

查询算子的实现



徐辰
华东师范大学
数据科学与工程学院
cxu@dase.ecnu.edu.cn

关系数据库的查询处理过程

- SQL \rightarrow Plans \rightarrow Best Plan \rightarrow Results

- SQL

```
SELECT Distinct C.name,C.type,I.ino  
FROM Customer C, Invoice I  
WHERE I.amount>10000 AND  
C.country="Sweden" AND  
C.cno=I.cno  
ORDER BY amount DESC
```

Plans

$$\Pi_{\text{name,type,ino}}(\sigma_{\text{amount}>10000 \wedge \text{country} = \text{'Sweden'}}(\text{Customer} \bowtie \text{Invoice}))$$
$$\Pi_{\text{name,type,ino}}(\sigma_{\text{amount}>10000}(\text{Invoice}) \bowtie \sigma_{\text{country} = \text{'Sweden'}}(\text{Customer}))$$
$$\Pi_{\text{name,type,ino}}(\sigma_{\text{amount}>10000}(\text{Invoice} \bowtie \sigma_{\text{country} = \text{'Sweden'}}(\text{Customer})))$$
$$\Pi_{\text{name,type,ino}}(\sigma_{\text{country} = \text{'Sweden'}}(\sigma_{\text{amount}>10000}(\text{Invoice}) \bowtie \text{Customer}))$$

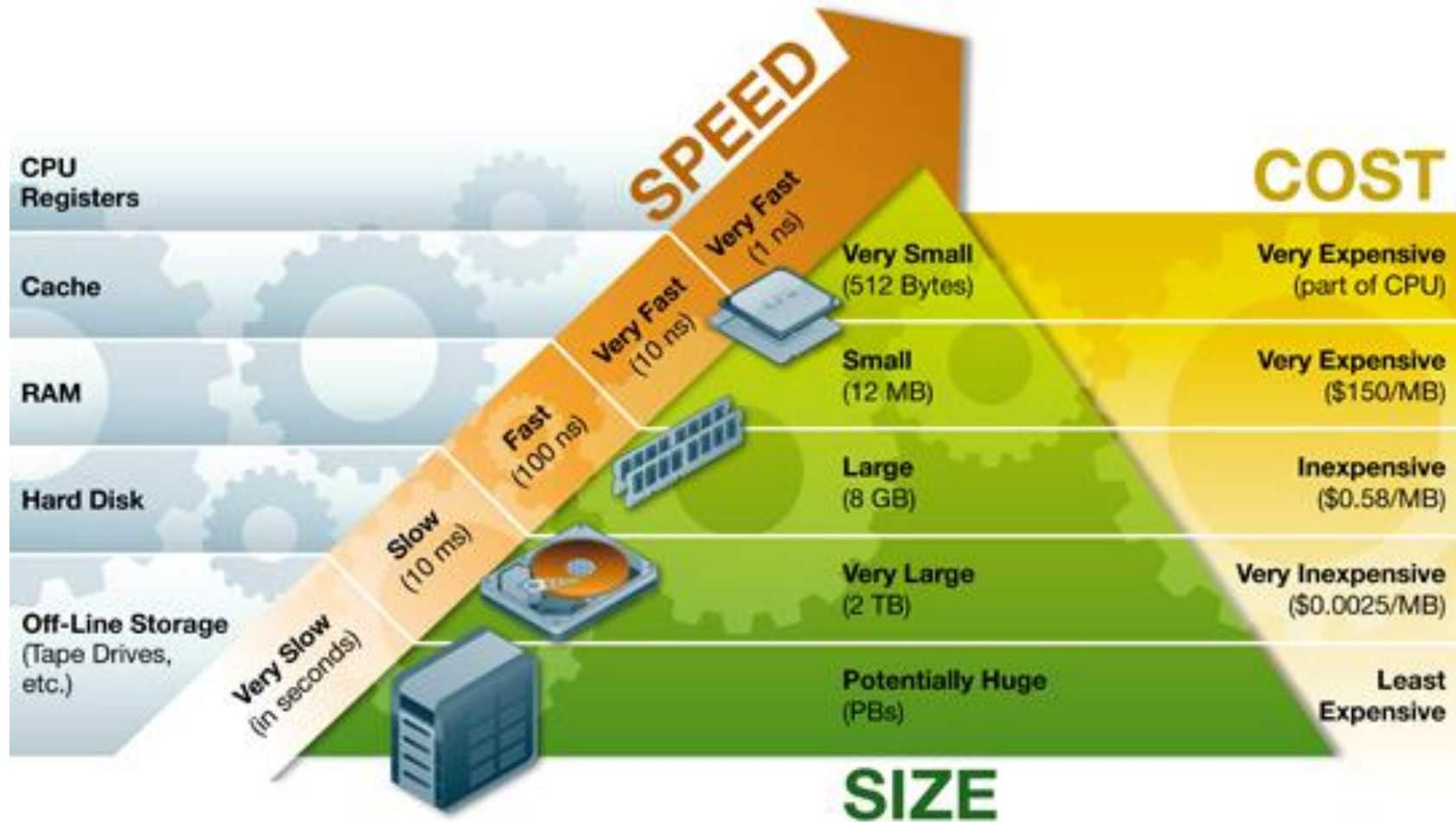
关系数据库的查询处理过程

- SQL → Plans: Interpretation
- Plans → Best Plan: Query Optimization
- Best Plan → Results: Query Evaluation

先考虑各个操作如何执行

- 选择 Selection
- 映射 Projection
- 链接 Join
- 排序 Sort
- 聚集（分组聚集） Aggregation
-

影响性能的因素：数据访问



影响性能的因素：数据访问

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zip	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

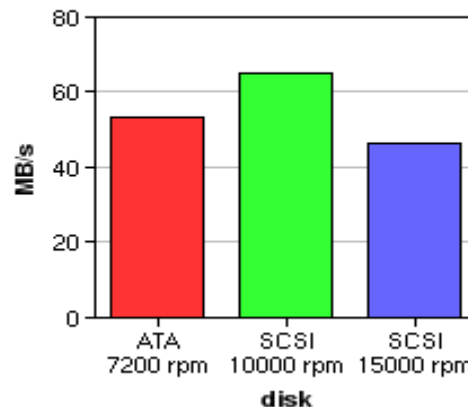


硬盘(I/O)的特点

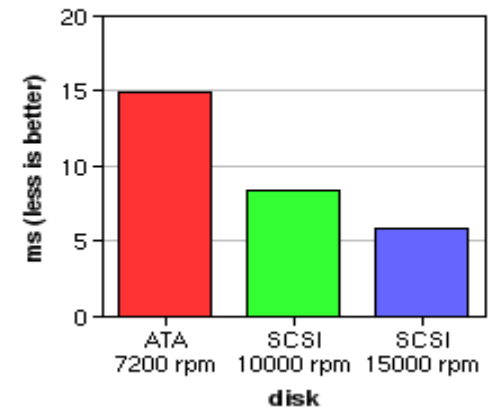
- 读写以页为单位 512 bytes
- 随机读取一页的动作：
 - 磁头移动到相应的位置
 - 磁碟旋转到相应的位置
 - 读取
- 顺序访问较随机访问快。



Disk sequential access



Disk random access time



闪存(I/O)的特点

SSD Performance Characteristics

Sequential read tput	250 MB/s	Sequential write tput	170 MB/s
Random read tput	140 MB/s	Random write tput	14 MB/s
Rand read access	30 us	Random write access	300 us

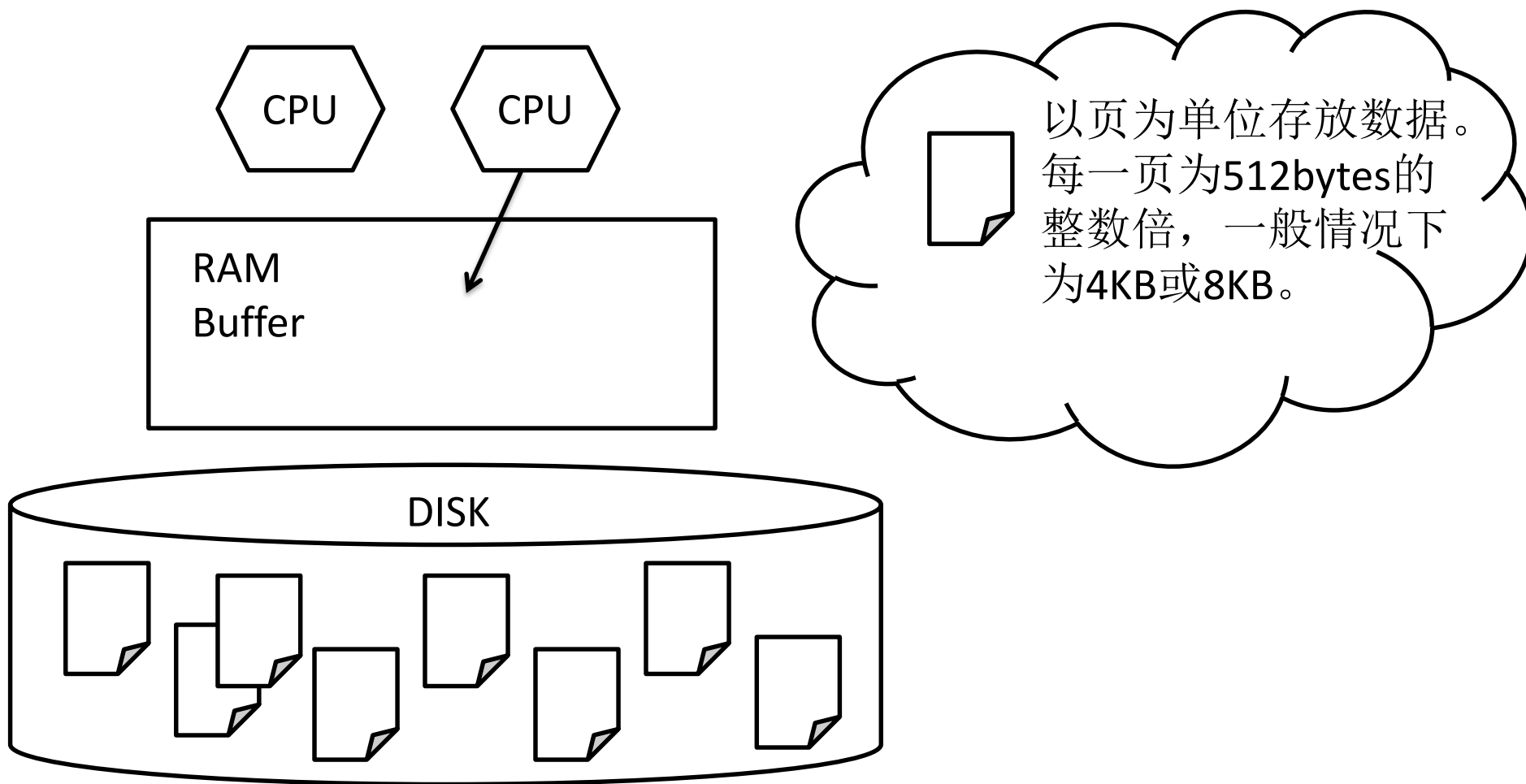
■ Why are random writes so slow?

- Erasing a block is slow (around 1 ms)
- Write to a page triggers a copy of all useful pages in the block
 - Find an used block (new block) and erase it
 - Write the page into the new block
 - Copy other pages from old block to the new block

提高数据库性能的宗旨之一

- 增加数据访问的局部性：
 - 对磁盘/闪存数据库而言
 - 减少I/O的次数；
 - 变随机访问为顺序访问。
 - 对内存数据库而言
 - 增加Cache的命中率。
 - 对并行或分布式数据库而言
 - 减少网络通信代价。

数据库的基本存储架构



基本操作一：选择

- 点查询：
SELECT name
FROM Employee
WHERE ID = 8478
- 范围查询：
SELECT name
FROM Employee
WHERE salary >= 120000 AND salary < 160000
- 多值查询：
SELECT name
FROM Employee
WHERE lastname = 'Gates' AND firstname = 'George'

基本操作二：投影/映射

Select Distinct major
From Student;

$\Pi_{\text{major}}(\mathbf{R})$

name	gender	major
李明	男	计算机专业
刘涛	女	法律专业
张茜	女	法律专业

major
计算机专业
法律专业

映射去重的实现方式之一：排序

2, 5, 2, 1, 2, 2, 4, 5, 4, 3, 4, 2, 1, 5, 2, 1, 3

排序后：

1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5

第一趟：

读取前6个元组到内存中的三个块中，对它们排序，
将它们作为子表 R_1 写出。同理得出子表 R_2 和 R_3 ，
其中 R_3 由最后5个元组排序后生成。

R_1 : 1 2 2 2 2 5

R_2 : 2 3 4 4 4 5

R_3 : 1 1 2 3 5

第二趟：首先把三个子表中的每一个块放入内存中

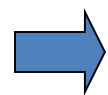
子表	内存中	磁盘上剩下的
$R_1 :$	1 2	2 2 , 2 5
$R_2 :$	2 3	4 4 , 4 5
$R_3 :$	1 1	2 3 , 5



1

因为 1 是排序后最小的第一个元组，所以在输出中产生 1 的一个副本，并且从内存的块中删除所有的 1

子表	内存中	磁盘上剩下的
$R_1 :$	2	2 2, 2 5
$R_2 :$	2 3	4 4, 4 5
$R_3 :$	2 3	5

 1,2

子表	内存中	磁盘上剩下的
$R_1 :$	5	
$R_2 :$	3	4 4, 4 5
$R_3 :$	3	5

 1,2,3

子表	内存中	磁盘上剩下的
$R_1 :$	5	
$R_2 :$	4 4	4 5
$R_3 :$	5	

➡ 1,2,3,4

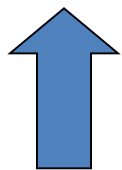
子表	内存中	磁盘上剩下的
$R_1 :$	5	
$R_2 :$	5	
$R_3 :$	5	

➡ 1,2,3,4,5

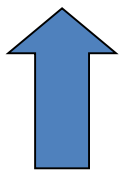
排序去重I/O复杂度分析

【算法分析（所需I/O）】

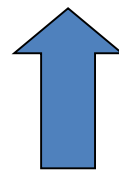
$$B(R) + B(R) + B(R) = 3 B(R)$$



用于在创建
排序子表时
读R的每一
一个块



用于将
各排序
子表写
到磁盘



用于在适当
的时候从子
表中读每一
一个块

思考题： 更好的映射去重方法？

基本操作三：连接

Select * From Major, Job
Where Major.sname = Job.sname;

$R \bowtie_{\text{sname}} S$

sname	job
李明	程序员
刘涛	金融分析师
张茜	金融分析师

sname	major
李明	计算机专业
刘涛	计算机专业
张茜	金融专业

sname	sname	major	job
李明	李明	计算机专业	程序员
刘涛	刘涛	计算机专业	金融分析师
张茜	张茜	金融专业	金融分析师

连接的实现方式之一： 嵌套循环

- $R(X,Y) \bowtie S(Y,Z)$ 算法

FOR each tuple s in S DO

FOR each tuple r in R DO

IF r and s join to make a tuple t THEN

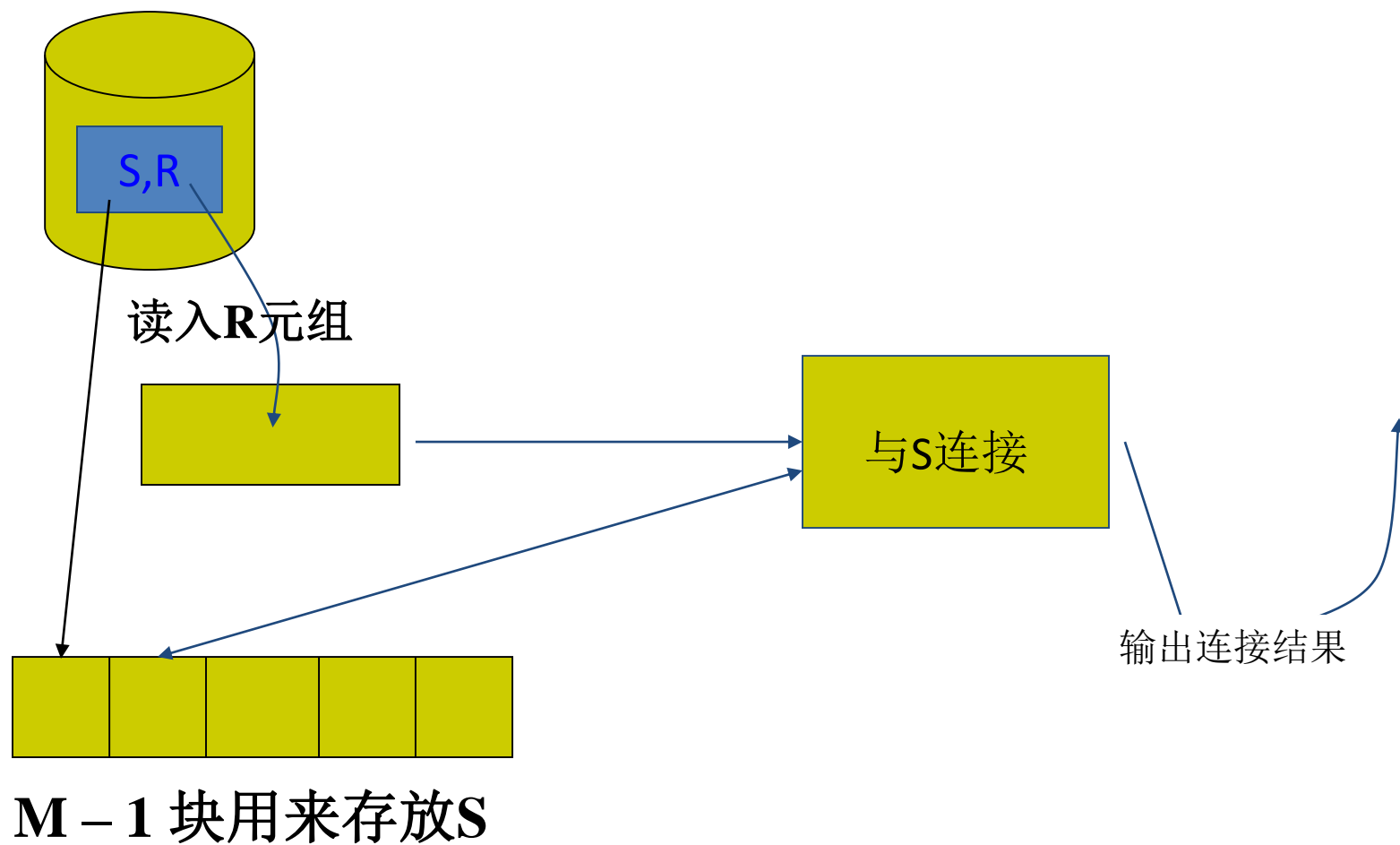
output s ;

基于块的嵌套循环

- 对作为操作对象的两个关系的访问均按块组织。
 - 确保了当在内层循环中处理关系R的元组时，我们可以用尽可能少的磁盘I/O来读取R
- 使用尽可能多的内存来存储属于关系S的元组，S是外层循环中的关系。
 - 将读到的R的每一个元组与能装入内存的尽可能多的S元组连接

假定 $B(R) \geq B(S) \geq M$, 则算法如下:

```
FOR each chunk of M-1 blocks of S DO BEGIN
    read these blocks into main-memory buffers;
    /*外层循环：对S进行*/
    organize their tuples into a search structure whose
        search key is the common attribute of R and S;
    FOR each block b of R DO BEGIN /*内层：一次一块地处理R*/
        read b into main memory;
        FOR each tuple t of b DO BEGIN /*处理当前块内的所有元组*/
            find the tuples of S in main memory that join with t;
            output the join of t with each of these tuples;
        END;
    END;
END;
END;
```



嵌套循环连接的I/O复杂度分析

- 假设S是较小的关系，则：

磁盘I/O次数为：

$$\frac{B(S)}{M-1} (M-1 + B(R))$$

外层循环的次数

内层读取M-1
块S和一块R

或：

$$B(S) + \frac{B(S)B(R)}{M-1}$$

当B(S)和B(R)都很大，**近似值** $B(S)B(R)/M$

如果 $B(S) \leq M-1$ ：等同于一趟连接算法

连接的实现方式之二：排序合并

对于要连接的关系 $R(X, Y)$ 和 $S(Y, Z)$, 已知有 M 块内存用作缓冲区, 进行下列操作:

- 1) 用 Y 作为排序关键字, 使用两阶段多路归并对 R 排序
- 2) 对 S 进行类似排序
- 3) 归并排好序的 R 和 S , 通常我们仅用两个缓冲区, 一个给 R 的当前块, 另一个给 S 的当前块。

排序合并连接I/O复杂度分析

- 磁盘I/O:
 - ❖ 一般情况下: $3 \times (B(R) + B(S))$
 - ❖ 特殊情况下: 额外的磁盘I/O依赖于是一个还是两个关系中具有公共的 Y 值 y 的元组太多。
- 对 M 的要求: 主要用来解决在 R 和 S 上执行两阶段、多路归并排序。

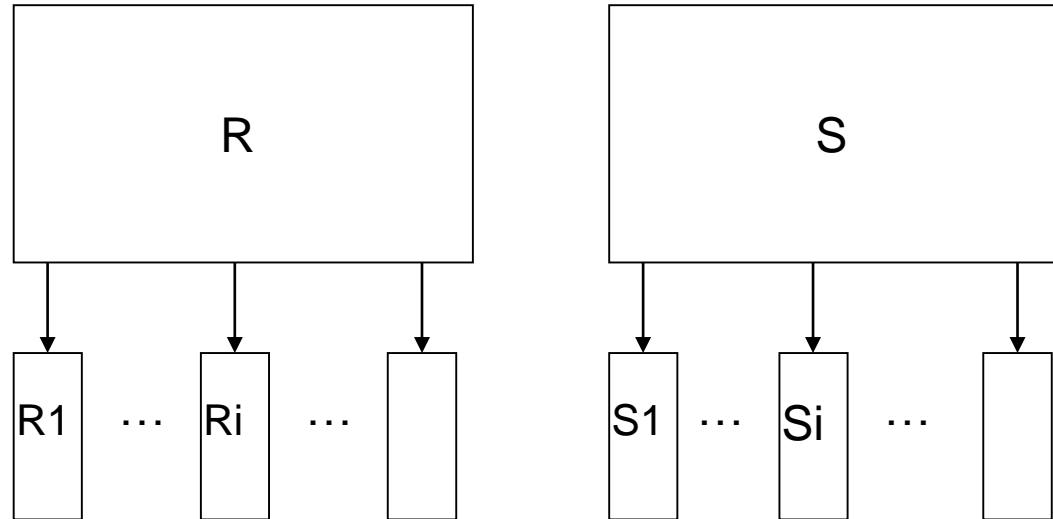
$$B(R) \leq M^2 \text{ 且 } B(S) \leq M^2$$

连接的实现方式之三：散列连接

- 散列连接的算法思想
 - 用连接属性Y作散列关键字，将R和S划分为可独立连接对。
 - 然后对对应桶对进行一趟连接。

散列连接 (Hash Join) 算法

Relations



Hash R and S by the same hash functions with **the join attributes as the hash key**.

必须用连接属性 Y 作 Hash 码

One-pass algorithm to each pair of corresponding buckets

$$R \bowtie S = (R1 \bowtie S1) \cup \dots (Ri \bowtie Si) \cup \dots$$

Cost: $3(B(R) + B(S))$ disk I/O Requirement: $\min(B(R), B(S)) \leq (M-1)(M-1)$
(每一对桶对中必须有一个能全部装入 $M-1$ 个缓冲区中)

散列连接I/O复杂度分析

【算法分析】

磁盘 I/O 数: $3(B(R) + B(S))$

大致所需M: $\min(B(R), B(S)) \leq M^2$ (每一对桶对中必须有一个能全部装入 $M - 1$ 个缓冲区中)

思考题：分组聚集的方法和I/O复杂度？

算子的语义

	一元操作	二元操作
一次单个元组	选择、投影	
一次整个关系	分组、去重	并、交、差、连接、积

算子执行算法总结

	一趟		一趟半		两趟			
技术	单遍扫描		嵌套循环		基于排序		基于散列	
条件	需要缓冲区大小M	磁盘IO	需要缓冲区大小M	磁盘IO	需要缓冲区大小M	磁盘IO	需要缓冲区大小M	磁盘IO
选择								
投影								
分组								
去重								
并								
交								
差								
连接								

总结

- 决定数据库的性能
 - 数据移动的代价和频率
- 提升数据库的性能
 - 减少数据移动
 - 提高数据访问局部性