Ellenőrző kérdések

5. Kis dolgozat kérdései

(9-10. előadás)

164. Adjunk meg a működés közbeni ellenőrzőpont képzésének lépéseit Undo naplózás esetén! (6 pont)

1. <START CKPT(T1,…,Tk)> naplóbejegyzés készítése, majd lemezre írása (FLUSH LOG), ahol T1,…,Tk az éppen aktív tranzakciók nevei.
2. Meg kell várni a T1,…,Tk tranzakciók mindegyikének normális vagy abnormális befejeződését, nem tiltva közben újabb tranzakciók indítását.
3. Ha a T1,…,Tk tranzakciók mindegyike befejeződött, akkor <END CKPT> naplóbejegyzés elkészítése, majd lemezre írása (FLUSH LOG).

165. Ha UNDO naplózás utáni helyreállításkor előbb <END CKPT> naplóbejegyzéssel találunk, akkor meddig kell visszamenni a napló olvasásában? (2 pont)

Ha előbb az <END CKPT> naplóbejegyzéssel találkozunk, akkor tudjuk, hogy az összes még be nem fejezett tranzakcióra vonatkozó naplóbejegyzést a legközelebbi korábbi <START CKPT(T1,…,Tk)> naplóbejegyzésig megtaláljuk. Ott viszont megállhatunk, az annál korábbiakat akár el is dobhatjuk.

166. Ha UNDO naplózás utáni helyreállításkor előbb <START CKPT(T1,…,Tk)> naplóbejegyzéssel találunk, akkor meddig kell visszamenni a napló olvasásában? (2 pont)

Ha a <START CKPT(T1,…,Tk)> naplóbejegyzéssel találkozunk előbb, az azt jelenti, hogy a katasztrófa az ellenőrzőpont-képzés közben fordult elő, ezért a T1,…,Tk tranzakciók nem fejeződtek be a hiba fellépéséig.
Ekkor a be nem fejezett tranzakciók közül a legkorábban (t) kezdődött tranzakció indulásáig kell a naplóban visszakeresnünk, annál korábbra nem.

167. Adjuk meg a REDO naplózás esetén a lemezre írás sorrendjét 5 lépésben! (5 pont)

(1) Ha egy T tranzakció v-re módosítja egy X adatbáziselem értékét, akkor egy <T,X,v> bejegyzést kell a naplóba írni.
(2) Az adatbáziselemek módosítását leíró naplóbejegyzések lemezre írása.
(3) A COMMIT naplóbejegyzés lemezre írása. (2. és 3. egy lépésben történik.)
(4) Az adatbáziselemek értékének cseréje a lemezen.
(5) A <T,end>-t bejegyezzük a naplóba, majd kiírjuk lemezre a naplót.

168. Adjuk meg a REDO naplózás esetén az R1 szabályt! (2 pont)

R1. Mielőtt az adatbázis bármely X elemét a lemezen módosítanánk, az X módosítására vonatkozó összes naplóbejegyzésnek, azaz <T,X,v>-nek és <T, COMMIT>-nak a lemezre kell kerülnie.

169. Adjunk meg egy példát REDO naplózás esetén a lemezre írás sorrendjére! (6 pont)

Lépés	Tevékenység	t	M-A	M-B	D-A	D-B	Napló
1)							<T,START>
2)	READ(A,t)	8	8		8	8	
3)	t := t*2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<T,A,16>
5)	READ(B,t)	8	16	8	8	8	
6)	t := t*2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<T,B,16>
8)							<T,COMMIT>
9)	FLUSH LOG						
10)	OUTPUT(A)	16	16	16	16	8	
11)	OUTPUT(B)	16	16	16	16	16	
12)							<T,END>
13)	FLUSH LOG						

170. Adjunk meg REDO naplózás esetén a helyreállítás algoritmusát! (8 pont)

For every Ti with <Ti, commit> in log:
- For all <Ti, X, v> in log:
  - Write(X, v)
  - Output(X)

171. Mi jellemző a módosított REDO naplózásra? (8 pont)

Nem használunk $<T_i,end>$ bejegyzést a befejezett tranzakciókra, helyette a be nem fejezetteket jelöljük meg $<T_i,abort>$-tal. (Módosított REDO napló)

172. Fogalmazzunk meg 3 különbséget az UNDO és REDO naplózás esetén! (3 pont)

- Az adat változás utáni értékét jegyezzük fel a naplóba
- Máshová rakjuk a COMMIT-ot, a kiírás elé => megtelhet a puffer
- Az UNDO protokoll esetleg túl gyakran akar írni => itt el lehet halasztani az írást

173. Mit hívunk piszkos puffereknek? (1 pont)

Már végrehajtott, de lemezre még ki nem írt módosításokat tárol.

174. Adjuk meg a működés közbeni ellenőrzőpont képzésének lépéseit REDO naplózás esetén! (6 pont)

1. $<START\ CKPT(T_1,\ldots,T_k)>$ naplóbejegyzés elkészítése és lemezre írása, ahol $T_1,\ldots,T_k$ az összes éppen aktív tranzakció.
2. Az összes olyan adatbáziselem kiírása lemezre, melyeket olyan tranzakciók írtak pufferekbe, melyek a START CKPT naplóba írásakor már befejeződtek, de puffereik lemezre még nem kerültek.
3. <END CKPT> bejegyzés naplóba írása, és a napló lemezre írása.

175. Adjuk meg az UNDO/REDO naplózás esetén az UR1 szabályt! (2 pont)

Mielőtt az adatbázis bármely X elemének értékét – valamely T tranzakció által végzett módosítás miatt – a lemezen módosítanánk, ezt megelőzően a <T,X,v,w> naplóbejegyzésnek lemezre kell kerülnie.

176. Adjuk meg az UNDO/REDO naplózás esetén a WAL elvet! (2 pont)

Write After Log elv: előbb naplózunk, utána módosítunk.

177. Hová kerülhet a COMMIT az UNDO/REDO naplózás esetén? (2 pont)

A <T, COMMIT> bejegyzés megelőzheti, de követheti is az adatbáziselemek lemezen történő bármilyen megváltoztatását.

178. Adjunk meg egy példát UNDO/REDO naplózás esetén a lemezre írás sorrendjére! (6 pont)

Lépés	Tevékenység	t	M-A	M-B	D-A	D-B	Napló
1)							<T,START>
2)	READ(A,t)	8	8		8	8	
3)	t := t*2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<T,A,8,16>
5)	READ(B,t)	8	16	8	8	8	
6)	t := t*2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<T,B,8,16>
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)							<T,COMMIT>
11)	OUTPUT(B)	16	16	16	16	16	

Megjegyzés: A <T,COMMIT> naplóbejegyzés kiírása kerülhetett volna a 9) lépés elé vagy a 11) lépés mögé is.

179. Mi az UNDO/REDO naplózás esetén a helyreállítás 2 alapelve? (4 pont)

(REDO): A legkorábbitól kezdve állítsuk helyre minden befejezett tranzakció hatását.

(UNDO): A legutolsótól kezdve tegyük semmissé minden be nem fejezett tranzakció tevékenységeit.

180. Mi lehet probléma az UNDO/REDO naplózás esetén? (2 pont)

Az UNDO naplózáshoz hasonlóan most is előfordulhat, hogy a tranzakció a felhasználó számára korrekten befejezettnek tűnik, de még a <T,COMMIT> naplóbejegyzés lemezre kerülése előtt fellépett hiba utáni helyreállítás során a rendszer a tranzakció hatásait semmissé teszi ahelyett, hogy helyreállította volna.

181. Adjuk meg az UR2 szabályt az UNDO/REDO naplózás esetén? (2 pont)

A <T,COMMIT> naplóbejegyzést nyomban lemezre kell írni, amint megjelenik a naplóban.

182. Adjunk meg a működés közbeni ellenőrzőpont képzésének lépéseit UNDO/REDO naplózás esetén! (6 pont)

Írjunk <END CKPT> naplóbejegyzést a naplóba, majd írjuk a naplót lemezre.

183. Adjunk meg a működés közbeni mentés 5 lépését! (5 pont)

(1) A <START DUMP> bejegyzés naplóba írása.

(2) A REDO vagy UNDO/REDO naplózási módnak megfelelő ellenőrzőpont kialakítása.

(3) Az adatlemez(ek) teljes vagy növekményes mentése.

(4) A napló mentése. A mentett naplórész tartalmazza legalább a 2. pontbeli ellenőrzőpont-képzés közben keletkezett naplóbejegyzéseket, melyeknek túl kell élniük az adatbázist hordozó eszköz meghibásodását.

(5) <END DUMP> bejegyzés naplóba írása.

184. Az Oracle milyen naplózást valósít meg? (2 pont)

Az Oracle az UNDO és a REDO naplózás egy speciális keverékét valósítja meg.

185. Az Oracle mit használ UNDO naplózás céljára? (3 pont)

A tranzakciók hatásainak semmissé tételéhez szükséges információkat a rollback szegmensek tartalmazzák. Minden adatbázisban van egy vagy több rollback szegmens, amely a tranzakciók által módosított adatok régi értékeit tárolja attól függetlenül, hogy ezek a módosítások lemezre íródtak vagy sem. A rollback szegmenseket használjuk az olvasási konzisztencia biztosítására, a tranzakciók visszagörgetésére és az adatbázis helyreállítására is.

186. Az Oracle mit használ REDO naplózás céljára? (2 pont)

A helyreállítás a napló (redo log) alapján történik. A napló olyan állományok halmaza, amelyek az adatbázis változásait tartalmazzák, akár lemezre kerültek, akár nem. Két részből áll: az online és az archivált naplóból.

187. Mit tartalmaz az Oracle rollback szegmense? (4 pont)

A naplóbejegyzések ideiglenesen az SGA (System Global Area) memóriapuffereiben tárolódnak, amelyeket a Log Writer (LGWR) háttérfolyamat folyamatosan ír ki lemezre. (Az SGA tartalmazza az adatbáziselemeket tároló puffereket is, amelyeket pedig a Database Writer háttérfolyamat ír lemezre.)

188. Milyen problémát kell megoldania a konkurenciavezérlésnek? (4 pont)

A tranzakciók közötti egymásra hatás az adatbázis-állapot inkonzisztenssé válását okozhatja, még akkor is, amikor a tranzakciók külön-külön megőrzik a konzisztenciát, és rendszerhiba sem történt.

189. Mit hívunk ütemezőnek? (2 pont)

A tranzakciós lépések szabályozásának feladatát az adatbázis-kezelő rendszer ütemező (scheduler) része végzi.

190. Mit hívunk ütemezésnek? (2 pont)

Az ütemezés (schedule) egy vagy több tranzakció által végrehajtott lényeges műveletek időrendben vett sorozata, amelyben az egy tranzakcióhoz tartozó műveletek sorrendje megegyezik a tranzakcióban megadott sorrenddel.

191. Milyen 2 módon biztosítja az ütemező a sorbarendezhetőséget? (2 pont)

Várakoztat, abortot rendel el, hogy a sorba- rendezhetőséget biztosítsa.

192. Mit hívunk konfliktuspárnak? (2 pont)

A konfliktus (conflict) vagy konfliktuspár olyan egymást követő műveletpár az ütemezésben, amelynek ha a sorrendjét felcseréljük, akkor legalább az egyik tranzakció viselkedése megváltozhat.

193. Milyen 3 esetben nem cserélhetjük fel a műveletek sorrendjét, mert inkonzisztenciát okozhatna? (3 pont)

(1) $r_i(X)$; $w_i(Y)$ konfliktus
(2) $w_i(X)$; $w_j(X)$ konfliktus
(3) $r_i(X)$; $w_j(X)$ és $w_i(X)$; $r_j(X)$ is konfliktus.

194. Mikor konfliktusekvivalens 2 ütemezés? (2 pont)

Azt mondjuk, hogy két ütemezés konfliktusekvivalens (conflict-equivalent), ha szomszédos műveletek nem konfliktusos cseréinek sorozatával az egyiket átalakíthatjuk a másikká.

195. Mikor konfliktus-sorbarendezhető egy ütemezés? (2 pont)

Azt mondjuk, hogy egy ütemezés konfliktus-sorbarendezhető (conflict-serializable schedule), ha konfliktusekvivalens valamely soros ütemezéssel.

196. Mikor konfliktus-sorbarendezhetőség elve? (3 pont)

Nem konfliktusos cserékkel az ütemezést megpróbáljuk soros ütemezéssé átalakítani. Ha ezt meg tudjuk tenni, akkor az eredeti ütemezés sorbarendezhető volt, ugyanis az adatbázis állapotára való hatása változatlan marad minden nem konfliktusos cserével.

197. Mi a kapcsolat a sorbarendezhetőség és a konfliktus-sorbarendezhetőség között? (2 pont)

Azt mondjuk, hogy egy ütemezés konfliktus-sorbarendezhető (conflict-serializable schedule), ha konfliktusekvivalens valamely

soros ütemezéssel. A konfliktus-sorbarendezhetőség elégséges feltétel a sorbarendezhetőségre, vagyis egy konfliktus-sorbarendezhető ütemezés sorbarendezhető ütemezés is egyben.

198. Az $r_1(A)$; $w_1(A)$; $r_2(A)$; $w_2(A)$; $r_1(B)$; $w_1(B)$; $r_2(B)$; $w_2(B)$; ütemezést alakítsuk soros ütemezéssé (5 pont)

1. $r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B);$
2. $r_1(A); w_1(A); r_2(A); r_1(B); w_2(A); w_1(B); r_2(B); w_2(B);$
3. $r_1(A); w_1(A); r_1(B); r_2(A); w_2(A); w_1(B); r_2(B); w_2(B);$
4. $r_1(A); w_1(A); r_1(B); r_2(A); w_1(B); w_2(A); r_2(B); w_2(B);$
5. $r_1(A); w_1(A); r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B);$

- **199. Adjunk példát sorbarendezhető, de nem konfliktus-sorbarendezhető ütemezésre (4 pont)**

w1(Y); w2(Y); w2(X); w1(X); w3(X);

Intuíció alapján átgondolva annak, hogy T_1 és T_2 milyen értéket ír X-be, nincs hatása, ugyanis T_3 felülírja X értékét. Emiatt S_1 és S_2 X-nek is és Y-nak is ugyanazt az értéket adja. Mivel S_1 soros ütemezés, és S_2-nek bármely adatbázis-állapotra ugyanaz a hatása, mint S_1-nek, ezért S_2 sorbarendezhető.
Ugyanakkor mivel nem tudjuk felcserélni $w_1(X)$-et $w_2(X)$-szel, így cseréken keresztül nem lehet S_2-t valamelyik soros ütemezéssé átalakítani. Tehát S_2 sorba- rendezhető, de nem konfliktus-sorbarendezhető.

200. Mi a konfliktus-sorbarendezhetség tesztelésének alapötlete? (2 pont)

Ha valahol konfliktusban álló műveletek szerepelnek S-ben, akkor az ezeket a műveleteket végrehajtó tranzakcióknak ugyanabban a sorrendben kell előfordulniuk a konfliktus-ekvivalens soros ütemezésekben, mint ahogyan az S-ben voltak.

201. Mikor mondjuk, hogy egy S ütemezés alapján T_1 *megelőzi* T_2 -t? (5 pont)

Adott a T1 és T2, esetleg további tranzakcióknak egy S ütemezése. Azt mondjuk, hogy T1 megelőzi T2-t, ha van a T1-ben olyan A1 művelet és a T2-ben olyan A2 művelet, hogy

- A1 megelőzi A2-t S-ben,
- A1 és A2 ugyanarra az adatbáziselemre vonatkoznak, és
- A1 és A2 közül legalább az egyik írás művelet.

202. Adjuk meg egy S ütemezéshez tartozó megelőzési gráf definícióját! (5 pont)

Ezeket a megelőzéseket a megelőzési gráfban (precedence graph) összegezhetjük. A megelőzési gráf csúcsai az S ütemezés tranzakciói. Ha a tranzakciókat T_i-vel jelöljük, akkor a T_i-nek megfelelő csúcsot az i egész jelöli. Az i csúcsból a j csúcsba akkor vezet irányított él, ha $T_i <_S T_j$.

203. Milyen kapcsolat van a konfliktusekvivalencia és a megelőzési gráfok között? (4 pont)

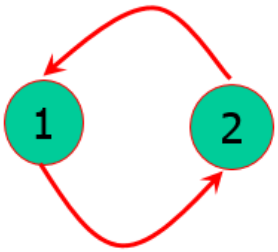
S1, S2 konfliktusekvivalens => gráf(S1) = gráf(S2)

Megjegyzés:
gráf(S_1)=gráf(S_2) $\not\Rightarrow$ S_1, S_2 konfliktusekvivalens

204. Adjunk példát arra, hogy két ütemezés megelőzési gráfja megegyezik, de nem konfliktusekvivalensek! (4 pont)

$S_1 = w_1(A)\ r_2(A)\ w_2(B)\ r_1(B)$

$S_2 = r_2(A)\ w_1(A)\ r_1(B)\ w_2(B)$



Nem lehet semmit sem cserélni!

205. Mit hívunk egy irányított, körmentes gráf esetében a csúcsok topologikus sorrendjének? (4 pont)

Egy körmentes gráf csúcsainak topologikus sorrendje a csúcsok bármely olyan rendezése, amelyben minden a -> b élre az a csúcs megelőzi a b csúcsot a topologikus rendezésben.

206. Hogyan lehet tesztelni a megelőzési gráf alapján egy ütemezés konfliktus-sorbarendezhetőségét? (4 pont)

Ha az S megelőzési gráf tartalmaz irányított kört, akkor S nem konfliktus-sorbarendezhető, ha nem tartalmaz irányított kört, akkor S konfliktus-sorbarendezhető, és a csúcsok bármelyik topologikus sorrendje megadja a konfliktusekvivalens soros sorrendet.

207. Mi jellemzős a passzív ütemezésre? (4 pont)

- hagyjuk a rendszert működni,
- az ütemezésnek megfelelő gráfot tároljuk,
- egy idő után megnézzük, hogy van-e benne kör,
- és ha nincs, akkor szerencsénk volt, jó volt az ütemezés.

208. Mi jellemzős az aktív ütemezésre és milyen 3 módszert lehet erre használni? (5 pont)

Az ütemező beavatkozik, és megakadályozza, hogy kör alakuljon ki.

- zárak
- időbélyegek
- érvényesítés