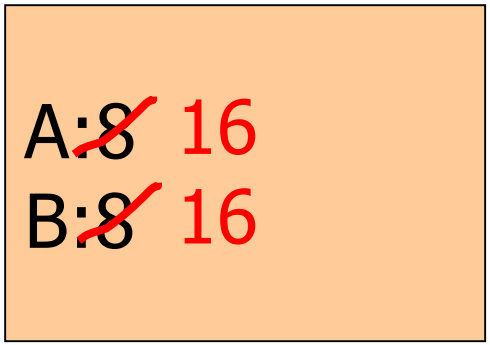
Undo log (SEMMISSÉGI NAPLÓZÁS)

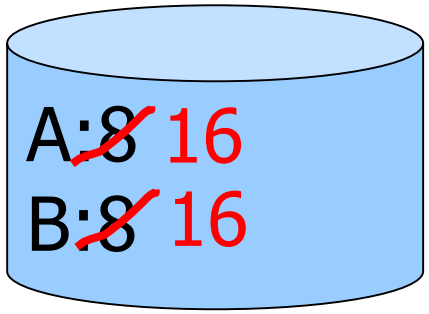
T1: Read (A,t); t ← t×2
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);
Output (B);

A=B

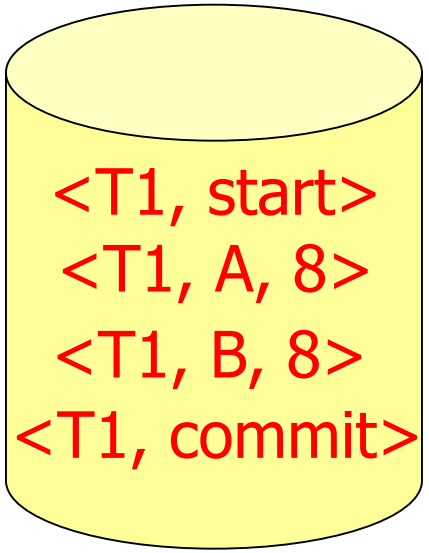
- **A régi értéket naplózzuk!**
- **Azonnali kiírás!**



MEMÓRIA



LEMEZ



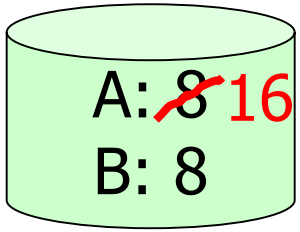
NAPLÓ

Mikor írjuk ki a naplót a lemezre?

- **A naplót először a memóriában frissítjük.**
- **Mi van, ha nem írjuk ki lemezre minden egyes változtatáskor, és elszáll a memória?**

MEMÓRIA

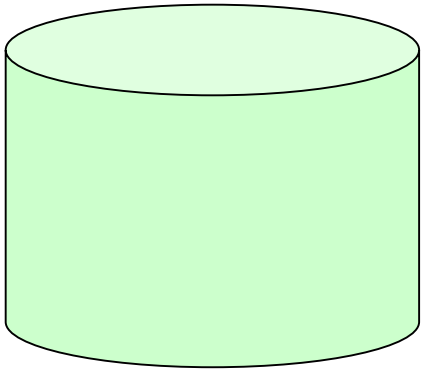
A: ~~8~~ 16
B: ~~8~~ 16
Napló:
<T_1,start>
<T_1, A, 8>
<T_1, B, 8>



Adatbázis

1. ROSSZ állapot

Napló

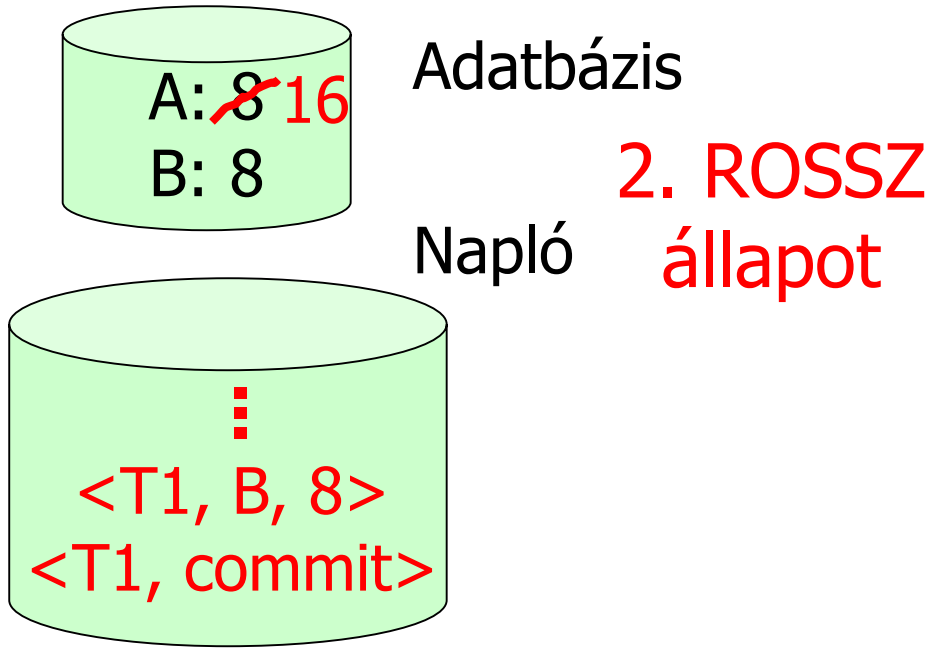


Mikor írjuk ki a naplót a lemezre?

- **A naplót először a memóriában frissítjük.**
- **Mi van, ha előbb írjuk ki lemezre, mint hogy az adatbázist frissítettük volna, és közben elszáll a memória?**

MEMÓRIA

A: ~~8~~ 16
B: ~~8~~ 16
Napló:
<T_1,start>
<T_1, A, 8>
<T_1, B, 8>
<T1, commit>



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.

Undo naplózás esetén a lemezre írás sorrendje

1. **Az adatbáziselemek módosítására vonatkozó naplóbejegyzések;**
2. **maguk a módosított adatbáziselemek;**
3. **a COMMIT naplóbejegyzés.**

- **Az első két 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.**

Undo naplózás esetén a lemezre írás sorrendje

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>
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>
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)	OUTPUT(B)	16	16	16	16	16	
11)							<T,COMMIT>
12)	FLUSH LOG						

Helyreállítás UNDO napló alapján

A helyreállítás-kezelő feladata a napló használatával az adatbázist konzisztens állapotba visszaállítani.

- **For every Ti with <Ti, start> in log:**
  - **If <Ti,commit> or <Ti,abort> in log, do nothing**
  - **Else**
    - **For all <Ti, *X*, *v*> in log:**
      - **write (*X*, *v*)**
      - **output (*X*)**
    - **Write <Ti, abort> to log**

Jó ez így?

Helyreállítás UNDO napló alapján

A helyreállítás során fontos a módosítások sorrendje!

(1) Let S = set of transactions with
    <Ti, start> in log, but no
    <Ti, commit> (or <Ti, abort>) record in log

(2) For each <Ti, X, v> in log,
    in reverse order (latest → earliest) do:
    - if Ti ∈ S then
        - write (X, v)
        - output (X)

(3) For each Ti ∈ S do
    - write <Ti, abort> to log

(4) Flush log

Ez miért lesz jó?

Helyreállítás UNDO napló alapján

Ha hiba történt → konzisztens állapot visszaállítása
→ nem befejezett tranzakciók hatásának törlése

Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.

- **Ha van (T, START) és van (T, COMMIT) → minden változás a lemezen van OK**

- **Ha van (T, START), de nincs (T, COMMIT), lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült → ezeket vissza kell állítani!**

Helyreállítás UNDO napló alapján

Ha hiba történt → konzisztens állapot visszaállítása
→ nem befejezett tranzakciók hatásának törlése

Második feladat: visszaállítás

A napló végéről visszafelé (pontosabban a hibától) haladva: megjegyezzük, hogy mely Ti-re találtunk (Ti, COMMIT) vagy (Ti, ABORT) bejegyzéseket.

Ha van egy (Ti; X; v) bejegyzés:

- **Ha láttunk már (Ti, COMMIT) bejegyzést (visszafelé haladva), akkor Ti már befejeződött, értékét kiírtuk a tárra**
  → nem csinálunk semmit
- **Minden más esetben (vagy volt (Ti, ABORT) vagy nem)**
  → X-be visszaírjuk v-t

Helyreállítás UNDO napló alapján

Ha hiba történt ➡ konzisztens állapot visszaállítása
➡ nem befejezett tranzakciók hatásának törlése

Harmadik feladat: Ha végeztünk, minden nem teljes *Ti*-re írjunk (*Ti*, ABORT) bejegyzést a napló végére, majd kiírjuk a naplót a lemezre (FLUSH LOG).

Mi van ha a helyreállítás közben hiba történik? Már bizonyos értékeket visszaállítottunk, de utána elakadunk.

➡ Kezdjük elölről a visszaállítást! Ha már valami vissza volt állítva, legfeljebb még egyszer „visszaállítjuk” ➡

nem történik semmi. (A helyreállítás idempotens!)

Helyreállítás Undo naplózással

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>
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>
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)	OUTPUT(B)	16	16	16	16	16	
11)							<T,COMMIT>
12)	FLUSH LOG						

- **Ha a hiba a 12) lépést követően jelentkezett:**
- **Tudjuk, hogy ekkor a <T, COMMIT> bejegyzést már lemezre írta a rendszer. A hiba kezelése során a T tranzakció hatásait nem kell visszaállítani, a T-re vonatkozó összes naplóbejegyzést a helyreállítás-kezelő figyelmen kívül hagyhatja.**

Helyreállítás Undo naplózással

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>
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>
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)	OUTPUT(B)	16	16	16	16	16	
11)							<T,COMMIT>
12)	FLUSH LOG						

- **Ha a hiba a 11) és 12) lépések között jelentkezett:**
- **Ha a COMMIT már lemezre íródott egy másik tranzakció miatt, akkor ez az előző eset.**
- **Ha a COMMIT nincs a lemezen, akkor T befejezetlen. B és A értékét visszaállítjuk, majd <T, ABORT>-t írunk a naplóba és a lemezre.**

Helyreállítás Undo naplózással

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>
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>
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)	OUTPUT(B)	16	16	16	16	16	
11)							<T,COMMIT>
12)	FLUSH LOG						

- **Ha a hiba a 10) és 11) lépések között lépett fel:**
- **Nincs COMMIT, tehát T befejezetlen, hatásainak semmissé tétele az előző esetnek megfelelően történik.**

Helyreállítás Undo naplózással

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>
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>
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)	OUTPUT(B)	16	16	16	16	16	
11)							<T,COMMIT>
12)	FLUSH LOG						

- **Ha a 8) és 10) lépések között következett be a hiba:**
- **Az előző esethez hasonlóan T hatásait semmissé kell tenni. Az egyetlen különbség, hogy az A és/vagy B módosítása még nincs a lemezen. Ettől függetlenül mindkét adatbáziselem korábbi értékét (8) állítjuk vissza.**

Helyreállítás Undo naplózással

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>
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>
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)	OUTPUT(B)	16	16	16	16	16	
11)							<T,COMMIT>
12)	FLUSH LOG						

- **Ha a hiba a 8) lépésnél korábban jelentkezik:**
- **Az 1. szabály miatt igaz, hogy mielőtt az A és/vagy B a lemezen módosulnának, a megfelelő módosítási naplóbejegyzésnek a lemezre írt naplóban meg kell jelennie. Ez most nem történt meg, így a módosítások sem történtek meg, tehát nincs is visszaállítási feladat.**

Az ellenőrzőpont-képzés

- A visszaállítás nagyon sokáig tarthat, mert el kell mennünk a napló elejéig.
- Meddig menjünk vissza a naplóban?
- Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
- Erre való a CHECKPOINT. Ennek képzése:

1. Megtiltjuk az új tranzakciók indítását.
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér.
3. A naplót a pufferből a háttértárra írjuk (FLUSH LOG),
4. Az adategységeket a pufferből a háttértárra írjuk.
5. A naplóba beírjuk, hogy CHECKPOINT.
6. A naplót újra a háttértárra írjuk: FLUSH LOG.
7. Újra fogadjuk a tranzakciókat.

- Ezután nyilván elég az első CHECKPOINT-ig visszamenni, hiszen előtte minden Ti már valahogy befejeződött.

Az ellenőrzőpont-képzés

1. <T1, START>
2. <T1,A,5>
3. <T2, START>
4. <T2,B,10>
5. <T2,C,15>
6. <T1,D,20>
7. <T1, COMMIT>
8. <T2, COMMIT>
9. <CKPT>
10. <T3, START>
11. <T3,E,25>
12. <T3,F,30>

RENDSZERHIBA

- Tegyük fel, hogy a 4. bejegyzés után úgy döntünk, hogy ellenőrzőpontot hozunk létre.
- A T1 és T2 aktív tranzakciók, meg kell várnunk a befejeződésüket, mielőtt a <CKPT> bejegyzést a naplóba írnánk.

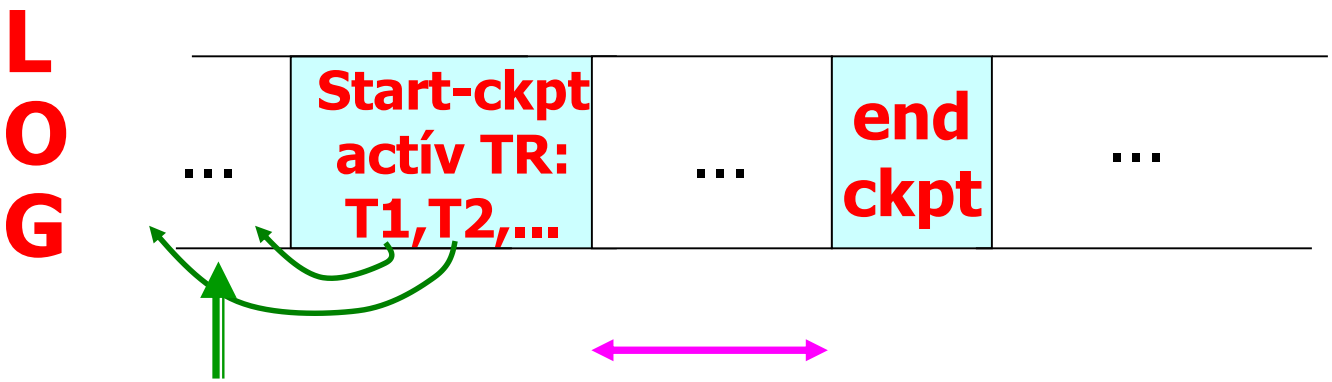
- Tegyük fel, hogy a naplórészlet végén rendszerhiba lép fel.
- HELYREÁLLÍTÁS: A naplót a végétől visszafelé elemezve T3-at fogjuk az egyetlen be nem fejezett tranzakciónak találni, így E és F korábbi értékeit kell csak visszaállítanunk.
- Amikor megtaláljuk a <CKPT> bejegyzést, tudjuk, hogy nem kell a korábbi naplóbejegyzéseket elemeznünk, végeztünk az adatbázis állapotának helyrehozásával.

Az ellenőrzőpont-képzés

- Probléma: Hosszú ideig tarthat, amíg az aktív tranzakciók befejeződnek. (Új tranzakciókat sokáig nem lehet kiszolgálni.)
- Megoldás: CHECKPOINT képzése működés közben.
- A módszer lépései:

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).

Működés közbeni ellenőrzőpont (non-quiescent checkpointing)

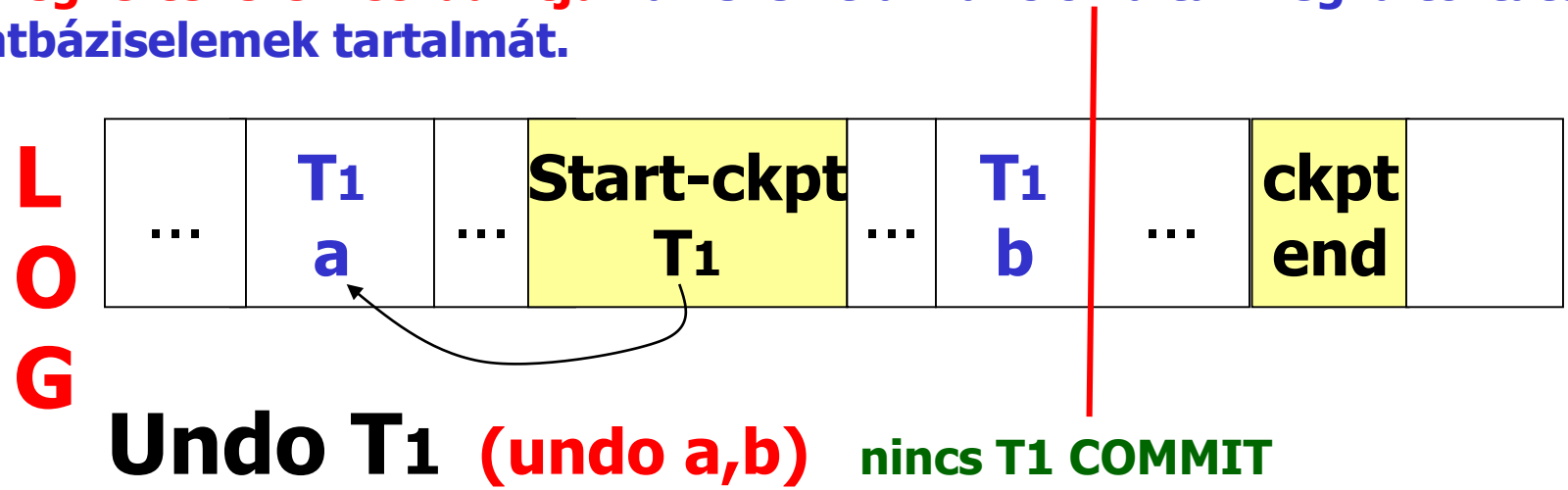


Csak az aktív tranzakciók előzményét kell megnézni!

A be nem fejezett tranzakciók által írt, "PISZKOS" adatok kerülhettek a lemezre!

Helyreállítás

- A naplót a végétől visszafelé elemezve megtaláljuk az összes be nem fejezett tranzakciót.
- A régi értékére visszaállítjuk az ezen tranzakciók által megváltoztatott adatbáziselemek tartalmát.



- Két eset fordulhat elő aszerint, hogy visszafelé olvasva a naplót az <END CKPT> vagy a <START CKPT(T1,...,Tk)> naplóbejegyzést találjuk előbb:

Két eset fordulhat elő aszerint, hogy visszafelé olvasva a naplót az <END CKPT> vagy a <START CKPT(T1,…,Tk)> naplóbejegyzést találjuk előbb:

1. **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.**

Két eset fordulhat elő aszerint, hogy visszafelé olvasva a naplót az <END CKPT> vagy a <START CKPT(T1,…,Tk)> naplóbejegyzést találjuk előbb:

2. **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.**
   - **Az ezt megelőző olyan START CKPT bejegyzés, amelyhez tartozik END CKPT, biztosan megelőzi a keresett összes tranzakció indítását leíró bejegyzéseket. (S..E.t.S.S..S)**
   - **Ha a START CKPT előtt olyan START CKPT bejegyzést találunk, amelyhez nem tartozik END CKPT, akkor ez azt jelenti, hogy korábban is ellenőrzőpont-képzés közben történt rendszerhiba. Az ilyen „ellenőrzőpont-kezdeményeket” figyelmen kívül kell hagyni.**

Helyreállítás

- **Következmény: ha egy <END CKPT> naplóbejegyzést kiírunk lemezre, akkor az azt megelőző START CKPT bejegyzésnél korábbi naplóbejegyzéseket törölhetjük.**
- **Megjegyzés: az ugyanazon tranzakcióra vonatkozó naplóbejegyzéseket összeláncoljuk, akkor nem kell a napló minden bejegyzését átnéznünk ahhoz, hogy megtaláljuk a még be nem fejezett tranzakciókra vonatkozó bejegyzéseket, elegendő csak az adott tranzakció bejegyzéseinek láncán visszafelé haladnunk.**

Helyreállítás

1. <T1, START >
2. <T1,A,5>
3. <T2, START >
4. <T2,B,10>
5. <START CKPT(T1,T2)>
6. <T2,C,15>
7. <T3, START >
8. <T1,D,20>
9. <T1, COMMIT >
10. <T3,E,25>
11. <T2, COMMIT >
12. <END CKPT>
13. <T3,F,30>

RENDSZERHIBA

- A naplót a végétől visszafelé vizsgálva úgy fogjuk találni, hogy T3 egy be nem fejezett tranzakció, 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 tranzakció a megelőző START CKPT naplóbejegyzés után indulhatott csak el.
- Megtaláljuk a <T3,E,25> bejegyzést, emiatt 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 nem találunk, így az adatbázison mást már nem kell megváltoztatnunk.

Helyreállítás

1. <T1, START >
2. <T1,A,5>
3. <T2, START >
4. <T2,B,10>
5. <START CKPT(T1,T2)>
6. <T2,C,15>
7. <T3, START >
8. <T1,D,20>
9. <T1, COMMIT >
10. <T3,E,25>
11. <T2, COMMIT >
12. <END CKPT>
13. <T3,F,30>

RENDSZERHIBA

- Visszafelé elemezve a naplót, megállapítjuk, hogy a T3, és a T2 nincs befejezve, tehát helyreállító módosításokat végzünk.
- Ezután megtaláljuk a <START CKPT(T1,T2)> naplóbejegyzést, emiatt az egyetlen be nem fejezett tranzakció csak a T2 lehet.
- A <T1, COMMIT> bejegyzést már láttuk, vagyis T1 befejezett tranzakció. Láttuk a <T3,START> bejegyzést is, így csak addig kell folytatnunk a napló elemzését, amíg a <T2, START> bejegyzését meg nem találjuk. Eközben még a B adatbáziselem értékét is visszaállítjuk 10-re.

UNDO naplózás összefoglalása

1. UNDO naplózás U1, U2 szabály
2. Helyreállítás algoritmusa
3. Ellenőrzőpont leállással
4. Ellenőrzőpont működés közben