Midterm Review

Question1

Imagine you are given the task of creating a database for a hospital. Among other things, you need to keep track of information about patients, doctors, and the relationships between them. If you decide to use an XML database, describe two different ways that you could capture the relationships between patients and doctors in the database

Answer

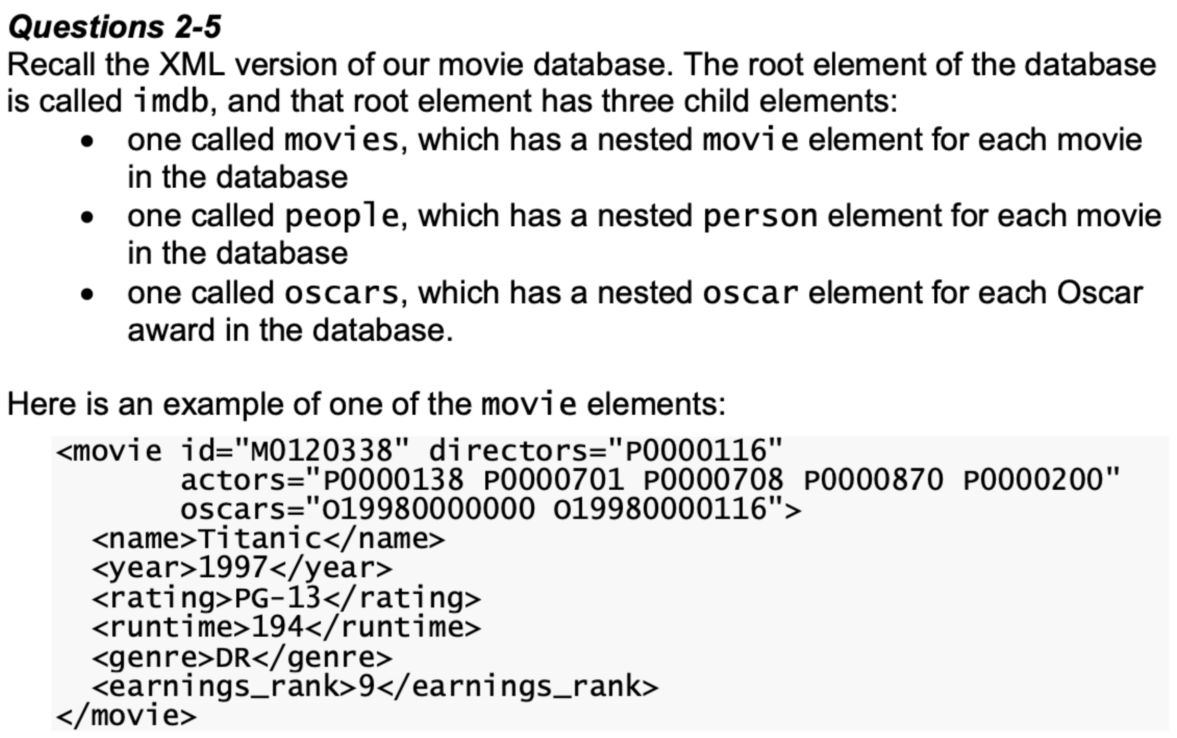
We can either choose to use IDREFS and have attributes act as the relation or make a child element nested in the parent. Thus we have 4 answers

- Patients has a child element called doctor for each doctor that has operated on them
- Patients has idrefs to the `doctor_id` (say) in the doctor table

OR

- doctor has a child element of patients
- doctor has an idref to patients

Question2



Questions 2-5
Recall the XML version of our movie database. The root element of the database is called `imdb`, and that root element has three child elements:

- one called `movies`, which has a nested `movie` element for each movie in the database
- one called `people`, which has a nested `person` element for each movie in the database
- one called `oscars`, which has a nested `oscar` element for each Oscar award in the database.

Here is an example of one of the `movie` elements:

```
<movie id="M0120338" directors="P0000116"
       actors="P0000138 P0000701 P0000708 P0000870 P0000200"
       oscars="O19980000000 O19980000116">
  <name>Titanic</name>
  <year>1997</year>
  <rating>PG-13</rating>
  <runtime>194</runtime>
  <genre>DR</genre>
  <earnings_rank>9</earnings_rank>
</movie>
```

The actors attribute above is an example of what type of attribute? How is this type of attribute similar to a foreign key? How is it different?

Answer 2

- Actors is a IDREFS attribute used to capture relations with another document or database in XML. It's similar to foreign keys in linking the Movies to the

Peoples who acted in them however foreign keys are unique for each element, IDREFS are not

Question 3

Write an XPath expression that obtains the names of all movies with an R rating from the 1990s. (There are many possible answers here. See if you can come up with at least two!)

Answer 3

- `//movie[rating="R" and year>=1990 and year < 1999]/name`
- `//movie[rating="R" and contains(year, "199")]/name`
- `//movie/name[../rating="R" and ../year>=1990 and ../year <=1999]`

Question 4

Write a FLWOR expression that solves the same problem as Question 3. Order the results alphabetically by name.

Answer4

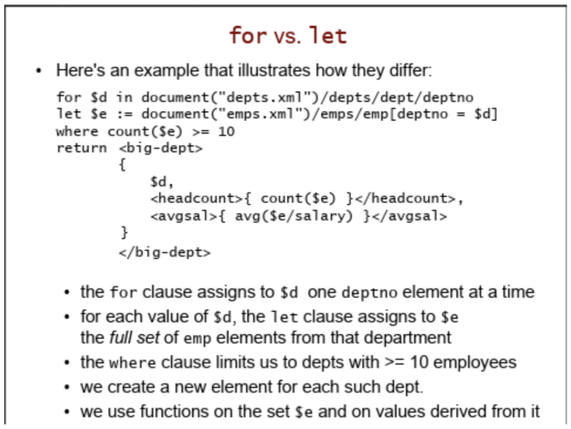
```
for $m in //movie[rating="R" and year>=1990 and year<=1999]
order by $m/name
return $m/name

for $m in //movie
where $m[rating="R" and year>=1990 and year<=1999]
order by $m/name
return $m/name
```

Question 5

Write a FLWOR expression that finds the average runtime of all movies with an R rating from the 1990s. Your query should return a single element of type average

Answer 5

The way I think about it is that a let clause is sort of like a nested for loop but it returns the full set instead of one variable at a time, it allows you to do things like use aggregates. See images from the coursepack below.



for vs. let

- Here's an example that illustrates how they differ:

```
for $d in document("depts.xml")/depts/dept/deptno
let $e := document("emps.xml")/emps/emp[deptno = $d]
where count($e) >= 10
return <big-dept>
        {
            $d,
            <headcount>{ count($e) }</headcount>,
            <avgsal>{ avg($e/salary) }</avgsal>
        }
        </big-dept>
```

- the `for` clause assigns to `$d` one `deptno` element at a time
- for each value of `$d`, the `let` clause assigns to `$e` the *full set* of `emp` elements from that department
- the `where` clause limits us to depts with >= 10 employees
- we create a new element for each such dept.
- we use functions on the set `$e` and on values derived from it

I think this may be helpful to other students, so I'll go ahead and make this public!

```
let $e := //movies/movie[rating="R" and contains(year, "19
9")]/runtime
return <avg_runtime>avg($e)</avg_runtime>
```

Question 6-9

Questions 6-9

Here is an example of one of the `person` elements from our XML movie database:

```
<person id="P0000243" directed="M2671706"
    actedIn="M0097441 M0107818 M0139654 M2671706"
    oscars="O20020000243 O19900000243">
  <name>Denzel Washington</name>
  <dob>1954-12-28</dob>
  <pob>Mount Vernon, New York, USA</pob>
</person>
```

6. Recall that a person element only includes the `directed` attribute if that person is a director – i.e., if they have directed one or more of the movies in the database. Write a query that returns the `name` and `pob` elements of all directors who were born in New York state (i.e., whose `pob` value ends with "New York, USA".

7. Write a query that produces, for every movie directed by a person born in New York state, a pair of elements for the name of that movie and the name of the director. Since both of the relevant elements are called `name`, you should reformat the results so that the element for the movie's name is called `movie` and the element for the director's name is called `director`.

8. Now write a query that produces, for each director born in New York state, a new element of type `ny_director` that each have the following nested child elements: the existing `name` element of the director, one called `birthplace` for their place of birth, and one or more elements of type `directed`, each of which has as its value the name of one of the movies that the person directed.

9. Finally, revise your query for Problem 8 so that you only include a director in the results if they have directed more than one of the movies in the database.

Answer 6-9

6)

```
for $p in //person[@directed]
where contains(pob, "New York, USA")
return ($p/name, $p/pob, " ")

for $p in //person[@directed and contains(pob, "New York, US
A")]
return ($p/name, $p/pob, " ")
```

7)

```
for $p in //person[@directed],
        $m in //movies
where contains(pob, "New York, USA") and contains($p/@directe
d, $m/@id)
return <movie>{string ($m/name)}</movie>, <director>{string
($p/name)}</director>
```

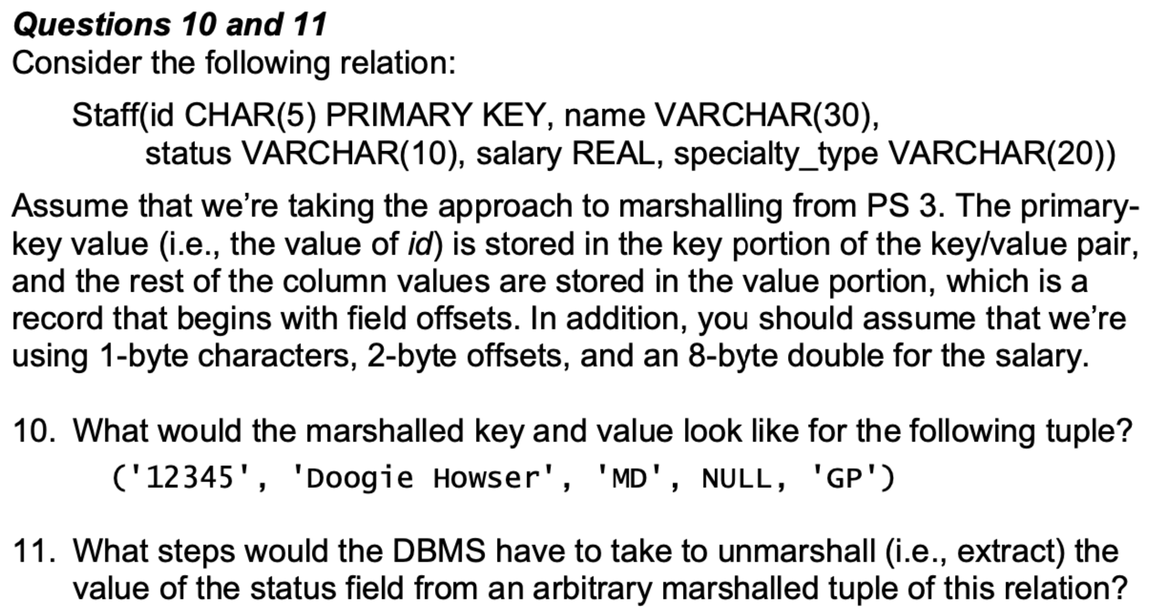
8)

```
for $p in //person[@directed]
return
<ny_director>{
    $p/name
    <birthplace>string($p/pob)</birthplace>
    for $m in //movie
    where contains($p/@directed, $m/@id)
    return <directed> {string($m/name)}</directed>
}
</ny_director>
```

9)

```
for $p in //person[@directed]
let $m:= //movie[contains($p/@directed, $m/@id)]
where coutn($m) > 1
return
<ny_director>{
    $p/name
    <birthplace>string($p/pob)</birthplace>
    for $m in //movie
    where contains($p/@directed, $m/@id)
    return <directed> {string($m/name)}</directed>
}
</ny_director>
```

Questions 10-11



Questions 10 and 11
Consider the following relation:

Staff(id CHAR(5) PRIMARY KEY, name VARCHAR(30),
status VARCHAR(10), salary REAL, specialty_type VARCHAR(20))

Assume that we're taking the approach to marshalling from PS 3. The primary-key value (i.e., the value of *id*) is stored in the key portion of the key/value pair, and the rest of the column values are stored in the value portion, which is a record that begins with field offsets. In addition, you should assume that we're using 1-byte characters, 2-byte offsets, and an 8-byte double for the salary.

10. What would the marshalled key and value look like for the following tuple?
`('12345', 'Doogie Howser', 'MD', NULL, 'GP')`

11. What steps would the DBMS have to take to unmarshall (i.e., extract) the value of the status field from an arbitrary marshalled tuple of this relation?

Answer 10-11

10)

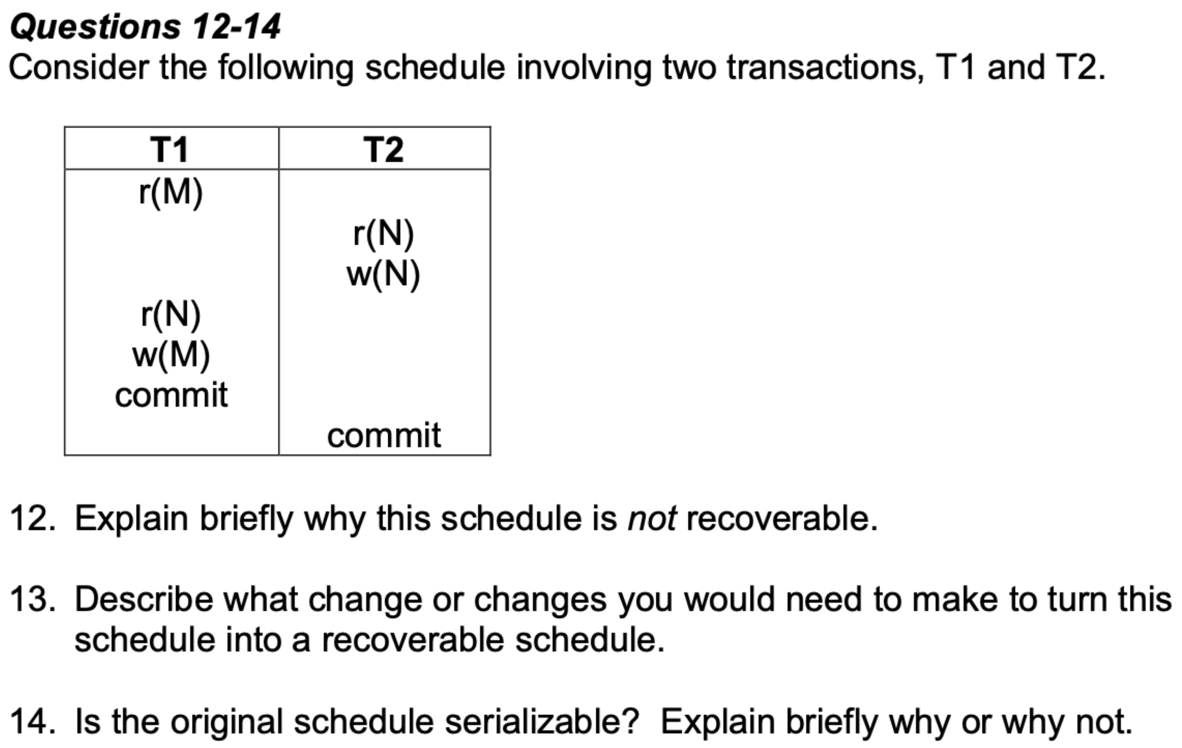
```
mashalled values
(5 + 1)*2 = 12 byte offset

|-2|12|25| -1|27|29|Doogie Howser|MD|GP
```

11)

- To unmarshall, first we'll find the column of status
- We need to first get to status which is the 4th offset (2∗4)since its the second column
- Read 2 bytes at the offset to get to offset value of 25
- Since status is not primary column, and its archer, we loop until no non-null values are left or non -2, and then read the length of the fields value which is 27-25 = 2

- Read 2 bytes from offset S to get the value of the filed

Answer 12-14



Questions 12-14
Consider the following schedule involving two transactions, T1 and T2.

T1	T2
r(M)	
	r(N)
	w(N)
r(N)	
w(M)	
commit	
	commit

12. Explain briefly why this schedule is *not* recoverable.

13. Describe what change or changes you would need to make to turn this schedule into a recoverable schedule.

14. Is the original schedule serializable? Explain briefly why or why not.

Answer 12-14

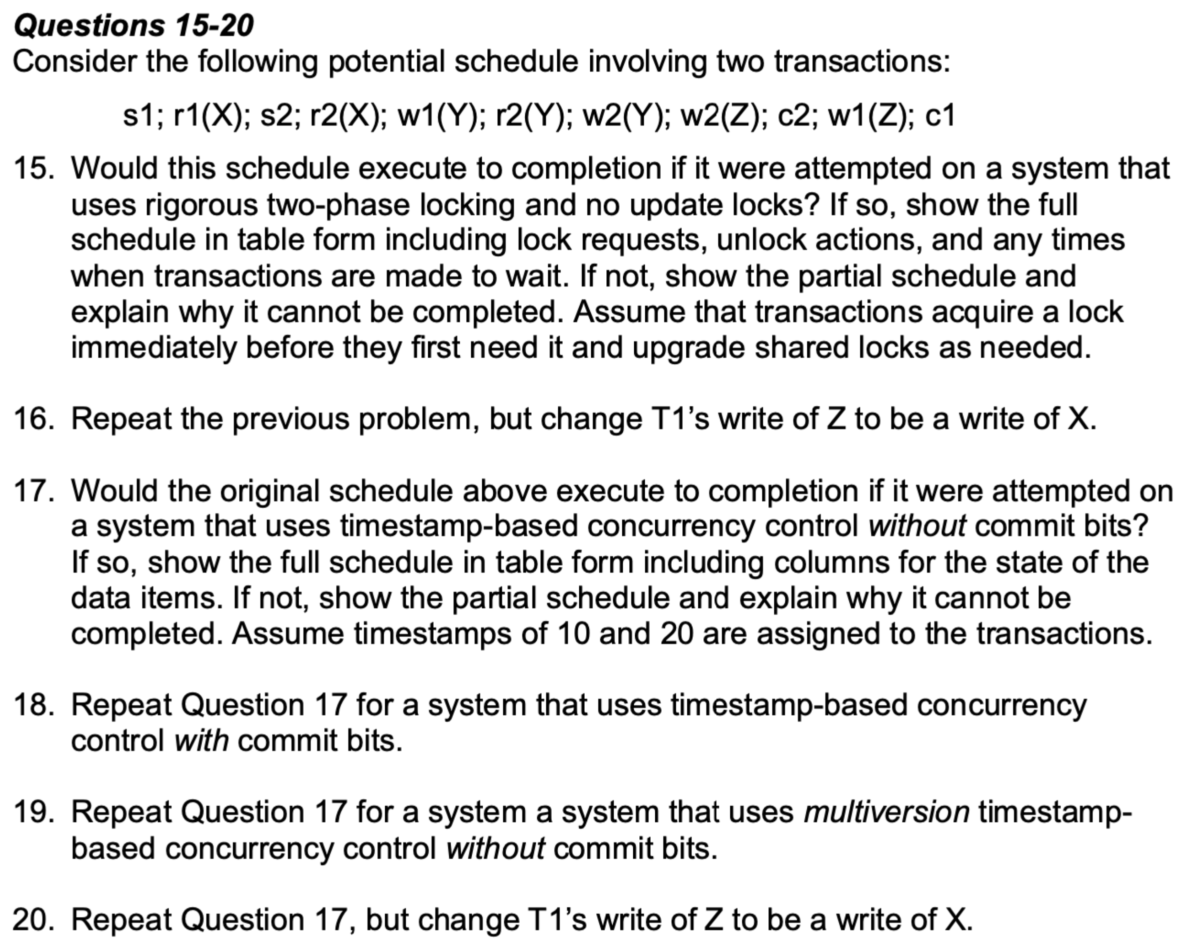
12)

- we have a dirty read which is why the schedule is not recoverable, recoverability ensures that if T2 crashes the system does not go into an inconsistent state however here since T1 commits before T1 we dont know if after restarting T2, T1 would behave the same way or not.

13) To become consistent, either T2 can commit before T1 so we can move T1's commit after T2's or we can commit T2 right after the `w(N)` and before T1's `r(N)`

14)The original schedule is serializable since its conflict serializable.

Question 15-19



Questions 15-20
Consider the following potential schedule involving two transactions:

s1; r1(X); s2; r2(X); w1(Y); r2(Y); w2(Y); w2(Z); c2; w1(Z); c1

15. Would this schedule execute to completion if it were attempted on a system that uses rigorous two-phase locking and no update locks? If so, show the full schedule in table form including lock requests, unlock actions, and any times when transactions are made to wait. If not, show the partial schedule and explain why it cannot be completed. Assume that transactions acquire a lock immediately before they first need it and upgrade shared locks as needed.

16. Repeat the previous problem, but change T1's write of Z to be a write of X.

17. Would the original schedule above execute to completion if it were attempted on a system that uses timestamp-based concurrency control *without* commit bits? If so, show the full schedule in table form including columns for the state of the data items. If not, show the partial schedule and explain why it cannot be completed. Assume timestamps of 10 and 20 are assigned to the transactions.

18. Repeat Question 17 for a system that uses timestamp-based concurrency control *with* commit bits.

19. Repeat Question 17 for a system a system that uses *multiversion* timestamp-based concurrency control *without* commit bits.

20. Repeat Question 17, but change T1's write of Z to be a write of X.

15. It would complete, but with some actions in a different order, as seen below:

T1	**T2**
sl(X); r(X)	
	sl(X); r(X)
xl(Y); w(Y)	
	sl(Y) denied; wait for T1

xl(Z); w(Z)
commit
u(X); u(Y); u(Z)	

	sl(Y); r(Y)
	xl(Y); w(Y)
	xl(Z); w(Z)
	commit
	u(X); u(Y); u(Z)

T1	T2
sl(X); r(X)	sl(X) r(X)
xl(Y); w(Y)	
	s(Y) (denied wait for T1)
xl(Z); w(Z)	
c1 u(X); u(Y); u(Z)	

	sl(Y); r(Y)
	xl(Y); w(Y)
	(upgraded
	xl(Z); w(Z)
	c2
	u(X); u(Y); u(Z)

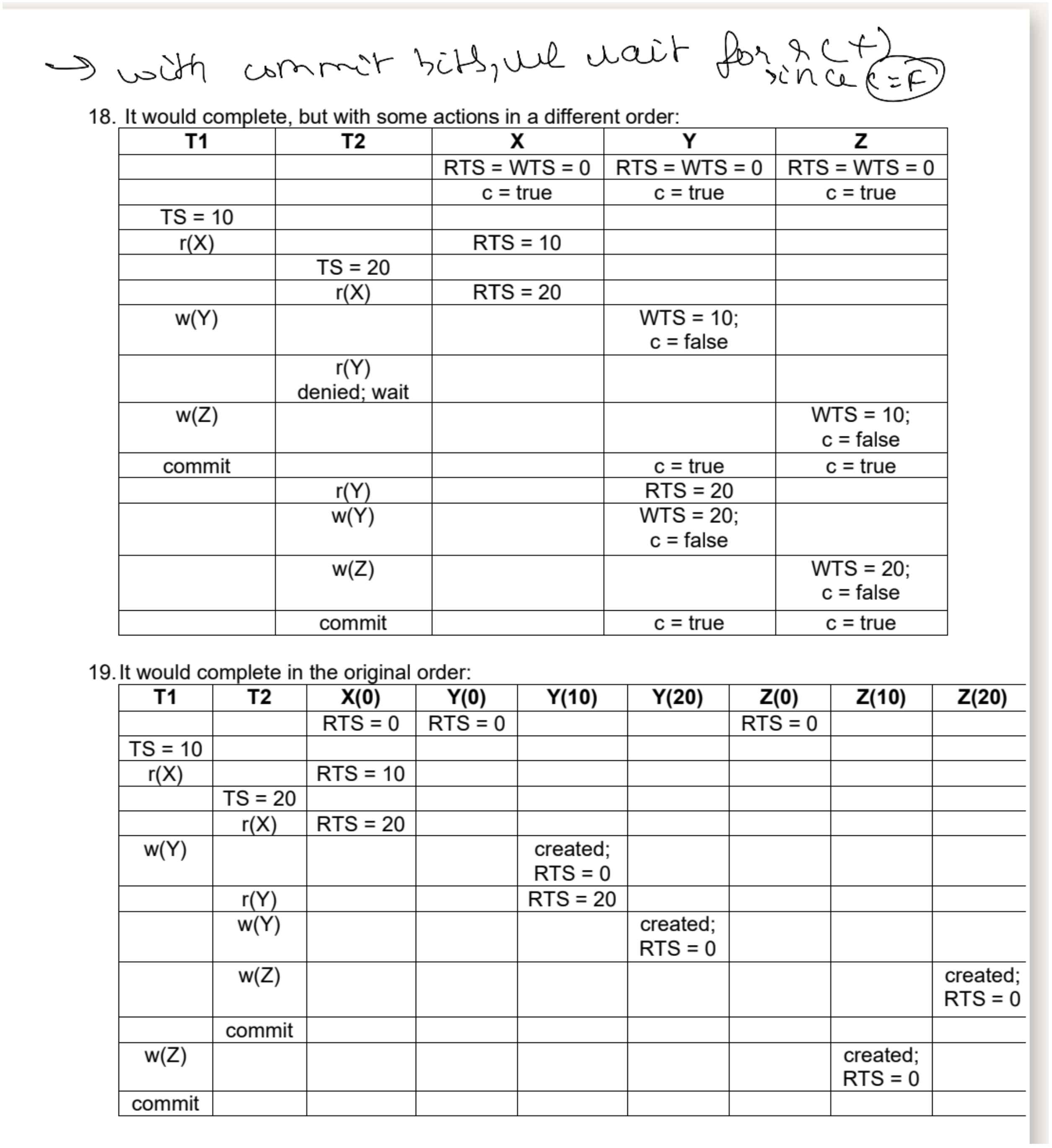
16. The revised schedule would produce deadlock:

T1	**T2**
sl(X); r(X)	
	sl(X); r(X)
xl(Y); w(Y)	
	sl(Y) denied; wait for T1
xl(X) denied; wait for T2	

☆ If T1's write of Z changes to X, we would try for an xl of X but get a deadlock

17. It would complete in the original order, as seen below:

T1	**T2**	**X**	**Y**	**Z**
		RTS = WTS = 0	RTS = WTS = 0	RTS = WTS = 0
TS = 10				
r(X)		RTS = 10		
	TS = 20			
	r(X)	RTS = 20		
w(Y)			WTS = 10	
	r(Y)		RTS = 20	
	w(Y)		WTS = 20	
	w(Z)			WTS = 20
	commit			
w(Z); ignore				
commit				



→ with commit bits, we wait for r(Y) since $\boxed{c = F}$

18. It would complete, but with some actions in a different order:

T1	T2	X	Y	Z
		RTS = WTS = 0	RTS = WTS = 0	RTS = WTS = 0
		c = true	c = true	c = true
TS = 10				
r(X)		RTS = 10		
	TS = 20			
	r(X)	RTS = 20		
w(Y)			WTS = 10; c = false	
	r(Y) denied; wait			
w(Z)				WTS = 10; c = false
commit			c = true	c = true
	r(Y)		RTS = 20	
	w(Y)		WTS = 20; c = false	
	w(Z)			WTS = 20; c = false
	commit		c = true	c = true

19. It would complete in the original order:

T1	T2	X(0)	Y(0)	Y(10)	Y(20)	Z(0)	Z(10)	Z(20)
		RTS = 0	RTS = 0			RTS = 0		
TS = 10								
r(X)		RTS = 10						
	TS = 20							
	r(X)	RTS = 20						
w(Y)				created; RTS = 0				
	r(Y)			RTS = 20				
	w(Y)				created; RTS = 0			
	w(Z)							created; RTS = 0
	commit							
w(Z)							created; RTS = 0	
commit								

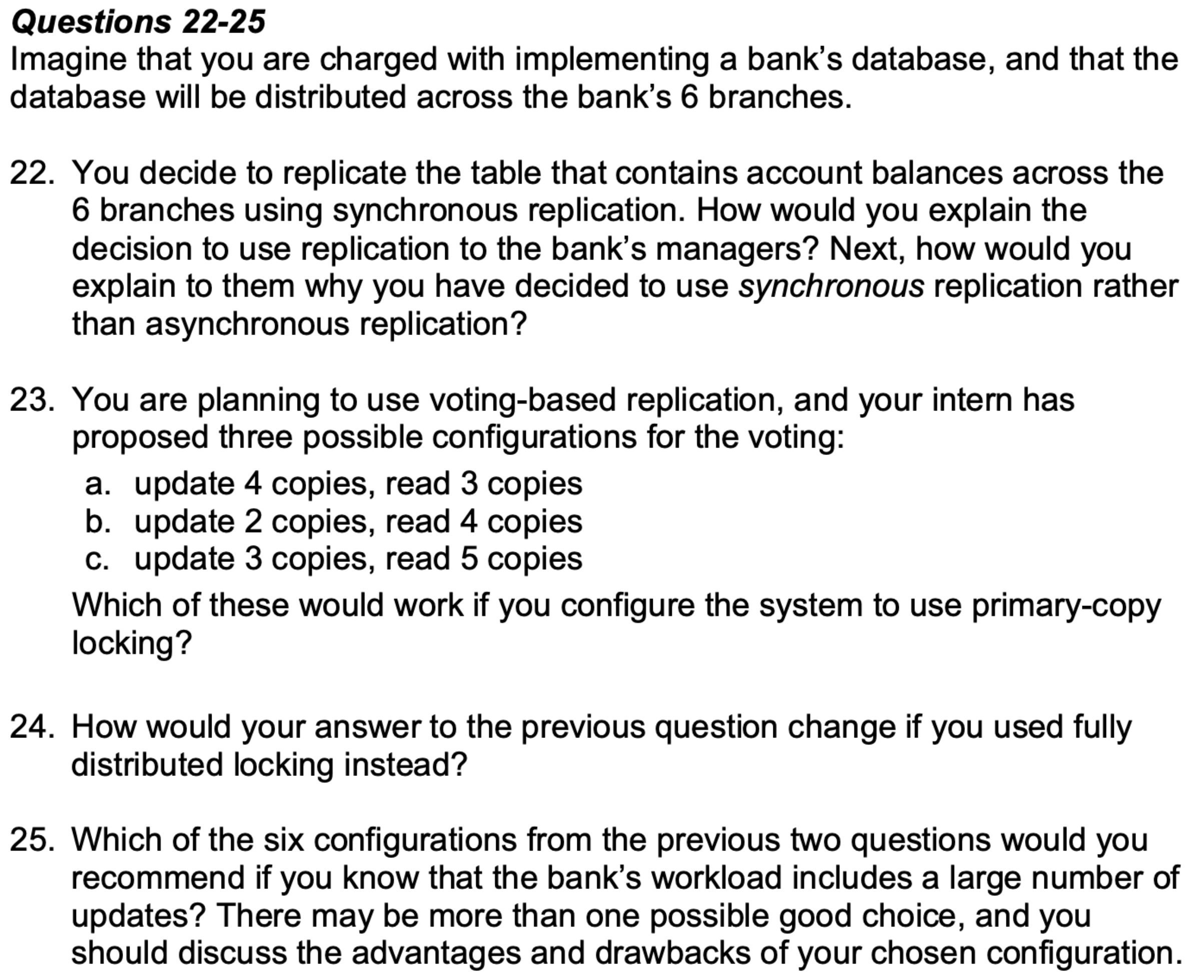
Question 21

What does it mean for a schedule to be cascadeless? How can a DBMS guarantee this property if it uses locks for concurrency control? How can a DBMS guarantee this property if it uses a timestamp-based approach?

Answer 21

Cascadeless means that rolling back one transaction does not lead to a cascading rollback of another one. For a DBMS, using strict or rigorous 2PL ensures cacadelessness, for multi-version timestamps we need to use commit bits.

Question 22-25



Questions 22-25
Imagine that you are charged with implementing a bank's database, and that the database will be distributed across the bank's 6 branches.

22. You decide to replicate the table that contains account balances across the 6 branches using synchronous replication. How would you explain the decision to use replication to the bank's managers? Next, how would you explain to them why you have decided to use *synchronous* replication rather than asynchronous replication?

23. You are planning to use voting-based replication, and your intern has proposed three possible configurations for the voting:
    a. update 4 copies, read 3 copies
    b. update 2 copies, read 4 copies
    c. update 3 copies, read 5 copies

    Which of these would work if you configure the system to use primary-copy locking?

24. How would your answer to the previous question change if you used fully distributed locking instead?

25. Which of the six configurations from the previous two questions would you recommend if you know that the bank's workload includes a large number of updates? There may be more than one possible good choice, and you should discuss the advantages and drawbacks of your chosen configuration.

Answer 22-25

22)

- Synchronous replication because user's need to be updated on money matters, everyone should read the latest updates and all branches should be updated in real time.

- replication allows for
  - fault tolerance
  - load balancing
  - efficient computation instead of a centralized source.

23)

`a and c` would be used, b would not be because we could update 2 copies and read from the remaining 4

24) Since we are not using primary locking, a single machine does not acquire exclusive or shared locks we need a majority of locks. For voting, we can clearly see that only `a` works because we have more than 3 updates `w>n/2` and proper number of read `r>n-w`

25) We should use 23 c in case of primary copy locking since it only required one copy to be locked and useful for expensive workload updates. We have reduced availability because if the primary site is unavailable we cannot do anything. however its less expensive

using 24 a, we need to update an extra copy however we never have to worry about 1 failure making reads and writes impossible.