表(如Student表)进行一遍处理
  - 把它的元组按hash函数分散到hash表的桶中
- 试探阶段(probing phase,也称为连接阶段join phase)
  - 对另一个表(SC表)进行一遍处理
  - 把SC表的元组也按同一个hash函数（hash码是连接属性）进行散列
  - 把SC元组与桶中来自Student表并与之相匹配的元组连接起来

# 连接操作的实现（续）

- 上面hash join算法前提：假设两个表中较小的表在第一阶段后可以完全放入内存的hash桶中

- 参见爱课程网9.1节动画《连接操作的实现(3)--散列连接 》