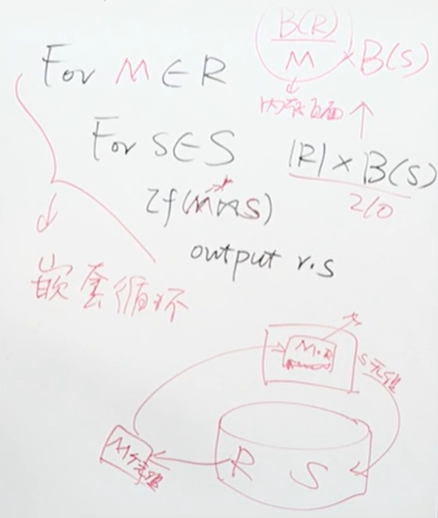
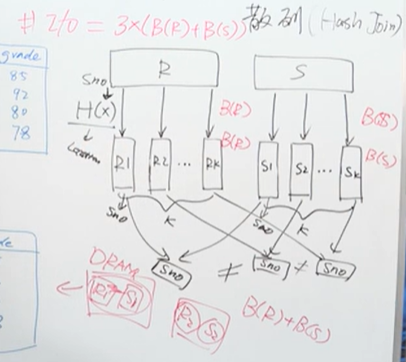
* 连接算子的实现-嵌套循环
  1. 
  2. IO代价：B(R)+B(S)\*B(R)/M
* 连接算子的实现-散列连接
  1. 
  2. IO代价：3\*(B(S)+B(R))
* 连接算子的实现-索引连接
  1. 将表S中每个元组取出来访问R的索引
  2. IO代价：|S|\*(H(B-tree)+1) [实际上B-tree节点缓存在内存中，树高访问-1]
* Sort Merge
  1. 3(B(R)+B(S))

**第*1*题：如课程中提到的，嵌套循环算法的I/O代价可以表示为 B(R) + B(S)\*B(R)/M，其中B(R)代表外循环表的页数，B(S)代表内循环表的页数，M代表内存能容纳的页数。那么以下哪种方案无法提升该算法的效率？**

A：增加内存容量

B：将小表放到内循环（作为S），大表放到外循环（作为R）√

C：将大表放到内循环（作为S），小表放到外循环（作为R）

D：更换速度更快的硬盘

**第*2*题：课程中介绍了两种连接操作执行算法，嵌套循环和散列连接。以下哪种场景更适合使用嵌套循环算法？**

A：一张表很大，一张表很小。小的表几乎可以容纳到内存中。

B：两张表都比较大，都无法容纳到内存中。√

C：两张表都比较小，都可以容纳到内存中。

D：几乎在所有情况下，散列连接的效率都更高。

**第*3*题：假设R表有1000行，S表有100行；每页可以容纳10行数据（无论R表或S表）；内存可以容纳3页；R表和S表均有x属性，并且在x属性上均创建了索引；假设x具备很高的辨识度并且分布均匀。那么，理论上实现R和S在x上的等值连接的最佳算法是：**

**B(R)=100 B(S)=10 M=3**

A：嵌套循环 10+10/3\*100=410

B：散列连接 3\*(100+10)=330

C：索引连接 <=100\*(log(1000)-1+1)=300√

D：都一样

1. 在做散列连接的时候，可能会遇到数据分布不均的情况，比如在连接属性上取值为k的元组特别多，多到连内存都容纳不下。请问这种情况应该如何应对？

针对连接属性上取值为k的元组再进行嵌套循环，划分为多个页依次进入内存进行连接。

2-除了课程介绍的连接算法，你是否能想到其他高效的算法，它甚至在某些情况下比前面介绍的算法更快？

当有某个表（R）特别小，可以完全放在内存内时，并且给这个表的连接属性做hash，大表（S）仍然仍然分成M段进入内存，IO开销为B(S)/M

Sort Merge：两个表都特别大，并且在连接属性上都有索引时，可以给两表需要连接的属性都做排序，两表都只需要按排序顺序将页面读到内存中，IO开销为3B(S)+3B(R)+(B(S)+B(R))=4(B(R)+B(S)) (可以减去冗余的一步merge，优化到3(B(R)+B(S)))

3-请为以下问题设计一个I/O代价低的算法：  
给定一张表R(a,b,c,d)，找到R中在属性a上发生次数少于k次的所有取值。（如果R中一共有n个元组在属性a上的取值为x，我们称x在R中发生了n次；假设R很大，无法容纳于内存中；假设a的不同取值个数很多，这些取值也无法都容纳于内存中。）

先对属性a进行归并排序，然后依次查次数，IO代价为3B(R)