

Adatbázisok elmélete 24. előadás

Katona Gyula Y.

Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.

I. B. 137/b

`kiskat@cs.bme.hu`

`http://www.cs.bme.hu/~kiskat`

2005

Védekezés hibák ellen, helyreállítás (emlékeztető)

Alapprobléma: nem fut le valamelyik tranzakció (sérül az atomiság) és emiatt inkonzisztens lesz az adatbázis.

Cél az, hogy újra konzisztens állapotba hozzuk az adatbázist (visszacsinálás vagy befejezés) úgy, hogy a tartósság megmaradjon: ha egy tranzakció már befejezte a munkáját, akkor annak hatása ne vesszen el.

Védekezés hibák ellen, helyreállítás (emlékeztető)

Alapprobléma: nem fut le valamelyik tranzakció (sérül az atomiság) és emiatt inkonzisztens lesz az adatbázis.

Cél az, hogy újra konzisztens állapotba hozzuk az adatbázist (visszacsinálás vagy befejezés) úgy, hogy a tartósság megmaradjon: ha egy tranzakció már befejezte a munkáját, akkor annak hatása ne vesszen el.

Alapfogalmak (emlékeztető)

COMMIT pont: az a pont, amikor a tranzakció minden érdemi munkával megvan, programhiba vagy ütemező miatt ABORT már biztos nem lehet.

Nem biztos, hogy ekkor minden hatása látszik is már a tranzakciónak, lehet, hogy nincs minden írása véglegesítve, de minden készen áll már erre.

Piszkos adat: Olyan adat, amit még nem COMMIT-ált tranzakció (azaz olyan, aki még meghalhat) írt az adatbázisba.

Ha ilyet olvas egy másik tranzakció, akkor baj lehet, ha az első ABORT-ál, de a második nem.

Lavina: egymás után kell ABORT-okat elrendelni a tranzakcióknál piszkos adatból eredő hiba miatt.

Megoldások piszkos adat és lavina ellen (emlékeztető)

Különböző megoldások a tranzakcióhibákból (programhiba vagy rendszer általi ABORT) származó problémákra:

- Olyan tranzakciótól, aki nem COMMIT-ált, nem olvasunk. (Nem olvasunk olyan értéket, amit olyan tranzakció írt, akinek még nem volt COMMIT).
- Hagyjuk, hogy minden tranzakció azt csinálja, amit akar, ha lavina lesz, akkor majd megoldjuk (UNDO protokoll)
- Zárolási protokollt kényszerítünk a tranzakciókra, ami biztosítja, hogy nem lesz piszkos adatból probléma, lavina:

szigorú 2PL:

- ★ 2PL
- ★ DB-be írás csak COMMIT után
- ★ zárok elengedése csak írás után

Tétel. *Ha mindegyik tranzakció a szigorú 2PL protokollt követi, akkor az ütemezés sorosítható lesz és lavinamentes.*

REDO protokoll-naplózás

Fő szabály:

- Mielőtt az a lemezen módosítunk egy X adatelemet, a (T, X, ν) és a $(T, COMMIT)$ bejegyzést is ki kell írunk a naplóba.

REDO protokoll-naplózás

Fő szabály:

- Mielőtt az a lemezen módosítunk egy X adatelemet, a (T, X, ν) és a $(T, COMMIT)$ bejegyzést is ki kell írunk a naplóba.

REDO protokoll

Ez a szigorú 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron:
 $(T_i, BEGIN)$, $(T_i, A, \text{új érték})$, $(T_i, ABORT)$

REDO protokoll-naplózás

Fő szabály:

- Mielőtt az a lemezen módosítunk egy X adatelemet, a (T, X, ν) és a $(T, COMMIT)$ bejegyzést is ki kell írunk a naplóba.

REDO protokoll

Ez a szigorú 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron:
 $(T_i, BEGIN)$, $(T_i, A, \text{új érték})$, $(T_i, ABORT)$
2. **COMMIT** után a napló háttértárra írása

REDO protokoll-naplózás

Fő szabály:

- Mielőtt az a lemezen módosítunk egy X adatelemet, a (T, X, ν) és a $(T, COMMIT)$ bejegyzést is ki kell írunk a naplóba.

REDO protokoll

Ez a szigorú 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron:
 $(T_i, BEGIN)$, $(T_i, A, \text{új érték})$, $(T_i, ABORT)$
2. COMMIT után a napló háttértárra írása
3. Tényleges írás az adatbázisba a háttértáron, nem a pufferben

REDO protokoll-naplózás

Fő szabály:

- Mielőtt az a lemezen módosítunk egy X adatelemet, a (T, X, ν) és a $(T, COMMIT)$ bejegyzést is ki kell írunk a naplóba.

REDO protokoll

Ez a szigorú 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron:
 $(T_i, BEGIN)$, $(T_i, A, \text{új érték})$, $(T_i, ABORT)$
2. COMMIT után a napló háttértárra írása
3. Tényleges írás az adatbázisba a háttértáron, nem a pufferben
4. Zárak elengedése

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- Különbség a az UNDO protokollhoz képest:
 - ★ Az adat változás utáni értékét jegyezzük fel a naplóba

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- **Különbség a az UNDO protokollhoz képest:**
 - ★ Az adat változás utáni értékét jegyezzük fel a naplóba
 - ★ Máshová rakjuk a COMMIT-ot, a kiírás elé

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- **Különbség a az UNDO protokollhoz képest:**
 - ★ Az adat változás utáni értékét jegyezzük fel a naplóba
 - ★ Máshová rakjuk a COMMIT-ot, a kiírás elé \Rightarrow **megtelhet a puffer**

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- **Különbség a az UNDO protokollhoz képest:**
 - ★ Az adat változás utáni értékét jegyezzük fel a naplóba
 - ★ Máshová rakjuk a COMMIT-ot, a kiírás elé \Rightarrow **megtelhet a puffer**
 - ★ Az UNDO protokoll esetleg túl gyakran akar írni

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- **Különbség a az UNDO protokollhoz képest:**
 - ★ Az adat változás utáni értékét jegyezzük fel a naplóba
 - ★ Máshová rakjuk a COMMIT-ot, a kiírás elé \Rightarrow **megtelhet a puffer**
 - ★ Az UNDO protokoll esetleg túl gyakran akar írni \Rightarrow **itt el lehet halasztani az írást**

Példa

T	t	A_M	B_M	A_D	B_D	Napló
				8	8	(T , BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	($T, A, 16$)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	($T, B, 16$)
						(T , COMMIT)
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk
2. A napló mentett részét nézzük visszafele, megkeressük azokat a tranzakciókat, amikre volt már COMMIT (a többi nem érdekes, mert ha még nem volt a COMMIT-juk kimentve, akkor nem is írtak a DB-be)

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk
2. A napló mentett részét nézzük visszafele, megkeressük azokat a tranzakciókat, amikre volt már COMMIT (a többi nem érdekes, mert ha még nem volt a COMMIT-juk kimentve, akkor nem is írtak a DB-be)
3. Addig megyünk vissza a naplóban, amíg biztosan konzisztens állapotot nem találunk (eleje vagy CHECKPOINT)

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk
2. A napló mentett részét nézzük visszafele, megkeressük azokat a tranzakciókat, amikre volt már COMMIT (a többi nem érdekes, mert ha még nem volt a COMMIT-juk kimentve, akkor nem is írtak a DB-be)
3. Addig megyünk vissza a naplóban, amíg biztosan konzisztens állapotot nem találunk (eleje vagy CHECKPOINT)
4. A COMMIT-tált tranzakciók írásait előlről kezdve (a legelső COMMIT-ált elejétől) megismételjük (ha már egyszer be volt írva, az se baj, akkor csak felülírjuk ugyanazzal). Ezt meg tudjuk tenni, mert ismerjük az új értékeket.

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk
2. A napló mentett részét nézzük visszafele, megkeressük azokat a tranzakciókat, amikre volt már COMMIT (a többi nem érdekes, mert ha még nem volt a COMMIT-juk kimentve, akkor nem is írtak a DB-be)
3. Addig megyünk vissza a naplóban, amíg biztosan konzisztens állapotot nem találunk (eleje vagy CHECKPOINT)
4. A COMMIT-tált tranzakciók írásait előlről kezdve (a legelső COMMIT-ált elejétől) megismételjük (ha már egyszer be volt írva, az se baj, akkor csak felülírjuk ugyanazzal). Ezt meg tudjuk tenni, mert ismerjük az új értékeket.
5. Minden nem befejezett T_i tranzakcióra ($T_i, ABORT$)-ot írunk a napló végére, (FLUSH LOG)

Megjegyzések a REDO helyreállításához

- Ha a napló háttértáron van, akkor mindent újra tudunk csinálni, ami meg még nem került ki, azzal kapcsolatban változtatás se történt, nem kell visszacsinálni semmit.

Megjegyzések a REDO helyreállításához

- Ha a napló háttértáron van, akkor mindent újra tudunk csinálni, ami meg még nem került ki, azzal kapcsolatban változtatás se történt, nem kell visszacsinálni semmit.
- Ha a helyreállítás során lenne újra hiba, akkor a napló marad, mert az már kint van, ez alapján újra kezdetjük a helyreállítást.

Megjegyzések a REDO helyreállításhoz

- Ha a napló háttértáron van, akkor mindent újra tudunk csinálni, ami meg még nem került ki, azzal kapcsolatban változtatás se történt, nem kell visszacsinálni semmit.
- Ha a helyreállítás során lenne újra hiba, akkor a napló marad, mert az már kint van, ez alapján újra kezdhetjük a helyreállítást.
- **Eredmény:** a háttértárra kikerült COMMIT-oknak megfelelő tranzakciók eredménye látszik, a többiekéből pedig semmi.

Példa

T	t	A_M	B_M	A_D	B_D	Napló
				8	8	(T , BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	($T, A, 16$)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	($T, B, 16$)
						(T , COMMIT)
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						

Példa

T_1	napló
	(T_1, BEGIN)
LOCK(A) LOCK(B)	
	(T_1, A, x) (T_1, B, y) (T_1, COMMIT)
WRITE(A) WRITE(B) UNLOCK(A) UNLOCK(B)	

(T_1, A, x) jelentése: T_1 A-ba x-et írja

Ekkor a tényleges írás nem történik meg, csak a naplóba kerül ez bele, a tényleges írás csak a COMMIT után jön.

Példa

T_1	napló
	(T_1, BEGIN)
LOCK(A) LOCK(B)	
	(T_1, A, x) (T_1, B, y) (T_1, COMMIT)
WRITE(A) WRITE(B) UNLOCK(A) UNLOCK(B)	

(T_1, A, x) jelentése: T_1 A-ba x-et írja

Ekkor a tényleges írás nem történik meg, csak a naplóba kerül ez bele, a tényleges írás csak a COMMIT után jön.

Ha a belső tár hibája a COMMIT háttértárra írása előtt történik, akkor még semmi valódi írás nem volt, azaz semmit se kell csinálni. Ha azonban a COMMIT után van, akkor a naplóban megvan minden utasítás, újra meg lehet csinálni T_1 -et.

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz
4. A naplóba beírjuk, hogy CHECKPOINT

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz
4. A naplóba beírjuk, hogy CHECKPOINT
5. A naplót is háttértárra írjuk

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT (T_1, \dots, T_k)), ahol T_i az összes éppen aktív tranzakció

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy **(START CHECKPOINT (T_1, \dots, T_k))**, ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: **FLUSH LOG**

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy **(START CHECKPOINT (T_1, \dots, T_k))**, ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: **FLUSH LOG**
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy **(START CHECKPOINT (T_1, \dots, T_k))**, ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: **FLUSH LOG**
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. **(END CHECKPOINT)** és **(FLUSH LOG)**

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT (T_1, \dots, T_k)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy **(START CHECKPOINT (T_1, \dots, T_k))**, ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: **FLUSH LOG**
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. **(END CHECKPOINT)** és **(FLUSH LOG)**

Visszaállítás

- Visszafelé olvasva, ha előbb **(END CHECKPOINT)** van \Rightarrow elég visszamenni a következő **START CHECKPOINT**-ig.

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT (T_1, \dots, T_k)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van \Rightarrow elég visszamenni a következő START CHECKPOINT-ig. \Rightarrow innen előre minden itt szereplő T_i -re és minden később kezdődő más tranzakcióra REDO

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT (T_1, \dots, T_k)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van \Rightarrow elég visszamenni a következő START CHECKPOINT-ig. \Rightarrow innen előre minden itt szereplő T_i -re és minden később kezdődő más tranzakcióra REDO
- Ha előbb (START CHECKPOINT (T_1, \dots, T_k))-ot találunk

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT (T_1, \dots, T_k)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van \Rightarrow elég visszamenni a következő START CHECKPOINT-ig. \Rightarrow innen előre minden itt szereplő T_i -re és minden később kezdődő más tranzakcióra REDO
- Ha előbb (START CHECKPOINT (T_1, \dots, T_k))-ot találunk \Rightarrow ezek nem mindegyike írta ki adatai (meg esetleg mások sem, amik még később kezdődtek)

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT (T_1, \dots, T_k)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van \Rightarrow elég visszamenni a következő START CHECKPOINT-ig. \Rightarrow innen előre minden itt szereplő T_i -re és minden később kezdődő más tranzakcióra REDO
- Ha előbb (START CHECKPOINT (T_1, \dots, T_k))-ot találunk \Rightarrow ezek nem mindegyike írta ki adatai (meg esetleg mások sem, amik még később kezdődtek) \Rightarrow elég visszamenni az előző (START CHECKPOINT)-hoz

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT (T_1, \dots, T_k)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van \Rightarrow elég visszamenni a következő START CHECKPOINT-ig. \Rightarrow innen előre minden itt szereplő T_i -re és minden később kezdődő más tranzakcióra REDO
- Ha előbb (START CHECKPOINT (T_1, \dots, T_k))-ot találunk \Rightarrow ezek nem mindegyike írta ki adatai (meg esetleg mások sem, amik még később kezdődtek) \Rightarrow elég visszamenni az előző (START CHECKPOINT)-hoz \Rightarrow onnan előre REDO

Előnyök, hátrányok

A CHECKPOINT ütemezése:

- adott idő letelte után

Előnyök, hátrányok

A CHECKPOINT ütemezése:

- adott idő letelte után
- adott lefutott tranzakció után

Előnyök, hátrányok

A CHECKPOINT ütemezése:

- adott idő letelte után
- adott lefutott tranzakció után

Ha ritkák a rendszerhibák, elég ritka CHECKPOINT.

UNDO/REDO protokoll-naplózás

- **UNDO hátránya:** COMMIT után mihamarabb írjunk

UNDO/REDO protokoll-naplózás

- UNDO hátránya: COMMIT után mihamarabb írjunk \Rightarrow sok írás

UNDO/REDO protokoll-naplózás

- **UNDO hátránya:** COMMIT után mihamarabb írjunk \Rightarrow sok írás
- **REDO hátránya:** Nem írunk, amíg nincs COMMIT

UNDO/REDO protokoll-naplózás

- **UNDO hátránya:** COMMIT után mihamarabb írjunk \Rightarrow sok írás
- **REDO hátránya:** Nem írunk, amíg nincs COMMIT \Rightarrow nagy memóriaigény

UNDO/REDO protokoll-naplózás

- **UNDO hátránya:** COMMIT után mihamarabb írjunk \Rightarrow sok írás
- **REDO hátránya:** Nem írunk, amíg nincs COMMIT \Rightarrow nagy memóriaigény

UNDO/REDO

Fő elv:

- Mielőtt az adatbázis bármely X elemének értékét a lemezen módosítanánk, a (T, X, v, w) naplóbejegyzésnek a lemezre kell kerülnie.

UNDO/REDO protokoll-naplózás

- **UNDO hátránya:** COMMIT után mihamarabb írjunk \Rightarrow sok írás
- **REDO hátránya:** Nem írunk, amíg nincs COMMIT \Rightarrow nagy memóriaigény

UNDO/REDO

Fő elv:

- Mielőtt az adatbázis bármely X elemének értékét a lemezen módosítanánk, a (T, X, v, w) naplóbejegyzésnek a lemezre kell kerülnie.

Nagyobb szabadság, hogy mikor írjunk.

UNDO/REDO protokoll-naplózás

- **UNDO hátránya:** COMMIT után mihamarabb írjunk \Rightarrow sok írás
- **REDO hátránya:** Nem írunk, amíg nincs COMMIT \Rightarrow nagy memóriaigény

UNDO/REDO

Fő elv:

- Mielőtt az adatbázis bármely X elemének értékét a lemezen módosítanánk, a (T, X, v, w) naplóbejegyzésnek a lemezre kell kerülnie.

Nagyobb szabadság, hogy mikor írjunk.

Nagyobb méretű napló. $\Rightarrow v, w$ nagyon nagy is lehet!

Példa

T	t	A_M	B_M	A_D	B_D	Napló
				8	8	(T , BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	($T, A, 8, 16$)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	($T, B, 8, 16$)
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
						(T , COMMIT)
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						

UNDO/REDO visszaállítás

- A legkorábbtól kezdve állítsuk vissza minden befejezett tranzakció hatását. (REDO)

UNDO/REDO visszaállítás

- A legkorábbtól kezdve állítsuk vissza minden befejezett tranzakció hatását. (REDO)
- A legutolsótól kezdve állítsuk tegyük semmissé minden be nem fejezett tranzakció hatását. (UNDO)

Példa

T	t	A_M	B_M	A_D	B_D	Napló
				8	8	(T , BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	($T, A, 8, 16$)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	($T, B, 8, 16$)
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
						(T , COMMIT)
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						

CHECKPOINT képzés működés közben

1. Írjuk a naplóba a **(START CHECKPOINT (T_1, \dots, T_k))** bejegyzést, ahol T_i az összes éppen aktív tranzakció
2. **(FLUSH LOG)**

CHECKPOINT képzés működés közben

1. Írjuk a naplóba a **(START CHECKPOINT (T_1, \dots, T_k))** bejegyzést, ahol T_i az összes éppen aktív tranzakció
2. **(FLUSH LOG)**
3. Írjuk a lemezre az **összes piszkos puffert**

CHECKPOINT képzés működés közben

1. Írjuk a naplóba a **(START CHECKPOINT (T_1, \dots, T_k))** bejegyzést, ahol T_i az összes éppen aktív tranzakció
2. **(FULSH LOG)**
3. Írjuk a lemezre az **összes piszkos puffert**
4. **(END CHECKPOINT)**
5. **(FULSH LOG)**

CHECKPOINT képzés működés közben

1. Írjuk a naplóba a **(START CHECKPOINT (T_1, \dots, T_k))** bejegyzést, ahol T_i az összes éppen aktív tranzakció
2. **(FULSH LOG)**
3. Írjuk a lemezre az **összes piszkos puffert**
4. **(END CHECKPOINT)**
5. **(FULSH LOG)**

Mindenképp elég visszamenni legfeljebb az előző CHECKPOINT-ig (mint a REDO-nál).

Példa

(T_1, BEGIN)

$(T_1, A, 4, 5)$

(T_2, BEGIN)

(T_1, COMMIT)

$(T_2, B, 9, 10)$

$(\text{START CHECKPOINT } (T_2))$

$(T_2, C, 14, 15)$

(T_3, BEGIN)

$(T_3, D, 19, 20)$

(END CHECKPOINT)

(T_2, COMMIT)

(T_3, COMMIT)

Védekezés lemezhiba ellen

- **A naplót külön lemezen tartjuk**
- **Nem dobjuk el a napló CHECKPOINT előtti részét sem**
- **REDO vagy UNDO/REDO protokollt használunk**

Védekezés lemezhiba ellen

- A naplót külön lemezen tartjuk
- Nem dobjuk el a napló CHECKPOINT előtti részét sem
- REDO vagy UNDO/REDO protokollt használunk

Így elvileg a kezdeti adatbázis ismeretében vissza tudjuk állítani a legutolsó állapotot.

Védekezés lemezhiba ellen

- A naplót külön lemezen tartjuk
- Nem dobjuk el a napló CHECKPOINT előtti részét sem
- REDO vagy UNDO/REDO protokollt használunk

Így elvileg a kezdeti adatbázis ismeretében vissza tudjuk állítani a legutolsó állapotot.

De a napló egy idő után nagyobb lesz, mint az adatbázis.

⇒ Időnként archiválunk

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Ha nem lehet leállítani \Rightarrow

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Ha nem lehet leállítani \Rightarrow

1. (START DUMP) a naplóba

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Ha nem lehet leállítani \Rightarrow

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Ha nem lehet leállítani \Rightarrow

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Ha nem lehet leállítani \Rightarrow

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Ha nem lehet leállítani \Rightarrow

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése
5. (END DUMP)

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Ha nem lehet leállítani \Rightarrow

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése
5. (END DUMP)

Helyreállítás

1. Megkeressük a legutolsó teljes mentést (volt (END DUMP))

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Ha nem lehet leállítani \Rightarrow

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése
5. (END DUMP)

Helyreállítás

1. Megkeressük a legutolsó teljes mentést (volt (END DUMP))
2. Módosítjuk az adatbázist a napló segítségével a CHECKPOINT-tól kezdve (ezért kell REDO vagy UNDO/REDO)

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Ha nem lehet leállítani \Rightarrow

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése
5. (END DUMP)

Helyreállítás

1. Megkeressük a legutolsó teljes mentést (volt (END DUMP))
2. Módosítjuk az adatbázist a napló segítségével a CHECKPOINT-tól kezdve (ezért kell REDO vagy UNDO/REDO)

Osztott adatbázisok

- Adatok vízszintes felosztása

Osztott adatbázisok

- Adatok vízszintes felosztása
 - ★ Egy bank több fiókja, saját ügyfelek

Osztott adatbázisok

- Adatok vízszintes felosztása
 - ★ Egy bank több fiókja, saját ügyfelek
 - ★ Üzlethálózat boltjai, saját eladások

Osztott adatbázisok

- Adatok vízszintes felosztása
 - ★ Egy bank több fiókja, saját ügyfelek
 - ★ Üzlethálózat boltjai, saját eladások
 - ★ Könyvár több fiókkal, saját katalógussal

Osztott adatbázisok

- Adatok vízszintes felosztása
 - ★ Egy bank több fiókja, saját ügyfelek
 - ★ Üzlethálózat boltjai, saját eladások
 - ★ Könyvár több fiókkal, saját katalógussal
- Adatok függőleges felbontása
 - ★ Bankban \Rightarrow ügyfél adatok helyben, hitelkártya adatok a központban

Osztott adatbázisok

- Adatok vízszintes felosztása
 - ★ Egy bank több fiókja, saját ügyfelek
 - ★ Üzlethálózat boltjai, saját eladások
 - ★ Könyvár több fiókkal, saját katalógussal
- Adatok függőleges felbontása
 - ★ Bankban \Rightarrow ügyfél adatok helyben, hitelkártya adatok a központban
 - ★ Üzletláncban \Rightarrow eladások helyben, megrendelések a központban

Osztott adatbázisok

- Adatok vízszintes felosztása
 - ★ Egy bank több fiókja, saját ügyfelek
 - ★ Üzlethálózat boltjai, saját eladások
 - ★ Könyvár több fiókkal, saját katalógussal
- Adatok függőleges felbontása
 - ★ Bankban \Rightarrow ügyfél adatok helyben, hitelkártya adatok a központban
 - ★ Üzletláncban \Rightarrow eladások helyben, megrendelések a központban
- Adatok többszörözése
 - ★ Párhuzamosítás miatt

Osztott adatbázisok

- Adatok vízszintes felosztása
 - ★ Egy bank több fiókja, saját ügyfelek
 - ★ Üzlethálózat boltjai, saját eladások
 - ★ Könyvár több fiókkal, saját katalógussal
- Adatok függőleges felbontása
 - ★ Bankban \Rightarrow ügyfél adatok helyben, hitelkártya adatok a központban
 - ★ Üzletláncban \Rightarrow eladások helyben, megrendelések a központban
- Adatok többszörözése
 - ★ Párhuzamosítás miatt
 - ★ Kommunikáció csökkentése miatt \Rightarrow gyakran szükséges adatok mindenhol (címjegyzék, telefonkönyv, chase-elés)

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.

Mikor lesz kész az egész tranzakció?

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.

Mikor lesz kész az egész tranzakció? \Rightarrow ha minden része kész

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.

Mikor lesz kész az egész tranzakció? \Rightarrow ha minden része kész

Hogyan vesszük észre? Mi van ha közben ABORT vagy hiba van?

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.

Mikor lesz kész az egész tranzakció? \Rightarrow ha minden része kész

Hogyan vesszük észre? Mi van ha közben ABORT vagy hiba van?

Példa

Áruházlánc központja lekérdezi minden boltban a mobiltelefon készletet. Ha valahol túl sok van, átküld belőle oda, ahol kevés van.

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.

Mikor lesz kész az egész tranzakció? \Rightarrow ha minden része kész

Hogyan vesszük észre? Mi van ha közben ABORT vagy hiba van?

Példa

Áruházlánc központja lekérdezi minden boltban a mobiltelefon készletet. Ha valahol túl sok van, átküld belőle oda, ahol kevés van.

\Rightarrow megszakadhat a kapcsolat menet közben, rossz az algoritmus, stb.

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T** , Felkészül)

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T** , Felkészül)
 - ★ Ezt mindenhova elküldi (még magának is)

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT **lesz majd**

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT **lesz majd**
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba
A koordinátornak elküldeni (**T, ABORT-Legyen**)-t

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba
A koordinátornak elküldeni (**T, ABORT-Legyen**)-t
 - ★ Második fázis
 - * Ha a koordinátor a (**T, Készenáll**)-t megkapta mindenkitől \Rightarrow

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba
A koordinátornak elküldeni (**T, ABORT-Legyen**)-t
 - ★ Második fázis
 - * Ha a koordinátor a (**T, Készenáll**)-t megkapta mindenkitől \Rightarrow
 - (**T, COMMIT**) a saját naplójába

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba
A koordinátornak elküldeni (**T, ABORT-Legyen**)-t
 - ★ Második fázis
 - * Ha a koordinátor a (**T, Készenáll**)-t megkapta mindenkitől \Rightarrow
 - (**T, COMMIT**) a saját naplójába
 - Mindenhova elküldi a (**T, COMMIT-Lesz**)-t

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba
A koordinátornak elküldeni (**T, ABORT-Legyen**)-t
 - ★ Második fázis
 - * Ha a koordinátor a (**T, Készenáll**)-t megkapta mindenkitől \Rightarrow
 - (**T, COMMIT**) a saját naplójába
 - Mindenhova elküldi a (**T, COMMIT-Lesz**)-t
 - * Ha a koordinátor a (**T, ABORT-Legyen**)-t kapta legalább egy állomástól

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba
A koordinátornak elküldeni (**T, ABORT-Legyen**)-t
 - ★ Második fázis
 - * Ha a koordinátor a (**T, Készenáll**)-t megkapta mindenkitől \Rightarrow
 - (**T, COMMIT**) a saját naplójába
 - Mindenhova elküldi a (**T, COMMIT-Lesz**)-t
 - * Ha a koordinátor a (**T, ABORT-Legyen**)-t kapta legalább egy állomástól
 - (**T, ABORT**) a saját naplójába

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba
A koordinátornak elküldeni (**T, ABORT-Legyen**)-t
 - ★ Második fázis
 - * Ha a koordinátor a (**T, Készenáll**)-t megkapta mindenkitől \Rightarrow
 - (**T, COMMIT**) a saját naplójába
 - Mindenhova elküldi a (**T, COMMIT-Lesz**)-t
 - * Ha a koordinátor a (**T, ABORT-Legyen**)-t kapta legalább egy állomástól
 - (**T, ABORT**) a saját naplójába
 - Mindenhova elküldi a (**T, ABORT-Lesz**)-t

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba
A koordinátornak elküldeni (**T, ABORT-Legyen**)-t
 - ★ Második fázis
 - * Ha a koordinátor a (**T, Készenáll**)-t megkapta mindenkitől \Rightarrow
 - (**T, COMMIT**) a saját naplójába
 - Mindenhova elküldi a (**T, COMMIT-Lesz**)-t
 - * Ha a koordinátor a (**T, ABORT-Legyen**)-t kapta legalább egy állomástól
 - (**T, ABORT**) a saját naplójába
 - Mindenhova elküldi a (**T, ABORT-Lesz**)-t
 - * Ha egy állomás a (**T, COMMIT-Lesz**)-t kapja \Rightarrow (**T, COMMIT**)

Kétfázisú véglegesítés (2PC)

- Alapelvek
 - ★ Minden állomás naplózza saját eseményeit
 - ★ Van egy koordinátor állomás, aki a döntést hozza majd
 - ★ Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
 - ★ A koordinátor saját naplójába (**T, Felkészül**)
 - ★ Ezt mindenhova elküldi (még magának is)
 - ★ Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
 - * Ha COMMIT várható (már csak ez lenne hátra) \Rightarrow (**T, Készenáll**) a saját naplóba
A koordinátornak elküldeni (**T, Készenáll**)-t
 - * Ha ABORT várható \Rightarrow (**T, ABORT-Legyen**) a saját naplóba
A koordinátornak elküldeni (**T, ABORT-Legyen**)-t
 - ★ Második fázis
 - * Ha a koordinátor a (**T, Készenáll**)-t megkapta mindenkitől \Rightarrow
 - (**T, COMMIT**) a saját naplójába
 - Mindenhova elküldi a (**T, COMMIT-Lesz**)-t
 - * Ha a koordinátor a (**T, ABORT-Legyen**)-t kapta legalább egy állomástól
 - (**T, ABORT**) a saját naplójába
 - Mindenhova elküldi a (**T, ABORT-Lesz**)-t
 - * Ha egy állomás a (**T, COMMIT-Lesz**)-t kapja \Rightarrow (**T, COMMIT**)
 - * Ha egy állomás a (**T, ABORT-Lesz**)-t kapja \Rightarrow (**T, ABORT**)

Helyreállítás

Egy adott állomáson:

- ★ Ha az utolsó bejegyzés **COMMIT, ABORT, COMMIT-Lesz** vagy **ABORT-Lesz** akkor a napló szerint helyreállítunk

Helyreállítás

Egy adott állomáson:

- ★ Ha az utolsó bejegyzés **COMMIT, ABORT, COMMIT-Lesz** vagy **ABORT-Lesz** akkor a napló szerint helyreállítunk
- ★ Ha az utolsó bejegyzés **Készenáll**, akkor nem világos a helyzet, vagy várunk, vagy kommunikálunk a többivel, vagy ...

Helyreállítás

Egy adott állomáson:

- ★ Ha az utolsó bejegyzés **COMMIT, ABORT, COMMIT-Lesz** vagy **ABORT-Lesz** akkor a napló szerint helyreállítunk
- ★ Ha az utolsó bejegyzés **Készenáll**, akkor nem világos a helyzet, vagy várunk, vagy kommunikálunk a többivel, vagy ...
- ★ Ha nincs semmilyen bejegyzés, akkor **ABORT** (vagy várunk)

Osztott zárolás

Ha nincs adattöbbszörözés \Rightarrow ✓

Osztott zárolás

Ha nincs adattöbbszörözés \Rightarrow \checkmark

Ha van \Rightarrow összhangban kell tartani a példányokat

Osztott zárolás

Ha nincs adattöbbszörözés \Rightarrow \checkmark

Ha van \Rightarrow összhangban kell tartani a példányokat \Rightarrow globális (logikai) LOCK és lokális LOCK

Osztott zárolás

Ha nincs adattöbbszörözés \Rightarrow \checkmark

Ha van \Rightarrow összhangban kell tartani a példányokat \Rightarrow globális (logikai) LOCK és lokális LOCK

Egyszerű modell

Minden LOCK globális és az egyik állomás nyilvántartja ezeket

Osztott zárolás

Ha nincs adattöbbszörözés \Rightarrow \checkmark

Ha van \Rightarrow összhangban kell tartani a példányokat \Rightarrow globális (logikai) LOCK és lokális LOCK

Egyszerű modell

Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás

Osztott zárolás

Ha nincs adattöbbszörözés \Rightarrow \checkmark

Ha van \Rightarrow összhangban kell tartani a példányokat \Rightarrow globális (logikai) LOCK és lokális LOCK

Egyszerű modell

Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás

Költség: egy LOCK-hoz 3 üzenet \Rightarrow igénylés, engedélyezés, feloldás

Osztott zárolás

Ha nincs adattöbbszörözés \Rightarrow \checkmark

Ha van \Rightarrow összhangban kell tartani a példányokat \Rightarrow globális (logikai) LOCK és lokális LOCK

Egyszerű modell

Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás

Költség: egy LOCK-hoz 3 üzenet \Rightarrow igénylés, engedélyezés, feloldás

\Rightarrow a zárállomás nagyon leterhelt lehet

Osztott zárolás

Ha nincs adattöbbszörözés \Rightarrow \checkmark

Ha van \Rightarrow összhangban kell tartani a példányokat \Rightarrow globális (logikai) LOCK és lokális LOCK

Egyszerű modell

Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás

Költség: egy LOCK-hoz 3 üzenet \Rightarrow igénylés, engedélyezés, feloldás

\Rightarrow a zárállomás nagyon leterhelt lehet

Elsődleges példány

Van egy elsődleges példány, ha valaki zárolni akar valamit, akkor az elsődleges példányt tároló állomáshoz fordul.

Osztott zárolás

Ha nincs adattöbbszörözés \Rightarrow \checkmark

Ha van \Rightarrow összhangban kell tartani a példányokat \Rightarrow globális (logikai) LOCK és lokális LOCK

Egyszerű modell

Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás

Költség: egy LOCK-hoz 3 üzenet \Rightarrow igénylés, engedélyezés, feloldás

\Rightarrow a zárállomás nagyon leterhelt lehet

Elsődleges példány

Van egy elsődleges példány, ha valaki zárolni akar valamit, akkor az elsődleges példányt tároló állomáshoz fordul.

Költség: mint előbb, de nem koncentrált forgalom

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK A-ja

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK A-ja
- Ha egy tranzakciónak van globális WLOCK A-ja, akkor egy másiknak nem lehet globális RLOCK A-ja

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK A-ja
- Ha egy tranzakciónak van globális WLOCK A-ja, akkor egy másiknak nem lehet globális RLOCK A-ja
- Lehet több tranzakciónak globális RLOCK A-ja

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK A-ja
- Ha egy tranzakciónak van globális WLOCK A-ja, akkor egy másiknak nem lehet globális RLOCK A-ja
- Lehet több tranzakciónak globális RLOCK A-ja
- Minden állomás az érvényes globális lock-ok figyelembevételével működik

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK A-ja
- Ha egy tranzakciónak van globális WLOCK A-ja, akkor egy másiknak nem lehet globális RLOCK A-ja
- Lehet több tranzakciónak globális RLOCK A-ja
- Minden állomás az érvényes globális lock-ok figyelembevételével működik

Hogyan lehet megszerezni egy globális RLOCK-ot vagy WLOCK-ot?

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK A-ja
- Ha egy tranzakciónak van globális WLOCK A-ja, akkor egy másiknak nem lehet globális RLOCK A-ja
- Lehet több tranzakciónak globális RLOCK A-ja
- Minden állomás az érvényes globális lock-ok figyelembevételével működik

Hogyan lehet megszerezni egy globális RLOCK-ot vagy WLOCK-ot? \Rightarrow többféle modell

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK A-ja
- Ha egy tranzakciónak van globális WLOCK A-ja, akkor egy másiknak nem lehet globális RLOCK A-ja
- Lehet több tranzakciónak globális RLOCK A-ja
- Minden állomás az érvényes globális lock-ok figyelembevételével működik

Hogyan lehet megszerezni egy globális RLOCK-ot vagy WLOCK-ot? \Rightarrow többféle modell

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
- Globális RLOCK A megszerzése:
 - ★ Ha az i állomás akar egy RLOCK A_i -t, nem kell üzeni, megnézzük milyen zár van A_i -n
 - ★ Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i , akkor engedélyezi

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
- Globális RLOCK A megszerzése:
 - ★ Ha az i állomás akar egy RLOCK A_i -t, nem kell üzenni, megnézzük milyen zár van A_i -n
 - ★ Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i , akkor engedélyezi
 - ★ Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i -t \Rightarrow globális RLOCK A

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
- Globális RLOCK A megszerzése:
 - ★ Ha az i állomás akar egy RLOCK A_i -t, nem kell üzeni, megnézzük milyen zár van A_i -n
 - ★ Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i , akkor engedélyezi
 - ★ Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i -t \Rightarrow globális RLOCK A
- Globális WLOCK A megszerzése
 - ★ Ha az i állomás akar egy WLOCK A_i -t, akkor üzen minden másik helyre ahol van A_j

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
- Globális RLOCK A megszerzése:
 - ★ Ha az i állomás akar egy RLOCK A_i -t, nem kell üzeni, megnézzük milyen zár van A_i -n
 - ★ Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i , akkor engedélyezi
 - ★ Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i -t \Rightarrow globális RLOCK A
- Globális WLOCK A megszerzése
 - ★ Ha az i állomás akar egy WLOCK A_i -t, akkor üzen minden másik helyre ahol van A_j
 - ★ Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i

- Globális RLOCK A megszerzése:
 - ★ Ha az i állomás akar egy RLOCK A_i -t, nem kell üzenni, megnézzük milyen zár van A_i -n
 - ★ Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i , akkor engedélyezi
 - ★ Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i -t \Rightarrow globális RLOCK A
- Globális WLOCK A megszerzése
 - ★ Ha az i állomás akar egy WLOCK A_i -t, akkor üzen minden másik helyre ahol van A_j
 - ★ Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi
 - ★ Ha mindenholnan engedélyezés jött, az i állomás felteszi a WLOCK A_i -t, mindenhova üzen, hogy WLOCK A_j -t \Rightarrow globális WLOCK A

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
 - Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
 - Globális RLOCK A megszerzése:
 - ★ Ha az i állomás akar egy RLOCK A_i -t, nem kell üzeni, megnézzük milyen zár van A_i -n
 - ★ Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i , akkor engedélyezi
 - ★ Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i -t \Rightarrow globális RLOCK A
 - Globális WLOCK A megszerzése
 - ★ Ha az i állomás akar egy WLOCK A_i -t, akkor üzen minden másik helyre ahol van A_j
 - ★ Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi
 - ★ Ha mindenholnan engedélyezés jött, az i állomás felteszi a WLOCK A_i -t, mindenhova üzen, hogy WLOCK A_j -t \Rightarrow globális WLOCK A
- \Rightarrow Ha az egyik állomás kért és kapott WLOCK A -t, akkor másik nyilván nem kaphat később sem WLOCK A -t, sem RLOCK A -t

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
 - Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
 - Globális RLOCK A megszerzése:
 - ★ Ha az i állomás akar egy RLOCK A_i -t, nem kell üzenni, megnézzük milyen zár van A_i -n
 - ★ Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i , akkor engedélyezi
 - ★ Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i -t \Rightarrow globális RLOCK A
 - Globális WLOCK A megszerzése
 - ★ Ha az i állomás akar egy WLOCK A_i -t, akkor üzen minden másik helyre ahol van A_j
 - ★ Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi
 - ★ Ha mindenholnan engedélyezés jött, az i állomás felteszi a WLOCK A_i -t, mindenhova üzen, hogy WLOCK A_j -t \Rightarrow globális WLOCK A
- \Rightarrow Ha az egyik állomás kért és kapott WLOCK A -t, akkor másik nyilván nem kaphat később sem WLOCK A -t, sem RLOCK A -t
- \Rightarrow Ha az egyik állomás kért és kapott RLOCK A -t, akkor másik nyilván nem kaphat később WLOCK A -t, de kaphat RLOCK A -t

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
 - Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
 - Globális RLOCK A megszerzése:
 - ★ Ha az i állomás akar egy RLOCK A_i -t, nem kell üzeni, megnézzük milyen zár van A_i -n
 - ★ Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i , akkor engedélyezi
 - ★ Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i -t \Rightarrow globális RLOCK A
 - Globális WLOCK A megszerzése
 - ★ Ha az i állomás akar egy WLOCK A_i -t, akkor üzen minden másik helyre ahol van A_j
 - ★ Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi
 - ★ Ha mindenholnan engedélyezés jött, az i állomás felteszi a WLOCK A_i -t, mindenhova üzen, hogy WLOCK A_j -t \Rightarrow globális WLOCK A
- \Rightarrow Ha az egyik állomás kért és kapott WLOCK A -t, akkor másik nyilván nem kaphat később sem WLOCK A -t, sem RLOCK A -t
- \Rightarrow Ha az egyik állomás kért és kapott RLOCK A -t, akkor másik nyilván nem kaphat később WLOCK A -t, de kaphat RLOCK A -t

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i -k többségén
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen az A_i -k többségén

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i -k többségén
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen az A_i -k többségén

⇒ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i -k többségén
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen az A_i -k többségén

⇒ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.

Miért jó a többségi zárolás?

Két tranzakció nem tud egyszerre WLOCK A -t szerezni, mert mindkettőnek a példányok több, mint felére kellene WLOCK-ot kapnia

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i -k többségén
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen az A_i -k többségén

⇒ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.

Miért jó a többségi zárolás?

Két tranzakció nem tud egyszerre WLOCK A -t szerezni, mert mindkettőnek a példányok több, mint felére kellene WLOCK-ot kapnia ⇒ van olyan példány, amire mindkettő kapna

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i -k többségén
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen az A_i -k többségén

⇒ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.

Miért jó a többségi zárolás?

Két tranzakció nem tud egyszerre WLOCK A -t szerezni, mert mindkettőnek a példányok több, mint felére kellene WLOCK-ot kapnia ⇒ van olyan példány, amire mindkettő kapna

Hasonlóan nem lehet egy tranzakciónak WLOCK A -ja, egy másiknak RLOCK A -ja.

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i -k többségén
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen az A_i -k többségén

⇒ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.

Miért jó a többségi zárolás?

Két tranzakció nem tud egyszerre WLOCK A -t szerezni, mert mindkettőnek a példányok több, mint felére kellene WLOCK-ot kapnia ⇒ van olyan példány, amire mindkettő kapna

Hasonlóan nem lehet egy tranzakciónak WLOCK A -ja, egy másiknak RLOCK A -ja.

Viszont lehet két különböző tranzakciónak RLOCK A -ja, hiszen egy példányon is lehet ilyen.

k az n -ből protokoll

Közös általánosítás:

Legyen n , hogy hány példány van A -ból és legyen $n \geq k \geq \lceil (n + 1)/2 \rceil$

k az n -ből protokoll

Közös általánosítás:

Legyen n , hogy hány példány van A -ból és legyen $n \geq k \geq \lceil (n + 1)/2 \rceil$

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen legalább $n + 1 - k$ db A_i -n
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen legalább k db A_i -n

k az n -ből protokoll

Közös általánosítás:

Legyen n , hogy hány példány van A -ból és legyen $n \geq k \geq \lceil (n + 1)/2 \rceil$

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen legalább $n + 1 - k$ db A_i -n
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen legalább k db A_i -n

$k = n \implies$ WALL

$k = \lceil (n + 1)/2 \rceil \implies$ többségi zárolás

k az n -ből protokoll

Közös általánosítás:

Legyen n , hogy hány példány van A -ból és legyen $n \geq k \geq \lceil (n + 1)/2 \rceil$

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen legalább $n + 1 - k$ db A_i -n
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen legalább k db A_i -n

$k = n \implies$ WALL

$k = \lceil (n + 1)/2 \rceil \implies$ többségi zárolás

k választásával hangolható a költség.

Miért jó a protokoll?

k az n -ből protokoll

Közös általánosítás:

Legyen n , hogy hány példány van A -ból és legyen $n \geq k \geq \lceil (n + 1)/2 \rceil$

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen legalább $n + 1 - k$ db A_i -n
- Globális WLOCK A megszerzéséhez kell, hogy lokális WLOCK A_i legyen legalább k db A_i -n

$k = n \implies$ WALL

$k = \lceil (n + 1)/2 \rceil \implies$ többségi zárolás

k választásával hangolható a költség.

Miért jó a protokoll? \implies hasonlóan a többségi bizonyításhoz

Vége

Itt az anyag vége!