姓名：胡瑞康

学号：22336087

ch15

问题 1

> 假设一个块中只能容纳一个元组,内存最多可以容纳三个块。当应用于对第一个属性排序以下元组时,请展示排序合并算法的每个传递中创建的运行:(kangaroo, 17), (wallaby, 21), (emu, 1), (wombat, 13), (platypus, 3), (lion, 8), (warthog,4), (zebra, 11), (meerkat, 6), (hyena, 9), (hornbill, 2), (baboon, 12)。
> 对第一个属性

```
初始分块:
运行 1: [('emu', 1), ('kangaroo', 17), ('wallaby', 21)]
运行 2: [('lion', 8), ('platypus', 3), ('wombat', 13)]
运行 3: [('meerkat', 6), ('warthog', 4), ('zebra', 11)]
运行 4: [('baboon', 12), ('hornbill', 2), ('hyena', 9)]

第 1 轮合并:

合并: [[('emu', 1), ('kangaroo', 17), ('wallaby', 21)], [('lion', 8), ('platypus', 3), ('wombat', 13)]]
->
[('emu', 1), ('kangaroo', 17), ('lion', 8), ('platypus', 3), ('wallaby', 21), ('wombat', 13)]

合并: [[('meerkat', 6), ('warthog', 4), ('zebra', 11)], [('baboon', 12), ('hornbill', 2), ('hyena', 9)]]
->
[('baboon', 12), ('hornbill', 2), ('hyena', 9), ('meerkat', 6), ('warthog', 4), ('zebra', 11)]

第 2 轮合并:
合并: [[('emu', 1), ('kangaroo', 17), ('lion', 8), ('platypus', 3), ('wallaby', 21), ('wombat', 13)], [('baboon', 12), ('hornbill', 2), ('hyena', 9), ('meerkat', 6), ('warthog', 4), ('zebra', 11)]]
->
[('baboon', 12), ('emu', 1), ('hornbill', 2), ('hyena', 9), ('kangaroo', 17), ('lion', 8), ('meerkat', 6), ('platypus', 3), ('wallaby', 21), ('warthog', 4), ('wombat', 13), ('zebra', 11)]

最终排序结果:
[('baboon', 12), ('emu', 1), ('hornbill', 2), ('hyena', 9), ('kangaroo', 17), ('lion', 8), ('meerkat', 6), ('platypus', 3), ('wallaby', 21), ('warthog', 4), ('wombat', 13), ('zebra', 11)]
```

问题 2

> 设 r 和 s 是没有索引的关系,并假设这些关系没有排序。在假设有无限内存的情况下,计算 r⋈s 的最低成本方法(以 I/O 操作为代价)是什么?这个算法需要多少内存?

最低成本方法：块嵌套循环连接（Block Nested Loop Join）

- **方法说明：**
  - 对于每一个关系 `r` 的块，遍历关系 `s` 的所有块，进行匹配。
- **I/O 操作成本：**
  - 假设 `r` 有 `|r|` 块，`s` 有 `|s|` 块。
  - 总成本 = `|r|` 读 + `|r| * |s|` 读
  - 由于无索引且无排序，块嵌套循环连接是最基本的方法。
- **所需内存：**
  - 需要至少 2 个缓冲块：一个用于 `r`，一个用于 `s`。

问题 3

> 假设您需要对一个大小为 40 GB 的关系进行排序,每个块为 4 KB,使用 40 MB 的内存。假设寻道的成本为 5 毫秒,而磁盘传输速率为每秒 40 MB。

a. 计算在 b_b = 1 和 b_b= 100 的情况下,对关系进行排序的成本,以秒为单位。

计算块的数量和内存中块的数量：

- 总块数：$b_r = \frac{40\times1024\times1024}{4} = 10,485,760$ 块
- 内存中块数：$M = \frac{40\times1024}{4} = 10,240$ 块
- $b_r/M = 1024$

情况一： $b_b = 1$

1. 计算每趟归并的参数：
   - $\lfloor M/b_b \rfloor - 1 = 10239$
   - $\log_{10239}(1024) \approx 0.75$，取上界为 1
2. 块传输次数：
   - $b_r \times (2 \times \lceil \log_{10239}(1024) \rceil + 1) = 10,485,760 \times (2 \times 1 + 1) = 31,457,280$ 次
3. 寻道次数：
   - $2 \times \lceil \frac{b_r}{M} \rceil + \lceil \frac{b_r}{b_b} \rceil \times (2 \times \lceil \log_{10239}(1024) \rceil - 1)$
   - $= 2 \times 1024 + 10,485,760 \times (2 \times 1 - 1) = 2048 + 10,485,760 = 10,487,808$ 次
4. 计算时间：
   - 传输时间：$31,457,280 \times 0.0001 = 3,145.728$ 秒
   - 寻道时间：$10,487,808 \times 0.005 = 52,439.04$ 秒
   - 总时间：$3,145.728 + 52,439.04 = 55,584.768$ 秒

情况二： $b_b = 100$

1. 计算每趟归并的参数：
   - $\lfloor M/b_b \rfloor - 1 = 101$
   - $\log_{101}(1024) \approx 1.5$，取上界为 2
2. 块传输次数：
   - $b_r \times (2 \times \lceil \log_{101}(1024) \rceil + 1) = 10,485,760 \times (2 \times 2 + 1) = 52,428,800$ 次

3. 寻道次数：
   - $2 \times \lceil \frac{b_r}{M} \rceil + \lceil \frac{b_r}{b_b} \rceil \times (2 \times \lceil \log_{101}(1024) \rceil - 1)$
   - $= 2 \times 1024 + 104,858 \times (4 - 1) = 2048 + 314,574 = 316,622$ 次
4. 计算时间：
   - 传输时间：$52,428,800 \times 0.0001 = 5,242.88$ 秒
   - 寻道时间：$316,622 \times 0.005 = 1,583.11$ 秒
   - 总时间：$5,242.88 + 1,583.11 = 6,825.99$ 秒

b. 在每种情况下,需要多少次归并操作?

情况一：$b_b = 1$

1. **归并段数量**：
   - 第一阶段创建的归并段数量：$\lceil \frac{b_r}{M} \rceil = \lceil \frac{10,485,760}{10,240} \rceil = 1024$ 个归并段。
2. **每趟归并参数**：
   - 每趟归并可以合并的归并段数：$\lfloor \frac{M}{b_b} \rfloor - 1 = 10239$ 个归并段。
3. **归并趟数**：
   - 由于 $1024 < 10239$，所以只需要 1 趟归并。

情况二：$b_b = 100$

1. **归并段数量**：
   - 第一阶段创建的归并段数量：$\lceil \frac{b_r}{M} \rceil = 1024$ 个归并段。
2. **每趟归并参数**：
   - 每趟归并可以合并的归并段数：$\lfloor \frac{M}{b_b} \rfloor - 1 = 101$ 个归并段。
3. **归并趟数**：
   - $\log_{101}(1024) \approx 1.5$，取上界为 2 趟。

c. 假设使用闪存存储设备代替磁盘,其延迟为 20 微秒,传输速率为每秒 400 MB。在这种设置中,重新计算在 b_b = 1 和 b_b= 100 的情况下,对关系进行排序的成本,以秒为单位。

- 每块传输时间：$4KB/400MB/s = 10.24$微秒
- 寻道时间：20微秒

情况一：$b_b = 1$

1. **块传输次数**：31,457,280次
2. **寻道次数**：10,487,808次

- **传输时间**：$31,457,280 \times 10.24 \times 10^{-6} = 322.122$秒
- **寻道时间**：$10,487,808 \times 20 \times 10^{-6} = 209.756$秒
- **总时间**：$322.122 + 209.756 = 531.878$秒

情况二：$b_b = 100$

1. **块传输次数**：52,428,800次
2. **寻道次数**：316,622次

- **传输时间**：$52,428,800 \times 10.24 \times 10^{-6} = 537.392$秒
- **寻道时间**：$316,622 \times 20 \times 10^{-6} = 6.332$秒
- **总时间**：$537.392 + 6.332 = 543.724$秒

ch17

1

> 说明ACID分别指的是什么,并说明其在数据库系统中的重要性。

原子性（Atomicity）：

- **定义**：原子性确保事务中的所有操作要么全部成功执行，要么全部不执行。如果事务中的任何一个操作失败，整个事务都会回滚到事务开始之前的状态。
- **重要性**：原子性保证了数据库的一致性。即使在系统崩溃或发生错误的情况下，数据库也不会处于不一致的状态。

一致性（Consistency）：

- **定义**：一致性确保事务在执行前后，数据库的状态始终符合预定义的规则（如约束、触发器、级联等）。事务的执行不会违反数据库的完整性约束。
- **重要性**：一致性确保了数据库的正确性。无论事务是否成功，数据库的状态都应该是有效的，符合所有定义的规则。

隔离性（Isolation）：

- **定义**：隔离性确保并发执行的多个事务彼此隔离，一个事务的执行不会受到其他事务的干扰。每个事务都感觉不到系统中有其他事务在并发执行。
- **重要性**：隔离性防止了并发事务之间的相互干扰，避免了数据不一致的问题，如脏读、不可重复读和幻读。

持久性（Durability）：

- **定义**：持久性确保一旦事务成功提交，其对数据库的更改将永久保存，即使系统发生故障（如断电、崩溃等），这些更改也不会丢失。
- **重要性**：持久性保证了数据的可靠性。一旦事务提交，数据就不会因为系统故障而丢失，确保了数据的长期可用性。

2

> 请给出一个包含两个事务的可串行化调度的示例,使得事务提交的顺序与串行化顺序不同。

示例：两个事务的可串行化调度，提交顺序与串行化顺序不同

假设有两个事务 T1 和 T2，操作账户 A 和 B 的余额。

- **事务 T1 (从 A 转账 100 到 B):**
    1. 读 A 的余额
    2. 读 B 的余额

3. A = A - 100
4. B = B + 100
5. 提交

- **事务 T2 (从 B 转账 50 到 A):**
  1. 读 B 的余额
  2. 读 A 的余额
  3. B = B - 50
  4. A = A + 50
  5. 提交

初始值：A = 200，B = 150

串行化顺序：T1 -> T2

- **按 T1 -> T2 执行：**
  - T1 执行后：A = 100，B = 250
  - T2 执行后：A = 150，B = 200

调度步骤（提交顺序 T2 先提交，T1 后提交）：

1. T1 读 A = 200
2. T1 读 B = 150
3. T2 读 B = 150
4. T2 读 A = 200
5. T1 写 A = 100
6. T1 写 B = 250
7. T2 写 B = 200
8. T2 写 A = 150
9. T2 提交
10. T1 提交

最终结果：A = 150，B = 200

结论：

- 虽然 T2 先提交，T1 后提交，但该调度等价于串行化调度 T1 -> T2。
- 因此，该调度是可串行化的，且提交顺序与串行化顺序不同。

3

> 考虑以下两个事务:
> T13:
> read(A);
> read(B);
> if A = 0 then B := B + 1;
> write(B).
> T14:
> read(B);

> read(A);
> if B = 0 then A := A + 1;
> write(A).
> 现在的一致性要求是 A = 0 ∨ B = 0,初始值为 A = B = 0。
> a. 证明涉及这两个事务的每个串行执行都保持数据库的一致性。
> b. 设计一个 T13 和 T14 的并发执行,产生一个不可串行化的调度。
> c. 是否存在 T13 和 T14 的并发执行产生可串行化的调度?

a

初始状态： A = 0，B = 0。

串行执行顺序1: T13 → T14

1. **执行T13:**
   - 读A = 0，读B = 0。
   - 因为A = 0，所以执行B := B + 1，即B = 1。
   - 写回B = 1。
   - 现在，数据库状态是A = 0，B = 1，满足A = 0 ∨ B = 0（A = 0）。
2. **执行T14:**
   - 读B = 1，读A = 0。
   - 因为B ≠ 0，不执行A := A + 1。
   - 写回A = 0。
   - 最终状态是A = 0，B = 1，仍然满足A = 0 ∨ B = 0。

串行执行顺序2: T14 → T13

1. **执行T14:**
   - 读B = 0，读A = 0。
   - 因为B = 0，所以执行A := A + 1，即A = 1。
   - 写回A = 1。
   - 现在，数据库状态是A = 1，B = 0，满足A = 0 ∨ B = 0（B = 0）。
2. **执行T13:**
   - 读A = 1，读B = 0。
   - 因为A ≠ 0，不执行B := B + 1。
   - 写回B = 0。
   - 最终状态是A = 1，B = 0，仍然满足A = 0 ∨ B = 0。

无论是T13先执行还是T14先执行，最终数据库状态始终满足一致性要求A = 0 ∨ B = 0。因此，任何串行执行顺序都能保持数据库的一致性。

b

初始状态：A = 0，B = 0

并发调度顺序：

1. T13读A = 0

2. T14读B = 0
3. T13读B = 0
4. T14读A = 0
5. T13执行B := B + 1，即B = 1
6. T14执行A := A + 1，即A = 1
7. T13写B = 1
8. T14写A = 1

A = 1，B = 1

根据(a)可知，没有任何串行调度可以产生A = 1，B = 1的结果。因此，上述并发调度是不可串行化的。

C

如果并发状态下，恰好T14读在T13写之后，或者T13读在T14写之后则可以。

调度1：

1. T13读A = 0。
2. T13读B = 0。
3. T13执行B := B + 1，即B = 1。
4. T13写B = 1。
5. T14读B = 1。
6. T14读A = 0。
7. T14不执行A := A + 1。
8. T14写A = 0。

最终状态：A = 0，B = 1，与T13 → T14的串行调度结果相同。

调度2：

1. T14读B = 0。
2. T14读A = 0。
3. T14执行A := A + 1，即A = 1。
4. T14写A = 1。
5. T13读A = 1。
6. T13读B = 0。
7. T13不执行B := B + 1。
8. T13写B = 0。

最终状态：A = 1，B = 0，与T14 → T13的串行调度结果相同。