

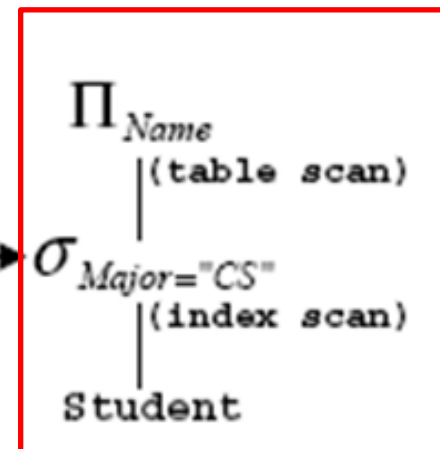
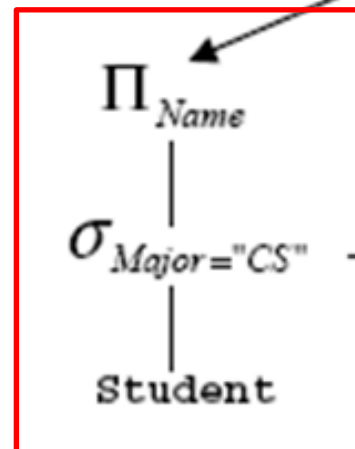
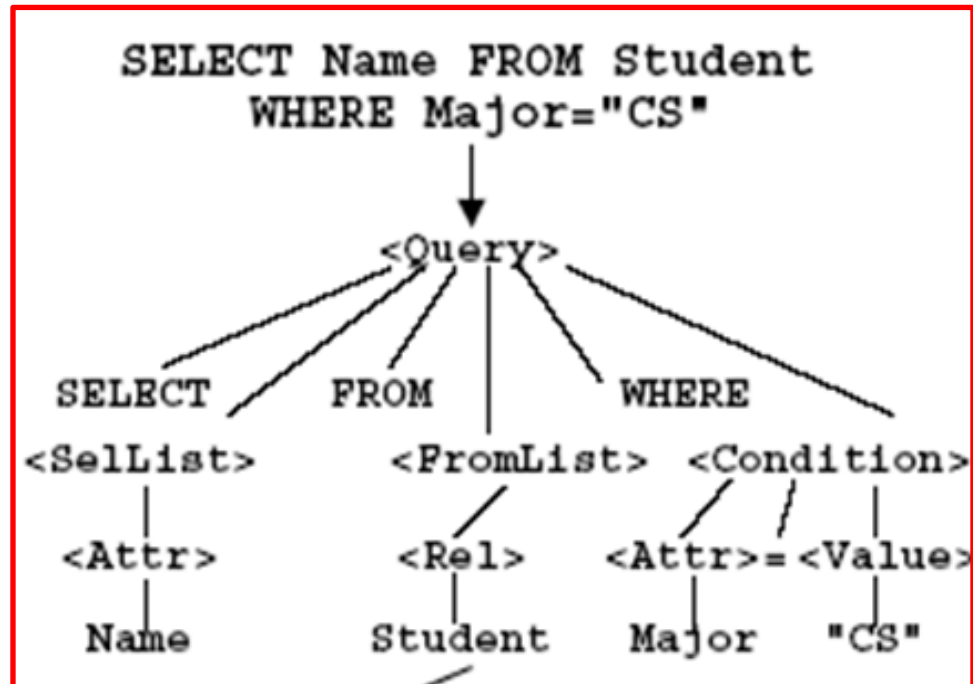
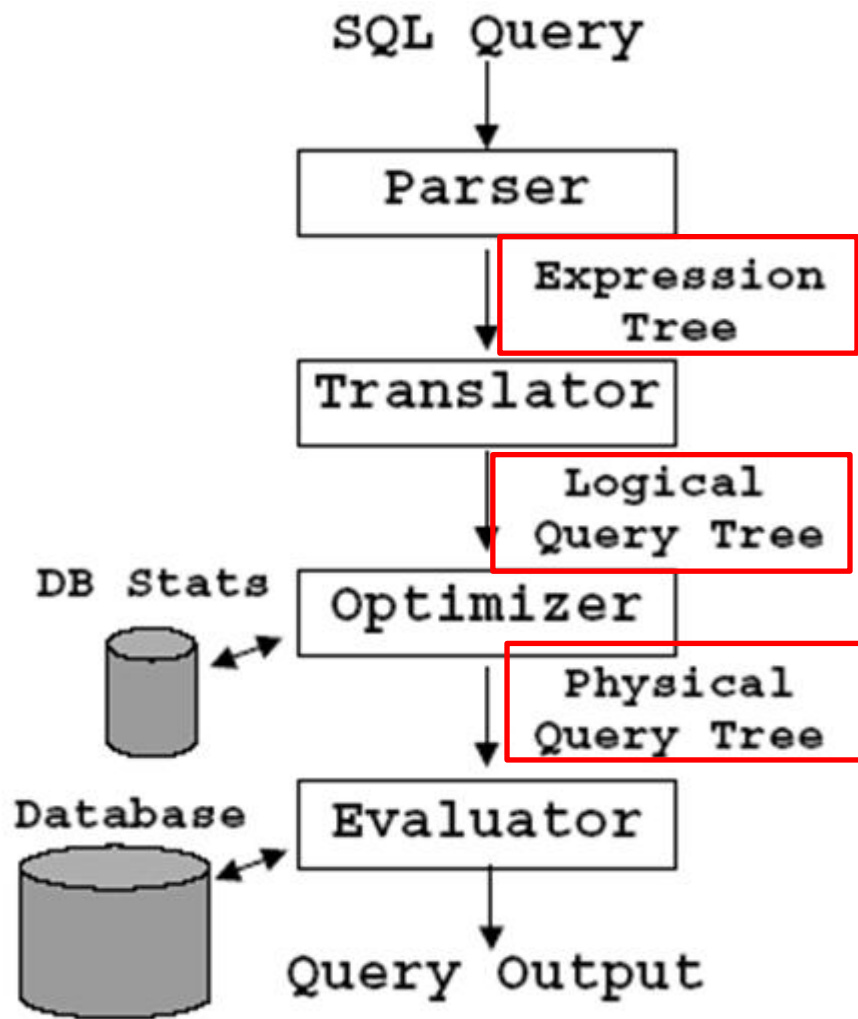
实现篇

第六章 查询处理与优化

主讲：王金宝

海量数据计算研究中心





- **Parser Tree (Expression Tree)**
 - 由select、from、where组成的语法树
- **Logical Query Plan Tree**
 - 由基本关系操作符组成的查询树
 - 如：选择、投影、连接等
- **Physical Query Plan Tree**
 - 由物理操作符组成的查询树
 - 物理操作符
 - 顺序扫描、索引扫描等
 - Nest-loop-join、Hash-join、sort-merge-join等



Example: SQL query

```
SELECT title
FROM StarsIn
WHERE starName IN (
    SELECT name
    FROM MovieStar
    WHERE birthdate LIKE '%1960' );
```

(找到1960年出生的影星参演的电影名字)



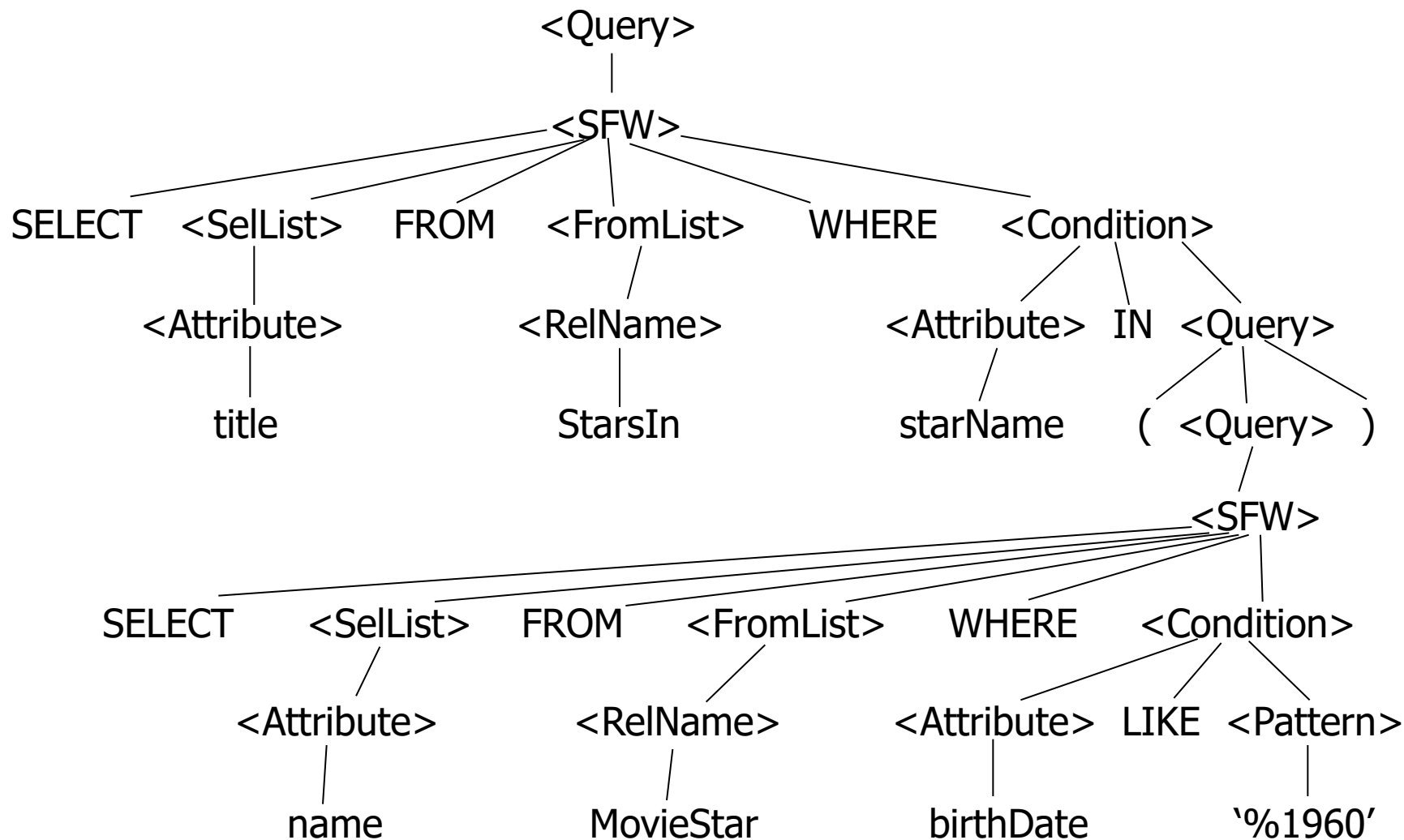


SQL的一个简单子集的语法

- <Query> ::= < SFW >**
- <Query> ::= (<Query>)**
- <SFW> ::= SELECT <SelList> FROM <FromList> WHERE <Condition>**
- <SelList> ::= <Attribute>, <SelList>**
- <SelList> ::= <Attribute>**
- <FromList> ::= <Relation>, <FromList>**
- <FromList> ::= <Relation>**
- <Condition> ::= <Condition> AND <Condition>**
- <Condition> ::= <Attribute> IN (<Query>)**
- <Condition> ::= <Attribute> = <Attribute>**
- <Condition> ::= <Attribute> LIKE <Pattern>**



Example: Parser Tree



Example: Generating Relational Algebra

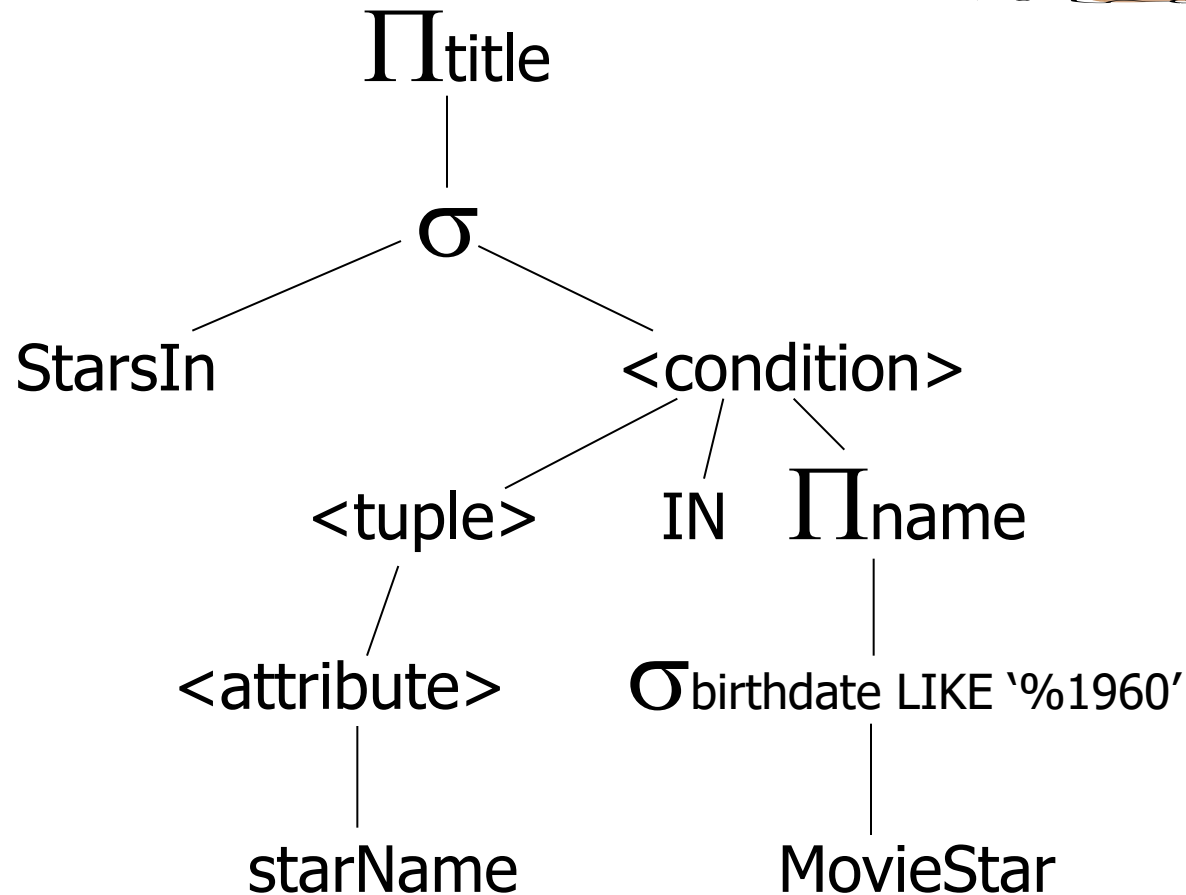


Fig. 7.15: An expression using a two-argument σ , midway between a parse tree and relational algebra



Example: Logical Query Plan

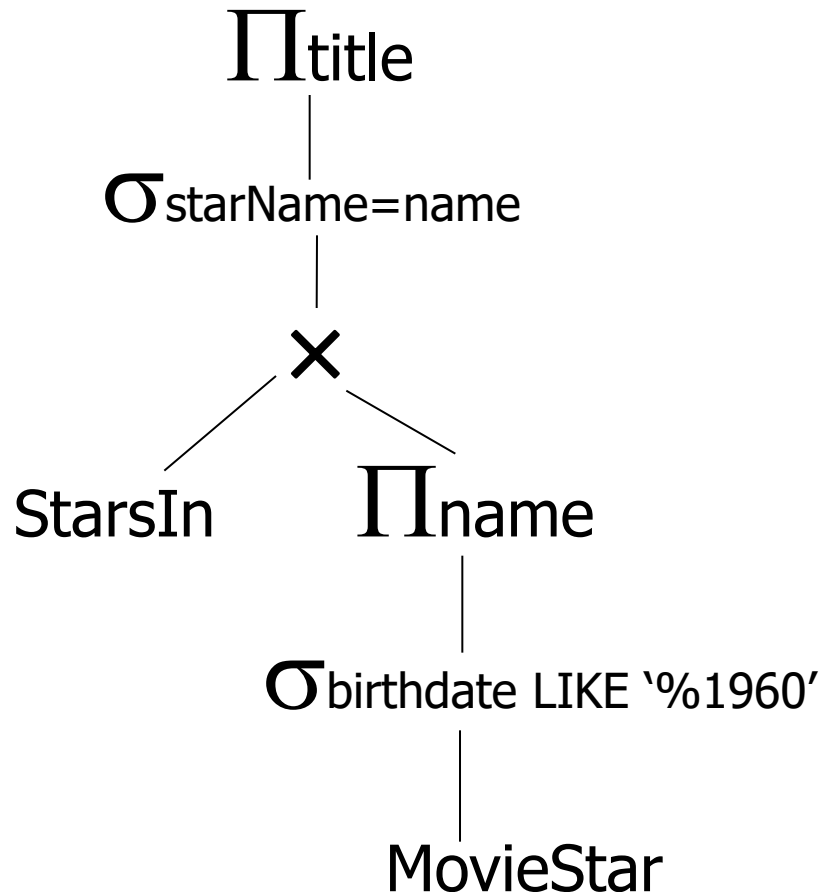


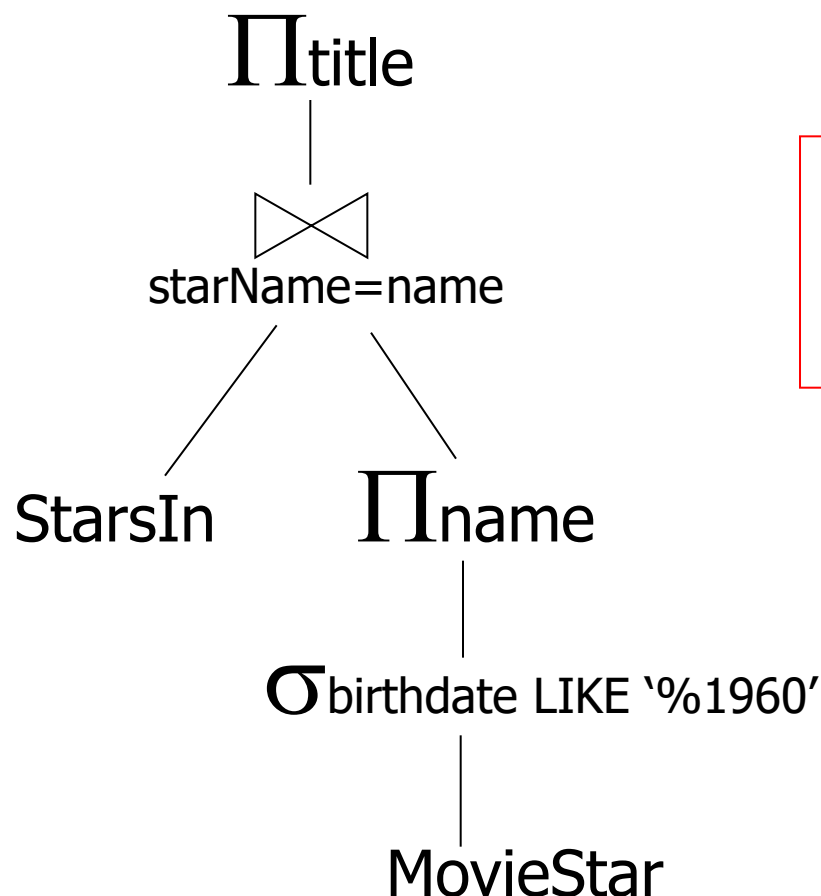
Fig. 7.18: Applying the rule for IN conditions



StarsIn(title, starName, year)

MovieStar(starName, birthdate, gender, nationality)

Example: Improved Logical Query Plan



Question:
Push project to
StarsIn?

Fig. 7.20: An improvement on fig. 7.18.



Example: Estimate Result Sizes

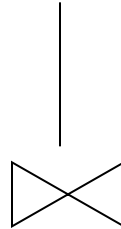
如何选择这个Join的物理操作符呢?

Nest Loop Join?

Sort Merge Join?

Hash Join?

StarsIn

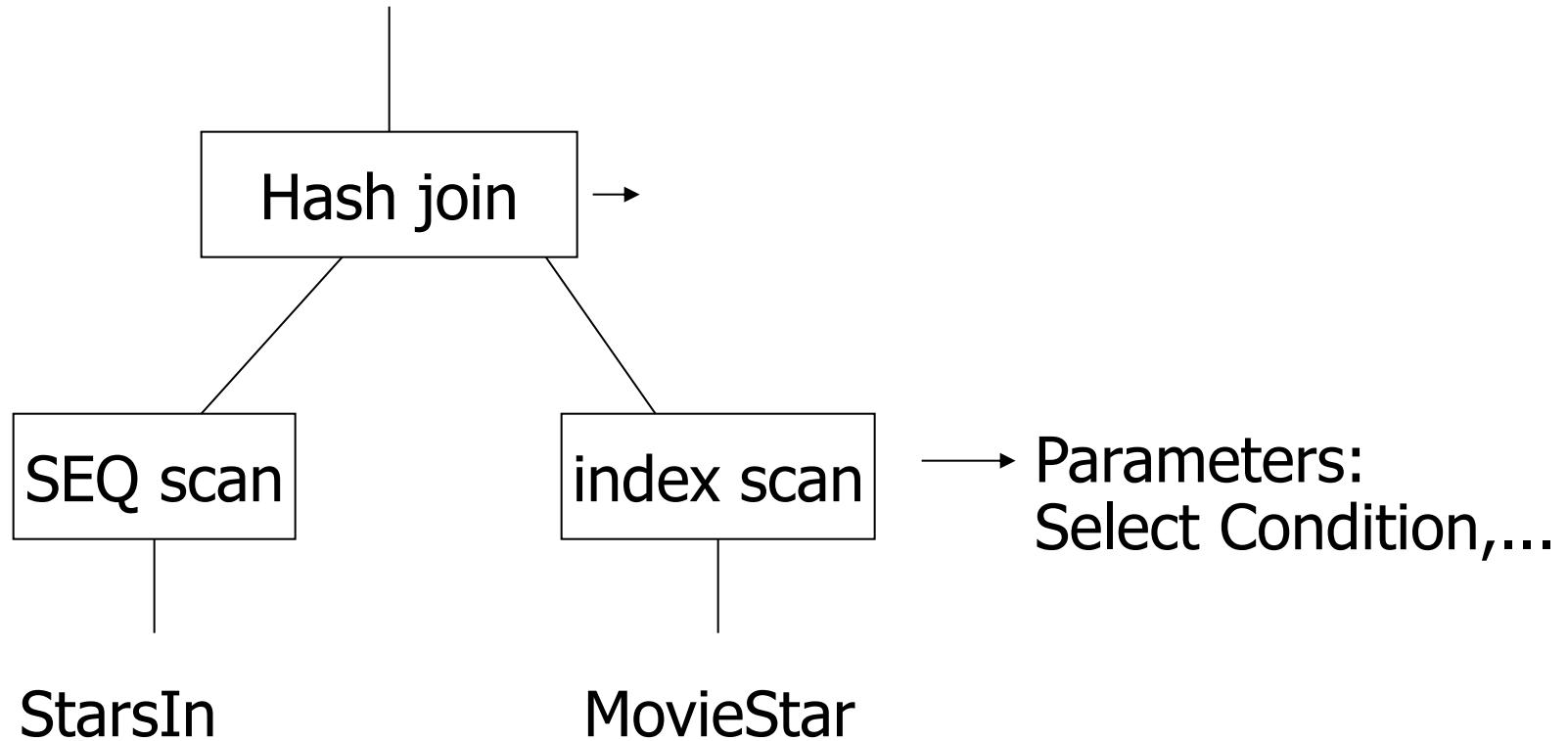


Need expected size

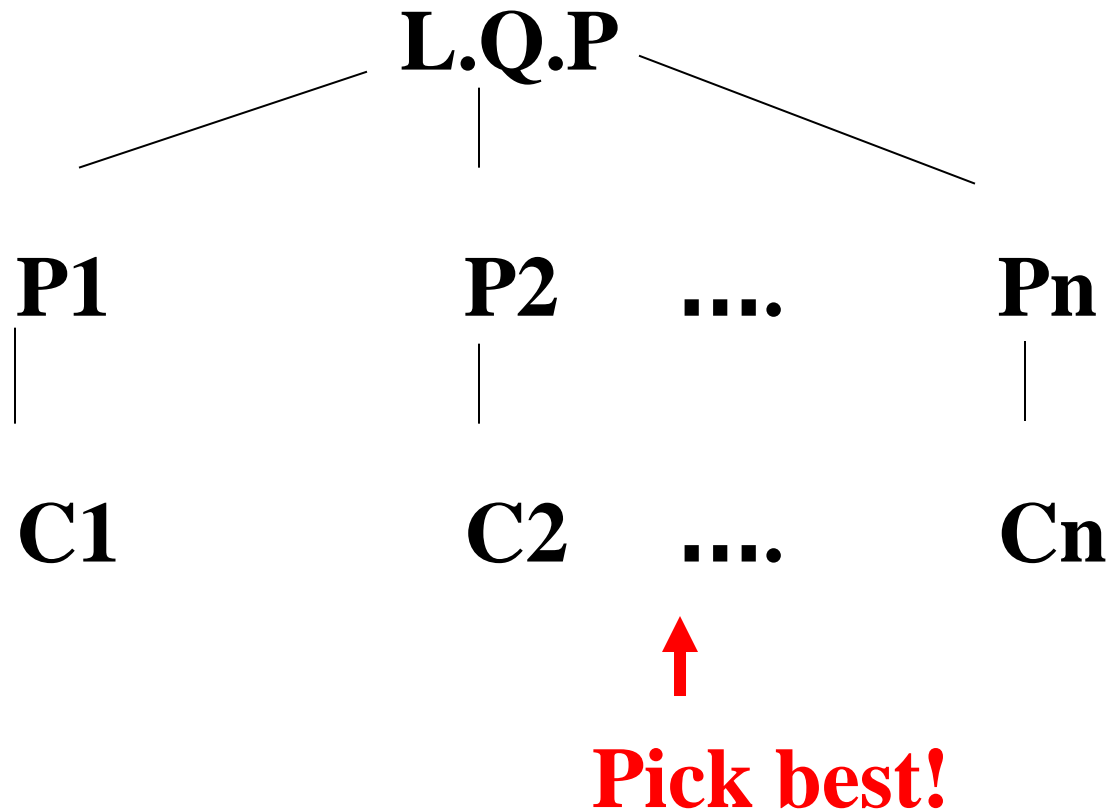
Π
 σ
MovieStar



Example: One Physical Plan



Example: Estimate costs



- 关系代数操作算法



6.1 关系代数操作算法

- 选择操作算法
- 投影操作算法
- 连接操作算法
- 集合操作算法



- 关系代数操作算法
 - 一趟算法、两趟算法、多趟算法
 - 基于排序、基于哈希、基于索引...
 - 选择、投影、分组聚集、连接、集合操作
 - 算法运行环境
 - $M+1$ 个缓冲区(输入和输出)+外存中存放的数据
 - 算法运行代价
 - 磁盘块存取数



外存中的Merge-Sort

- 文件R存储在 B_R 个磁盘块中, 排序R中的记录并写回磁盘, 可用缓冲区有 $M+1$ 个块
 - 第一阶段: 每次读入R的 M 个块, 在内存中排序 M 个块中的记录, 写入磁盘上的一个子表, 写操作使用1个缓冲区, 第一阶段结束时有 B_R/M 个有序子表



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中



输入缓冲区1



输入缓冲区2



输出缓冲区

输出:



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

1
3

输入缓冲区1

2
4

输入缓冲区2

输出缓冲区

输出：



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

3

输入缓冲区1

2
4

输入缓冲区2

1

输出缓冲区

输出：



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

3

输入缓冲区1

4

输入缓冲区2

1
2

输出缓冲区

输出：



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

3

输入缓冲区1

4

输入缓冲区2

输出缓冲区

输出:

1
2



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

输入缓冲区1

4

输入缓冲区2

3

输出缓冲区

输出:

1
2



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

4
9

输入缓冲区1

4

输入缓冲区2

3

输出缓冲区

输出:

1
2



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

9

输入缓冲区1

4

输入缓冲区2

3
4

输出缓冲区

输出:

1
2



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

9

输入缓冲区1

4

输入缓冲区2

输出缓冲区

输出:

1
2
3
4



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

9

输入缓冲区1

输入缓冲区2

4

输出缓冲区

输出:

1
2
3
4



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

9

输入缓冲区1

7
10

输入缓冲区2

4

输出缓冲区

输出:

1
2
3
4



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

9

输入缓冲区1

10

输入缓冲区2

4
7

输出缓冲区

输出:

1
2
3
4



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

9

输入缓冲区1

10

输入缓冲区2

输出缓冲区

输出:

1
2
3
4
4
7



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

输入缓冲区1

10

输入缓冲区2

9

输出缓冲区

输出:

1
2
3
4
4
7



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

输入缓冲区1

输入缓冲区2

9
10

输出缓冲区

输出:

1
2
3
4
4
7



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

有序列表1和2，分别
存储在两个磁盘块中

输入缓冲区1

输入缓冲区2

输出缓冲区

输出:

1
2
3
4
4
7
9
10



用2个输入缓冲区和1个输出缓冲区合并两个有序列表

1	2
3	4
4	7
9	10

删除有序列表1和2

输入缓冲区1

输入缓冲区2

输出缓冲区

输出:

1
2
3
4
4
7
9
10



外存中的Merge-Sort

- 文件R存储在 B_R 个磁盘块中, 排序R中的记录并写回磁盘, 可用缓冲区有 $M+1$ 个块
 - 第一阶段: 每次读入R的M个块, 在内存中排序M个块中的记录, 写入磁盘上的一个子表, 写操作使用1个缓冲区, 第一阶段结束时有 B_R/M 个有序子表
 - 第二阶段: 使用M个缓冲区块合并至多M个有序子表, 产生一个更长的有序子表, 所有子表参与一次合并后, 子表个数约为原有的 $1/M$
 - 最终: 经过多轮合并, 得到一个有序子表, 包含R中所有记录



每一趟I/O, 规模减小为 $1/M$, 规模减小到1, 排序结束.

外存中的Merge-Sort

$\log_M B_R$ 趟I/O, 每次 B_R 块

磁盘存取代价: $2\log_M B_R \times B_R$ (读+写)

- $B_R=8$, 每个磁盘块容纳2个记录, $M=2$

6	2
5	1
3	6
9	4
12	11
2	19
7	13
13	7

8个磁盘块



1	3	2	7
2	4	11	7
5	6	12	13
6	9	19	13

第一趟I/O: 得到4
个有序子表



1	2
2	7
3	7
4	11
5	12
6	13
6	13
9	19

第二趟I/O: 得到2
个有序子表



1
2
2
3
4
5
6
6
7
7
9
11
12
13
13
19

第三趟I/O:
得到1个有
序子表



建立外存中的哈希文件

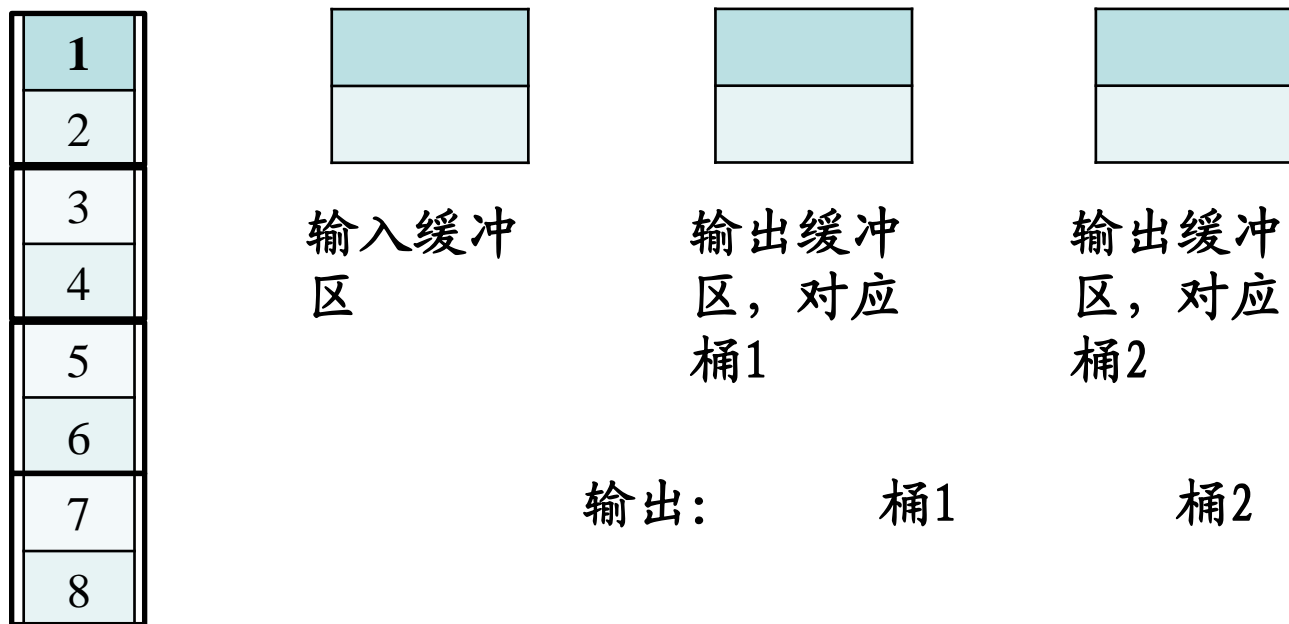
- 文件R存储在 B_R 个磁盘块中, 建立R的哈希文件并写回磁盘, 可用缓冲区有 $M+1$ 个块
 - 1个块用作输入缓冲区, M 个块用作 M 个桶的输出缓冲区
 - 依次读入R中的一个块B, 对其中每一个记录 t 使用哈希函数计算桶号 $h(t)$, 把 t 放入 $h(t)$ 号桶对应的缓冲区块, 若该缓冲区块满, 则写出到外存, 初始化该缓冲区为空, B中所有记录处理完毕后, 读入R的下一个块
 - R中没有下一块时, 结束

建立哈希文件需要一趟I/O, 哈希文件包含 M 个桶, 每个桶理想情况下包含磁盘块数为原文件的 $1/M$, 磁盘存取代价: $2B_R$

如果需要每个桶的大小不超过 M 个块, 则可以为每个桶继续建立哈希文件, 经过 $\log_M B_R - 1$ 次达到目的, I/O代价 $2B_R \times (\log_M B_R - 1)$



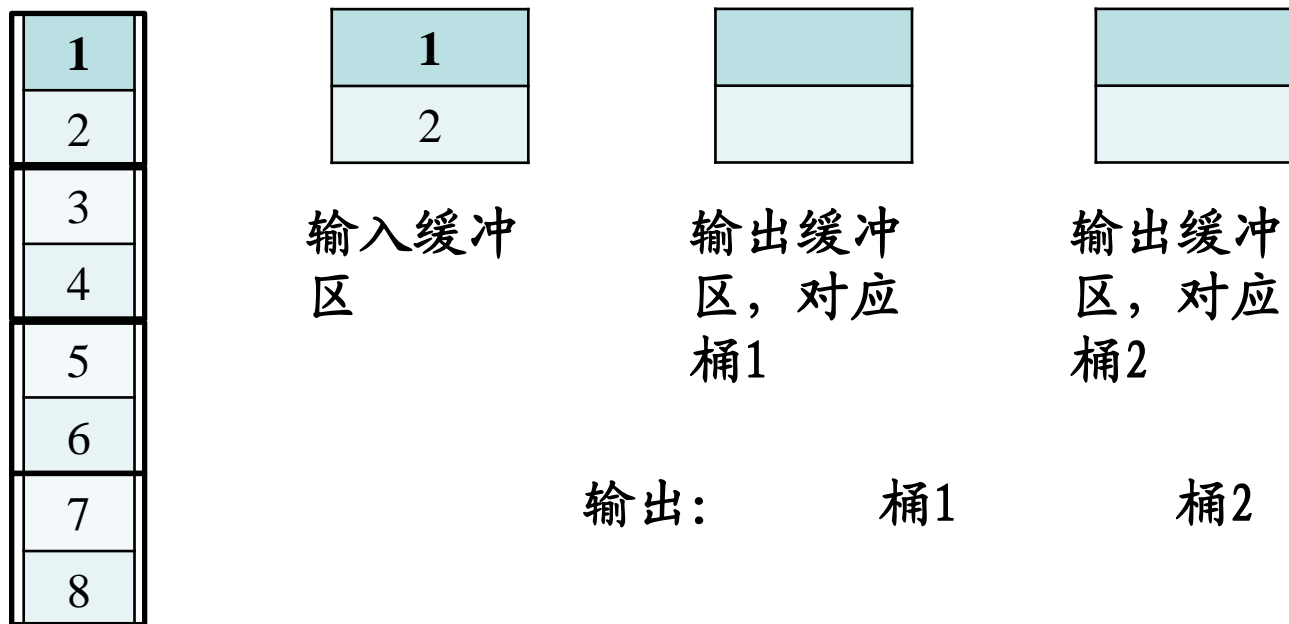
用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



输入文件，4个磁盘块，
每个磁盘块2个记录



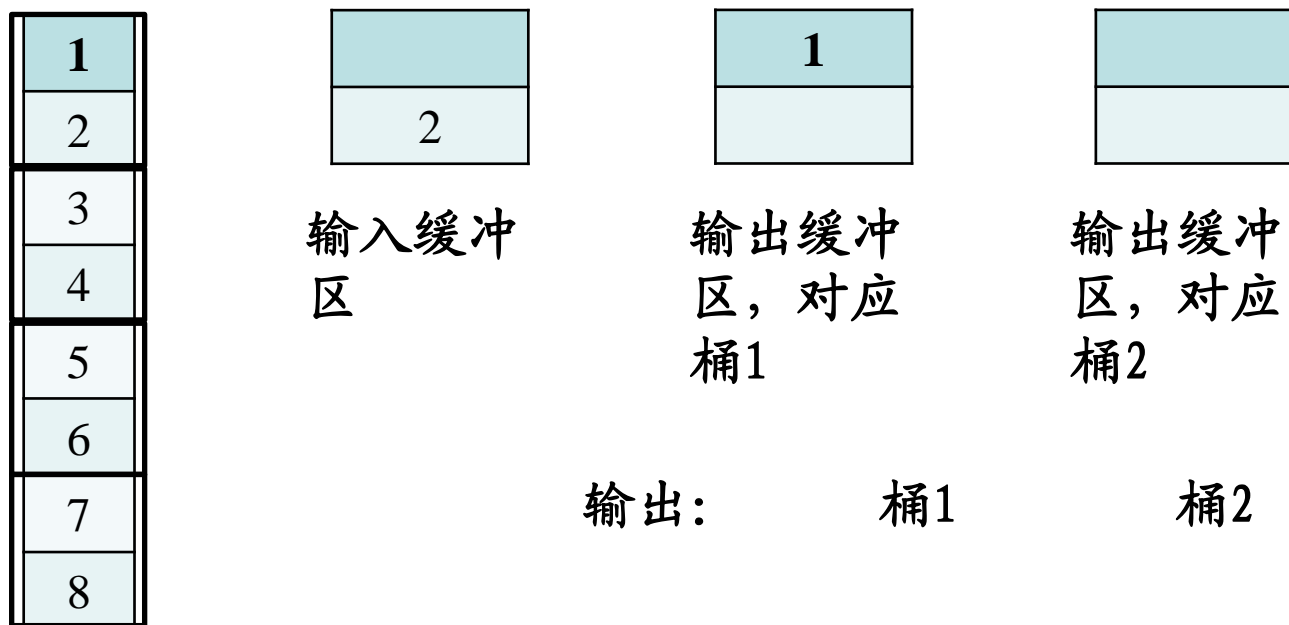
用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



输入文件，4个磁盘块，
每个磁盘块2个记录



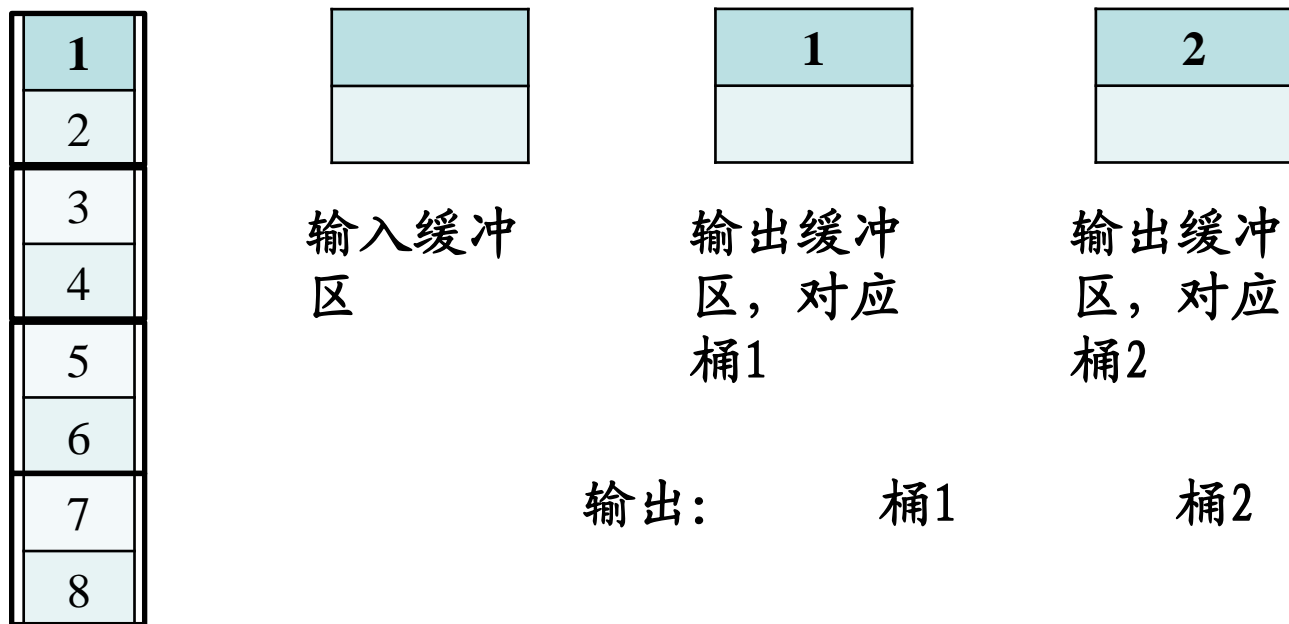
用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



输入文件，4个磁盘块，
每个磁盘块2个记录



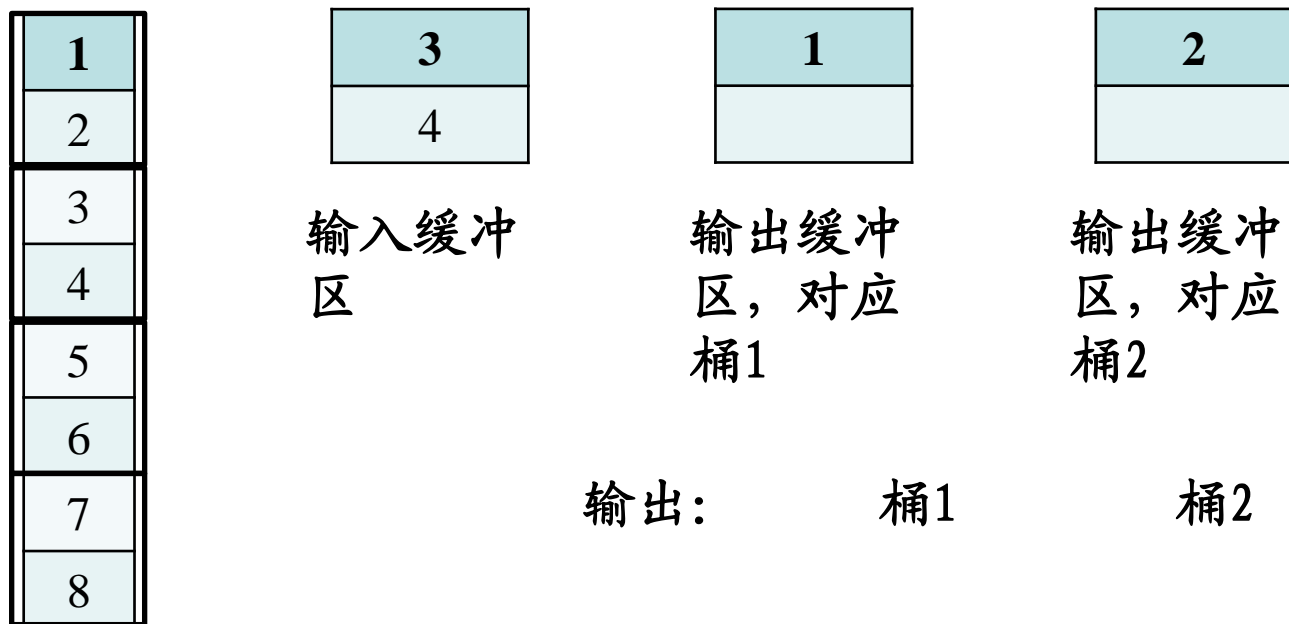
用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



输入文件，4个磁盘块，
每个磁盘块2个记录



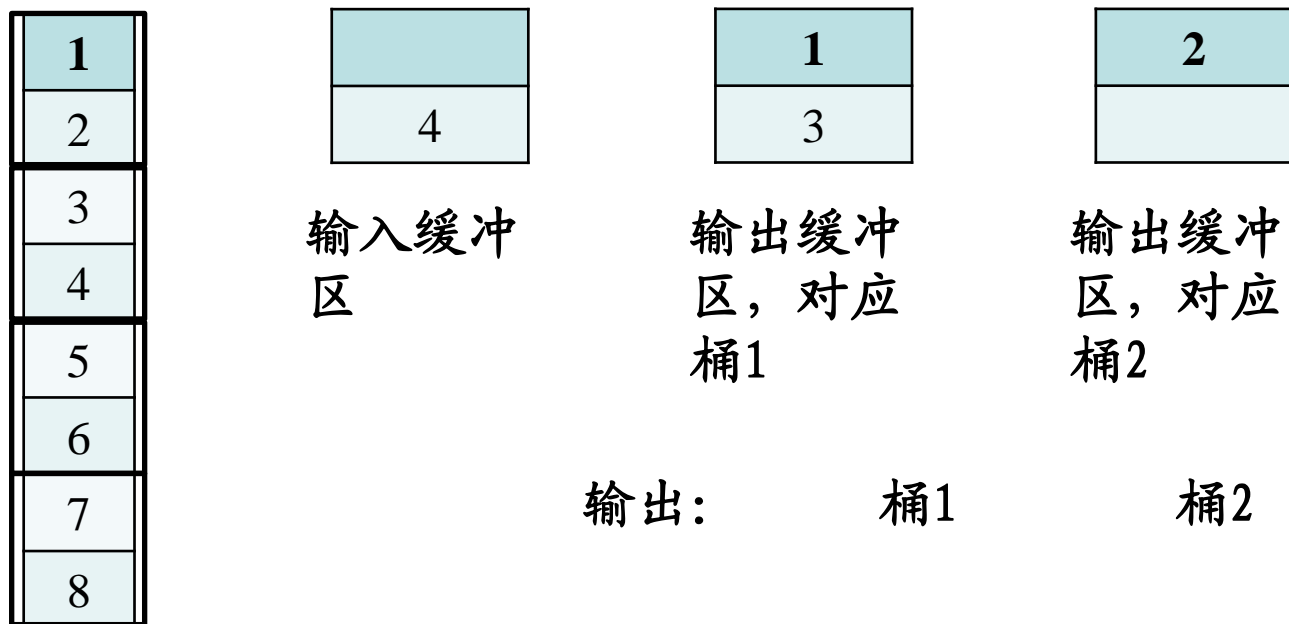
用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



输入文件，4个磁盘块，
每个磁盘块2个记录



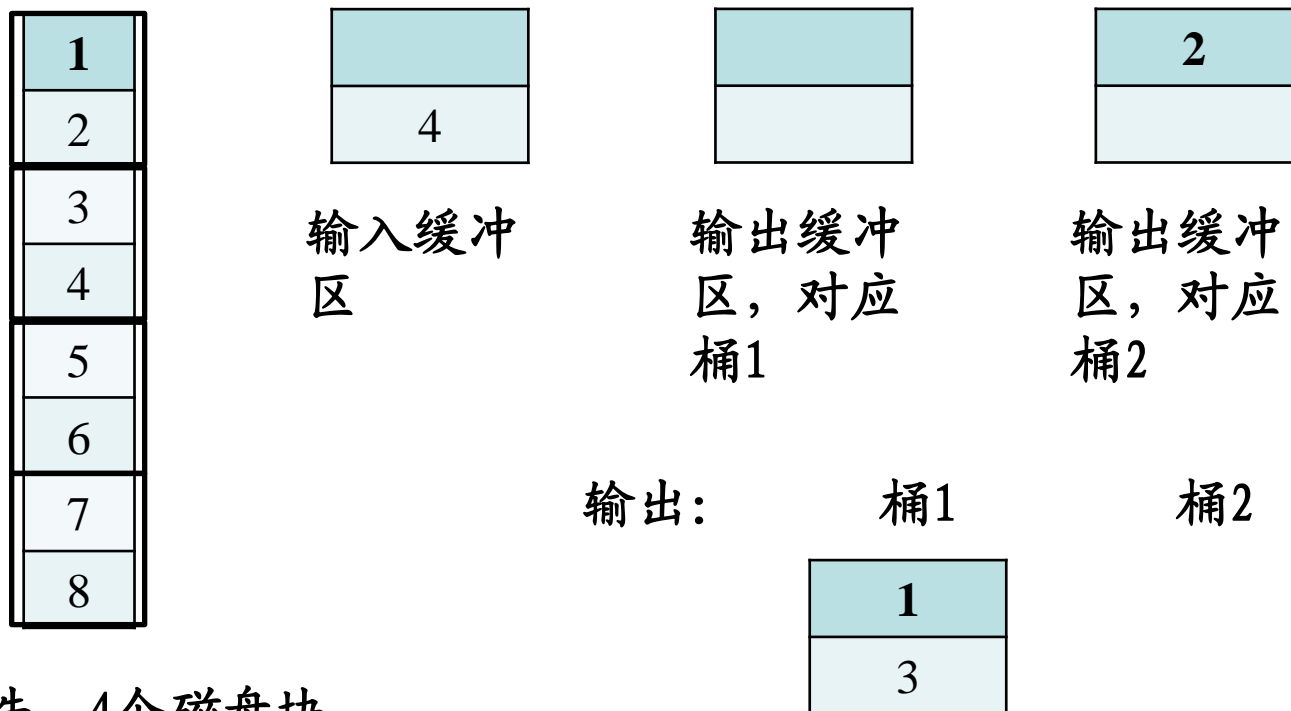
用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



输入文件，4个磁盘块，
每个磁盘块2个记录



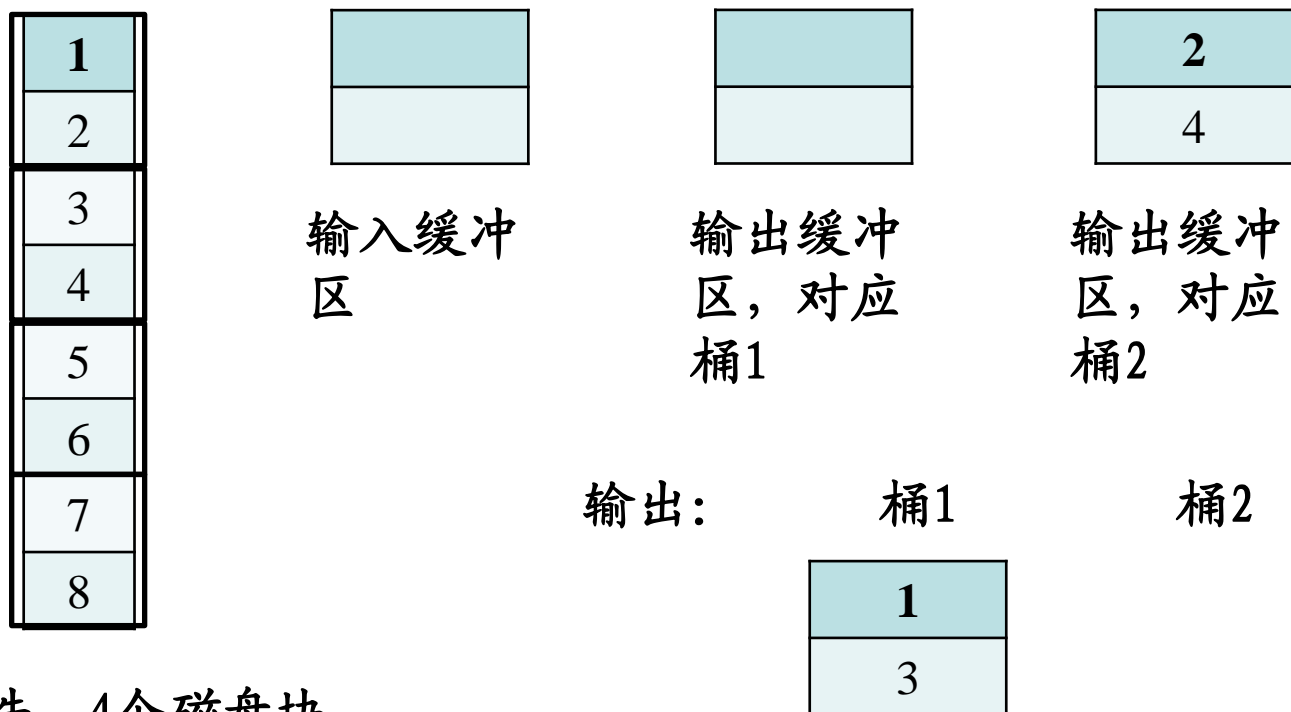
用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



输入文件，4个磁盘块，
每个磁盘块2个记录



用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



输入文件，4个磁盘块，
每个磁盘块2个记录



用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$

1
2
3
4
5
6
7
8

5
6

输入缓冲
区

输出缓冲
区，对应
桶1

输出缓冲
区，对应
桶2

输出：

桶1

桶2

1
3

2
4

输入文件，4个磁盘块，
每个磁盘块2个记录



用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$

1
2
3
4
5
6
7
8

6

输入缓冲
区

5

输出缓冲
区，对应
桶1

输出缓冲
区，对应
桶2

输出：

桶1

桶2

1
3

2
4

输入文件，4个磁盘块，
每个磁盘块2个记录



用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$

1
2
3
4
5
6
7
8

输入缓冲
区

5

输出缓冲
区，对应
桶1

6

输出缓冲
区，对应
桶2

输出：

桶1

桶2

1
3

2
4

输入文件，4个磁盘块，
每个磁盘块2个记录



用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$

1
2
3
4
5
6
7
8

7
8

输入缓冲
区

5

输出缓冲
区，对应
桶1

6

输出缓冲
区，对应
桶2

输出：

桶1

桶2

1
3

2
4

输入文件，4个磁盘块，
每个磁盘块2个记录



用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x) = x \% 2$

1
2
3
4
5
6
7
8

输入缓冲
区

8

输出缓冲
区，对应
桶1

5
7

输出缓冲
区，对应
桶2

6

输出：

桶1

桶2

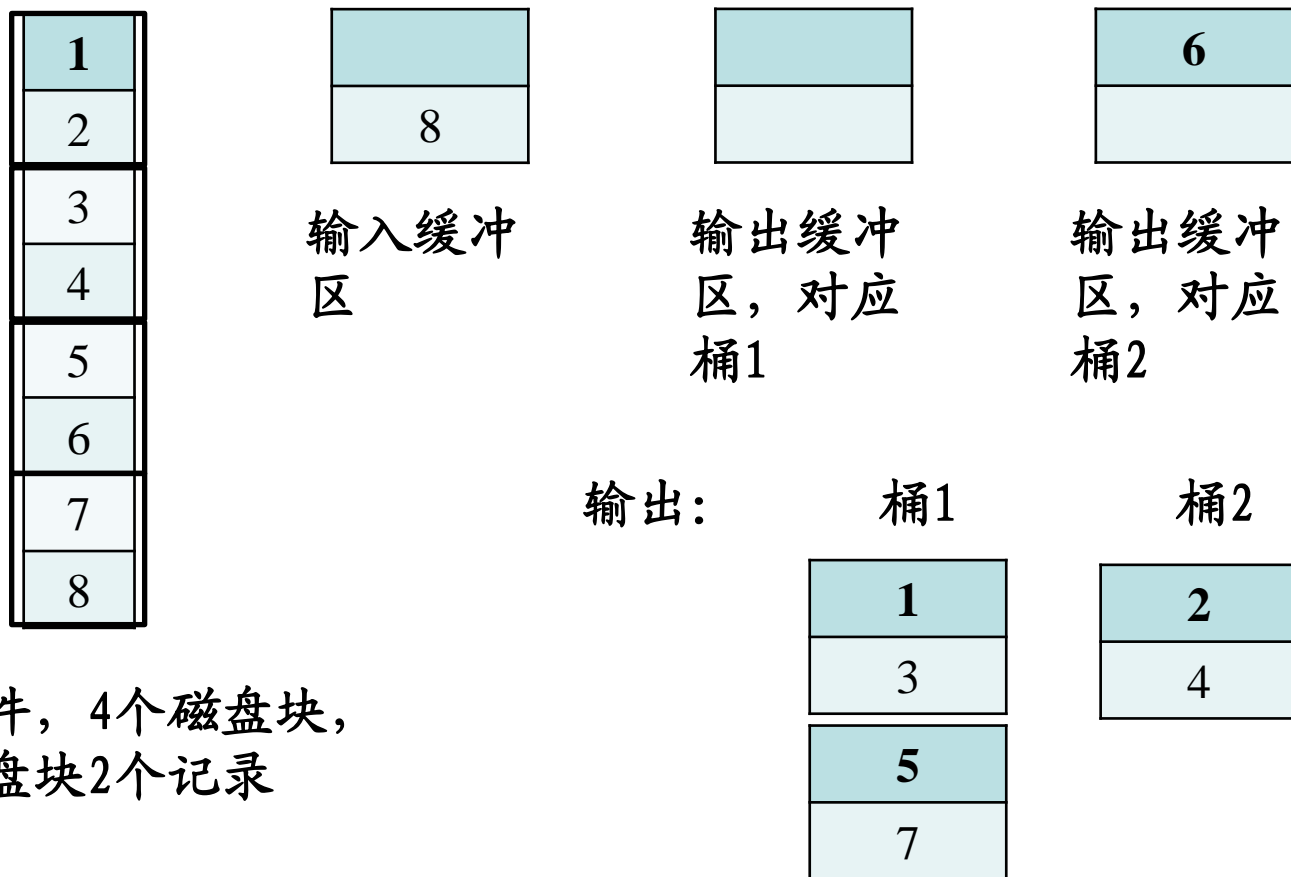
1
3

2
4

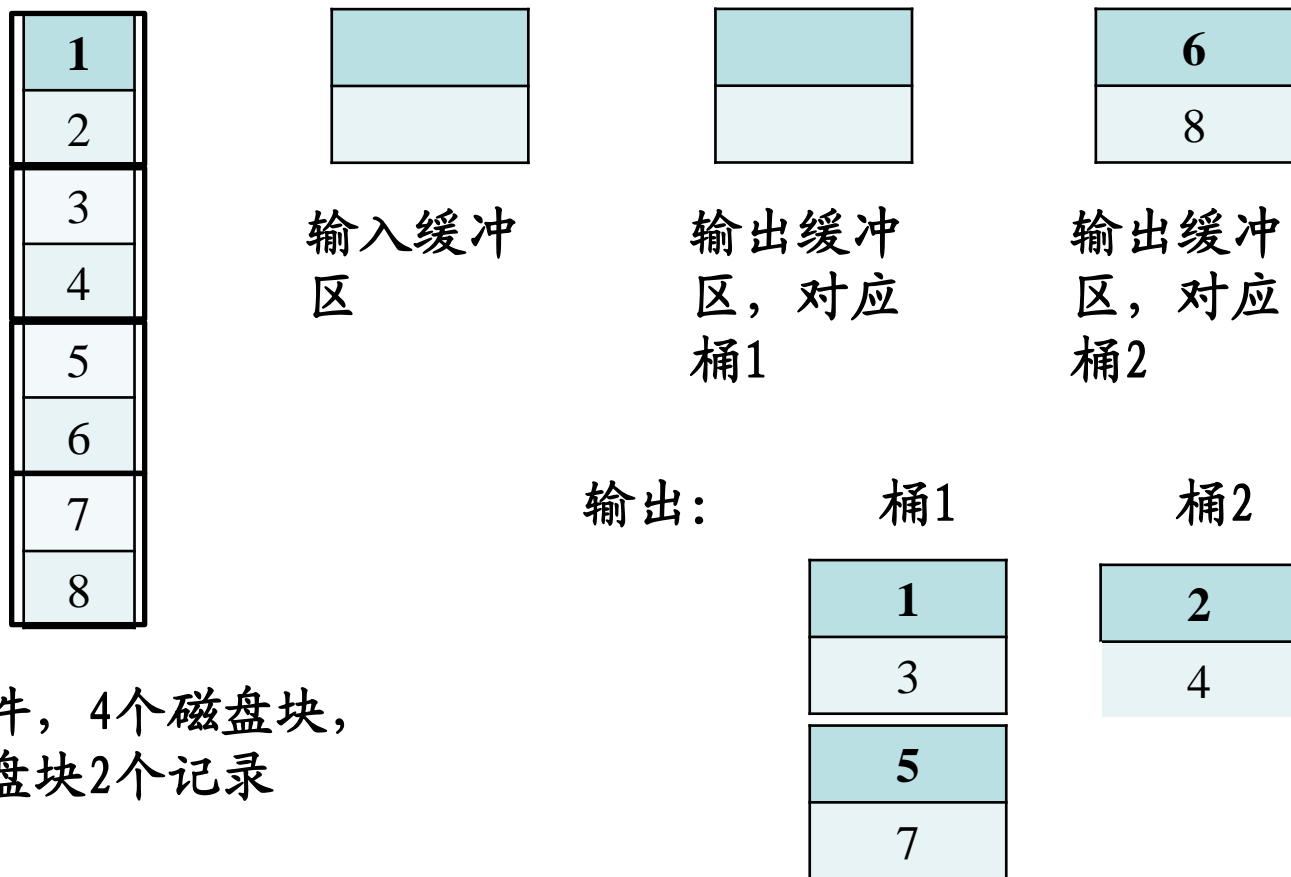
输入文件，4个磁盘块，
每个磁盘块2个记录



用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$



输入文件，4个磁盘块，
每个磁盘块2个记录



用1个输入缓冲区和2个输出缓冲区构建有2个桶的哈希文件，哈希函数为 $h(x)=x\%2$

1
2
3
4
5
6
7
8

输入文件，4个磁盘块，
每个磁盘块2个记录



输入缓冲
区



输出缓冲
区，对应
桶1



输出缓冲
区，对应
桶2

输出：

桶1


桶2

1
3
5
7

2
4
6
8

有两个桶的哈希文件，每
个桶占两个磁盘块



- 
- 选择操作算法
 - 投影操作算法
 - 连接操作算法
 - 集合操作算法



- 使用SQL语言，选择操作表示如下

```
SELECT *  
FROM R  
WHERE C1 AND C2 OR C3 ...
```

- 选择条件可以是简单条件(简单选择操作)
 - 仅包含关系R的一个属性的条件
- 选择条件也可以是复合条件(复杂选择操作)
 - 由简单条件经AND、OR、NOT等逻辑运算符连接而成的条件



简单选择操作算法

1. 线性搜索算法

- 顺序地读取被查询关系的每个元组；
- 测试该元组是否满足选择条件；
- 如果满足，则作为一个结果元组输出。

2. 二元搜索算法

- 条件: 某属性相等比较且关系按该属性排序。
- 对查询关系用二元搜索找到元组

如果关系具有 N 个磁盘块

二元搜索需要 $O(\log(N))$ 时间



3. 主索引或HASH搜索算法

- 条件: 主索引属性或Hash属性上的相等比较
- 使用主索引或HASH方法搜索操作关系.

4. 使用主索引查找满足条件的元组

- 条件: 主索引属性上的非相等比较
- 使用主索引选择满足条件的所有元组。

5. 使用聚集索引查找满足条件的元组

- 条件: 具有聚集索引的非键属性上相等比较
- 使用这个聚集索引读取所有满足条件的元组

6. B -树和 B^+ -树索引搜索算法

- 条件: B 树或 B^+ 树索引属性上相等或非相等比较
- 使用 B^+ 树索引搜索查找所有满足条件组

复杂选择操作算法

7. 合取选择算法

- 合取条件中存在简单条件 C
- C 涉及的属性上定义有某种存取方法
- 存取方法适应于上述六个算法之一
- 用相应算法搜索关系, 选择满足 C 的元组, 并检验是否满足其他条件, 若满足, 作为结果元组。

8. 使用复合索引的合取选择算法

- 如果合取条件定义在一组属性上的相等比较
- 而且存在一个由这组属性构成的复合索引
- 使用这个复合索引完成选择操作。



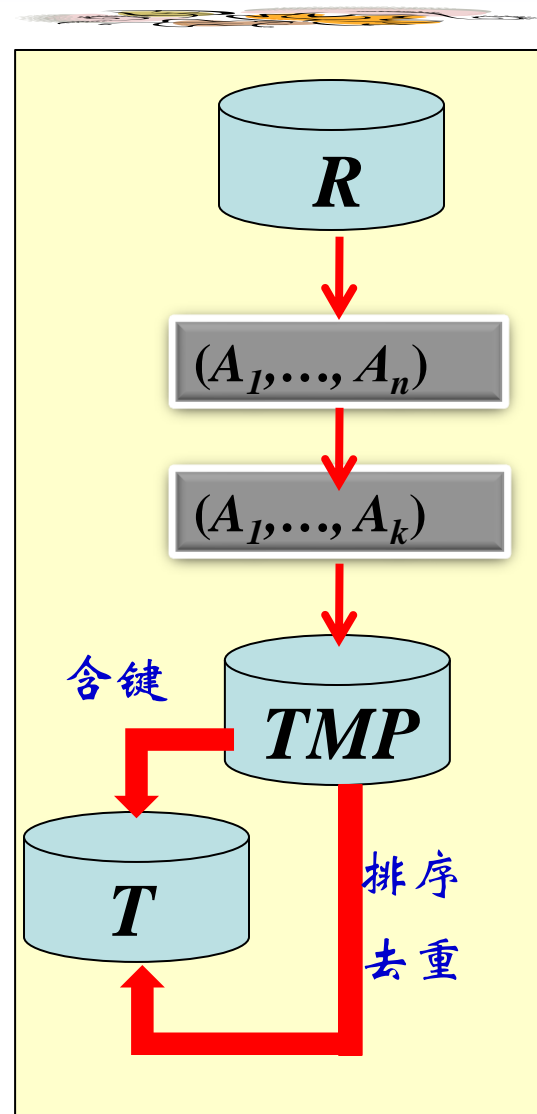


- 选择操作算法
- **投影操作算法**
- 连接操作算法
- 集合操作算法



投影操作的实现算法

- 设 $\Pi_{A_1, \dots, A_k}(R)$ 是 R 上的投影操作
 - 若 $\{A_1, \dots, A_k\}$ 中包括 R 的键
 - 存取 R 的所有元组一次即可完成;
 - 操作结果有与 R 同样个数元组, 每个元组仅包括 A_1, A_2, \dots, A_k 的值.
 - 如果投影属性表中不包含 R 的键
 - 需要删除操作结果中的重复元组
 - 可利用排序算法来实现投影操作



• 投影操作算法(基于排序的一趟算法)

输入: 具有 n 个元组的关系 R 。

输出: $T = \Pi_{A_1, \dots, A_k}(R)$ 。

FOR R 中每个元组 r DO

$r[A_1, \dots, A_k]$ 读入 TMP ;

IF $\{A_1, \dots, A_k\}$ 中包含 R 的键属性 THEN 将 TMP 写入 T ; 结束;

ELSE 排序 TMP ; $i=1$; $j=2$;

 WHILE ($i \leq n$) DO

 写 $TMP(i)$ 到 T ;

 WHILE ($TMP(i) = TMP(j)$) DO

$j=j+1$;

$i=j$; $j=j+1$;

• 投影操作算法(基于排序的两趟、多趟算法)

输入: 具有 n 个元组的关系 R , 缓冲区块数 $M+1$, 投影属性 A_1, \dots, A_k .

输出: $T = \Pi_{A_1, \dots, A_k}(R)$ 。

每次读入 R 的 M 个块进入缓冲区, 按 A_1, \dots, A_k 排序并将每个元组的 A_1, \dots, A_k 写入一个子表, 直至 R 中所有块处理完毕;

M 路归并排序子表直至子表个数首次不大于 M , 每次将 M 个子表归并成一个子表

读入每个子表的第一个块进入缓冲区(最多 M 个块)

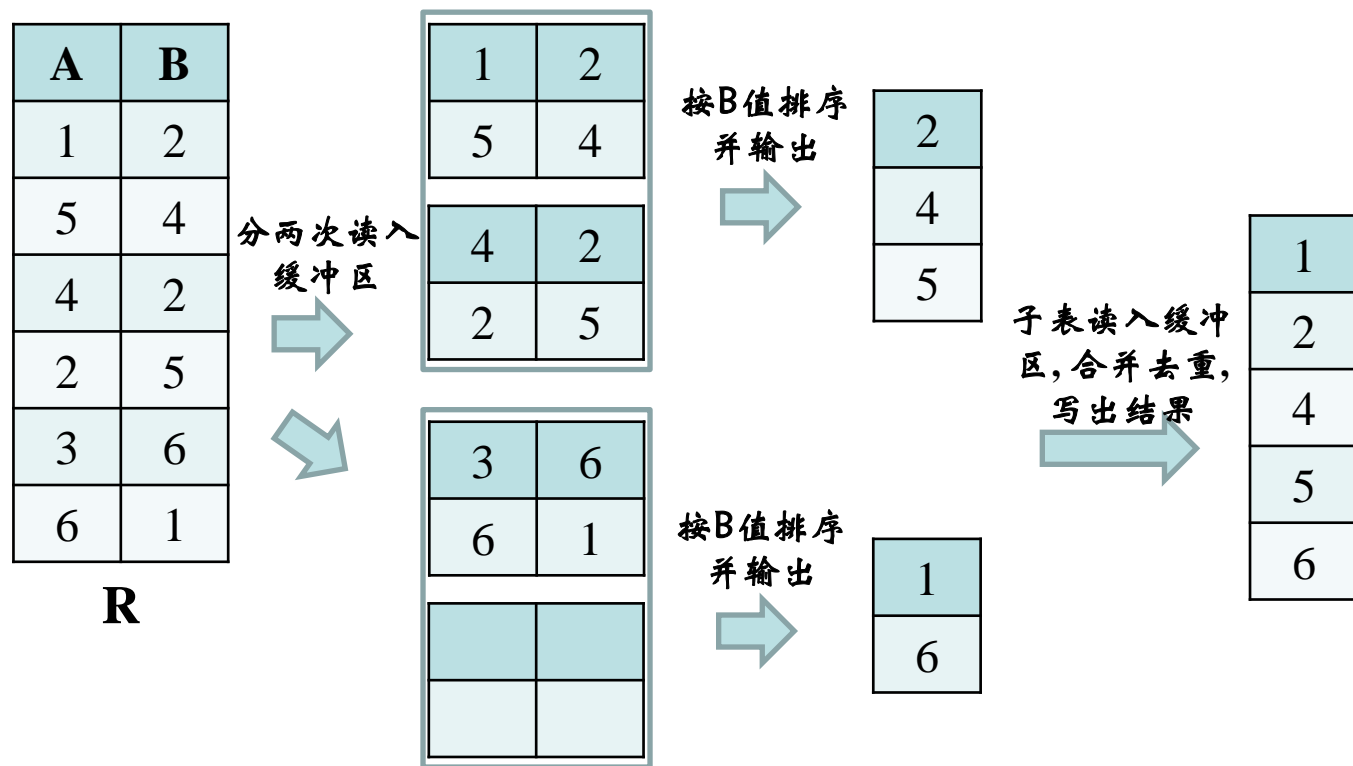
将最小元组 t 写入 T , 去除缓冲区中每个块内与 t 相同的元组, 若某个块为空, 读入相同子表的下一个块, 若相同子表无下一个块, 忽略该子表, 处理直至所有子表中的块都耗尽

基于排序的 k 趟投影操作算法:

磁盘存取代价: $2\log_M B_R \times B_R$

• 投影操作算法(基于排序的两趟、多趟算法)

计算 $\Pi_B(R)$ $M=2$, 每个块容纳2个元组





- 选择操作算法
- 投影操作算法
- **连接操作算法**
- 集合操作算法



等值连接操作算法



- 等值连接和自然连接是应用最多的连接操作，两者的操作算法无本质区别。
- 下边主要讨论自然连接
 - 循环嵌套连接(Nest-Loop-Join)算法
 - 排序合并连接(Sort-Merge-Join)算法
 - 哈希连接(Hash-Join)算法

注意：后续对连接算法代价分析中**没有加入**写出结果的代价



Nest-Loop- Join

输入: $R(A_1, \dots, A_i, \dots, A_n),$
 $S(B_1, \dots, B_j, \dots, B_m),$
连接条件 $R.A_i = S.B_j$

输出: R 与 S 的连接 T

FOR R 的每个磁盘块 X DO

读 X 到缓冲区 M_R ;

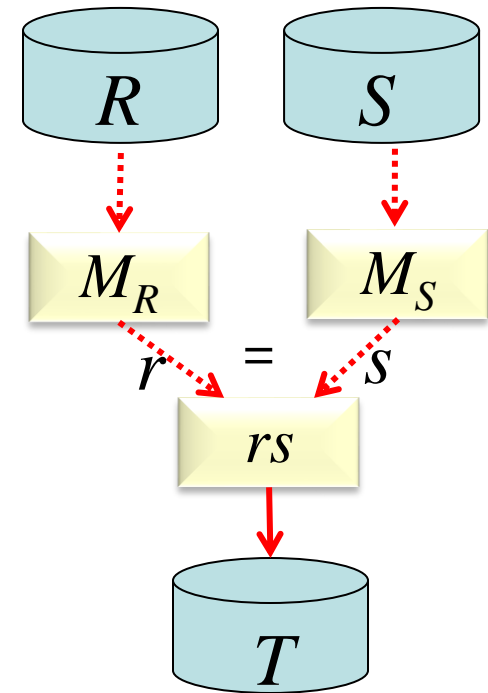
FOR S 的每个磁盘块 Y DO

读 Y 到缓冲区 M_S ;

FOR $\forall r \in M_R, \forall s \in M_S$ DO

IF $r[A_i] = s[B_j]$

THEN (rs) 存入缓冲区, 写入 T ;



如何优化?

使用3个缓冲区, 算法的磁盘存取块数: $B_R + B_R B_S$

• 优化

- 假定 $B(R) \leq B(S)$, $B(R) \geq M$
- 一次读入尽可能多的元组
- 使用尽可能多的 $(M-1)$ 内存块来存储属于关系 R 的元组, R 是外层循环中的关系。
- 性能分析:
 - 外层循环的迭代次数为 $B(R)/(M-1)$
 - 每一次迭代时, 读取 R 的 $M-1$ 个块, 和 S 的 $B(S)$ 个块
 - 这样, 磁盘 I/O 的数量为:

$$\frac{B(R)}{M-1} ((M-1) + B(S)) \quad \text{即:} \quad B(R) + \frac{B(S) \times B(R)}{M-1}$$





思考：

假定 $B(S) = 1000$ 且 $B(R) = 500$ ，缓冲区为102个块。我们将使用100个内存块来为R进行缓冲。因此算法中的外层循环需迭代5次。

每一次迭代中，在第二层循环内必须用1000个磁盘I/O来完整地读取S。因此，磁盘I/O的总数量是 $500 + 5 \times 1000 = 5500$ 。

若颠倒R和S的内外层嵌套关系，情况如何呢？

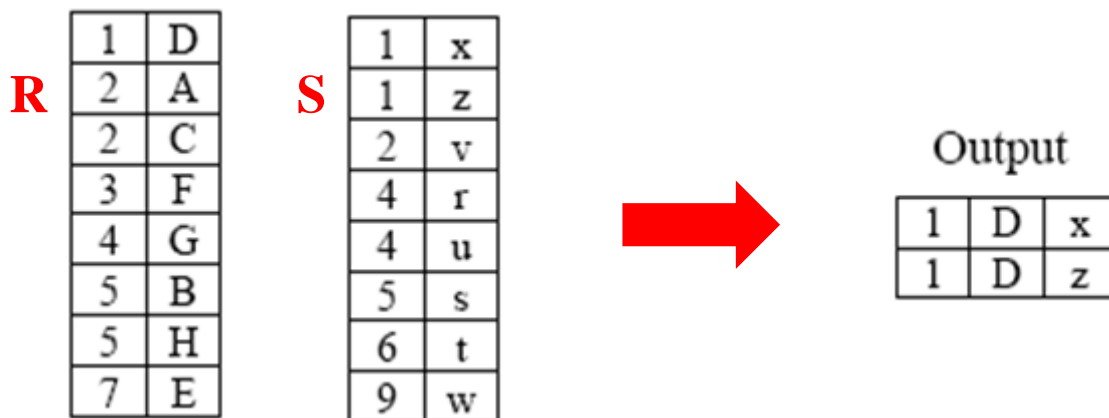
$$1000 + 10 \times 500 = 6000$$





Sort-Merge-Join

- 如果关系 R 和 S 的元组已经在连接属性 $R.A_i$ 和 $S.B_j$ 上物理地排序
 - 按排序顺序扫描 R 和 S , 查找在 $R.A_i$ 和 $S.B_j$ 上具有相同值的 R 和 S 的元组, 进行连接.



- 磁盘存取块数: 至少 $(B(R) + B(S))$





Sort-Merge-Join

- 如果关系 R 和 S 的元组都未排序
 - 分别按照连接属性 $R.A$ 和 $S.B$ 排序关系 R 、 S
 - 按排序顺序扫描 R 和 S , 查找在 $R.A_i$ 和 $S.B_j$ 上具有相同值的 R 和 S 的元组, 进行连接.



• Sort-Merge Join 算法

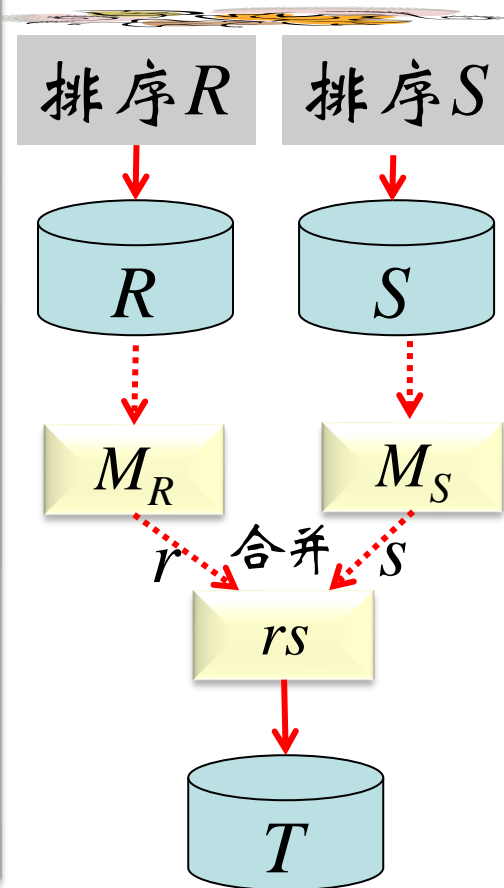
输入: $R(A_1, \dots, A_i, \dots, A_n)$,
 $S(B_1, \dots, B_j, \dots, B_m)$,
连接条件 $R.A_i = S.B_j$

输出: R 与 S 的连接 T

1. 按属性 $R.A_i$ 值排序 R ;
2. 按属性 $S.B_j$ 值排序 S ;
3. 扫描 R 和 S 一遍, 产生

$$T = \{R(k)S(m) \mid R(k)[A_i] = S(m)[B_j]\}.$$

假设缓冲区足够
容纳连接属性上
取值相同的元组
所在的块



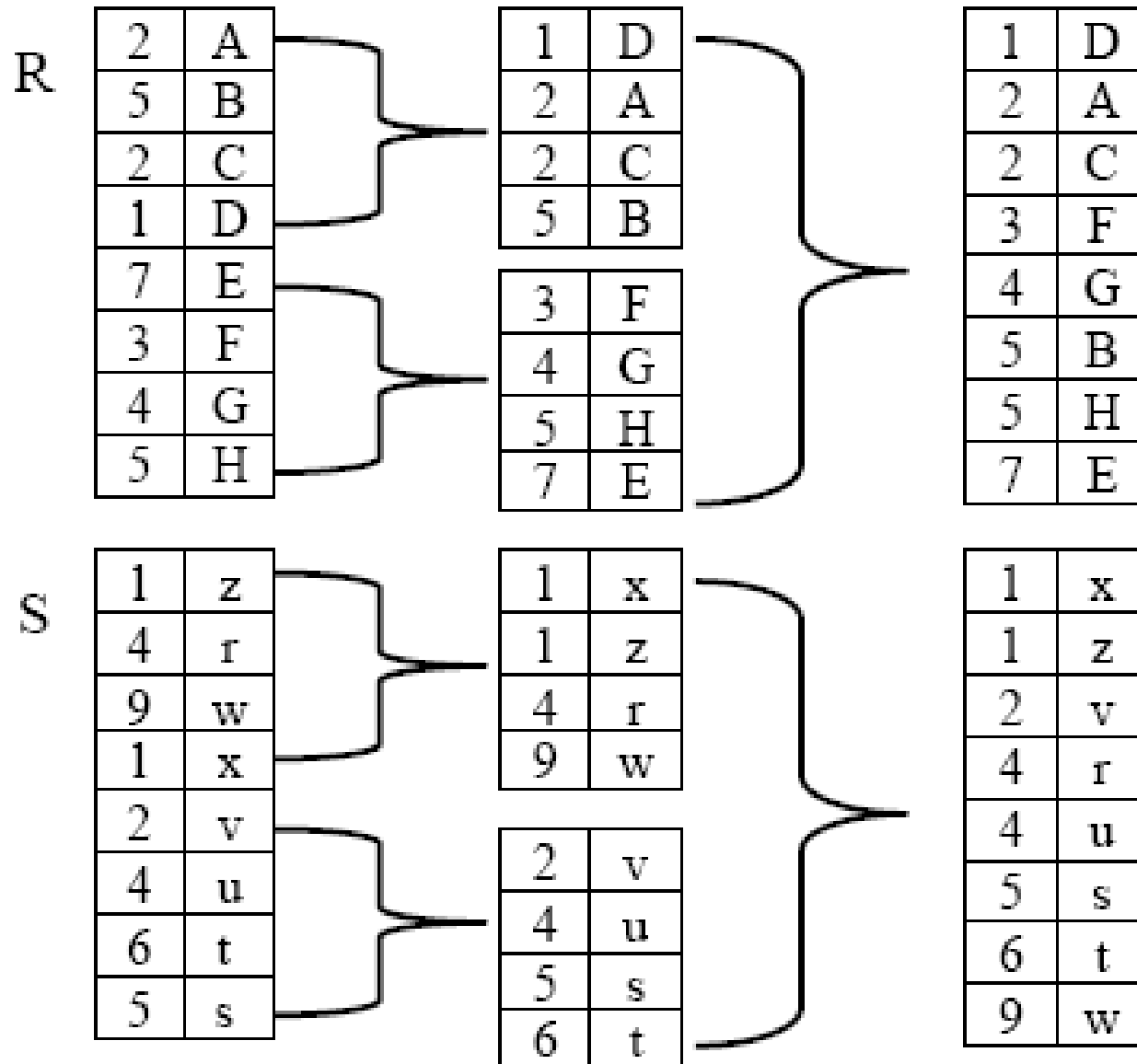
算法的磁盘存取块数:

$$2B(R) \log B(R) + 2B(S) \log B(S) + B(R) + B(S)$$

Sort-Join Example

Sort Phase


M=4. blocking factor=1.





Sort-Join Example

Merge Phase

M=4. blocking factor=1.

R	1	D		In memory after join on 1.
	2	A		
	2	C		
	3	F		
	4	G		
	5	B		
	5	H		
	7	E		

S	1	x		Brought in for join on 1.
	1	z		
	2	v		In memory after join on 1.
	4	r		
	4	u		
	5	s		
	6	t		
	9	w		

Buffer

1	D	R
1	x	S
1	z	extra
		extra

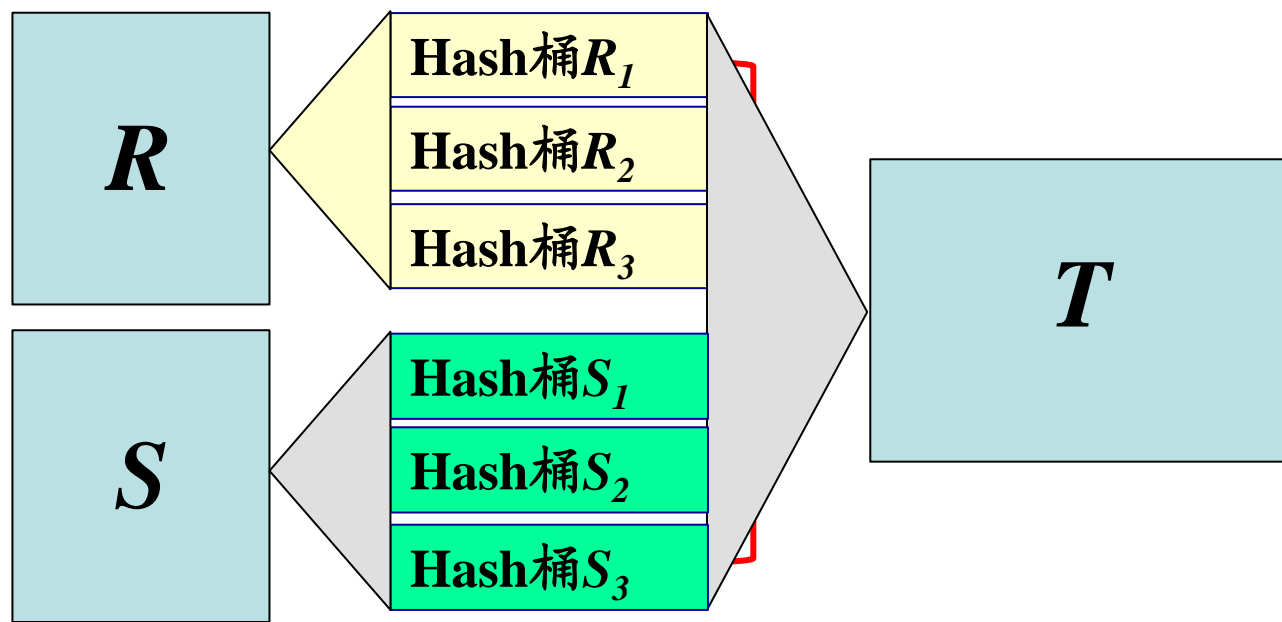
Output

1	D	x
1	D	z

Hash-Join

问题：为什么 R_1 不与 S_2 做连接？

- 第一阶段(Hash)
 - 扫描 R 和 S ，使用定义在连接属性上的Hash函数把 R 和 S 的元组分别构造成Hash文件 HR 和 HS ;
- 第二阶段(Probe)
 - 对于 HR 和 HS 的每对对应Hash桶，考察其中 R 和 S 的元组在连接属性上的值，产生 R 和 S 的连接结果。



Hash-Join算法(两趟算法)

输入: 关系 $R(A_1, \dots, A_i, \dots, A_n)$, $S(B_1, \dots, B_j, \dots, B_m)$, 连接条件 $R.A_i = S.B_j$, Hash函数 $h(x)$, 值域 $\{1, \dots, N\}$

输出: R 与 S 的连接 T

FOR 每个 $t \in R$ DO

t 写入 H_R 的第 $h(t[A_i])$ 个Hash桶;

ENDFOR

FOR 每个 $s \in S$ DO

s 写入 H_S 的第 $h(s[B_j])$ 个Hash桶;

ENDFOR;

FOR $i=1$ TO N DO

连接 H_R 和 H_S 的第 i 个Hash桶, 结果写入 T ;

ENDFOR 思考: H_R 和 H_S 的第 i 个桶可以一趟完成Join的条件是什么?

扩展到k趟算法:

设 $B_R \leq B_S$, 如果 R 的哈希桶块数大于 $M-1 \approx M$, 继续哈希, 直到 R 的每个桶不大于 $M-1 \approx M$ 为止

每次读入 R 的一个桶, 逐个读入 S 对应桶的每一个块, 在内存中计算当前桶的连接结果

完成所有桶的处理后算法结束

算法的磁盘存取块数:

$(2\log_M B_R - 1)(B_R + B_S)$

算法的磁盘存取块数: $3(B_R + B_S)$



Hash Join Example

Partition Phase

R	2	A
	5	B
	2	C
	1	D
	7	E
	3	F
	4	G
	5	H

S	1	z
	4	r
	9	w
	1	x
	2	v
	4	u
	6	t
	5	s

Partitions for R

$$h(x) = 0$$

3	F

$$h(x) = 1$$

1	D	4	G
7	E		

$$h(x) = 2$$

2	A	2	C
5	B	5	H

Partitions for S

$$h(x) = 0$$

9	w
6	t

$$h(x) = 1$$

1	z	1	x
4	r	4	u

$$h(x) = 2$$

2	v
5	s

$$M=4, \text{ bfr}=2, h(x) = x \% 3$$

Hash Join Example

Join Phase on Partition 1

Partition 1 for R

$h(x) = 1$

1	D	4	G
7	E		

Buffers

1	D
7	E

4	G

Output

1	D	x
1	D	z
4	G	r
4	G	u

Partition 1 for S

$h(x) = 1$

1	z	1	x
4	r	4	u

1	z
4	r

1	x
4	u

Note that both relations fit entirely in memory, but can perform join by having only one relation in memory and reading 1 block at a time from the other one.



- 三种连接算法的小结

- 输入关系R和S中磁盘块数分别为 B_R 和 B_S
- 缓冲区大小为 $M+1$ (不区分 M 和 $M-1$)

连接算法	算法代价	I/O趟数
Nest Loop Join	$\min\{B_R, B_S\} + \frac{B_R B_S}{M}$	\
Sort Merge Join	$2B_R \log_M B_R + 2B_S \log_M B_S + B_R + B_S$	$\log_M \max\{B_R, B_S\}$
Hash Join	$(2 \log_M \min\{B_R, B_S\} - 1)(B_R + B_S)$	$\log_M \min\{B_R, B_S\}$





- 选择操作算法
- 投影操作算法
- 连接操作算法
- **集合操作算法**





- 输入关系的约束
 - 具有相同的属性集合
 - 并且属性的排列顺序必须也相同
- 实现这些操作的常用算法
 - 首先利用排序算法在相同的键属性上排序两个操作关系；
 - 然后扫描这两个排序后的关系，完成并、交或差操作。



• 思考分组聚集操作算法

– M+1个缓冲区

- M个用于读入数据
- 1个附加的、用于输出的缓冲区

– 在多路归并排序的过程中完成分组聚集

- 第一轮生成排序子表时，输出数据中每个分组属性的值对应一个记录
 - 分组属性值+聚集值（中间结果）
- 聚集函数对应中间结果类型
 - Count、Sum、Max和Min对应自身
 - Avg对应Count和Sum
- 归并的时候更新每个分组属性的值对应的聚集结果
- 最后一轮归并生成最终聚集结果



- SQL查询处理过程

- 语法分析树
- 初始的逻辑查询计划、改进的逻辑查询计划
- 物理查询计划

- 关系代数操作算法

- 基于排序的算法
- 基于哈希的算法
- 基于嵌套循环算法





**Now let's go to
Next Chapter**

