1-请根据你自己的理解找出以下查询的最佳关系代数查询计划。

Select sno, sname

From Student, SC

Where Student.sno = SC.sno

And grade<60

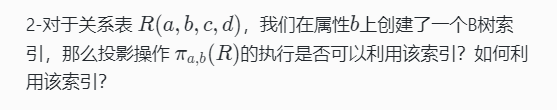
And cno in

(Select cno

From Course

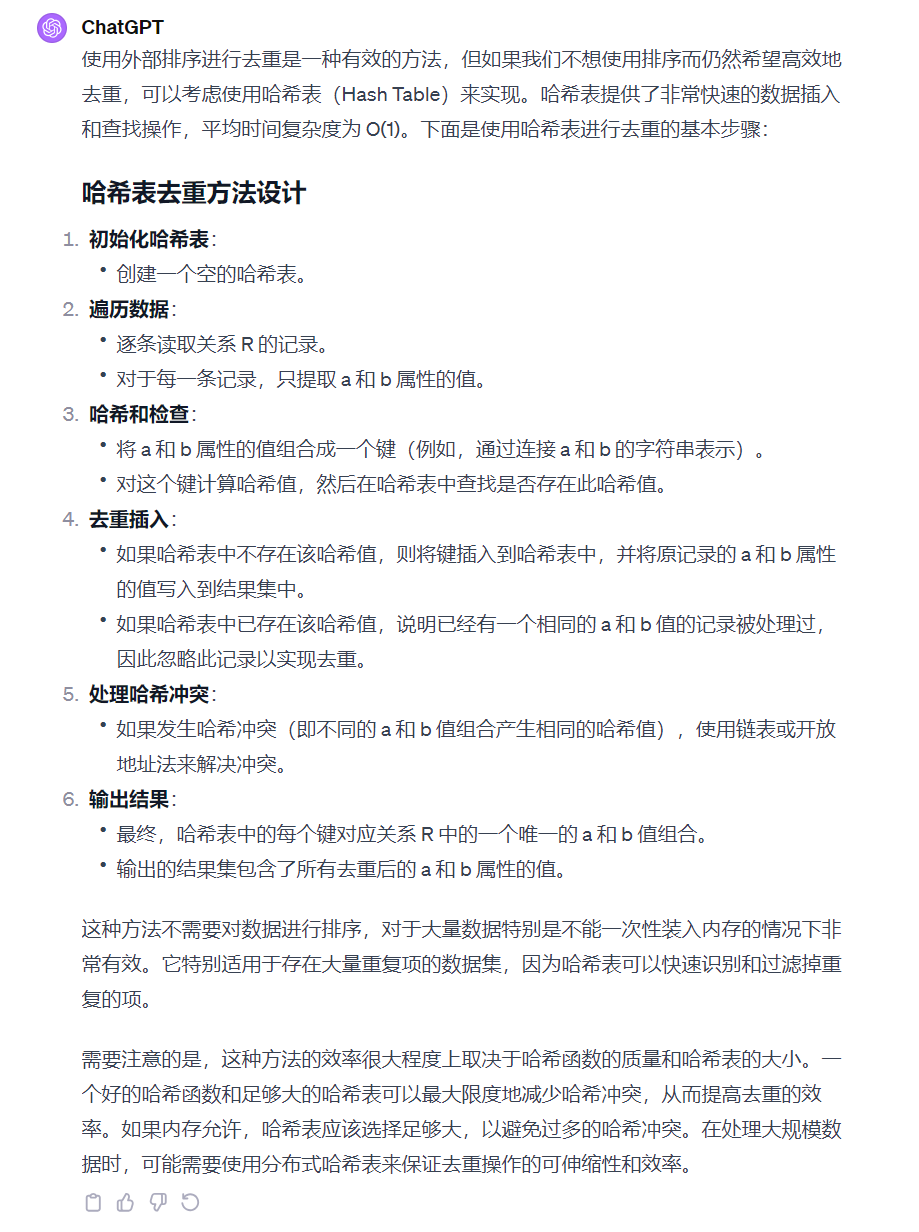
Where cname = 'Math I');







3-课程中我使用了外部排序（External Sorting）的方式实现了投影操作的去重。请你设计另一种去重的方法，不使用排序，仍然可以达到很高的效率。



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？

在数据库连接算法的范畴内，除了散列连接、嵌套循环连接和索引连接外，还存在一些其他的优化策略或算法，它们在特定情况下可以提供更高效的性能：

**1、Bitmap索引连接：**

如果连接的列上有Bitmap索引，可以利用Bitmap索引的位操作来加速连接过程。这种方法在数据仓库查询中特别有效，尤其是在列式存储数据库中。

**2、分区连接：**

类似于散列连接中的分区思想，但是是在物理存储层面上对数据进行分区。如果两个表都按照连接键进行了分区，并且分区策略相同，那么连接操作可以在分区间独立进行，从而实现并行处理。

**3、并行连接：**

利用现代多核处理器的计算能力，可以将连接操作分解为多个小的任务并行执行，从而提高查询的执行效率。

**4、流水线连接（Pipelined Join）：**

在某些基于流的处理系统中，如Spark或Flink，可以在数据流动时动态执行连接操作，这种方式可以减少存储需求并实现实时处理。

**5、适应性连接（Adaptive Join）：**

根据数据的实际分布动态选择最佳的连接策略。例如，SQL Server的“适应性查询处理”功能会在查询执行时根据数据的实际行数选择使用嵌套循环连接还是哈希连接。

**6、星型连接：**

在数据仓库的星型模式或雪花模式查询中，星型连接可以有效利用维度表的索引来加速对事实表的查询。

在实际应用中，数据库管理系统的查询优化器会根据数据的特征、索引的可用性以及系统资源的情况来选择最合适的连接算法。这些算法可能是上述算法的变体，或者是特定于某一数据库产品的专有算法。

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？



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。）

为了设计一个I/O代价低的算法来处理这个问题，我们可以采用外部排序和分区的方法来优化对表R的扫描和计数操作。以下是一个可能的算法设计：

**分区遍历（Partition-Based Scan）:**

将表R的数据按照属性a的值范围进行分区，每个分区的大小应该小于或等于内存的大小。

逐个分区地读取数据，这样每次只有一个分区的数据被加载到内存中。

**内存中计数（In-Memory Counting）:**

对于每个分区，使用哈希表或树结构来计数属性a的取值及其出现次数。

每次处理一个分区，就更新一个全局的数据结构（如一个临时的计数表），该表记录了每个不同的a值在已处理分区中的累计出现次数。

**归并计数（Merge Counts）:**

一旦所有分区都被处理完毕，我们就有了一个包含所有a值及其出现次数的全局计数表。

遍历这个计数表，找出出现次数少于k次的所有a值。

**外部排序（External Sorting）:**

如果全局计数表仍然太大，无法一次性装入内存，则可以使用外部排序算法来对计数结果进行排序，这有助于快速找到出现次数少于k次的a值。

**二次扫描（Second Scan）:**

如果需要输出这些a值对应的所有记录，可以在计数完成后，根据已经确定的出现次数少于k次的a值，再次扫描表R，输出所有相关记录。

此方法的优点是，它不需要一次性将整个表R或所有不同的a值加载到内存中，而是通过分区和逐步处理来减少内存需求和I/O代价。每个分区可以独立处理，甚至可以并行处理以进一步提高效率。这种方法也适用于分布式系统，其中每个分区可以分配给不同的节点进行处理。