数据库考试题

考试试题：

1. 体系结构：
   a. 假设一个事务是将SQL嵌入到C语言中实现的，在运行过程中，有大约90%的时间是花在SQL语句执行上的，如果只对SQL语句实施并行，那么期望能够获得多大的加速比，请说明理由。17.4 / 20.5

> **17.4** Suppose a transaction is written in C with embedded SQL, and about 80 percent of the time is spent in the SQL code, with the remaining 20 percent spent in C code. How much speedup can one hope to attain if parallelism is used only for the SQL code? Explain.
> **Answer:** Since the part which cannot be parallelized takes 20% of the total running time, the best speedup we can hope for has to be less than 5.

> **17.4** 假设一个事务是用嵌入式SQL的C语言编写的，大约80%的时间花在SQL代码中，剩下的20%花在C代码中。如果并行性仅用于SQL代码，人们希望达到多少加速？解释一下。|答：由于无法并行化的部分需要总运行时间的20%，我们可以希望的最佳加速必须小于5。

   b. 一些数据库操作，比如连接操作，在数据（比如，连接涉及的其中一个关系表中的数据）存放在内存中和不在内存中这两种情况下，速度会有明显差异。请利用上述现象，解释超线性加速比（superliner speedup）现象，即一个程序的加速比高于分配给它的资源数量。17.5 / 20.9
   超线性加速比是指一个程序在使用更多资源（如内存、处理器核心等）时，其性能提升的速度超过了资源的增加速度。这种现象在某些情况下可能会出现，特别是当程序的性能受限于资源瓶颈时。
   考虑一个数据库连接操作的例子，在数据存放在内存和不在内存的情况下：数据存放在内存中：当数据存放在内存中时，数据库连接操作可以直接在内存中进行，而不需要进行磁盘I/O操作。由于内存访问速度远远快于磁盘访问速度，因此连接操作的速度会非常快。
   数据不存放在内存中：如果数据不在内存中，连接操作将需要进行磁盘I/O操作来访问存储在磁盘上的数据。由于磁盘I/O速度远远慢于内存访问速度，连接操作的速度会显著降低。
   现在假设我们增加了系统的内存容量，使得更多的数据可以存放在内存中。在这种情况下，连接操作的性能可能会超过线性增长，即它的执行时间可能比预期更快。这是因为更多的数据可以存放在内存中，减少了磁盘I/O的次数，从而提高了连接操作的速度。
   具体来说，当程序所需资源的增加导致了资源瓶颈的消除或减轻时，超线性加速比现象就可能发生。在这种情况下，程序的性能提升可能会超过资源的增加量，从而导致超线性加速比。

当数据适合内存时，连接操作等数据密集型操作的执行速度通常会更快。这是因为内存访问速度远远快于磁盘访问速度，而且内存中的数据可以直接进行操作，而不需要进行磁盘I/O操作。因此，在这种情况下，操作的执行时间可能会比预期更快。

当操作在数据不适合内存的情况下执行时，需要进行大量的磁盘I/O操作来读取和处理数据，这会导致操作的执行速度变慢。因此，操作的执行时间可能会超过预期，即使分配给应用程序的资源量没有增加，也可能会看到加速。

换句话说，当数据适合内存时，操作的执行速度提高了，而资源的使用量保持不变，这导致了超线性加速的现象。因此，适当管理内存和数据的存储位置可以显著影响操作的执行速度和应用程序的性能。

2. 并行数据库

    a. 请描述流水线并行的优点和缺点。 18.13

Describe the benefits and drawbacks of pipelined parallelism.
Answer:

- **Benefits:** No need to write intermediate relations to disk only to read them back immediately.
- **Drawbacks:**
  a. Cannot take advantage of high degrees of parallelism, as typical queries do not have large number of operations.
  b. Not possible to pipeline operators which need to look at all the input before producing any output.
  c. Since each operation executes on a single processor, the most expensive ones take a long time to finish. Thus speed-up will be low despite the use of parallelism.

描述流水线并行的优点和缺点。
Answer:

•优点：无需将中间关系写入磁盘，只需立即读回它们。

- **Drawbacks:**
  A.不能利用高度并行性，因为典型的查询没有大量的操作。 b.流水线操作员不可能在产生任何输出之前查看所有输入。 由于每个操作都在一个处理器上执行，所以最紧张的操作需要很长时间才能完成。 因此，尽管使用并行性，但加速将很低。

好处：

消除中间磁盘I/O的需求：流水线并行消除了将中间结果写入磁盘，只为了立即读取它们进行进一步处理的需要。这减少了与磁盘I/O相关的开销，提高了性能。

缺点：

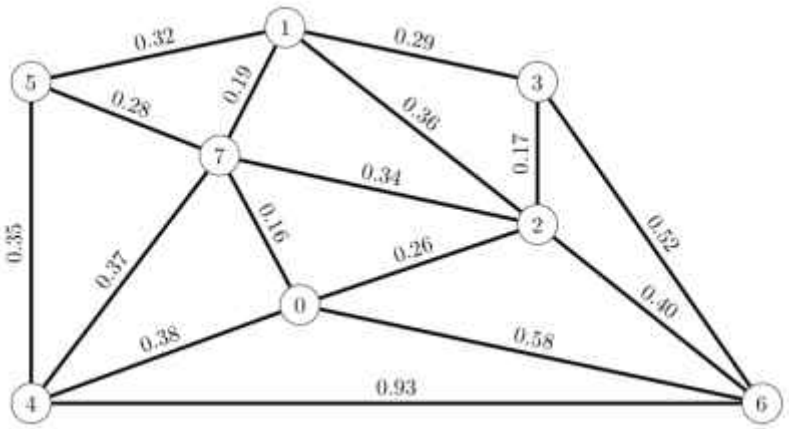
可扩展性有限：流水线并行可能无法充分利用高度并行化，因为典型的查询没有大量操作。因此，并行度的潜力可能无法充分利用，导致计算资源的利用不足。

无法流水线化某些操作：某些操作需要访问所有输入数据才能产生任何输出，这使得无法流水线化这些操作。因此，对于包含此类操作的查询，可能无法实现流水线并行的好处。

对于耗费资源的操作，执行时间较长：由于每个操作都在单个处理器上执行，因此计算资源消耗大的操作可能需要较长时间才能完成，从而降低整体性能。尽管使用了并行处理，但性能提升可能受到管

线中最慢操作的限制。

b. 对于下面这样一个图，要找一个从节点5到节点6的路径，使得路径上的权重和最小。请设计一个并行策略完成这个任务。



我们可以使用并行 Dijkstra 算法。该算法将图分割成多个子图，每个子图由一个处理器负责计算。每个处理器计算其子图中的最短路径，并将结果传递给邻近的处理器，以便合并结果。具体来说，我们可以按照以下步骤实现并行 Dijkstra 算法：1. 将图分割成多个子图，每个子图包含一些节点。2. 每个处理器负责计算其子图中的最短路径，从节点 5 开始。3. 每个处理器计算完毕后，将结果传递给邻近的处理器，以便合并结果。4. 重复步骤 3，直到所有处理器都计算完毕。5. 最后，合并所有处理器的结果，得到从节点 5 到节点 6 的最短路径。

3. 分布式数据库

a. 请列举一个读一次、写所有可用（read one, write all available）方法导致错误状态的一个例子。 19.6

19.6 Give an example where the read one, write all available approach leads to an erroneous state.
Answer: Consider the balance in an account, replicated at N sites. Let the current balance be $100 – consistent across all sites. Consider two transactions T_i and T_2 each depositing $10 in the account. Thus the balance would be $120 after both these transactions are executed. Let the transactions execute in sequence: T_i first and then T_2. Let one of the sites, say s, be down when T_i is executed and transaction t_j reads the balance from site s. One can see that the balance at the primary site would be $110 at the end.

19.6 举一个例子，其中读一个，写所有可用的方法导致错误状态。 答：考虑一个帐户中的余额，在N个站点复制。 让当前余额为$100–在所有网站上保持一致。 考虑两个trans操作T1和T2每个在帐户中存入$10。 因此，在执行这两笔交易后，余额将为$120。 让tramac tions按顺序执行：先T1，然后T2。 让其中一个站点（比如s）在执行T1时关闭。事务t2从站点s读取余额。 人们可以看到，在主站点的余额将是end110在年底。

假设账户余额被复制到了N个站点上，并且当前余额为$100，在所有站点上都是一致的。考虑两个交易Ti和T2，每个交易向账户存入$10。因此，在这两个交易执行后，余额将为$120。让这两个交易按顺序执行：先执行T，然后执行T2。假设当执行T时，其中一个站点（例如站点s）处于离线状态，并且交易T从站点s读取余额。可以看到，最终主站点的余额将为$110。

b. 讨论集中式数据库和分布式数据库的优缺点。19.16

19.16 讨论集中式和分布式数据库的相对优势。

Answer:

•分布式数据库允许用户方便和透明地访问未存储在站点上的数据，同时允许每个站点控制其自己的本地数据。分布式数据库可以比集中式系统更可靠，因为如果一个站点出现故障，数据库可以继续运行，但如果集中式系统出现故障，数据库将无法继续正常运行。此外，分布式数据库允许并行执行查询，并可能将一个查询拆分为多个部分以提高吞吐量。

4. 时空数据和移动性

a. 数据库中，时间有哪两种类型，如何区分。25.1

25.1 What are the two types of time, and how are they different? Why does it make sense to have both types of time associated with a tuple?

Answer: A temporal database models the changing states of some aspects of the real world. The time intervals related to the data stored in a temporal database may be of two types - *valid time* and *transaction time*. The valid time for a fact is the set of intervals during which the fact is true in the real world. The transaction time for a data object is the set of time intervals during which this object is part of the physical database. Only the transaction time is system dependent and is generated by the database system.

Suppose we consider our sample bank database to be bitemporal. Only the concept of valid time allows the system to answer queries such as - "What was Smith's balance two days ago?". On the other hand, queries such as - "What did we record as Smith's balance two days ago?" can be answered based on the transaction time. The difference between the two times is important. For example, suppose, three days ago the teller made a mistake in entering Smith's balance and corrected the error only yesterday. This error means that there is a difference between the results of the two queries (if both of them are executed today).

有效时间（valid time）：指的是数据在现实世界中的真实有效期。换句话说，有效时间是数据在现实世界中被认为是真实的时间范围，即数据在现实世界中的存在期间。例如，如果一个人的出生日期是1990年1月1日，那么在1990年1月1日之后的任何时间点，这个人的出生日期仍然是1990年1月1日，这就是有效时间。

事务时间（transaction time）：指的是数据对象在物理数据库中存在的时间范围。换句　　话说，事务时间是数据在数据库中被创建、修改和删除的时间范围，即数据在数据库　　中的存在期间。事务时间是由数据库系统生成的，它与系统相关。

在一个元组（tuple）中关联这两种时间是有意义的，因为它们分别代表了不同的时间概念，并提供了不同的信息：

有效时间反映了数据在现实世界中的真实有效期，帮助我们理解数据在现实世界中的状态和变化。

事务时间反映了数据在数据库系统中的存在期间，帮助我们追踪数据在数据库中的历史和变化

b. 如果通过增加一个时间属性来将关系转换为时态关系，函数依赖能保持吗，为什么？25.9

25.9 Will functional dependencies be preserved if a relation is converted to a temporal relation by adding a time attribute? How is the problem handled in a temporal database?

Answer: Functional dependencies may be violated when a relation is augmented to include a time attribute. For example, suppose we add a time attribute to the relation *account* in our sample bank database. The dependency *account-number*→*balance* may be violated since a customer's balance would keep changing with time.

To remedy this problem temporal database systems have a slightly different notion of functional dependency, called *temporal functional dependency*. For example, the temporal functional dependency:

$$account\text{-}number \xrightarrow{\tau} balance$$

over *Account-schema* means that for each instance *account* of *Account-schema*, all snapshots of *account* satisfy the functional dependency

$$account\text{-}number \rightarrow balance;$$

i.e at any time instance, each account will have a unique bank balance corresponding to it.

5. 大数据

   a. 假设你希望将一个大学的schema建模成一个图（graph），对于下面的每一个关系，他们是建模成节点还是边？这个模型能捕捉sections和courses之间的连接吗？ 10.9
   (1) student (2) instructor (3) course (4) section (5) takes (6) teaches

Answer:

Each relation corresponding to an entity (student, instructor, course, and section) would be modeled as a node. *Takes* and *teaches* would be modeled as edges. There is a further edge between *course* and *section*, which has been merged into the *section* relation and cannot be captured with the above schema. It can be modeled if we create a separate relation that links sections to courses.

首先，对于每个实体（student、instructor、course 和 section），将建立一个节点来表示。然后，使用边来表示实体之间的关系，例如学生选修课程（takes）、教师教授课程（teaches）等。

然而，在课程（course）和章节（section）之间存在一种特殊的关系，即每个课程可能包含多个章节。然而，在上述的模式中，课程和章节之间的关系已经被合并到章节关系中，无法被准确地捕获。

为了解决这个问题，可以创建一个额外的关联关系，将章节和课程之间的关系单独表示出来。这样，每个章节可以与一个或多个课程相关联，更好地反映了课程和章节之间的关系。

b. . 假如你需要保存很多的小文件（每个大小2K），从分布式文件系统和分布式键值存储中选择，你会选择哪种，为什么？ 10.1

Answer:
The key-value store, since the distributed file system is designed to store a moderate number of large files. With each file block being multiple megabytes, kilobyte-sized files would result in a lot of wasted space in each block and poor storage performance.

分布式文件系统通常设计用于存储少量但较大的文件，其中每个文件块大小通常为多兆字节。如果将大量千字节大小的文件存储在分布式文件系统中，将导致每个文件块内的空间浪费较多，并且存储性能可能不佳。因为文件系统需要将每个小文件分配给一个完整的文件块，即使文件只占用该块的一小部分，也会导致大量的空间浪费。

相比之下，分布式键值存储系统更适合存储大量小文件。键值存储系统通常采用更灵活的数据结构，可以有效地存储和管理大量小型数据对象，而不会浪费大量的存储空间。每个文件都可以作为一个键值对存储，其中键是文件的唯一标识符，而值则是文件的内容。由于键值存储系统不需要分配固定大小的块来存储文件，因此可以更有效地利用存储空间，并且通常具有更好的存储性能。

键值存储，因为分布式文件系统设计用于存储中等数量的大文件。由于每个文件块都有多兆字节，KB大小的 fles 会导致每个块中浪费大量空间，存储性能也会变差。