# 第八章 关系查询处理和查询优化

## 查询处理步骤

RDBMS查询处理分为四个阶段：

### 1. 查询分析

对查询语句进行扫描、词法分析和语法分析 从查询语句中识别出语言符号 进行语法检查和语法分析

### 2. 查询检查

根据数据字典对合法的查询语句进行语义检查 根据数据字典中的用户权限和完整性约束定义对用户的存取权限进行检查 检查通过后把SQL查询语句转换成等价的关系代数表达式 RDBMS一般都用查询树(语法分析树)来表示扩展的关系代数表达式 把数据库对象的外部名称转换为内部表示

### 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：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 查询优化方法选择的依据： 基于规则(rule based) 基于代价(cost based) 基于语义(semantic based)

### 查询执行

依据优化器得到的执行策略生成查询计划 代码生成器(code generator)生成执行查询计划的代码

## 实现查询操作的算法示例

### 选择操作的实现

#### 1. 简单的全表扫描方法

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(或散列)扫描方法

适合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

### 连接操作的实现

#### 1. 嵌套循环方法(nested loop)

对外层循环(Student)的每一个元组(s)，检索内层循环(SC)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

#### 2. 排序-合并方法(sort-merge join 或merge join)

适合连接的诸表已经排好序的情况 排序 - 合并连接方法的步骤：

- 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，把它们连接起来
- 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

### 3.索引连接(index join)方法

① 在SC表上建立属性Sno的索引，如果原来没有该索引 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

### 4.Hash Join方法

把连接属性作为hash码，用同一个hash函数把R和S中的元组散列到同一个hash文件中

- 划分阶段(partitioning phase)：
  - 对包含较少元组的表(比如R)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(S)进行一遍处理
  - 把S的元组散列到适当的hash桶中
  - 把元组与桶中所有来自R并与之相匹配的元组连接起来

# 关系系统的查询优化

## 查询优化概述

### 查询优化的必要性

查询优化极大地影响RDBMS的性能

### 查询优化的可能性

关系数据语言的级别很高，使DBMS可以从关系表达式中分析查询语义

### 由DBMS进行查询优化的好处

- 用户不必考虑如何最好地表达查询以获得较好的效率
- 系统可以比用户程序的优化做得更好
  - 优化器可以从数据字典中获取许多统计信息，而用户程序则难以获得这些信息
  - 如果数据库的物理统计信息改变了，系统可以自动对查询重新优化以选择相适应的执行计划
  - 优化器可以考虑数百种不同的执行计划，而程序员一般只能考虑有限的几种可能性。
  - 优化器中包括了很多复杂的优化技术，这些优化技术往往只有最好的程序员才能掌握。系统的自动优化相当于使得所有人都拥有这些优化技术

### 代价模型

RDBMS通过某种代价模型计算出各种查询执行策略的执行代价，然后选取代价最小的执行方案

- 集中式数据库
    - 单用户系统 总代价 = I/O代价 + CPU代价
    - 多用户系统 总代价 = I/O代价 + CPU代价 + 内存代价
- 分布式数据库 总代价 = I/O代价 + CPU代价 + 内存代价 + 通信代价

### 查询优化目标

- 选择有效策略
- 求得给定关系表达式的值
- 使得查询代价最小(实际上是较小)

### 实际系统的查询优化步骤

1.将查询转换成某种内部表示，通常是语法树 2.根据一定的等价变换规则把语法树转换成标准(优化)形式

3. 选择低层的操作算法 对于语法树中的每一个操作

- 计算各种执行算法的执行代价
- 选择代价小的执行算法

4. 生成查询计划(查询执行方案) 查询计划是由一系列内部操作组成的。

## 关系代数等价变换规则

代数优化策略：通过对关系代数表达式的等价变换来提高查询效率 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
- 上面的优化策略大部分都涉及到代数表达式的变换

## 常用的等价变换规则

设E1、E2等是关系代数表达式，F是条件表达式 1.连接、笛卡尔积交换律 2.连接、笛卡尔积的结合律 3.投影的串接定律 4.选择的串接定律 5.选择与投影的交换律 6.选择与笛卡尔积的交换律 7.选择与并的分配律 8.选择与差运算的分配律 9.选择对自然连接的分配律 10.投影与笛卡尔积的分配律 11.投影与并的分配律

## 查询树的启发式优化

典型的启发式规则：

1. 选择运算应尽可能先做。在优化策略中这是最重要、最基本的一条
2. 把投影运算和选择运算同时进行

- 如有若干投影和选择运算，并且它们都对同一个关系操作，则可以在扫描此关系的同时完成所有的这些运算以避免重复扫描关系

3. 把投影同其前或其后的双目运算结合起来
4. 把某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算
5. 找出公共子表达式

- 如果这种重复出现的子表达式的结果不是很大的关系并且从外存中读入这个关系比计算该子表达式的时间少得多，则先计算一次公共子表达式并把结果写入中间文件是合算的
- 当查询的是视图时，定义视图的表达式就是公共子表达式的情况 遵循这些启发式规则，应用上节的等价变换公式来优化关系表达式的算法。 算法：关系表达式的优化 输入：一个关系表达式的查询树 输出：优化的查询树 方法： （1）分解选择运算 利用规则4把形如6F1 ∧F2 ∧ … ∧ Fn (E)变换为6F1 (6F2(… (6Fn(E))… )) （2）通

过交换选择运算，将其尽可能移到叶端 对每一个选择，利用规则4～9尽可能把它移到树的叶端。 （3）通过交换投影运算，将其尽可能移到叶端 对每一个投影利用规则3，5，10，11中的一般形式尽可能把它移向树的叶端。 注意： ①规则3使一些投影消失 ②规则5把一个投影分裂为两个，其中一个有可能被移向树的叶端 （4）合并串接的选择和投影，以便能同时执行或在一次扫描中完成 利用规则3～5把选择和投影的串接合并成单个选择、单个投影或一个选择后跟一个投影。 使多个选择或投影能同时执行，或在一次扫描中全部完成 尽管这种变换似乎违背"投影尽可能早做"的原则，但这样做效率更高。 （5）对内结点分组 把上述得到的语法树的内节点分组。 每一双目运算(×， ，∪，-)和它所有的直接祖先为一组(这些直接祖先是6，π运算)。 如果其后代直到叶子全是单目运算，则也将它们并入该组。但当双目运算是笛卡尔积(×)，而且其后的选择不能与它结合为等值连接时除外。把这些单目运算单独分为一组。

# 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 选择的方法：

- 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

**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。
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
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
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 则可以用［例1-C4］中介绍的索引扫描方法 否则使用全表顺序扫描
6. 对于用OR连接的析取选择条件，一般使用全表顺序扫描

**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
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 Bs/(K-1) 显然应该选块数小的表作为外表

## 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

**一、统计信息**

基于代价的优化方法要计算各种操作算法的执行代价，与数据库的状态密切相关 数据字典中存储了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)
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 该列最大值 该列最小值 该列上是否已经建立了索引 索引类型(B+树索引、Hash索引、聚集索引)
3. 对索引(如B+树索引) 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)

**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
2. 索引扫描算法的代价估算公式

- 如果选择条件是码 = 值 如［例1-C2］，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
- 如果选择条件涉及非码属性 如［例1-C3］，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) 最坏的情况下，满足条件的元组可能会保存在不同的块上，此时，cost=L+S
- 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

3. 嵌套循环连接算法的代价估算公式 前面已经讨论过了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， $cost = Br + BrBs/(K-1) + (Frs * Nr * Ns)/Mrs$ 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
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 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

- 查询处理是RDBMS的核心，查询优化技术是查询处理的关键技术
- 关系系统的查询优化
  - 代数优化：关系代数表达式的优化
    - 关系代数等价变换规则
    - 关系代数表达式的优化算法
  - 物理优化：存取路径和低层操作算法的选择
- 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