W4111 - Introduction to Databases Section 02, Fall 2020

Name: Haoran Zhu
UNI: hz2712

I intend to use 1 late day on this homework, which makes the effective deadline of this homework 23-NOV-2020 at 11:59 PM.

Overview

This assignment requires written answers to questions. You may have to copy and paste some SQL statements or scripts into the answer document you submit. You may also have to insert diagrams.

There are 20 questions worth 5 points each. A homework assignment contributes 10 points to your final score. Dividing your score on this assignment by 10 yields the points credited to your final score.

The homework is due 22-NOV-2020 at 11:59 PM.

Question 1

Question: Suppose you have data that should not be lost on disk failure, and the application is write-intensive. What type of disk system would you use to store the data and how you organize the data files on the disks. (Note: Your answer should consider differences between primarily sequential writes and primarily random writes).

Answer:
RAID 1 is preferred when writes are primarily random. RAID 5 is preferred when writes are primarily sequential. Both RAID 1 and RAID 5 offer fault tolerance -- the system will keep functioning when one of the physical disks fail.

RAID 5 has a significant time overhead for random writes as a block write requires 2 block reads and 2 block writes. So RAID 1 is preferred for random writes. In contrast, RAID 5 has much lower time overhead for large sequential writes as the parity block can be computed from the new blocks without any reads. RAID 5 also has lower storage overhead and offers good security, which makes it the better option for large sequential writes. If writes are primarily sequential, we can keep data files sequentially on adjacent cylinders on the disks by allocating consecutive blocks of a file to disk blocks that are consecutively numbered.

Question 2

Question: Both database management systems (DBMS) and operating systems (OS) provide access to files and implement optimizations for file data access. A DBMS provides significantly better performance for data in databases in some scenarios. How does the DBMS do this? What information does it use? How does it optimize access? Provide some examples.

Answer:
First, since DBMS knows the access patterns, it can optimize I/O performance based on access patterns. For example, DBMS may group blocks onto the same sector/cylinder if accessed together/sequentially. Also, DBMS can adjust access orderings (e.g. reading cylinders 1,9,8,2,3 is changed to reading cylinders 1,2,3,8,9). Finally, DBMS can "predict future" according to access patterns and prefetch blocks to speed up data access.

Beside access patterns, DBMS optimizes query. And there are mechanisms such as index that help DBMS to quickly locate target data. For example, if the query involves "R join L" and only R has indexes, DBMS can convert the query into its equivalent form "L join R". In this way, locating data in R is much faster with the help of index. Another example is hash join, which DBMS uses to optimize equi-joins and natural joins.

Question 3

Question: Briefly explain CHS addressing and LBA for disks. Which approach is more common and why? Is it possible to convert a CHS address to an LBA. If yes, how?

Answer:.
Cylinder-head-sector (CHS) address is a method for giving addresses to each physical block of data on a disk. CHS identifies individual sectors on a disk by their position in a track, where the track is determined by the head and cylinder numbers. Logical block addressing (LBA) is a linear addressing method for specifying location of blocks of data on a disk.

LBA is more common than CHS. This is because CHS only supports limited disk capacity. Also, nowadays, disks adopt platters with the same density, which means there are more sectors on the outer track than inner track. This makes it very difficult for CHS addressing to locate a specific sector. Thus, LBA is the more common approach.

It's possible to convert a CHS address to an LBA. The conversion formula is $\mathbf{A = (c \cdot N_{heads} + h) \cdot N_{sectors} + (s - 1)}$ where A is the LBA address, (c, h, s) is the CHS address, N_{heads} is the number of heads on the disk, and $N_{sectors}$ is the max number of sectors per track.

References:

1. https://www.partitionwizard.com/help/what-is-chs.html
2. https://en.wikipedia.org/wiki/Cylinder-head-sector

3. https://en.wikipedia.org/wiki/Logical_block_addressing#CHS_conversion

Question 4

Question: Explain why the allocation of records to blocks in a file affects database-system performance significantly.

Answer:
If related records are allocated to a single block, then we can obtain most if not all desired records in a query with only one disk access. Thus, by appropriate allocation of records to blocks, the number of disk accesses are reduced, which increases the performance of the database system significantly.

References:
1. https://www2.cs.sfu.ca/CourseCentral/354/louie/Chap10_practice_key.pdf

Question 5

Question: Give benefits and disadvantages of variable length record management versus fixed length record management

Answer:
Variable length record management
- Advantages:
    - Memory efficient -- use only as much memory as needed
    - Storage of multiple record types in a file
    - Able to accommodate unusual data, which are not originally planned
- Disadvantages:
    - Difficult to perform operations such as insert, delete, and update (to a different length)
    - Difficult to scan through the data compared to fixed length records

Fixed length record management
- Advantages:
    - Easy and fast access and searching
    - Operations such as insert and delete are also easier
- Disadvantages:
    - Lack of flexibility
    - May lead to memory waste
    - Hard to change a field length

References:
1. http://www.eli.sdsu.edu/courses/spring95/cs596_3/notes/databases/lect10.html

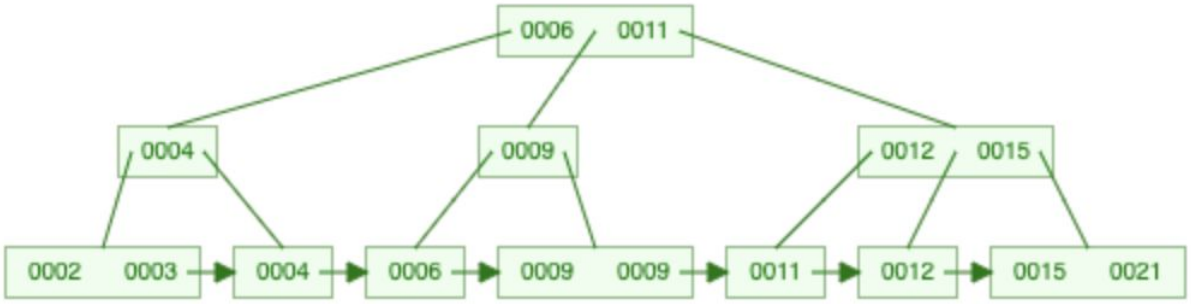
Question 6

Question: Build and draw a B+ tree after inserting the following values. Assume the maximum degree of the B+ tree is 3.

Values: 3, 11, 12, 9, 4, 6, 21, 9, 15, 2

You may use one of the online simulators.

Answer:



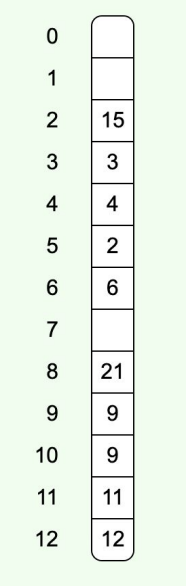
Question 7

Question: Perform the same insertions in the same order for a hash index. Assume that:

1. The size of the hash table is 13.
2. The hash function is simple modulo.
3. The size of a bucket is one entry.
4. The size of each bucket is one value.
5. The index algorithm uses linear probing to resolve conflicts/duplicates.

You may use one of the online simulators.

Answer:



Question 8

Question: When is it preferable to use a dense index rather than a sparse index? Explain your answer.

Answer:
Dense index is preferred when records are not sequentially ordered on the search-key of the index. Dense index is also preferred when the index file is fairly small compared to memory size so that the index file can be easily loaded into memory.

References:
1. http://web.cs.ucla.edu/classes/fall04/cs143/solutions/chap12a.pdf

Question 9

Question: Since indexes improve search/lookup performance, why not create an index on every combination of columns in a table?

Answer:
- Each index created means more disk space is consumed. It's a memory waste if that index is not queried on frequently.
- Creating a new index increases backup size and requires maintenance. Creating unnecessary indexes may cause huge maintenance/recovery overhead.
- Index adds overhead on update of the table. Also, too many index files may not fit in RAM. Thus, creating an index for every combination of columns causes huge and unnecessary overhead.

References:

1. https://www.sqlskills.com/blogs/kimberly/indexes-just-because-you-can-doesnt-mean-you-should/

Question 10

Question: Consider the table below. Add indexes that you think are appropriate for the table and explain your choices. You may use MySQL Workbench to add the indexes. Paste the resulting create statement in the answer section.

Choosing indexes is not possible without understanding use cases/access patterns. Define five use cases and the index you define to support the use case. See the answer section for an example.

```
CREATE TABLE `customers` (
  `id` int NOT NULL AUTO_INCREMENT,
  `company` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `last_name` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `first_name` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `email_address` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `job_title` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `business_phone` varchar(25) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `home_phone` varchar(25) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `mobile_phone` varchar(25) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `fax_number` varchar(25) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `address` longtext COLLATE utf8mb4_unicode_ci,
  `city` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `state_province` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `zip_postal_code` varchar(15) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `country_region` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`));
```

Sample Answer:

Use Case 1: A user wants to be able to find a customer(s) by country_region; country_region and state_province; country_region, state_province, city; just by city.

```
ALTER TABLE `world`.`customers`
ADD INDEX `location_idx` (`country_region` ASC, `state_province` ASC, `city` ASC) VISIBLE;
;
```

```
ALTER TABLE `world`.`customers`
ADD INDEX `city_idx` (`city` ASC) VISIBLE;
;
```

Your Answer:
Use Case 1: A user wants to be able to find a customer(s) by last_name; last_name and first_name as it's common to search customers by their name

```
ALTER TABLE `COMS4111HW3`.`customers`
ADD INDEX `last_name_idx` (`last_name` ASC, `first_name` ASC) VISIBLE;
;
```

Use Case 2: A user wants to be able to find a customer(s) by email_address as it's common to search customers by email

```
ALTER TABLE `COMS4111HW3`.`customers`
ADD INDEX `email_idx` (`email_address` ASC) VISIBLE;
;
```

Use Case 3: A user wants to be able to find a customer(s) by business_phone as it's common to search customers by business phone number

```
ALTER TABLE `COMS4111HW3`.`customers`
ADD INDEX `bs_phone_idx` (`business_phone` ASC) VISIBLE;
;
```

Use Case 4: A user wants to be able to find a customer(s) by country_region; country_region and state_province; country_region, state_province, city as it's common to search customers by the place where they live

```
ALTER TABLE `COMS4111HW3`.`customers`
ADD INDEX `location_idx` (`country_region` ASC, `state_province` ASC, `city` ASC) VISIBLE;
;
```

Use Case 5: A user wants to be able to find a customer(s) by job_title; job_title and last_name as it's common to search customers by their job title. Last name may be used to further distinguish customers with the same job title.

```
ALTER TABLE `COMS4111HW3`.`customers`
ADD INDEX `job_idx` (`job_title` ASC, `last_name` ASC) VISIBLE;
;
```

Question 11

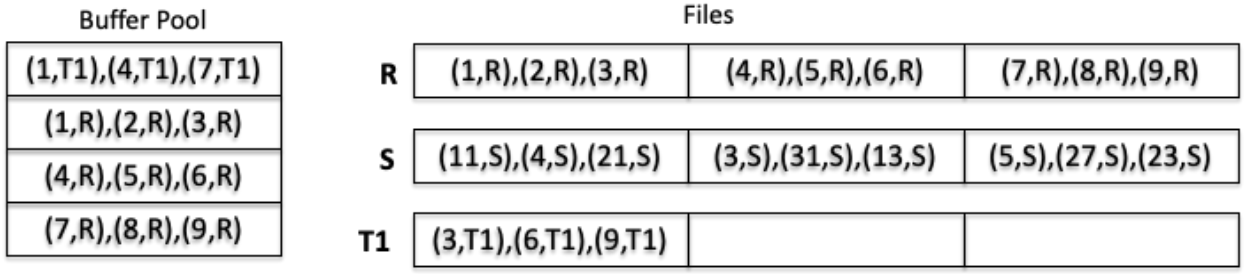
Question: Assume that:

1. The query processing engine can use four blocks in the buffer pool to hold disk blocks.
2. The records are fixed size, and each block contains 3 records.
3. There are two relations are R and S. Their formats on disk are:

R	(1,R),(2,R),(3,R)	(4,R),(5,R),(6,R)	(7,R),(8,R),(9,R)
S	(11,S),(4,S),(21,S)	(3,S),(31,S),(13,S)	(5,S),(27,S),(23,S)

Explain how a partition hash join would perform a natural join. You should illustrate your explanation using diagrams of the form below for various steps in the processing:

Step N:



The notation (n,X) means the record in file/relation X with value n for the key. Ti is a temporary file/relation that is created during the processing. You will likely have to create more than one temporary file.

Answer:

Design choice: the number of partitions for R and S is 3. In other words, the hash function partitions R and S into 3 buckets, respectively. The hash function used is simple mod.

(1,R), (2,R), (3,R)	(4,R), (5,R), (6,R)	(7,R), (8,R), (9,R)

(11,S), (4,S), (21,S)	(3,S), (31,S), (13,S)	(5,S), (27,S), (23,S)

Step 1: load R into buffer, partition tuples of R using the hash function, and load these partitions into T1 in which each bucket contains tuples with the same join attribute

(1,R), (2,R), (3,R)
(4,R), (5,R), (6,R)
(7,R), (8,R), (9,R)
(2,T1),(5,T1),(8,T1)

T1:

(3,T1),(6,T1),(9,T1)	(1,T1),(4.T1),(7,T1)	(2,T1),(5,T1),(8,T1)

Step 2: load S into buffer. During this step, some blocks in the buffer need to be cleared to make room for tuples in S.

(11,S), (4,S), (21,S)

(3,S), (31,S), (13,S)
(5,S), (27,S), (23,S)

T1:

(3,T1),(6,T1),(9,T1)	(1,T1),(4.T1),(7,T1)	(2,T1),(5,T1),(8,T1)

Step 3: partition tuples of S into T2 in which each bucket contains tuples with the same join attribute

(11,S), (4,S), (21,S)
(3,S), (31,S), (13,S)
(5,S), (27,S), (23,S)
(11,T2), (5,T2), (23,T2)

T1:

(3,T1),(6,T1),(9,T1)	(1,T1),(4.T1),(7,T1)	(2,T1),(5,T1),(8,T1)

T2:

(21,T2), (3,T2), (27,T2)	(4,T2), (31,T2), (13,T2)	(11,T2), (5,T2), (23,T2)

Step 4: clear buffer and load tuples from bucket 0 of T1 and T2 into buffer

(3,T1),(6,T1),(9,T1)
(21,T2), (3,T2), (27,T2)

T1:

(3,T1),(6,T1),(9,T1)	(1,T1),(4.T1),(7,T1)	(2,T1),(5,T1),(8,T1)

T2:

(21,T2), (3,T2), (27,T2)	(4,T2), (31,T2), (13,T2)	(11,T2), (5,T2), (23,T2)

Step 5: search for the tuple with the same value for the join attribute

(3,T1),(6,T1),(9,T1)
(21,T2), (3,T2), (27,T2)
(3,F)

T1:

(3,T1),(6,T1),(9,T1)	(1,T1),(4.T1),(7,T1)	(2,T1),(5,T1),(8,T1)

T2:

(21,T2), (3,T2), (27,T2)	(4,T2), (31,T2), (13,T2)	(11,T2), (5,T2), (23,T2)

F:

Step 6: clear first two blocks in the buffer, load tuples from bucket 1 of T1 and T2 into buffer, and search for the tuple with the same value for the join attribute

(1,T1),(4.T1),(7,T1)
(4,T2), (31,T2), (13,T2)
(3,F), (4,F)

T1:

(3,T1),(6,T1),(9,T1)	(1,T1),(4.T1),(7,T1)	(2,T1),(5,T1),(8,T1)

T2:

(21,T2), (3,T2), (27,T2)	(4,T2), (31,T2), (13,T2)	(11,T2), (5,T2), (23,T2)

F:

Step 7: clear first two blocks in the buffer, load tuples from bucket 2 of T1 and T2 into buffer, and search for the tuple with the same value for the join attribute

(2,T1),(5,T1),(8,T1)
(11,T2), (5,T2), (23,T2)
(3,F), (4,F), (5,F)

T1:

(3,T1),(6,T1),(9,T1)	(1,T1),(4.T1),(7,T1)	(2,T1),(5,T1),(8,T1)

T2:

(21,T2), (3,T2), (27,T2)	(4,T2), (31,T2), (13,T2)	(11,T2), (5,T2), (23,T2)

F:

Step 8: load the resulting relation from join operation into relation F

(2,T1),(5,T1),(8,T1)
(11,T2), (5,T2), (23,T2)
(3,F), (4,F), (5,F)

T1:

(3,T1),(6,T1),(9,T1)	(1,T1),(4.T1),(7,T1)	(2,T1),(5,T1),(8,T1)

T2:

(21,T2), (3,T2), (27,T2)	(4,T2), (31,T2), (13,T2)	(11,T2), (5,T2), (23,T2)

F:

(3,F), (4,F), (5,F)		

Question 12

Question: Give three reasons why a query processing engine might use a sort-merge join instead of a hash join? What are the key differences between sort-merge and hash join?

Answer:

- Relations are already sorted on the join attribute(s)
- A range of values of the join attribute(s) is required after the join operation. For example, sort-merge join is preferred for the following query: *select * from A natural join B where 10 < A.joinAttribute < 20*
- Sorting on the join attribute(s) is required after the join operation. For example, sort-merge join is preferred for the following query: *select * from A natural join B order by [joinAttribute]*

A sort-merge join is performed by sorting the two relations to be joined according to the join attribute(s) (if necessary) and then merging them together. Hash join is performed by hashing two relations into partitions based on their join attribute(s) and joining the partitions from two relations with the same hash value.

References:

1. https://stackoverflow.com/questions/1111707/what-is-the-difference-between-a-hash-join-and-a-merge-join-oracle-rdbms/1114288

Question 13

Question:

Let r and s be relations with no indices, and assume that the relations are not sorted. Assuming infinite memory, what is the lowest-cost way (in terms of I/O operations) to compute $r \bowtie s$? What is the amount of memory required for this algorithm?

Answer:

- Store the entire smaller relation in memory

- Read the larger relation block by block and perform nested loop join with the larger relation as the outer relation

The number of I/O is $(b_r + b_s)$ where b_r is the number of blocks in r and b_s is the number of blocks in s. The required memory is $\min(b_r + b_s) + 2$ pages.

References:

1. https://www.db-book.com/db6/practice-exer-dir/12s.pdf

Question 14

Question:

Rewrite/transform the following query into an equivalent query that would be significantly more efficient.

```
select
        people.playerid, people.nameLast, people.throws,
        batting.teamid, batting.yearid, ab, h, rbi
from
        (people join batting using(playerid))
where teamid='BOS' and yearID='1960';
```

Answer:

```
select
        *
from
        ((select playerid, nameLast, throws from people) as a
    join
    (select playerid, teamid, yearid, ab, h, rbi from batting where teamid='BOS' and yearID='1960') as b
    using(playerid));
```

Question 15

Question: Suppose that a B+-tree index on (dept_name, building) is available on the relation department. (Note: This data comes from the database at https://www.db-book.com/db7/index.html)

What would be the best way to handle the following selection? What strategy/techniques would the query optimizer/processor use?

$$\sigma_{(building < \text{“Watson”}) \wedge (budget < 55000) \wedge (dept\_name = \text{“Music”})}(department)$$

Answer:
The query optimizer would first convert the above query into

σ (dept_name = 'Music') ∧ (building < 'Watson') ∧ (budget < 55000) (department)

and then process it.

First, we can reorder predicates connected by "∧" operator. Thus, dept_name predicate is placed at the beginning, followed by building predicate. Second, reordering predicates takes advantage of the fact that leaf nodes in B+ tree are stored in order. With a B+ tree on (dept_name, building), scanning based on (dept_name = 'Music') ∧ (building < 'Watson') is much faster than the original query.

The query processor will go through the records in B+ tree with dept_name = 'Music' until it finds a record either with different dept_name or with building >= 'Watson'. Then the processor needs to find records with budget < 55000 from records it just went through. With the help of B+ tree, execution would be much faster.

Question 16

Question: Consider the following relational algebra expression

$$\pi_{a,b,c}(R \bowtie S)$$

This is a project on the result of a natural join on R and S. Assume that column *a* comes from R, column *b* comes from S and that *c* is the join column. Also assume that both R and S have many large columns. Write an equivalent query that will be more efficient, and explain why.

Answer:
The equivalent query is

π a,c (R) ⋈ π b,c (S)

The reason why we push the projection down to both relations is that it will eliminate the redundancy and thus reduce the size of relations to be joined. In this way, larger parts of R and S can be fit into memory. The number of times when we have to retrieve records from disk will be reduced, and the query will become more efficient.

Question 17

Question:

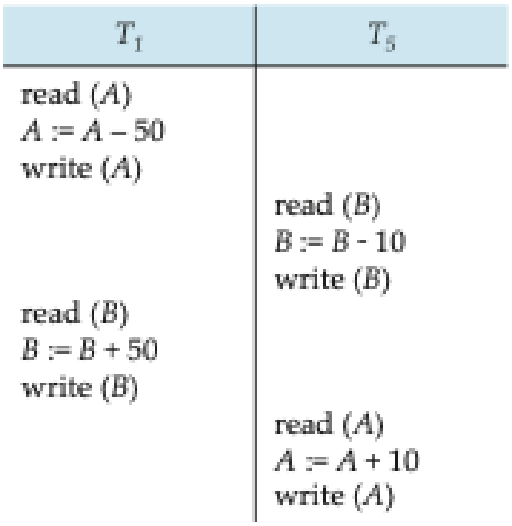
For each of the following isolation levels, give an example of a schedule that respects the specified level of isolation but is not serializable:

a. Read uncommitted

b. Read committed

c. Repeatable read

Use the notation/diagram format that the slides and textbook use for schedules. An example is:



T_1	T_5
read (A)	
$A := A - 50$	
write (A)	
	read (B)
	$B := B - 10$
	write (B)
read (B)	
$B := B + 50$	
write (B)	
	read (A)
	$A := A + 10$
	write (A)

Answer:

Read Uncommitted:

Step	T1	T2
1	read(A)	
2	A = 1	
3	write(A)	
4		read(A)
5		A += 1
6		write(A)
7	read(A)	

In the above schedule, T2 reads the value of A (which is 1) at step 4 before T1 commits its write at step 3. The schedule is not serializable because T1 reads the value of A (which is 2) written by T2 at step 7, thus resulting in a cycle.

Read Committed: Initially, A = 0

Step	T1	T2
1	read(A)	
2		A += 1
3		write(A)
4		commit
5	read(A)	

In the above schedule, T1 reads A = 1 at step 1 and A = 2 at step 2. This is because A is modified by T2 and the modified value (A = 2) is committed. Thus, T1 sees the modified value at step 5. This schedule is not serializable because T1's first read needs to happen before T2, but T1's second read must happen after T2.

Repeatable Read: Initially, A = 0; B = 0

Step	T1	T2
1	read(A)	
2		A += 1
3		write(A)
4		B += 1
5		write(B)
6		commit
7	read(A)	
8	read(B)	

In the above example, T1 reads A = 0 at both step 1 and step 7 even though T2 set A = 2 at step 2. Also, T1 reads B = 1 at step 8 because T2 has modified B to 1 in step 4 and the value is committed. This schedule is not serializable because T1 has to read A = 0, which means T1 must happen before T2. However, T1 also has to read B = 1, which is a value modified by T2. This means T1 has to happen after T2, which causes conflict in a serial schedule.

References:

1. https://www.chegg.com/homework-help/following-isolation-levels-give-example-schedule-respects-sp-chapter-14-problem-20e-solution-9780073523323-exc

Question 18

Question: Explain the difference between a *serial schedule* and a *serializable schedule.*

Answer:
A schedule is a sequence of instructions that specify the chronological order in which instructions of concurrent transactions are executed.

A serial schedule is a schedule in which one transaction must complete before another transaction can start. A serializable schedule is a schedule that is equivalent to a serial schedule.

- A serializable schedule does not have to be serial as long as it can be transformed into an equivalent serial schedule.
- A serial schedule is always a serializable schedule.

Question 19

Question: What are the benefits and disadvantages of strict two phase locking?

Answer:
Benefits:

- Ensures serializability
- Only produces cascadeless schedules (data written by a transaction will not be read until it commits), which makes recovery easy.

Disadvantages:

- Since not every ordinary two phase locking can be converted into using strict two phase locking, concurrency is reduced.

References:

1. https://www.chegg.com/homework-help/benefit-strict-two-phase-locking-provide-disadvantages-resul-chapter-15-problem-20e-solution-9780073523323-exc

Question 20

Question: Outline the no-steal and force buffer management policies.

Answer:
No-steal policy requires that updates made by an uncommitted transaction cannot be flushed to disk and overwrite committed data on disk.

Force policy requires that all updates made by a transaction must be flushed to disk before the transaction commits.