Adatbázisrendszerek
9. előadás: Tranzakciók és konkurencia
Tranzakció kezelés és konkurencia kontroll

2020. április 21.

DEBRECENI
EGYETEM

- Egyfelhasználós rendszer. Egyidőben legfeljebb egy felhasználó használhatja a rendszert.
- Többfelhasználós rendszer. Egyidejűleg (konkurens módon) több felhasználó érheti el a rendszert.
- Konkurencia típusa
  - **Összefésült egyszálú feldolgozás.** A folyamatok (processzek) konkurens végrehajtása egy CPU-n fésülődik össze.
  - **Párhuzamos feldolgozás.** A folyamatok (processzek) egyidejűleg (konkurens módon) több CPU-n hajtódnak végre.

Tranzakció (heurisztikus)

Adatbázis folyamatok egy olyan logikai egysége, amely egy vagy több adatbázis-hozzáférési műveletet (olvasás - kinyerés, írás - beszúrás, frissítés, törlés) tartalmaz.

- Egy tranzakció (műveletek egy halmaza) lehet önálló, melyet egy magas szintű nyelv specifikál (SQL) és interaktív módon hajtunk végre, illetve lehet beágyazva egy programon belül.
- A tranzakció határai: `Begin transaction` és `End transaction`
- Egy alkalmazói program több, egymástól elkülönülő tranzakciót tartalmazhat, melyeket a fenti határok közé foglalunk be.

Az adatbázis egyszerű modellje

A tranzakció kezelés szempontjából az adatbázis egy egyszerű modelljét használjuk.

- Az adatbázis nevesített adatelemek egy összessége.
- Az adatok szemcsézettsége (granularitása): az adatok különböző méretű egységei - mező, rekord, egy teljes lemez blokk. (Az ismertetett fogalmak függetlenek a konkrét finomságtól.)
- Az alapműveletek az olvasás és az írás:
  - `read_item(X)`: beolvassa az `X` nevű adatbázis elemet egy program változóba. A jelölések egyszerűsítése kedvéért ezt a program változót szintén `X`-szel jelöljük.
  - `write_item(X)`: kiírja az `X` program változó értékét az adatbázis `X` nevű elemébe.

Az adatbázis egyszerű modellje

- Írási és olvasási műveletek jellemzői: az adatátvitel alapegysége a lemezről a számítógép fő memórájába illetve vissza a blokk. Általában egy beolvasandó vagy kiírandó adategység ettől kisebb, egy rekord egy mezője az adatbázisban, bár lehetnek olyan nagyobb egységek is, mint egy rekord vagy akár egy teljes blokk is.

Írási és olvasási műveletek

- A `read_item(X)` utasítás az alábbi lépéseket tartalmazza:
  - Megkeresi az `X` elemet tartalmazó lemezblokk címét.
  - Átmásolja ezt a lemezblokkot a fő memória pufferébe, (amennyiben ez a blokk nincs már benne valamelyik fő memória pufferben).
  - Átmásolja az `X` elemet a pufferből az `X` nevű program változóba.
- A `write_item(X)` utasítás az alábbi lépéseket tartalmazza:
  - Megkeresi az `X` elemet tartalmazó lemezblokk címét.
  - Átmásolja ezt a lemezblokkot a fő memória pufferébe, (amennyiben ez a blokk nincs már benne valamelyik fő memória pufferben).
  - Átmásolja az `X` elemet az `X` nevű program változóból a puffer megfelelő területére.
  - Visszamásolja a frissített blokkot a pufferből a lemezre (rögtön vagy egy későbbi időpontban).

Példa két tranzakcióra

T_1 tranzakció

```
read_item(X);
X:=X-N;
write_item(X);
read_item(Y);
Y:=Y+N;
write_item(Y);
```

T_2 tranzakció

```
read_item(X);
X:=X+M;
write_item(X);
```

Miért van szükség konkurencia kontrollra? 8

- Az elveszett frissítés problémája. Akkor fordul elő, amikor két tranzakció, amely ugyanazokat az adatbázis elemeket éri el, úgy fésülődik össze, hogy egyes adatbázis elemek hibásakká válnak.
- Az időleges frissítés (dirty read) problémája. Akkor fordul elő, amikor egy tranzakció frissít egy adatbázis elemet, ami után valamilyen oknál fogva a tranzakció hibásan fejeződik be. Ezt a frissített elemet más tranzakció is eléri mielőtt az még visszaállna az eredeti értékére.
- A helytelen összegzés problémája. Amikor egy tranzakció rekordok egy összegző függvényét számolja, amíg egy másik tranzakció ezen rekordok közül néhányat frissít. Ekkor az összegző függvény olyan értékekkel számolhat, amelyek még a frissítés előtt vannak, míg mások már a frissítés után.

Az elveszett frissítés problémája

Tegyük fel, hogy az alábbi módon fésülődnek össze a T_1 és T_2 tranzakciók.

T_1 tranzakció	T_2 tranzakció
`read_item(X);`	
`X:=X-N;`	
	`read_item(X);`
	`X:=X+M;`
`write_item(X);`	
`read_item(Y);`	
	`write_item(X);`
`Y:=Y+N;`	
`write_item(Y);`	

Az elveszett frissítés problémája

Ha a két tranzakciót egymás után hajtjuk végre, akármilyen sorrendben, akkor az X adatbázis elem új értéke $X - N + M$ lesz, az Y-é pedig $Y + N$. Ezzel szemben a fenti összefésülés esetén $X + M$-t fogja X tartalmazni hiszen T_2, a T_1 módosítása után de még mentés előtt, újra beolvassa X-t a régi értékkel. Így X T_1 általi frissítése $(-N)$ elveszik. (Foglalt helyek száma)

Az időleges frissítés problémája

Tegyük fel, hogy az alábbi módon fésülődnek össze a T_1 és T_2 tranzakciók.

T_1 tranzakció	T_2 tranzakció
`read_item(X);`	
`X:=X-N;`	
`write_item(X);`	
	`read_item(X);`
	`X:=X+M;`
	`write_item(X);`
`read_item(Y);`	
`Y:=Y+N;`	
`write_item(Y);`	

Az időleges frissítés problémája

Tegyük fel, hogy a fenti módon fésülődtek össze a T_1 és T_2 tranzakciók. Ha a rendszerben hiba történik a T_1 tranzakció teljes véglegesítése előtt de még az `X` adatbázis elem értékét sikerül elmenteni, akkor a T_2 tranzakció hibásan ezzel az értékkel fog dolgozni és nem az eredetivel. Ezért a végeredmény a rossz $X - N + M$ érték lesz a jó $X + M$ helyett, amit akkor kapnánk ha visszaállítanánk a T_1 tranzakció előtti állapotot. A T_2 végrehajtásánál az `X` adatbázis elem beolvasásakor ún. piszkos olvasás történik.

A helytelen összegzés problémája

T_1 tranzakció	T_2 tranzakció
	`sum:=0;`
	`read_item(A);`
	`sum:=sum+A;`
`read_item(X);`	
`X:=X-N;`	
`write_item(X);`	
	`read_item(X);`
	`sum:=sum+X;`
	`read_item(Y);`
	`sum:=sum+Y;`
`read_item(Y);`	
`Y:=Y+N;`	
`write_item(Y);`	

A T_2 tranzakció azután olvassa be X-et, hogy N-et már levontunk belőle, míg az előtt olvassa be Y-t, hogy N-et hozzáadtunk volna, az eredmény hibás összegzés lesz.

Miért van szükség helyreállításra? 15

A helyreállítás (recovery) lehetséges okai:

- Számítógép hiba (rendszerösszeomlás). Hardver vagy szoftver hiba fordul elő a tranzakció végrehajtásakor. Amennyiben a hardver sérül, úgy a számítógép memóriájában lévő információ elveszhet.
- Tranzakció vagy rendszer hiba. Bizonyos műveletek a tranzakcióban hibát eredményezhetnek, pl. nullával való osztás vagy egész túlcsordulás. Tranzakció hiba fordulhat még elő hibás paraméter értéknél programozói hiba esetén. Emellett a felhasználó is megszakíthatja a tranzakciót.
- Lokális hibát vagy kivételt észlel a tranzakció. Bizonyos esetekben szükség lehet a tranzakció törlésére, pl. az adatok nem állnak rendelkezésre, vagy nincs elégséges fedezet egy adott összeg lehívására egy banki rendszerben.

Miért van szükség helyreállításra? 16

A helyreállítás (recovery) lehetséges okai:

- Konkurencia kontrol kikényszerítés. A konkurencia kontrol dönthet a tranzakció megszakítása mellett és később újraindíthatja azt. Ennek oka lehet a szerializálhatóság követelménye (ld. később), vagy mert több tranzakció deadlock (holtpont) állapotban van.
- Lemezhiba. A lemez egyes blokkjai elvesztették az adataikat. Ennek oka lehet pl. az író-olvasó fej megsérülése, ami a tranzakció írása vagy olvasása kapcsán egyaránt előfordulhat.
- Fizikai problémák, katasztrófák. Végtelen listája olyan problémáknak, mint a légkondicionáló hibája, tűz, szabotázs, a lemez túlírása, vagy rossz lemez felmountolása az operátor által.

Definíció

A tranzakció egy végrehajtás alatt álló program, amely az adatbázis-feldolgozás egy logikai egységét alkotja. Egy tranzakció egy vagy több adatbázis-hozzáférési műveletből (beszúrás, törlés, módosítás és lekérdezés) áll.

A tranzakciót alkotó adatbázis-műveletek vagy egy alkalmazói programba vannak beágyazva, vagy interaktívan is megadhatók egy magas szintű lekérdező nyelv (például SQL) segítségével. A tranzakció határait megadhatjuk az explicit begin transaction és end transaction utasításokkal egy alkalmazói programban, ebben az esetben a két utasítás között elhelyezkedő összes adatbázis-hozzáférési művelet egy tranzakciót alkot.

Egy alkalmazói program egynél több tranzakciót is tartalmazhat, ha több tranzakcióelhatároló utasítás szerepel benne.
Ha a tranzakciót alkotó adatbázis-műveletek nem módosítják az adatbázist, csak lekérdezik azt, akkor a tranzakciót read-only tranzakciónak nevezzük.

A tranzakció állapotai és műveletei

Állapotok:

- aktív állapot
- részlegesen véglegesített állapot
- véglegesített (commit) állapot
- hibás állapot
- megszakított állapot

Műveletek:

- `begin_transaction`
- `read` vagy `write`
- `end_transaction`
- `commit_transaction`
- `rollback` vagy `abort`

A visszaállítás során az alábbi műveletek használhatóak

- `undo` hasonló a rollback-hez azt kivéve, hogy elemibb művelet minthogy egy egész tranzakcióra vonatkozna
- `redo` bizonyos tranzakció műveleteket újra végrehajt a biztonság kedvéért

A tranzakció végrehajtás folyamatábrája

ide jön egy ábra

A rendszer log

- A log vagy napló-fájl. A log nyomon követi az összes olyan tranzakció műveletet, amely hatással van az adatbázis elemeire.
- Ez az információ szükséges lehet a visszaállítás engedélyezésére hibás tranzakció esetén.
- A log a lemezen van, ezért immunis minden hibatípusra kivéve a lemezhibákat és katasztrófákat.
- A log-ot periódikusan archiválni kell (pl. szalagon) a fenti katasztrófák kijavítására.

- A log-rekordok típusai: `[start_transaction,T]`, `[write_item, T,old_value,new_value]`, `[read_item,T,X]`, `[commit,T]`, `[abort,T]`, ahol T egy egyértelmű tranzakció azonosító.
- Azok a helyreállító protokollok, amelyek elkerülik a kaszkádolt (lépcsős) rollback-et, nem igénylik a `read` műveletet a log-ban, míg mások igen. Erős protokollok egyszerűbb `write` bejegyzéseket igényelnek, amelyek nem tartalmazzák a `new_value`-t.

Helyreállítás a rendszer loggal

Ha a rendszer összeomlik, akkor helyreállíthatjuk konzisztens adatbázis állapotba a log vizsgálatával megfelelő módszerek révén.

- Mivel a log minden olyan írási műveletről tartalmaz egy rekordot, amely megváltoztatja valamelyik adatbázis elem értékét, az `undo` segítségével lehetséges egy tranzakció írási műveletei hatásának visszaállítása úgy, hogy az összes érintett adatbázis elem értékét az írási művelet log-bejegyzésebeli `old_value` értékére állítjuk vissza.
- Szintén használhatjuk a `redo`-t arra, hogy kikényszerítsük az írási műveletek hatását minden a tranzakcióban érintett adatbázis elem értékének a log-bejegyzésbeli megfelelő `new_value`-re való állításával.

Definíció

Egy tranzakció akkor éri el a véglegesítési (commit) pontját, ha az összes adatbázis-hozzáférési művelete sikeresen végrehajtódott és ezen műveletek hatása kiírásra került a log-fájlba.

A véglegesítési pontja után a tranzakciót véglegesítettnek nevezzük és feltételezzük, hogy összes hatása állandó bejegyzésre került az adatbázisban. A tranzakció ezután egy `[commit,T]` bejegyzést tesz a log-ba.

Tranzakciók visszavonása (rollback): azon tranzakcióknál szükséges, amelyeknél van `[start_transaction,T]` bejegyzés a log-ban, azonban nincs `[commit,T]` bejegyzés.

ACID tulajdonságok

Atomosság (atomicity): A tranzakció a feldolgozás atomi egysége; vagy teljes egészében végrehajtódik, vagy egyáltalán nem.

Konzisztenciamegőrzés (consistency preservation): Egy tranzakció konzisztenciamegőrző, ha teljes és önálló végrehajtása az adatbázist konzisztens állapotból konzisztens állapotba viszi át.

Elkülönítés (isolation): Egy tranzakciónak látszólag más tranzakcióktól elkülönítve kell végrehajtódnia. Ez azt jelenti, hogy a tranzakció végrehajtása nem állhat kölcsönhatásban semelyik másik konkurensen végrehajtott tranzakcióval sem.

Tartósság vagy állandóság (durability vagy permanency): Egy véglegesített tranzakció által az adatbázison véghezvitt módosításoknak meg kell őrződniük az adatbázisban. Ezeknek a módosításoknak semmilyen hiba miatt nem szabad elveszniük.

Az ACID tulajdonságok biztosításáért a DBMS konkurenciavezérlő és naplózó/helyreállító alrendszerei a felelősek.

Tranzakció ütemezés (schedule vagy history)

A különböző tranzakciókban lévő műveletek sorrendje, amikor a tranzakciókat összefésülve egy szálon hajtjuk végre.

Az ütemezés a tranzakció műveletek egy olyan sorrendje, amely megfelel az egyenkénti tranzakcióbeli sorrendnek, azaz ha két, egyazon tranzakcióbeli műveletnél az egyik megelőzi a másikat, úgy ez a sorrend megmarad az összefésülés után is.

Típusok:

- Visszaállítható ütemezés. Amikor egyetlen egy olyan T tranzakció sem véglegesítődik addig, amíg nem véglegesítődik minden olyan T' tranzakció, amely olyan elemet ír ki, amelyet T beolvas. Ekkor egyetlen tranzakciót sem szükséges visszaállítani.
- Kaszkádmentes ütemezés. Amikor minden tranzakció csak olyan adatbázis elemet olvas be, amelyet egy már elfogadott tranzakció írt ki.

Szeriális ütemezés

Egy S ütemezést szeriálisnak nevezünk, ha minden ütemezésbeli T tranzakcióra fennáll, hogy az összes T-beli művelet közvetlenül egymás után hajtódik végre az ütemezésben.

Szerializálható ütemezés

Egy S ütemezés szerializálható, ha ekvivalens ugyanazon tarnzakciók egy szeriális ütemezésével.

Fajtái:

- Eredmény ekvivalens ütemezés. Amikor a két ütemezés ugyanazt a végső adatbázis állapotot eredményezi.
- Konfliktus ekvivalens ütemezés. Amikor a két ütemezésben bármely két konfliktusos művelet (pl. ugyanazt az adatbázis elemet akarja felülírni két tranzakció) sorrendje ugyanaz.
- Konfliktus szerializálható ütemezés. Amikor az ütemezés konfliktus ekvivalens egy szeriális ütemezéssel.

Megjegyzések a szerializálhatóságról

1. A szerializálhatóság nem jelenti azt, hogy az ütemezés maga szeriális.
2. A szerializálhatóságból következik, hogy az ütemezés helyes. Ez azt jelenti, hogy az adatbázis konzisztens állapotban marad a tranzakciók végrehajtása után.
3. A szerializálhatóságot nehéz ellenőrizni, mivel nem könnyű előre meghatározni, hogy egy ütemező hogyan fésüli össze a műveleteket.
4. A gyakorlatban protokollokat használnak a szerializálhatóság biztosítására.
5. Egy ütemezés kezdete és vége nem meghatározható, ezért a teljes ütemezés ellenőrzését a véglegesített tranzakciókbeli műveletek ellenőrzésére redukálják.

Megjegyzések a szerializálhatóságról

6. A gyakorlatban a legtöbb DBMS-ben a kétfázisú zárolást (two phase locking) használják.
7. Egy gyengébb ekvivalencia ütemezések között az ún. nézet (view) ekvivalencia, melyet itt nem tárgyalunk.
8. Van algoritmus a konfliktus szerializálhatóság ellenőrzésére, amely a precedencia gráfon alapszik.

Célja

- Az elkülönítés kikényszerítése (pl. teljes kizárással) a konfliktusos tranzakciók között.
- Az adatbázis konzisztenciájának megőrzése a tranzakciók konzisztencia megőrző végrehajtása révén.
- Az olvasás-írás és írás-írás konfliktusok feloldása.

Példa

Ha egy konkurens végrehajtási környezetben a T_1 konfliktusba kerül a T_2-vel az A adatelem miatt, akkor a létező konkurencia kontroll dönt arról, ha T_1-nek vagy T_2-nek szüksége van A-ra, illetve ha más tranzakciókat vissza kell állítani vagy várakoztatni kell.

A kétfázisú zárolás módszere

- Két művelet: `Lock(X)` és `Unlock(X)` az X adatbázis elemen.
- A zárolás (`Lock`) művelet biztosítja az engedélyt (a) olvasásra, (b) egy adatelem írására egy tranzakció által.
- A feloldás (`Unlock`) művelet törli ezeket az engedélyeket az adatelemről.
- A két művelet atomi művelet.
- Két zárolási mód: (a) megosztás (`shared lock`) és (b) kizárás (`write lock`). Megosztás esetén egynél több ilyet jegyezhetünk be egy adatbázis elemre olvasás céljából, azonban ekkor kizárást már nem jegyezhetünk be semmilyen tranzakció által. Csak egy kizárás jegyezhető be egy adatbázis elemre egy időben és ekkor egyetlen tranzakció sem jegyezhet be megosztást erre az adatbázis elemre.

A kétfázisú zárolás módszere

- A Lock Manager menedzseli a zárolásokat az adatelemeken egy zárolási tábla (lock table) segítségével tartva ezeket nyilván.