Exercise 1

March 9, 2023

1. In a hard disk drive (HDD), the average seek time is 12 ms, the rotation delay is 4 ms, and the transfer rate is 4MB/sec. For simplicity, we assume in this question only 1MB equals 1000KB.

   (a) What is the seek time delay?

   (b) What is the rotation delay?

   (c) What will be the disk access time for a transfer size of 8MB? What will be the disk access time for a transfer size of 8KB?

   (d) In a solid state drive, what will be the disk access time for a transfer size of 8MB when the transfer rate is 4MB/sec? Is an SSD faster than an HDD for the same amount of data transfer (Assuming the base sequential data transfer rates are the same for the given two drives.)? Why?

Solution:

(a) The seek time delay/seek latency is the period that the head of the actuator arm moves from a position to a required track.

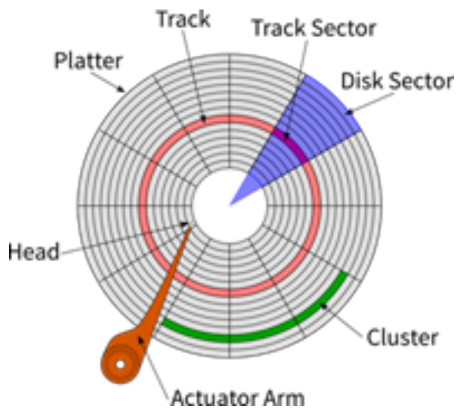


Figure 1: Components of a Hard Disk Drive (HDD).

(b) The rotation delay/rotation latency is the waiting period that the rotation of the disk brings the required track sector to the head of the actuator arm.

(c) Disk access time for 8MB:

$$seektime + rotationtime + \frac{transferlength}{Bandwidth}(ms) \tag{1}$$

$$\longrightarrow 12 \quad + \quad 4 \quad + \quad \frac{8}{4} \times 1000(ms) \quad = \quad 2016ms$$

Disk access time for 8kB:

$$seektime + rotationtime + \frac{transferlength}{Bandwidth}(ms) \tag{2}$$

$$\longrightarrow 12 \quad + \quad 4 \quad + \quad \frac{8}{4 \times 1000} \times 1000(ms) \quad = \quad 18ms$$

Comments: A comparison of the two cases highlights that sequentially reading large data pays off as seek time is buried under a lot of transfer time. For example, in the first case, seek time is only 0.6% of the total time while nearly all the time is spent on transferring data. In the second case, seek time is 66.7% of the total time while only a small fraction of the time is spent on data transfer.

(d) Disk access time of SSD:

$$\frac{transferlength}{Bandwidth} = \frac{8}{4}(s) = 2(s) \tag{3}$$

Unlike an HDD, an SSD does not have any moving parts. Hence there is no rotation delay or seek delay in an SSD. Therefore, the speed of SSDs is higher than that of HDDs given the assumptions in the question.

2. There are two different machines where machine A has a smaller cache with an average 50% cache hit ratio (H) and the other machine (machine B) has a much larger cache with an average 90% cache hit ratio. However, the memory access time of machine A is 100C and the memory access time of machine B is 400C (i.e., memory access in machine A is faster than memory access in machine B), where C is the cache access time. Which machine has an overall faster effective memory access time?

Solution:
Effective memory access time of A:

$$0.5 \times C + (1 - 0.5) \times 100C = 50.5C \tag{4}$$

Effective memory access time of B:

$$0.9 \times C + (1 - 0.9) \times 400C = 40.9C \tag{5}$$

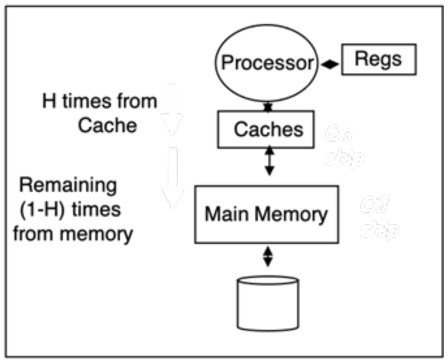


Figure 2: Hierarchical Memory Structure.

3. Discuss the pros and cons of different database types that we saw in class, i.e., RDBs, Simple File-based, OO, etc.

Solution:

- Simple file
  - Pros: Fast
  - Cons: less reliable, hard to maintain
- RDBS
  - Pros: Very reliable, fast
  - Cons: Can be slow for some applications
- Object Oriented
  - Pros: Reliable, can handle complex data
  - Cons: Slow, not much development has been done on this type in real life
- NoSQL
  - Pros: Flexible, scale linearly
  - Cons: Can cause consistency issues

4. Discuss example applications of different types of NoSQL databases.

Solution:

- *Applications for key-value databases*: Suitable if the dataset does not need a complex relational table type of structure but can be expressed with simple key-value pairs. The simple structure allows faster insertion and search and scales quickly. For example, a shopping cart database in an e-commerce site.
- *Applications for document storages*: Well suited when different kinds of documents do not always have the same structure/sections. For example – a database of news articles.
- *Applications for column-based NoSQL databases*: They are used in data analysis applications/tasks.
- *Applications for graph databases* : Well suited for storing connection data such as social network connections and spatial data.

Exercise 2

March 20, 2023

1. Discuss the advantages and disadvantages of different database architectures such as centralized, distributed, P2P, etc as we saw in class for different application scenarios.

   Solution:

   (a) Centralised –
   - **Pros**: Suitable for simple applications, easy to manage.
   - **Cons**: May not scale well.

   (b) Distributed -
   - **Pros**: Scalable, suitable for large applications and applications that need data access from different physical locations.
   - **Cons**: System administration and crash recovery are difficult. These systems usually have some data inconsistency issues.

   (c) WWW -
   - **Pros**: Very convenient to access and share data.
   - **Cons**: Has security issues, no guarantee on availability or consistency. Extreme levels of administration issues. Ultimately did not take off because of this reason.

   (d) Grid -
   - **Pros**: High processing capability as well as access at different locations.
   - **Cons**: Similar issues to distributed databases. Less used nowadays, very similar to distributed systems with administration done locally by each owner.

   (e) P2P -
   - **Pros**: Suitable when the nodes of the network cannot be planned in advance, or some may leave and join frequently.
   - **Cons**: Difficult to design transaction models. Applications are usually limited to simple file sharing.

   (f) Cloud-based Databases -
   - **Pros**: On-demand resources, cost-effective, maintenance done externally by the cloud provider.
   - **Cons**: Has some privacy and confidentiality issues among others – but most trusted providers can address any issues emerging on this type relatively easily, e.g., Amazon, etc.

2. Consider the different scenarios below and discuss which database architecture is the most suitable choice and why:

   - FriendBook is a new startup app that will launch its operation soon. They have only one office without much budget right now, but they are expecting high growth in the scale of millions of users across the globe in a couple of years. Which of the following database architecture is the most suitable choice for this scenario?
     - **Cloud storage**
     - World wide web
     - Distributed database
     - Centralised database

     Solution:
     Comments: Cloud storage allows for data to be stored across multiple servers in data centers, making it easier to scale horizontally as the number of users grows. This type of architecture also provides better reliability and fault tolerance.

- FriendBook is a new social network site that will launch its operation soon. They have offices in many major cities in the USA. They need a database that can handle millions of users across the globe. For preserving privacy and security, they need their own data storage system, which is not shared or owned by any other company. Which of the following database architecture is the most suitable choice for this scenario?
  - Cloud storage
  - World wide web
  - **Distributed database**
  - Centralised database

Solution:

Comments: Unlike the previous scenario, if data is transferred and stored in a 3rd party storage like the cloud, the security is not in the hands of FriendBook (including encryption guarantee, data discloser agreement, etc.). Hence, having the setup of their own distributed database (as they are located across many cities with many users across the globe) is a more suitable solution.

Note: There is no universal truth or final answer on which architecture should be chosen for an application in some cases. The characteristics, advantages, and disadvantages guide us on which one is more suitable over the others and even then some decisions are borderline if the pros – cons are approaching the same level between the two choices.

3. List the ACID properties that we mentioned in the introduction part of the subject. Then discuss the following two scenarios using a transaction that has the following steps:

   (a) read object $O1$ into X.
   (b) read object $O2$ into Y.
   (c) set $X \longleftarrow X + Y$
   (d) set $O1 \longleftarrow X$
   (e) set $O2 \longleftarrow 0$
   (f) set $O3 \longleftarrow X/2$

   Now, let us assume that during execution there was a failure at step 6. Is atomicity achieved in the following two scenarios?

   - **Scenario A**: The transaction rolled back to the start of step 5 then ended.
   - **Scenario B**: The transaction rolled back to the start of step 4 then ended.

   Solution:

   ACID properties are mentioned in Slide 14 of lecture 3.

   In Scenario A, atomicity is not achieved because the change made in step 4 was not undone. In Scenario B, atomicity is achieved because no change is reflected in the database.

   Comments: Note that the first 3 steps in Scenario B do not affect atomicity as they do not make any change to the database. Achieving atomicity prevents updates to the database from occurring partially.

4. Discuss what a query optimizer does and how it works briefly. Then discuss which approach out of the two we saw in class would be more suitable for the following scenarios for the optimizer to take.

   - **Scenario A**: Given a table with 10 million tuples, run the following query:

```
SELECT customer
FROM Table
WHERE spend BETWEEN 100 AND 200
AND birth_year> 2000;
```

- **Scenario B**: Given 5 tables with 1000 tuples in each table, run a query:

```
SELECT T1.name, T2.salary, T3.qualification, T4.phone, T5.leader
FROM Table1 T1
    INNER JOIN Table2 T2 ON T2.id = T1.id
    INNER JOIN Table3 T3 ON T3.id = T1.id
    INNER JOIN Table4 T4 ON T4.id = T1.id
    INNER JOIN Table5 T5 ON T5.department = T1.department
WHERE T1.age > 50;
```

Solution:

Queries are generally converted to Relational Algebra expressions internally first. Then the system tries to create alternate plans and pick the best plan to execute in terms of execution time. There are two general approaches to this. One is searching/enumerating all the plans and choosing the best one. Another approach is using heuristics to choose a plan. (or a combination of two can be done as well.) For Scenario A, the heuristic approach would be suitable due to the simplicity of the query. For Scenario B, the enumerating approach would be more suitable due to the complexity of the query.

5. Review the examples on nested-loop join and block nested-loop join given in the lecture. These are two important algorithms for joining two tables. Discuss and then re-calculate step by step why the latter one can be more efficient.

   Solution:

   After reiterating the examples/calculations, we see the latter one is most efficient because each block in the inner relation is read once for each block in the outer relation (instead of once for each tuple in the outer relation).

Exercise 3

March 27, 2023

1. What indices are suitable if a table is frequently used for finding records based on three criteria: a list of users' name, a range of users' birthday and a spatial region covering users' residence?

   Solution:

   - **Users' name**: Hash index.
   - **Users' birthday**: B+ tree index.
   - **Users' residence**: R-tree index or another spatial index such as a quadtree index.

2. Review the points on indexing with $B+$ trees. Assume a database table has 10,000,000 records and the index is built with a $B+$ tree. The maximum number of children of a node, is denoted as n. How many steps are needed to find a record if $n = 4$? How many steps are needed to find a record if $n = 100$?

   Solution:

   - When $n = 4$, the maximum height of the tree is

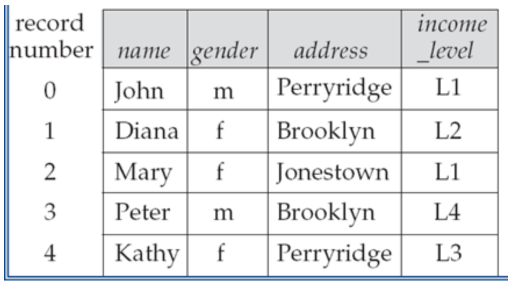
     $$\lceil \log_{\lceil n/2 \rceil}(K) \rceil = \lceil \log_2(10000000) \rceil = 24$$

     Therefore, 24 steps are needed.
   - When $n = 100$, the maximum height of the tree is

     $$\lceil \log_{\lceil n/2 \rceil}(K) \rceil = \lceil \log_{50}(10000000) \rceil = 5$$

     Therefore, 5 steps are needed.

3. Given the database table (shown below), build a bitmap index for the address attribute.



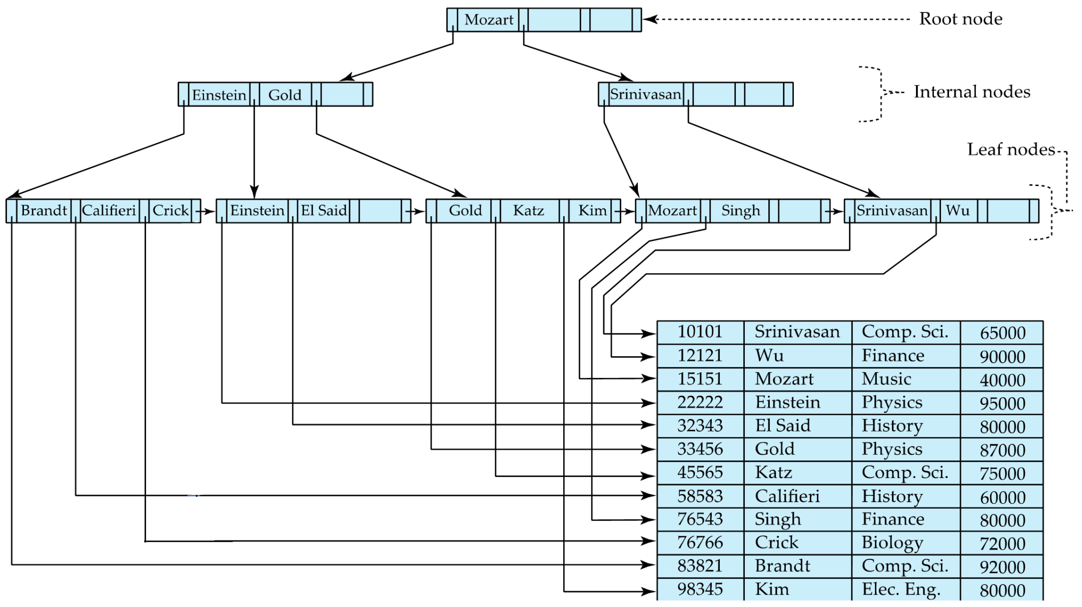
record number	*name*	*gender*	*address*	*income _level*
0	John	m	Perryridge	L1
1	Diana	f	Brooklyn	L2
2	Mary	f	Jonestown	L1
3	Peter	m	Brooklyn	L4
4	Kathy	f	Perryridge	L3

   Solution:

   Bitmap for address:

   - Perryridge : 10001
   - Brooklyn : 01010
   - Jonestown : 00100

4. Given the B+tree example below please search for "Crick" by going through the search algorithm from our lectures step by step.

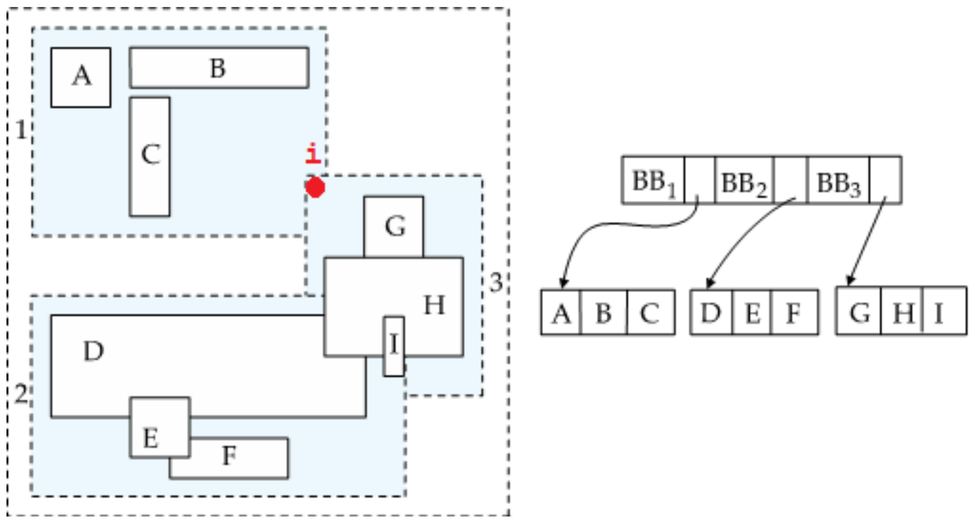


10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Solution:

Algorithm is on Slide 2 Lecture 6.

5. Given the R-tree below please visit the nodes of the R-tree in a best-first manner as discussed in class to find the 1st nearest neighbour of query point "*i*". Is there anything peculiar that you notice while traversing an R-tree?



Solution:

In this traversal we first visit node BB_1 as it overlaps with the query point. And find that object B is the closest to query point i. The issue here is that we cannot stop at this point in the traversal as i overlaps with BB_3 as well so we need to investigate the data there too. We then figure out that G is the closest object overall. Due to overlaps in R-tree branches two or more branches of an R-tree need to be investigated in many query types. In addition, as each internal node represents a bounding box, thus we are not sure about the position of objects in a bounding box which may necessitate that we investigate multiple bounding boxes to determine a nearest neighbour in this case.

Exercise 4

April 3, 2023

1. Discuss why the isolation property of ACID properties will apply to both an Online shopping platform as well as an Online banking system despite that they are different applications dealing with different data.

   Solution:

   Both types of systems may need to serve a large number of customers at any given time. For achieving consistency, both systems require that a transaction does not use the values that have been modified by another uncommitted transaction. A naïve approach to achieve this is handling one customer at a time. But the efficiency of the service would be extremely low with this approach. Therefore, the systems should allow different transactions to run concurrently. At the same time, the changes to the data are made as if the transactions run on a serial schedule, i.e., a transaction runs as if it is the only transaction in the system and does not become aware of other concurrent transactions which is the objective of isolation. Isolation helps with achieving a high level of efficiency while maintaining the consistency of the data that is manipulated.

2. In a nested transaction, a transaction PARENT has three sub-transactions A, B, and C. For each of the following scenarios, answer which of these four transactions' commits can be made durable, and which ones have to be forced to rollback.

   - Scenario 1: Commit by A, B, and C; but PARENT rolls back.
   - Scenario 2: Commit by A, B, C, and PARENT.
   - Scenario 3: Commit by A, B, and PARENT; but C rolls back.

   Solution:

   - Scenario 1: Based on the rollback rule, all four transactions roll back, with no durable commit by any transaction.
   - Scenario 2: All four transactions make durable commits, with no roll back.
   - Scenario 3: Durable commits by A, B, and Parent; roll back by C only.

3. Discuss why using a single process for all the transaction processing monitor services is not a good idea.

   Solution:

   One error in the process can impact all the transactions. This could lead to poor performance as well as monitors of different transactions cannot be distributed either.

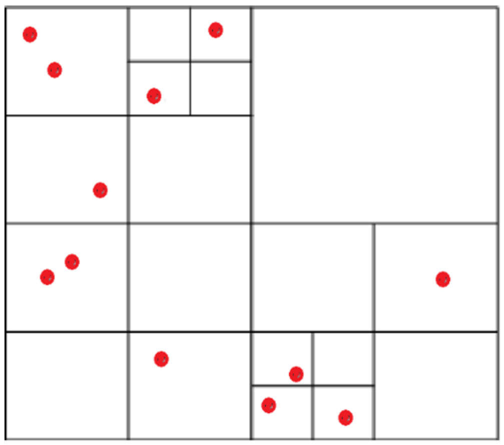
4. We have seen flat transactions in class. The following flat transaction definition creates a problem for the system and shows why flat transactions alone cannot be used. Please discuss the example and explain the associated issue:

```
GiveEndofYearBonus()
   {  real bonus;
    receive(bonus);
    exec sql BEGIN WORK;
        exec SQL UPDATE customer
        set account = account + :bonus
    exec sql COMMIT WORK;
   }
```

   Solution:

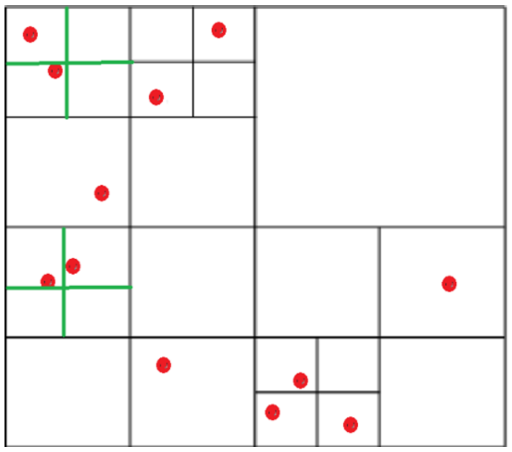
   The customer table here could be a large table with millions of customers. This means this transaction can potentially be doing a lot of work during its execution and take a long time. Any failure in this transaction's execution would mean a lot of work lost. So better not to have this transaction as a flat transaction.

5. In the following quadtree the division of space is based on a rule that each quadrant could only have at most one data point. Based on that we can see that the quadtree is not properly formed. Please finish the space subdivision process for this tree.



Solution:

The quadtree can be finished as follows.



6. What is the value of `AccBalance1` after running the following transaction operations? Discuss why save points are needed.

```
BEGIN WORK
SAVE WORK 1
AccBalance1 = 100;
AccBalance2 = 50;
SAVE WORK 2
AccBalance3 = AccBalance1 + AccBalance2;
AccBalance1 = 0;
AccBalance2 = 0;
SAVE WORK 3
ROLLBACK WORK (2)
AccBalance3 = AccBalance1;
AccBalance1 = 0;
SAVE WORK 4
AccBalance3=AccBalance3 + AccBalance2;
ROLLBACK WORK (2)
AccBalance2 = AccBalance2 + 100;
COMMIT WORK
```

Solution:

The value of AccBalance1 is still 100 as the last rollback action takes the state to save point 2, where the value of AccBalance1 had not been changed to 0 but was changed to 100. Without save points, the transaction needs to be started from scratch every time it needs to rollback.

Exercise 5

April 17, 2023

1. Isolation property in ACID properties states that each transaction should run without being aware or in interference with another transaction in the system. If that is the case, we can run transactions sequentially by locking the whole database itself and adhering to the Isolation property through a big lock per transaction. Review why this may not be an ideal solution.

   Solution:

   This is in fact one type of, simplistic, concurrency control, i.e., making sure that all transactions run in a sequence, i.e., in some order. But then we are not benefiting from the potential use of idle resources such as accessing disk while another transaction is using the CPU. So even under single CPU situations, concurrency can speed up things that we are not doing. In addition, if there is a long-running transaction in the system, holding up every other transaction till that long one finish is going to make the associated company's customers very unhappy. So in short, although we want our executions to be equal to the outcome of a serial execution of a given set of transactions, we do not really want to run them one after the other but rather in concurrency with each other.

2. Given two transactions, per operation of each transaction, we can use locks to make sure concurrent access is done properly to individual objects that are used in both transactions i.e., they are not accessed at the same time. This is after all what the operating systems do, e.g., lock a file while one program is accessing it so others cannot change it at the same time. Give two transactions showing that this is not enough to achieve the isolation property of transactions for RDBMSs.

   Solution:

Answer (A is initially 0 on disk)	Transaction 1 at Process 1	Transaction 2 at Process 2
Operation 1	Lock(A)	
Operation 2	Read(A)	
Operation 3	Unlock(A)	
Operation 4	A = A +100	
Operation 5		Lock(A)
Operation 6		Read(A)
Operation 7		A = A + 50
Operation 8		Write(A)
Operation 9		Unlock(A)/Commit
Operation 10	Lock(A)	
Operation 11	Write(A)	
Operation 12	Unlock(A)/Commit	

   This execution order above is not equal to T1 running first nor to T2 running first and does not obey the Isolation property, although, for each disk access, we first obtained a lock properly. Thus we need something more for proper concurrency control in DBMSs.

3. Given the following two tasks where initial values of B are 50, and C is 0, and operations are labeled as Op1, Op2, etc, per task and we know that Task 1 has done its first operation already, please show all possible concurrent execution orders of the following tasks. Then state which orders make sense and which ones do not. If these were two transactions running concurrently this way, what property of transactions we would be violating with the invalid concurrent execution orders? The tasks are:

   - Task 1:
     - Op1) $B = B - 5$
     - Op2) $C = 50$
   - Task 2:
     - Op1) $B = B + 10$

   Solution:

   The orders are:
   Task 1 Op1, Task 1 Op2, Task 2 Op1; OR
   Task 1 Op1, Task 2 Op1, Task 1 Op2.

   The final values of B and C are 55 and 50 in both. Both are equal as if Task 1 is running first and Task 2 running later. We potentially can contradict the Isolation property but we do not in these cases. The reason is Task 2 observes the change of Task 1 on B (assuming immediate writes to disk) and then there is no other action on Object B by Task 1 again. So it is equal to as if Task 1 ran first and then Task 2 from object B's point of view. From object C, as Task 2 did not touch it we appear to be not concerned with that creating a problem.

4. Now for the same question above replace Op1 and Op2 of Task 1 with $B = B \times 2$. Do your results change? Why? Discuss.

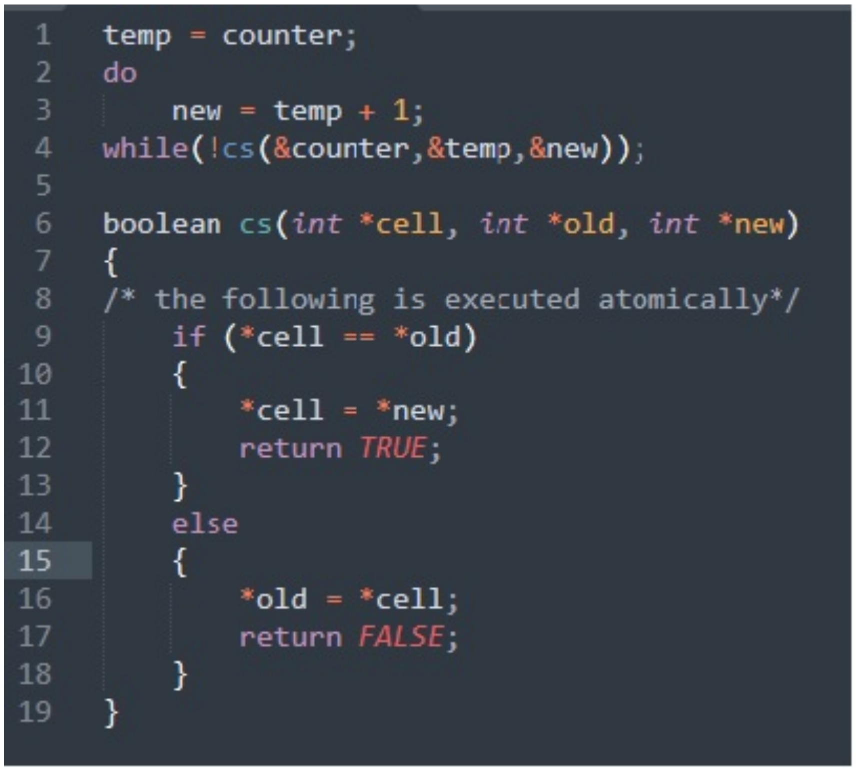
   Solution:

   The results do change. The orders are still:
   Task 1 Op1, Task 1 Op2, Task 2 Op1; OR
   Task 1 Op1, Task 2 Op1, Task 1 Op2.

   But this time the output of the first order is 210 for B and 220 for the second order. And only 210 makes sense because if Task 1 was done first and then Task 2 we would get 210 (and if Task 2 was done first and then Task 1 then we would get 240 and still not 220). We are now not adhering to the Isolation property of transactions and the two transactions running in a peculiar way concurrently and it is the only way we can get 220 and hence they know about each other!

5. Please run the Compare and Swap method we saw in class with two threads running concurrently where the initial value of the counter variable is 0. Show at least one concurrent execution scenario and see that the algorithm works for concurrent access. You should assume that operations do not run in parallel, i.e. at the same time, with each other but in an interleaved fashion for concurrency.

   Solution:



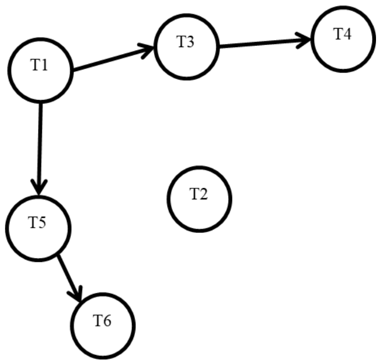
```
 1  temp = counter;
 2  do
 3      new = temp + 1;
 4  while(!cs(&counter,&temp,&new));
 5
 6  boolean cs(int *cell, int *old, int *new)
 7  {
 8  /* the following is executed atomically*/
 9      if (*cell == *old)
10      {
11          *cell = *new;
12          return TRUE;
13      }
14      else
15      {
16          *old = *cell;
17          return FALSE;
18      }
19  }
```

   (a) Thread 1 reads the value of the counter as 0 and stores it in temp (line 1).
   (b) Thread 2 reads the value of the counter as 0 and stores it in temp (line 1).
   (c) Thread 1 calculates new as 1 (line 3).
   (d) Thread 1 calls the function **CS(&counter, &temp, &new)** in line 4.
   (e) For Thread 1, The comparison (*cell==*old) succeeds (in line 9) since the counter still has the value of 0, so the value of the counter is successfully updated to 1 (line 11). A TRUE is returned from the CS function (line 12).
   (f) Thread 1 breaks out of the while loop (line 4), as a TRUE is returned from the CS function.
   (g) Thread 2 also calculates new as 1 (line 3) and calls the function **CS(&counter, &temp, &new)** in line 4.
   (h) However, at this point, the counter has already been updated by Thread 1, so the comparison (*cell==*old) fails (in line 9) for Thread 2. Hence, the value of temp is updated with the new counter value (line 16), and a FALSE is returned (line 17).
   (i) The while loop for Thread 2 will repeat (line 4).
   (j) Eventually Thread 2 will retry (the function CS) with the updated values and it will succeed to increase the counter.

Exercise 7

May 1, 2023

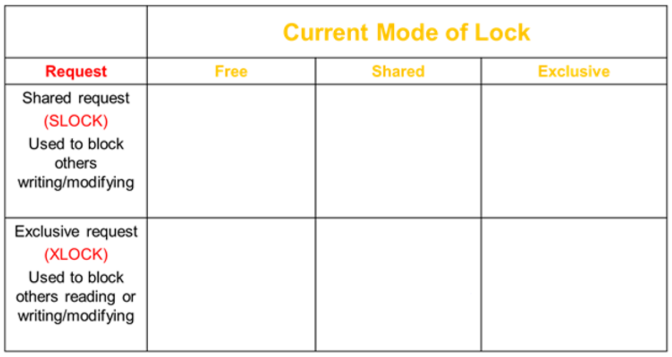
1. Given the graph below for six transactions we can see the dependency between the transactions. If we add a link from T6 to T3, the graph becomes more complicated. Please in that case list After(T6) and Before(T6). Does this link cause a wormhole to appear, in otherwise a set of dependencies where there were no wormholes in the original version of the graph? Instead, if we have a link added from T4 to T1, please list After(T4) and Before(T4). Does this link cause a wormhole? Which one of the two modifications represents dependencies for a history that we can call isolated?



Solution:

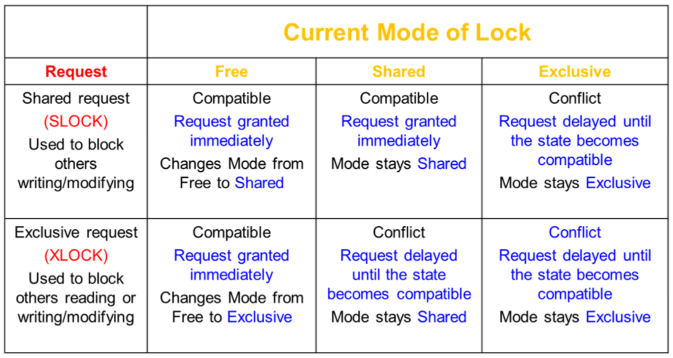
Please review the wormhole theorem first. Then: If we add a link from T6 to T3 then After$(T6) = T3, T4$ and Before $(T6) = T1, T5$. As After(T6) does not have any transactions in its intersection with Before(T6), by adding that link we did not create a cycle or basically, there is no wormhole transaction created. If a cycle was formed then T6 would have been a part of it and would create a wormhole transaction. As the previous state of the graph did not have a wormhole and this new version does not create a wormhole transaction as well, then we can say this modification's dependencies come from an isolated history. On the contrary with the second modification: with a link to T1 from T4, After$(T4) = T1, T3$ and Before $(T4) = T1, T3$. The two sets overlap and thus their intersection is not an empty set and transaction T1 becomes a wormhole for example. Thus, the history that lead to that set of dependencies cannot be an isolated history.

2. Given the following matrix for locking, try to fill the matrix without referring to the lecture slides. Then discuss why we bother having a Slock as well as an Xlock. Why not just have an Xlock in DBMS?



	Current Mode of Lock		
Request	Free	Shared	Exclusive
Shared request (SLOCK) Used to block others writing/modifying			
Exclusive request (XLOCK) Used to block others reading or writing/modifying			

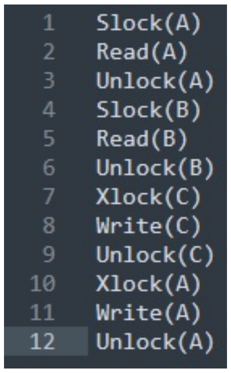
Solution:



	Current Mode of Lock		
Request	**Free**	**Shared**	**Exclusive**
Shared request (SLOCK) Used to block others writing/modifying	Compatible Request granted immediately Changes Mode from Free to Shared	Compatible Request granted immediately Mode stays Shared	Conflict Request delayed until the state becomes compatible Mode stays Exclusive
Exclusive request (XLOCK) Used to block others reading or writing/modifying	Compatible Request granted immediately Changes Mode from Free to Exclusive	Conflict Request delayed until the state becomes compatible Mode stays Shared	Conflict Request delayed until the state becomes compatible Mode stays Exclusive

The reason we have a Slock is to increase the level of concurrency. Else, if we had only one type of lock, Xlock, and for every object access, we would need to use it and two or more transactions that would read objects and not write anything back to the disk would have to wait for each other unnecessarily.

3. Given the operations for a transaction T1 below, please list the lines that this transaction is executing that cannot happen with two-phase locking. Briefly explain.



```
 1   Slock(A)
 2   Read(A)
 3   Unlock(A)
 4   Slock(B)
 5   Read(B)
 6   Unlock(B)
 7   Xlock(C)
 8   Write(C)
 9   Unlock(C)
10   Xlock(A)
11   Write(A)
12   Unlock(A)
```

Solution:

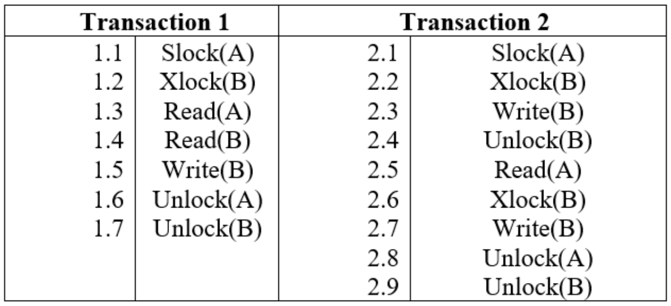
Lines 4, 7, and 10 cannot happen with two-phase locking because the first unlock operation is in Line 3. There cannot be a lock operation after the start of the shrinking phase, i.e., the transaction cannot get further locks after the first unlock operation.

4. Discuss why two-phase locking guarantees serializability.

Solution:

Review lecture 14 slides 6-8. You will see that with two-phase we want to prevent transactions from getting in-between each other and causing cycles in the dependencies basically.

5. Assume the following two transactions start at nearly the same time and there is no other concurrent transaction. The 2nd operation of both transactions is Xlock(B). Is there a potential problem if Transaction 1 performs the operation first? What if Transaction 2 performs the operation first?



Transaction 1		Transaction 2	
1.1	Slock(A)	2.1	Slock(A)
1.2	Xlock(B)	2.2	Xlock(B)
1.3	Read(A)	2.3	Write(B)
1.4	Read(B)	2.4	Unlock(B)
1.5	Write(B)	2.5	Read(A)
1.6	Unlock(A)	2.6	Xlock(B)
1.7	Unlock(B)	2.7	Write(B)
		2.8	Unlock(A)
		2.9	Unlock(B)

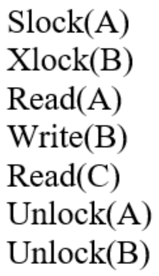
Solution:

If Transaction 1 executes Step 1.2 before Transaction 2 executes Step 2.2, there would be no problem. This is because T1 releases the lock at the end. T2 has to wait till then. However, if Transaction 2 executes Step 2.2 first, Transaction 1 may have a dirty read of B if it tries to run Step 1.2 immediately after Transaction 2 acquires the Xlock on B. This is because when Transaction 2 releases the lock on B (Step 2.4), Transaction 1 will be granted the Xlock on B (Step 1.2). After that, Transaction 1 will read B (Step 1.4). After Transaction 1 completes, Transaction 2 will acquire another Xlock on B (Step 2.6) and then modify the object. In other words, the reading of B by Transaction 1 happens between two writes of B by Transaction 2.

Exercise 8

May 8, 2023

1. Review two-phase locking and the degrees of isolation slides. What degree of isolation does the following transaction provide?

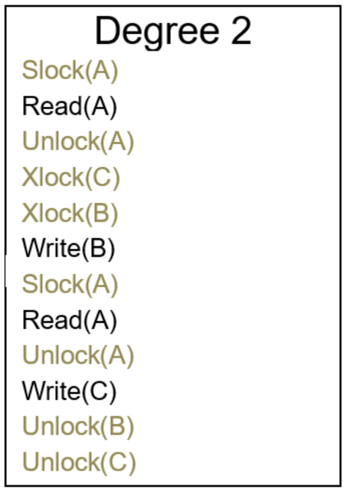


Slock(A)
Xlock(B)
Read(A)
Write(B)
Read(C)
Unlock(A)
Unlock(B)

Solution:

Degree 1 as there is one read operation 'Read(C)' without taking any lock. The only write operation has an exclusive lock associated with it. The transaction is two-phase with respect to exclusive lock.

2. The following operations are given with Degree 2 isolation locking principles in place. Convert the locking sequence to Degree 3.



Degree 2
Slock(A)
Read(A)
Unlock(A)
Xlock(C)
Xlock(B)
Write(B)
Slock(A)
Read(A)
Unlock(A)
Write(C)
Unlock(B)
Unlock(C)

Solution:

```
Degree 3
Slock(A)
Read(A)
Xlock(C)
Xlock(B)
Write(B)
Read(A)
Unlock(A)
Write(C)
Unlock(B)
Unlock(C)
```

3. The following transactions are issued in a system at the same time. Answer for both scenarios.

   (a) **Scenario 1:** When the value of A is 3, which of the following transactions can run concurrently from the beginning till commit (that is, all operations and locks are compatible to run concurrently with another one) and which ones need to be delayed? Please give an explanation for the delayed transactions. Note that, the order of start of transactions can be deducted from the beginning positions of the transactions in the table given.

   (b) **Scenario 2:** When the value of A is 2, which of the following transactions can run concurrently from the beginning till commit (that is, all operations and locks are compatible to run concurrently with another one) and which ones need to be delayed? Please give an explanation for the delayed transactions. Let's assume T1 starts slightly earlier than others for this case. [1]

	T2	T3
	Lock (U,A)	Lock (IX,A)
T1	Read A	Read A
Lock (S,A)	if(A ==3) {	if(A ==3){
Read A	Lock(X,A)	Lock(X,A)
Unlock A	Write A	Write A
	}	}
	Unlock A	Unlock A

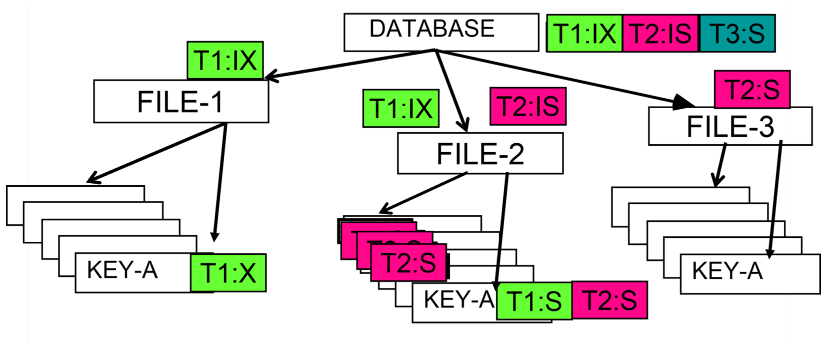
Solution:

Scenario 1: None of the transactions can run concurrently. T2's update lock and T3's IX conflict with each other, and the subsequent X locks for A==3 will conflict. T3's IX lock conflicts with T1's S lock. Although T1's shared lock and T2's update lock are compatible if T1 gets the lock first, but T1 gets the lock later which does not work as the compatibility matrix we saw in class shows. So only one can run at a time while the other transactions must be delayed.

Scenario 2: When A is 2, T2 and T3 will not request exclusive lock. Hence, T1 and T2 can run concurrently as T2's update lock can be granted if T1 gets the shared lock on A first. T1 and T3 cannot run concurrently - one of them must be delayed as the locks are not compatible. T2 and T3 cannot run concurrently - one of them must be delayed as the locks are not compatible.

[1] *There can be different versions of the compatibility matrix. Please use the compatibility matrix from the lecture slides for this exercise.

4. Review the concepts of granular locks then answer the following question. Given the hierarchy of database objects and the corresponding granular locks in the following picture, which transactions can run if the transactions arrive in the order T1-T2-T3? What if the order is T3-T2-T1? Note that locks from the same transaction are in the same colour. We assume that the transactions need to take the locks when they start to run.



Solution:

If the order of the arrival of transactions is T1-T2-T3, then T1 and T2 can run in parallel while T3 waits. This is because T1's IX lock at the root node is not compatible with T3's S lock at the same node.

If the order is T3-T2-T1, then T3 and T2 can run while T1 waits. This is due to a similar reason as above. This example shows that the order of transactions can be a deciding factor in the set of parallel-running transactions. We should also note that granular locks can lead to the delay of transactions at any level in the hierarchy below the root node, e.g., a transaction may need to wait for a lock at the FILE-3 node or a KEY-A node due to lock compatibility issues.

5. With two-phase locking we have already seen a successful strategy that will solve concurrency problems for DBMSs. Then discuss why someone may want to invent something like Optimistic Concurrency control in addition to that locking mechanism.

Solution:

Two-phase locking or in general locking assumes the worst, i.e. there are many updates in the system and most of them will lead to conflicts in access to objects. This means there is a good rationale to pay the overhead of locking and stop problems from occurring in the first place. But what if the DBMS is one such that people tend to work on different parts of data, or most of the operations are read operations, and as such there aren't many conflicts at all? Then there is no need to pay the overhead of lock management but rather it may be better to allow transactions to run freely and have a simple check when they finish whether there was any conflict with concurrent transactions. Most of the time there will not be so one will get increased concurrency with less overheads. Obviously, the reverse is also true, i.e., if there were really many conflicting writes then doing optimistic concurrency control means many problems would be observed only after running the transactions and a lot of work will need to be wasted to preserve the consistency of the data. So there is no clear answer but depending on the situation a strategy may be good or bad.

Exercise 9

May 15, 2023

1. Let's define a simple one-phase atomic commit protocol for DBMS. In this protocol, the coordinator tells the participants whether to commit or abort at the end and that is it, for this protocol, no other phase, etc is deemed necessary by the implementer company: what is the problem with such a protocol: describe three possible scenarios for analysis where one-phase atomic commit protocol would not work.

   Solution:

   - Scenario 1: Two participants receive a commit command but cannot commit as they are in a deadlock. Even when we resolve the deadlock due to an abort still the other should not be able to commit by itself probably.
   - Scenario 2: A participant does not receive the command from the coordinator due to a network connection problem. There is no fallback strategy defined.
   - Scenario 3: A participant decides to abort but the coordinator wants it to commit. No option is left and no potential way forward depending on the problem of the participant.

2. Assume a two-phase commit involves a coordinator and three participants, P1, P2 and P3. What would happen in the following scenarios?

   - Scenario 1: The coordinator crashed after receiving a 'yes' vote from all participants.
   - Scenario 2: P1 and P2 voted yes and P3 voted no.
   - Scenario 3: P2 crashed when it was about to send a vote message to the coordinator.

   Solution:

   - Scenario 1: The main fallback, in this case, is: The participants will send getDecision messages to the coordinator individually because they cannot receive further instruction from the coordinator.
   - Scenario 2: The coordinator will send doAbort messages to all participants. To achieve atomicity, all the participants should commit or none will commit.
   - Scenario 3: Since the coordinator cannot receive the vote from P2 within the timeout period, it will abort the transaction by sending doAbort messages to all participants.

3. Discuss how timestamp strategy differs in concurrency control w.r.t its original form in distributed transactions.

   Solution:

   Each transaction is assigned a globally unique timestamp by the coordinator. The order of the transactions' timestamps determines which transaction has the priority to access objects at distributed servers. The timestamp is passed with each object access. This assures that if transaction X is before transaction Y based on the timestamp ordering, X will be before Y in their conflicting access to objects at all servers.

4. What is the mean time to failure value for different RAID systems (e.g., RAID 0 with 2 disks and RAID 2 with 2 disks, RAID 1 with 2 disks, RAID 1 with 3 disks, RAID 3 with 3 disks, RAID 6 with 5 disks), please review/discuss.

   Solution:

   Let's label the probability of one disk failure as p, and the mean time to failure of one individual disk is MTTF. P is between 0 and 1.

   RAID 0 with 2 disks and RAID 2 with 2 disks – The system fails if one of the disks fails. The probability that one of the two disks fails (disk A or disk B) is $p + p = 2p$. So, as failure probability doubles mean time to failure is halved = $\frac{1}{2} \times MTTF$.

   RAID 1 with 2 disks - The system fails if both of the disks fail at the same time. The probability that both disks fail (disk A and disk B) is $p * p = p^2$ as p is between 0 and 1, mean time to failure of the system will increase to $MTTF^2$.

RAID 1 with 3 disks - The system fails if three of the disks fail at the same time. The probability that all disks fail (disk A and disk B and disk C) is $p * p * p = p^3$. So, the mean time to failure of the system is accordingly $MTTF^3$.

RAID 3 with 3 disks- The system fails if 2 of the 3 disks fail at the same time. The probability that 2 disks fail is $p * p = p^2$. There are 3 different possible combinations of 2 disk failures (A, B; or A, C; or B, C), so the probability that any of the 2 disks out of these 3 disks fail is $3p^2$. The mean time to failure of the system is $\frac{1}{3} \times MTTF^2$.

RAID 6 with 5 disks - The system fails if 3 out of the 5 disks fail at the same time. The probability that 3 disks fail is $p * p * p = p^3$. There are 10 different possible combinations of 3 disks failures out of 5 disks (A,B,C; or A,B,D; or A,B,E; A,C,D; or A,C,E; or A,D,E; or B,C,D; or B,C,E; or B,D,E; or C,D,E), so the probability that any of the 3 disks out of these 5 disks fail is $10p^3$. Mean time to failure of the system is then $\frac{1}{10} \times MTTF^3$.

5. In a Failvote system, in which of the following cases can we accept an action?

Total number of devices	Number of agreeing devices	Accept?
10	6	
10	5	
10	4	
5	3	
5	2	

Solution:

Total number of devices	Number of agreeing devices	Accept?
10	6	y
10	5	n
10	4	n
5	3	y
5	2	N

6. In a Failfast system, in which of the following cases can we accept an action?

Total number of devices	Number of working devices	Number of agreeing devices	Accept?
10	6	4	
10	6	3	
10	5	3	
5	5	3	
5	4	2	
5	2	2	
5	1	-	

Solution:

Total number of devices	Number of working devices	Number of agreeing devices	Accept?
10	6	4	y
10	6	3	n
10	5	3	y
5	5	3	y
5	4	2	n
5	2	2	y
5	1	-	n

Exercise 10

May 22, 2023

1. Which of the following RAID configurations that we saw in class has the lowest disk space utilization? Your answer needs to have explanations with calculations for each case.

   - (1) RAID 0 with 2 disks
   - (2) RAID 1 with 2 disks
   - (3) RAID 3 with 3 disks

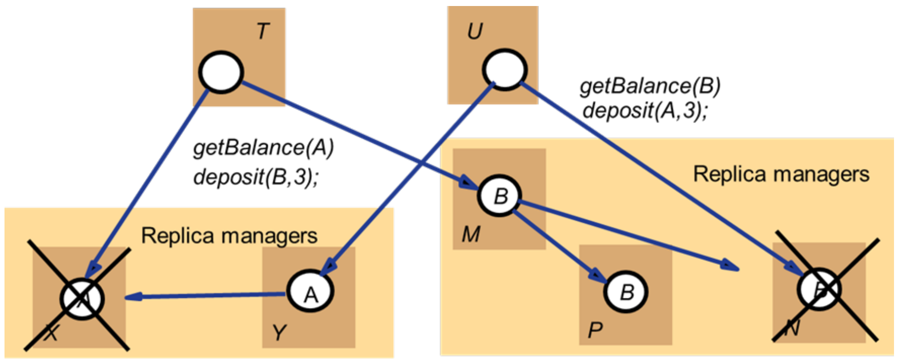
   Where does this lack of utilization of space go, i.e., where we can use such a configuration as it has some benefits gained due to the loss of space utilization?

   Solution:

   - In case 1, the space utilization is 100% because the two disks store contiguous blocks of a file in RAID 0.
   - In case 2, the space utilization is 50% because RAID 1 uses mirroring. MTTF increases so for cases where a disk can fail easily this is good. The system operates even when a disk fails.
   - In case 3, the space utilization is (3-1)/3=66.7 because RAID 3 uses one disk for storing parity data.

   Case 2 has the lowest disk space utilization with an explanation as given underlined above as a part of the answer.

2. Given the two following transactions T and U that run on replica managers X, Y, M, P, and N, we have seen in class a simple version of the Available Copies strategy that would not work. First review the problem that would occur if X and N were to crash during execution. Then state the solution that we have discussed in class. Last but not least, discuss what would happen under the final solution, if rather than X and N becoming unavailable we have the following scenario: If Y were to become unavailable during the execution and right after U accessed A at Y, but X and N do not fail, rest of the assumptions of this scenario is the same as we discussed in class.



   Solution:

   For the first discussion part, please refer to lecture 17 slides 12 to 17. Please review them and see that with the new rule Available Copies stops dangerous executions from happening. Now for the case where Y fails instead of others. The scenario is different. We wish that: U can lock Y and is delayed at X as A is locked by T there, similar to the previous scenario. On M, P, and N either U gets the lock first and T waits, in which case there is a deadlock which will be resolved by a timer, or T locks on N as well and U also waits there and T finishes first and N later. In any case, as you see there is no problem as T and U see each other. Nevertheless, the executions above will not occur with the final solution, as seeing Y is gone U will self-terminate based on the new rule for our final solution. As you can see, the implementation of the final strategy we have seen, and in fact many strategies in DBMSs, are only approximations and in general pessimistic ones so that they guarantee proper executions but sometimes terminate transactions that would not really cause harm.

3. What are the key differences in logging-based recovery under deferred database modification vs immediate database modification scheme?

   Solution:

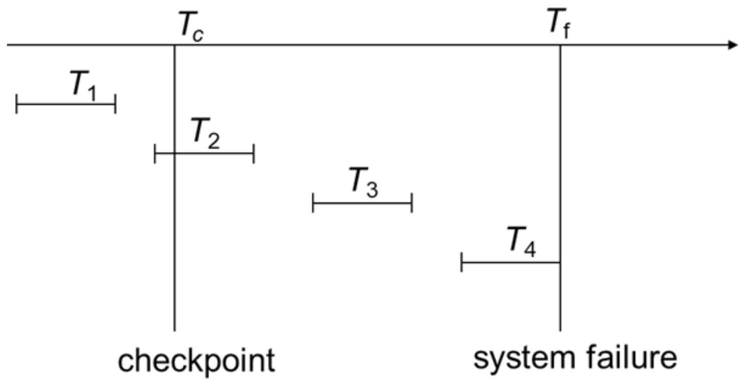
   In the deferred database scheme, writes are deferred. Due to this, we only need to consider redos and there is no undoing a transaction when a failure occurs. Therefore there is no reason to hold the old value in the log for an object along with the new value. In the immediate modification version, as writes are not necessarily delayed, there may be work to do for transactions that wrote objects but did not commit. In particular, we need to undo as well as redo. For this to happen, we need to keep the old value of objects in logging as well as the new value. For more information read slides 12 to 18 from lecture 19.

4. If I were to design a new RAID level, say RAID level H, with the following rules, what would be the storage utilization as well as the MTTF of this scheme? What are the advantages and disadvantages, please discuss. In RAID level H (for hypothetical): we have one data disk that is replicated to a mirror disk in a funny way, only every other block is replicated! Assume the mirror disk is the same size as the original.

   Solution:

   In terms of storage utilization, we still have halved the utilization as disk sizes are the same. The throughput is increased only 50% of the time as half the data is replicated but writes have slowed down as well and 50% of the time as well. Interestingly, the system only continues to fully work if the original disk is alive. So MTTF did not change ironically. Unless we renew the definition of failure or having partial data is of some use this is the case. One needs to have block-level MTTF (such as considering bad sectors) rather than full disk failure to find new values of MTTF.

5. In the following figure the first vertical line Tc denotes the point where checkpointing was done and the second on the right, Tf, is where a system crash occurs. Please discuss what would change if the checkpointing was done right at the beginning of each transaction instead of the following case in the figure.



   Solution:

   If we were to do checkpointing at the beginning of each transaction, then after a system failure there would be much less to do during recovery time. This is because unlike the case above there would be much fewer transactions to consider for redo/undo. T3 for example would not be considered if at the beginning of T4, we did a checkpoint. Thus, with more checkpointing recovery becomes fast. Having said this, this does not come for free. There would be many output operations to the disk for each checkpoint as well as entries to the log regarding checkpointing. Thus, the cost is during transaction processing there would be more overheads. One needs to consider the frequency of checkpointing carefully. There is no one good frequency and it is deployment dependent. For systems where immediate recovery is of utmost importance then frequency can be increased. For systems where transaction processing should be done fast, but recovery can take a long time, less checkpointing should be considered.

6. DBMS experts have long observed that the recovery of interactive transactions is more difficult to deal with than the recovery of batch transactions. Explain why this may be the case based on discussions we had on user interactions so far and suggest a solution. Consider an automatic teller machine transaction interacting with a user in which cash is withdrawn as an example.

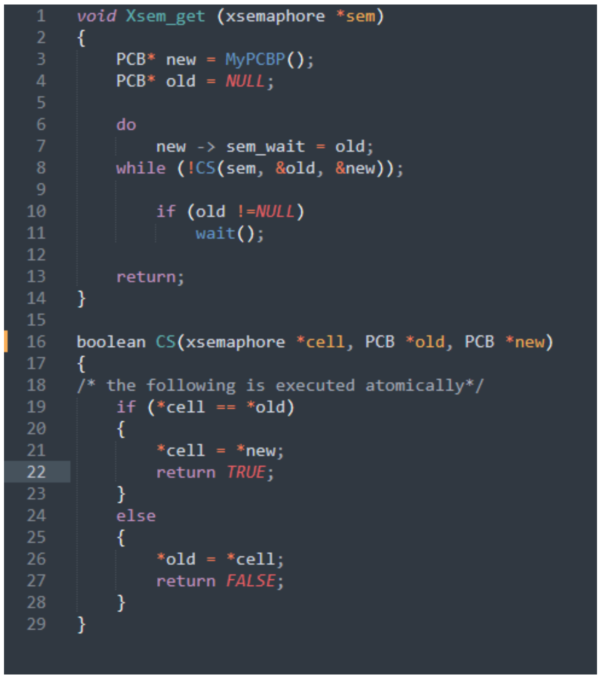
   Solution: Interactive transactions are more difficult to recover from than batch transactions because some actions may be irrevocable. For example, a bank machine may have given money to a customer. The best way to deal with this is to try to do all output statements at the end of the transaction to the user. That way if the transaction aborts in the middle, no harm will be done. Output operations should ideally be done atomically; for example, ATM machines should give out notes as one operation and deliver all the notes together at the end, instead of delivering notes one at a time and putting other operations in the middle for such a transaction. The same principle applies in other cases where there are such interactions involved. If output operations cannot be done atomically, a log of output operations should be maintained and the system should allow performing recovery to be performed manually later by alerting the appropriate organization staff member, for example, putting cash back into the customer account if needed.

Exercise 6

April 24, 2023

1. Given the Xsem_give and Xsem_get functions that we saw in class, assume there are three processes that want to access a critical section of code and use this semaphore concept. The first process starts and executes the Xsem_get function until the last ”return” line of code when the second process is taken on board by the CPU at that time and the first process is temporarily taken out of the CPU. The second process then executes the Xsem_get function. Please step by step explain what the second process does (assuming there is no other process coming into the CPU to consider during this). Then give the line of code in Xsem_get that the second process stops at. Then, at that state, assume a third process comes in, and again step by step show what happens to this new process with Xsem_get. In this question, you can assume the first process is really the very first to request access to that semaphore and did not wake up from a wait() call. Discuss the values of sem, old and new during the execution steps.

   Solution:



```
1   void Xsem_get (xsemaphore *sem)
2   {
3       PCB* new = MyPCBP();
4       PCB* old = NULL;
5
6       do
7           new -> sem_wait = old;
8       while (!CS(sem, &old, &new));
9
10          if (old !=NULL)
11              wait();
12
13      return;
14  }
15
16  boolean CS(xsemaphore *cell, PCB *old, PCB *new)
17  {
18  /* the following is executed atomically*/
19      if (*cell == *old)
20      {
21          *cell = *new;
22          return TRUE;
23      }
24      else
25      {
26          *old = *cell;
27          return FALSE;
28      }
29  }
```

   (a) When the first process is at ’return’ (line 13), it means the process acquired a lock on the semaphore (this is the first process to get the semaphore). The value of sem is now pointing to the first process’s new (MyPCB) value. The value of old is NULL.

   (b) If the second process enters now, it will execute the CS function (line 8). However, as the sem is already pointing to process 1’s PCB, the if condition in the CS function (line 19) will be false, so the old value will be updated with the current sem (within the else condition in line 26), and a False will be returned.

   (c) Hence, process 2 will spin once in the while loop (line 8) as a False was returned.

   (d) In the next iteration, process 2 will be able to break out of the while loop as both cell (sem) and old are now equal (as old got updated to the current sem in the previous iteration).

   (e) However, as the old value is not NULL (process 1 has been moved out of the CPU and couldn’t give up the semaphore), process 2 will call wait (line 11) and keep sleeping.

   (f) At this point, the semaphore wait list contains process 2 (waiting for process 1 to give up the semaphore).

(g) Similarly, if a third process gets CPU time now, it will also eventually call wait() on the semaphore and keep sleeping.

(h) At this point, the semaphore wait list contains both processes 2 and 3 (waiting for process 1).

(i) After a while, when process 1 gets CPU, it will be able to finish its task with the shared resource and will give up the semaphore. It will also wake up the oldest waiting process (process 2), so that process 2 can get the semaphore. At this point, only process 3 will remain in the semaphore wait list.

2. If we use the following comments to lock and unlock access to objects then: Which transactions below are in deadlock if they start around the same time?

T1	T2	T3	T4
Begin	Begin	Begin	Begin
LOCK(C)	LOCK(A)	LOCK(C)	LOCK(B)
Write C	Write A	Write C	Write B
UNLOCK(C)	LOCK(B)	UNLOCK(C)	LOCK(A)
End	Write(B)	End	Write A
	UNLOCK(A)		UNLOCK(A)
	UNLOCK(B)		UNLOCK(B)
	End		End

Solution:

T2 and T4 are in a deadlock as each of them will wait for the other to release a lock while holding a lock that the other needs to acquire to complete.

3. What are the dependencies in the following history (a sequence of tuples in the form (Ti, Oi, Tj))? Draw the dependency graph mapping to this dependency set as well.

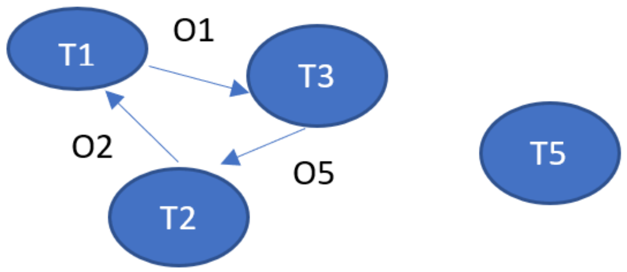
$$H =< (T1, R, O1), (T3, W, O5), (T3, W, O1), (T2, R, O5), (T2, W, O2), (T5, R, O4), (T1, R, O2), (T5, R, O3) >$$

Solution:

Dependencies are given as:

$$DEP(H) = < T1, O1, T3 >, < T3, O5, T2 >, < T2, O2, T1 >$$

We can also build the following dependency graph based on this like the following:



4. Given the solution above for question 3, can we say the history is equal to a serial history. If yes, show one such history. If not, show that there is a wormhole.

Solution:

There exists a cycle in the dependency graph, i.e., there are wormhole transactions (e.g., $T1$ is before and after of $T3$ at the same time). Therefore finding equivalent serial execution is not possible.

5. Repeat questions 3 and 4 with history:

$$H =< (T3, W, O5), (T3, W, O1), (T2, R, O5), (T2, W, O2), (T5, R, O4), (T1, R, O1), (T1, R, O2), (T5, R, O3) >$$

Solution:

Dependencies are given as:

$$DEP(H) = <T3, O5, T2>, <T3, O1, T1>, <T2, O2, T1>$$

This basically reverses the direction of dependency in the previous graph for O1. Thus there is no more cycles in the dependency graph, and hence we are equal to for example the following serial history:

$$H =< (T3, W, O5), (T3, W, O1), (T2, R, O5), (T2, W, O2), (T1, R, O1), (T1, R, O2), (T5, R, O4), (T5, R, O3) >$$

where $T3$ runs before $T2$ which runs before $T1$ and that is before $T5$. The DEP of this is the same as the DEP of the history given in the question.