1. feladat: Legális-e a 2. táblázat által mutatott ütemezés? Ha nem, mit kellene javítani rajta, hogy azzá váljon?

2. feladat: Ellenőrizd, hogy az 1. táblázaton látható ütemezés legális-e! Rajzold meg a sorosíthatósági gráfot, döntsd el, hogy sorosítható-e az ütemezés! Ha igen, adj egy soros ekvivalenst, ha nem, mutasd meg, miért nem! Hogyan nézne ki a gráf, ha egyszerű zármodellt használnánk?

3. feladat: Legális-e a 3. táblázat szerinti ütemezés? A tranzakciók követik-e a 2PL-t? Hol van az alábbi tranzakciók zárpontja? Mi egy soros ekvivalens ütemezés?

4. feladat: Időbélyeges tranzakciókezelést használunk R/W modellben. Jegyezd fel az alábbi sorozat minden művelete után az R(A), R(B), W(A), W(B) értékeit, ha kezdetben mindegyik 0. Mely tranzakciók abortálnak? r_i és w_i a T_i tranzakció olvasás (r) és írás műveleteit (w) jelöli, és t(T_i) = i.

$$r_1(A), r_2(B), r_1(B), w_3(B), r_2(B), w_4(A), r_4(B), w_1(A), w_3(B)$$

5. feladat: Oldd meg a 4. feladatot verziókezeléssel kiegészítve! Most mi történik?

6. feladat: Egy rendszerleállás után a napló vége az 5. táblázat szerinit bejegyzéseket tartalmazza. Melyek a redo helyreállítás lépései? Mi lesz a helyreállítás után A, B és C értéke?

Gondolkodtató kérdések

1. feladat: A naplózás tárhely igényét szeretnénk optimalizálni. Helyes-e a következő érvelés? Mivel egy tranzakciónak csak a COMMIT pontjáig van szüksége a naplóra – hiszen a COMMIT utáni műveletek biztosan lefutnak –, ezért szigorú 2PL alkalmazásával megelőzzük a lavinahatást, és a COMMIT naplózása helyett így a naplóból már törölhetjük az adott tranzakcióhoz tartozó bejegyzéseket (ha garantáljuk ezen törlés atomicitását).
2. feladat: Hogyan biztosítja a holtpontmentességet a 2PL?
3. feladat: Igaz-e, hogy egy kétfázisú protokoll estén a tranzakciók mindig helyesen futnak le? (Mit jelenthet az, hogy „helyesen”?)
4. feladat: Lehet-e konkurensen módosítani egy állományt, amire B* fa épül? Mikor lehet felszabadítani a gyökér elemet fogó zárat?
5. feladat: Miért fontos a sorosíthatóság?
6. feladat: Egy ütemezés nem sorosítható, ennek ellenére érvényes lehet-e az izolációs elv?
7. feladat: Ha a naplófájl tartalmaz minden információt a változásokról, akkor miért kell az adatbázis?
8. feladat: Mondj példát kézenfekvő soros ekvivalensre 2PL és időbélyeges tranzakciókezelés esetén!
9. feladat: Mikor érdemes 2PL-t és mikor időbélyeges tranzakciókezelést alkalmazni?
10. feladat: Mi történik a sorosíthatósági gráffal, ha egy tranzakció abortál?
11. feladat: Az időbélyeges tranzakciókezelés miként véd a holtpont ellen?
12. feladat: Mondj példát vagy ellenpéldát a következő esetekre!
    (a) Időbélyeges tranzakciókezelés esetén egy tranzakció READ esetén abortál.
    (b) Időbélyeges tranzakciókezelés esetén egy tranzakció WRITE esetén abortál.
    (c) Időbélyeges tranzakciókezelést verziókezeléssel együtt alkalmazunk. Egy tranzakció READ esetén abortál.
    (d) Időbélyeges tranzakciókezelést verziókezeléssel együtt alkalmazunk. Egy tranzakció WRITE esetén abortál.
13. feladat: Hogyan tároljuk az időbélyegeket? Mit tudunk mondani a számukra vonatkozóan? Meddig kell fenntartani?

Kiegészítő feladatok

1. feladat: Ellenőrizd, hogy a 4. táblázaton látható ütemezés legális-e! Rajzold meg a sorosíthatósági gráfot, döntsd el, hogy sorosítható-e az ütemezés! Ha igen, adj egy soros ekvivalenst, ha nem, mutasd meg, miért nem! Hogy nézne ki a gráf, ha RLOCK-WLOCK modellt használnánk (ahol csak olvasunk, ott RLOCK-al, ahol írunk (is) ott WLOCK-al helyettesítjük értelemszerűen a LOCK-ot)?

2. feladat: Egy tranzakció az 1. ábra által mutatott hierarchiában a D, I és J adategységekhez szeretne hozzáférni, majd befejezné működését. Milyen zárakat kell és hova elhelyezni-e, illetve milyen sorrendben kell felszabadítania a zármenedzsernek, ha mindenkor a lehető legkevesebb zárat szeretné fenntartani?

	T_1	T_2	T_3	T_4
(1)		**RLOCK F**		
(2)	**RLOCK A**			
(3)			**RLOCK D**	
(4)			**UNLOCK D**	
(5)	**UNLOCK A**			
(6)			**WLOCK B**	
(7)	**RLOCK D**			
(8)				**RLOCK A**
(9)	**WLOCK E**			
(10)				**UNLOCK A**
(11)			**UNLOCK B**	
(12)				**WLOCK A**
(13)	**UNLOCK D**			
(14)	**RLOCK C**			
(15)				**UNLOCK A**
(16)		**UNLOCK F**		
(17)			**WLOCK A**	
(18)		**WLOCK B**		
(19)	**UNLOCK E**			
(20)	**UNLOCK C**			
(21)				**RLOCK D**
(22)				**UNLOCK D**
(23)			**WLOCK C**	
(24)			**UNLOCK C**	
(25)		**WLOCK D**		
(26)			**UNLOCK A**	
(27)		**RLOCK E**		
(28)		**UNLOCK E**		
(29)				**RLOCK E**
(30)		**UNLOCK B**		
(31)	**RLOCK F**			
(32)		**UNLOCK D**		
(33)	**UNLOCK F**			
(34)				**UNLOCK E**

1. táblázat

	T_1	T_2	T_3
(1)			RLOCK B
(2)			READ B
(3)		WLOCK B	
(4)			RLOCK A
(5)	RLOCK A		
(6)		WRITE B	
(7)			READ A
(8)	READ A		
(9)	UNLOCK A		
(10)			UNLOCK B
(11)			UNLOCK A

2. táblázat

	T_1	T_2	T_3
(1)	LOCK A		
(2)		LOCK B	
(3)			LOCK C
(4)			LOCK D
(5)	LOCK E		
(6)	UNLOCK A		
(7)			UNLOCK D
(8)		LOCK A	
(9)		LOCK D	
(10)	UNLOCK E		
(11)		UNLOCK B	
(12)			UNLOCK C
(13)		UNLOCK A	
(14)		UNLOCK D	

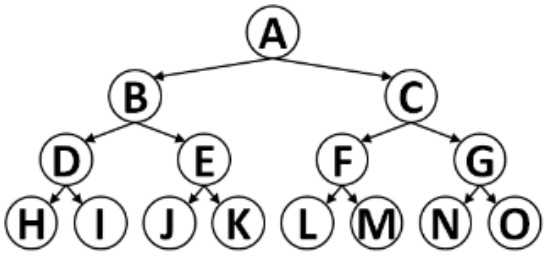
3. táblázat

	T_1	T_2	T_3	T_4
(1)	**LOCK A**			
(2)		**LOCK B**		
(3)	READ A			
(4)	**UNLOCK A**			
(5)		WRITE B		
(6)		**LOCK C**		
(7)			**LOCK A**	
(8)		READ C		
(9)			READ A	
(10)		**UNLOCK C**		
(11)			**UNLOCK A**	
(12)			**LOCK C**	
(13)		**UNLOCK B**		
(14)			READ C	
(15)	**LOCK A**			
(16)				**LOCK B**
(17)	READ A			
(18)				READ B
(19)			**UNLOCK C**	
(20)				**UNLOCK B**
(21)			**LOCK B**	
(22)	WRITE A			
(23)	**UNLOCK A**			
(24)			READ B	
(25)			**UNLOCK B**	
(26)		**LOCK A**		
(27)				**LOCK B**
(28)		WRITE A		
(29)				WRITE B
(30)		**UNLOCK A**		
(31)				**UNLOCK B**

4. táblázat

chekpoint
(T_1, begin)
(T_2, begin)
(T_2, A, 20)
(T_2, B, 10)
(T_1, A, 2)
(T_3, begin)
(T_1, C, 5)
(T_1, commit)
(T_3, C, 6)
(T_3, commit)

5. táblázat



1. ábra. Zárolás hierarchikus adategységeken