唯一性：人名可能重复，而 ID 是唯一的标识符。使用 ID 可以准确区分具有相同或相似名称的不同作者。
数据一致性：如果作者的其他信息发生变化（如姓名变更或其他个人信息更新），只需要在作者的记录中更新，而无需更改所有书的文档。这保证了数据的一致性和完整性。
查询效率：使用 ID 作为引用可以更快地索引和查询，特别是在关联大量数据的情况下，ID 通常是优化过的键，可以加速查找和检索操作。
数据模型的扩展性和灵活性：使用 ID 作为关联键允许数据库模型在未来容易地扩展，可以添加更多关于作者的信息而不影响书的文档结构。
引用完整性：在数据库层面，使用 ID 可以维护引用完整性，确保数据引用的有效性。

查询 db.inventory.find({ tags: ["computer", "music"] }) 默认情况下是寻找 tags 数组与查询数组完全匹配的文档。
这个查询表达的是寻找 tags 属性正好是包含 "computer" 和 "music" 这两个字符串的数组的文档。换句话说，它查找的是 tags 数组与查询中数组**精确匹配**的文档，包括数组中值的顺序和数量。
因此，选项 A {... tags: ["computer", "music"]... } 是满足这个查询条件的，因为它的 tags 数组完全匹配查询条件中的数组。
选项 B 的文档不匹配，因为它们只包含查询数组中的一个值。选项 C 的第二个文档包含了额外的值 "movie"，这超出了查询数组的匹配。选项 D 的文档除了第一个以外都不满足精确匹配的要求。
所以根据这个查询语句，我们应该选择 A。如果查询是使用 $all 操作符，例如 db.inventory.find({ tags: { $all: ["computer", "music"] } })，则会选择所有包含 "computer" 和 "music" 的文档，无论顺序如何，那样的话选项 C 将是正确答案。

第 4 题(本题2分)：大部分系统都是对存储空间进行分页管理的。请问，分页模式的优势不包括?

A：有利于减少存储空间的碎片化，提升空间利用率。 ✗

B：有利于提升数据访问的性能。 ✗

C：有利于提升内存缓存的效率。 ✓

D：有利于减少空间管理的成本（即减少空间管理对CPU和内存资源的消耗）。 ✗

分页模式（paging）主要是出于内存管理的需要而设计的，它将物理内存分割为固定大小的块，称为“页”。系统加载这些页到内存中，这样即使是较大的数据集也能被分成多个小块来管理。这种方法确实可以提升空间利用率、数据访问性能，以及减少管理成本。但是，关于选项 C 的说法“有利于提升内存缓存的效率”可能并不总是准确的。

选项 C 提到的“内存缓存效率”通常是指数据被频繁访问时缓存的有效性，这依赖于多种因素，包括缓存替换策略、数据的局部性原理等。**分页系统确实可以提高数据访问性能，因为它允许操作系统将常用的页保留在快速的物理内存中。**然而，分页本身并不直接提升缓存的效率。实际上，如果页的大小设置不当，或者如果应用程序的访问模式不符合分页大小，那么可能会导致缓存命中率下降，因为每次页替换都可能涉及到较大量的数据移动。

因此，如果必须在给定选项中选择一个不是分页模式优势的，选项 C 可能是最合适的，因为分页管理本身并不保证提升“内存缓存的效率”，这更多地依赖于内存缓存的设计和数据访问模式。分页更多是提高虚拟内存管理的效率，而缓存效率需要额外的策略和优化。**提升的是数据访问的效率而不是数据缓存的效率。**

第 5 题(本题2分)：课程中提到，内存由于比较昂贵且无法持久地保存数据，通常只作为数据缓存。那么，什么数据不适合被放在缓存中?

A：经常被修改的数据 ✗

B：像Inode这样的组织结构数据 ✗

C：刚被插入的数据 ✗

D：刚被删除的数据 ✓