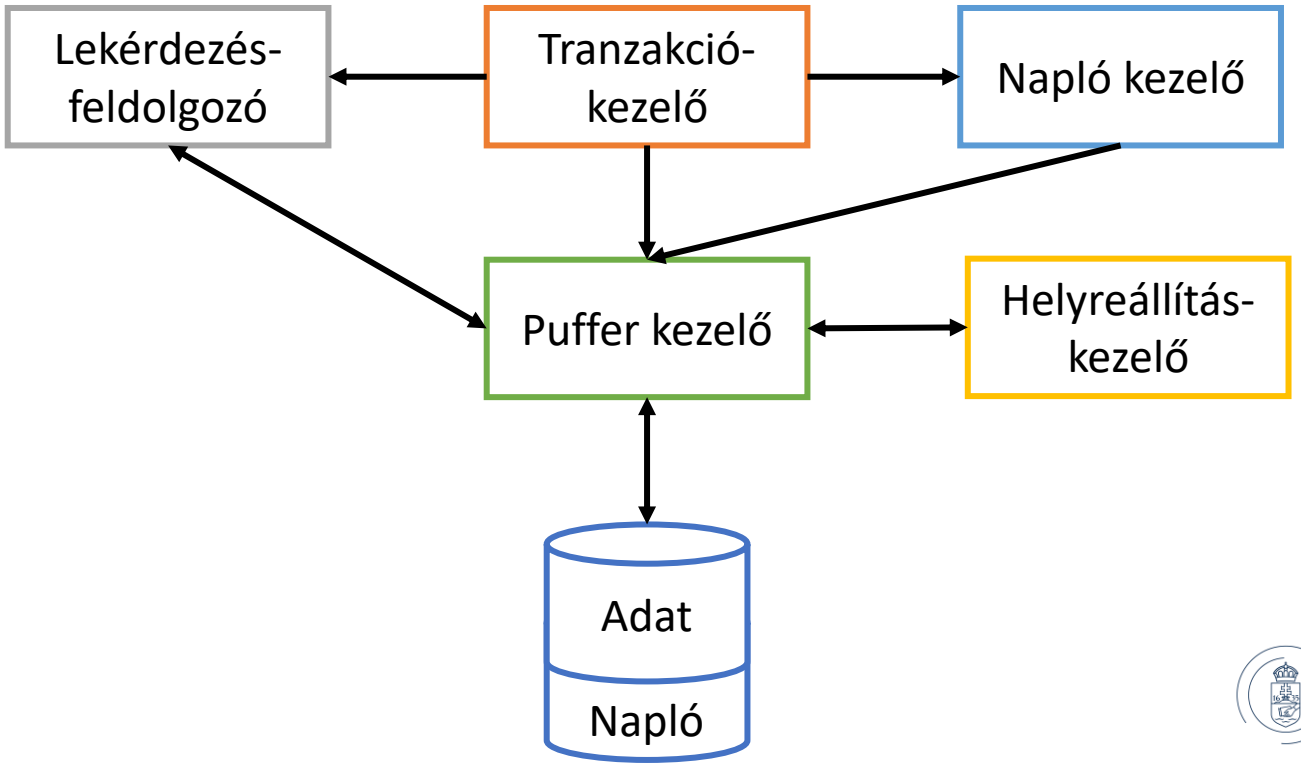
ELTE | IK
INFORMATIKAI KAR

Adatbázisok 2

Naplózás

Tranzakciókezelő

- A tranzakciók korrekt végrehajtásának biztosítása a tranzakciókezelő feladata.
- Számos feladata van: jelzéseket ad át a naplókezelőnek, biztosítja a párhuzamos végrehajtást (ütemezés) stb.



A naplókezelő

- Feladatai:
  - Karbantartja a naplót.
  - Együtt kell működnie a pufferkezelővel, hiszen az információ elsődlegesen a memóriába jelenik meg és bizonyos időközönként ezek tartalmát a lemezre kell másolni.
- A napló (mivel adat) a lemezen helyet foglal el.
- Ha baj van, akkor a helyreállítás-kezelő aktivizálódik.
  - Megvizsgálja a naplót, és ha szükséges, akkor a naplót használva helyreállítja az adatokat.

A hibák elleni védekezés

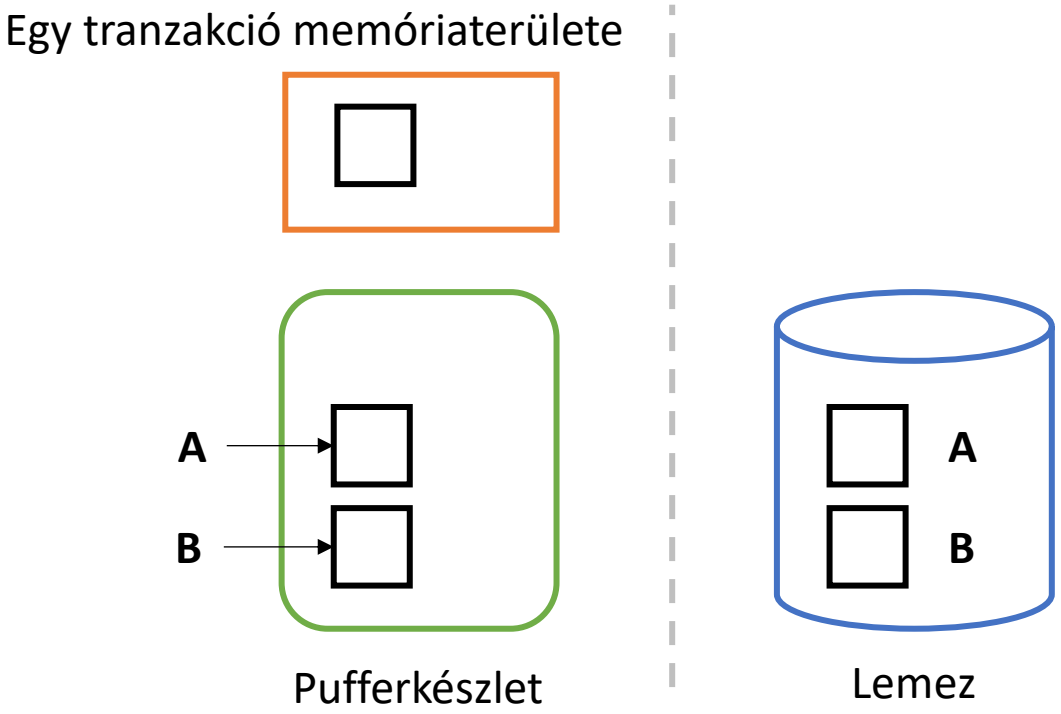
- Feltesszük, hogy a háttértár nem sérül, azaz csak a memória, illetve a pufferkészlet egy része.
- Az ilyen sérülések elleni védekezés két részből áll:
  - Felkészülés a hibára: naplózás.
  - Hiba utáni helyreállítás: a napló segítségével.
- A naplózásnak és a helyreállításnak összehangban kell lenniük.
- Több különböző naplózási és helyreállítási protokoll létezik.

Adatábáziselem, adategység

- Feltesszük, hogy az adatbázis adatbáziselemekből áll.
- Az adatbáziselemek a tárolt adatok egyfajta funkcionális egysége, amelyek értéket tranzakciókkal lehet olvasni vagy írni.
- Egy adatbáziselem lehet:
  - Reláció
  - Rekord
  - Blokk
- A naplózás szempontjából a harmadik (blokk) a legcélszerűbb választás.
- Azt fogjuk feltételezni, hogy egy adatbáziselem nem nagyobb egy blokknál.

A tranzakciók alaptevékenységei

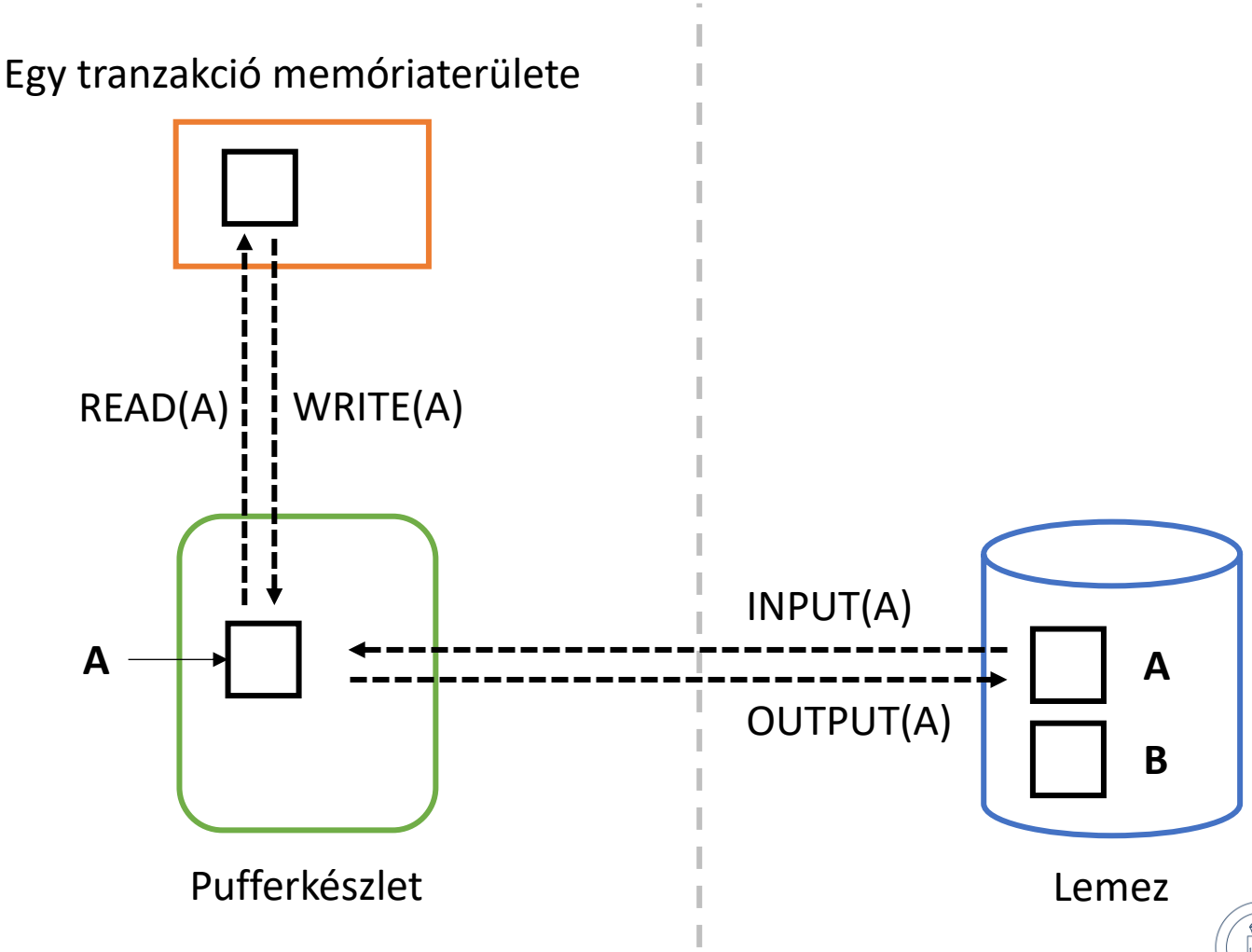
- A tranzakció és az adatbázis kölcsönhatásának három fontos helyszíne van:
  - Az adatbázis elemeit tartalmazó lemezblokkok területe.
  - A pufferkezelő által használt memóriaterület.
  - A tranzakció memóriaterülete.



Adatmozgások alapműveletei

- **INPUT(X):** Az **X** adatbáziselemet tartalmazó lemezblokk másolása a pufferkészletben.
- **READ(X, t):** Az **X** adatbáziselem bemásolása a tranzakció **t** lokális változójába. Ha X nincs a pufferkészletben, akkor előbb az INPUT(X) hajtódik végre.
- **WRITE(X, t):** A **t** lokális változó tartalmaz az **X** adatbáziselem pufferkészletben lévő másolatába íródik.
- **OUTPUT(X):** Az **X** adatbáziselemet tartalmazó puffer lemezre másolása.
- Megjegyzések:
  - A READ és WRITE műveleteket a tranzakciók használják.
  - Az INPUT és OUTPUT műveleteket a pufferkezelő alkalmazza.
  - Az OUTPUT műveletet a naplókezelő is kérheti a naplóra vonatkozólag. (FLUSH LOG)

Adatmozgások alapműveletei



Példa tranzakció

- Adott egy T tranzakció, amely megduplázza az A és B adatbáziselemek értékét.

Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B
				5	8
READ(A, t)	5	5		5	8
t = t*2	10	5		5	8
WRITE(A, t)	10	10		5	8
READ(B, t)	8	10	8	5	8
t = t*2	16	10	8	5	8
WRITE(B, t)	16	10	16	5	8
OUTPUT(A)	16	10	16	10	8
OUTPUT(B)	16	10	16	10	16

Naplózás

- A napló nem más, mint naplóbejegyzések sorozata.
- Ha hiba (rendszerhiba) keletkezik, akkor a helyreállításkezelő a naplót használja a helyreállításhoz.
- A naplót a naplókezelő írja és kizárólag írásra van megnyitva.
- A napló időrendben tartalmazza a történéseket.
- A naplóblokkok elsődlegesen a memóriában vannak tárolva, de amint lehet fizikai tárolóra írja őket a rendszer.

Naplóbejegyzések

- **<START T>** - ez a bejegyzés jelzi a **T** tranzakció végrehajtásának a kezdetét.
- **<COMMIT T>** - A **T** tranzakció rendben befejeződött, több módosítást nem kíván végrehajtani.
- **<ABORT T>** - A **T** tranzakció nem tudott sikeresen befejeződni.
- **<T, A, r, u>** - A **T** tranzakció módosította az **X** adatbáziselemet, amelynek a módosítás előtti értéke **r** volt, a módosítás utáni értéke pedig **u**.

- Megjegyzések:
  - A módosító bejegyzések egyes protokollok esetén eltérhetnek.
  - A módosító bejegyzések a **WRITE** műveletekhez készülnek el!

Naplózási megközelítések

- Altalánosan tekintve két naplózási megközelítést használhatunk:
  - **Semmiségi naplózás (UNDO)** – a nem befejezett tranzakciók hatását semmissé tesszük.
  - **Helyrehozó naplózás (REDO)** – a befejezett tranzakciók hatását megismételjük.
- Ezen kívül lesz egy harmadik típusú megközelítés, a fentiek vegyítésével: **semmiségi/helyrehozó**, azaz **UNDO/REDO** naplózás.

Semmiségi (UNDO) naplózás

- Az UNDO naplózásnál a módosító bejegyzés 3 komponensű lesz.
- **<T, X, v>** - A T tranzakció módosította az X adatbáziselemet, amelynek a módosítás előtti értéke **v** volt.
- Helyreállításkor a nem befejezett tranzakciók (COMMIT nélküli tranzakciók) hatásait semmissé tesszük, a már befejezett tranzakciókkal pedig már nem kell foglalkoznunk.

UNDO naplózás szabályai

- **U1:** Ha a T tranzakció módosítja az X adatbáziselemet, akkor a <T, X, v> típusú naplóbejegyzést azt megelőzően kell lemezre kiírni, mielőtt X új értéket a lemezre írná a rendszer.
- **U2:** Ha a tranzakció hibamentesen teljesen befejeződött, akkor a COMMIT naplóbejegyzést csak azt követően szabad a lemezre írni, hogy a tranzakció által módosított összes adatbáziselem már a lemezre íródott, de ezután viszont a lehető leggyorsabban.

Lemezre írások sorrendje UNDO naplózásnál

1. Az adatbáziselemek módosítására vonatkozó naplóbejegyzések kiírása.
2. Maguknak a módosított adatbáziselemeknek a kiírása.
3. A COMMIT naplóbejegyzés kiírása.

Példa UNDO naplózásra

#	Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B	Napló
1.							<START T>
2.	READ(A, t)	5	5		5	8	
3.	t = t*2	10	5		5	8	
4.	WRITE(A, t)	10	10		5	8	<T, A, 5>
5.	READ(B, t)	8	10	8	5	8	
6.	t = t*2	16	10	8	5	8	
7.	WRITE(B, t)	16	10	16	5	8	<T, B, 8>
8.	**FLUSH LOG**						
9.	OUTPUT(A)	16	10	16	10	8	
10.	OUTPUT(B)	16	10	16	10	16	
11.							**<COMMIT T>**
12.	**FLUSH LOG**						

Helyreállítás UNDO napló alapján

- A naplót a végéről kezdjük átvizsgálni (a legutolsó bejegyzéstől a korábbiak irányába).
- A vizsgálat során meg kell jegyezni minden olyan tranzakciót, amihez már láttunk COMMIT vagy ABORT bejegyzést.
- Ha a vizsgálat során a helyreállítás kezelő olyan módosító bejegyzést talál, amelyhez tartozó tranzakció még nem fejeződött be, akkor a módosítás hatását semmissé kell tenni (a régi értéket visszaírni).
- Ha minden naplóbejegyzés feldolgozásra került, akkor minden befejezetlen tranzakcióra vonatkozólag egy ABORT naplóbejegyzést kell a naplóba írni.
- Végezetül pedig ki kell váltani a napló lemezre írását (FLUSH LOG).

Helyreállítási algoritmus

```
S = {};                          # befejezett tranzakciók halmaza
A = {};                          # befejezetlen tranzakciók halmaza
minden naplóbejegyzésre (az utolsótól az elsőig)
    ha a naplóbejegyzés <COMMIT T> vagy <ABORT T>:
        S := S ∪ {T};
    ha a naplóbejegyzés <X, T, v>:
        ha T ∉ S:
            WRITE(X,v);          # tranzakciókezelő művelete
            OUTPUT(X);           # pufferkezelő művelete
            A := A ∪ {T};
vége
minden T_i ∈ A
    <ABORT T_i> naplóbejegyzés írása a naplóba;
vége
FLUSH LOG;                       # naplókezelő kéri a napló kiírását
```

Példa helyreállításra (UNDO napló) #1

#	Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B	Napló
1.							<START T>
2.	READ(A, t)	5	5		5	8	
3.	t = t*2	10	5		5	8	
4.	WRITE(A, t)	10	10		5	8	<T, A, 5>
5.	READ(B, t)	8	10	8	5	8	
6.	t = t*2	16	10	8	5	8	
7.	WRITE(B, t)	16	10	16	5	8	<T, B, 8>
8.	**FLUSH LOG**						
9.	OUTPUT(A)	16	10	16	10	8	
10.	OUTPUT(B)	16	10	16	10	16	
11.							**<COMMIT T>**
12.	**FLUSH LOG**						

Magyarázat (#1)

- A hiba a **12. lépést** követően jelentkezett.
- Mivel a FLUSH LOG már megtörtént, ezért tudjuk, hogy a **<COMMIT T>** naplóbejegyzés **már a lemezen van**.
- Így a helyreállítás-kezelő, amikor olvassa a naplót látni fogja ezt a bejegyzést.
- Ebből tudni fogja, hogy a **T** tranzakció **sikeresen befejeződött**, így a tranzakciót hatásait nem kell visszaállítani. Azaz, az összes T-re vonatkozó naplóbejegyzést figyelmen kívül hagyhatja.

Példa helyreállításra (UNDO napló) #2

#	Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B	Napló
1.							<START T>
2.	READ(A, t)	5	5		5	8	
3.	t = t*2	10	5		5	8	
4.	WRITE(A, t)	10	10		5	8	<T, A, 5>
5.	READ(B, t)	8	10	8	5	8	
6.	t = t*2	16	10	8	5	8	
7.	WRITE(B, t)	16	10	16	5	8	<T, B, 8>
8.	**FLUSH LOG**						
9.	OUTPUT(A)	16	10	16	10	8	
10.	OUTPUT(B)	16	10	16	10	16	
11.							**<COMMIT T>**
12.	**FLUSH LOG**						

Magyarázat (#2)

- A hiba a **11. és 12. lépések** között jelentkezett.
- Elképzelhető, hogy a **<COMMIT T>** naplóbejegyzés **már a lemezen van,** mert egy másik tranzakció miatt már a lemezre kellett írni.
    - Ilyenkor az előző eset áll fenn.
- Ha a **COMMIT** naplóbejegyzés nincs a lemezen, akkor a helyreállítás-kezelő, amikor olvassa a naplót nem fogja látni.
- Ezért a helyreállítás-kezelőnek az A és B adatbáziselemek értékét vissza kell állítania és az **<ABORT T>** bejegyzést kell a naplóba írni.
- Végül a naplót a lemezre kell írni (**FLUSH LOG**).

Példa helyreállításra (UNDO napló) #3

#	Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B	Napló
1.							<START T>
2.	READ(A, t)	5	5		5	8	
3.	t = t*2	10	5		5	8	
4.	WRITE(A, t)	10	10		5	8	<T, A, 5>
5.	READ(B, t)	8	10	8	5	8	
6.	t = t*2	16	10	8	5	8	
7.	WRITE(B, t)	16	10	16	5	8	<T, B, 8>
8.	**FLUSH LOG**						
9.	OUTPUT(A)	16	10	16	10	8	
10.	OUTPUT(B)	16	10	16	10	16	
11.							**<COMMIT T>**
12.	**FLUSH LOG**						

Magyarázat (#3)

- A hiba a **9. és 10. lépések** között jelentkezett.
- A **COMMIT** naplóbejegyzés még nincs a naplóban, így a helyreállításkezelő sem fogja látni, amikor olvassa a naplót a lemezről.
- A valóságban az **A** adatbáziselem már módosításra került a lemezen a **B** pedig még nem.
- Viszont a helyreállításkezelő ezt nem tudhatja, ezért mindkét adatbáziselem korábbi értékét vissza kell állítani és az **<ABORT T>** bejegyzést kell a naplóba írni.
- Végül a naplót a lemezre kell írni (**FLUSH LOG**).

Példa helyreállításra (UNDO napló) #4

#	Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B	Napló
1.							<START T>
2.	READ(A, t)	5	5		5	8	
3.	t = t*2	10	5		5	8	
4.	WRITE(A, t)	10	10		5	8	<T, A, 5>
5.	READ(B, t)	8	10	8	5	8	
6.	t = t*2	16	10	8	5	8	
7.	WRITE(B, t)	16	10	16	5	8	<T, B, 8>
8.	**FLUSH LOG**						
9.	OUTPUT(A)	16	10	16	10	8	
10.	OUTPUT(B)	16	10	16	10	16	
11.							**<COMMIT T>**
12.	**FLUSH LOG**						

Magyarázat (#4)

- A hiba az **5. és 6. lépések** között jelentkezett.
- Mivel a naplóban nincs módosító bejegyzés ezért biztosak lehetünk abban, hogy a lemezen sem történt még semmilyen módosítás.
- Ezért nincs mit semmissé tenni.
- Mivel a tranzakció elindult, ezért a helyreállítás kezelőnek az **<ABORT T>** bejegyzést kell a naplóba írni.
- Végül a naplót a lemezre kell írni (**FLUSH LOG**).

Ellenőrzőpont

- Helyreállításkor vissza kell mennünk a napló elejéig.
    - Ha nagy méretű a napló, akkor a visszaállítás nagyon sokáig tarthat.
- Ennek a megoldására vezetjük be az ellenőrzőpontokat (CHECKPOINT).

Ellenőrzőpont képzés

1. Megtiltjuk az új tranzakciók indítását.
2. Megvárjuk, amíg minden futó tranzakció **véget ér** (COMMIT vagy ABORT).
3. A naplót a lemezre írjuk (FLUSH LOG).
4. A naplóba bejegyzést készítünk az ellenőrzőpontról (CHECKPOINT) és a naplót újra a lemezre írjuk (FLUSH LOG).
5. Újra fogadjuk a tranzakciókat.

- **Megjegyzés**: Helyreállításkor így elég az előző CHECKPOINT bejegyzésig visszamenni a naplóban, hiszen előtte már minden tranzakció befejeződött.

Ellenőrzőpont képzés példa

- Tegyük fel, hogy a **4. bejegyzés után** úgy döntünk, hogy ellenőrzőpontot hozunk létre.
- A **T1** és **T2 aktív** tranzakciók, ezért meg kell várnunk a befejeződésüket.
- Miután befejeződtük (7., 8.) beírjuk a naplóba a **<CKPT>** bejegyzést (9. bejegyzés).

Helyreállítás:

- Tegyük fel, hogy a 12. bejegyzés után hiba lép fel.
- A helyreállítás során elég csak a <CKPT> bejegyzésig visszamennünk.

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

Mi a baj a bemutatott módszerrel?

Ellenőrzőpont leállással

- Hosszú ideig tarthat, amíg az aktív tranzakciók véget érnek.
- Amíg erre várunk, addig nem tudunk új tranzakciókat indítani.

Megoldás:

- Hozzunk létre működés közbeni ellenőrzőpontot!

Működés közbeni ellenőrzőpont képzés

1. **<START CKPT<T1,...,Tk>** naplóbejegyzés készítése, majd lemezre írása (FLUSH LOG), ahol a T1,...,Tk az éppen aktív tranzakciók nevei.
2. Meg kell várni a T1,...,Tk tranzakciók mindegyikének a befejeződését (COMMIT vagy ABORT), **nem tiltva** közben újabb tranzakciók indítását.
3. Ha a T1,...,Tk tranzakciók mindegyike befejeződött, akkor **<END CKPT>** naplóbejegyzés elkészítése, majd lemezre írása (FLUSH LOG).

Helyreállítás működés közbeni ellenőrzőponttal

- A naplót ugyanúgy a végétől visszafelé vizsgáljuk és a be nem fejezett tranzakciók hatását semmissé tesszük.
- Az ellenőrzőpont csak **abban hoz változást**, hogy meddig kell visszamennünk a naplóban.
- Két eset fordulhat elő aszerint, hogy visszafelé olvasva a naplót az **<END CKPT>** vagy a **<START CKPT(T1,…,Tk)>** naplóbejegyzést találjuk előbb.

Helyreállítás működés közbeni ellenőrzőponttal

1. **<END CKPT>** naplóbejegyzéssel találkozunk előbb.
   - Ebben az esetben tudjuk, hogy az összes még be nem fejezett tranzakcióra vonatkozó naplóbejegyzést a következő <START CKPT(T1,...,Tk)> naplóbejegyzésig megtaláljuk
   - Ott viszont megállhatunk, az annál korábbiakat akár el is dobhatjuk.

Helyreállítás működés közbeni ellenőrzőponttal

2. **<START CKPT(T1,...,Tk)>** naplóbejegyzéssel találkozunk előbb.
   - Ez azt jelenti, hogy a hiba ellenőrzőpont képzés közben fordult elő, ezért az ellenőrzőpont kezdésekor épp aktív tranzakciók még nem biztos, hogy mind befejeződtek.
   - Ekkor a be nem fejezett tranzakciók közül a legkorábban kezdődött tranzakció indulásáig kell a naplóban visszamennünk, de annál korábbra nem.
   - Az ezt megelőző olyan <START CKPT> amelyikhez tartozik END biztosan megelőzi a keresett összes tranzakció indítását leíró bejegyzéseket.
   - Ha a <START CKPT> előtt olyan <START CKPT> bejegyzést találunk, amelyhez nem tartozik END, akkor ez azt jelenti, hogy korábban is ellenőrzőpont képzés közben történt hiba. Ezeket figyelmen kívül hagyhatjuk.

Megjegyzések

- Ha egy **<END CKPT>** naplóbejegyzést kiírunk lemezre, akkor az azt megelőző **<START CKPT(T1,…,Tk)>** bejegyzésnél korábbi naplóbejegyzéseket törölhetjük.
- Ha az ugyanazon tranzakcióra vonatkozó naplóbejegyzéseket **összeláncoljuk**, akkor nem kell a napló minden bejegyzését átnéznünk, elegendő csak az adott tranzakció bejegyzéseinek a láncán visszafelé haladnunk.

Példa helyreállításra (#1)

- A rendszerhiba a **13. bejegyzés** után lépett fel.
- Ha visszafelé haladunk a naplóban, akkor először **<END CKPT>** bejegyzéssel találkozunk (a módosító bejegyzés után).
- Ebből tudjuk, hogy az összes be nem fejezett tranzakció az előző **<START CKPT>** után indulhatott csak el.
- Ezért csak az 5. naplóbejegyzésig kell visszamennünk (**<START CKPT>**).

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

Példa helyreállításra (#2)

- A rendszerhiba a **10. bejegyzés** után lépett fel.
- Visszafelé haladva a naplóban, látni fogjuk, hogy a **T2** és **T3** tranzakciók még nincsenek befejezve, ezért a hatásukat semmissé kell tenni.
- Ahogy haladunk a naplóban, meg fogjuk találni az ellenőrzőpont kezdetét (**5. bejegyzés**), ahol meg tudjuk nézni az abban a pillanatban aktív tranzakciókat.
- Mivel a **T1**-hez találtunk **COMMIT-ot** (9. bejegyzés), ezért csak a **T2** kezdetéig kell visszamennünk (3. bejegyzés), tovább nem.

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

Helyrehozó (REDO) naplózás

- Az REDO naplózásnál a módosító bejegyzés szintén 3 komponensű lesz.
- **<T, X, v>** - A T tranzakció módosította az X adatbáziselemet, amelynek a módosítás utáni értéke **v** lett.
- Helyreállításkor a már befejezett tranzakciók (COMMIT-al rendelkező tranzakciók) hatásait ismét végrehajtjuk (helyrehozzuk), a befejezetlen (COMMIT nélküli) tranzakciókkal pedig nem kell foglalkoznunk.
- Egy új naplóbejegyzést is bevezetünk, az **<END T>**-t, ami azt jelenti, hogy a T tranzakció teljesen lezárult, azaz minden módosítása és a tranzakcióhoz kapcsolódó összes bejegyzés is a lemezen van.

REDO naplózás szabályai

- **R1:** Mielőtt az adatbázis bármely X elemét a lemezen módosítanánk, az X módosítására vonatkozó összes naplóbejegyzésnek, azaz a módosító bejegyzéseknek és a COMMIT bejegyzésnek is a lemezre kell kerülnie.

Lemezre írások sorrendje REDO naplózásnál

1. A módosító naplóbejegyzések lemezre írása.
2. A COMMIT naplóbejegyzés lemezre írása.
3. Az adatbáziselemek értékének cseréje a lemezen.
4. **<END T>** bejegyzés naplóba írása, majd a napló lemezre írása.

Példa REDO naplózásra

#	Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B	Napló
1.							<START T>
2.	READ(A, t)	5	5		5	8	
3.	t = t*2	10	5		5	8	
4.	WRITE(A, t)	10	10		5	8	<T, A, 5>
5.	READ(B, t)	8	10	8	5	8	
6.	t = t*2	16	10	8	5	8	
7.	WRITE(B, t)	16	10	16	5	8	<T, B, 8>
8.							**<COMMIT T>**
9.	**FLUSH LOG**						
10.	OUTPUT(A)	16	10	16	10	8	
11.	OUTPUT(B)	16	10	16	10	16	
12.							**<END T>**
13.	**FLUSH LOG**						

Helyreállítás REDO napló alapján

- Először átvizsgáljuk a naplót és feljegyezzük azokat a tranzakciókat, amelyekhez volt COMMIT bejegyzés, de nem volt END.
    - Ezeknek a hatását kell helyrehoznunk.
- Ezután a naplót a legrégibb bejegyzésektől az újabbak felé haladva átvizsgáljuk (fentről lefelé) – ellenkező irányba, mint az UNDO estén.
    - A módosításokat újra végrehajtjuk.
- A legvégén a helyrehozott tranzakciókhoz END naplóbejegyzést írunk be.

Helyreállítási algoritmus (REDO)

```
INPUT: S      # tranzakciók, amelyekhez van COMMIT, de nincs END
minden naplóbejegyzésre (az elsőtől az utolsóig)
    ha a naplóbejegyzés <X, T, v>:
        ha T ∈ S:
            WRITE(X,v);          # tranzakciókezelő művelete
            OUTPUT(X);           # pufferkezelő művelete
vége
minden T_i ∈ S
    <END T_i> naplóbejegyzés írása a naplóba;
vége
```

Módosított REDO napló

- A tankönyvben (Ullman) egy kissé eltérő REDO napló szerepel, ezt nevezzük módosított REDO-nak.
- Ebben a változatban nem használunk <END T> bejegyzést a befejezett tranzakciókra, helyette a helyreállítás során a be nem fejezetteket jelöljük meg <ABORT T> bejegyzéssel.

Helyreállítás a módosított REDO napló esetén

1. Meghatározzuk a **befejezett** tranzakciókat (amelyekhez van COMMIT).
2. Elemezzük a naplót az elejétől kezdve és minden **módosító** bejegyzésnél:
   - Ha a tranzakcióhoz **nincs COMMIT**, akkor nem kell tenni semmit.
   - Ha a tranzakcióhoz **van COMMIT**, akkor újra elvégezzük a módosítást.
3. Minden T be nem fejezett tranzakcióra vonatkozóan **<ABORT T>** naplóbejegyzést kel la naplóba írni, és a naplót ki kell írni a lemezre.

Példa REDO naplózásra

#	Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B	Napló
1.							<START T>
2.	READ(A, t)	5	5		5	8	
3.	t = t*2	10	5		5	8	
4.	WRITE(A, t)	10	10		5	8	<T, A, 5>
5.	READ(B, t)	8	10	8	5	8	
6.	t = t*2	16	10	8	5	8	
7.	WRITE(B, t)	16	10	16	5	8	<T, B, 8>
8.							**<COMMIT T>**
9.	**FLUSH LOG**						
10.	OUTPUT(A)	16	10	16	10	8	
11.	OUTPUT(B)	16	10	16	10	16	

Magyarázat (#1)

- A rendszerhiba a **10. lépés** után következik be.
- A **<COMMIT T>** bejegyzésnél már a lemezen van, ezért a rendszer a tranzakciót befejezetként azonosítja.
- Ezért a naplót az elejétől kezdve kell vizsgálni és minden módosítást újra végrehajtani.

Példa REDO naplózásra

#	Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B	Napló
1.							<START T>
2.	READ(A, t)	5	5		5	8	
3.	t = t*2	10	5		5	8	
4.	WRITE(A, t)	10	10		5	8	<T, A, 5>
5.	READ(B, t)	8	10	8	5	8	
6.	t = t*2	16	10	8	5	8	
7.	WRITE(B, t)	16	10	16	5	8	<T, B, 8>
8.							**<COMMIT T>**
9.	**FLUSH LOG**						
10.	OUTPUT(A)	16	10	16	10	8	
11.	OUTPUT(B)	16	10	16	10	16	

Magyarázat (#2)

- A rendszerhiba az **5. lépés** után következik be.
- A **<COMMIT T>** bejegyzés **még nincs** a naplóban, így a lemezen sincs.
- Ezért a tranzakció **befejezetlennek** tekintendő.
- Ennek megfelelően biztosak lehetük benne, hogy a lemezen az adatbáziselemek még nem változtak, ezért nincs mit helyreállítani.
- Végül egy **<ABORT T>** bejegyzést írunk a naplóba.

Ellenőrzőpont képzés REDO napló esetén

- **Probléma**: A befejeződött tranzakciók módosításainak a lemezre írása a befejeződés után sokkal később is történhet.
- **Következmény**: ezért az egy pillanatban aktív tranzakciók számát nincs értelme korlátozni, tehát nincs értelme az egyszerű ellenőrzőpont képzésnek sem -> csak működés közbeni ellenőrzőpont lesz.
- A **célunk**, hogy az ellenőrzőpont kezdete és vége között, a már befejezett tranzakciók módosításait a lemezre kell írni.
- Ehhez számon kell tartani az ún**. piszkos puffereket (dirty buffers)**, amelyekben már végrehajtott, de lemezre még ki nem írt módosítások vannak.

Ellenőrzőpont képzés

1. A **<START CKPT(T1,…,Tk)>** naplóbejegyzés elkészítése és lemezre írása, ahol a T1,…Tk az összes éppen aktív tranzakció.
2. Az összes olyan adatbáziselem kiírása a lemezre, amelyeket olyan tranzakciók írtak, amelyek a **<START CKPT(T1,…,Tk)>** naplóba írásakor már befejeződtek, de a puffereik lemezre még nem kerültek.
3. **<END CKPT>** bejegyzés naplóba írása és a napló lemezre írása.

Példa helyreállításra (#1)

- A rendszerhiba a **12. bejegyzés** után lépett fel.
- A 6. lépésben elindított ellenőrzőpont képzés már befejeződött (10. bejegyzés).
- Ezért tudjuk, hogy azon tranzakciók módosításai, amelyek a START előtt már befejeződtek, a lemezen vannak.
- Így elég azokat a tranzakciókat vizsgálni (helyreállítani), amelyek a START bejegyzésben fel vannak sorolva vagy azután indultak el (T2, T3).
- A START-ban szereplő legkorábban elkezdődött tranzakcióig kell csak visszamennünk.

#	Bejegyzés
1.	<START T1>
2.	<T1, A, 5>
3.	<START T2>
4.	<COMMIT T1>
5.	<T2, B, 10>
6.	**<START CKPT(T2)>**
7.	<T2, C, 15>
8.	<START T3>
9.	<T3, D, 20>
10.	**<END CKPT>**
11.	<COMMIT T2>
12.	<COMMIT T3>

Példa helyreállításra (#2)

- A rendszerhiba a **11. bejegyzés** után lépett fel.
- Nagyon hasonló az előző példához.
- Az egyetlen különbség, hogy a T3 tranzakció nem fejeződött be, így annak a műveleteit nem kell újra végrehajtanunk.
- Helyreállítás során a naplóba be kell írnunk az <ABORT T3> bejegyzést.

#	Bejegyzés
1.	<START T1>
2.	<T1, A, 5>
3.	<START T2>
4.	<COMMIT T1>
5.	<T2, B, 10>
6.	**<START CKPT(T2)>**
7.	<T2, C, 15>
8.	<START T3>
9.	<T3, D, 20>
10.	**<END CKPT>**
11.	<COMMIT T2>
12.	<COMMIT T3>

Példa helyreállításra (#3)

- A rendszerhiba a **8. bejegyzés** után lépett fel.
- Az utolsó ellenőrzőpont bejegyzés egy START és nem END.
- Ezért az ezt megelőző **<END CKPT>** bejegyzéshez tartozó **<START CKPT(S1,…,Sm)>** bejegyzésig vissza kell mennünk, és helyre kell állítanunk azokat a tranzakciókat, amelyek ez után indultak vagy szerepelnek az S1,…,Sm tranzakciók között.
- Ha nincs előző **<END CKPT>,** akkor a napló elejéig kell visszamenni.

#	Bejegyzés
1.	<START T1>
2.	<T1, A, 5>
3.	<START T2>
4.	<COMMIT T1>
5.	<T2, B, 10>
6.	**<START CKPT(T2)>**
7.	<T2, C, 15>
8.	<START T3>
9.	<T3, D, 20>
10.	**<END CKPT>**
11.	<COMMIT T2>
12.	<COMMIT T3>

Az UNDO és REDO naplózás hátrányai

- Az **UNDO** naplózás esetén az adatokat a tranzakció befejezésekor azonnal a lemezre kell írni, így **nő a végrehajtandó lemezműveletek száma**.
- A **REDO** naplózás minden módosított adatbáziselem pufferben tartását igényli egészen a tranzakció teljes befejezésig, így **nő a tranzakciók átlagos pufferigénye.**

- Próbáljuk meg megoldani ezeket a problémákat.

Semmiségi/helyrehozó (UNDO/REDO) naplózás

- Egyesítsük a két fajta megközelítés előnyeit.
- A naplóbejegyzés 4 komponensű. A **<T, X, u, v>** azt jelenti, hogy a **T** tranzakció módosította az X adatbáziselemet, amelynek régi értéke **u**, új értéke pedig **v.**
- A <COMMIT T> bejegyzés megelőzheti, de követheti is az adatbáziselemek lemezen történő megváltoztatását.
- **UR1 szabály**: Mielőtt az adatbázis bármely X elemének értéket valamely **T** tranzakció által végzett módosítás miatt a lemezen módosítanánk, ezt megelőzően a módosító bejegyzésnek a lemezre kell kerülnie.

Helyreállítás UNDO/REDO esetén

A következő alapelveket követjük:

- **(REDO):** A legkorábbitól kezdve állítsuk helyre minden befejezett tranzakció hatását.
- **(UNDO):** A legutolsótól kezdve tegyük semmissé minden be nem fejezett tranzakció tevékenységeit.

Példa UNDO/REDO naplózásra

#	Tevékenység	t	Mem A	Mem B	Lemez A	Lemez B	Napló
1.							<START T>
2.	READ(A, t)	5	5		5	8	
3.	t = t*2	10	5		5	8	
4.	WRITE(A, t)	10	10		5	8	<T, A, 5, 10>
5.	READ(B, t)	8	10	8	5	8	
6.	t = t*2	16	10	8	5	8	
7.	WRITE(B, t)	16	10	16	5	8	<T, B, 8, 16>
8.	**FLUSH LOG**						
9.	OUTPUT(A)	16	10	16	10	8	
10.							**<COMMIT T>**
11.	OUTPUT(B)	16	10	16	10	16	

UNDO vagy REDO legyen előbb?

- Tegyük fel, hogy teljesen **befejeződött** T tranzakció módosított egy adatbáziselemet, ezért helyreállításkor az elem új értékát állítjuk helyre.
- De emellett egy másik (**be nem fejezett**) tranzakció is módosította **ugyanazt** az adatbáziselemet, ezért helyreállításkor a régi értéket kell visszaírnunk.
- Attól függően egy UNDO -> REDO vagy REDO -> UNDO sorrendet választunk, az eredmény eltérő lesz.
    - És egyik sorrend sem lenne helyes! (nem konzisztens adatbázis)
- **Mit tegyünk?** A konkurencia kezelésnél fogjuk megoldani, hogy ilyen esetek ne fordulhassanak elő.

Befejezett változtatások megsemmisítése

- Előfordulhat, hogy egy tranzakció már befejeződött, a változtatásait már a lemezre írtuk, a naplóba bekerült a COMMIT, viszont a COMMIT naplóbejegyzés **még nem került a lemezre** és ilyenkor történik a hiba.
- A helyreállításkor ezért egy teljesen befejezett tranzakciót tennénk semmissé.

Megoldás:

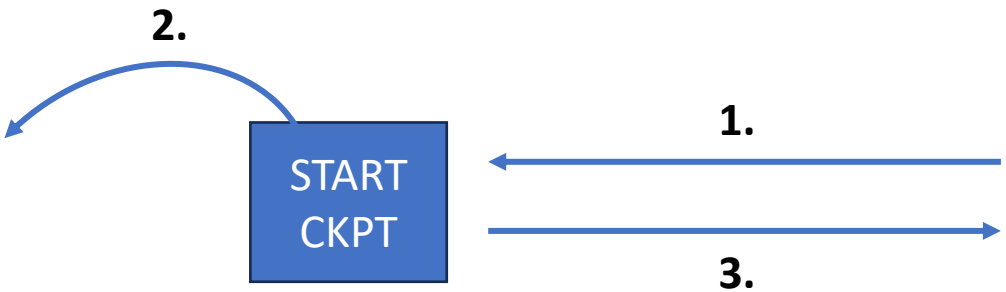
- **UR2 szabály:** A <COMMIT T> naplóbejegyzést azonnal a lemezre kell írni, amint megjelenik a naplóban.

Ellenőrzőpont képzés UNDO/REDO esetén

1. Írjunk a naplóba **<START CKPT(T1,...,Tk)>** naplóbejegyzést, ahol T1,...,Tk az aktív tranzakciók, majd írjuk a naplót a lemezre.
2. Írjuk a lemezre az összes **piszkos puffert**, tehát azokat, amelyek módosított adatbáziselemeket tartalmaznak.
3. Írjunk **<END CKPT>** bejegyzést a naplóba, majd írjuk a naplót a lemezre.

Helyreállítás

1. A napló végétől megyünk az utolsó érvényes ellenőrzőpont kezdetéig:
   - Meghatározzuk a befejezett tranzakciók halmazát (S).
   - Semmissé tesszük azoknak a tranzakcióknak a hatását, amelyek nincsenek S-ben.
2. Az ellenőrzőpont kezdetekor aktív tranzakciók hatását semmissé tesszük.
   - Csak azokat kell vizsgálni, amelyek nincsenek benne az S-ben.
3. Az utolsó ellenőrzpont kezdetétől a napló végéig haladva:
   - Helyrehozzuk az S tranzakcióinak a hatását.



Példa helyreállításra (#1)

- A rendszerhiba a **12. bejegyzés** után lépett fel.
- Ebben az esetben elég a START CKPT bejegyzésig visszamenni, mert:
  - Bár a T2 aktív volt a START időpontjában, de a 11. lépésben befejeződött és minden módosítását elvégezte az END CKPT előtt.
  - Ezért tudjuk, hogy ez módosítás már a lemezen van.
- Tehát a T2 és T3 tranzakciókat helyreállítjuk, a T1-et pedig figyelmen kívül hagyjuk (mivel a még a START CKPT előtt befejeződött).

#	Bejegyzés
1.	<START T1>
2.	<T1, A, 4, 5>
3.	<START T2>
4.	<COMMIT T1>
5.	<T2, B, 9, 10>
6.	**<START CKPT(T2)>**
7.	<T2, C, 14, 15>
8.	<START T3>
9.	<T3, D, 19, 20>
10.	**<END CKPT>**
11.	<COMMIT T2>
12.	<COMMIT T3>

Példa helyreállításra (#2)

- A rendszerhiba a **11. bejegyzés** után lépett fel.
- A T2 befejezett, de a T3 befejezetlen tranzakció.
- T2 hatását helyreállítjuk, a T3 hatását semmissé tesszük.
- Csak a START CKPT-ig kell visszamennünk.

#	Bejegyzés
1.	<START T1>
2.	<T1, A, 4, 5>
3.	<START T2>
4.	<COMMIT T1>
5.	<T2, B, 9, 10>
6.	**<START CKPT(T2)>**
7.	<T2, C, 14, 15>
8.	<START T3>
9.	<T3, D, 19, 20>
10.	**<END CKPT>**
11.	<COMMIT T2>
12.	<COMMIT T3>

Példa helyreállításra (#3)

- A rendszerhiba a **7. bejegyzés** után lépett fel.
- Ha a hiba az <END CKPT> bejegyzés előtt lép fel, akkor egyszerűen figyelmen kívül hagyjuk a hozzá tartozó START CKPT bejegyzést és az előzőek szerint járunk el.

#	Bejegyzés
1.	<START T1>
2.	<T1, A, 4, 5>
3.	<START T2>
4.	<COMMIT T1>
5.	<T2, B, 9, 10>
6.	**<START CKPT(T2)>**
7.	<T2, C, 14, 15>
8.	<START T3>
9.	<T3, D, 19, 20>
10.	**<END CKPT>**
11.	<COMMIT T2>
12.	<COMMIT T3>

Gyakorlati megjegyzések

- **Write-ahead Log (WAL)** elv: előbb naplózunk, utána módosítunk. Azaz mielőtt bármilyen adatbáziselemet módosítanánk a lemezen, előbb a módosításra vonatkozó naplóbejegyzéseknek kell a lemezre kerülnie.
- A valós implementációkban a **WAL** elvet követik, amely a bemutatott megközelítések közül az **UNDO/REDO** naplózáshoz áll a legközelebb.

Naplózás az Oracle adatbázisban

- Az Oracle az UNDO és a REDO naplózás egy **speciális keverékét** valósítja meg.
- Rendszerhiba esetén a helyreállítás-kezelő automatikusan aktivizálódik, amikor a rendszer újraindul.
- A helyreállítás a napló **(redo log)** alapján történik. Két részből áll: **online** és **archivált** naplóból.
- Az online napló kettő vagy több online naplófájlból áll.
- A naplóbejegyzések ideiglenesen az **SGA** (System Global Area) memóriapuffereiben tárolódnak, amelyeket a Log Writer (**LGWR**) háttérfolyamat folyamatosan ír ki a lemezre.

Redo log

- Az online naplófájlok ciklikusan töltődnek fel.
- Pl. ha két fájlból áll, akkor először az elsőt írja tele a LGWR, aztán a másodikat, és ezután újra az elsőt.
- A rendszer azt is lehetővé teszi, hogy az egyes fájlokat több példányban is tároljuk (akár külön lemezen). Ezek egyszerre módosulnak és ha az egyik lemez megsérül, a másik még mindig rendelkezésre áll.
- Lehetőség van arra, hogy a megtelt online naplófájlokat archiváljuk, ezek fogják alkotni az archivált vagy online naplót.

ARCHIVELOG mód

- A naplókezelő két módban működhet: **ARCHIVELOG** és **NOARCHIVELOG**.
- **ARCHIVELOG** módban az adatbázis teljesen visszaállítható, ilyenkor a betelt naplókról másolatok készülnek (másik lemezre, akár fizikailag távol)
- **NOARCHIVELOG** – a betelt naplók felülíródnak, másolat nélkül.
    - Ezt főleg teszteléskor érdemes használni, vagy ha nem fontos az adatok visszaállíthatósága.

Rollback szegmensek

- A tranzakciók hatásainak a **semmissé** tételéhez szükséges információkat a **rollback szegmensek** (UNDO szegmensek) tartalmazzák.
- Minden adatbázisban van egy vagy több rollback szegmens, amely a tranzakciók által módosított adatok **régi értékeit tárolja**.
- Ezek felhasználhatóak az adatbázis helyreállítására, a tranzakciók visszagörgetésére (ROLLBACK utasítás), olvasási konzisztencia biztosítására.
- A rollback szegmens bejegyzésekből áll, amely tárolja a módosított blokk azonosítóját és régi értékeit.
- Egy tranzakcióhoz tartozó bejegyzések össze vannak **láncolva**, így könnyen visszakereshetőek.

Rollback szegmensek

- A rollback szegmenseket sem a felhasználók sem az adminisztrátorok nem olvashatják, egyedül a rendszer (SYS a tulajdonos).
- Ha egy tranzakció befejeződött, a hozzá tartozó rollback szegmens még nem feltétlenül törölhető, mert elképzelhető, hogy más tranzakciónak szüksége van még az adatbáziselemek régi értékeire.
- Amikor betelik a szegmens, akkor a rendszer az elejéről kezdi újra feltölteni.
- Egy adatbázis létrehozáskor automatikusan létrejön egy SYSTEM nevű rollback szegmens, amely nem törölhető. Emellett létre hozhatunk több rollback szegmenst is.

A naplózás naplózása

- Amikor egy rollback bejegyzés a rollback szegmensbe kerül, a naplóban erről is készül egy bejegyzés, hiszen a rollback szegmensek is az adatbázis részét képezik.

1. Ha rendszerhiba történik, először a napló alapján visszaállításra kerül az adatbázisnak a hiba előtti állapota, amely inkonzisztens is lehet. Ez a folyamat a **rolling forward**.
2. A helyreállítás során a rollback szegmens is visszaállítódik, amelyben az aktív tranzakciók által végrehajtott tevékenységek semmissé tételéhez szükséges információk találhatók. Ezek alapján el tudjuk végezni a tranzakciók visszagörgetését. Ez a folyamat a **rolling back**.

Az adatbázis mentése

- Az adatbázisról **mentéseket** is készíthetünk (a naplózás mellett)
- Hiba esetén ilyenkor először a mentésből visszaállítjuk az adatbázist és a naplót felhasználva visszaállítjuk a naprakész állapotot.
- A mentés két szintje:
  - **Teljes mentés** (full dump) – az egész adatbázisról másolat készül.
  - **Növekményes mentés** (incremental dump) – az adatbázisnak csak azon elemeiről készítünk másolatot, melyek az utolsó teljes vagy növekményes mentés óta megváltoztak.

Összefoglalás

- Naplózási megközelítések:
  - Semmiségi (UNDO)
  - Helyrehozó (REDO)
  - Semmiségi/helyrehozó (UNDO/REDO)
- Gyakorlatban a **WAL** elvet követik, a naplózás legjobban az UNDO/REDO-hoz áll legközelebb.

Tankönyv fejezetek

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom:

  Adatbázisrendszerek megvalósítása

  - 8. fejezet (teljesen): A rendszerhibák kezelése
- Silberschatz, Korth, & Sudarshan: **Database System Concepts**
  - Chapter 19. Recovery System