**Adatbázisok II.**

**III. Ellenőrző kérdések**

139. A legjobb átfutás mit optimalizál? (2 pont)

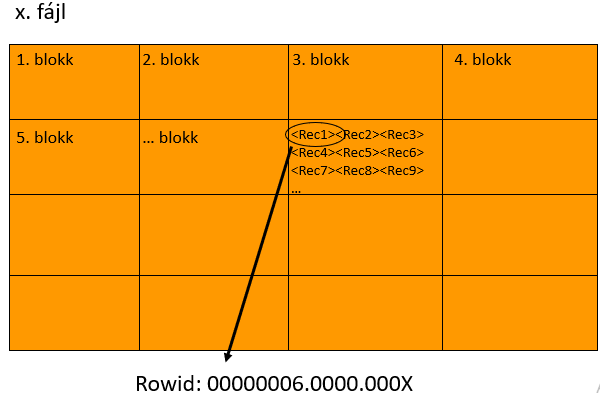
* Az adatstatisztikák szerepet játszanak a terv meghatározásában
  + **Legjobb átfutás**: minden sort minél hamarabb – először számoljon, aztán gyorsan térjen vissza.
  + Legjobb válaszidő: az első sort minél hamarabb – számítás közben már térjen vissza

140. A legjobb válaszidő mit optimalizál? (2 pont)

* Az adatstatisztikák szerepet játszanak a terv meghatározásában
  + Legjobb átfutás: minden sort minél hamarabb – először számoljon, aztán gyorsan térjen vissza.
  + **Legjobb válaszidő**: az első sort minél hamarabb – számítás közben már térjen vissza

141. Adjuk meg a ROWID szerkezetét, és egy példát is rá Oracle esetében! (2 pont)

* Az Oracle az összes adatot adatfájlokban tárolja
* Az adatfájl adatblokkokból áll
* Az adatblokkok sorokat tartalmaznak
* Minden sort a blokkban elfoglalt helye azonosít
  + ROWID: <Blokk>.<Sor>.<Fájl>
  + Példa:



142. Mi az “Explain plan for<SQL-utasítás>” utasítás hatása? (2 pont)

* Elmenti a tervet (sorforrások + műveletek) Plan\_Table-be
* (Plan\_Table nézete (vagy külső eszköz) formázza olvasható tervvé

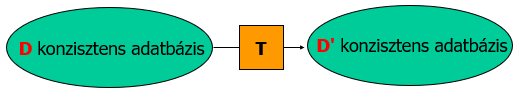
143. Jellemezzük a SELECT \*FROM emp WHERE rowid= ‘00004F2A.00A2.000C’ utasítást! (4 pont)

* Táblaelérés rowid alapján (Egyetlen tábla, nincs index)
  + Egy sor megkeresése
  + Azonnal a blokkra megy és kiszűri a sort
  + A leggyorsabb módszer egy sor kinyerésére ha tudjuk a rowid-t

144. Mit jelent a konzisztens állapot és mit jelent a konzisztens adatbázis? (2 pont)

* Konzisztens állapot: kielégíti az összes feltételt (megszorítást)
* Konzisztens adatbázis: konzisztens állapotú adatbázis
* *MJ.:*
  + Konzisztencia: az adtok előre megadott feltételeket, predikátumokat elégítenek ki.
  + Pl.: megengedett értékek korlátozása
  + Az adatbázis a valóságot próbálja reprezentálni, de minden összefüggést (vagyis a valóság teljes szemantikáját) lehetetlen megadni
    - Vegyük észre: az adatbázis nem lehet állandóan konzisztens

145. Mit hívunk tranzakciónak és mi jellemző rá? (4 pont)

* Tranzakció: konzisztenciát megtartó adatkezelő műveletek sorozata  
  
* Ezek után mindig feltesszük:

Ha T tranzakció konzisztens állapotból indul és T tranzakció csak egyedül fut le 🡪 T konzisztens állapotban hagyja az adatbázist

146. Mit jelent a tranzakció atomossági tulajdonsága? (2 pont) *(lent, 1. vázlatpont)*

147. Mit jelent a tranzakció konzisztencia tulajdonsága? (2 pont)*(lent, 2. vázlatpont)*

148. Mit jelent a tranzakció elkülönítési tulajdonsága? (2 pont) *(lent, 3. vázlatpont)*

149. Mit jelent a tranzakció tartóssági tulajdonsága? (2 pont) *(lent, 4. vázlatpont)*

* A tranzakció az adatbázis-műveletek végrehajtási egysége, amely DML(Data Manipulation Language – Adatmanipulációs nyelv)-beli utasításokból áll, és a következő tulajdonságokkal rendelkezik:
  + **(146) (A) Atomosság (atomicity):** a tranzakció „mindent vagy semmit” jellegű végrehajtása (vagy teljesen végrehajtjuk, vagy egyáltalán nem hajtjuk végre)
  + **(147) (C) Konzisztencia: (consistency**): az a feltétel, hogy a tranzakció megőrizze az adatbázis konzisztenciáját, azaz a tranzakció végrehajtása után is teljesüljenek az adatbázisban előírt konzisztenciamegszorítások (integritási megszorítások), azaz az adatelemekre és a közöttük lévő kapcsolatokra vonatkozó elvárások.
  + **(148) (I) Elkülönítés (isolation**): az a tény, hogy minden tranzakciónak látszólag úgy kell lefutnia, mintha ez alatt az idő alatt semmilyen másik tranzakciót sem hajtanánk végre.
  + **(149) (D) Tartósság (durability**): az a feltétel, hogy ha egyszer egy tranzakció befejeződött, akkor már soha többé nem veszhet el a tranzakciónak az adatbázison kifejtett hatása.
* Ezek a tranzakció ACID-tulajdonságai

150. A tranzakció-feldolgozónak milyen három feladata van? (3 pont)

* A tranzakciófeldolgozó a következő 3 feladatot hajtja végre:
  + naplózás
  + konkurenciavezérlés
  + holtpont feloldása

151. A tranzakciók melyik tulajdonságát biztosítja a naplózás? (1 pont)

* Naplózás
  + Annak érdekében, hogy **a tartósságot** biztosítani lehessen, az adatbázis minden változását külön feljegyezzük (naplózzuk) lemezen.
  + A naplókezelő (log managger) többféle eljárásmód közül választja ki azt, amelyet követni fog.
  + Ezek az eljárásmódok biztosítják azt, hogy teljesen mindegy, mikor történik a rendszerhiba vagy a rendszer összeomlása, a helyreállítás-kezelő meg fogja tudni vizsgálni a változások naplóját, és ez ez alapján vissza tudja állítani az adatbázist valamilyen konzisztens állapotba.
  + A naplókezelő először a pufferekbe írja a naplót, és egyeztet a pufferkezelővel, hogy a pufferek alkalmas időpillanatokban garantáltan íródjanak ki lemezre, ahol már az adatok túlélhetik a rendszer összeomlását.

152. A tranzakciók melyik tulajdonságát biztosítja a konkurenciakezelés? (1 pont)

153. Mi az ütemező feladata? (2 pont)

* Konkurenciavezérlés/konkurenciakezelés
  + (**152**,) A tranzakcióknak úgy kell látszódniuk, mintha egymástól függetlenül, **elkülönítve** végeznénk el őket. (Elkülönítés – isolation)
  + (**153**.) Az ütemező (konkurenciavezérlés-kezelő) feladata, hogy **meghatározza az összetett tranzakciók résztevékenységeink egy olyan sorrendjét, amely biztosítja azt, hogy ha ebben a sorrendben hajtjuk végre a tranzakciók elemi tevékenységeit, akkor az összehatás megegyezik azzal, mintha a tranzakciókat egyenként és egységes egészként hajtottuk volna végre**.
  + *Példa:*
    - 
    - A két utasítássorozat nem elkülönülve jön, hanem összefésülődve: 
    - A saját sorrend megmarad mindkettőn belül.
    - ekkor olyan állapot is kialakulhat, ami nem jött volna létre, ha egymás után futnak le a tranzakciók. (például egy tábla egy bizonyos értékét nem mindegy, hogyan változtatjuk)
    - A tipikus ütemező ezt amunkát azáltal látja el, hogy az adatbázis bizonyos részeire elhelyezett zárakat (lock) karbantartja.
    - Ezek a zárak megakadályoznak két tranzakciót abban, hogy rossz kölcsönhatással használják ugyanazt az adatrészt. A zárakat rendszerint a központi memória zártáblájában (lock table) tárolja a rendszer.
    - Az ütemező azzal befolyásolja a lekérdezések és más adatbázis-műveletek végrehajtását, hogy megtiltja a végrehajtómotornak, hogy hozzányúljon az adatbázis zár alá helyezett részeihez.
* (*Holtpont feloldása*
  + *A tranzakciók az ütemező által engedélyezett zárak alapán versenyeznek az erőforrásokért. így előfordulhat, hogy olyan helyzetbe kerülnek, amelyben egyiküket sem lehet folytatni, mert mindegyiknek szüksége lenne valamire, amit egy másik tranzakció birtokol.*
  + *A tranzakciókezelő feladata, hogy ilyenkor közbeavatkozzon, és töröljön (abortáljon) egy vagy több tranzakciót úgy, hogy a többit már folytatni lehessen.)*

154. Mitől sérülhet a konzisztencia? (4 pont)

* **Tranzakcióhiba** (hibásan megírt, rosszul ütemezett, félbehagyott tranzakciók.)
* **Adatbázis-kezelési hiba** ( az adabázis-kezelő valamelyik komponense nem, vagy rosszul hajtja végre a feladatát.)
* **Hardverhiba** (elvész egy adat, vagy megváltozik az értéke)
* **Adatmegosztásból származó hiba**
  + *pl:*
    - T1: 10% fizetésemelést ad minden programozónak
    - T2: minden programozót átnevez rendszerfejlesztőnek

155. A belső társérülés elleni védekezés milyen két lépésből áll? (4 pont)

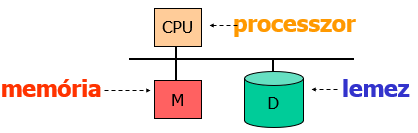
* Azt mindig feltesszük, hogy a háttértár nem sérül, azaz csak a memória, illetve a puffer egy része száll el. Az ilyen belső társérülés elleni védekezés két részből áll:
  1. Felkészülés a hibára: naplózás
  2. Hiba után helyreállítás: a napló segítségével egy konzisztens állapot helyreállítása
* *MJ.*: Természetesen a naplózás és a hiba utáni helyreállítás összhangban vannak, de van több különböző naplózási protokoll (és ennek megfelelően helyreállítás)

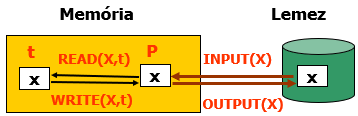
156. Mit hívunk adatbáziselemnek (/adategységnek)? (2 pont)

* *Feltesszük, hogy az adatbázis adategységekből, elemekből áll.*
* Az adatbáziselem (database element**) a fizikai adatbázisban tárolt adatok egyfajta funkcionális egysége, amelynek értékét tranzakciókkal lehet elérni (kiolvasni) vagy módosítani (kiírni)**
* *MJ*.: Az adatbáziselem lehet:
  + reláció
  + relációsor
  + lemezblokk
  + lap

157. A tranzakció és az adatbázis kölcsönhatásának milyen három fontos helyszíne van? (3 pont)

* A tranzakció és az adatbázis kölcsönhatásának 3 fontos helyszíne van:
  1. Az adatbázis elemeit tartalmazó lemezblokkok területe (D)
  2. A pufferkezelő által használt virtuális vagy valós memóriaterület (M)
  3. a tranzakció memóriaterülete (M)



* *MJ.:*
  1. Olvasás
     + Ahhoz, hogy a tranzakció egy X adatbáziselemet beolvashasson, azt előbb memóriapuffer(ek)be (P) kell behozni, ha még nincs ott
     + Ezt követően tudja a puffer(ek) tartalmát a tranzakció a saját memóriaterületére (t) beolvasni  
       
  2. Írás
     + Az adatbáziselem új értékének kiírása fordított sorrendben történik: az új értéket a tranzakció alakítja ki a saját memóriaterületén, majd ez az új érték másolódik át a megfelelő puffer(ek)be
     + Fontos, hogy egy tranzakció sohasem módosíthatja egy adatbáziselem értékét közvetlenül a lemezen!

158. Mit jelent az INPUT(X) művelet? (2 pont) (1.)

159. Mit jelent a READ(X,t) művelet? (4 pont) (2.)

160. Mit jelent a Write(X,t) művelet? (4 pont) (3.)

161. Mit jelent az Output(X) művelet? (2 pont) (4.)

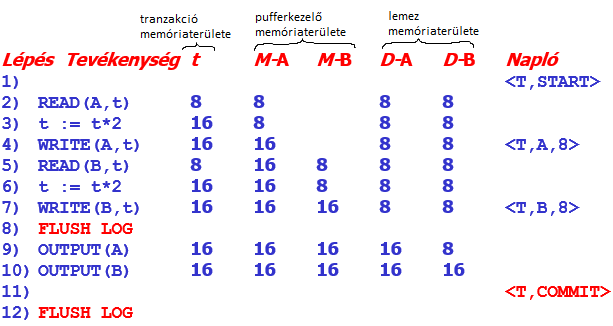
* Az adatmozgások alapműveletei:
  1. (**158.) INPUT(X):** Az X adatbáziselemet tartalmazó lemezblokk másolása a memóriapufferbe.
  2. (**159**.)**READ(X, t):** Az X adatbáziselem bemásolása a tranzakció t lokális változójába.
     + Részletesebben: ha az X adatbáziselemet tartalmazó blokk nincs a memóriapufferben, akkor előbb végrehajtódik INPUT(X). Ezután kapja meg a t lokális változó X értékét.
  3. (**160**.) **WRITE(X, t):** A t lokális változó tartalma az X adatbáziselem memóriapufferbeli tartalmába másolódik.
     + Részletesebben: ha az X adatbáziselemet tartalmazó blokk nincs a memóriapufferben, akkor előbb végrehajtódik INPUT(X). Ezután másolódik át a t lokális változó értéke a pufferbeli X-be.
  4. (**161**.) **OUTPUT(X):** Az X adatbáziselemet tartalmazó puffer kimásolása lemezre.

162. Adjuk meg az Undo naplózás U1 és U2 szabályát! (4 pont)

* *MJ.: Undo és Redo naplózás*
  + Redo (helyreállító) naplózás: A katasztrófák hatásának kijavítását követően a tranzakciók hatását meg kell ismételni, és az általuk adatbázisba írt új értékeket ismételten ki kell írni.
  + Undo (semmiségi) : Egyes tranzakciók hatását viszont vissza kívánjuk vonni, azaz kérjük az adatbázis visszaállítását olyan állapotba, mintha a tekintett tranzakció nem is működött volna.
* Undo naplózás szabályai
  + **U1**. Ha a T tranzakció módosítja az X adatbáziselemet, akkor a (T, X, régi érték) naplóbejegyzést azelőtt kell a lemezre írni, mielőtt az X új értékét a lemezre írná a rendszer.
  + **U2**. Ha a tranzakció hibamentesen befejeződött, akkor a COMMIT naplóbejegyzést csak azután szabad a lemezre írni, ha a tranzakció által módosított összes adatbáziselem már a lemezre íródott, de ezután rögtön.

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

* MJ.: Undo naplózás esetén a lemezre írás sorrendje
  1. Az adatbázis módosítására vonatkozó naplóbejegyzések
  2. maguk a módosított adatbáziselemek
  3. a COMMIT naplóbejegyzés
* Megjegyzések:
  + - Az első lépés minden módosított adatbáziselemre vonatkozóan önmagában, külön-külön végrehajtandó (nem lehet a tranzakció több módosítására csoportosan megtenni! )
    - A naplóbejegyzések lemezre írásának kikényszerítésére a naplókezelőnek szüksége van **a FLUSH LOG** műveletre, mely felszólítja a pufferkezelőt az összes korábban még ki nem írt naplóblokk lemezre való kiírására.



164. Adjuk meg Undo naplózás esetén a helyreállítás algoritmusát! (8 pont)



* MJ: Ha hiba történt 🡪 konzisztens állapot visszaállítása 🡪 nem befejezett tranzakciók hatásának törlése

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

* MJ.: Az ellenőrzőpont-képzés
  1. A CHECKPOINT megadja, hogy meddig kell visszaolvasnunk a naplózásban egy esetleges hiba bekövetkeztét követő helyreállítás során (úgynevezett ellenőrzőpont)
  2. Képzése (63.o.)
     + Megtiltjuk az új tranzakciók indítását.
     + Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér.
     + A naplót a pufferből a háttértárra írjuk (FLUSH LOG)
     + A naplóba beírjuk, hogy CHECKPOINT.
     + A naplót újra a háttértárra írjuk: FLUSH LOG.
     + Újra fogadjuk a tranzakciókat.
  3. Ezután nyilván elég a CHECKPOINT-ig visszamenni, hiszen előtte minden Ti már valahogy befejeződött.
  4. Probléma: Hosszú ideig tarthat, amíg az aktív tranzakciók befejeződnek. (Új tranzakciókat sokáig nem lehet kiszolgálni) 🡪 Megoldás: CHECKPOINT képzése működés közben
* **A módszer lépései: (működés közbeni ellenőrzőpont-képzés)**
  + - ,

166. Ha UNDO naplózás utáni helyreállításkor előbb <ENDCKPT> naplóbejegyzéssel találkozunk, akkor meddig kell visszamenni a napló olvasásában? (2 pont) (1.)

167. Ha UNDO naplózás utáni helyreállításkor előbb <STARTCKPT(T1,…,Tk)>naplóbejegyzéssel találkozunk, akkor meddig kell visszamenni a napló olvasásában? (2 pont) (2.)

* Két eset fordulhat elő aszerint, hogy visszafelé olvasva a naplót az naplóbejegyzést találjuk előbb:
  1. **(166.)** Ha előbb az naplóbejegyzéssel találkozunk
     + tudjuk, hogy az összes még be nem fejezett tranzakcióra vonatkozó naplóbejegyzést a legközelebbi korábbi naplóbejegyzésig megtaláljuk
     + ott viszont megtaláljuk, az annál korábbiakat akár el is dobhatjuk.
  2. **(167.)** Ha a naplóbejegyzéssel találkozunk előbb, az azt jelenti, hogy a katasztrófa az ellenőrzőpont-képzés közben fordult elő, ezért tranzakciók nem fejeződtek be a hiba fellépéséig.
     + Ekkor a be nem fejezett tranzakciók közül a legkorábban (t) kezdődött tranzakció indulásáig kell a naplóban visszakeresünk, annál korábbra nem.

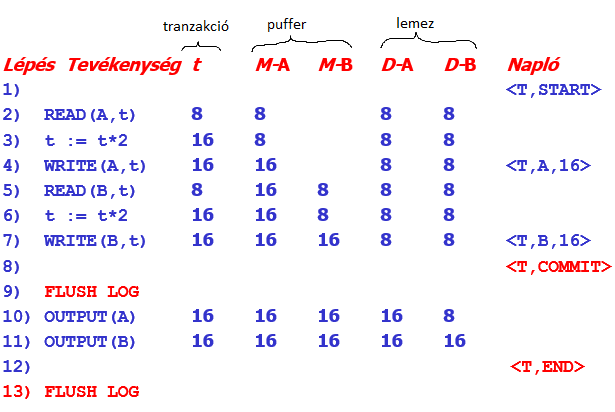
168. Adjuk meg a REDO naplózás esetén a lemezre írás sorrendjét 5 lépésben! (5 pont)

* MJ. Ismétlés: Undo és Redo naplózás
  + Redo (helyreállító) naplózás: A katasztrófák hatásának kijavítását követően a tranzakciók hatását meg kell ismételni, és az általuk adatbázisba írt új értékeket ismételten ki kell írni.
  + Undo (semmiségi) : Egyes tranzakciók hatását viszont vissza kívánjuk vonni, azaz kérjük az adatbázis visszaállítását olyan állapotba, mintha a tekintett tranzakció nem is működött volna.
* A helyrehozó (REDO) naplózás esetén a lemezre írás sorrendje
  1. Ha egy Tranzakció -re módosítja egy X adatbáziselem értékét, akkor egy bejegyzést kell a naplóba írni.
  2. Az adatbáziselemek módosítását leíró **naplóbejegyzések lemezre írása**.
  3. A naplóbejegyzés lemezre írása. (2. és 3. egy lépésben történik)
  4. Az adatbáziselemek értékének **cseré**je a lemezen.
  5. A -t bejegyezzük a naplóba, majd **kiírjuk a lemezre a naplót**.

169. Adjuk meg a REDO naplózás esetén az R1 szabályt! (2 pont) (A helyrehozó naplózás szabálya)

R1. Mielőtt az adatbázis bármely elemét a lemezen módosítanánk, az módosítására vonatkozó összes   
 naplóbejegyzésnek, azaz -nek és -nak a lemezre kell kerülnie.

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



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

**Helyreállítás a REDO naplóból:**



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

**Helyreállítás a módosított REDO naplóból:**

* **Nem használunk bejegyzést a befejezett tranzakciókra, helyette a be nem fejezetteket jelöljük meg -tal. (módosított REDO napló).**
* Mj.: Folyamatának lépései
  1. Meghatározzuk a befejezett tranzakciókat (COMMIT).
  2. Elemezzük a naplót az elejétől kezdve. Minden naplóbejegyzés esetén:
     1. Ha be nem fejezett tranzakció, akkor nem kell tenni semmit.
     2. Ha befejezett tranzakció, akkor értéket kell írni az adatbáziselembe.
  3. Minden be nem fejezett tranzakcióra vonatkozóan naplóbejegyzést kell a naplóba írni, és a naplót ki kell írni a lemezre ().

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

* REDO különbsége az UNDO protokollhoz képest:
  + Az adat változás utáni értékét jegyezzük fel a naplóba.
  + Máshová rakjuk a -ot: a kiírás elé 🡪 megtelhet a puffer
  + Az UNDO protokoll esetleg túl gyakran akar írni 🡪 itt el lehet halasztani az írást

174. Mit hívunk piszkos puffereknek? (1 pont)

* MJ.: Helyrehozó naplózás ellenőrzőpont-képzés használatával
  + Új probléma: a befejeződött tranzakciók módosításainak lemezre írása a befejeződés után sokkal később is történhet.
  + Következmény: ugyanazon pillanatban aktív tranzakciók számát nincs értelme korlátozni, tehát nincs értelme az egyszerű ellenőrzőpont-képzésnek.
  + A kulcsfeladat – amit meg kell tennünk az ellenőrzőpont-készítés kezdete és befejezése közötti időben – az összed olyan adatbáziselem lemezre való kiírása, melyeket befejezett tranzakciók módosítottak, és még nem voltak lemezre írva.
  + Ennek megvalósításához a pufferkezelőnek nyilván kell tartani a piszkos puffereket (dirty buffers) melyekben már végrehajtott, de lemezre még ki nem írt módosításokat tárol. Azt is tudnunk kell, hogy mely tranzakciók mely puffereket módosíották.
* **Piszkos pufferek (dirty buffers):** olyan pufferek, melyek már végrehajtott, de lemezre még ki nem írt módosításokat tárolnak.

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

Helyrehozó naplózás ellenőrzőpont-képzés használatával

* MJ.: Be tudjuk fejezni az ellenőrzőpont-képzést az aktív tranzakciók (normális vagy abnormális) befejezésének kivárása nélkül, mert ők ekkor még amúgy sem engedélyezik lapjaik lemezre írását.
* **Lépései:**
  1. naplóbejegyzés elkészítése és lemezre írása, ahol az összes éppen aktív tranzakció.
  2. Az összes olyan adatbáziselem kiírása lemezre, melyeket olyan tranzakciók írtak a pufferekbe, melyek a naplóba írásakor már befejeződtek, de puffereik lemezre még nem kerültek.
  3. bejegyzés naplóba írása, és a napló lemezre írása.

176. Adjuk meg az UNDO/REDO naplózás esetén az UR1 szabályt! (2 pont) (lent, 3. pont)

177. Adjuk meg az UNDO/REDO naplózás esetén a WAL elvet! (2 pont) (lent, 4. vázlatpont)

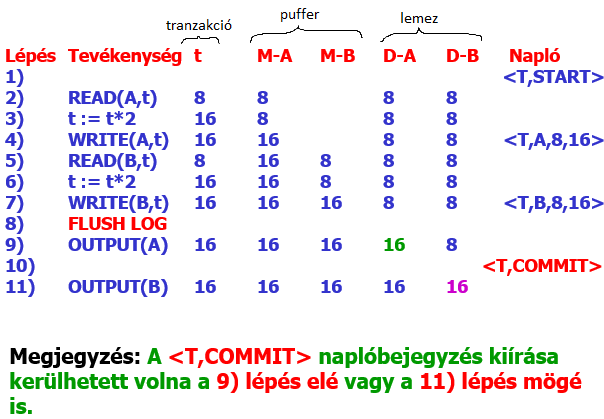
178. Hová kerülhet a COMMIT az UNDO/REDO naplózás esetén? (2 pont) (lent, 5. vázlatpont)

* MJ.: Semmiségi/helyrehozó (undo/redo) naplózás
  + A semmiségi (undo) naplózás esetén az adatokat a tranzakció befejezésekor nyomban lemezre kell írni, nő a végrehajtandó lemezműveletek száma.
  + A helyrehozó (redo) naplózás minden módosított adatbázisblokk pufferben tartását igényli egészen a tranzakció rendes és teljes befejezéséig, így a napló kezelésével nő a tranzakciók átlagos pufferigénye.

Megoldás

* MJ.: a naplóbejegyzés négy komponensű:
  + a naplóbejegyzés azt jelenti, hogy a tranzakció az adatbázis elemének korábbi értékét -re módosította.
* **(176) UR1**: Mielőtt az adatbázis bármely elemének értékét – valamely tranzakció által végzett módosítás miatt – a lemezen módosítanánk, ezt megelőzően a naplóbejegyzésnek a lemezre kell kerülnie.
* **(177) WAL –** Write After Log elv: előbb naplózunk, utána módosítunk
* **(178) Nagyobb szabadság:** A bejegyzés **megelőzheti, de követheti is** az adatbáziselemek lemezen történő bármilyen megváltoztatását.
* **MJ.**: nagyobb méretű napló – régi és új értéket is tároljuk

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



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

* A semmisségi (undo)/helyrehozó (redo) módszer alapelvei a következők:
  1. REDO: A legkorábbitól kezdve állítsuk helyre minden befejezett tranzakció hatását.
  2. UNDO: A legutolsótól kezdve tegyük semmissé minden be nem fejezett tranzakció tevékenységét.

181. Mi lehet probléma az UNDO/REDO naplózás esetén? (2 pont) *(lent, 1. karika vázlatpont)*

182. Adjuk meg az UR2 szabályt az UNDO/REDO naplózás esetén? (2 pont) *(lent, 3. karika vázlatpont)*

* Helyreállítás UNDO/REDO naplózás esetén
* **(181.) Probléma** (befejezett változtatást is megsemmisítünk): Az UNDO naplózáshoz hasonlóan most is előfordulhat, hogy a tranzakció a felhasználó számára korrekten befejezettnek tűnik, de még a naplóbejegyzés lemezre kerülése előtt fellépett hiba utáni helyreállítás során a rendszer a tranzakció hatásait semmissé teszi ahelyett, hogy helyreállította volna.
* Amennyiben ez a lehetőség problémát jelent, akkor a semmisségi/helyrehozó naplózás során egy további szabályt célszerű bevezetni:
* **(182) UR2**: A naplóbejegyzést nyomban lemezre kell írni, amint megjelenik a naplóban.

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

* Semmisségi/helyrehozó naplózás ellenőrzőpont-képzéssel
  + Egyszerűbb, mint a másik két naplózás estén.
  1. Írjunk a naplóba naplóbejegyzést,   
     ahol az aktív tranzakciók,   
     majd írjuk a naplót lemezre.
  2. Írjuk a lemezre az összes piszkos puffert (dirty buffers), tehát azokat, amelyek egy vagy több módosított adatbáziselemet tartalmaznak.  
     A helyrehozó naplózástól eltérően itt az összes piszkos puffert lemezre írjuk, nem csak a már befejezett tranzakciók által módosítottakat.
  3. Írjunk naplóbejegyzést a naplóba, majd írjuk a naplót lemezre.

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

* MJ.: Mentések szintjei
  + Teljes mentés (full dump): amikor az egész adatbázisról másolat készül
  + Növekményes mentés (incremental dump): amikor az adatbázisnak csak azon elemiből készítünk másolatot, melyek az utolsó teljes vagy növekményes mentés óta megváltoztak.
* **Működés közbeni mentés lépései**

1. A 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 a 2. pontbeli ellenőrzőpont-képzés közben keletkezett naplóbejegyzéseket, melyeknek túl kell élniük az adatbázist hordozó eszköz meghibásodását.
5. bejegyzés naplóba írása

* Megjegyzés
  + A mentés befejezésekor eldobhatjuk a nplónak azt a részét, amelyre nincs szükség a 2. pontban végrehajtott ellenőrzőpont-képzéshez tartozó helyreállítási folyamat szabályai szerint.
  + UNDO napló nem használható: mivel UNDO naplózás esetén az OUTPUT műveletek a módosítási bejegyzés naplóba írását követően bármikor lefuthatnak, ezért előfordulhat, olyan eredményt kapunk, mintha egy tranzakció nem atomosan hajtódott volna végre.

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

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

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

* A tranzakciók hatásainak semmisé 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.

187. 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 lemere kerültek, akár nem.
* A napló két részből áll: az online és az archivált (hosszú távú tárolásra szolgál) naplóból.

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

* A rollback szegmens rollback bejegyzésekből áll.
* Egy rollback bejegyzést többek között a megváltozott blokk azonosítóját (fájlsorszám és a fájlon belüli blokkazonosító) és a blokk régi értékét tárolja.
* A rollback bejegyzés mindig előbb kerül a rollback szegmensbe, mint ahogy az adatbázisban megtörténik a módosítás.
* Az ugyanazon tranzakciókhoz tartozó bejegyzések össze vannak láncolva, így könnyen visszakereshetők, ha az adott tranzakciót vissza kell görgetni.