第九章 关系系统及其查询优化

本章内容

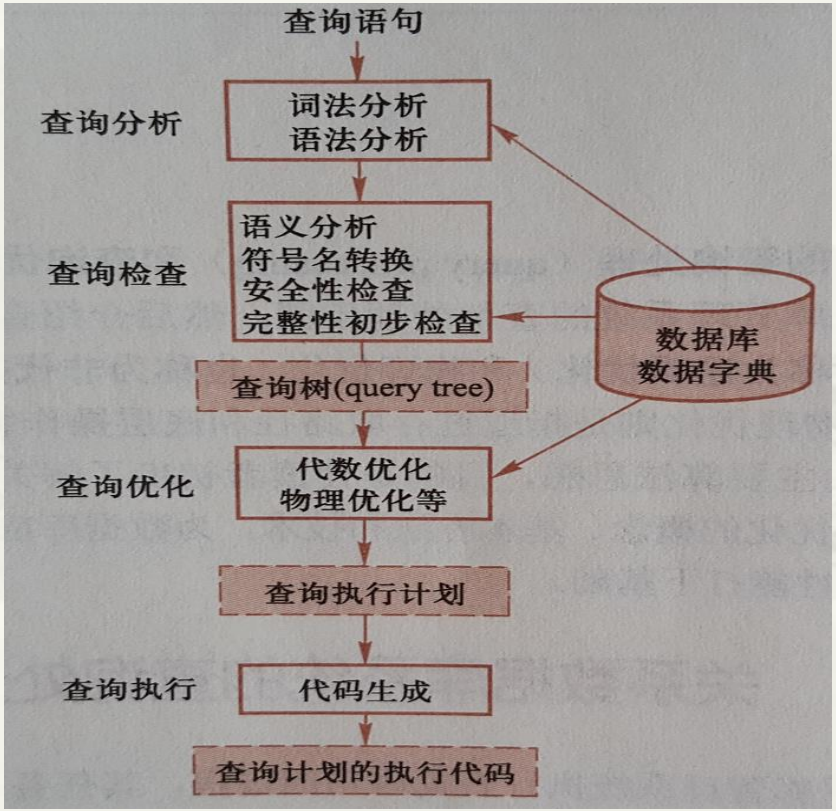
关系数据库系统：

- 查询处理
- 查询优化
  - 代数优化
  - 物理优化

关系数据库系统的查询处理

分为4个步骤：

* 查询分析
* 查询检查
* 查询优化
* 查询执行



1. 查询分析

- 查询分析的任务：对查询语句进行扫描、词法分析和语法分析
  - 词法分析：从查询语句中识别出正确的语言符号
  - 语法分析：进行语法检查

2. 查询检查

- 查询检查的任务
  - 合法权检查
  - 视图转换
  - 安全性检查
  - 完整性初步检查
- 根据数据字典检查名称的有效性，权限合法性。
- 转换对视图的操作。

3. 查询优化

- 查询优化：选择一个高效执行的查询处理策略
- 查询优化分类
  - 代数优化/逻辑优化：指关系代数表达式的优化
  - 物理优化：指存取路径和底层操作算法的选择
- 查询优化的选择依据
  - 基于规则(rule based)
  - 基于代价(cost based)
  - 基于语义(semantic based)

4. 查询执行

- 依据优化器得到的执行策略生成查询执行计划
- 代码生成器(code generator)生成执行查询计划的代码
- 两种执行方法
  - 自顶向下
  - 自底向上

查询操作实现的几个实例

Select * from student
where 〈条件表达式〉

条件表达式的几种情况:

1. 无条件:
2. Sno='200234556'
3. Sage > 20
4. Sdept='cs' and sage >20

选择操作的实现

* 全表扫描方法（Table Scan）

1. 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
2. 适合小表，不适合大表

* 索引扫描方法（Index Scan）

1. 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)
2. 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

实例：SELECT * FROM Student WHERE Sdept='CS' AND Sage>20

假设Sdept和Sage上都有索引

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

实例：SELECT * FROM Student WHERE Sdept='CS' AND Sage>20

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。

连接操作的实现

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

1. 嵌套循环方法
2. 排序合并方法
3. 索引连接方法
4. HASH JOIN 方法

Nested-Loop Join 嵌套循环方法

- 计算 **theta join** 算法 $r \bowtie_\theta s$

  for each **tuple** t_r in r do begin

  for each tuple t_s in s do begin

  test pair (t_r,t_s) to see if they satisfy the join condition θ

  if they do, add $t_r \cdot t_s$ **to the result.**

  end

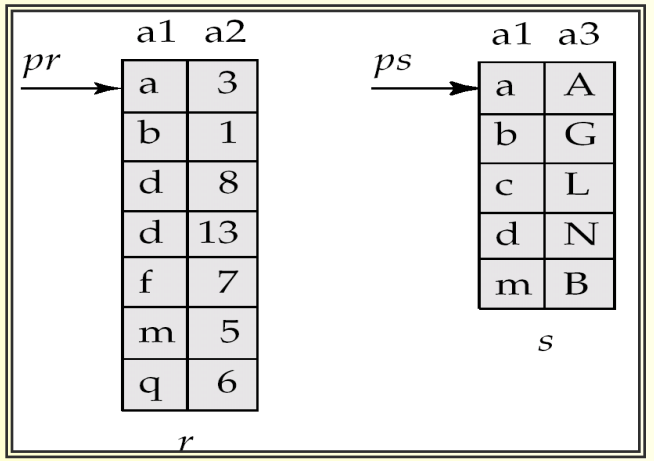
  end

- r 称为outer relation **and** s 称为 inner relation **.**
- 不需要任何索引,用于任何种类的连接条件
- 开销比较大,需要检查两个关系中的每对元组.

Merge-Join排序合并方法

根据连接属性,对两个关系排序 合并与 连接

适用于等值连接或自然连接

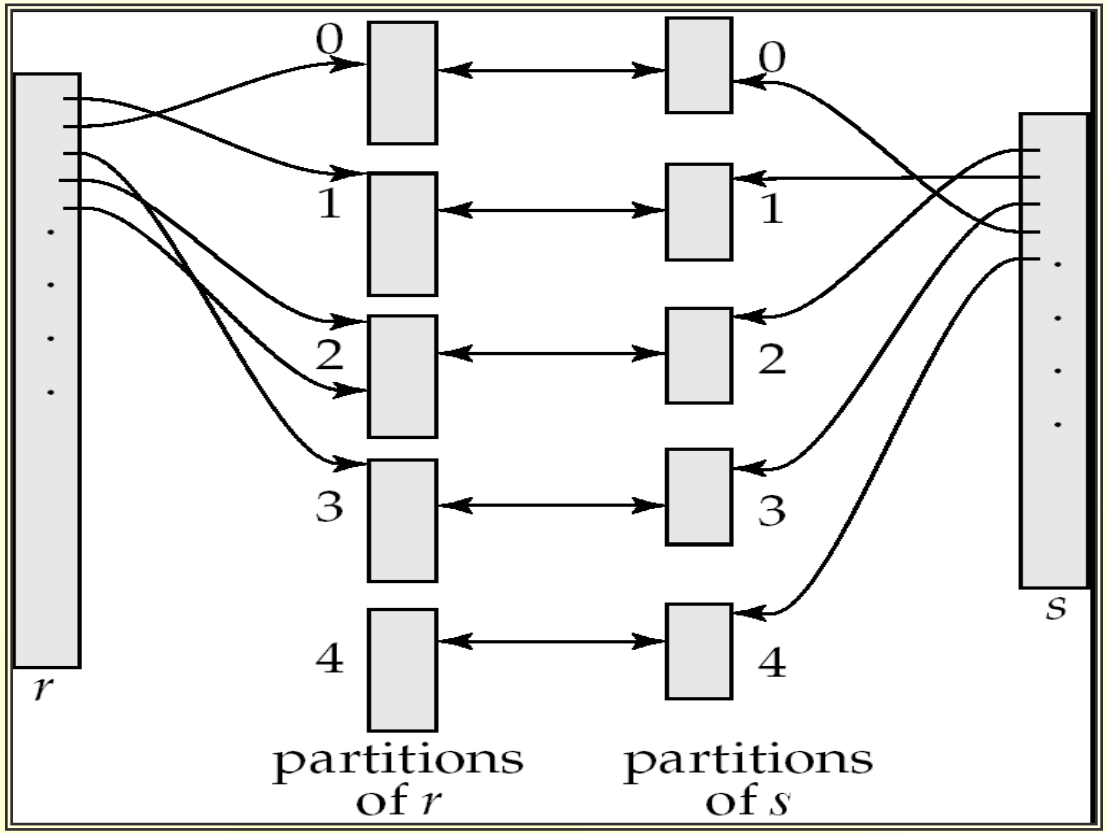


Hash-Join

- 适用适用于等值连接或自然连接
- **hash function *h*** 可以用来分割两个关系中的元组.

- ***h*** 把 ***JoinAttrs*** (连接属性)映射到{0, 1, ..., *n*},
  - $r_0, r_1, \ldots, r_n$ 表示r 元组的分割
    - **Each tuple $t_r \in r$ is put in partition r_i where $i = h(t_r[JoinAttrs])$.**
  - $r_0, r_1 \ldots, r_n$ 表示 ***s*** 元组的分割
    - **Each tuple $t_s \in s$ is put in partition s_i, where $i = h(t_s[JoinAttrs])$.**

Hash-Join (Cont.)



索引连接

- 对两个连接的关系, 它们公共的属性建立索引。
- 根据索引进行两个关系的连接。
- 循环, 直到所有元组都连接完。

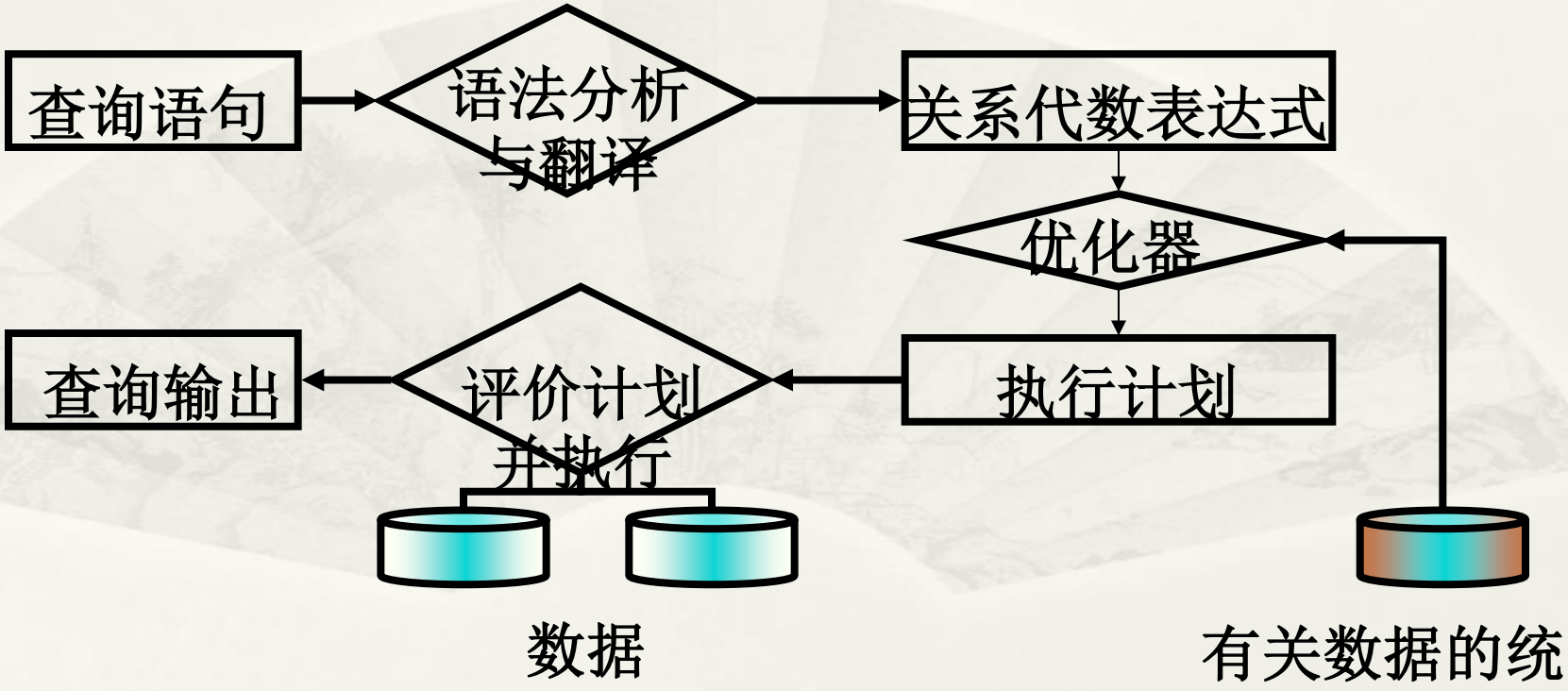
关系数据库系统的查询优化

- 目的
  - 加快查询速度
  - 制定查询计划
  - 找出最佳的优化策略
- 最优定义
  - 时间花费最小
  - 查询代价最小

查询优化的过程

- 将查询请求转换为对应的查询树
- 优化、分析查询，试图选择最佳的查询策略
  - 分析数据库实例，数据库的统计信息
    - 表、索引、内存（Cache）的状况
  - 选择Join的顺序
  - 决定最终的*存取策略*
- 编译所选择的查询计划
- 执行查询，将结果集送给请求者

查询优化的过程（图示化）



查询性能——例子

* 求选修了 ‘C02’课程的学生姓名

```
Select student.sname
from student,sc
Where      student.sno = sc.sno And
           sc.cno = ‘C02’
```

其中：　　student人数：1000

　　sc记录数：　　10000

　　选C02的人数：　　50

假设内存中的一个块能放10个student元组,100个sc元组,每秒读写20块,内存存放5个student块,一个sc块。

通过对不同执行方案的对比来看查询优化的重要性

* 第一种方案：
* $Q_1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='2'}(Student \times SC))$

需要105+2*5*10000=100000 秒~27个小时

读取块数：1000/10(学生表放入内存需要的块)+1000/(10*5) * 10000/100 (读SC表20遍,每一遍100块) =2100块 =105秒
连接后元组: 为10的七次方 每块装10个元组 需要10000000/20=50000 秒（内存计算结果放入外存）
读入连接后的元组 也需要 50000秒, 满足条件的元组假设为50个,全部可以存放在内存,则总的时间为105+2*50000=100000秒

查询性能——例子

* 第二种方案：

$\pi_{sname}(\sigma_{sc.cno='c02'}(student \bowtie sc))$

需要时间105+50+50=205秒~3分钟

读总的块还是2100块，化105秒，但自然连接后结果只有10^4个，写出这些元组需要10^4/10/20=50秒，读取中间结果也需要50秒。

查询性能——例子

* 第三种方案：

$\pi_{sname}(student \bowtie \sigma_{sc.cno='c02'} sc))$

需要5+5=10秒

先对SC表作选择，读一次SC表，存取100块花费时间为5秒，读学生表一遍，共100块，花时间5秒。结果输出。

代数优化

❖把代数表达式Q1变换为Q2、 Q3

$$Q_1=\pi_{Sname}(\sigma_{Student.Sno=SC.Sno\wedge Sc.Cno='2'}(Student\times SC))$$

$$Q_2=\pi_{Sname}(\sigma_{Sc.Cno='2'}(Student \bowtie SC))$$

$$Q_3=\pi_{Sname}(Student \bowtie \sigma_{SC.Cno='2'}(SC))$$

- 有选择和连接操作时，先做选择操作，这样参加连接的元组就可以大大减少，这是代数优化。

物理优化

- 在Q_3中SC表的选择操作算法有全表扫描或索引扫描，经过初步估算，索引扫描方法较优。
- 对于Student和SC表的连接，利用Student表上的索引，采用索引连接代价也较小，这就是物理优化。

查询代价

- 什么是好的性能
  - 代价最小的查询就是好的查询
- 查询代价
  - CPU运算的代价（0.1）
  - 网络传输的代价（1）
  - 物理I/O的代价（18）
  - 逻辑I/O的代价（2）

查询代价（续）

- 集中式数据库
  - 执行开销主要包括
    - 磁盘存取块数(I/O代价)
    - 处理机时间(CPU代价)
    - 查询的内存开销
  - I/O代价是最主要的
- 分布式数据库
  - 总代价=I/O代价+CPU代价+内存代价＋通信代价

查询性能

* 影响查询代价的因素
  * 硬件设置
  * 磁盘和文件管理
  * 数据库的设计
  * 查询的设计
  * 统计信息的正确性

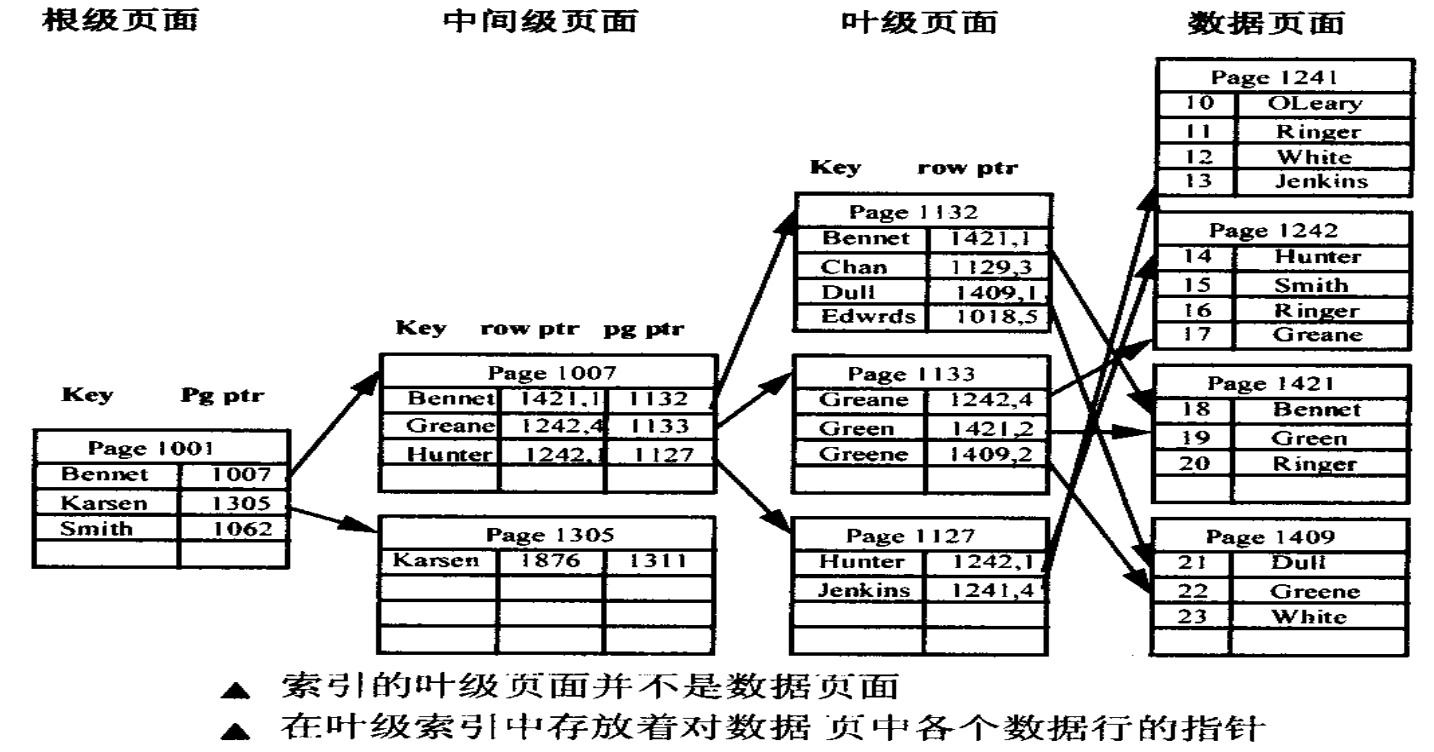
查询性能——索引

- 索引（Index）
  - 是为了加速对表中数据行的检索而创建的一种分散存储结构
  - 索引是对表建立的，由除了存放表的数据页面以外的索引页面组成
  - 就象一张对照表

查询性能——索引

* 索引的种类
  * 聚簇索引
    * 在表中的数据行根据特定的属性进行物理排序
    * 用户通过建立聚簇索引来规定数据行的物理排序
    * 一个表至多有一个聚簇索引
    * 有利于范围搜索
  * 非聚簇索引
    * 索引与数据行的存放顺序无关
    * 索引作为表的附加信息
    * 有利于单行查询，不利于范围查询

查询性能——索引的数据结构



- 索引的叶级页面并不是数据页面
- 在叶级索引中存放着对数据页中各个数据行的指针

查询性能——聚簇索引的数据结构

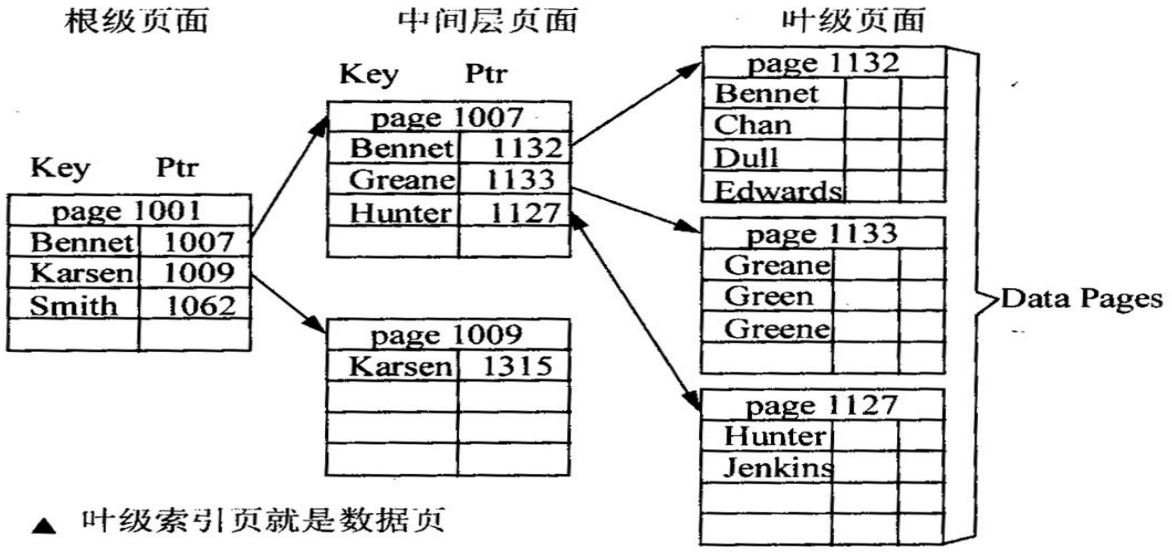


图 6-4 聚簇索引的结构

查询性能——关系键（主码）

- 索引(Index)与键(Key)
  - 索引与键之间无严格的对应关系(有些DBMS自动为PK建立相应的索引)
  - 索引是在数据库物理设计时确定的
  - 而键是数据库的逻辑特性
  - 索引可以是唯一、也可以是非唯一的

查询性能——查询的设计

- 我们的任务
  - 设计合理的查询，使DBMS中的查询优化器能“读懂”，以便找到代价最低的查询计划

查询性能——查询的设计

* 使用合适的运算符

建议	避免
sage = 26	sage != 26
sage >= 26	sage > 25
sdept Is Null	sdept Is Not Null
sdept In (‘CS’,’MA’)	sdept Not In (‘CS’,’MA’)
sname Like ‘rob%’	sname Not Like ‘%ort’

少做or(即in): Sage in (25,26,27)

查询性能——查询的设计

- 在连接属性上建立索引
  - 优化器会选择一个表(具有较少的匹配行)进行扫描
  - 对另一个表(具有较多的匹配行，索引)进行查询，找出匹配的行

查询性能——查询的设计

- 索引覆盖
  - Where子句中的属性，与索引中的属性相同且顺序相同——索引覆盖
  - 在关键查询的相关属性上建立索引
  - 如：sc_index(sno,grade desc)
  - Where sno = ‘95001’ and grade <=59
  - Where grade <= 59 and sno = ‘95001’ X

查询性能——索引策略

* 索引越多越好吗？
  * 增加查询覆盖的机会
  * 改善查询的性能
  * 减缓更新的速度

查询性能——索引策略

* 策略
  * 建立适当的、关键的索引
  * 大批数据的更新
    * 删除索引→更新数据→重建索引
  * 维护索引
    * 删除索引→重建索引（清洗索引）

查询优化具体的策略

* 代数优化
* 物理优化

代数优化 (等价规则介绍)

- 各种连接运算的交换律 $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$
- 各种连接运算的结合律
- 投影的串接 $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
- 选择的串接 $\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L_1}(E)$
- 选择与投影的交换律 $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$

代数等价（续）

- 选择与连接的交换律
- 选择与并的分配律
- 选择与差的分配律
- 选择对自然连接的分配律
- 投影与连接的分配律
- 投影与并的分配律

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$

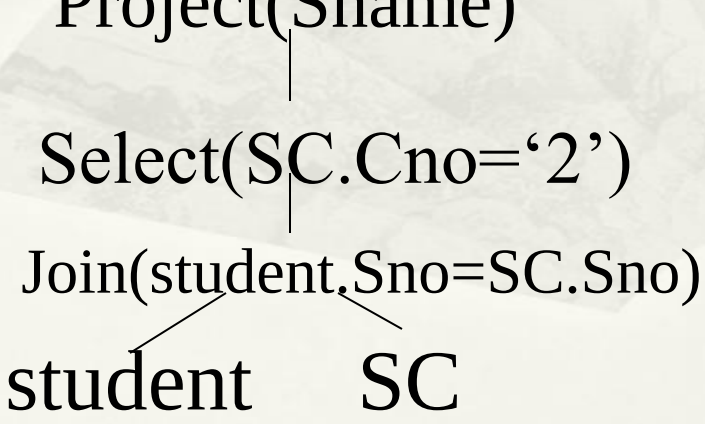
设E_1和E_2有相同的属性名

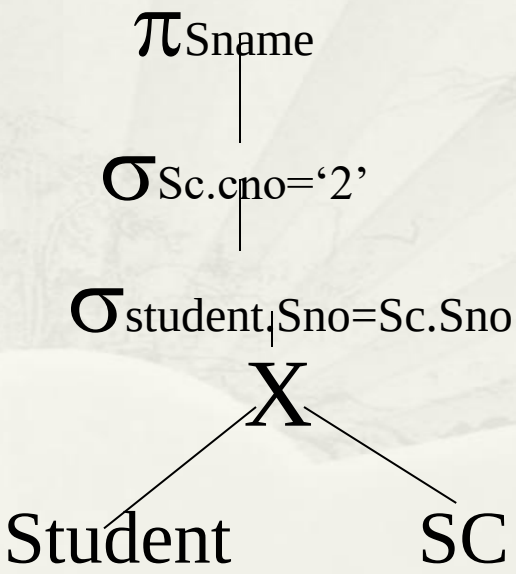
查询优化步骤（回顾）

* 把查询转换成某种内部表示（语法树）
* 把语法树转换成标准形式
* 根据代数等价原则变换
  * 选择运算尽量先做
  * 投影和选择运算同时进行
  * 投影同前或后的双目运算结合起来
  * 把某些选择同它前面的X乘积结合起来成为一个连接运算
  * 找出公共子表达式
* 选择低层的存取路径
* 生成查询计划，选择代价最小的

例如：select student.sname from student, SC where student.sno=SC.sno and SC.cno=‘2’

语法树--》 关系代数语法树→优化





优化：根据关系代数等价变化规则

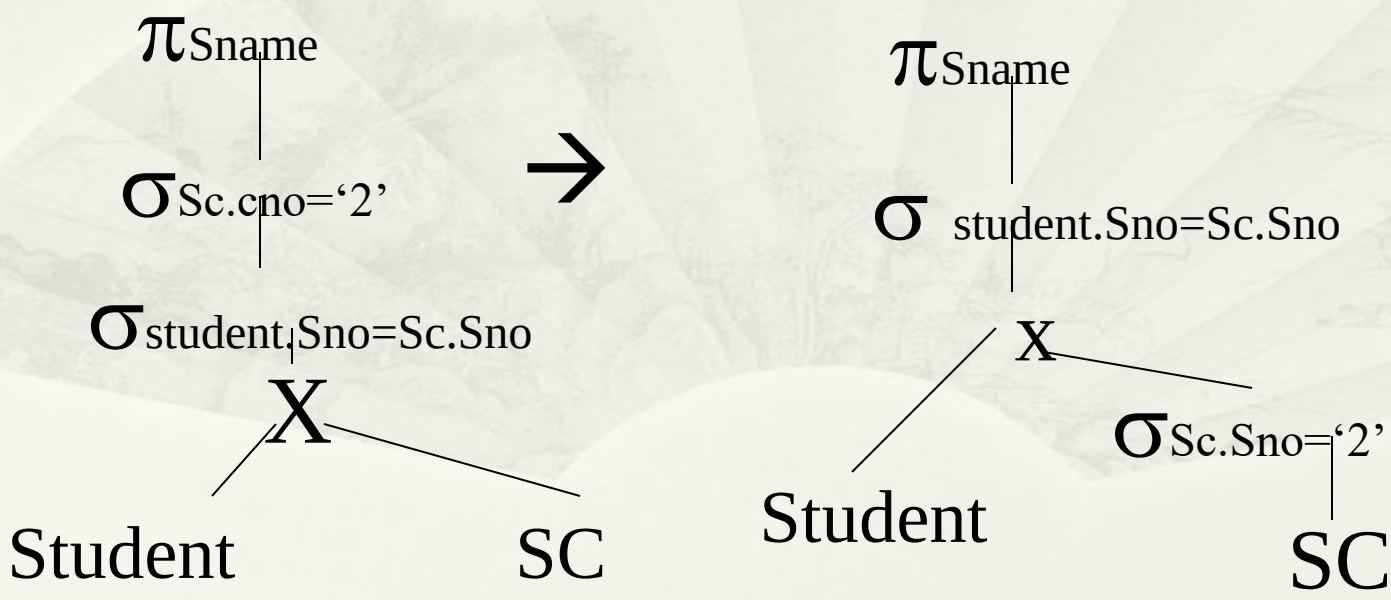
* 连接的交换律，结合律。
* 选择与投影的交换律，选择与连接的交换律，投影与连接的交换律。
* 投影与选择的串接定律

语法树--》 关系代数语法树→优化

根据交换律，结合律等价原则

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

$$\sigma_{\theta 0}(E1 \bowtie_{\theta} E2) = (\sigma_{\theta 0}(E1)) \bowtie_{\theta} E2$$



物理优化

- 代数优化改变查询语句的次序和组合，不涉及底层的存取路径
- 物理优化选择高效合理的存取路径，优化查询计划。

物理优化

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

* 基于规则的启发式优化
* 基于代价估算的优化
* 两者结合的优化方法

基于启发式规则的存取路径选择优化

* 选择操作

小关系：全表顺序扫描

大关系：选择主码索引，或选择列索引，或组合索引等

* 连接操作
  - 如已排序，选排序合并算法
  - 如连接属性上有索引，则索引连接算法
  - 如一个表较小，则使用HASH连接算法
  - 嵌套循环算法

基于代价的估算

* 统计信息
* 代价估算示例
* 优化方法

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)

统计信息（续）

（2）对基表的每个列

- 该列不同值的个数(m)
- 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)
- 可以计算选择率(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，则要计算每个值的选择率，$f=$具有该值的元组数/N

统计信息（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)

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

（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

代价估算示例（续）

（2）索引扫描算法的代价估算公式（续）

- 如果选择条件涉及非码属性
  - 若为B+树索引，选择条件是相等比较，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，所以(最坏的情况)cost=L+S

代价估算示例（续）

（2）索引扫描算法的代价估算公式（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

代价估算示例（续）

K:内存缓冲区块数
Br: R关系占用块数
Bs:S关系占用块数

（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+BrBs/(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

代价估算示例（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

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

小结

- 查询处理的流程
- 查询优化是查询处理的关键技术

- 查询处理是RDBMS的核心

课堂练习（1）

- Select cname
- From student,course,sc
- Where student.sno=sc.sno and sc.cno=course cno and student.sdept='IS';
- 画出关系代数表示的语法树,并用关系代数表示的优化算法对原始语法数进行优化处理.

课堂练习（2）

* 试述关系数据库管理系统查询优化的一般策略