第9章：查询执行

Query Execution

邹兆年

哈尔滨工业大学
计算机科学与技术学院
海量数据计算研究中心
电子邮件：znzou@hit.edu.cn

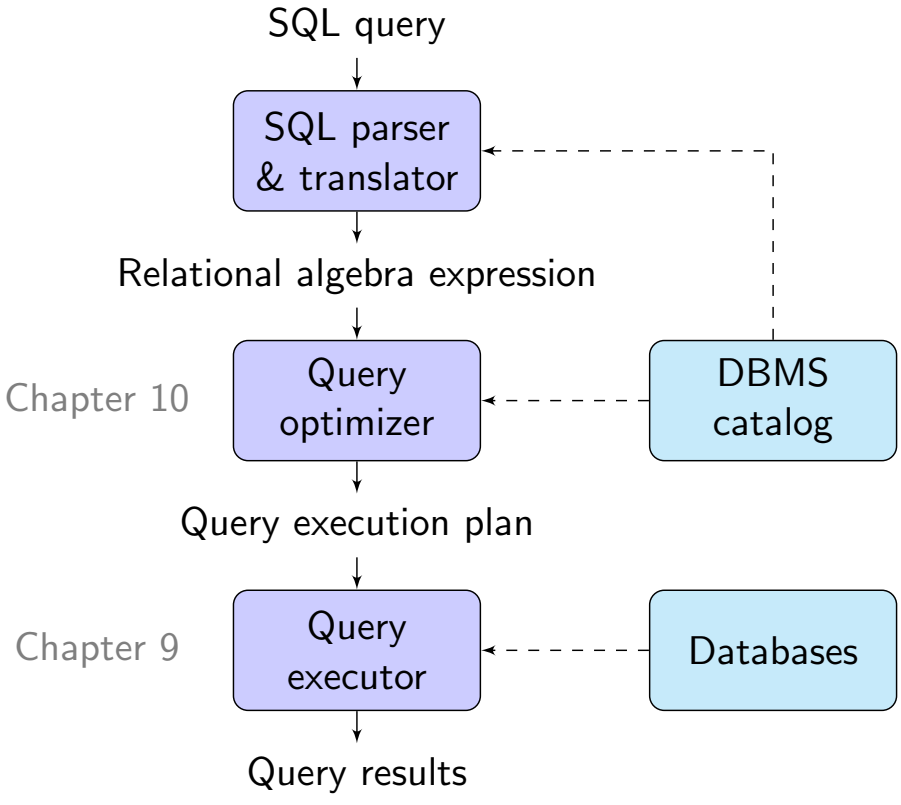
2021年春

教学内容[1]

[1]课件更新于2021年4月18日

Overview

Overview



SQL Parser & Translator

The SQL parser and transaltor transform an SQL query to an relational algebra expression

Example (Query Translation)

- Relation: instructor(ID, name, dept_name, salary)
- SQL query:

```
SELECT ID, salary FROM instructor WHERE salary < 75000;
```

- Relational algebra expression:

$$\Pi_{\texttt{ID,salary}}(\sigma_{\texttt{salary}<75000}(\texttt{instructor}))$$

Query Optimizer

The query optimizer transforms an initial relational algebra expression to an optimized relational algebra expression and finally to a physical query execution plan

Example (Query Optimization)

Initial expression	Alternative expression	Query execution plan
$\Pi_{ID,salary}$	$\sigma_{salary<75000}$	$\Pi_{ID,salary}$
$\sigma_{salary<75000}$	$\Pi_{ID,salary}$	$\sigma_{salary<75000}$; use index
instructor	*instructor*	*instructor*

External Sort

排序(Sorting)

按照排序键对元组进行排序是DBMS中非常重要的操作

- 用户使用`ORDER BY`对查询结果进行排序
- 批量加载(bulk loading) B+树的第一步是对索引项(index entry)进行排序
- 排序是关系代数操作执行过程的重要步骤

外排序(External Sorting)

当数据规模大到无法全部载入内存时，需要使用外排序算法

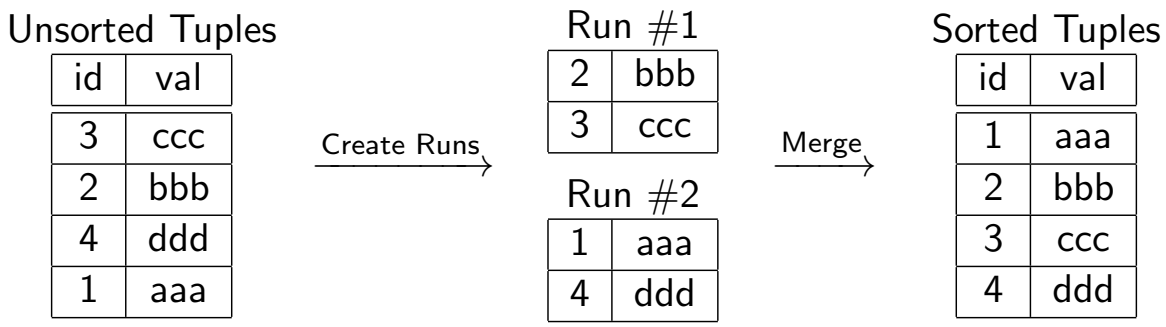
最小化外排序算法的磁盘访问(disk access)开销

- CPU计算时间在外排序算法的执行时间中只占很少一部分
- 外排序算法的执行时间主要用于磁盘访问
- 磁盘访问开销用磁盘输入/输出(disk I/O)来近似衡量

External Sort
External Merge Sort

两趟多路外存归并排序(Two-Pass Multiway External Merge Sort)

- 第1阶段: 创建归并段(run)
- 第2阶段: 归并(merge)



记法

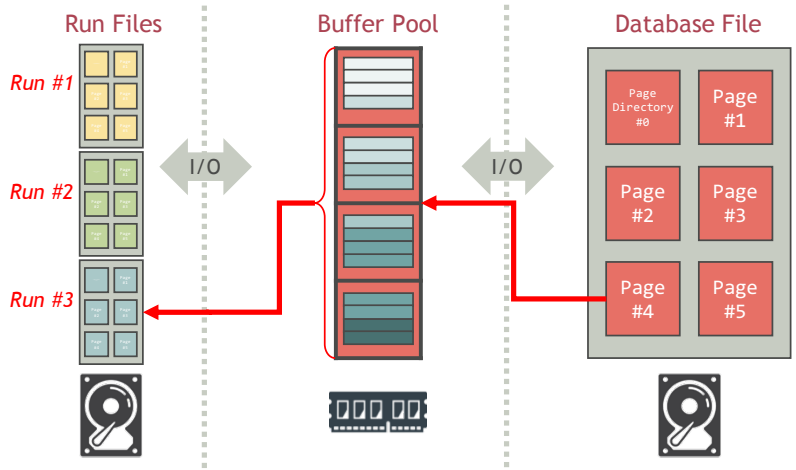
- N: R的元组数
- M: 缓冲池中可用内存页数
- B: 每块最多存B个元组
- $B(R) = \lceil N/B \rceil$: R的块数

创建归并段(Create Runs)

将关系R划分为$\lceil B(R)/M \rceil$个归并段(run)

过程

1: **for** R的每M块 **do**
2: 将这M块读入缓冲池中M页
3: 对这M页中的元组按排序键(sort key)进行排序，形成归并段(run)
4: 将该归并段写入文件



算法运行实例

Example (归并排序)

- $R =$ | 2 5 | 2 1 | 2 2 | 4 5 | 4 3 | 4 2 | 1 5 | 2 1 | 3 | (每个数代表一个元组)
- $N = 17$
- $M = 3$
- $B = 2$
- $B(R) = 9$

创建归并段

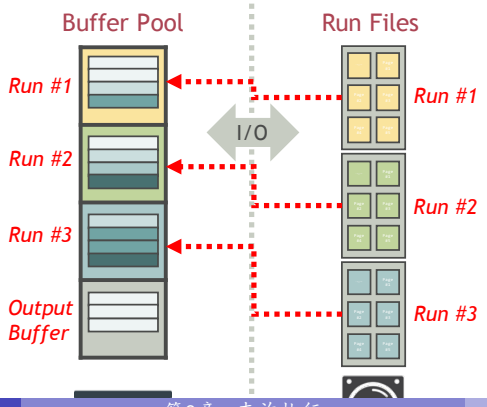
- $R_1 =$ | 1 2 | 2 2 | 2 5 |
- $R_2 =$ | 2 3 | 4 4 | 4 5 |
- $R_3 =$ | 1 1 | 2 3 | 5 |

多路归并(Multiway Merge)

将$\lceil B(R)/M \rceil$个归并段中的元组进行归并

过程

1: 将每个归并段的第1块读入缓冲池的1页
2: **repeat**
3: 找出所有输入缓冲页中最小的排序键值v
4: 将所有排序键值等于v的元组写入输出缓冲区
5: 任意输入缓冲页中的元组若归并完毕，则读入其归并段的下一页
6: **until** 所有归并段都已归并完毕



算法运行实例

多路归并

Run	In memory	Waiting on disk
R_1	1 2	2 2 | 2 5
R_2	2 3	4 4 | 4 5
R_3	1 1	2 3 | 5

Output: 1 1 | 1 (Red integers indicate tuples just outputted)

Run	In memory	Waiting on disk
R_1	2	2 2 | 2 5
R_2	2 3	4 4 | 4 5
R_3	2 3	5

Output: 1 1 | 1 2 | 2 2

Run	In memory	Waiting on disk
R_1	2 2	2 5
R_2	3	4 4 | 4 5
R_3	3	5

Output: 1 1 | 1 2 | 2 2 | 2 2

Run	In memory	Waiting on disk
R_1	2 5	
R_2	3	4 4 | 4 5
R_3	3	5

Output: 1 1 | 1 2 | 2 2 | 2 2 | 2

Run	In memory	Waiting on disk
R_1	5	
R_2	3	4 4 | 4 5
R_3	3	5

Output: 1 1 | 1 2 | 2 2 | 2 2 | 2 3 | 3

Run	In memory	Waiting on disk
R_1	5	
R_2	4 4	4 5
R_3	5	

Output: 1 1 | 1 2 | 2 2 | 2 2 | 2 3 | 3 4 | 4

Run	In memory	Waiting on disk
R_1	5	
R_2	4 5	
R_3	5	

Output: | 1 1 | 1 2 | 2 2 | 2 2 | 2 3 | 3 4 | 4 4 |

Run	In memory	Waiting on disk
R_1	5	
R_2	5	
R_3	5	

Output: | 1 1 | 1 2 | 2 2 | 2 2 | 2 3 | 3 4 | 4 4 | 5 5 | 5 |

算法分析

分析算法时不考虑结果输出操作

- 输出结果时产生的I/O不计入算法的I/O代价
  输出结果可能直接作为后续操作的输入，无需写入文件
- 输出缓冲区不计入可用内存页数
  如果输出结果直接作为后续操作的输入，那么输出缓冲区将计入后续操作的可用内存页数

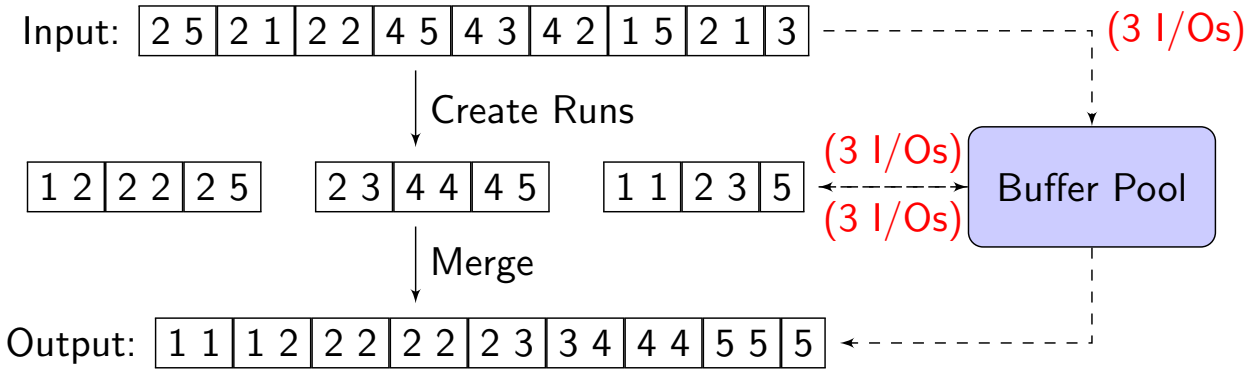
算法分析

I/O代价: $3B(R)$

- 在创建归并段时，R的每块读1次，合计$B(R)$次I/O
- 将每个归并段写入文件，合计$B(R)$次I/O
- 在归并阶段，每个归并段扫描1次，合计$B(R)$次I/O

可用内存页数要求: $B(R) \leq M^2$

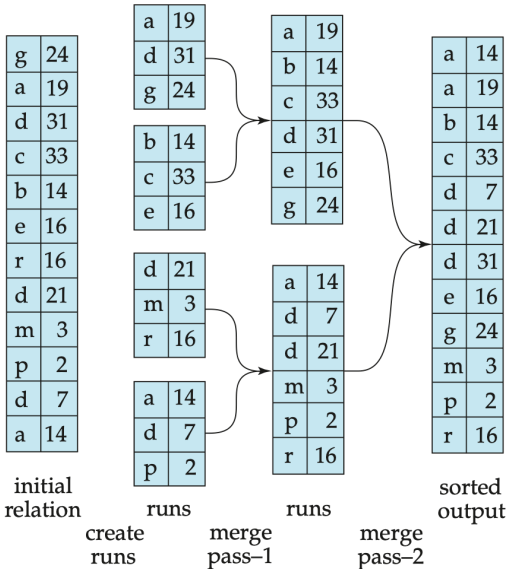
- 每个归并段不超过M页
- 最多M个归并段



多趟多路外存归并排序(Multi-Pass Multiway External Merge Sort)

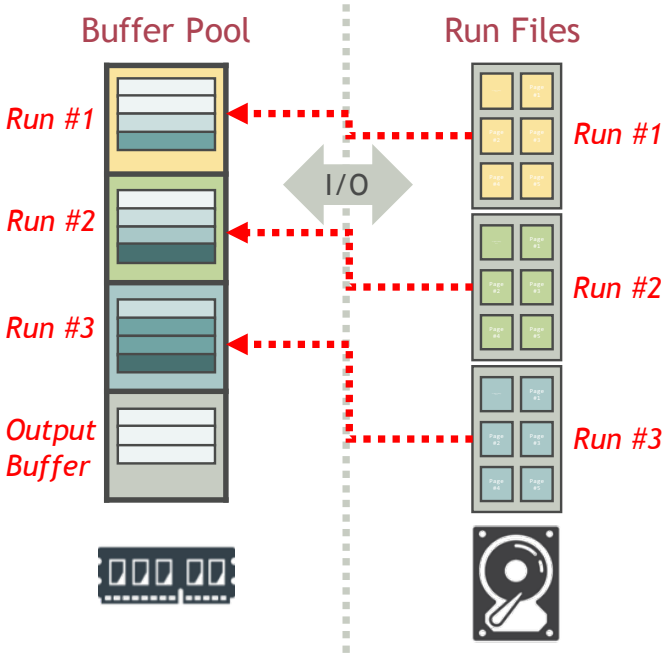
若$B(R) > M^2$，则需要执行多趟多路外存归并排序

- I/O代价是$(2m-1)B(R)$，其中m算法执行的趟数



多路归并排序的优化

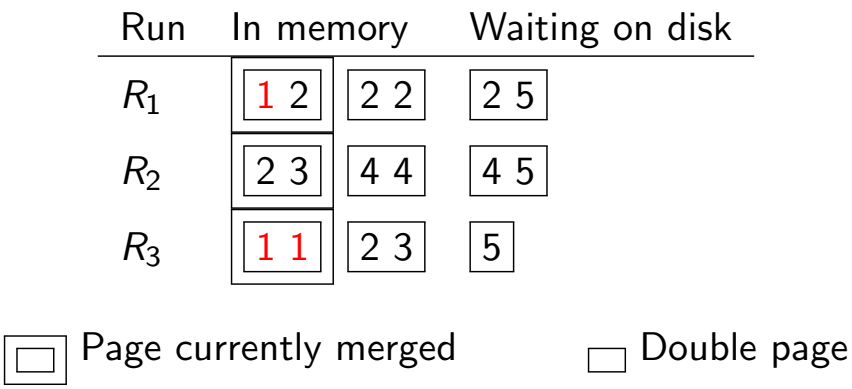
当某缓冲页中所有元组都已归并完毕，DBMS需读入其归并段的下一块
此时，归并进程被挂起(suspend)，直至I/O完成



双缓冲(Double Buffering)

为每个归并段分配多个内存页作为输入缓冲区，并组成环形(circular)

- 在当前缓冲页中所有元组都已归并完毕时，DBMS直接开始归并下一缓冲页中的元组
- 与此同时，DBMS将归并段文件的下一块读入空闲的缓冲页



Execution of Operations of Relational Algebra

Execution of Operations of Relational Algebra
Execution of Selection Operations

选择操作的执行

- 方法1: 基于扫描的选择算法(Scanning-based Selection)
- 方法2: 基于哈希的选择算法(Hash-based Selection)
- 方法3: 基于索引的选择算法(Index-based Selection)

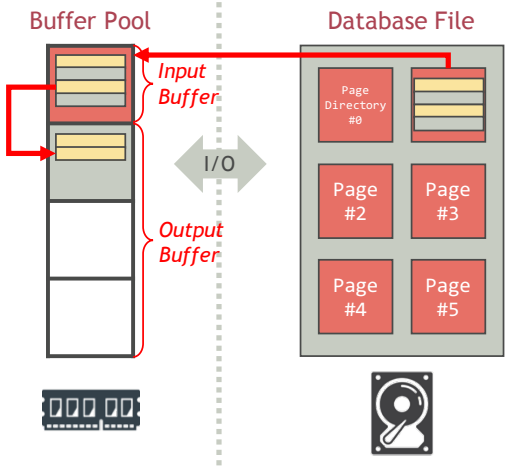
记法

- $T(R)$: 关系R的元组数
- $B(R)$: 关系R的块数
- M: 缓冲池可用内存页数
- $V(R, A)$: 关系R的属性集A的不同值的个数

基于扫描的选择算法(Scanning-based Selection)

算法

1: **for** R的每一块P **do**
2: 将P读入缓冲池
3: **for** P中每条元组t **do**
4: **if** t满足选择条件 **then**
5: 将t写入输出缓冲区



算法分析

分析算法时不考虑结果输出操作

- 输出结果时产生的I/O不计入算法的I/O代价
  输出结果可能直接作为后续操作的输入，无需写入文件
- 输出缓冲区不计入可用内存页数
  如果输出结果直接作为后续操作的输入，那么输出缓冲区将计入后续操作的可用内存页数

算法分析

I/O代价: $B(R)$ (R采用聚簇存储)

- R的元组连续存储于文件中
- R的每块只读1次

I/O代价: $T(R)$ (R不采用聚簇存储)

- R的元组不连续存储于文件中
- 最坏情况下，R的元组均在不同页上

可用内存页数要求: $M \geq 1$

- 至少需要1页作为缓冲区，用于读R的每一块

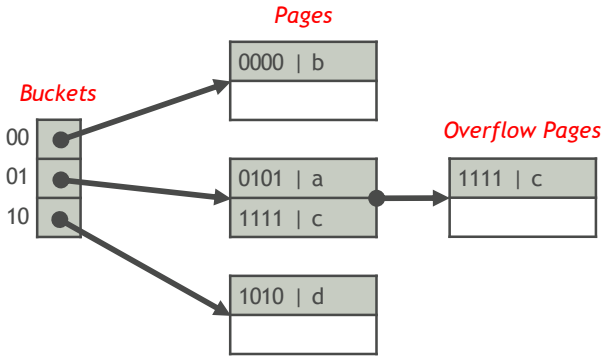
基于哈希的选择算法(Hash-based Selection)

使用该算法的前提条件

- 选择条件的形式是$K = v$
- 关系R采用哈希文件组织形式(hash-based file organization)
- 属性K是R的哈希键(hash key)

算法

1: 根据$hash(v)$确定结果元组所在的桶
2: 在桶页面中搜索键值等于v的元组，并将元组写入输出缓冲区



算法分析

I/O代价 $\approx \lceil B(R)/V(R,K) \rceil$

- 属性K有$V(R,K)$个不同的值
- 每个桶平均有$\lceil B(R)/V(R,K) \rceil$个页(很不准确的估计)

可用内存页数要求: $M \geq 1$

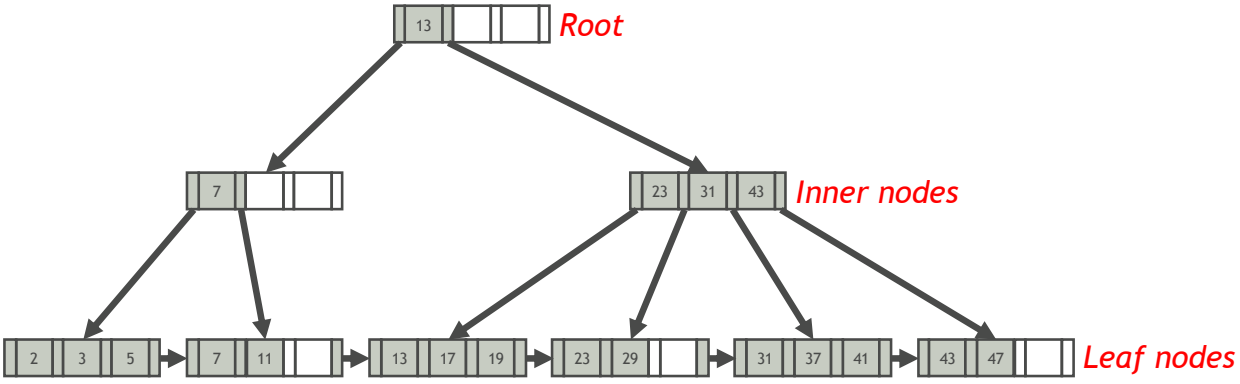
- 至少需要1页作为缓冲区，用于读桶的每一页

基于索引的选择算法(Index-based Selection)

使用该算法的前提条件

- 选择条件的形式是$K = v$或$l \leq K \leq u$
- 关系R上建有属性K的索引

在索引上搜索满足选择条件的元组，并将元组写入输出缓冲区



算法分析

I/O代价 $\approx \lceil B(R)/V(R,K) \rceil$ (如果索引是聚簇索引)

- 结果元组连续存储于文件中
- 属性K有$V(R,K)$个不同的值
- 结果元组约占$\lceil B(R)/V(R,K) \rceil$个页(很不准确的估计)

I/O代价 $\approx \lceil T(R)/V(R,K) \rceil$ (如果索引是非聚簇索引)

- 约有$\lceil T(R)/V(R,K) \rceil$个结果元组(很不准确的估计)
- 结果元组不一定连续存储于文件中
- 最坏情况下，所有结果元组均在不同页上

可用内存页数要求: $M \geq 1$

- 至少需要1页作为缓冲区，用于读B+树的节点

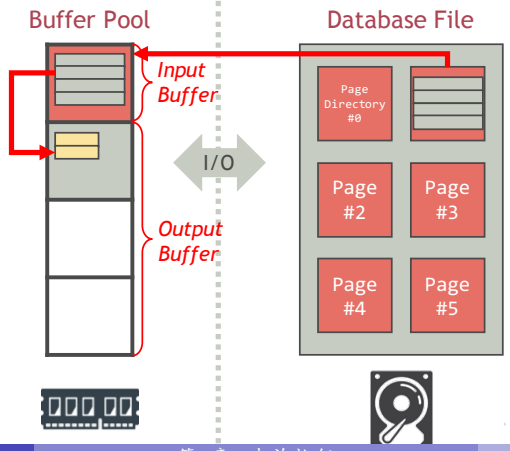
Execution of Operations of Relational Algebra
Execution of Projection Operations

不带去重的投影算法

算法

1: **for** R的每一块P **do**
2: 将P读入缓冲池
3: **for** P中每条元组t **do**
4: 将t向投影属性集做投影
5: 将投影元组写入输出缓冲区

该算法在数据访问模式(access pattern)上与基于扫描的选择算法相同



算法分析

I/O代价: $B(R)$ (R采用聚簇存储)

- R的元组连续存储于文件中
- R的每块只读1次

I/O代价: $T(R)$ (R不采用聚簇存储)

- R的元组不连续存储于文件中
- 最坏情况下，R的元组均在不同页上

可用内存页数要求: $M \geq 1$

- 至少需要1页作为缓冲区，用于读R的每一块

Execution of Operations of Relational Algebra
Execution of Duplicate Elimination Operations

去重(Duplicate Elimination)操作

关系R上的去重操作$\delta(R)$返回R中互不相同的元组

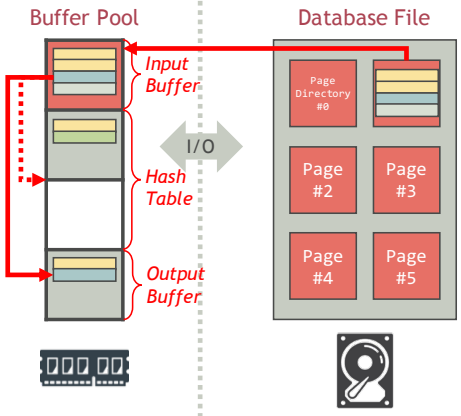
- 方法1: 一趟去重算法(One-pass Duplicate Elimination)
- 方法2: 基于排序的去重算法(Sort-based Duplicate Elimination)
- 方法3: 基于哈希的去重算法(Hash-based Duplicate Elimination)

一趟去重算法(One-Pass Duplicate Elimination)

算法

1: **for** R的每一块P **do**
2: 将P读入缓冲池
3: **for** P中每条元组t **do**
4: **if** 未见过t **then**
5: 将t写入输出缓冲区

在可用内存页中用哈希表记录见过的元组，哈希键为整个元组



算法运行实例

Example (一趟去重算法)

- $R =$ | 2 5 | 2 1 | 2 2 | 4 5 | 4 3 | 4 2 | 1 5 | 2 1 | 3 |
- $M = 4$
- $B = 2$

Input buffer (1 block)	Hash table ($\leq M-1$ blocks)	Output buffers
2 5	2 5	2 5
2 1	2 5 | 1	2 5 | 1
2 2	2 5 | 1	2 5 | 1
4 5	2 5 | 1 4	2 5 | 1 4
4 3	2 5 | 1 4 | 3	2 5 | 1 4 | 3
4 2	2 5 | 1 4 | 3	2 5 | 1 4 | 3
1 5	2 5 | 1 4 | 3	2 5 | 1 4 | 3
2 1	2 5 | 1 4 | 3	2 5 | 1 4 | 3
3	2 5 | 1 4 | 3	2 5 | 1 4 | 3

算法分析

该算法在数据访问模式上与基于扫描的选择算法相同

I/O代价: $B(R)$ (R采用聚簇存储)

- R的元组连续存储于文件中
- R的每块只读1次

I/O代价: $T(R)$ (R不采用聚簇存储)

- R的元组不连续存储于文件中
- 最坏情况下，R的元组均在不同页上

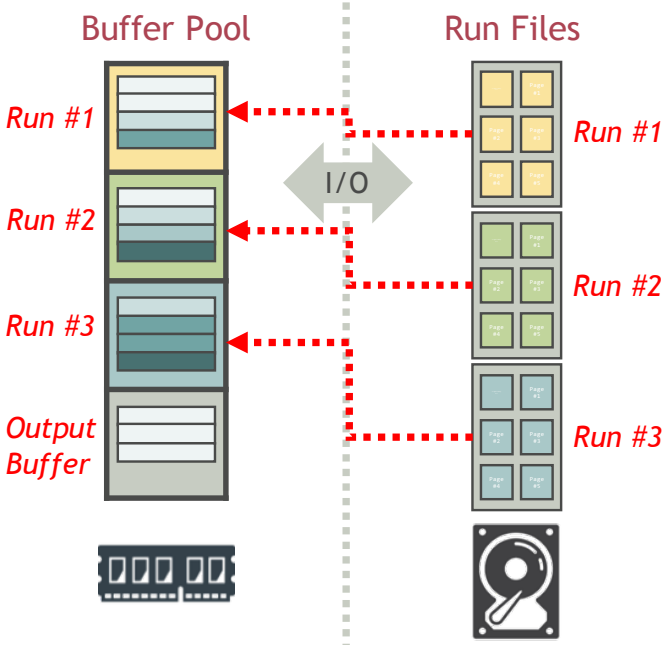
可用内存页数要求: $B(\delta(R)) \leq M-1$

- R中互不相同的元组$\delta(R)$必须能在$M-1$页中存得下

基于排序的去重算法(Sort-based Duplicate Elimination)

基于排序的去重算法与多路归并排序(multiway merge sort)算法本质上一样，两点区别如下:

- 在创建归并段(run)时，按整个元组进行排序
- 在归并阶段，相同元组只输出1个，其他全部丢弃



算法运行实例

Example (基于排序的去重算法)

- $R =$ | 2 5 | 2 1 | 2 2 | 4 5 | 4 3 | 4 2 | 1 5 | 2 1 | 3 |
- $M = 3$
- $B = 2$

创建归并段

- $R_1 =$ | 1 2 | 2 2 | 2 5 |
- $R_2 =$ | 2 3 | 4 4 | 4 5 |
- $R_3 =$ | 1 1 | 2 3 | 5 |

多路归并

Run	In memory	Waiting on disk
R_1	1 2	2 2 | 2 5
R_2	2 3	4 4 | 4 5
R_3	1 1	2 3 | 5

Output: 1 (Red integers indicate the duplicate records)

Run	In memory	Waiting on disk
R_1	2	2 2 | 2 5
R_2	2 3	4 4 | 4 5
R_3	2 3	5

Output: 1 2

Run	In memory	Waiting on disk
R_1	2 2	2 5
R_2	3	4 4 | 4 5
R_3	3	5

Output: 1 2

Run	In memory	Waiting on disk
R_1	2 5	
R_2	3	4 4 | 4 5
R_3	3	5

Output: 1 2

Run	In memory	Waiting on disk
R_1	5	
R_2	3	4 4 | 4 5
R_3	3	5

Output: 1 2 | 3

Run	In memory	Waiting on disk
R_1	5	
R_2	4 4	4 5
R_3	5	

Output: 1 2 | 3 4

Run	In memory	Waiting on disk
R_1	5	
R_2	4 5	
R_3	5	

Output: 1 2 | 3 4

Run	In memory	Waiting on disk
R_1	5	
R_2	5	
R_3	5	

Output: 1 2 | 3 4 | 5

算法分析

I/O代价: $3B(R)$

- 在创建归并段时，R的每块读1次，合计$B(R)$次I/O
- 将每个归并段写入文件，合计$B(R)$次I/O
- 在归并阶段，每个归并段扫描1次，合计$B(R)$次I/O

可用内存页数要求: $B(R) \leq M^2$

- 每个归并段不超过M页
- 最多M个归并段

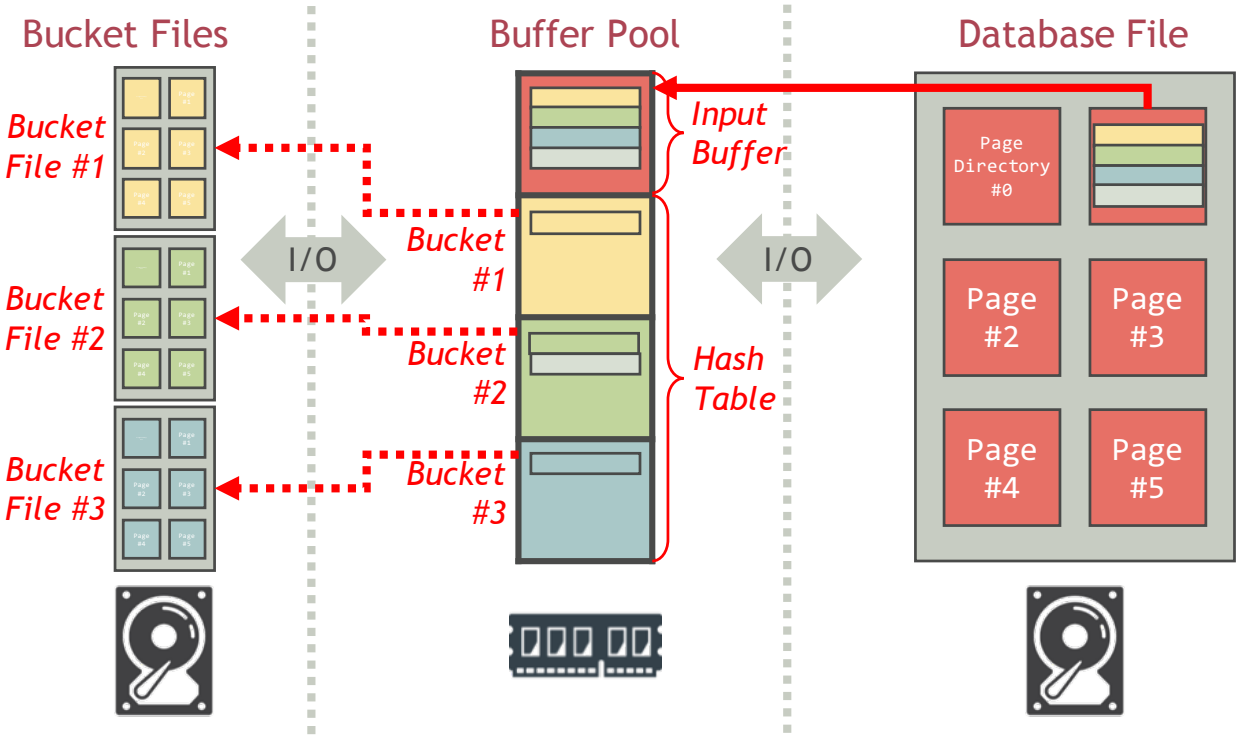
基于哈希的去重算法(Hash-based Duplicate Elimination)

算法

1: // 哈希分桶
2: **for** R的每一块P **do**
3: 将P读入缓冲池
4: 将P中元组哈希到$M-1$个桶$R_1, R_2, \ldots, R_{M-1}$中(哈希键为整个元组)
5: // 逐桶去重
6: **for** $i = 1, 2, \ldots, M-1$ **do**
7: 在桶R_i上执行一趟去重(one-pass duplicate elimination)算法，并将结果写到输出缓冲区

哈希分桶

重复元组必落入相同桶中



按桶去重

每个桶R_i的去重结果放在一起就得到了R的去重结果

- $\delta(R) = \bigcup_{i=1}^{M-1} \delta(R_i)$
- $\delta(R_i) \cap \delta(R_j) = \emptyset$ for $i \neq j$

算法运行实例

Example (基于哈希的去重算法)

- $R =$ | 2 5 | 2 1 | 2 2 | 4 5 | 4 3 | 4 2 | 1 5 | 2 1 | 3 |
- $M = 4$
- $B = 2$
- $h(K) = K \mod 3$

哈希分桶

- $R_0 =$ | 3 3 |
- $R_1 =$ | 1 4 | 4 4 | 1 1 |
- $R_2 =$ | 2 5 | 2 2 | 2 5 | 2 5 | 2 |

逐桶去重

- $\delta(R_0) =$ | 3 |
- $\delta(R_1) =$ | 1 4 |
- $\delta(R_2) =$ | 2 5 |

算法分析

I/O代价: $3B(R)$

- 在哈希分桶时，R的每块读1次，合计$B(R)$次I/O
- 将每个桶写入文件，合计$\sum_{i=1}^{M-1} B(R_i) \approx B(R)$次I/O
- 在每个桶R_i上执行一趟去重算法的I/O代价是$B(R_i)$

可用内存页数要求: $B(R) \leq (M-1)^2$

- 共$M-1$个桶
- 每个桶不超过$M-1$块，因此在每个桶上执行一趟去重算法时，可用内存页数满足要求

Execution of Operations of Relational Algebra
Execution of Aggregation Operations

聚集操作(Aggregation Operations)的执行

聚集操作和去重操作的执行在本质上一样

- 方法1: 一趟聚集算法(One-pass Aggregation)
- 方法2: 基于排序的聚集算法(Sort-based Aggregation)
- 方法3: 基于哈希的聚集算法(Hash-based Aggregation)

聚集算法的设计和分析留作课后练习

Execution of Operations of Relational Algebra
Execution of Set Operations

集合差操作的执行

- 方法1: 一趟集合差算法(One-pass Set Difference)
- 方法2: 基于排序的集合差算法(Sort-based Set Difference)
- 方法3: 基于哈希的集合差算法(Hash-based Set Difference)

一趟集合差算法(One-Pass Set Difference)

算法

```
1: // 构建(build)阶段
2: 在M − 1个可用内存页中建立一个内存查找结构(哈希表或平衡二叉树)，查找键是整个元组
3: for S的每一块P do
4:     将P读入缓冲池
5:     将P中元组插入内存查找结构
6: // 探测(probe)阶段
7: for R的每一块P do
8:     将P读入缓冲池
9:     for P中每条元组t do
10:        if t不在于内存查找结构中 then
11:            将t写入输出缓冲区
```


算法运行实例

Example (一趟集合差算法)

- $R =$ | 1 5 | 8 2 | 3 10 | 4 7 | 6 9 |
- $S =$ | 4 11 | 9 5 | 7 3 | 6 12 | 8 10 |
- $M = 6$
- $B = 2$

Order	The Mth block	Output
1	1 5	1
2	8 2	1 2
3	3 10	1 2
4	4 7	1 2
5	6 9	1 2

算法分析

I/O代价: $B(R)+B(S)$

- 在构建(build)阶段，S的每块只读1次，合计$B(S)$次I/O
- 在探测(probe)阶段，R的每块只读1次，合计$B(R)$次I/O

可用内存页数要求: $B(S) \leq M-1$

- 内存查找结构约占$B(S)$页

基于哈希的集合差算法(Hash-based Set Difference)

算法

1: // 哈希分桶(与基于哈希的去重算法的分桶方法相同)
2: 将R的元组哈希到$M-1$个桶$R_1, R_2, \ldots, R_{M-1}$中(哈希键为整个元组)
3: 将S的元组哈希到$M-1$个桶$S_1, S_2, \ldots, S_{M-1}$中(哈希键为整个元组)
4: // 逐桶计算集合差
5: **for** $i = 1, 2, \ldots, M-1$ **do**
6: 使用一趟集合差(one-pass set difference)算法计算$R_i - S_i$，并将结果写入输出缓冲区

- R和S中相同的元组一定分别落入同号桶R_i和S_i中
- $R - S = \bigcup_{i=1}^{M-1}(R_i - S_i)$
- $(R_i - S_i) \cap (R_j - S_j) = \emptyset$ for $i \neq j$

算法运行实例

Example (基于哈希的集合差算法)

- $R =$ | 1 5 | 8 2 | 3 10 | 4 7 | 6 9 |
- $S =$ | 4 11 | 9 5 | 7 3 | 6 12 | 8 10 |
- $M = 4$
- $B = 2$
- $h(K) = K \mod 3$

R的桶

- $R_0 =$ | 3 6 | 9 |
- $R_1 =$ | 1 10 | 4 7 |
- $R_2 =$ | 5 2 | 8 |

S的桶

- $S_0 =$ | 9 3 | 6 12 |
- $S_1 =$ | 4 7 | 10 |
- $S_2 =$ | 11 5 | 8 |

集合差

- $R_0 - S_0 = \emptyset$
- $R_1 - S_1 =$ | 1 |
- $R_2 - S_2 =$ | 2 |

算法分析

I/O代价: $3B(R) + 3B(S)$

- 在对R进行哈希分桶时，R的每块读1次，合计$B(R)$次I/O
- 将R的桶全部写入文件，需$\sum_{i=1}^{M-1} B(R_i) \approx B(R)$次I/O
- 在对S进行哈希分桶时，S的每块读1次，合计$B(S)$次I/O
- 将S的桶全部写入文件，需$\sum_{i=1}^{M-1} B(S_i) \approx B(S)$次I/O
- 使用一趟集合差算法计算$R_i - S_i$的I/O代价是$B(R_i) + B(S_i)$

可用内存页数要求: $B(S) \leq (M-1)^2$

- S共有$M-1$个桶
- S的每个桶不超过$M-1$块

基于排序的集合差算法(Sort-based Set Difference)

算法

1: // 创建归并段
2: 将R划分为$\lceil B(R)/M \rceil$个归并段(每个归并段按整个元组进行排序)
3: 将S划分为$\lceil B(S)/M \rceil$个归并段(每个归并段按整个元组进行排序)
4: // 归并
5: 读入R和S的每个归并段的第1页
6: **repeat**
7: 找出输入缓冲区中最小的元组t
8: **if** $t \in R$且$t \notin S$ **then**
9: 将t写入输出缓冲区
10: 从输入缓冲区中删除t的所有副本
11: 任意输入缓冲页中的元组若归并完毕，则读入其归并段的下一页
12: **until** R的所有归并段都已归并完毕

算法分析

I/O代价: $3B(R) + 3B(S)$

- 在对R创建归并段时，R的每块只读1次，合计$B(R)$次I/O
- 将R的归并段全部写入文件，需$B(R)$次I/O
- 在对S创建归并段时，S的每块只读1次，合计$B(S)$次I/O
- 将S的归并段全部写入文件，需$B(S)$次I/O
- 在归并阶段，对R和S的每个归并段各扫描1次，合计$B(R) + B(S)$次I/O

可用内存页数要求: $B(R) + B(S) \leq M^2$

- R和S的每个归并段均不超过M块
- R和S共有不超过M个归并段

集合并操作的执行

集合并操作和集合差操作的执行在本质上一样

- 方法1: 一趟集合并算法(One-pass Set Union)
- 方法2: 基于排序的集合并算法(Sort-based Set Union)
- 方法3: 基于哈希的集合并算法(Hash-based Set Union)

集合并算法的设计和分析留作课后练习

集合交操作的执行

集合交操作和集合差操作的执行在本质上一样

- 方法1: 一趟集合交算法(One-pass Set Intersection)
- 方法2: 基于排序的集合交算法(Sort-based Set Intersection)
- 方法3: 基于哈希的集合交算法(Hash-based Set Intersection)

集合交算法的设计和分析留作课后练习

Execution of Operations of Relational Algebra
Execution of Join Operations

连接(Join)操作的执行

下面以$R(X, Y) \bowtie S(Y, Z)$为例，介绍连接操作的执行算法

- 方法1: 一趟连接算法(One-Pass Join)
- 方法2: 嵌套循环连接算法(Nested-Loop Join)
- 方法3: 排序归并连接算法(Sort-Merge Join)
- 方法4: 基于哈希的连接算法(Hash-based Join)
- 方法5: 基于索引的连接算法(Index-based Join)

一趟连接算法(One-Pass Join)

假设: $B(S) \leq B(R)$

算法

1: // 构建(build)阶段
2: 在$M-1$个可用内存页中建立一个内存查找结构(哈希表或平衡二叉树)，查找键是$S.Y$
3: **for** S的每一块P **do**
4: 将P读入缓冲池
5: 将P中元组插入内存查找结构
6: // 探测(probe)阶段
7: **for** R的每一块P **do**
8: 将P读入缓冲池
9: **for** P中每条元组r **do**
10: **for** 内存查找结构中每条键值等于$r.Y$的元组s **do**
11: 连接r和s，并将结果写入输出缓冲区

算法运行实例

Example (一趟连接算法)

- $R(X, Y) = \{(1,1),(5,5),(3,2),(3,1),(2,1),(4,2)\}$
- $S(Y, Z) = \{(2,6),(1,7),(1,8),(2,5),(2,7)\}$
- $M = 4$
- $B = 2$

Order	The Mth block	Output
1	$(1,1),(5,5)$	$(1,1,7),(1,1,8)$
2	$(3,2),(3,1)$	$(3,2,6),(3,2,5),(3,2,7),(3,1,7),(3,1,8)$
3	$(2,1),(4,2)$	$(2,1,7),(2,1,8),(4,2,6),(4,2,5),(4,2,7)$

算法分析

I/O代价: $B(R)+B(S)$

- 在构建(build)阶段，S的每块只读1次，合计$B(S)$次I/O
- 在探测(probe)阶段，R的每块只读1次，合计$B(R)$次I/O

可用内存页数要求: $B(S) \leq M-1$

- 内存查找结构约占$B(S)$页

基于元组的嵌套循环连接(Tuple-based Nested-Loop Join)

算法

1: **for** S的每个元组s **do**
2: **for** R的每个元组r **do**
3: **if** r和s满足连接条件 **then**
4: 连接r和s，并将结果写入输出缓冲区

- S称为外关系(outer relation)
- R称为内关系(inner relation)

算法分析

I/O代价: $T(S)(T(R)+1)$

- 外关系S的每个元组只读1次，每次产生1个I/O，合计$T(S)$次I/O
- 内关系R的每个元组读$T(S)$次，每次产生1个I/O，合计$T(S)T(R)$次I/O

可用内存页数要求: $M \geq 2$

- 1页作为读S的缓冲区
- 1页作为读R的缓冲区

基于块的嵌套循环连接(Block-based Nested-Loop Join)

假设: $B(S) \leq B(R)$

算法

1: **for** 外关系S的每$M-1$块 **do**
2: 将这$M-1$块读入缓冲池
3: 用一个内存查找结构来组织这$M-1$块中的元组
4: **for** 内关系R的每一块P **do**
5: 将P读入缓冲池
6: **for** P中每条元组r **do**
7: **for** 内存查找结构中能与r进行连接的元组s **do**
8: 连接r和s，并将结果写入输出缓冲区

算法运行实例

Example (基于块的嵌套循环连接算法)

- $R(X,Y) = \{(1,1),(5,5),(3,2),(3,1),(2,1),(4,2)\}$
- $S(Y,Z) = \{(2,6),(1,7),(1,8),(2,5),(2,7)\}$
- $M = 3$
- $B = 2$

First $M-1$ blocks	Mth block	Output
$(2,6),(1,7),(1,8),(2,5)$	$(1,1),(5,5)$	$(1,1,7),(1,1,8)$
$(2,6),(1,7),(1,8),(2,5)$	$(3,2),(3,1)$	$(3,2,6),(3,2,5),(3,1,7),(3,1,8)$
$(2,6),(1,7),(1,8),(2,5)$	$(2,1),(4,2)$	$(2,1,7),(2,1,8),(4,2,6),(4,2,5)$
$(2,7)$	$(1,1),(5,5)$	
$(2,7)$	$(3,2),(3,1)$	$(3,2,7)$
$(2,7)$	$(2,1),(4,2)$	$(4,2,7)$

算法分析

I/O代价: $B(S) + \frac{B(R)B(S)}{M-1}$

- 外关系S的每一块只读1次，合计$B(S)$次I/O
- 内关系R扫描$B(S)/(M-1)$次，合计$\frac{B(R)B(S)}{M-1}$次I/O

可用内存页数要求: $M \geq 2$

- 至少1页作为读S的缓冲区
- 1页作为读R的缓冲区

排序归并连接(Sort-Merge Join)

算法

1: // 创建归并段
2: 将R划分为$\lceil B(R)/M \rceil$个归并段(每个归并段按$R.Y$进行排序)
3: 将S划分为$\lceil B(S)/M \rceil$个归并段(每个归并段按$S.Y$进行排序)
4: // 归并
5: 读入R和S的每个归并段的第1页
6: **repeat**
7: 找出输入缓冲区中元组Y属性的最小值y
8: **for** R中满足$R.Y = y$的元组r **do**
9: **for** S中满足$S.Y = y$的元组s **do**
10: 连接r和s，并将结果写入输出缓冲区
11: 任意输入缓冲页中的元组若归并完毕，则读入其归并段的下一页
12: **until** R或S的所有归并段都已归并完毕

算法运行实例

Example (SMJ)

- $R(X, Y) =$
  $\{(1,1),(5,4),(3,2),(3,1),(6,3),(2,1),(4,2),(8,5),(4,1),(3,4)\}$
- $S(Y, Z) =$
  $\{(2,6),(1,7),(1,8),(5,9),(5,3),(2,5),(3,1),(2,7),(3,7),(4,9)\}$
- $M = 4$
- $B = 2$

创建归并段

- $R_1 = \{(1,1),(2,1),(3,1),(3,2),(6,3),(5,4)\}$
- $R_2 = \{(4,1),(4,2),(3,4),(8,5)\}$
- $S_1 = \{(1,7),(1,8),(2,5),(2,6),(5,3),(5,9)\}$
- $S_2 = \{(2,7),(3,1),(3,7),(4,9)\}$

多路归并

Run	In memory	Waiting on disk
R_1	$(1,1),(2,1)$	$(3,1),(3,2),(6,3),(5,4)$
R_2	$(4,1),(4,2)$	$(3,4),(8,5)$
S_1	$(1,7),(1,8)$	$(2,5),(2,6),(5,3),(5,9)$
S_2	$(2,7),(3,1)$	$(3,7),(4,9)$

Run	In memory	Waiting on disk
R_1	$(1,1),(2,1)$	$(6,3),(5,4)$
	$(3,1),(3,2)$	
R_2	$(4,1),(4,2)$	$(3,4),(8,5)$
S_1	$(1,7),(1,8)$	$(5,3),(5,9)$
	$(2,5),(2,6)$	
S_2	$(2,7),(3,1)$	$(3,7),(4,9)$

Output: $(1,1,7),(1,1,8),(2,1,7),(2,1,8),(3,1,7),(3,1,8)$

Run	In memory	Waiting on disk
R_1	$(3,2)$	$(6,3),(5,4)$
R_2	$(4,2)$	$(3,4),(8,5)$
S_1	$(2,5),(2,6)$	$(5,3),(5,9)$
S_2	$(2,7),(3,1)$	$(3,7),(4,9)$

Run	In memory	Waiting on disk
R_1	$(3,2)$	
	$(6,3),(5,4)$	
R_2	$(4,2)$	
	$(3,4),(8,5)$	
S_1	$(2,5),(2,6)$	
	$(5,3),(5,9)$	
S_2	$(2,7),(3,1)$	$(3,7),(4,9)$

Output: $(1,1,7),(1,1,8),(2,1,7),(2,1,8),(3,1,7),(3,1,8),$
$(3,2,5),(3,2,6),(3,2,7),(4,2,5),(4,2,6),(4,2,7)$
后续过程略

算法分析

I/O代价: $3B(R)+3B(S)$

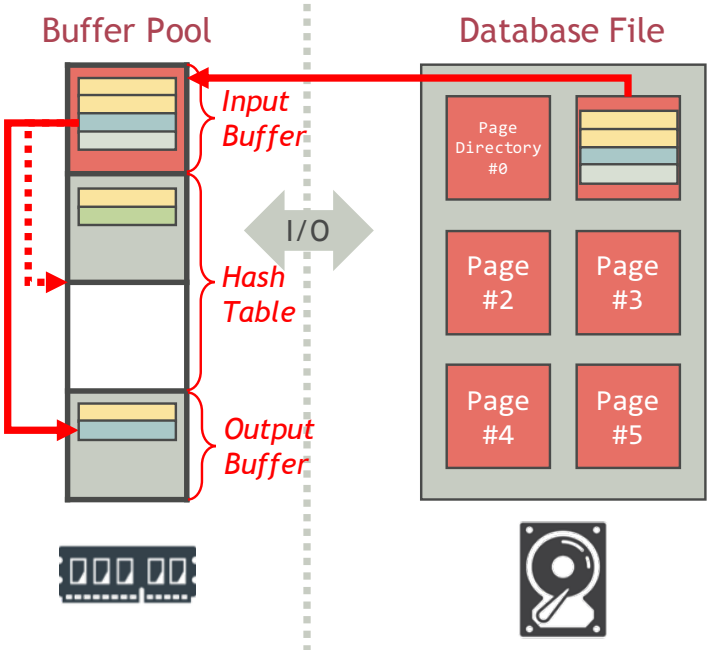
- 在对R创建归并段时，R的每块只读1次，合计$B(R)$次I/O
- 将R的归并段全部写入文件，需$B(R)$次I/O
- 在对S创建归并段时，S的每块只读1次，合计$B(S)$次I/O
- 将S的归并段全部写入文件，需$B(S)$次I/O
- 在归并阶段，对R和S的每个归并段各扫描1次，合计$B(R)+B(S)$次I/O

可用内存页数要求: $B(R)+B(S)\leq M^2$

- R和S的每个归并段均不超过M块
- R和S共有不超过M个归并段

哈希连接(Hash Join)

如果一趟连接算法使用的内存查找结构是哈希表，则该算法称为哈希连接算法



Grace哈希连接(Grace Hash Join)

算法

1: // 哈希分桶
2: 将R的元组哈希到$M-1$个桶$R_1, R_2, \ldots, R_{M-1}$中(哈希键为$R.Y$)
3: 将S的元组哈希到$M-1$个桶$S_1, S_2, \ldots, S_{M-1}$中(哈希键为$S.Y$)
4: // 逐桶连接
5: **for** $i = 1, 2, \ldots, M-1$ **do**
6: 使用一趟连接(one-pass join)算法计算$R_i \bowtie S_i$，并将结果写入输出缓冲区

- R和S中相同的元组一定分别落入同号桶R_i和S_i中
- $R \bowtie S = \bigcup_{i=1}^{M-1}(R_i \bowtie S_i)$
- $(R_i \bowtie S_i) \cap (R_j \bowtie S_j) = \emptyset$ for $i \neq j$

算法分析

I/O代价: $3B(R) + 3B(S)$

- 在对R进行哈希分桶时，R的每块读1次，合计$B(R)$次I/O
- 将R的桶全部写入文件，需$\sum_{i=1}^{M-1} B(R_i) \approx B(R)$次I/O
- 在对S进行哈希分桶时，S的每块读1次，合计$B(S)$次I/O
- 将S的桶全部写入文件，需$\sum_{i=1}^{M-1} B(S_i) \approx B(S)$次I/O
- 使用一趟连接算法计算$R_i \bowtie S_i$的I/O代价是$B(R_i) + B(S_i)$

可用内存页数要求: $B(S) \leq (M-1)^2$

- S共有$M-1$个桶
- S的每个桶不超过$M-1$块

基于索引的连接(Index-based Join)

假设: 关系S上建有属性Y的索引

算法

1: **for** R的每一块P **do**
2: 将P读入缓冲池
3: **for** P中每条元组r **do**
4: 在索引上查找键值等于$r.Y$的S的元组集合T
5: **for** $s \in T$ **do**
6: 连接r和s，并将结果写入输出缓冲区

算法运行实例

Example (基于索引的连接)

- $R(X, Y) = \{(1,1),(5,5),(3,2),(3,1),(2,1),(4,2)\}$
- $S(Y, Z) = \{(2,6),(1,7),(1,8),(2,5),(2,7)\}$
- $M = 2$
- $B = 2$

Order	Buffer	Output
1	$(1,1),(5,5)$	$(1,1,7),(1,1,8)$
2	$(3,2),(3,1)$	$(3,2,6),(3,2,5),(3,2,7),(3,1,7),(3,1,8)$
3	$(2,1),(4,2)$	$(2,1,7),(2,1,8),(4,2,6),(4,2,5),(4,2,7)$

算法分析

I/O代价: $B(R) + \frac{T(R)T(S)}{V(S,Y)}$ (若索引是非聚簇索引)

- R的每块只读1次，合计$B(R)$次I/O
- 对于R的每个元组r，S中平均约有$T(S)/V(S,Y)$个元组能与r连接
- 因为索引是非聚簇索引，这些元组在文件中不一定连续存储。最坏情况下，读每个元组产生1次I/O，合计$\frac{T(R)T(S)}{V(S,Y)}$次I/O

I/O代价: $B(R) + T(R)\lceil\frac{B(S)}{V(S,Y)}\rceil$ (若索引是聚簇索引)

- 因为索引是非聚簇索引，所以对于R的每个元组r，S中能与r连接的元组一定连续存储于S的文件中，约占$\lceil\frac{B(S)}{V(S,Y)}\rceil$个块

可用内存页数要求: $M \geq 2$

- 1页作为读R缓冲区
- 1页作为读索引节点缓冲区

Execution of Expressions

查询计划的执行方法

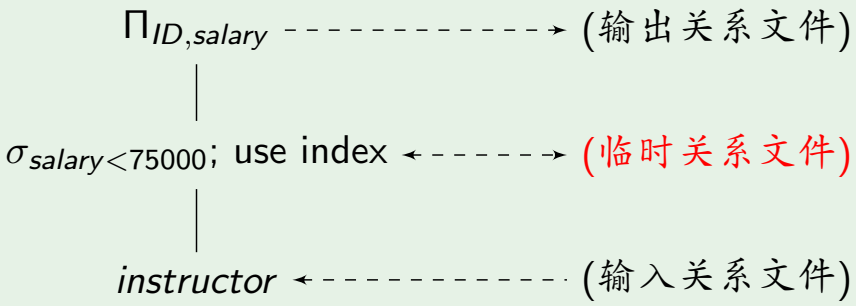
如何执行由多个操作构成的查询计划(query plan)?

- 方法1: 物化执行(Materialization)
- 方法2: 流水线执行(Piplining)，火山模型(The Volcano Model)

物化执行(Materialization)

- 自底向上执行查询计划中的操作
- 每个中间操作的执行结果写入临时关系文件，作为后续操作的输入

Example (物化执行)



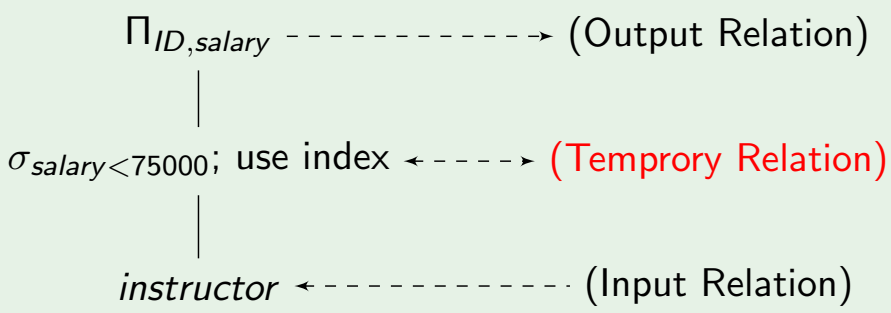
物化执行的缺点

缺点1: 物化(materialize)临时关系增加了查询执行的代价

- 执行完一个操作后，临时关系必须写入文件(除非临时关系非常小)
- 执行后续操作时，临时关系文件再被读入缓冲区

缺点2: 用户获得查询结果的时间延迟大

Example (物化执行)



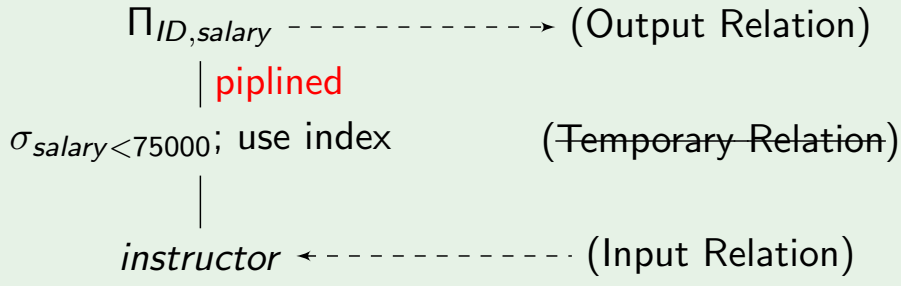
流水线执行(Piplining)/火山模型(The Volcano Model)

火山模型(The Volcano Model)将查询计划中若干操作组成流水线(pipline)，一个操作的结果直接传给流水线中下一个操作

- 避免产生一些临时关系，避免了读写这些临时关系文件的I/O开销
- 用户能够更快地得到查询结果

几乎所有DBMS都使用流水线执行查询计划

Example (流水线执行)



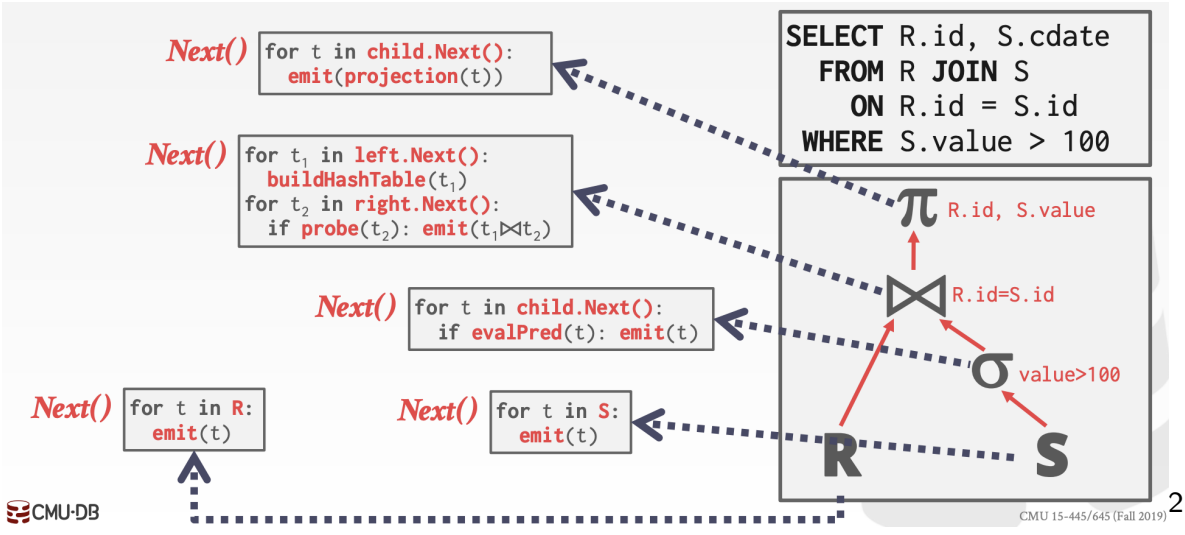
迭代器模型(Iterator Model)

使用迭代器(iterator)实现流水线中每个操作

- `open()`: 启动迭代器
- `next()`: 返回该操作的下一条结果元组
- `close()`: 关闭迭代器
- 迭代器需要维护其自身执行状态

迭代器模型(Iterator Model)

- DBMS不断调用查询计划中最顶层操作的`next()`函数
- 如果一个操作的输入是通过流水线获得的，那么在执行该操作的`next()`函数时将调用其输入操作的`next()`函数



[2] 来源: Andy Pevlo, CMU 15-445/645

总结

1. Overview
2. External Sort
   - External Merge Sort
3. Execution of Operations of Relational Algebra
   - Execution of Selection Operations
   - Execution of Projection Operations
   - Execution of Duplicate Elimination Operations
   - Execution of Aggregation Operations
   - Execution of Set Operations
   - Execution of Join Operations
4. Execution of Expressions

习题

1. Describe the *one-pass aggregation algorithm* and analyze its I/O cost and memory requirement
2. Describe the *hash-based aggregation algorithm* and analyze its I/O cost and memory requirement
3. Describe the *sort-based aggregation algorithm* and analyze its I/O cost and memory requirement
4. Write pseudocode for an iterator that implements a join algorithm (*one-pass join*, *block-based nested loop join*, *sort-merge join*, *hash join*, *index-based join*)