## 8.4. A semmisségi/helyrehozó (undo/redo) naplózás

Láthattuk, hogy a naplózás két különböző megközelítése abban mutat eltérést, hogy a napló az adatbáziselemek értékének módosítása esetén a régi (módosítás előtti) vagy az új (módosítás utáni) érteket tartalmazza. Mindkét módszernek vannak bizonyos hátrányai is:

• A semmisségi (undo) naplózás alkalmazása megköveteli, hogy az adatokat a tranzakció befejezésekor nyomban lemezre írjuk, ezzel (esetleg jelentősen) növeljük a végrehajtandó lemezműveletek számát.

• Másik oldalról, a helyrehozó (redo) naplózás minden módosított adatbázisblokk pufferben tartását igényli, egészen a tranzakció rendes és teljes befejezéséig (commit), a napló kezelésével együtt (esetleg jelentősen) növeli a tranzakciók átlagos pufferigényét.

• Mindkét naplózási módszer az ellenőrzőpont képzése közben ellentétes igényeket támaszt a pufferek lemezre írása szempontjából, kivéve, ha az adatbáziselemek teljes blokkok vagy blokkok sokasága. Például, ha a puffer tartalmaz egy *A* adatbáziselemet, melyet egy rendesen és teljesen befejezett tranzakció módosított, és tartalmaz egy *B* adatbáziselemet is, melyet olyan tranzakció módosított, melyre vonatkozóan a COMMIT bejegyzés még nem került lemezre, akkor az *R*1 szabálynak megfelelően, a puffer lemezre másolását igényeljük *A* miatt, viszont tiltjuk ennek megtételét *B* miatt.

Most a *semmisségi/helyrehozó* (undo/redo)-nak nevezett naplózást vizsgáljuk meg. Ez a módszer a tevékenységek elvégzési sorrendjének rugalmasságát növeli azáltal, hogy bővíti a naplózott információk körét.

### 8.4.1. A semmisségi/helyrehozó (undo/redo) naplózás szabályai

A semmisségi/helyrehozó naplózás, egyetlen különbséggel, ugyanolyan típusú naplóbejegyzéseket használ, mint a naplózás többi módszere. E módszerben az adatbáziselem értékének módosítását leíró naplóbejegyzés négykomponensű. A <*T*,*X*,*v*,*w*> naplóbejegyzés azt jelenti, hogy a *T* tranzakció az adatbázis *X* elemének korábbi *v* értékét *w*-re módosította. A semmisségi/helyrehozó naplózást alkalmazó rendszer a következő előírást kell hogy betartsa:

*UR*1: Mielőtt az adatbázis bármely *X* elemének értékét – valamely *T* tranzakció által végzett módosítás miatt – a lemezen módosítanánk, ezt megelőzően a <*T*,*X*,*v*,*w*> módosítást leíró naplóbejegyzésnek lemezre kell kerülnie.

A semmisségi/helyrehozó naplózás, *UR*1 szabálya csak azokat a feltételeket kényszeríti, amelyek a semmisségi és a helyrehozó naplózási szabályok mindegyikében szerepelnek. Speciálisan, a <COMMIT *T*> bejegyzés megelőzheti és követheti is az adatbáziselemek lemezen történő bármilyen megváltoztatását.

**8.11. példa:** A 8.9. ábra, az utoljára a 8.7. példában látott, *T* tranzakcióhoz tartozó naplóbejegyzések sorrendjének egy változatát mutatja. Megjegyezzük, hogy a módosítást leíró naplóbejegyzések már az *A* és *B* adatbáziselemeknek mind a régi, mind az új értékét tartalmazzák. Ebben a sorozatban a <COMMIT *T*> naplóbejegyzés kiírását az *A* és *B* adatbáziselemek lemezre való írása közé tettük. A 10) lépés kerülhetett volna a 9) lépés elé vagy a 11) lépés mögé is. 

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Lépés | Tevékenység | t | M-A | M-B | D-A | D-B | Napló |
| 11) |  |  |  |  |  |  | <START *T*> |
| 12) | READ(A,t) | 18 | 18 |  | 18 | 18 |  |
| 13) | t := t\*2 | 16 | 18 |  | 18 | 18 |  |
| 14) | WRITE(A,t) | 16 | 16 |  | 18 | 18 | <*T*,*A*,8,16> |
| 15) | READ(B,t) | 18 | 16 | 18 | 18 | 18 |  |
| 16) | t := t\*2 | 16 | 16 | 18 | 18 | 18 |  |
| 17) | WRITE(B,t) | 16 | 16 | 16 | 18 | 18 | <*T*,*B*,8,16> |
| 18) | FLUSH LOG |  |  |  |  |  |  |
| 19) | OUTPUT(A) | 16 | 16 | 16 | 16 | 18 |  |
| 10) |  |  |  |  |  |  | <COMMIT *T*> |
| 11) | OUTPUT(B) | 16 | 16 | 16 | 16 | 16 |  |

**8.9. ábra.** Tevékenységek és naplóbejegyzéseik lehetséges sorrendje semmisségi/helyrehozó naplózás használatakor

### 8.4.2. Helyreállítás a semmisségi/helyrehozó (undo/redo) naplózás használatakor

Amikor a semmisségi/helyrehozó naplózást használjuk, és helyreállításra kényszerülünk, akkor a módosítást leíró naplóbejegyzésben megtaláljuk mind a *T* tranzakció hatásainak semmissé tételéhez szükséges régi, mind a *T* tranzakció hatásainak helyreállításához szükséges új adatbáziselem-értékeket. A semmisségi/helyrehozó módszer alapelvei:

1. A legkorábbitól kezdve állítsuk helyre minden befejezett tranzakció hatásait.

2. A legutolsótól kezdve tegyük semmissé minden be nem fejezett tranzakció cselekedeteit.

Megjegyezzük, hogy mindkét eljárásra szükségünk van. A rugalmasság lehetővé teszi, hogy a COMMIT bejegyzés és a lemezen végrehajtott adatbázis-módosítások egymáshoz viszonyított sorrendje kötetlen legyen, így előfordulhat az is, hogy egy befejezett tranzakció néhány vagy összes változtatása még nem került lemezre, és az is, hogy egy be nem fejezett tranzakció néhány vagy összes változtatása már lemezen is megtörtént.

**8.12. példa:** Tegyük fel, hogy az események a 8.9. ábrán látható sorrendben történtek. A hiba fellépésének időpontja függvényében különböző helyreállítási lehetőségeink vannak.

**A késleltetett véglegesítés problémája**

A semmisségi naplózáshoz hasonlóan a semmisségi/helyrehozó naplózás is olyan viselkedést mutat, hogy a tranzakció a felhasználó számára korrekten befejezettnek tűnik (például: az ügyfél számítógép-hálózaton vásárolt egy repülőjegyet, majd levált a hálózatról), s még a <COMMIT *T*> 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 szabály használatát javasoljuk:

*UR*2 A <COMMIT *T*> naplóbejegyzést nyomban lemezre kell írni, amint megjelenik a naplóban.

Ennek teljesítéséért a 8.9. példánkban a 10) lépés után egy FLUSH LOG lépést kell beiktatnunk.

1. Feltéve, hogy a katasztrófa a <COMMIT *T*> naplóbejegyzés lemezre írását követően fordul elő, ekkor *T*-t befejezett tranzakciónak tekintjük. 16-ot írunk mind az *A*, mind *B* adatbáziselemekbe. Az események jelenlegi sorrendjében *A*-nak már 16 a tartalma, de *B*-nek lehet, hogy nem, aszerint, hogy a hiba a 11) lépés előtt vagy után következett be.

2. Ha a katasztrófa a <COMMIT *T*> naplóbejegyzés lemezre írását megelőzően következett be, akkor *T* befejezetlen tranzakciónak számít. Ez esetben az *A* és *B* adatbáziselemek korábbi értéke, 8 íródik lemezre. Ha a hiba a 9) és 10) lépések között következett be, akkor *A* értéke már 16 volt a lemezen, és emiatt a 8-ra való visszaállítás feltétlenül szükséges. Ebben a konkrét példában a *B* értéke nem igényelne visszaállítást (mert még meg sem változott), ha pedig a hiba a 9) lépés előtt következik be, akkor *A* sem igényelné a visszaállítást. Mivel általában nem lehetünk biztosak abban, vajon a visszaállítás szükséges-e vagy sem, így (a biztonság kedvéért) mindig végre kell hajtanunk a visszaállítást.



### 8.4.3. Semmisségi/helyrehozó naplózás ellenőrzőpont-képzéssel

A működés közbeni ellenőrzőpont-képzés valamivel egyszerűbb a semmisségi/helyrehozó naplózás alkalmazásakor, mint más naplózási módszereknél volt. Csak a következőket kell tennünk:

1. Írjunk a naplóba <START CKPT (*T*1, …, *Tk*)> naplóbejegyzést, ahol *T*1, …, *Tk*-k az összes éppen aktív tranzakciók, majd írjuk a naplót lemezre.

2. Írjuk lemezre az összes *piszkos* puffert, tehát azokat, melyek egy vagy több módosított adatbáziselemet tartalmaznak. A helyrehozó naplózástól eltérően itt az összes piszkos puffert lemezre írjuk, nemcsak a már befejezett tranzakciók által módosítottakat.

előttire, vagy pedig előbb visszaállítjuk, és utána a *T* által írottra rekonstruáljuk. E szituációban egyik út sem helyes, mert a végső adatbázis-állapot nem felel meg egyik – atomosnak elvárt – tranzakció hatásának sem.

A gyakorlatban az adatbázisrendszereknek a módosítások naplózásánál többet kell tenniük. Biztosítaniuk kell, hogy ilyen szituációk ne fordulhassanak elő. A konkurencia kérdéseivel foglalkozó fejezetben vizsgáljuk azt is, mit jelent a *T* és *U* tranzakciók elkülönítése, amivel az ugyanazon *X* adatbáziselemen való kölcsönhatásuk előfordulása elkerülhető. A 10.1. részben kifejezetten az olyan helyzetek megelőzésével foglalkozunk, amikor a *T* tranzakció egy piszkos – más tranzakció által módosított, de még nem véglegesített – *X* adatbáziselemet használna.

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

A 2) ponttal kapcsolatban megjegyezzük, hogy a semmisségi/helyrehozó naplózás által, a lemezre írások sorrendjére vonatkozóan biztosított rugalmasság miatt, megengedhetjük a be nem fejezett tranzakciók adatainak lemezre való kiírását. Így megengedhetjük a teljes blokknál kisebb adatbáziselemek használatát is, melyek közös pufferbe kerülnek. A tranzakciókra vonatkozóan egyetlen előírást kell tennünk:

• A tranzakció semmilyen értéket nem írhat (még a memóriapufferbe sem), amíg biztosak nem vagyunk abban, hogy nem abortál.

Amint a 10.1. részben látni fogjuk, ezt a megszorítást szinte mindig be kell tartani ahhoz, hogy elkerülhessük a tranzakciók közötti inkonzisztens kölcsönhatást. Megjegyezzük, hogy a helyrehozó naplózás használatakor a fenti feltétel nem elégséges, éppen ezért írja elő az *R*1 szabály, hogy ha egy tranzakció *B*-t módosítja, akkor a tranzakcióra vonatkozó COMMIT naplóbejegyzésnek előbb kell lemezre íródnia, s csak azután írhatjuk *B*-t lemezre.

**8.13. példa:** A 8.10. ábra a semmisségi/helyrehozó naplózás alkalmazását mutatja egy, a 8.8. ábrán (helyrehozó naplózás) látottal megegyező esetre. Csak a módosításokat leíró naplóbejegyzéseket cseréltük, megadva bennük a régi és új értékeket. Az egyszerűség kedvéért feltételeztük, hogy a régi érték mindig eggyel kisebb az új értéknél.

<START *T*1>

<*T*1,*A*,4,5>

<START *T*2>

<COMMIT *T*1>

<*T*2,*B*,9,10>

<START CKPT (*T*2)>

<*T*2,*C*,14,15>

<START *T*3>

<*T*3,*D*,19,20>

<END CKPT>

<COMMIT *T*2>

<COMMIT *T*3>

**8.10. ábra.** A semmisségi/helyrehozó (undo/redo) naplózás naplója

Amint a 8.9. példában is, az ellenőrzőpont képzésének kezdetekor *T*2 az egyetlen aktív tranzakció. Minthogy ez a napló semmisségi/helyrehozó napló, így lehetséges, hogy *T*2 által *B*-nek adott új érték, 10, lemezre íródik, ami nem volt megengedett a helyrehozó naplózásban. Most lényegtelen, hogy ez a lemezre írás mikor történik meg. Az ellenőrzőpont képzése alatt biztosan lemezre írjuk *B*-t (ha még nem került oda), mivel minden piszkos (változásban érintett) puffert kiírunk lemezre. Hasonlóan *A*-t – melyet a befejezett *T*1 tranzakció alakított ki – is lemezre fogjuk írni, ha még nem került oda.

Ha a katasztrófa ezen eseménysorozat végén jelentkezik, akkor a *T*2-t és *T*3-at teljesen és rendesen befejezett (COMMIT) tranzakciónak tekintjük. *T*1 tranzakció az ellenőrzőpontnál korábbi. Minthogy <END CKPT> bejegyzést találunk a naplóban, így *T*1-ről biztosan tudjuk, hogy teljesen és rendesen befejeződött, valamint az általa okozott módosítások lemezre íródtak. Ezért, mint a 8.9. példában is, a *T*2 és *T*3 által végzett módosítások helyreállítandók, *T*1 pedig figyelmen kívül hagyható. Amikor olyan tranzakció hatásait állítjuk helyre, mint amilyen a *T*2 is, akkor a naplóban nem kell a <START CKPT (*T*2)> bejegyzésnél korábbra visszatekinteni, mert tudjuk, hogy az ellenőrzőpont-képzést megelőzően *T*2 által végzett módosítások az ellenőrzőpont képzése alatt lemezre íródtak.

Másik példaként, tegyük fel, hogy a katasztrófa éppen <COMMIT *T*3> bejegyzés lemezre írását megelőzően fordult elő. Ekkor *T*2-t befejezett, *T*3-at pedig befejezetlen tranzakciónak kell tekintenünk. *T*2 tevékenységét helyreállítandó *C* értékét a lemezen 15-re írjuk; *B*-t már nem kell 10-re írnunk a lemezen, mert tudjuk, hogy ez már lemezre került az <END CKPT> előtt. A helyreállító naplózástól eltérő módon pedig a *T*3 hatásait semmissé tesszük, azaz a lemezen *D* tartalmát 19-re írjuk. Ha *T*3 az ellenőrzőpont-képzés kezdetekor már aktív tranzakció lett volna, akkor a naplóban a megelőző START CKPT bejegyzésig kellene visszakeresnünk, azért hogy megtaláljuk *T*3 semmissé teendő tevékenységeit (az azokat leíró naplóbejegyzéseket). 