## 8.2. Semmisségi (undo) naplózás

A naplózás tanulmányozását annak elemzésével kezdjük, hogy milyen úton biztosítható a tranzakciók atomossága – ami az adatbázisra nézve abban mutatkozik meg, hogy a tranzakciót vagy teljes egészében végrehajtjuk, vagy egyáltalán nem. A *napló* nem más, mint a *naplóbejegyzések* (log records) sorozata, melyek mindegyike arról tartalmaz valami információt, hogy mit tett egy tranzakció. A tranzakciók tevékenysége nyomon követhető azáltal, hogy a tranzakció működésének hatása lépésenként naplózódik, ugyanez történik az összes tranzakcióval. A tranzakciók nyomkövetése bonyolultabbá teszi a naplózást; nem elegendő egyszerűen a tranzakció végén a tranzakció történetének naplózása.

Ha rendszerhiba fordul elő, akkor a napló segítségével rekonstruálható, hogy a tranzakció mit tett a hiba fellépéséig. A naplót – az archívmentéssel együtt – használhatjuk akkor is, amikor eszközhiba keletkezik a naplót nem tároló lemezen. Általánosságban a katasztrófák hatásának kijavítását követően a tranzakciókat meg kell ismételni, és az általuk adatbázisba írt új értékeket ismételten ki kell írni. Egyes tranzakciók a munkájukat vissza kívánják vonni, azaz kérik az adatbázis visszaállítását olyan állapotba, mintha a tekintett tranzakció nem is működött volna.

Az általunk vizsgált első naplózási stílus, melyet *semmisségi* (undo) *naplózásnak*[[1]](#footnote-1) neveznek, csak az utóbbi típusú helyreállításra alkalmas. Ha nem teljesen biztos, hogy a tranzakció hatásai teljesen befejeződtek és lemezen tárolódtak, akkor minden olyan változtatást, melyet a tranzakció tehetett az adatbázisban, semmissé kell tenni, azaz az adatbázist vissza kell állítani a tranzakció működése előtti állapotába.

Ebben a részben a naplóbejegyzések alapelgondolását kívánjuk bemutatni, beleértve a tranzakció teljes és hibátlan befejezését, a *véglegesítési* (commit) tevékenységet, és ennek hatását az adatbázis állapotára és a naplózásra. Áttekintjük azt is, hogy maga a napló hogyan keletkezik a memóriában és hogyan íródik ki a lemezre a „flush-log” (naplókiírás) művelettel. Végül megvizsgáljuk konkrétan a semmisségi naplózást, és megtanuljuk, hogyan használhatjuk a katasztrófákból való helyreállításra. Elkerülendő azt, hogy helyreállítás során a teljes naplót át kelljen vizsgálni, bemutatjuk az „ellenőrzőpont-képzés” (checkpointing) ötletét, mely lehetővé teszi, hogy a napló régi részét eldobjuk[[2]](#footnote-2). Az ellenőrzőpont-képzés módszerét a semmisségi naplózáshoz kapcsolódóan ebben a fejezetben tanulmányozzuk.

### 8.2.1. Naplóbejegyzések

Úgy kell tekintenünk, hogy a napló mint fájl, kizárólag bővítésre van megnyitva. Tranzakció végrehajtásakor a naplókezelőé a feladat, hogy minden fontos eseményt a naplóban rögzítsen. A napló blokkjai mindenkor naplóbejegyzésekkel vannak feltöltve, mindegyik bejegyzés egy-egy naplózandó eseményre vonatkozik. A naplóblokkokat elsődlegesen a memóriában hozza létre a rendszer, és a pufferkezelő az adatbázisrendszer többi blokkjaihoz hasonlóan kezeli őket. A naplóblokkokat, amint csak lehetséges, a nem illékony tárolóra írja a rendszer, erről bővebben a 8.2.2. részben szólunk.

A naplózás minden típusa a naplóbejegyzésnek számos formáját használja. E részben a következőkkel foglalkozunk:

1. <START *T*> : Ez a bejegyzés jelzi a *T* tranzakció (végrehajtásának) elkezdődését.

2. <COMMIT *T*>: A *T* tranzakció rendben befejeződött, az adatbázis elemein már semmi további módosítást nem kíván végrehajtani. A *T* által végrehajtott összes adatbázis-módosítás már megtörtént a lemezen. Minthogy azt nem tudjuk felügyelni, hogy a pufferkezelő mikor dönt a memóriablokkok lemezre másolásáról, így általában nem lehetünk biztosak abban, hogyha meglátjuk a <COMMIT *T*> naplóbejegyzést, akkor a változtatások a lemezen már megtörténtek. Ha ragaszkodunk ahhoz, hogy a módosítások már a lemezen is megtörténjenek, ezt az igényt a naplókezelőnek kell kikényszerítenie (mint például a semmisségi naplózás esetében).

3. <ABORT *T*>: A *T* tranzakció nem tudott sikeresen befejeződni. Ha a *T* tranzakció abortált, az általa tett változtatásokat nem kell a lemezre másolni. A tranzakció-kezelő feladata annak biztosítása, hogy az ilyen változtatások ne jelenjenek meg a lemezen, vagy ha volt valami hatásuk a lemezen, akkor az törlődjék. Az abortált tranzakció hatásainak helyreállításával a 10.1.1. részben foglalkozunk.

A semmisségi (undo) naplózáshoz csak egyetlen további naplóbejegyzésre van szükségünk, a *módosítási bejegyzésre* (update record), mely a <*T*,*X*,*v*> hármas. Ezen bejegyzés jelentése: a *T* tranzakció módosította az *X* adatbáziselemet, melynek módosítás előtti értéke *v* volt. A módosítási bejegyzés által leírt változtatás rendesen csak a memóriában történt meg, a lemezen nem; azaz a naplóbejegyzés a WRITE tevékenységre vonatkozik, nem pedig az OUTPUTra! (Emlékeztetünk a két művelet közötti különbségre, amit a 8.1.4. részben már láttunk.) Megjegyezzük még, hogy a semmisségi naplózás nem rögzíti az adatbáziselem új értékét, csak a módosítás előtti értéket. Amint látni fogjuk, a semmisségi naplózást alkalmazó rendszerekben a helyreállítás-kezelő feladata a tranzakció lehetséges hatásainak semmissé tétele, amelyhez elegendő csak a régi értékek tárolása.

### 8.2.2. A semmisségi naplózás szabályai

Ahhoz, hogy a rendszerhibák utáni helyreállításra a semmisségi naplózást használhassuk, a tranzakcióknak két előírást kell kielégíteniük. Ezek a szabályok arra vonatkoznak, hogy a pufferkezelőnek hogyan kell működnie, valamint előírnak bizonyos, a tranzakció szabályos befejezésekor elvégzendő tevékenységeket. Ezeket láthatjuk itt összefoglalva:

*U*1: Ha a *T* tranzakció módosítja az *X* adatbáziselemet, akkor a <*T*,*X*,*v*> típusú naplóbejegyzést *azt megelőzően* kell lemezre kiírni, mielőtt *X* új értékét a lemezre írná a rendszer.

*U*2: Ha a tranzakció hibamentesen teljesen befejeződött, akkor a COMMIT naplóbejegyzést csak *azt követően* szabad lemezre írni, hogy a tranzakció által módosított összes adatbáziselem már lemezre íródott, de ezután viszont a lehető leggyorsabban.

Összefoglalva az *U*1 és *U*2 szabályokat, az egy tranzakcióhoz tartozó lemezre írásokat a következő sorrendben kell megtenni:

a) Az adatbáziselemek módosítására vonatkozó naplóbejegyzések kiírása.

b) Maguknak a módosított adatbáziselemeknek a kiírása.

c) A COMMIT naplóbejegyzés kiírása.

Az a) és b) lépések minden módosított adatbáziselemre vonatkozóan önmagukban, külön-külön végrehajtandók (nem lehet a tranzakció több módosítására csoportosan megtenni)!

A naplóbejegyzések lemezre írásának kikényszerítésére a naplókezelőnek szüksége van a *flush-log* parancsra, mely felszólítja a pufferkezelőt az összes korábban még ki nem írt naplóblokkoknak a lemezre való kiírására, valamint azon pufferek kiírására, amelyek tartalma utolsó kiírásuk óta megváltozott. A FLUSH LOG parancsot a tevékenységek közé fogjuk iktatni. A tranzakció-kezelőnek szüksége van arra is, hogy a pufferkezelőt az adatbáziselemekre vonatkozó OUTPUT akció végrehajtására felszólíthassa. A folytatásban be fogjuk mutatni a tranzakció lépései közé illesztett OUTPUT tevékenységet is.

**8.3. példa:** A semmisségi naplózás fényében vizsgáljuk meg újra a 8.2. példában már megismert tranzakciót. A 8.3. ábra a 8.2. ábra kibővítése, bemutatván a naplóbejegyzéseket is, és a naplókiírási tevékenységet is a *T* tranzakció végrehajtása során. Megjegyezzük, hogy a fejlécben rövidítéseket kellett használnunk; M-*A* rövidítést használjuk „*A*-nak memóriapufferbe másolása”, D-*B*-t pedig „*B*-nek lemezre másolása” jelentéssel, és hasonlóan a többi rövidítésben is.

A 8.3. ábra 1) sorában a *T* tranzakció elkezdődik. Az első, ami történik, az a <START *T*> bejegyzés naplóba írása. A 2) sor *A*-nak *T* általi beolvasását jelenti. A 3) sor *t* módosítása, melynek nincs semmilyen hatása sem a lemezen tárolt adatbázisra, sem annak memóriapufferben található egyetlen részére sem. Sem a 2), sem a 3) sor nem igényel naplóbejegyzést, mert nincs hatásuk az adatbázisra.

A 4) sor *A* új értékének pufferbe írása. *A* ezen módosítására vonatkozik a <*T*,*A*,8> naplóbejegyzés, mely azt rögzíti, hogy *A* korábbi értékét, 8-at *T* megváltoztatta. Megjegyezzük, hogy az új érték, 16, nincs megemlítve a semmisségi naplózás naplójában.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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> |
| 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> |
| 18) | FLUSH LOG |  |  |  |  |  |  |
| 19) | OUTPUT(A) | 16 | 16 | 16 | 16 | 18 |  |
| 10) | OUTPUT(B) | 16 | 16 | 16 | 16 | 16 |  |
| 11) |  |  |  |  |  |  | <COMMIT *T*> |
| 12) | FLUSH LOG |  |  |  |  |  |  |

**8.3. ábra.** Tevékenységek és naplóbejegyzéseik

Az 5)-től 7)-ig sorokban a *B*-re vonatkozóan ugyanazon lépések hajtódnak végre, mint korábban *A*-ra. E ponton a *T* rendben befejeződött, tevékenységét véglegesíteni kell. A megváltozott *A* és *B* értékét lemezre kell másolni, betartva a semmisségi (undo) naplózás két szabályát, a következő lépéseknek kötött sorrendben kell megtörténnie.

Első, hogy *A* és *B* addig nem másolható lemezre, amíg a módosítást leíró naplóbejegyzések lemezre nem kerülnek. Ezt a 8) lépéssel biztosítjuk, a FLUSH LOG hatására az eddigi összes naplóbejegyzés lemezre íródik. E kiírást követően a 9) és 10) lépések *A*-t és *B*-t lemezre másolják. Ezeket a lépéseket a *T* teljes befejeződése, a véglegesítés érdekében a tranzakció-kezelő igénye szerint a pufferkezelő valósítja meg.

S ekkor lehetséges a *T* teljes és sikeres befejezése, ezt jelzendő a 11) lépésben <COMMIT *T*> bejegyzés a naplóban íródik. Végül a 12) lépésben ismét ki kell adni a FLUSH LOG utasítást azért, hogy biztosítsuk a <COMMIT *T*> naplóbejegyzés lemezre való kiírását. Ezen naplóbejegyzés lemezre való kiírása nélkül, bár olyan helyzetben vagyunk, hogy a tranzakció teljesen és hibamentesen befejeződött, ennek a napló későbbi elemzésekor nem fogjuk nyomát találni. Az ilyen szituációk olyan furcsa viselkedést eredményezhetnek, hogy hiba esetén, amint a 8.2.3. részben látni fogjuk, a felhasználó azt tapasztalja, hogy a tranzakció hibamentesen rendesen befejeződött, a lemezre kiírt módosítások mégis semmissé váltak, a tranzakció ténylegesen abortált[[3]](#footnote-3). 

### 8.2.3. Helyreállítás a semmisségi naplózás használatával

Tételezzük fel, hogy rendszerhiba fordult elő. Előfordulhat, hogy valamely adott tranzakció által végzett adatbázis-módosítások közül bizonyosak lehet, hogy már lemezre íródtak, míg más módosítások – melyeket ugyanezen tranzakció hajtott végre – nem jutottak el a lemezre. Ha így történt, a tranzakció nem atomosan hajtódott végre, ennek következtében az adatbázis inkonzisztens állapotba kerülhetett. A *helyreállítás-kezelő* (recovery manager) feladata – a napló használatával – az adatbázist konzisztens állapotba visszaállítani.

Ebben a részben csak a legegyszerűbb helyreállítás-kezelő módszerrel foglalkozunk, mely a teljes naplót látja, függetlenül annak méretétől, és a napló vizsgálatával hajtja végre az adatbázis módosításait. A 8.2.4. részben egy sokkal finomabb megközelítést mutatunk be, amikor ellenőrzőpont periodikus készítésével a rendszer korlátozza azt a távolságot, ameddig a helyreállítás-kezelőnek a korábbi történéseket (a naplóban) vizsgálnia kell.

A helyreállítás-kezelő első feladata a tranzakciók felosztása sikeresen befejezett és nem befejezett tranzakciókra. Ha található <COMMIT *T*> naplóbejegyzés, akkor az *U*2 szabálynak megfelelően a *T* tranzakció által végrehajtott összes adatbázis-módosítások már korábban lemezre íródtak. Így a *T* tranzakció önmagában, a hiba fellépésekor, nem hagyhatta az adatbázist inkonzisztens állapotban.

**Háttértevékenységek, naplózás, pufferkezelés**

A 8.3. ábrán látottaknak megfelelően a tranzakció tevékenységei és a naplóbejegyzések sorozata azt az elképzelést sugallják, mintha ezek a tevékenységek elkülönülten következnének be. Ugyanakkor az adatbázis-kezelő rendszer számos tranzakció szimultán kezelését kell hogy megoldja. Így egy *T* tranzakció négy naplóbejegyzése a naplóban más tranzakciók naplóbejegyzéseivel keveredhet. Ezen felül, ha a másik tranzakciók valamelyike is a napló lemezre írását kezdeményezi (FLUSH LOG kiadásával), akkor a *T*-re vonatkozó naplóbejegyzések esetleg már korábban lemezre kerülnek, mint ahogy azt a 8.3. ábrán látható FLUSH LOG utasítások okoznák. Abból nem származik probléma, ha az adatbázis módosítására vonatkozó naplóbejegyzések a szükségesnél korábban jelennek a naplóban. A <COMMIT *T*> naplóbejegyzést úgysem fogjuk a *T* OUTPUT utasításai végrehajtásának befejezésénél korábban kiírni, ezzel biztosítani tudjuk, hogy a módosított adatbázisértékek korábban megjelenjenek a lemezen, mint a COMMIT naplóbejegyzés.

Kényes helyzet áll elő, ha az *A* és *B*, két adatbáziselem, közös blokkban található. Akkor egyikük lemezre írása maga után vonja a másik kiírását is. Legrosszabb esetben az egyik adatbáziselem túl korai kiírásával megsértjük az *U*1 szabályt. Ez szükségessé tehet további előírásokat a tranzakcióra nézve azért, hogy a semmisségi naplózási módszer használható legyen. Például a 9.3. részben ismertetett zárolási módszert kell használnunk annak megelőzésére, nehogy két tranzakció, egyszerre, ugyanazon blokkot használja (e példában lemezblokkokat tekintünk adatbáziselemeknek). Ilyen és hasonló problémák akkor jelentkeznek, amikor az adatbáziselemek blokkok részei, ez motiválja azt a javaslatunkat, hogy a blokkokat adatbáziselemeknek tekintsük.

Amennyiben feltételezzük, hogy a naplóban találunk <START *T*> bejegyzést, de nem találunk <COMMIT *T*> bejegyzést, akkor a *T* végrehajthatott az adatbázisban olyan módosításokat, melyek még a hiba fellépése előtt lemezre íródtak, amíg más változtatások a memória pufferekben sem történtek meg, vagy a memória pufferben megtörténtek ugyan, de a lemezre már nem íródtak ki. Ilyen esetben a *T* *nem komplett tranzakció,* és hatását semmissé kell tenni, azaz a *T* által módosított adatbáziselemek értékét vissza kell állítani korábbi értékeikre. Szerencsére az *U*1 szabály betartása biztosítja, hogy ha *T*, a hiba jelentkezése előtt, az *X* értékét módosította, akkor a hiba jelentkezése előtt már a lemezen lévő naplóba kellett kiíródni egy <*T*,*X*,*v*> bejegyzésnek. S így a helyreállítás során módunkban áll a *v* értéket az *X* adatbáziselembe visszaírni. Megjegyezzük, hogy ez a szabály bizonyítottnak tekinti, hogy *X* korábbi értéke *v* volt, de ennek tényleges ellenőrzésére alkalmatlan. (A <*T*,*X*,*v*> naplóbejegyzésnek hinnünk kell, annak helyességét önmagában nem tudjuk ellenőrizni.)

Minthogy a naplóban számos, rendesen befejezett és teljesen be nem fejezett tranzakció nyomát találhatjuk, s ezek közül több tranzakció módosíthatta az *X* adatbáziselemet is, így nagyon ügyelnünk kell arra, hogy milyen sorrendben állítjuk vissza *X* korábbi tartalmát. Ezért a helyreállítás-kezelő a naplót a végéről kezdi átvizsgálni (tehát az utoljára felírt bejegyzéstől a korábban felírtak irányába). Amint halad a napló átvizsgálásával, megjegyzi mindazon *T* tranzakciókat, melyekre vonatkozóan a naplóban <COMMIT *T*> vagy <ABORT *T*> bejegyzést talált. Amint halad visszafelé, amikor <*T*,*X*,*v*> bejegyzést lát, akkor:

1. Ha ugyanerre a *T* tranzakcióra vonatkozó COMMIT bejegyzéssel már találkozott, akkor nincs teendője, *T* rendesen és teljesen befejeződött, hatásait nem kell tehát semmissé tenni.

2. Minden más esetben *T* nem teljes vagy abortált tranzakció. A helyreállítás-kezelő *X* értékét *v*-re kell, hogy cserélje.[[4]](#footnote-4)

Miután a helyreállítás-kezelő végrehajtotta a fenti változtatásokat, minden, korábban abortált, nem teljes *T* tranzakcióra vonatkozóan <ABORT *T*> naplóbejegyzést ír a naplóba, és kiváltja annak naplófájlba való kiírását is (FLUSH LOG). Ekkor az adatbázis normál használata folytatódhat, új tranzakciók végrehajtása kezdődhet.

### 8.2.4. Az ellenőrzőpont-képzés

Mint láttuk, a helyreállítás elvben a teljes napló átvizsgálását igényli. Ha a naplózásban (az eddig bemutatott) semmisségi (undo) naplózás módszerét követjük, akkor, ha egy tranzakció a COMMIT naplóbejegyzést már kiírta a naplóba, akkor az ezen tranzakcióra vonatkozó naplóbejegyzésekre a helyreállítás során nincs már szükség[[5]](#footnote-5). Gondolhatnánk arra, hogy a tranzakcióra vonatkozó, a COMMITot megelőző naplóbejegyzéseket törölhetnénk a naplóból, de ezt nem mindig tehetjük meg. Ennek oka az, hogy gyakran sok tranzakció működik egyszerre, ha a naplót egy tranzakció befejezése után csonkítanánk, esetleg elveszítenénk más, még aktív tranzakciókra vonatkozó bejegyzéseket, s így – ha szükség lenne rá –, nem tudnánk a naplót a helyreállításra használni.

E lehetséges probléma megoldására a legegyszerűbb mód, a naplóra vonatkozóan, ismétlődően *ellenőrzőpontot* képezni. Az egyszerű ellenőrzőpont képzése:

1. Új tranzakció indítási kérések kiszolgálásának leállítása.

2. A még aktív tranzakciók helyes és teljes befejezésének vagy abortálásának és a COMMIT vagy az ABORT bejegyzés naplóba írásának kivárása.

<START *T*1>

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

<START *T*2>

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

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

<*T*1,*D*,20>

<COMMIT *T*1>

<COMMIT *T*2>

<CKPT>

<START *T*3>

<*T*3,*E*,25>

<*T*3,*F*,30>

**8.4. ábra.** Napló egyszerű ellenőrzőpont-képzéssel

3. A napló lemezre kiírása (FLUSH).

4. <CKPT>[[6]](#footnote-6) naplóbejegyzés képzése és kiírása a naplóba, és ismételt FLUSH.

5. Tranzakció indítási kérések kiszolgálása.

Az ellenőrzőpont kiírását megelőzően végrehajtott tranzakciók mind befejeződtek, s az *U*2 szabálynak megfelelően módosításaik már lemezre kerültek. Ennek megfelelően – ezen tranzakciók tevékenységére nézve – egy esetleges későbbi hiba elhárításakor már nem igényel a rendszer helyreállítást. A helyreállítás során a naplót a végétől visszafelé csak a <CKPT> bejegyzésig kell elemezni azért, hogy a nem befejezett tranzakciókat azonosítsuk. Amikor a <CKPT> bejegyzést megtaláljuk, ebből tudjuk, hogy már láttuk az összes befejezetlen tranzakciót. Mivel az ellenőrzőpont-képzés alatt újabb tranzakció nem indulhatott, látnunk kellett a befejezetlen tranzakciókhoz tartozó összes naplóbejegyzést. Ezért nem szükséges a <CKPT> bejegyzésnél korábbi naplórészt elemeznünk, s – ha más okból már nincs szükségünk rá – biztonsággal törölhetjük vagy felülírhatjuk.

**8.5. példa:** Tegyük fel, hogy a napló így kezdődik:

<START *T*1>

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

<START *T*2>

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

S ekkor döntünk ellenőrzőpont létrehozásáról. Minthogy *T*1 és *T*2 aktív (nem befejezett) tranzakciók, meg kell várnunk befejeződésüket, mielőtt a <CKPT> bejegyzést a naplóba írnánk.

A napló egy lehetséges folytatását a 8.4. ábra mutatja. Tegyük fel, hogy e ponton lép fel rendszerhiba. A naplót a végétől visszafelé elemezve, *T*3-at fogjuk az egyetlen be nem fejezett tranzakciónak találni, és így *E* és *F* korábbi értékeit, 25-öt és 30-at kell csak visszaállítanunk. Amikor megtaláljuk a <CKPT> bejegyzést, tudjuk, hogy nem kell a korábbi naplóbejegyzéseket elemeznünk, és tudjuk, hogy az adatbázis állapotának helyrehozásával végeztünk. 

### 8.2.5. Ellenőrzőpont képzés a rendszer működése közben[[7]](#footnote-7)

A 8.2.4. részben bemutatott ellenőrzőpont-képzési technika problémája, hogy gyakorlatilag le kell állítani a rendszer működését (nem engedni új tranzakciók indítását) az ellenőrzőpont elkészültéig. Minthogy az aktív tranzakciók még hosszabb időt igényelhetnek a normális vagy abnormális befejeződésükig, így a felhasználó számára a rendszer leállítottnak tűnhet. Egy jóval bonyolultabb módszerrel, a *működés közbeni ellenőrzőpont-képzéssel* elérjük, hogy az ellenőrzőpont-képzés alatt új tranzakciók indítását ne kelljen szüneteltetni. E módszer lépései:

1. <START CKPT (*T*1, …, *Tk*)> naplóbejegyzés készítése és a naplóbejegyzés lemezre írása (FLUSH LOG).
 *T*1, …, *Tk* az éppen *aktív* tranzakciók nevei.

2. Meg kell várni a *T*1, …, *Tk* tranzakciók mindegyikének normális vagy abnormális befejeződését, nem tiltva közben újabb tranzakciók indítását.

3. Ha a *T*1, …, *Tk* tranzakciók mindegyike befejeződött, akkor <END CKPT> naplóbejegyzés elkészítése és a naplóbejegyzés lemezre írása (FLUSH LOG).

Az ilyen típusú napló felhasználásával a következőképpen tudunk rendszerhiba után helyreállítani: a naplót a végétől visszafelé elemezve megtaláljuk az összes nem befejezett tranzakciót, régi értékére visszaállítjuk az ezen tranzakciók által megváltoztatott adatbáziselemek tartalmát. Két eset fordulhat elő aszerint, hogy visszafelé olvasván a naplót, az <END CKPT> naplóbejegyzést vagy a <START CKPT (*T*1, …, *Tk*)> naplóbejegyzést találjuk előbb.

• Ha előbb az <END CKPT> naplóbejegyzéssel találkozunk, akkor tudjuk, hogy az összes még be nem fejezett tranzakcióra vonatkozó naplóbejegyzést a legközelebbi korábbi <START CKPT (*T*1, …, *Tk*)> naplóbejegyzésig megtaláljuk. Ennél a <START CKPT (*T*1, …, *Tk*)> naplóbejegyzésnél megállhatunk, a még korábbiakat már nem kell használnunk, azokat el is dobhatjuk.

• Amennyiben a <START CKPT (*T*1, …, *Tk*)> naplóbejegyzéssel találkozunk előbb, az azt jelenti, hogy a katasztrófa az ellenőrzőpont-képzés közben fordult elő. Ennek következtében *T*1, …, *Tk* tranzakciók nem fejeződtek be (legalábbis nem tudtuk a befejeződést regisztrálni) a hiba fellépéséig. Ekkor a be nem fejeződött tranzakciók közül a legkorábban kezdődött tranzakció indulásáig kell a naplóban visszakeresnünk, annál korábbra nem. Az ezt megelőző START CKPT bejegyzés biztosan megelőzi a keresett összes tranzakció indítását leíró bejegyzéseket.[[8]](#footnote-8) Ezenfelül, ha ugyanazon tranzakció naplóbejegyzéseire nézve láncokat is használunk, akkor nem kell a napló minden bejegyzését átnéznünk ahhoz, hogy megtaláljuk a még be nem fejezett tranzakciókra vonatkozó bejegyzéseket, elegendő csak az adott tranzakció bejegyzései láncán visszafelé haladnunk.

Általános szabályként, ha egy <END CKPT> naplóbejegyzést kiírunk lemezre, akkor a megelőző START CKPT bejegyzésnél korábbi naplóbejegyzéseket törölhetjük.

**8.6. példa:** Tegyük fel, hogy a napló, mint a 8.5. példában is, így kezdődik:

<START *T*1>

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

<START *T*2>

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

S most úgy döntünk, hogy működés közbeni ellenőrzőpontot hozunk létre. Minthogy e pillanatban *T*1 és *T*2 aktív (nem befejezett) tranzakciók, ezért a következő naplóbejegyzést kell felírnunk:

<START CKPT (*T*1, *T*2)>

Tegyük fel, hogy amíg *T*1 és *T*2 befejeződésére várunk, azalatt egy másik tranzakció, *T*3 elkezdődik. A napló egy lehetséges folytatását a 8.5. ábrán mutatjuk be.

<START *T*1>

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

<START *T*2>

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

<START CKPT (*T*1, *T*2)>

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

<START *T*3>

<*T*1,*D*,20>

<COMMIT *T*1>

<*T*3,*E*,25>

<COMMIT *T*2>

<END CKPT>

<*T*3,*F*,30>

**8.5. ábra.** Napló működés közbeni ellenőrzőpont-képzéssel

<START *T*1>

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

<START *T*2>

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

<START CKPT (*T*1, *T*2)>

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

<START *T*3>

<*T*1,*D*,20>

<COMMIT *T*1>

<*T*3,*E*,25>

**8.6. ábra.** Napló ellenőrzőpont-képzés közben történt rendszerkatasztrófa során

Tételezzük fel, hogy most lépett fel valamilyen hiba. A naplót a végétől visszafelé vizsgálva, úgy fogjuk találni, hogy *T*3 egy be nem fejezett tranzakció, s ezért hatásait semmissé kell tenni. Az utolsó naplóbejegyzés arról informál bennünket, hogy az *F* adat­báziselembe a 30 értéket kell visszaállítani. Amikor az <END CKPT> naplóbejegy­zést találjuk, tudjuk, hogy az összes be nem fejezett tranzakciók a megelőző START CKPT naplóbejegyzés után indulhattak csak el. Tovább visszafelé elemezve, megtaláljuk a <*T*3,*E*,25> bejegyzést, mely megmondja nekünk, hogy az *E* adatbáziselem értékét 25-re kell visszaállítani. Ezen bejegyzés és a START CKPT naplóbejegyzés között további elindult, de be nem fejeződött tranzakcióra vonatkozó bejegyzést, és további adatbázis-módosításra vonatkozó bejegyzést nem találunk, így az adatbázison mást már nem kell megváltoztatnunk.

Tegyük fel most, hogy az ellenőrzőpont képzése közben történt katasztrófa, s a napló vége a 8.6. ábrán bemutatott. Visszafelé elemezve a naplót, azonosítjuk a *T*3, majd a *T*2 tranzakciókat, melyek nincsenek befejezve, s helyreállító módosításokat kell tennünk. Amikor megtaláljuk a <START CKPT (*T*1, *T*2)> naplóbejegyzést, megtudjuk, hogy az egyetlen további olyan tranzakció, mely lehetséges, hogy nincs befejezve, a *T*1. Minthogy azonban a <COMMIT *T*1> bejegyzést már láttuk, ebből tudjuk, hogy *T*1 *nem* be nem fejezett tranzakció. Láttuk már továbbá a <START *T*3> bejegyzést is, s így már tudjuk, hogy csak addig kell folytatnunk a napló visszafelé elemzését, amíg *T*2 START bejegyzését meg nem találjuk. Eközben még a *B* adatbáziselem értékét is visszaállítjuk 10-re. 

1. Az undo naplózást semmisségi naplózásnak, vagy visszavonási naplózásként is fordítják. *A fordító megjegyzése.* [↑](#footnote-ref-1)
2. Ha „csak” a helyreállításra használjuk a naplót, akkor az utolsó ellenőrzőpont-képzésnél korábban keletkezett naplórészt valóban eldobhatjuk, de ha a naplót a korábban történt akciók utólagos elemzésére is fel kívánjuk használni, akkor meg kell tartanunk. Itt figyelembe kell vennünk, hogy a napló állandóan és jelentősen növekedik, de szerencsére a háttértárolók kapacitása is nő, a fajlagos tárolási költség pedig csökken. *A fordító megjegyzése.* [↑](#footnote-ref-2)
3. A tranzakció valójában nem abortált, de egy később fellépett hiba elemzésekor, mivel a rendszer nem talál <COMMIT *T*> naplóbejegyzést, úgy tekinti, hogy a tranzakció nem tudott teljesen és rendesen befejeződni, ezért a helyreállító rendszer a tranzakció hatásait semmissé teszi, végeredményben az történik, mintha *T* abortált volna. *A fordító megjegyzése.* [↑](#footnote-ref-3)
4. Ha *T* abortált, akkor az összes hatását az adatbázisban mindenképpen vissza kell állítani. [↑](#footnote-ref-4)
5. A tranzakció teljesen és helyesen befejeződött, nem kell tehát semmissé tenni hatásait, a napló bejegyzéseire ez okból már valóban nincs szükség, a tevékenységek utólagos elemzése miatt azonban még e naplóbejegyzések is fontosak lehetnek. *A fordító megjegyzése.* [↑](#footnote-ref-5)
6. CKPT – Checkpoint (ellenőrzőpont) rövidítése. *A lektor megjegyzése.* [↑](#footnote-ref-6)
7. Az eredeti műben „nonquiescent checkpointing”, azaz „nem nyugalmi ellenőrzőpont-képzés”-ként szerepel. *A fordító megjegyzése.* [↑](#footnote-ref-7)
8. Mivel működés közbeni ellenőrzőpont-képzéssel dolgozunk, előfordulhat, hogy a be nem fejeződött tranzakciók némelyike az előző ellenőrzőpont-képzés kezdete és befejezése között indult el. [↑](#footnote-ref-8)