**Tranzakciók alaptevékenységei**

A tranzakció és az adatbázis kölcsönhatásának 3 színhelye van:

* Az adatbázis elemeit tartalmazó lemezblokkok
* A pufferkezelő által használt memóriaterület
* A tranzakció memóriaterülete

Ahhoz, hogy egy tranzakció beolvashasson egy adatbázis elemet, előbb a lemezblokkról a pufferkezelő memóriaterületére kell beolvasni. Ezt követően tudja a tranzakció a saját memóriaterületére beolvasni. Azt, hogy a puffer tartalma mikor íródik ki a lemezre, a pufferkezelő szabályozza. A területek közötti adatmozgást a következő alapműveletekkel tudjuk leírni:

1. INPUT(X): Az *X* adatbáziselemet tartalmazó lemezblokk másolása a memória pufferbe.
2. READ(X,t): Az *X* adatbáziselem bemásolása a tranzakció *t* lokális változójába. Ezután kapja meg a *t* lokális változó az *X* értékét.
3. WRITE(X,t): A *t* lokális változó tartalma az *X* adatbáziselem memóriapufferbeli tartalmába másolódik. Ezután másolódik át a *t* lokális változó értéke a pufferbeli *X*-be.
4. OUTPUT(X): Az *X* adatbáziselemet tartalmazó puffer kimásolása lemezre.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tevékenység | t | Mem A | Mem B | Lemez A | Lemez B |
| READ(A,t) | 18 | 18 |  | 18 | 18 |
| t := t\*2 | 16 | 18 |  | 18 | 18 |
| WRITE(A,t) | 16 | 16 |  | 18 | 18 |
| READ(B,t) | 18 | 16 | 18 | 18 | 18 |
| t := t\*2 | 16 | 16 | 18 | 18 | 18 |
| WRITE(B,t) | 16 | 16 | 16 | 18 | 18 |
| OUTPUT(A) | 16 | 16 | 16 | 16 | 18 |
| OUTPUT(B) | 16 | 16 | 16 | 16 | 16 |

**Általános tudnivalók:**

* A tranzakciók atomossági tulajdonsága a naplózás segítségével biztosítható (vagy végrehajtjuk, vagy nem)
* A napló nem más, mint naplóbejegyzések sorozata
* Ha hiba keletkezik, a napló segítségével rekonstruálni tudjuk, hogy a tranzakció mit tett a hiba fellépéséig.
* Három módszert fogunk erre megnézni:
  + Undo (semmiségi) naplózás – Ha egy tranzakció nem fejeződött be, akkor visszaállítjuk az adatbázist, a tranzakció előtti állapotba.
  + Redo naplózás – Ha egy tranzakció nem érkezett mindent megcsinálni, akkor az elejétől kezdve újra elvégezzük a lépéseit.
  + Undo-redo naplózás
* A naplót a naplókezelő írja és kizárólag írásra van megnyitva.
* A naplóblokkok elsődlegesen a memóriában vannak létrehozva, de amint lehet, fizikai tárolóra írja őket a rendszer.
* Naplóbejegyzések:
  + <START T> - a T tranzakció végrehajtása elkezdődött
  + <COMMIT T> - a T tranzakció végrehajtása befejeződött, már nem kell több adatbázis elemet írnia, viszont a lemezen még nem biztos, hogy megtörténtek a módosítások
  + <ABORT T> - a T tranzakció nem tudott sikeresen befejeződni.
  + <T, X, v> - módosítási bejegyzése, a T tranzakció az X adatbázis elemet módosította, a módosítás előtti értéke pedig v volt.

**Semmiségi naplózás**

Ahhoz, hogy semmiségi naplózást használhassunk a tranzakcióknak két szabályt kell kielégíteni. Ezek a pufferkezelő működésére vonatkoznak:

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

Összefoglalva a lemezre írási sorrend:

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

A naplóbejegyzések lemezre írásának a kikényszerítésére a naplózónak szüksége van a FLUSH LOG parancsra, amely felszólítja a puffer kezelőt a korábban még ki nem írt naplóblokkok lemezre írására.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Lépés | Tevékenység | t | Mem-A | Mem-B | Lemez-A | Lemez-B | Napló |
| 11) |  |  |  |  |  |  | <START *T*> |
| 12) | READ(A,t) | 18 | 18 |  | 18 | 18 |  |
| 13) | t := t\*2 | 16 | 18 |  | 18 | 18 |  |
| 14) | WRITE(A,t) | 16 | 16 |  | 18 | 18 | <*T*,*A*,8> |
| 15) | READ(B,t) | 18 | 16 | 18 | 18 | 18 |  |
| 16) | t := t\*2 | 16 | 16 | 18 | 18 | 18 |  |
| 17) | WRITE(B,t) | 16 | 16 | 16 | 18 | 18 | <*T*,*B*,8> |
| 18) | FLUSH LOG |  |  |  |  |  |  |
| 19) | OUTPUT(A) | 16 | 16 | 16 | 16 | 18 |  |
| 10) | OUTPUT(B) | 16 | 16 | 16 | 16 | 16 |  |
| 11) |  |  |  |  |  |  | <COMMIT *T*> |
| 12) | FLUSH LOG |  |  |  |  |  |  |

**Helyreállítás UNDO naplózással**

Tegyük fel, hogy hiba keletkezett a rendszerbe. Ha a tranzakciók csak egy része került a lemezre, akkor az adatbázis inkonzisztens állapotba kerülhet. A helyreállítás-kezelő feladata az adatbázist konzisztens állapotba visszaállítani.

Az első feladat eldönteni, hogy melyik tranzakciók fejeződtek be sikeresen és melyek nem. Amelyiknél COMMIT található a naplóban, arról tudjuk, hogy befejeződött, viszont ahol nincs COMMIT csak START, ott tudjuk, hogy nem. Ebben az esetben a tranzakciók hatását semmissé kell tenni. A naplót a végétől kell visszafelé vizsgálni. Ha talál egy módosító bejegyzést két eset lehetséges:

1. A tranzakcióhoz már volt korábban COMMIT, ekkor semmit nem kell tenni.
2. Ha a tranzakció abortált vagy nem teljes, akkor az X értékét v-re kell cserélni.

Ha ez kész, akkor a nem teljes tranzakciókhoz <ABORT T> naplóbejegyzést kell írni, aztán pedig kiváltani a napló lemezre írását (FLUSH LOG). Ezután folytatódhat az adatbázis normális működése.

**Ellenőrzőpont képzést**

Hogy ne kelljen mindig az egész naplót megvizsgálni, ellenőrzőpontot képzünk. Lépései:

1. Új tranzakció indítási kérések kiszolgálásának leállítása.
2. A még aktív tranzakciók helyes és teljes befejezésének vagy abortálásának és a COMMIT vagy az ABORT bejegyzés naplóba írásának kivárása.
3. A napló lemezre kiírása (FLUSH).
4. <CKPT> naplóbejegyzés képzése és kiírása a naplóba, és ismételt FLUSH.
5. Tranzakció indítási kérések kiszolgálása.

Az ellenőrzőpont segítségével a naplót csak addig kell vizsgálnunk, amíg megtaláljuk a CKPT bejegyzést.

**Működés közbeni ellenőrzőpont képzés**

Az előző módszer hátránya, hogy a képzése közben le kell állítani az új tranzakciók indítását. Ezt oldja meg a működés közbeni ellenőrzőpont. Lépései:

1. <START CKPT (*T*1, …, *Tk*)> naplóbejegyzés készítése és a naplóbejegyzés lemezre írása (FLUSH). *T*1, …, *Tk* az éppen *aktív* tranzakciók nevei.
2. Meg kell várni a *T*1, …, *Tk* tranzakciók mindegyikének normális vagy abnormális befejeződését, nem tiltva közben újabb tranzakciók indítását.
3. Ha a *T*1, …, *Tk* tranzakciók mindegyike befejeződött, akkor <END CKPT> naplóbejegyzés elkészítése és a naplóbejegyzés lemezre írása (FLUSH).

Ezzel a módszerrel két eset lehetséges.

1. Napló olvasása közben END CKPT-vel találkozunk. Ekkor biztosak lehetünk benne, hogy a következő START CKPT-ig elég visszamenni.
2. Ha olvasás közben STAR CKPT-vel találkozunk. Ekkor vissza kell menni a legkorábbi tranzakcióig, amely a felsorolásban benne van.

**Helyrehozó naplózás**

Különbségek a semmiségi naplózással:

* Figyelmen kívül hagyja a be nem fejezett tranzakciókat és megismétli a normálisan befejeződött tranzakciók módosításait.
* Elvárja a COMMIT lemezre írását még mielőtt az adatbáziselemeket a lemezre írná.
* A régi értékek helyett az új értékeket kell a naplóban megőriznünk.

A napló módosító bejegyzése ugyanúgy néz ki: <T, X, u>, jelentése viszont – a T tranzakció az X adatbázis elemet módosította, új értéke u. A helyrehozó naplózási szabály:

* R1: Mielőtt az adatbázis bármely *X* elemét a lemezen módosítanánk, szükséges, hogy az *X* ezen módosítására vonatkozó összes naplóbejegyzése, azaz <*T*,*X*,*v*> és <COMMIT *T*>, a lemezre kerüljenek.

A lemezre írások sorrendje:

1. Az adatbáziselemek módosítását leíró naplóbejegyzések lemezre írása.
2. A COMMIT naplóbejegyzés lemezre írása.
3. Az adatbáziselemek értékének tényleges cseréje a lemezen.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Lépés | Tevékenység | t | Mem-A | Mem-B | Lemez-A | Lemez-B | Napló |
| 11) |  |  |  |  |  |  | <START *T*> |
| 12) | READ(A,t) | 18 | 18 |  | 18 | 18 |  |
| 13) | t := t\*2 | 16 | 18 |  | 18 | 18 |  |
| 14) | WRITE(A,t) | 16 | 16 |  | 18 | 18 | <*T*,*A*,16> |
| 15) | READ(B,t) | 18 | 16 | 18 | 18 | 18 |  |
| 16) | t := t\*2 | 16 | 16 | 18 | 18 | 18 |  |
| 17) | WRITE(B,t) | 16 | 16 | 16 | 18 | 18 | <*T*,*B*,16> |
| 18) |  |  |  |  |  |  | <COMMIT *T*> |
| 19) | FLUSH LOG |  |  |  |  |  |  |
| 10) | OUTPUT(A) | 16 | 16 | 16 | 16 | 18 |  |
| 11) | OUTPUT(B) | 16 | 16 | 16 | 16 | 16 |  |

**Helyreállítás REDO naplózás esetén**

A helyreállítás lépései:

1. Meghatározni a befejezett (COMMIT) tranzakciókat.
2. Elemezni a naplót az elejétől kezdve. Minden <*T*,*X*,*v*> naplóbejegyzés megtalálásakor:
   1. Ha *T* nem befejezett tranzakció, akkor nem kell tenni semmit.
   2. Ha *T* befejezett tranzakció, akkor *v* értéket kell az *X* adatbáziselembe írni.
3. Minden *T* be nem fejezett tranzakcióra vonatkozóan <ABORT *T*> naplóbejegyzést kell a naplóba írni, és a naplót ki kell írni lemezre (FLUSH LOG).

**Ellenőrzőpont képzést REDO naplózás esetén**

1. <START CKPT (*T*1, …, *Tk*)> naplóbejegyzés elkészítése és kiírása lemezre, ahol  
*T*1, …, *Tk* az összes éppen aktív (még be nem fejezett) tranzakció.

2. Az összes olyan adatbáziselem kiírása lemezre, melyeket olyan tranzakciók írtak pufferekbe, melyek a START CKPT naplóba írásakor már befejeződtek, de puffereik lemezre még nem kerültek.

3. <END CKPT> bejegyzés naplóba írása és a napló lemezre írása (FLUSH LOG).

Ellenőrzőpont esetén a helyreállítás annyiban változik, hogyha END CKPT-t találtunk, akkor