数据库技术-查询处理及优化

赵亚伟

zhaoyw@ucas.ac.cn

中国科学院大学 大数据分析技术实验室

2017.12.10

目录

- 查询处理的基本步骤
- 算法代价估算
- 表达式运算结果估算
- 表达式计算
- 表达式等价转换
- 启发式优化方法

问题提出

- 数据库的主要功能是存储和管理好数据供用户查询使用，查询操作最多、最常用。
- 关系数据库的查询操作一般用查询语言（例如SQL语言）描述，查询语句重点表达查找条件和结果，而把查找的实施过程以及查找策略的选择留给**DBMS**负责。
- **DBMS**如何实现查询操作?

查询处理

- **DBMS**接收到高级语言书写的查询后，首先做词法和语法分析，确认正确后，产生查询内部表示，然后由**DBMS**制定一个执行策略，包括如何访问文件数据，如何存储中间结果，直至得到一个正确的查询结果。
- 查询处理：是指从数据库中提取数据所涉及的一系列活动。
- 查询优化：查询处理的一个重要环节，一个查询有多种执行策略，从中选择一个适合策略的过程称之为查询优化。

查询优化的分类

- 查询优化大致分两类
  - 代价优化：针对多个候选策略逐个进行代价估算，从中选取代价低的作为执行策略，基于代价的优化很昂贵，对大型据库是值得计算的。
  - 规则优化：仅涉及查询语句本身，根据一些启发式规则，如“先选择、投影，后连接”等即可完成优化，有时也称为代数优化、启发式优化。根据经验，不保证成功，但效率高。
- 这些优化形式不是截然分开的，对一个查询优化的执行计划是多种优化方式的综合结果
- 注意：商业化数据库产品采用综合方式进行优化。

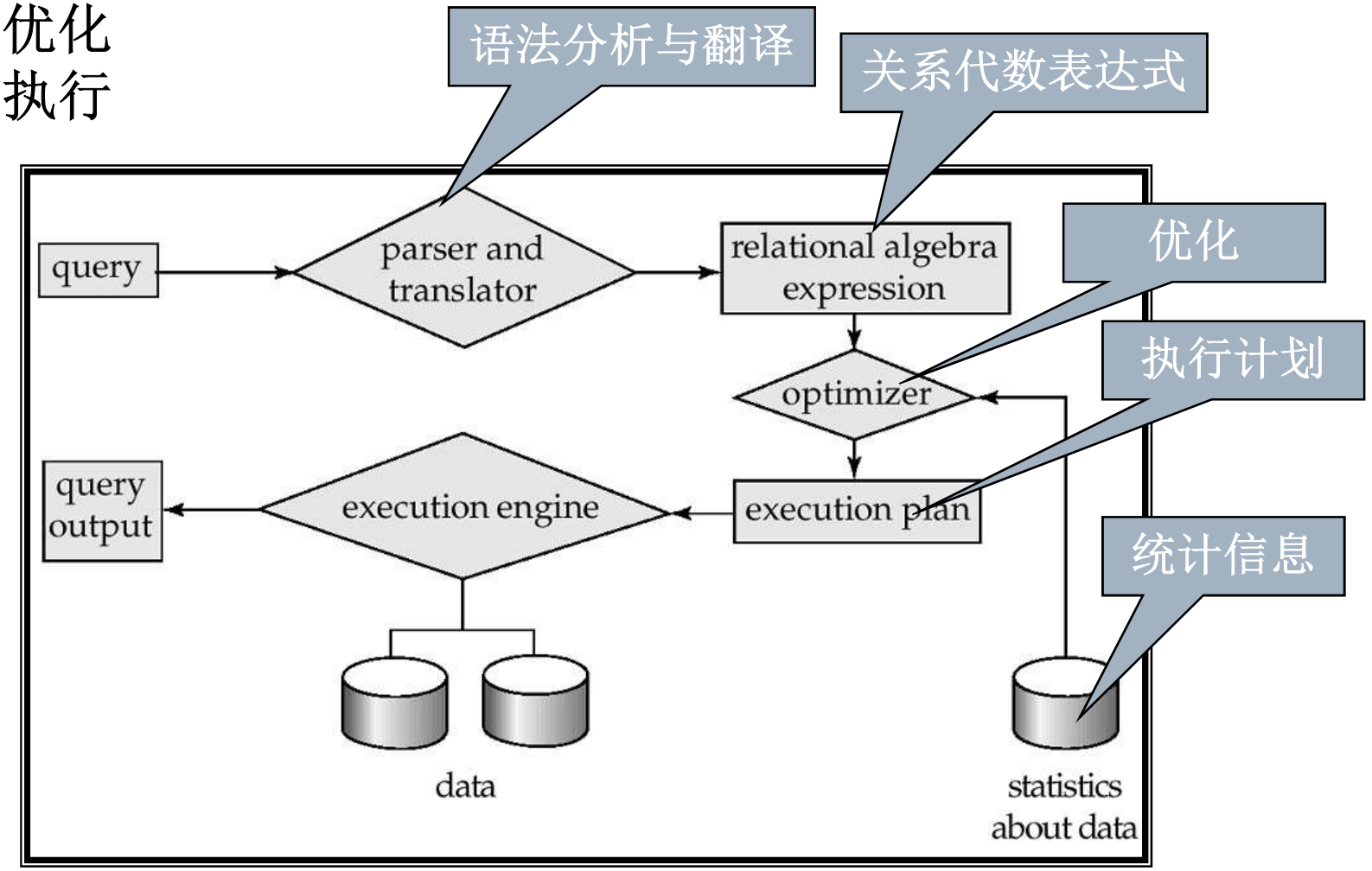
- 输入：SQL语句
- 输出：满足查询条件的数据
- 中间过程：查询处理及优化

查询处理及优化过程：

(1) 语法分析与翻译

(2) 优化

(3) 执行



各步骤功能

- 语法分析与翻译
  - 将查询转换为内部格式-语法分析器-语法检查
  - 转化为等价的多个关系代数表达式
  - 语法检查，关系名验证

  翻译过程类似编译器的语法分析器
- 优化器：根据统计信息计算各方案的代价，选择一个代价小的执行计划
- 执行引擎
  - 执行一个查询执行计划，返回一个执行结果

例子

- 给定一个查询会有几种不同的计算方法:

  select balance
  from account
  where balance<2500

  可翻译成下面两个关系表达式中的任何一个:

  $\sigma_{balance<2500}(\Pi_{balance}(account))$

  $\Pi_{balance}(\sigma_{balance<2500}(account))$

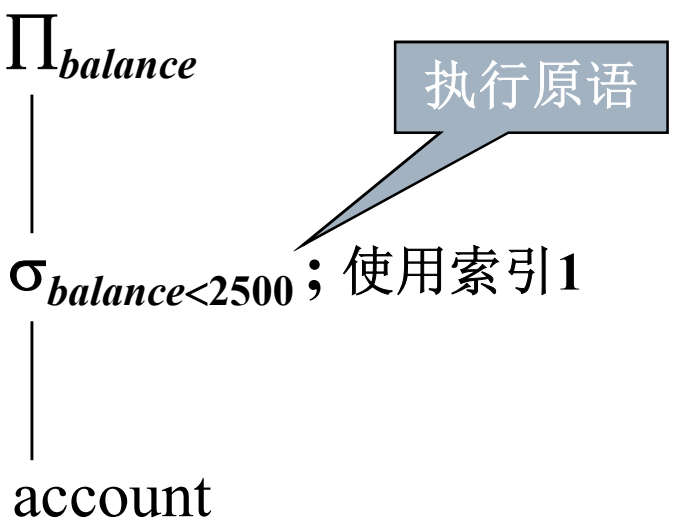
  可以用不同的算法来执行每个关系代数运算

查询执行计划的一种描述

- 如何构造具有最小查询执行代价的查询执行计划？
- 许多数据库不用关系代数表示查询，而是采用语法树（SQL查询结构 + 注释）表示。
- 由于依赖很多参数，查询优化器对操作代价精确计算是困难的，但对每个操作的执行代价的粗略估计是可能的。

完整的查询计划

- 一个完整的查询计划：
  - 关系代数表达式，分解为原子运算，便于代价估算
  - 执行原语：带注释的关系代数运算。表达式的注释说明了如何执行每个操作，注释可以是某个具体操作采用的算法，也可以是要使用的方法(如索引，排序等)
  - 查询执行计划：用于执行一个查询的原语操作序列



一个查询执行计划

下面先重点讨论基于代价的优化技术

查询代价的测量

- 执行一次查询的开销包括如下成份：
  - 访问外存的开销（简称I/O开销）：主要指寻找、读、写外存数据块的代价。数据库文件 + 临时文件（体积太大，内存放不下）。
  - 存储开销：主要指存储中间文件的开销。
  - 计算开销：主要指在内存(缓冲区)执行各类操作，如定位、排序、归并记录和计算属性值的开销。
  - 通信开销：主要指分布式数据库的数据在各结点之间传输的开销。

- 大型数据库系统中，磁盘存取通常是最主要的代价。
- 更为准确的度量应估算下列因素：
  - 执行寻道操作的数据-----（搜索时间）
  - 读取的磁盘块数量-----（读块时间）
  - 写入磁盘块的数量-----（写块时间）

  将这些数量分别与平均寻道时间、读磁盘块平均传输时间、写磁盘块的平均传输时间相乘并将结果累加。实际的查询优化器在计算操作代价时还要将**CPU**的代价考虑进去。

  另外：注意，写盘比读盘费时间

代价估算简化

- 简化模型：(通常忽略次要因素，抓主要矛盾）
  - 传输块数 ***number of block transfers***
  - 忽略顺序和随机**I/O** 的差别，忽略**CPU**的花销
  - 常用最坏情形估计 **worst case estimates**
- 类似研究自由落体运动，忽略空气阻力建立模型
- 注意：在工程中的实际系统不忽略 **CPU** 花销，要考虑随机和顺序的影响。

查询处理运算

- 常用的查询处理运算
  - 选择运算
  - 连接运算
  - 投影运算
  - 集合运算
  - 排序
  - 聚集运算
  - …

查询处理运算的相关问题

- 查询处理运算采取了哪些算法？
  - 如选择运算采用：基本扫描算法（线性搜索、二分搜索）、等值索引扫描算法、比较索引扫描算法、…
  - 如连接运算采用：嵌套循环连接、块嵌套循环连接、索引嵌套循环连接、…
  - 如投影运算采用：重复消除、属性选择
  - …
- 选用哪种算法，则需要对算法的性能进行度量，算法的代价是最重要的指标，是基于代价查询优化技术的基础。

算法的代价如何计算？

教材上介绍了多种算法的代价估算方法，掌握主要思想即可

目录

- 查询处理的基本步骤
- 算法代价估算
- 表达式运算结果估算
- 表达式计算
- 表达式等价转换
- 启发式优化方法

代价计算的两个问题

- 代价计算是精确计算还是估算？
  - 算法代价计算是基于关系的统计信息以及一些基本原则，因此精确计算是不现实的，只能进行估算
- 由程序员做还是系统做？
  - 优化器有理由比程序员做得更好：
    - 优化器具有丰富的可使用的信息
    - 当数据库发生变化时优化器容易再次进行优化
    - 优化器能够对多种实现策略逐一进行考虑
    - 优化器集中了最优秀的程序员的智慧和经验

算法代价如何估算？

算法代价估算

- 算法代价估算基于以下两个基本条件
  - 利用关系的统计信息，这些信息是已知条件，是前提，由系统提供，元数据
  - 一些原则，如磁盘存取比内存存取慢，所以磁盘存取是代价的决定因素
- 一般以访问的磁盘块数量作为衡量算法代价大小的指标。

对象：原子运算（如，$\sigma_{balance<2500}$）

输入：统计信息（如，n_r, b_r …）

输出：代价（块）

用于代价估算的统计信息1

- DBMS目录存储了关系的统计信息有：
  - n_r: 关系r的元组数
  - b_r: 包含关系r中元组的块数
  - l_r: 关系r每个元组的字节数
  - f_r: 关系r的块因子，即一个块容纳的关系r中元组的个数
  - 假设关系r的元组物理上完全存储在一个文件中，则

    $b_r = [n_r / f_r]$

用于代价估算的统计信息2

- **Dist(A, r) (或V(A, *r*)):** 关系r中属性A中出现的非重复值个数（distinct），该值与$\Pi_A(r)$的大小相同，如果A是关系r的码，则V(A, *r*)等于n_r

- **Max(A, r):** 关系r中属性A所具有的最大值。
- **Min(A, r):** 关系r中属性A所具有的最小值。

用于代价估算的统计信息3

- **SC(A,r)**：关系**r**的属性**A**的选择基数。表示关系**r**中满足属性**A**上的一个等值条件的平均元组数。
  - 若**A**为**r**的码属性，则**SC(A,r) ＝1**
  - 若**A**为非码属性，并假定**Dist(A,r)**个不同的值在元组上均匀分布，则**SC(A,r) ＝ (n_r / Dist(A,r))**，即每个属性值的平均记录数。
  - 说明：**Dist(A,r)**与**SC(A,r)**中的**A**可以是属性组。
- **$SF_{A\ op\ a}(R)$**: 关系**R**上的谓词“**A op a**”选择率，表示关系**R**上满足谓词“**A op a**”的元组所占百分比
  - **$0 \le SF_{A\ op\ a}(R) \le 1$**

用于代价估算的统计信息4

- 有关索引的统计信息：
  - f_i：树形结构(如B^+树)索引i的内部结点的平均扇出（即n，见索引部分的课件）。
  - HT_i：索引i的层数(深度)。（助记HeighT）

  对于关系r的属性A上所建的平衡树索引（如B^+树），$HT_i = [\log_{f_i}(Dist(A,r))]$

  对于散列索引，$HT_i = 1$
  - LB_i：对于顺序索引，索引i中最低层索引块数目，即索引叶层的块数。（助记Leaf Block）

  对于散列索引，LB_i就是索引中的块数。

- 算法A的代价估计记为E_A。
- 依据统计信息进行代价估算是不精确的
  - 统计信息的局限性：统计信息是简化的，泛化的，根据这样的信息精确估算一个运算的执行代价是不可能的
  - 统计信息本身不精确：许多系统并不是每次修改都更新统计信息，而是在系统处于轻负荷状态（闲时）时进行更新。
  - 仅考虑用读（仅考虑查询）磁盘块总数来衡量操作的代价，忽略其他因素，这是一种假定，一种简化。
- 因此，代价是估算的而不是精算。

选择运算-基本扫描算法(无索引)

- 文件扫描
  - 是用于定位、检索满足选择条件的记录的搜索算法，是存取数据最低级的操作。
- 以下算法仅说明基本思想。
- 基本扫描算法(无索引或不采用索引)
  - **A1**（线性搜索，无序情况）：对每个文件块扫描，且对所有记录进行测试，检验是否满足条件。对码属性的选择操作，系统找到所需要的记录后停止，不必搜索其他记录。
  - 线性搜索的最坏情况下的代价是$E_{A1}=b_r$(磁盘块)，平均代价$E_{A1}= (b_r /2)$（平均读块数）
  - 缺点是搜索速度慢，优点是适用面广

- **A2**（二分搜索，有序情况）：若文件按某一属性A排序，且选择条件是该属性上的等值比较，则可用二分搜索来定位符合选择条件的记录。
  - 代价：$E_{A2} = [\log_2 b_r] + [SC(A,r)/f_r] - 1$

    $SC(A,r) = n_r \times SF_{A=a}(R) = n_r / Dist(A,r)$
  - 说明：$[\log_2 b_r]$定位满足条件属性值过程需要访问的块，$[SC(A,r)/f_r]$提取数据（重复连续）所访问的块，减1是因为计算$[SC(A,r)/f_r]$重复了定位过程已经确定的一个块，所以要减去这个块。
- 缺点：需要有序文件
- 优点：效率高

例子：代价估算(A1,A2)

- 假设有account表的如下统计信息:
  - $f_{account}$ =20（块因子，一个块容20条记录）
  - Dist (branch-name, account) =50
  - Dist (balance, account) =500　//例子中没有用到
  - $n_{account}$=10000
  - $b_{account}=n_{account}/f_{account}$ =500（表占用的块数）

- 考虑查询: $\sigma_{branch\text{-}name\ =\text{‘Perryridge’}}$(account)

- 若文件无序

  对account扫描I/O块: $E_{A1}=b=500$

- 若文件有序

  SC(branch-name)= n/Dist (branch-name)

  = 10000/50=200 //每个支行平均有200个记录

  $E_{A2}=[\log_2 b_r] + [SC(A,r)/f_r] -1$

  $= [\log_2 500] + [200/20] -1=9+10-1=18$

- 上述基本运算没有使用索引，如果使用索引会如何？

索引扫描

- 使用索引的搜索算法称为索引扫描。
- 条件：表在选择条件的属性上建有索引。
- 方法：访问索引，根据索引项的指示去访问数据元组。
  - 无序索引：访问满足等值条件的元组（如，**Hash**索引）
  - 有序索引：访问满足范围查找条件的一系列元组（如，**B**$^+$索引）
- 代价包括两部分：索引扫描代价＋数据扫描代价

选择运算–等值索引扫描算法

- 等值索引扫描算法代价估算：
  - **A3**（主索引，码属性等值比较（一个值））：若使用**B$^+$**树，操作代价等于该树的高度加上取该记录的一次**I/O**操作（解释：主码或侯码为搜索码且有序，建索引，形成主索引，检索值唯一）。
  - **A4**（主索引，非码属性等值比较（可能多个值））：若使用**B$^+$**树，操作代价等于该树的高度加上取搜索码值的磁盘块数**I/O**操作。（解释：搜索码有序但非侯码，可多值）
  - **A3, A4**的代价相同：$E_{A3,4} = HT_i + [SC(A, r)/f_r]$（**SC(A, r)/$f_r$**：单值平均元组数除以块元组数，隐喻：连续的，主索引，结果为块数）

- **A5(辅助索引，等值比较)**：若索引字段是码属性，使用辅助索引可检索到等值条件的唯一一条记录；若使用B^+树，操作代价等于树的高度加上取该记录的一次I/O操作。

  代价：$E_{A5} = E_{A4} = HT_i + [SC(A, r)/f_r]$ （实质上+1）

  若索引字段是非码属性，则可检索到多条记录。若使用B^+树，操作代价等于树的高度加上取该多条记录的I/O操作，若记录分散在不同磁盘块，代价会更高。

  代价：$E_{A5} = HT_i + SC(A, r)$

  （取平均元组数，并假设每个元组分布在不同的块中，一个元组一块，平均最坏情况）

选择运算-比较索引扫描算法

- 比较索引扫描算法代价估算（等值也是比较的一种，这里约定比较为非等值）
  - **A6(主索引，比较)**：形如$A>v$或$A\geq v$的比较条件，若是主索引，找到满足条件的第一条记录后，对$A>v$，下一条记录后到尾记录，对$A \geq v$，则从该记录到尾记录。

    如果利用索引并假设有一半的元组满足条件，则代价估算：

    $E_{A6}=HT_i+b_r / 2$

  - 对于$A<v$或$A\leq v$，则不需要使用索引，只要从第一条记录搜索到不满足条件的记录为止。

    假设有一半的元组满足条件，则代价： $E_{A6}=b_r / 2$

- **A7(辅助索引，比较)**：可以使用有序辅助索引指导涉及>、≥、<、≤，对于< 和≤从最小值开始直到*v*为止；对于>和≥则相反。

- 辅助索引代价包括索引记录、指针及文件记录的**I/O**操作，由于文件记录的分散存储可能需要读取更多的磁盘块，因此，可能效率不如线性搜索。

代价估算（平均）：$E_{A7} = HT_i + LB_i/2 + n_r/2$

代价估算（最坏）：$E_{A7} = HT_i + LB_i + n_r$

树+叶+记录

$LB_i/2$：平均索引叶层的块数（最好情况1块，最坏访问所有叶节点，LB_i）

$n_r/2$：元组分散，平均块数（最好情况访问一个元组一个块，最坏情况访问所有n_r元组并访问n_r块）

例子：等值索引扫描

- 考虑查询：$\sigma_{\text{branch-name} = \text{‘Perryridge’}}$(account)使用branch-name上的索引，主索引，B^+索引，树高$HT_i=2$，元组数n=10000，branch-name唯一值数量Dist (branch-name)=50，$f_r=20$
- 计算：

  SC(branch-name)= n/ Dist (branch-name)= 10000/50=200

  $E_{A4}= HT_i + [SC(A,r)/f_r]= 2 + 200/20=12$

  查询代价为12块

例子：比较索引扫描

- 考虑查询: $\sigma_{balance<1200}$ (account)，使用balance上的索引，辅助索引，假定一半满足条件
  - 假定balance的B^+树索引（辅助索引）节点可存放20个指针，500个不同的balance值，且每个节点至少半满，故该树有25~50个叶节点（500个balance值存储在叶节点，500/20~500/10），即LB_i=50（取大值，最坏情况）
  - balance上的索引深度为3, 即HT_i=3
- 利用索引扫描：
  - $E_{A7} = HT_i + LB_i /2 + n_r /2 = 3+50/2+10000/2=5028$
- 若全表扫描
  - 对account扫描I/O块：E_{A1}=b=500
- 所以，使用索引并不一定能取得高效

选择运算-复杂索引扫描算法

- 复杂选择是指合取、析取和取反作为条件的选择运算。
  - 合取：$\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n}(r)$，必须所有θ都满足
  - 析取：$\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$，单个简单条件θ的并
  - 取反：$\sigma_{\neg \theta}(r)$，所有θ为假
  - θ表示一个选择条件

- **A8(利用一个索引的合取选择)：选索引**
  - 首先判断是否存在某一个简单条件中的某个属性上的一条存取路径（索引或有序）。若存在，则可以用选择算法**A2**到**A7**的一个来检索满足该条件的记录。然后在内存缓冲区中，通过测试每条检索到的记录是否满足其余的简单条件来最终完成操作。代价由所选择的算法决定。
  - 如查询选择一个条件为：**where branch-name="Perryridge" and balance=1200**，且**branch-name**有索引，则按该索引读记录至内存，在内存中判断每个记录是否满足**balance=1200**，故代价取决于基于**branch-name**索引查询算法。
  - 如果有多个索引，选代价小的

- **A9(使用组合索引的合取选择)：当做一个索引**
  - 如果存在合适的复合索引(在多个属性上建立一个索引)供其使用。如果选择声明的是一个或多个属性上的等值条件，并且在这些属性字段的组合上又存在复合索引，则可以直接基于索引查询。索引类型将决定使用**A3**、**A4**、**A5**算法中的一个。（将组合索引作为一个属性上的索引处理）
- **A10(通过记录标识的交实现合取选择)：读索引**
  - 如果各条件所涉及的字段上有带记录指针的索引，则对每个索引进行扫描，获取指向满足单个条件的记录指针。所有检索到的指针交集就是满足合取条件的指针集合，然后利用指针集合检索文件记录。
  - 代价：满足条件的每个属性索引查询算法代价 + 指针交集读记录代价

- **A11(通过记录标识符的并进行析取)：**
  - 如果析取选择中所有条件均有相应的存取路径存在，则逐个扫描索引获取指向满足单个条件的元组指针
  - 检索到的所有指针的并集就是指向满足析取条件的全体元组指针的集合
  - 利用这些指针检索文件记录
  - 代价：满足条件的每个属性索引查询算法代价 + 指针并集读记录代价
- 即使只有一个条件不存在存取路径，也需要对整个关系进行线性扫描找出满足条件的元组，所以析取最有效的存取方法是线性扫描，扫描的同时对每个元组进行析取条件测试

例子：合取情况的索引选择

- 考虑查询:

  select account-number

  from account

  where branch-name=“Perryridge” and balance=1200

- 如果在两个条件属性上均有索引，选择哪一个索引？假定有关**account**的统计信息同前

  $f_{account}$ =20

  Dist (branch-name, account) =50

  Dist(balance, account) =500

  $n_{account}$=10000

  $b_{account}$=$n_{account}$ /$f_{account}$ =500

- 若branch-name为主索引，balance为辅索引，则
  - 使用branch-name索引: E_{A4}=12（P39计算结果）
  - 使用balance索引: $E_{A5}= HT_i + SC(A,r)=HT_i + n/Dist(A,r) = 2+10000/500=22$
  - 故选branch-name索引
- 若branch-name, balance均为辅助索引(非聚集)
  - 由于，使用branch-name索引：E_{A5}=2+10000/50=502
  - 故选balance索引

排序

- 为什么讨论排序代价？原因：
  - **SQL**查询有时会要求对结果进行排序，**order by**
  - 连接运算中，当输入的关系已经排序时能够得到高效实现
- 根据能否完全放入内存，排序分为两种情况：
  - 内排序：内存中能够完全容纳整个关系，可采用内存排序算法，如**quicksort**等
  - 外排序：不能完全放入内存的关系，常用外部排序归并算法，重点讨论这一种
- 前面介绍的都是一趟算法，外排序为两趟算法。

外部排序归并算法

- 基本思想：设三个班已经分别排序，现全年级排序，在各班第一中选第一名，移动到年级第一名位置，以此类推，如果要排全校序，再归并。
- 两阶段排序（令M表示内存中页面数(块数)）：
  - 第一阶段：段内排序（班内排序），建立多个排序好的归并段
  - 第二阶段：归并段归并（年级、全校排序），若归并段总数N小于M，每段分配一个内存页，剩余的空间还应能够存放结果的一页。
- 对N个归并段进行归并，故称N路归并

例子：N路归并

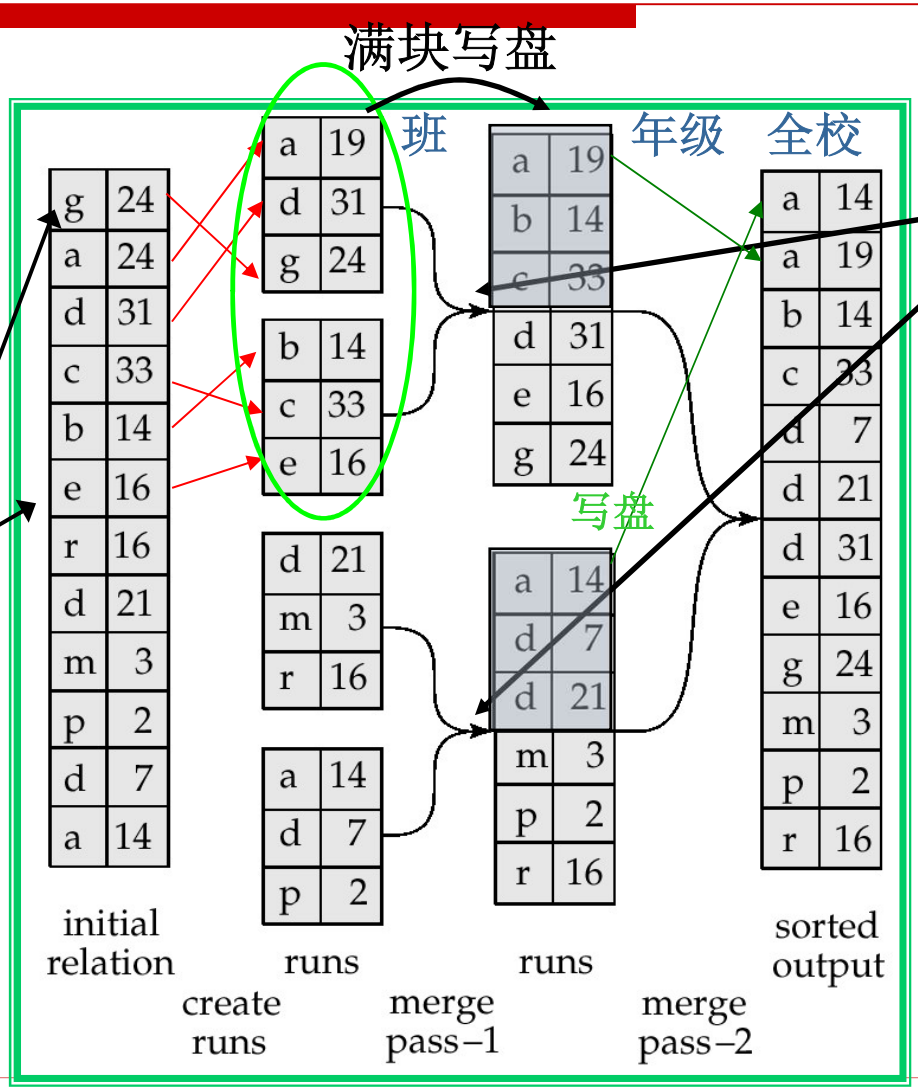
设内存Buf装3页面，2页存记录1页存结果

每页 3 记录

第一轮，读2页

第二轮，读2页

小块在内存中排序，比较，写盘



归并
比较各块头，移动最小的到合并结果，满块写盘，循环

注意
中间排序结果页未表示。

外排序代价估算

- **Cost analysis**：代价分析
  - 归并总轮数：$\lceil \log_{M-1}(b_r/M) \rceil$，不包括第1轮读
  - 磁盘IO（首次读1轮，b_r，以后各轮读写各1，$2b_r$）
  - 总代价（用磁盘IO评估）：$b_r(2\lceil \log_{M-1}(b_r/M) \rceil + 1)$
- 上述例子的代价计算：
  - $b_r(2\lceil \log_{M-1}(b_r/M) \rceil + 1) = 4\times(2\log_2(4/3)+1) =$ $4\times(2\times1+1)=12$ 次块传输。（简单算：$3\ b_r=12$）
  - 注意，$\log_2(4/3)$向上取整为1。
  - 说明：教材上的例子与图没有对应，例子假设$b_r=12$，每个元组占一块，图示则3个元组占1块。所以，教材上的例子的代价为60块，图例代价为12块。

连接运算

- 连接运算的算法很多，包括嵌套循环连接、块嵌套循环连接、索引嵌套循环连接、归并连接、散列连接等。
- 以等值连接为例加以说明，等值连接的表示形式为$r \bowtie_{r.A=s.B} s$，A、B分别表示关系*r*与*s*的属性或属性集

连接运算-嵌套循环连接

- 以上述等值连接为例来说明，A、B分别表示关系*r*与*s*的属性或属性集。
- 嵌套循环连接：遍历法。

for each 元组 t_r **in r do begin**　// t_r表示r的元组
  for each 元组 t_s **in s do begin**
    测试元组对(t_r , t_s)是否满足连接条件θ
    如果满足，把(t_r·t_s)加到结果中 // (t_r·t_s)表示二者拼接的新元组
  end
end

- 处于循环内层的关系称内层关系(如关系s), 处于循环外层的关系称为外层关系(如关系r)

嵌套循环连接代价分析

- 算法检查的元组数目是$n_r \times n_s$
- 最坏情况是缓冲区只能容纳每个关系的一个块，这时共需要$n_r \times b_s + b_r$次块存取，b_s 和b_r 分别表示含有关系r 和s中元组的块数。
- 最好情况是缓冲区能容纳两个关系，每一块只需要读取一次，共需要$b_s + b_r$次块存取
- 根据算法可知，如果一个关系能一次读入内存，那么最好将该关系作为内层关系处理，这时的代价与上述最好情况相同，即共需要b_s+b_r次块存取，可见，内循环的优化收益更大

外层循环一次要读内层循环的b_s块，n_r次循环要读$n_r \times b_s$次，外层读了b_r块

$\therefore n_r \times b_s + b_r$

例子：嵌套循环连接代价估算

- 考虑*depositor* ⋈*customer*，其中
  - $n_{customer}$=10000, customer的记录数
  - $b_{customer}$=400, customer的磁盘块数
  - $n_{depositor}$=5000, depositor的记录数
  - $b_{depositor}$=100, depositor的磁盘块数
- *customer*做外层关系，*depositor*做内层关系，则连接代价为：

  $n_{customer} \times b_{depositor} + b_{customer} = 10000 \times 100 + 400 = 1000400$
- *depositor*做外层关系， *customer*做内层关系，则连接代价为：

  $n_{depositor} \times b_{customer} + b_{depositor} = 5000 \times 400 + 100 = 2000100$

连接运算-块嵌套循环连接

- 缓冲区太小而不能同时容纳两个关系时，如果在块的基础上而不是在元组的基础上处理关系，可以节省块存取次数
- 块嵌套循环连接：是循环嵌套的一个变种，内层关系的每一个块与外层关系的每一个块形成一对。遍历法

```
for each 块Br of r do begin
  for each 块Bs of s do begin
    for each 元组t_r in Br do begin
      for each 元组t_s in Bs do begin
          测试元组对(t_r , t_s)是否满足连接条件θ
          如果满足，把(t_r·t_s)加到结果中
      end
    end
  end
end
```

块嵌套循环连接代价分析

- 最坏情况：对于外层关系中每一块，内层关系s的每一块只需读一次，而不是对外层关系的每一个元组读一次。因此，最坏情况下共需$b_r \times b_s + b_r$次访问。使用较小的关系作为外层关系更有效。
- 最好情况：缓冲区能容纳两个关系，每一块只需要读取一次，共需要$b_s + b_r$次块存取，同前

外层循环一次要读内层循环的b_s块，b_r次循环要读$b_r \times b_s$次，外层读了b_r块

$\therefore b_r \times b_s + b_r$

例子：块嵌套循环连接代价估算

- 考虑*depositor* ⋈ *customer*，其中
  - $n_{customer}$=10000, customer的记录数
  - $b_{customer}$=400, customer的磁盘块数
  - $n_{depositor}$=5000, depositor的记录数
  - $b_{depositor}$=100, depositor的磁盘块数
- 嵌套循环连接：*customer*做外层关系，*depositor*做内层关系，则连接代价为：
  - $n_{depositor} \times b_{customer} + b_{depositor} = 5000 \times 400 + 100 = 2000100$
- 块嵌套循环连接：*depositor*做外层关系，*customer*做内层关系，则连接代价为：
  - $b_{depositor} \times b_{customer} + b_{depositor} = 100 \times 400 + 100 = 40100$

连接运算-索引嵌套循环连接

- 索引嵌套循环连接：在嵌套循环连接中，若内层循环的连接属性上有索引，则可以用索引查找代替文件查找。对于外层关系**r**的每一个元组t_r，可以利用索引查找**s**中和元组t_r满足连接条件的元组。只读内层索引，不读数据文件。
- 可以使用已有索引或为了计算该连接专门建立临时索引的情况下使用。

索引嵌套循环连接代价分析

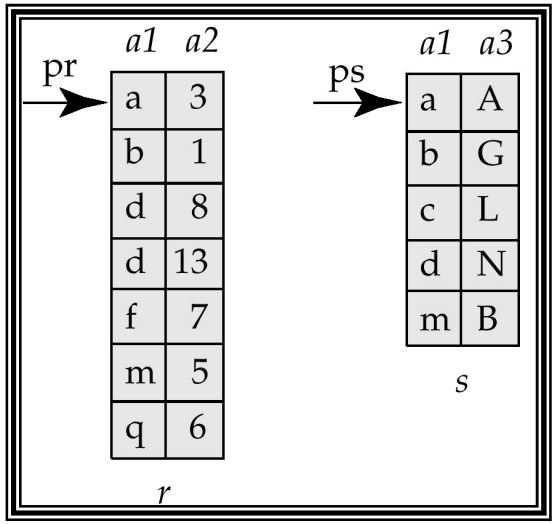
- 代价分析：对于外层关系r的每一个元组，需要在关系s的索引上进行查找，检索相关元组。
- 最坏情况，如果缓冲区只能容纳关系r的一页和索引的一页。读取关系r需b_r次磁盘存取，b_r指含有关系r中记录的磁盘块数，对于关系r中的每个元组，在s的索引上进行查找。连接代价为$b_r + n_r \times c$，$n_r \times c$可参考A3、A4、A5（索引等值比较）进行估算。（其中n_r是关系r中的记录数，c：对外层一个元组，查内存索引及取元组的代价，可估计为用连接条件在选择的代价）
- 如两个关系在连接字段上有索引，用小的一个作外层，因为此时内层有索引，不需要全扫描，而外层要全扫描，小一些好

例子：索引嵌套循环连接代价

- 求*depositor* ⋈ *customer*
- *customer*在连接字段*customer-name*上建**B$^+$tree** 索引，每索引结点含**20**索引项，**10,000**条记录，树高为**4**，找主记录再读一块，**c=5**
- *depositor*有**5000**记录（其他参数见**P57**）
- 用块嵌套循环连接方法的代价
  - **400*100 + 100 = 40,100 disk accesses** 最坏情况
- 用索引嵌套循环连接方法的代价
  - **100 + 5000 * 5 = 25,100 disk accesses.** 少了**37%**

连接运算-归并连接

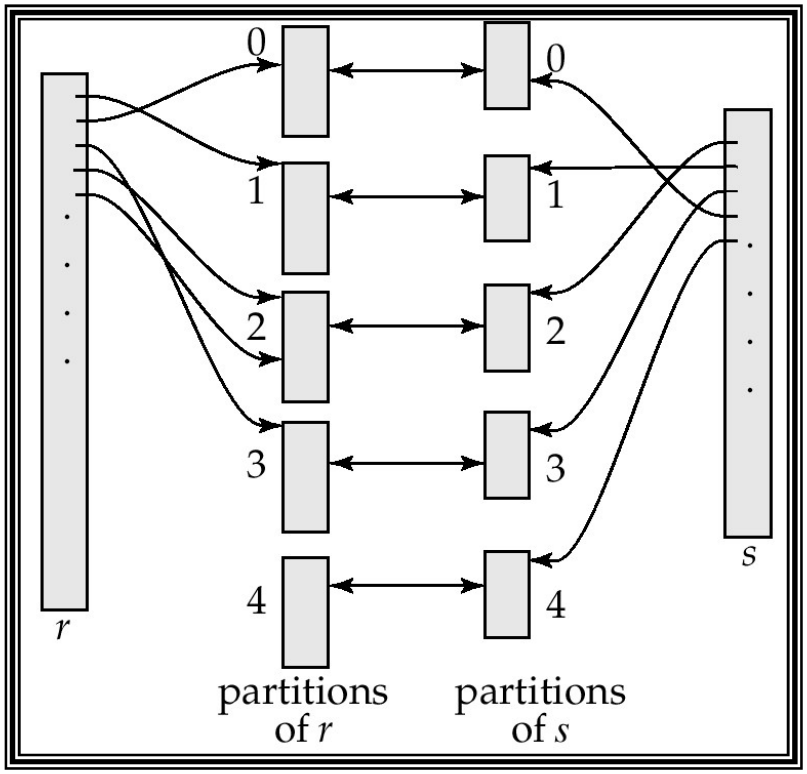
- 归并连接又称为排序归并连接：如果关系r、s已分别按连接属性A、B排序，那么依此顺序扫描两文件，找出所有A、B属性值相等的记录。显然，只要A、B为关键属性，经一次扫描，即可找出所有匹配元组。
- 由于需要连接的两个文件都只需要读一遍,所需访问磁盘次数为两个文件块数之和： b_s+b_r



如，s为外层，r为内层。开始：s选a，比较r的第一个记录a，满足，取出a，再比较r的b记录，b>a，则s指向b，满足，取出b，r指向下一条记录d，d>b，s下移c，d>c，s继续下移d，满足，取出d，r继续下移，满足，取出d，以此类推

连接运算-散列连接

- 由于r.A和s.B两属性具有相同的域，故可作为Hash关键字，用同一Hash函数将r、s映射到同一个Hash文件中。
- 一般r、s文件各扫描一次即可将文件中的记录(或记录指针)散列到Hash文件桶(划分)中。
- 依次检索每个桶，只要把每个桶中匹配的元组取出，即可获得连接结果。
- 由于一个桶中元组不会太多，可用嵌套循环法找出匹配元组。



核心思想：相同的属性值一定在相同的桶中，仅比较对应桶中的属性即可。

散列连接的形式化描述

- 假设
  - **h是将JoinAttrs值映射到[0, 1, …, n_h]的散列函数, JoinAttrs表示r与s自然连接中的公共属性。**
  - **H_{r0}, H_{r1}, …, H_{rm}表示关系r的元组分区，开始为空。每个元组$t_r \in r$被放入分区H_{ri}中，其中i=h(t_r[JoinAttrs])**
  - **H_{s0}, H_{s1}, …, H_{sn}表示关系s的元组分区，开始为空。每个元组$t_s \in s$被放入分区H_{si}中，其中i=h(t_s[JoinAttrs])**
- 比较连接
  - 如果h(t_r[JoinAttrs])=h(t_s[JoinAttrs])则进行t_r[JoinAttrs]和t_s[JoinAttrs]的比较(因不同的值可能有相同的Hash)，若相等则进行连接
  - 如果h(t_r[JoinAttrs])≠h(t_s[JoinAttrs])则不需要进行比较连接

散列连接代价分析

- 划分阶段：两个关系**r**和**s**的划分需要对这两个关系分别进行一次完整的读入和写出，该操作需要**$2(b_r+b_s)$**次块存取。
- 探查阶段：每个分区分别读入一次，需要**b_r+b_s**次块存取。
- 分区所占块数可能比**b_r+b_s**略多，因为有些区是部分满的块（偏斜），存取这些部分满的块的额外代价每个关系最多不超过**$2n_h$**
- 所以，散列连接的代价估计是：

  $3(b_r+b_s)+4n_h$，**$4n_h << 3(b_r+b_s)$**，所以可以忽略

目录

- 查询处理的基本步骤
- 算法代价估算
- 表达式运算结果估算
- 表达式计算
- 表达式等价转换
- 启发式优化方法

为什么要进行
表达式运算结果估算？

表达式结果统计信息的估算

- 查询处理运算的各种算法代价的计算依据一个假设 — 已知关系的基本信息
- 这些基本信息十分重要，没有这些基本信息就无法计算出代价具体值，也就无法进行优化了
- 这些基本信息有两个来源
  - 根据关系的统计信息，如对永久关系可直接获得这些信息
  - 根据表达式结果统计信息估算，表达式的结果仍然是关系，如临时关系、中间结果等，只有通过统计估算才能获得统计信息

表达式结果统计信息如何估算？

表达式结果是关系
因此估算的结果之一是元组规模
关系的元组规模在代价估算中常用

选择运算结果估计

- 选择运算结果大小估计依赖于选择谓词，分别讨论
- $\sigma_{A=a}(r)$
  - 假设取值是均匀分布的，关系r的一些记录的属性A的取值为a，则选择结果的估计值为n_r/Dist(A, r)个元组（平均值）
  - 上述假设在现实中的有些情况不成立，如关系account中的branch-name属性，大的支行比小的支行账户数目多。但多数情况下的近似是合理的。
- $\sigma_{A \le v}(r)$
  - 假设值是平均分布, 若v<min(A, r), 则A $\le v$的记录数为0; 若$v \ge$ max(A, r), 则记录数为n_r; 否则, 为

$$n_r \cdot (v\text{-min}(A, r)/(\max(A, r)\text{-min}(A, r)))$$

其中, max(A, r)表示关系r的属性A的最大值, min(A, r)同理。

- 合取$\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n}(r)$: 对每一个$\theta_i$，采用上述$\sigma_{\theta i}(r)$的估算方式得到其大小为$s_i$，因此，关系中一个元组满足条件$\theta_i$的概率为$s_i / n_r$(中选率)。据此，可以估算出满足全部选择条件的元组数量为:

  $$n_r \cdot (s_1 \cdot s_2 \cdot \ldots \cdot s_n / n_r^n) \qquad \text{(独立同分布)}$$

- 析取$\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$: 元组满足整个析取式的概率为1减去元组不满足任何一个条件的概率，即

  $1-(1-(s_1/n_r))(1-(s_2/n_r))\ldots(1-(s_n/n_r))$, 该值乘$n_r$即为满足析取条件的元组数. 满足条件$\theta_i$的概率为$s_i / n_r$(中选率)

- 取反$\sigma_{\neg\theta}(r)$: 在无空值情况下，其结果为不在$\sigma_\theta(r)$中的关系r的元组集，元组数为$n_r - \sigma_\theta(r)$; 如果有空值，则再减去空值数目。

连接运算结果估算

- 主要考虑以下两种运算的结果集的大小
  - 笛卡儿积（相对简单）
  - 自然连接（重点探讨）
- 笛卡儿积
  - $r \times s$包含 $n_r \cdot n_s$ 个元组，每个元组占用 l_r+l_s 个字节，由此可以计算出笛卡儿积的大小

自然连接结果估算

- 自然连接结果集大小的估计：
  - 基于主码、外码连接的情况：结果集的元组数等于外码所在关系的元组数（一对多的多的一方）。
  - 一般情况：（假设连接属性A的每个值在关系的元组中等概率出现），结果集的元组数为：

    $n_r \cdot n_s$ / Dist(A, s) 或$n_r \cdot n_s$ / Dist(A, r)
- 说明：
  - 当Dist(A, s)与Dist(A, r)不同时，取两个估计值中较小者。
  - 若两个关系中的元组在连接属性上的取值能匹配上的很少时，上述估计值太高。但实际上这种情况很少发生。

直观例子-连接运算结果估计

$n_{depositor}$=7
$n_{customer}$=13
连接后元组数为7。

$n_r \cdot n_s$/Dist(A, s)和$n_r \cdot n_s$/Dist(A, r)中大的值把没有满足条件的元组也做了连接，所以要取小值。

可以利用上述计算方法演算一下

depositor

	ID	customer_name	ac
1	01	hayes	A-
2	02	johnson	A-
3	03	johnson	A-
4	04	jones	A-
5	05	lindsay	A-
6	06	smith	A-
7	07	turner	A-

customer

	customer_name	customer_street	customer_city
1	adams	spring	pittsfield
2	brooks	senator	brooklyn
3	curry	north	rye
4	glenn	sand hill	woodside
5	green	walnut	stamford
6	hayes	main	harrison
7	johnson	alma	palo alto
8	jones	main	harrison
9	lindsay	park	pittsfield
10	smith	north	rye
11	turner	putnam	stamford
12	williams	nassau	princeton

depositor⋈customer

	customer_name	account_number	customer_street	customer_city
1	hayes	A-102	main	harrison
2	johnson	A-101	alma	palo alto
3	johnson	A-201	alma	palo alto
4	jones	A-217	main	harrison
5	lindsay	A-222	park	pittsfield
6	smith	A-215	north	rye
7	turner	A-305	putnam	stamford

其他运算结果估计

- 投影：形如$\Pi_A(r)$的投影大小(元组数或记录数)估计为**Dist(A,r)**， 投影消除了重复元组
- 聚集：由于聚集运算结果的一个元组代表一组取值，即估计元组数为**Dist(A,r)**
- 集合：如果一个集合运算的两个输入是对同一个关系的选择，可以将该运算重写成析取、合取或取反，如可以把$\sigma_{\theta 1}(r) \cup \sigma_{\theta 2}(r)$重写为$\sigma_{\theta 1 \vee \theta 2}(r)$
- 只要参与集合运算的两个关系是对同一个关系的选择，**r∪s**估计为**r+s**，**r∩s**估计为**r-s**，**r-s**估计为**r**，这三种估计提供了结果集合的上界。

表达式结果是关系
因此估算的结果之二是元组取值个数
元组取值个数在代价估算中也常用

不同取值个数的估计

- 关系r中属性A的不同取值个数Dist(A, *r*)估算包括：
  - 选择操作的估算：Dist(A, $\sigma_\theta(r)$)
  - 连接操作的估算：Dist(A, $r \bowtie s$)
  - 聚集操作估算
- Dist(A, $\sigma_\theta(r)$)估算：
  - 选择条件A=特定值，如A=3，则Dist(A, $\sigma_\theta(r)$) = 1
  - 如 θ: A in {a,b,c}，则Dist(A,$\sigma_\theta(r)$) = 3
  - 若选择条件为A op v的形式，op为比较运算，则Dist(A,$\sigma_\theta(r)$) = Dist(A, *r*)* $SF_{A\ op\ v}(r)$， $SF_{A\ op\ v}(r)$ 为关系r上的谓词“A op v”选择率（参考P26）
  - 其他，近似处理为min(Dist(*A*,*r*), $n_{\sigma\theta(r)}$)，用概率论可得更好结果，但上面已经够用

- **Dist(A, $r \bowtie s$)估算**
  - 若A中所有属性全来自r，则Dist(A, $r \bowtie s$)可估计为min (Dist(A,r), $n_{r \bowtie s}$)
  - 跨表属性集合A上值的个数，A1来自r，A2来自s，则Dist(A, $r \bowtie s$)可估计为

    Dist(A, $r \bowtie s$) = min(Dist(A1, r)*Dist(A2 – A1, s), Dist(A1 – A2, r) * Dist(A2, s), $n_{r \bowtie s}$)
- 聚集运算的估算（从略，参考教材自学）

小结

- 查询处理过程分为(1) 语法分析与翻译(2) 优化(3) 执行，其中优化是一个技术性很强的环节
- 算法代价估算的目的是实现查询优化
- 表达式结果估算的目的是为算法代价估算提供必要的信息
- 估算所利用的信息是统计信息，统计信息有两类，一类是系统提供的（基本表），另一类是根据已知的基表统计信息及表达式估算出来的（主要包括元组规模和属性取值个数）

目录

- 查询处理的基本步骤
- 算法代价估算
- 表达式运算结果估算
- 表达式计算
- 表达式等价转换
- 启发式优化方法

表达式计算

- 表达式计算(又称组合操作)所解决的问题是：当计算包括多个运算表达式的时候，中间计算结果如何处理？
- 两种解决方案：
  - 实体化：以适当的顺序将每次计算的结果存储到临时关系以备用，若临时关系很大，则需要写到磁盘上。（有库存，物化，备用，买方市场）
  - 流水线：在所谓流水线上同时计算多个运算，一个运算结果传递给下一个，不必保存临时关系。（零库存，不物化，需要时准备，卖方市场）

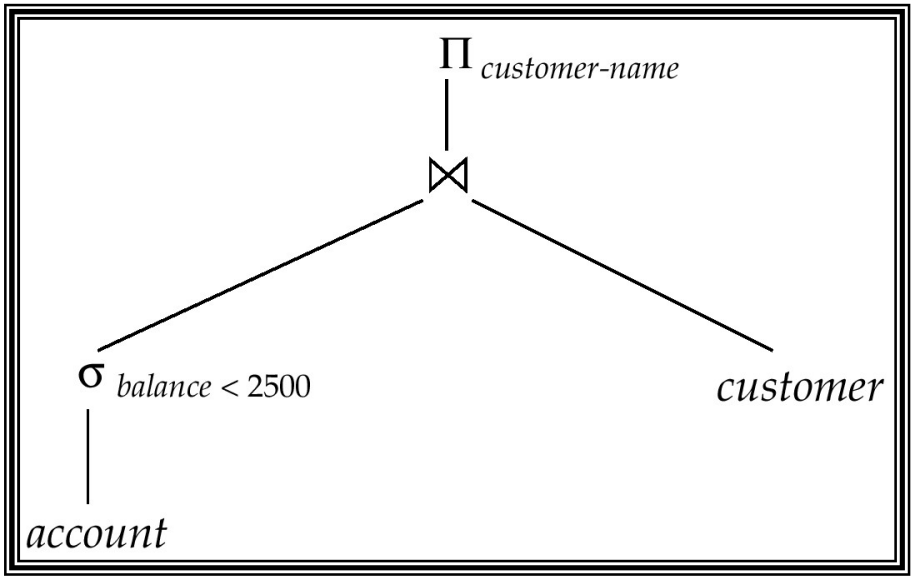
表达式计算-实体化

- 考虑一个表达式：

  $\Pi_{customer\text{-}name}(\sigma_{balance<250}(account) \bowtie customer)$

  该表达式的运算树标识了从内层(底层)开始计算，临时关系依次建立

产生临时关系f_1



产生结果关系f

产生临时关系f_2

实体化的代价

- 实体化计算的代价包括两个部分
  - 运算代价，参考前述各项计算
  - 临时关系的读写操作，当缓冲区满时，写磁盘的代价估算为b_r或n_r/f_r，n_r表示关系r中元组估计数，f_r表示关系的块因子(每块的关系r中的记录数)，b_r见统计信息部分

  实体化计算代价为上述两部分的和。
- 由于存在写盘操作，所以实体化方法的代价高，但总是可行的

表达式计算-流水线

- 考虑一个表达式：$\Pi_{a1,a2}(r \bowtie s)$
  - 当连接运算产生一个结果元组时，该元组马上传送给投影运算去处理，避免了中间结果的创建，而直接产生最后结果。
- 流水线执行算法
  - 一个运算所用算法的选择与流水线技术的选择不是相互独立的。如对连接运算，归并连接在未排序时是不能使用的（排序，内存不足），而索引嵌套循环连接算法是可以的。
- 流水线计算的代价取决于相应的执行算法
- 代价低，但有时不可行

目录

- 查询处理的基本步骤
- 算法代价估算
- 表达式运算结果估算
- 表达式计算
- 表达式等价转换
- 启发式优化方法

查询优化的多个策略如何产生？

关系表达式的转换

- 如果两个关系表达式在任一种有数据库实例种都会产生相同的元组集，则称它们是等价的。
- 等价规则指出两种不同形式的表达式是等价的。
- 等价规则(很多，参考教材：查询优化相关部分)

1. 合取选择运算可分解为单个选择运算序列, 该变换称为σ级联

$$\sigma_{\theta 1 \wedge \theta 2}(E) = \sigma_{\theta 1}(\sigma_{\theta 2}(E))$$

2. 交换律

$$\sigma_{\theta 1}(\sigma_{\theta 2}(E)) = \sigma_{\theta 2}(\sigma_{\theta 1}(E))$$

3. 投影运算序列中只有最后一个运算是需要的,其余的可省略.该转换也可称为Π级联

$$\Pi_{L1}(\Pi_{L2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L1}(E)$$

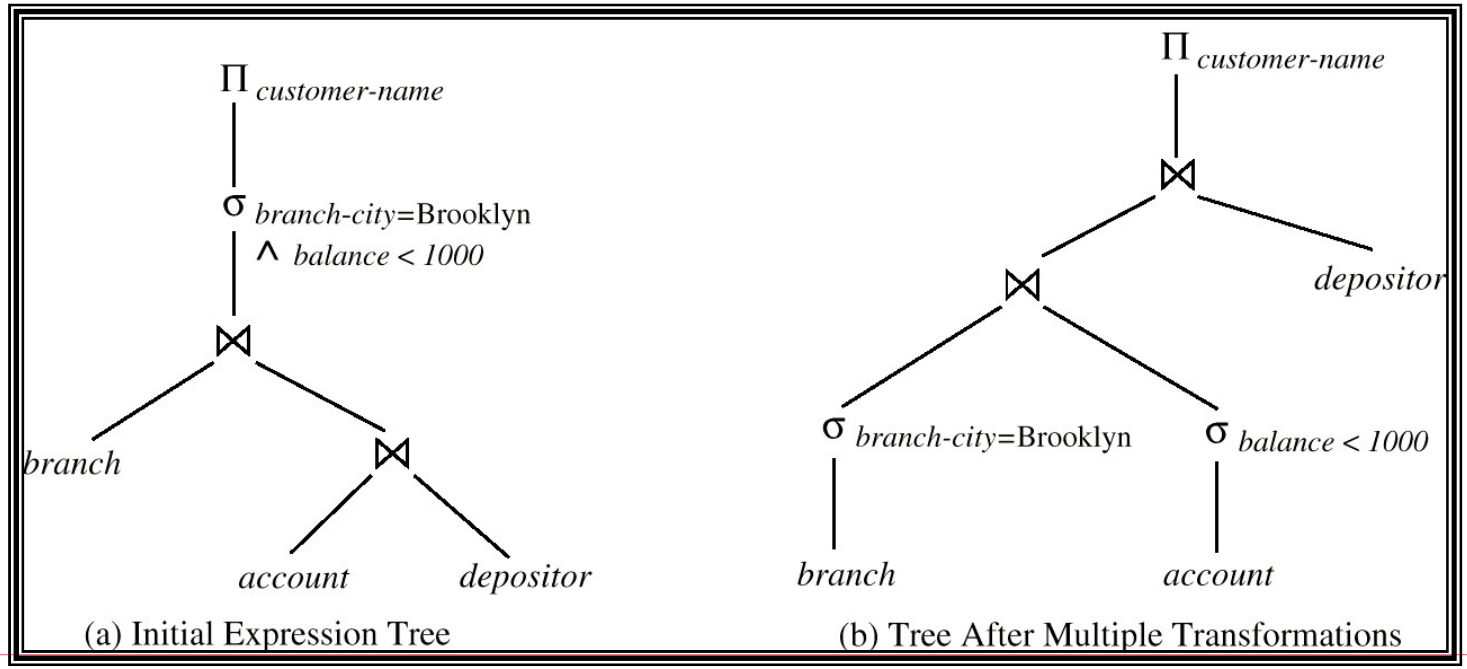
4. 选择运算可与笛卡儿积以及θ连接相结合

a. $\sigma_{\theta}(E1 \times E2) = E1 \bowtie_{\theta} E2$

b. $\sigma_{\theta 1}(E1 \bowtie_{\theta 2} E2) = E1 \bowtie_{\theta 1 \wedge \theta 2} E2$

等价规则的应用

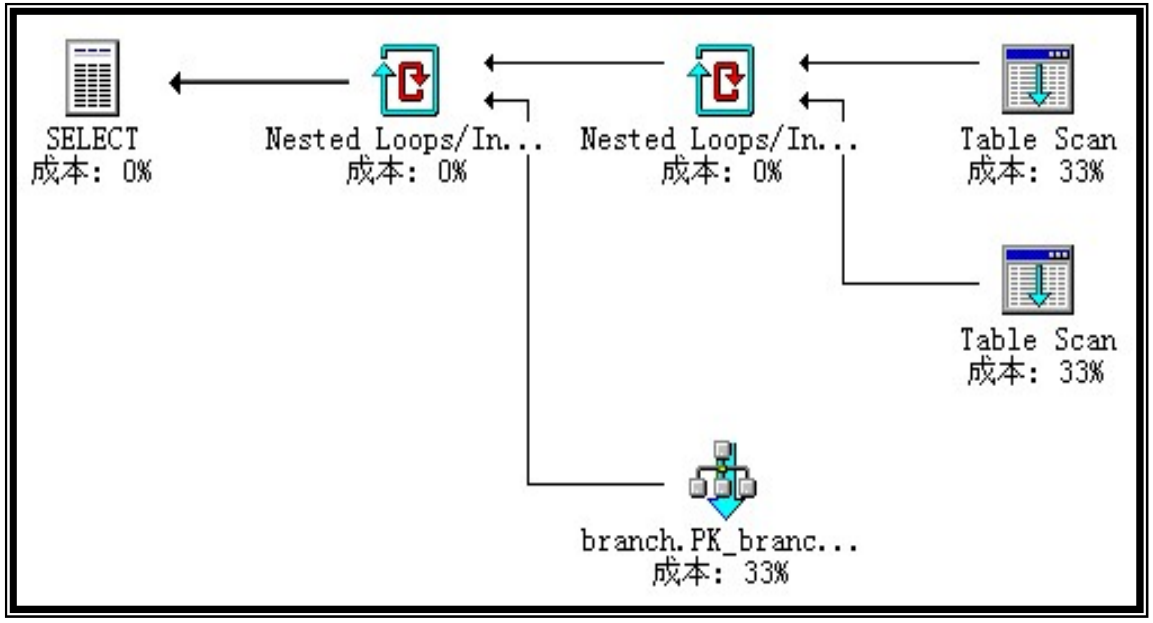
- 规则说明两个表达式等价，并未表明孰优孰劣
- 生成的表达式作为一种备选执行策略，需要进行代价估算后才能确定最优
- 一组等价规则被称为最小等价规则集，若其中任意一条规则都不能被其他规则组合导出。一般查询优化器使用最小等价规则集



例子：一个实际的执行计划

- 表达式

  select ***loan.branch_name,amount*** **from** ***loan,branch,depositor***
  where ***loan.branch_name='Perryridge'***



目录

- 查询处理的基本步骤
- 算法代价估算
- 表达式运算结果估算
- 表达式计算
- 表达式等价转换
- 启发式优化方法

关于启发式优化

- 启发式优化
  - 利用启发式信息进行执行计划的优化过程，这些启发式信息是一些经验技术的总结，由于优化不能总是保证奏效,所以称其为启发式。
  - 启发式优化需要依据一些规则进行

理解：启发式优化

- 使用“窍门”求解问题，往往能达到事半功倍的目的
- 这种窍门来自经验，与求解的问题密切有关，在大多数情况下能很快解决问题
- 对许多任务来说，有可能使用与任务有关的信息而大大减少优化的代价，并找到比较满意的解
- 这种与任务或问题有关的信息称为启发信息，利用启发信息进行的优化叫做启发式优化

启发式规则

- 常用启发式规则
  - 尽早执行选择运算：根据前面的讨论可知,这条规则一般情况下是可以减少执行代价的。但也有特殊情况。如$\sigma_{\theta}(r \bowtie s)$，如果θ是关于s的条件且s的连接属性上有索引，r很小。则先执行选择操作意味着对s进行一遍扫描，其代价比r与s在有索引的连接可能要高的多。
  - 尽早执行投影运算：选择运算要先于投影运算，前者能大大减小关系。投影运算减小了元组，从而使磁盘块能够容纳更多的元组，减少了I/O操作代价

- 在执行连接前对关系进行适当预处理，如建立索引或排序可显著提高连接的效率
- 投影和选择运算同时进行，如有若干同应和选择运算，并且都是对同一关系操作，则可以在扫描此关系的同时完成两个操作,避免重复扫描
- 把某些选择同在它前面要执行的笛卡儿积结合起来称为一个连接运算，连接运算的效率要高于笛卡儿积运算
- 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，并且从外存读入这个关系比计算该子表达式的时间少的多，则先计算一次公共子表达式并把结果写入中间关系是合算的。

典型启发式算法的步骤

- 将合取选择分解为单个选择运算的序列，有助于将选择运算向查询树的下层移动。
- 把选择运算移到查询树下面一边尽可能早执行的地方。
- 确定哪些选择运算与连接运算将产生最小的关系(即所含元组数最少的关系)，这一步考虑的是满足选择或连接条件中的中选率
- 将跟有选择条件的笛卡儿运算替换称连接运算.
- 将投影属性加以分解并在查询树上尽可能下移，若有必要，可引入新的投影
- 识别哪些可用流水线方式执行其运算的子树，并采用流水线方法执行

选择操作

- 通过选择合理的路径达到优化的一些规则
  - 对于小关系，不必考虑其它存取路径，直接顺序扫描
  - 对于主搜索码的等值查询，最多只有一个元组满足条件，因此应优先采用主关键字上的索引或散列
  - 对于非主搜索码的等值查询，要估计中选的元组数在关系中所占的比例，比例小(＜20％)可用无序索引，比例大只能用聚集索引或顺序扫描
  - 对于合取选择条件，若诸子条件涉及的同性中有两个以上存在存取路径，应优先选择检索结果记录较少的存取路径。

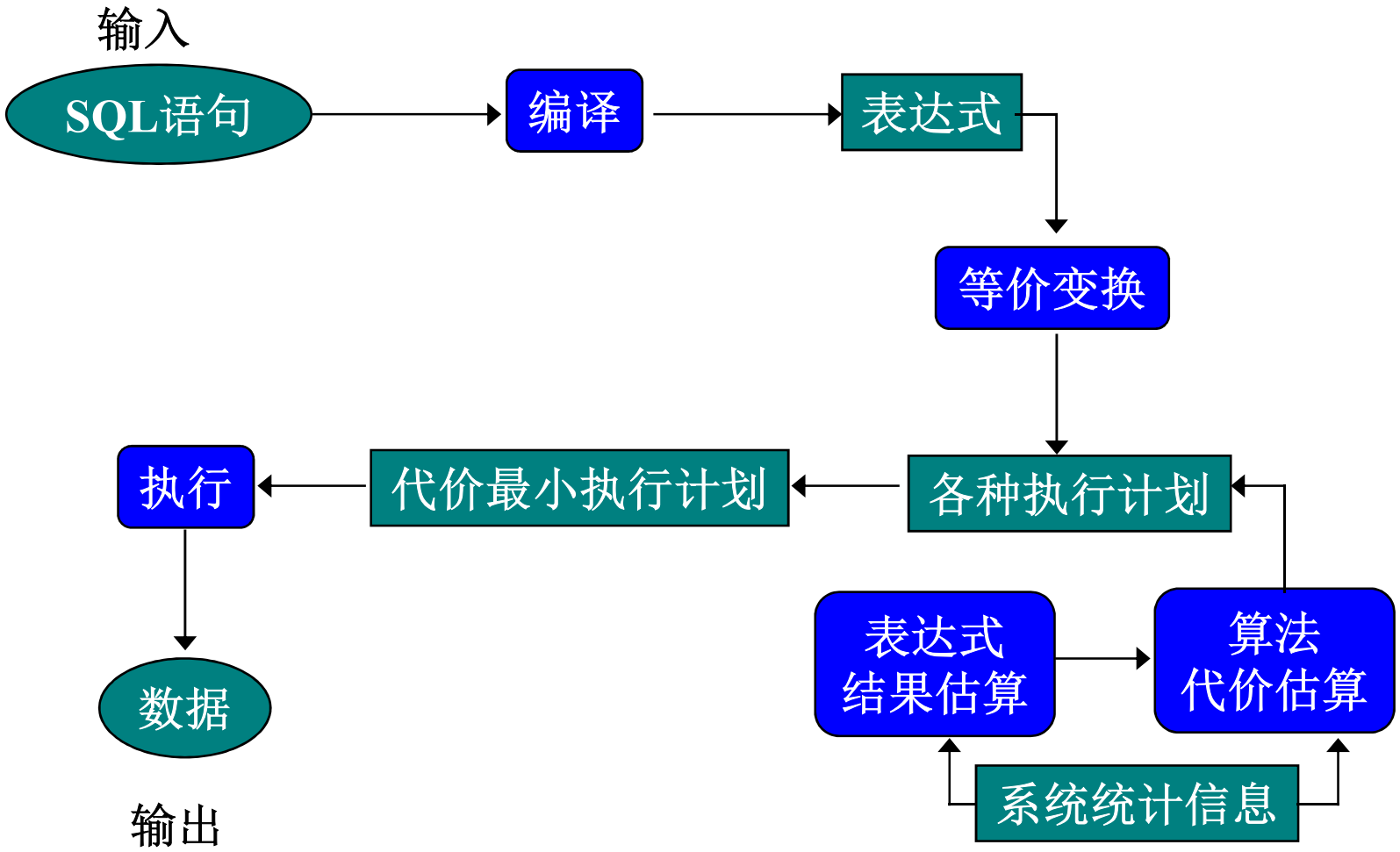
连接操作

- 如果两个关系都已按连接属性排序，则应优先采用归并排序法。即使只有一个关系已按连接属性排序，而另一关系很小，也可考虑先对此小关系按连接属性排序，而后用排序归并法连接。
- 如果两个关系中有某个关系存在连接属性上的索引或散列，则令其为内关系，另一关系为外关系，顺序扫描外关系，而利用索引或散列寻找匹配元组，无需多次扫描内关系
- 当参与连接的两关系尚无其它任何存取路径时，可用嵌套循环法或散列连接法。实际使用上述各种方法时，还可进一步优化。例如，对于嵌套循环法．假设外文件占**B1**块，内文件占**B2**块，缓冲区为**n**块，那么总的读块次数为**B1+[B1/n-1]×B2**。显然，当**n＞2**时，应将物理块少的关系作为外关系。

小结

- 查询优化是一个过程，执行计划的选择依据代价估算，代价估算可采用查询处理运算的相关算法，但这些算法的代价计算则需要表达式结果估算信息
- 表达式结果估算的依据是系统提供的目录信息和对各运算结果的估计
- 表达式等价变换产生各种执行计划，利用各运算算法及表达式结果估算可估算出每个执行计划的代价
- 启发式优化根据启发式规则实现，与表达式结果估算不同，启发式优化基本不需要复杂的计算
- 物理优化将文件组织、查询处理等方法考虑进来实现优化的方法
- 查询优化实现了优化过程的自动化，将程序员解放出来

附录：知识结构


输入
SQL语句
编译
表达式
等价变换
各种执行计划
代价最小执行计划
执行
数据
输出
表达式
结果估算
算法
代价估算
系统统计信息