Tranzakció, mint a helyreállítás egysége

Helyrellítás hiba esetén

- A naplóállomány sorai naplóbejegyzések (log records), melyek a tranzakció tevékenységeit tárolják, ezek segítségével rekonstruálható az amit a tranzakció tett a rendszerhiba előtt.
- Ha rendszerhiba fordul elő, a naplóbejegyzéseket felhasználva a helyreállításkezelő egy helyes adatbázist kell visszaállítson.

Több típusú naplózás is van:

- semmisségi (undo),
- helyrehozó (redo),
- semmisségi/ helyrehozó. (undo/redo)

Semmisségi (undo) naplózás

- akkor használható helyreállításra, ha nem lehet tudni, hogy a tranzakció helyesen végrehajtódott-e vagy sem, módosításai tárolódtak-e a lemezen,
- ezért semmissé kell tenni minden változtatást, amit a tranzakció esetleg végzett
- az adatbázist a tranzakció indulása előtti állapotba kell visszaállítani.
- a naplóblokkok is először a memóriában jönnek létre
- a pufferkezelő bizonyos időközönként a lemezre írja őket.

A naplóbejegyzések formái:

< START T > – a T tranzakció kezdetét jelzi.

<COMMIT T> – a T tranzakció sikeresen befejeződött, nem akar több módosítást végezni az adatbázison.

– nem tudjuk pontosan a pufferkezelő mikor másolja a memória tartalmát a lemezre, nem biztos, hogy ha <COMMIT T> bejegyzést találunk a napló-állományban a tranzakció összes művelete már a lemezre is ki van mentve.

<ABORT T> – a T tranzakció nem tudott sikeresen befejeződni,

– a módosításokat nem kell lemezre másolni.

– ha a módosítások egy része már ki volt mentve a lemezre, akkor a tranzakció indulása előtti állapotot vissza kell állítani.

< T, X, v > – a T tranzakció módosította az X adatbáziselemet, melynek módosítás előtti értéke v volt.

– a módosítás csak a pufferbeli értékre vonatkozik, tehát a WRITE műveletre és nem az OUTPUT-ra.

Semmisségi naplózás esetén a következő szabályokat kell betartanunk:
*U*1: A < *T*, *X*, *v* > típusú naplóbejegyzést kell először a naplóállományba kiírni és utána lehet az *X* adatbáziselem új értékét is lemezre írni.
*U*2: Ha a tranzakció sikeresen ért véget, először a tranzakció által módosított összes adatbáziselemet a lemezre kell írni és utána a COMMIT naplóbejegyzést is minél hamarabb a naplóállományba kell menteni.

Egy tranzakció végrehajtása során a következő sorrendet kell betartanunk:
A START `naplóbejegyzés kiírása` a naplóállományba.
Ismételd minden módosított adatbáziselemre:
A naplóállományba azon `naplóbejegyzés kiírása`, amely az adatbáziselem módosítására vonatkozik.
`Adatbáziselem` módosított értékének a `lemezre írása`
A COMMIT `naplóbejegyzés kiírása` a naplóállományba

- A naplókezelő egy FLUSH LOG parancsot használ, melynek segítségével kikényszeríti, hogy a pufferkezelő a lemezre írja a még ki nem mentett naplóblokkokat.
- A tranzakciókezelő az OUTPUT parancs segítségével szólítja fel a pufferkezelőt, hogy az adatbáziselem módosított értékét a lemezre írja.

példa:

- eladunk a raktárból egy bizonyos m mennyiséget egy adott áruból egy adott vevőnek.
- a vevővel szerződése van a cégnek. Követjük, hogy mennyi az elszállított mennyiség, tehát az eladott mennyiséget hozzá kell adjuk az elszállított mennyiséghez.

- legyen X azon adatbáziselem, mely egy adott áru raktáron levő mennyiségét tárolja,
- Y ugyanazon áru egy adott vevőnek egy szerződésen belül elszállított mennyisége.
- Egy tranzakció segítségével oldjuk meg a feladatot, mert nem szeretnénk, hogy a raktáron levő mennyiségből levonjuk, de a szállított mennyiséghez ne adjuk hozzá, mert ebben az esetben az adatbázis inkonzisztens állapotba jut és az áru el nem szállítottnak jelenik meg. A megoldás röviden:

```
BEGIN TRANSACTION
X:= X - m;
Y:= Y + m;
COMMIT TRANSACTION
```

- legyen $m = 10$,
- jelöljük az *X* adatbázis elem memóriabeli értékét *Mem_X*, illetve háttértárólón levő értékét *D_X*-el
- jelöljük az *Y* adatbázis elem memóriabeli értékét *Mem_Y*, illetve háttértárólón levő értékét *D_Y*-el
- a naplóbejegyzéseket a *Napló* oszlopban.

	Tevékenység	***v***	***Mem–X***	***Mem–Y***	***D–X***	***D–Y***	***Napló***
1)							< START *T* >
2)	READ(*X*, *v*)	50	50		50	20	
3)	*v*: = *v* – 10	40	50		50	20	
4)	WRITE(*X*,*v*)	40	40		50	20	< *T*, *X*, 50 >
5)	READ(*Y*,*v*)	20	40	20	50	20	
6)	*v*:= *v* + 10	30	40	20	50	20	
7)	WRITE (*Y*, *v*)	30	40	30	50	20	< *T*, *Y*, 20 >
8)	FLUSH LOG						
9)	OUTPUT (*X*)	30	40	30	40	20	
10)	OUTPUT (*Y*)	30	40	30	40	30	
11)							< COMMIT *T*>
12)	FLUSH LOG						

Helyreállítás semmisségi naplózással

A helyreállítás-kezelő első feladata a tranzakciók elemzése, melyek azon tranzakciók, melyek sikeresen befejeződtek és melyek nem.

1. Ha van < COMMIT T_s> naplóbejegyzés, akkor a semmisségi naplózás második szabálya alapján a T_s tranzakció által végzett módosítások már lemezre íródtak.

2. Ha a naplóállományban találunk < START T_h > bejegyzést, de nem találunk <COMMIT T_h> bejegyzést, azt jelenti,

- T_h nem komplett tranzakció
- hatását semmissé kell tenni
- a módosított összes adatbáziselemet vissza kell állítani a T_h indulása előtti értékre. (U1)

- A naplóállományban sok tranzakció tevékenysége is naplózva van, ugyanazt az adatbázis-elemet több tranzakció is módosíthatja.
- A helyreállítás-kezelő a naplóállományt a végéről kezdi átvizsgálni.
  - Ha < T_s, X, v> bejegyzést talál és a T_s-re már találkozott (hátulról jövet) <COMMIT T_s> bejegyzéssel, azt jelenti, hogy T_s sikeresen végetért, a módosítások ki is lettek mentve a lemezre. T_s tranzakciót nem kell semmissé tenni.
  - Ha a helyreállítás-kezelő <T_h, X, v> bejegyzést talál és a T_h-ra nem találkozott <COMMIT T_h> bejegyzéssel, a T_h hatását semmissé kell tegye, vagyis X adatbáziselem értékét v-re kell állítsa.
  - Ha a tranzakció abortált, akkor is a v értéket kell visszaállítani.
  - a semmissé tett tranzakciókra vonatkozóan <ABORT> bejegyzést helyez el a naplóállományba, majd FLUSH LOG-ot.

A fenti példában minden attól függ, hogy a COMMIT bekerült-e vagy sem a naplóállományba.

- ha igen, akkor a tranzakció sikeresnek tekinthető, nem kell semmissé tenni.
- ha a COMMIT nincs kiírva a naplóállományba:
  - akkor visszafele haladva találkozik a <*T*, *Y*, 20> bejegyzéssel és az *Y* adatbáziselem értékét 20-ra állítja vissza,
  - hasonlóan a <*T*, *X*, 50> bejegyzés esetén az *X* értékét 50-re állítja.
  - mikor befejezte a munkát, <ABORT *T*> bejegyzést ír a naplóállományba.

Ha az első FLUSH LOG előtt áll be a hiba, akkor nincs mit helyreállítani, mivel még nem történt lemezre írás a semmisségi naplózás első szabálya alapján.

Ellenőrzőpont-képzés

Azért, hogy a helyreállítás-kezelő ne kelljen az egész naplóállományt átvizsgálja, ún. ellenőrzőpontokat helyeznek el a naplóállományba.

1. <START CKPT (T_1, T_2, ..., T_m) > naplóbejegyzés lemezre írása (FLUSH), ahol T_1, T_2, ..., T_m az éppen aktív tranzakciók.
2. T_1, T_2, ..., T_m tranzakciók sikeres vagy sikertelen befejezésétmegvárni,
   - közben újabb tranzakciók indulhatnak.
   - sikeresek módosításainak a lemezre mentése megtörtént, COMMIT a naplóban
   - amelyek abortáltak, azok esetében az <ABORT> bejegyzés került a naplóállományba, a módosításokat nem kellett lemezre írni.
3. <END CKPT> naplóbejegyzés lemezre írása (FLUSH).

Helyreállítás ellenőrzőponttal kiegészített semmisségi naplózás segítségével

A naplót a rendszer visszafele olvassa.

1. Ha először egy <END CKPT> bejegyzést talál
   - azt jelenti, hogy a legközelebbi <START CKPT (T_1, T_2, ..., T_m) > -ig az összes be nem fejezett tranzakciót megtaláljuk.
   - ennél a <START CKPT (T_1, T_2, ..., T_m) >-nél megállhatunk, mivel az összes aktív tranzakció a listában.
   - az <END CKPT> azt mutatja, hogy a T_1, T_2, ..., T_m tranzakciók esetén a mentés megtörtént.
   - közben indulhattak más tranzakciók, azok közül a be nem fejezett tranzakciókat semmissé kell tennünk, helyreállítsuk a naplóállományban elmentett régi értékre (<T,X, v> típusú bejegyzésekből).

2. Ha először <START CKPT (T_1, T_2, ..., T_m) > naplóbejegyzést talál

– azt jelenti, hogy a hiba az ellenőrzőpont képzése közben történt, tehát a T_1, T_2, ..., T_m tranzakciók nem fejeződtek be.
– meg kell keresnünk azt a tranzakciót, mely legkorábban indult (<START> bejegyzés).
– nem elég az előző <END CKPT>-ig visszamenni, mert indulhattak a tranzakciók az előző ellenőrzőpont képzése közben is.
– a T_1, T_2, ..., T_m tranzakciókat semmisségi naplózás segítségével helyre kell állítanunk.

példa: Legyenek a következő naplóbejegyzések:

<START T_1>
<T_1, X, 10>
<START T_2>
< T_2, Y, 20>
<START CKPT (T_1, T_2)>
< T_2, Z, 30>
<START T_3>
< T_1,P, 40>
<COMMIT T_1>
< T_3, Q, 50>
<COMMIT T_2>
<END CKPT>
< T_3, R, 60>

a) Ebben a pontban hiba lép fel.

- Hátulról elemezve a naplóállományt, először <END CKPT> bejegyzést találunk, T_1, T_2 tranzakciók sikeres véget értek és a módosításaik lemezre mentése is megtörtént.
- Viszont a T_3 tranzakció nem befejezett, semmissé kell tenni hatásait:

$$R \leftarrow 60$$

$$Q \leftarrow 50$$

b) ha az ellenőrzőpont képzése közben lép fel hiba, a <COMMIT T_1> után.

- először a <START CKPT (T_1, T_2)> bejegyzést találjuk, tehát a T_1, T_2 tranzakciók valószínű nem befejezett tranzakció.
- a T_1 esetén a COMMIT bejegyzést megtaláljuk,
- a T_2 által végzett módosításokat kell semmissé tegyük,
- T_3 nem befejezett, T_3 módosításait is semmissé kell tennünk, de COMMIT T_1 előtt nem volt módosítása

$$Z \leftarrow 30$$

$$Y \leftarrow 20$$

- T_3 nem befejezett, T_3 módosításait is semmissé kell tennünk, de COMMIT T_1 előtt nem volt módosítása

Helyrehozó (redo) naplózás

A semmisségi naplózás problémája:

- csak akkor tudjuk a tranzakciót befejezni, ha az adatbázison végzett összes módosítás már lemezre íródott.

Takarékoskodhatnánk a lemezre írással, ha az adatbázis módosítások csak a memóriában történnének.

- ha több felhasználó is módosítja ugyanazt az adatbáziselemet, így csak az utolsó értéket kellene kiírnunk.

A helyrehozó naplózás:

- elkerüli az adatbáziselem azonnali kiírását a lemezre,
- a naplóállomány minden módosítást rögzít.

Undo versus Redo

Semmisségi (Undo) naplózás:

- be nem fejezett tranzakciók hatásait semmissé teszi,
- befejezett tranzakciókkal nem tesz semmit.

Helyrehozó (Redo) naplózás:

- a be nem fejezett tranzakciókat figyelmen kívül hagyja
- megismétli a befejezett tranzakciók által végzett változtatásokat.

A helyrehozó naplózás esetében a <*T*, *X*, *v*> alakú naplóbejegyzések esetén az *X* adatbáziselem a *T* tranzakció által módosított (új) értékét tartalmazza. A helyrehozó naplózás szabálya:

R1: A <*T*, *X*, *v*> és a <COMMIT *T*> naplóbejegyzéseket kell először a naplóállományba kiírni és utána lehet az *X* adatbáziselem új értékét is lemezre írni.

A START `naplóbejegyzés kiírása` a naplóállományba.

Ismételd minden módosított adatbáziselemre:

A naplóállományba azon `naplóbejegyzés kiírása`, melyek az adatbáziselem módosítására vonatkozik.

A COMMIT `naplóbejegyzés kiírása` a naplóállományba.

Ismételd minden módosított adatbáziselemre:

`Adatbáziselem` módosított értékének a `lemezre írása`.

	Tevékenység	***v***	***Mem-*** **X**	***Mem-*** **Y**	***D-*** **X**	***D -*** **Y**	***Napló***
1)							<START *T*>
2)	READ(*X*, *v*)	50	50		50	20	
3)	*v*: = *v* – 10	40	50		50	20	
4)	WRITE(*X*,*v*)	40	40		50	20	<*T*, *X*, 50>
5)	READ(*Y*,*v*)	20	40	20	50	20	
6)	*v*:= *v* + 10	30	40	20	50	20	
7)	WRITE (*Y*, *v*)	30	40	30	50	20	<*T*, *Y*, 20>
8)							<COMMIT *T*>
9)	FLUSH LOG						
10)	OUTPUT (*X*)	30	40	30	40	20	
11)	OUTPUT (*Y*)	30	40	30	40	30	

Helyreállítás helyrehozó naplózás esetén

Az R1 szabály fontos következménye:

- ha a naplóban nincs <COMMIT *T*> bejegyzés, akkor a módosítások az adatbázisban nem történtek meg, tehát nem kell őket visszaállítani.
- tehát a be nem fejezett tranzakciókkal nincs gondja a helyreállítás-kezelőnek.
- probléma a befejezett tranzakciókkal van, mivel nem tudjuk, hogy lemezre íródtak-e vagy sem, mivel először a naplóállományba a COMMIT kerül.
- a helyrehozó naplózás a módosított (új) értéket a naplóbejegyzésekben tárolja, így a helyreállítás-kezelő ezeket az értékeket lemezre írja, ha esetleg már ki voltak írva, akkor ismét kíirja.

Hiba esetén a következőket kell tennünk:

1. Megkeresni a befejezett tranzakciókat.
2. A naplóállományt az elejétől kezdve végigjárni. (Mivel több befejezett tranzakció is adhatott új értéket ugyanazon *X* adatbáziselemnek, a végső érték kell az érvényes legyen.)
3. Minden <*T*, *X*, *v*> típusú bejegyzés esetén
   a) ha *T* nem befejezett, nincs semmi dolgunk
   b) ha *T* befejezett tranzakció, akkor a *v* értéket az *X* adatbáziselembe kiírni.
4. Minden *T* be nem fejezett tranzakcióra vonatkozóan <ABORT *T*> bejegyzést írni a naplóállmányba és a naplót lemezre írni.

példa: A példa esetén a következő esetek állhatnak fenn:

i) A hiba a 9-es lépés után következik, a <COMMIT *T*> már lemezre íródott, a helyreállítás *T*-t befejezettnek tekinti és minden módosítást (esetleg ismét) kiír a lemezre ($X \leftarrow 40$, $Y \leftarrow 30$).

ii) A hiba a 8-as és 9-es lépések között jelentkezik, attól függően, hogy a <COMMIT *T*> bejegyzés ki lett-e mentve a naplóállományba, előbbi i) eset, különben iii) eset.

iii) A hiba a 8-as lépés előtt jelenik meg, *T* be nem fejezett tranzakciónak számít, az *X*, illetve *Y* értéke nem változott meg, <ABORT *T*> bejegyzés kerül a naplóállományba.

Helyrehozó naplózás ellenőrzőpont-képzés segítségével

- a befejezett tranzakciók módosításainak a lemezre írása a COMMIT után történik;
- az ellenőrzőpontok kezdete és vége között lemezre kell írnunk az összes olyan adatbáziselemet, melyet befejezett tranzakciók módosítottak;
- a pufferkezelőnek nyilván kell tartania, az ún. piszkos puffereket, melyek a már végrehajtott, de lemezre még ki nem írt módosításokat tárol;
- az ellenőrzőpont végét beállíthatjuk, anélkül, hogy megvárjuk az aktív tranzakciók (normális vagy abnormális) befejezését, mivel a lemezre írás még akkor sem történik meg.

Ellenőrzőpont képzése a helyrehozó naplózás esetén:

1. <START CKPT (T_1, T_2, ..., T_m) > naplóbejegyzés lemezre írása, ahol T_1, T_2, ..., T_m az összes éppen aktív (még nem befejezett) tranzakció.

2. Azon adatbáziselemek lemezre írása, melyeket olyan tranzakciók írtak a pufferekbe, melyek a START CKPT naplóba írásakor a COMMIT-jukhoz már eljutottak, de a puffereik még nem voltak kimentve.

3. <END CKPT> naplóbejegyzés lemezre írása.

<START T_1>
<T_1, X, 10>
<START T_2>
<COMMIT T_1>
< T_2, Y, 20>
<START T_3>
<START CKPT (T_2, T_3)>
< T_2, Z, 30>
< T_3, P, 40>
<END CKPT>
<COMMIT T_2>
< T_3, R, 60>
<COMMIT T_3>

- ellenőrzőpont-képzés pillanatában a T_1 tranzakció elért a COMMIT-hoz
- <END CKPT> előtt $X \leftarrow 10$.(T_1)

Helyreállítás ellenőrzőponttal kiegészített helyrehozó naplózás segítségével

Hátulról keresünk a naplóállományban az utolsó ellenőrzőpontig.

1. a hiba előtt <END CKPT> (<START CKPT (T_1, T_2, ..., T_m)> vége)

– azok a tranzakciók, melyek elérték a COMMIT pontjukat a <START CKPT (T_1, T_2, ..., T_m)> naplóbejegyzés előtt nincs többet gondunk, mert az END CKPT előtt ezen tranzakciók által végzett módosítások lemezre íródtak.

– helyreállítani maradnak a T_1, T_2, ..., T_m tranzakciók és azok, melyek indultak az ellenőrzőpont-képzés közben, legyenek ezek T_{m+1}, T_{m+2}, ..., T_p.

– meg kell keresnünk a T_1, T_2, ..., T_p tranzakciók közül, melyik indult legkorábban, legyen ez T_e

– a <START T_e> ponttól indulva a naplóban előre haladunk és használjuk a helyrehozó naplózás helyreállítási technikáját a T_1, T_2, ..., T_p tranzakciókra.
  - a be nem fejezett tranzakciók nem érdekesek, mivel nem írtak még semmit lemezre és nem érték el a COMMIT pontjukat.
  - tranzakciók, melyek már elérték a COMMIT pontjukat a naplóban megtaláljuk az általuk módosított új értékeket és ezeket felhasználva az adatbáziselemek új értékét a lemezre is kiírjuk.

2. a hiba előtt <START CKPT (T_1, T_2, ..., T_m)> típusú bejegyzést találunk,

– nem biztos, hogy ezek előtt COMMIT pontig elért tranzakciók által végzett módosítások ki lettek mentve a pufferből. (az <END CKPT>-ig kellett volna megtörténjen, de közben hiba lépett fel).
– visszafele kell keresnünk egy előbbi <START CKPT (U_1, U_2, ..., U_m)> bejegyzésig, és a helyrehozó naplózás helyreállítási technikáját használva helyre kell állítanunk az összes tranzakciót, mely ezen pont után a COMMIT pontját elérte.

példa: 1) Ha a legutolsó naplóbejegyzés után hiba lép fel, visszafele haladva először <END CKPT> ellenőrzőpont bejegyzést találunk.

- a megfelelő START a T_2, T_3 tranzakciókra vonatkozik,
- a T_1 tranzakció által végzett módosítások az ellenőrzőpont végére ki lettek mentve,
- a T_2, T_3 -al kell foglalkoznunk,
- mindkettőre van COMMIT, mind a kettő módosításait helyre kell állítanunk.
- megkeressük a <START T_2>-t, onnan indulva a következő adatbáziselemek értékét kell lemezre írnunk:

$$Y \leftarrow 20;\ Z \leftarrow 30;\ P \leftarrow 40;\ R \leftarrow 60.$$

2) ha a hiba a <COMMIT T_2> után lép fel
   - először END CKPT pontot kapunk,
   - T_3 nem ért el a COMMIT-ig,
   - T_2 által végzett módosításokat kell lemezre írnunk ($Y \leftarrow 20$; $Z \leftarrow 30$),
   - T_3-ra <ABORT T_3> bejegyzés a naplóba.

3) ha a hiba az <END CKPT> előtt jelent meg,
   - egy <START CKPT>-al találkozunk először
   - visszafele kell haladjunk a naplóállományban egy előző <START CKPT> pontig, ha nincs, akkor egészen a napló elejéig kell visszalépjünk.
   - a T_1 az egyedüli tranzakció, mely elérte a COMMIT pontját, módosításait kell lemezre írnunk ($X \leftarrow 10$),
   - <ABORT T_2>, <ABORT T_3> a naplóállományba.

A semmisségi/helyrehozó (undo/redo) naplózás

A semmisségi, illetve helyrehozó naplózási módszereknek bizonyos hátrányai vannak:

– a semmisségi naplózás esetén nagy a lemezműveletek száma;
– a helyrehozó naplózás esetén nagy a pufferigény, mivel a módosított adatbáziselemeket több ideig is a pufferben tartja;
– ellenőrzőpontképzésnél a két módszer ellentétes igényeket támaszt.
  - a puffer egy *X* és egy *Y* adatbáziselemet tartalmaz,
  - az *X* adatbáziselemet egy olyan tranzakció módosította, mely sikeres véget ért és a semmisségi naplózást használva feltétlenül lemezre kell menteni,
  - az *Y* adatbáziselemet egy olyan tranzakció módosította, mely esetében helyrehozó naplózást használ a rendszer és a COMMIT bejegyzés még nem került a lemezre, akkor az *Y* értékét nem írhatjuk lemezre.

A semmisségi/helyrehozó naplózás esetén a módosítását leíró naplóbejegyzés alakja: <T, X, r, u> r - régi, u - új

Szabályok:

UR1: Mielőtt az X adatbáziselem értékét a lemezen módosítanánk, a <T, X, r, u> bejegyzést a naplóállományba kell írnunk.

UR2: A <COMMIT T> naplóbejegyzést, amint megjelenik a naplóban, nyomban lemezre kell írni.

- A COMMIT bejegyzés megelőzheti vagy követheti az adatbáziselemek lemezre írását.
- UR2 azért szükséges, mivel a hiba közvetlen a COMMIT bejegyzés lemezre kerülése előtt felléphet és a felhasználó már megkapta a COMMIT-ról szóló beszámolót, valószínű, hogy az adatbáziselemek módosítása is megtörtént a lemezen, de mivel a COMMIT bejegyzés nem lett kimentve, helyreállítás esetén a tranzakciót semmissé teszi a rendszer.

példa: A <COMMIT *T*> bejegyzés kerülhetett volna a 9-es lépés elé, vagy a 12-es lépés után is.

	Tevékenység	***v***	***Mem–X***	***Mem–Y***	***D–X***	***D–Y***	***Napló***
1)							<START *T*>
2)	READ(*X*, *v*)	50	50		50	20	
3)	*v*: = *v* – 10	40	50		50	20	
4)	WRITE(*X*,*v*)	40	40		50	20	<*T*, *X*, 50, 40>
5)	READ(*Y*,*v*)	20	40	20	50	20	
6)	*v*:= *v* + 10	30	40	20	50	20	
7)	WRITE (*Y*, *v*)	30	40	30	50	20	<*T*, *Y*, 20, 30>
8)	FLUSH LOG						
9)	OUTPUT (*X*)	30	40	30	40	20	
10)							<COMMIT *T*>
11)	FLUSH LOG						
12)	OUTPUT (*Y*)	30	40	30	40	30	

Helyreállítás semmisségi/helyrehozó naplózás esetén

Helyreállításkor a semmisségi/helyrehozó naplózás esetén a következő két alapelvet kell betartani:

– minden befejezett tranzakciót állítsunk helyre a legkorábbitól kezdve (helyrehozó naplózás segítségével)
– minden be nem fejezett tranzakciót tegyünk semmissé a legutolsótól kezdve (semmisségi naplózás segítségével)

A COMMIT bejegyzés és az adatbáziselemek lemezre mentésének a sorrendje nincs lerögzítve, előfordulhat:

- a befejezett tranzakciók által végzett módosítások még nem lettek lemezre mentve,
- a be nem fejezett tranzakciók által végzett módosítások már lemezre lettek írva.

példa: a) hiba a <COMMIT *T*> naplóbejegyzés lemezre írása után:

- a *T* tranzakciót befejezettnek tekintjük és helyreállítjuk abban a sorrendben, ahogy az események megjelentek. (*X*←40, *Y*←30)

b)hiba a <COMMIT *T*> naplóbejegyzés lemezre írása előtt

- a *T* tranzakció befejezetlen tranzakciónak számít és semmissé kell tenni. *X* ← 50, *Y* ←20-ra

Bizonyos esetekben már nem lenne szükséges a lemezre írás, de a biztonság kedvéért a visszaállítást mindig végre kell hajtani.

Semmisségi/helyrehozó naplózás ellenőrzőpont-képzéssel

A semmisségi/helyrehozó naplózás esetén az ellenőrzőpont-képzés a következőképpen történik:

1. Képezzük a <START CKPT (T_1, T_2, ..., T_m)> naplóbejegyzést az összes aktív tranzakcióval és írjuk ki a lemezre.
2. Írjuk lemezre az összes piszkos puffert, azokat, melyek módosított adatbáziselemet tartalmaznak.
   - Ha a helyrehozó naplózás esetén csak a már befejezett tranzakciók által módosított puffereket mentettük lemezre, itt az összest lemezre írjuk.
3. Az <END CKPT> naplóbejegyzést a naplóba írjuk, majd lemezre mentjük.

a be nem fejezett tranzakciók által végzett módosításokat is lemezre írja.
Egy előírást viszont be kell tartani, mely a konkurencia esetén is nagyon fontos, hogy a tranzakciók közötti inkonzisztens kölcsönhatást megelőzzük. A

tranzakció semmilyen értéket nem írhat, amíg biztosak nem vagyunk abban, hogy nem abortál.

példa:

< START T_1 >
< T_1, X, 10, 15 >
< START T_2 >
< T_2, Y, 20, 25 >
< T_1, Z, 30, 35 >
< START T_3 >
< COMMIT T_1 >
< START CKPT (T_2, T_3) >
< T_3, P, 40, 45 >
< T_2, Q, 50, 55 >
< END CKPT >
< COMMIT T_2 >

< START T_4 >
< T_3, R, 60, 65 >
< T_4, U, 70, 75 >
< START T_5 >
< COMMIT T_4 >
< T_5, V, 80, 85 >

a) a hiba ebben a pontban lép fel.

- visszafele haladva a naplóállományban azonosítjuk
  - a befejezett tranzakciókat: $\{T_4, T_2\}$,
  - a be nem fejezetteket: $\{T_3, T_5\}$.
- A T_1 tranzakció a COMMIT pontját elérte az ellenőrzőpont előtt, az ellenőrzőpont-képzés közben a T_1 által végzett módosítások lemezre íródtak, tehát a T_1 tranzakcióval nincs semmi dolgunk.
- a COMMIT T_2 az ellenőrzőpont után jelenik meg, nem biztos, hogy a hiba fellépésekor lemezre íródtak a módosítások, ezért megkeressük a

<START T_2>-t, ami a < START T_4> előtt található és a T_2 és T_4 által végzett összes módosítást lemezre írjuk ($Y \leftarrow 25$; Q $\leftarrow$ 55; $U \leftarrow$75).

- A $\{T_3, T_5\}$ nem érték el a COMMIT pontjukat, ezért semmissé kell tennünk az általuk végzett módosításokat, ami előfordulhat, hogy lemezre íródtak ($P \leftarrow$ 40, $R \leftarrow$60, $V \leftarrow$ 80).

b) a hiba a COMMIT T_2 után lép fel

- a T_4 és T_5 még el sem indult,
- az ellenőrzőpont-képzés sikeres volt, ez azt jelenti, hogy a T_1 módosításai lemezre íródtak,
- a T_2 befejezett, helyrehozó naplózással kell helyreállítanunk
- a T_3 nem befejezett tranzakció, semmissé kell tennünk.

c) az ellenőrzőpont-képzés közben vagy még hamarabb (ha az <END CKPT> még nem került a naplóállományba), lép fel a hiba, egy előző ellenőrzőpontot kell keresnünk vagy a naplóállomány elejére kell mennünk
   - azon tranzakciókat, melyek elérték a COMMIT pontot a hiba fellépése előtt (a mi esetünkben esetleg a T_1 tranzakció), helyrehozó naplózással kell helyreállítanunk
   - a be nem fejezetteket, a $\{T_2, T_3\}$-at pedig semmissé kell tennünk.

- A semmisségi/helyrehozó naplózás alkalmazásakor a helyreállítás során nem részleteztük, hogy először a semmisségi (undo) vagy helyrehozó (redo) lépéseket tesszük meg.
- Előfordulhat, hogy ugyanazt az *X* adatbáziselemet két tranzakció is módosítja, egyik sikeres, másik sikertelen véget ér, a helyreállítást akármilyen sorrendben végeznénk a várt eredményt nem érnénk el.
- Az ABKR-ek a módosítások naplózása mellett a konkurencia problémáját is meg kell oldják.