



Transaktionhallinta - samanaikaisuus

- Tietokannalla on tyypillisesti useita samanaikaisia käyttäjiä (prosesseja).
- On toivottavaa, että
 - yhdenkään käyttäjän toiminta ei hidastuisi kohtuuttomasti, vaikka muita käyttäjiä olisi runsaastikin
 - yhdenkään käyttäjän toiminta ei ainakaan saisi estyä kokonaan
- tietokanta säilyy eheänä eli että jokaisen transaktion toiminnot pysyvät periaatteessa loogisesti erillään muiden transaktioiden toiminnoista (eristyyvyys; välttämätön ominaisuus)

1



Transaktionhallinta - samanaikaisuus

- Esim. nosto pankkitililtä:
- sama proseduuri tilinosto(X, summa) :
 - (X = tietyn tilin saldo tili-relaatiossa, alussa esim. 2000)

T1
read(X, v)
v:=v-500;
write(X, v)

T2
read(X, u)
u:=u-1000;
write(X, u)

jos T1 ja T2 alkavat suunnilleen samaan aikaan ja etenevät vuorotellen 'huonossa kontrollissa', voi tilin X saldo olla lopussa 1500, 1000 tai 500 (oikea)

2



Transaktionhallinta - samanaikaisuus

- Kontrolloitu suoritus olisi **sarjallinen suoritus** (serial schedule), jossa joko T1 suoritetaan kokonaan ennen kuin T2 aloitetaan tai päinvastoin.
 - Tällöin lopputulos molempien jälkeen on 500, mutta välitulokset on joko 1000 tai 1500.
- Yleisesti sarjallinen suoritus aiheuttaa joidenkin transaktioiden huomattavaa viivästymistä ja saattaa jopa estää suorituksen (edellinen epäonnistuu, joten seuraavaa ei voida aloittaa)

3



Transaktionhallinta - samanaikaisuus

- Samanaikaisuuden hallinnan alijärjestelmän tehtävänä on **lieventää hallitusti** sarjallisuuden vaatimusta eli sallia rinnakkaisuutta, mutta eliminoida sen haitat.
- Aiemmin määriteltiin transaktioiden T1, ..., Tn **ajojärjestys (schedule, history)** näiden operaatioista (luku/kirjoitus) muodostuvaksi jonoksi, jossa kunkin transaktion Ti operaatiot ovat samassa järjestyksessä kuin ne ovat transaktion Ti sisällä. Muiden transaktioiden operaatioita voidaan kuitenkin suorittaa Ti:n operaatioiden välissä
 - T1 = [t11, t12, t13], T2 = [t21, t22], T3 = [t31, t32, t33]
 - S = [t11 t31 t12 t21 t22 t32 t13 t33]

4



Transaktionhallinta - samanaikaisuus

- Esim. sarjallinen ajoitusjärjestys:
 - T1: read(x1, v1) :
 - T1: write(x1, v2)
 - T1: commit;
 - T2: read(x2, u2)
 - T2: read(x1, u1)
 - T2: write(x1, u3)
 - T2: commit;
- Transaktiojoukon **jokainen** sarjallinen ajoitusjärjestys on oikea; vrt. eristyyvyyden määrittely.

5



Transaktionhallinta - samanaikaisuus

- Huom. **Kahden sarjallisen ajoituksen tulos ei ole aina sama, jos tapahtumat eivät ole toisistaan riippumattomia**
- Esim:
 - T1=Lyhennys(Laina,s): read(Laina,v), write(Laina,v-s)
 - T2=Lisääkorko(Laina,p): read(Laina,u), write(Laina,u+p*u)
 - Olkoon alkuarvo 2000, s=500, p=0.1
 - Lyhennys(Laina,500), Lisääkorko(Laina,0.1) antaa tuloksen 1515 ja
 - Lisääkorko(Laina,0.1), Lyhennys(Laina,500) antaa tuloksen 1520
 - molemmat ovat määritelmän mukaan oikein, jos transaktiot ovat erillisiä (jos halutaan kontrolloida järjestystä, pitää operaatiot laittaa samaan transaktioon)

6



Transaktionhallinta - samanaikaisuus

- Rinnakkaisessa ajoituksessa on ainakin **jokin vaihe**, jossa on samanaikaisesti kesken enemmän kuin yksi transaktio.
- Rinnakkaisia ajoituksia on tyypillisesti monia:
 - transaktion vuoro voi päättyä melkein missä kohdassa tahansa (käytännössä esim. siirräntäoperaation kohdalla; vrt. käyttöjärjestelmätason prosessinhallinta).
- Rinnakkainen ajoitus on oikea, jos se on ekvivalentti jonkin sarjallisen ajoituksen kanssa. Ajoitusta kutsutaan tällöin **sarjallistuvaksi** (serializable)

7



Transaktionhallinta - samanaikaisuus

- Ekvivalenssi voidaan määritellä eri tavoilla, mutta yleisesti käytetään **konfliktiekvivalenssia**:
 - keskenään konfliktioivien operaatioiden järjestys on sama kuin sarjallisessa ajoituksessa
- Operaatiot konfliktioivat, jos
 - ne kuuluvat eri transaktioihin,
 - ne kohdistuvat samaan tietokalkioon, ja
 - ainakin toinen operaatio on write-operaatio.

8



Transaktionhallinta - samanaikaisuus

- Esimerkiksi ajoitus


```

T1: read(x1, v1)
T2: read(x2, v2)
T1: write(x1, v3)
T1: commit;
T2: read(x1, v4)
T2: write(x1, v5)
T2: commit;
      
```
- on konfliktiekvivalentti ajoituksen (T1; T2), mutta ei ajoituksen (T2; T1) kanssa

9



Transaktionhallinta - samanaikaisuus

Esimerkkejä samanaikaisuusongelmista:

- 'lost update' -ongelma
 - Tiilitäntö-esimerkissä esim. seuraava ajoitus:

```

T1      T2
read(X, v);
v:=v-500;
read(X, u);
u:=u-1000;
write(X, v);
commit;
write(X, u);
commit;
      
```

'toistokelvoton luku' (unrepeatable read) tulos eri ennen päivitystä ja sen jälkeen

- johtaa tulokseen X = 1000; transaktion T1 tekemä X-päivitys häviää, kun T2 'kirjoittaa sen päälle'.

10



Transaktionhallinta - samanaikaisuus

Peruuntuvan päivityksen ongelma

```

T1      T2
read(X, v);
v:=v-500;
write(X, v);
read(X, u);
u:=u-1000;
write(X, u);
commit;
abort;
      
```

'liikainen luku' (dirty read): T2 lukee T1:n muuttaman arvon, joka kuitenkin peruuntuu.

- (tulos = 500, vaikka vain T2 toteutuu)

11



Transaktionhallinta - samanaikaisuus

Koosteoperaation suoritus muiden transaktioiden rinnalla -ongelma

- Esim. summan lasku voi johtaa eri tuloksiin riippuen siitä, mitkä muut transaktiot ovat jo ehtineet päivittää summauksen kohteena olevia tietokalkioita ja mitkä tekevät sen vasta myöhemmin.

```

T1      T2      T3
s := 0
s := s + u    u := u + x    v := v - y
s := s + v    missä kohdassa nämä ?
s := s + w
      
```

- Esimerkit rikkovat transaktioiden eristyvyyttä vastaan.

12



Transaktionhallinta - samanaikaisuus

- Eristyvyysrikkomuksissa on kolme päätyyppiä eli eristyvyysanomaliala:
 - 1° **likainen kirjoitus** (dirty write)
 - transaktio kirjoittaa toisen, sitoutumattoman, transaktion kirjoittaman tietoalkion **päälle**
 - 2° **likainen luku** (dirty read)
 - transaktio lukee likaisen eli ei-pysyvän tietoalkion arvon (sitoutumattoman kirjoittaman arvon)
 - 3° **toistokelvoton luku** (unrepeatable read)
 - toinen transaktio kirjoittaa T1:n lukeman tietoalkion ennen kuin T1 sitoutuu

13



Transaktionhallinta - samanaikaisuus

- Likainen kirjoitus:**
- Esim. Tietokannan eheysrajoite: $X = Y$, T1 asettaa $X := Y := 1$; T2 asettaa $X := Y := 2$ (kumpikin transaktio erillään säilyttää eheyden)
- Seuraava ajoitus rikkoo eheyden (lopputulos: $X = 2, Y = 1$):


```
T1: u:=1;
T1: write(X, u);
T2: v:=2;
T2: write(X, v);
T2: write(Y, v);
T2: commit;
T1: write(Y, u);
T1: commit;
```
- T2:n operaatio **write(X, v)** on likainen kirjoitus (muut kirjoitusoperaatiot ovat puhtaita).

14



Transaktionhallinta - samanaikaisuus

- Likainen luku:**
 - Esim. Tietokannan eheysrajoite $X > 0, Y > 0$,
 - T1 asettaa X:lle arvon 1; T2 asettaa Y:lle saman arvon kuin X:llä on.

```
T1: u := 0;
T1: write(X, u);
T2: read(X, v);
T2: write(Y, v);
T2: commit;
T1: u := 1;
T1: write(X, u);
T1: commit;
```
- Tässä T2:n lukuoperaatio on likainen, koska T1 ei ole lukuhetkellä sitoutunut (ja on kirjoittanut T2:n lukeman arvon)

15



Transaktionhallinta - samanaikaisuus

- SQL:ssa on lause **SET TRANSACTION**, jolla sovellusohjelmassa voidaan valita aloitettavan transaktion **eristyvyystaso** (isolation level). Mahdolliset tasot vaatavuudeltaan nousevassa järjestyksessä:
 - 1) **sitoutumattomien luku** (read uncommitted): transaktio saa lukea likaista tietoa tai toistokelvottomasti, mutta ei kirjoita likaista
 - 2) **sitoutuneiden luku** (read committed): transaktio saa lukea toistokelvottomasti, mutta ei kirjoita eikä lue likaista
 - 3) **toistokelpoinen luku** (repeatable read): transaktio ei kirjoita eikä lue likaista eikä lue toistokelvottomasti
 - 4) **sarjallistuvuus** (serializable): kuten 3; lisäksi ns. **haamujen** (phantom) esiintyminen on kielletty
- Oletustaso on standardissa sarjallistuvuus; käytännössä esimerkiksi Oraclessa taso 2.

16



Transaktionhallinta - samanaikaisuus

- Haamu:** erikoistapaus, joka syntyy, kun tietokantaan lisätään transaktiossa T rivi, joka täyttää toisen transaktion S käsittelemien rivien valintaehdon.
- Esim.


```
T: insert into employee values (... , dno=5)
S: select sum(salary) ... where dno=5
```
- Ajoitus (T,S) ottaa mukaan myös uuden työntekijän palkan, ajoitus (S,T) sitä vastoin ei. Jos S ehtii ottaa relaation käyttöönsä ennen lisäystä, kesken laskennan ilmestyvä rivi on ns. **haamu**.

17



Transaktionhallinta - samanaikaisuus

- Eristyvyystaso on yhteydessä samanaikaisuuden hallinnan käytäntöön, esim. lukitusperiaatteeseen. **Ankara** (strict) **kaksivaiheinen lukituskäytäntö** takaa transaktioille eristyvyystason 3.
- Korkea eristyvyystaso rajoittaa samanaikaisia operaatioita. Transaktion eristyvyystaso voidaan asettaa oletusta alemmaksi, jos halutaan lisätä samanaikaisuutta (riski korjausten tarpeelle kasvaa), tai toisinpäin ylemmäksi:


```
set transaction isolation level read committed;
...
update taulu set .....
commit;
```

18



Transaktionhallinta - samanaikaisuus

- Samanaikaisuuden hallinnan sisältämä kontrolli transaktioiden suoritukselle (eristävyyden takaaminen) voidaan hoitaa:
 - asettamalla tietokoneille **lukkoja** (locks):
 - operointi on sallittu vain transaktiolle, joka on saanut haltuunsa tietokoneen lukon (käyttöoikeuden)
 - seuraamalla transaktioiden ajoitusta niihin liittyvien **aikaleimojen** (timestamp) avulla
 - ylläpitämällä tietokoneiden useita arvoja ('vanha' ja 'uusi'): **moniversioteknikalla**
 - **optimistisilla menetelmillä**: antamalla transaktioiden suorittaa varsinaiset operaationsa ja tarkistamalla sitten valdointivaiheessa, ettei suoritukseen sisälly ristiriitaisia tilanteita
 - (operaatiot kohdistuvat tietokoneiden tilapäisiin kopioihin, joten menetelmä on tavallaan moniversiainen)
- **Lukitus** on yleisimmin käytössä.

19



Transaktionhallinta - samanaikaisuus

- **Lukko** (lock) on tietokoneen käyttöä valvova muuttuja.
- Lukkoja on erityyppisiä:
 - **lukulukko** (read lock; shared lock) antaa oikeuden lukea tietokoneen, mutta ei kirjoittaa sitä
 - usealla transaktiolla voi samanaikaisesti olla lukulukko tiettyyn tietokoneeseen (lukuoperaatiot eivät häiritse toisiaan, 'shared')
 - **kirjoituslukko** (write lock, exclusive lock) antaa oikeuden kirjoittaa tietokoneen (ja lukea) tietokoneen arvon
 - kirjoituslukko on poissulkeva, 'yksityinen':
 - **vain yhdellä** transaktiolla voi olla **samanaikaisesti** kirjoituslukko tietokoneeseen X
 - **muilla ei voi olla edes lukulukkoa tietokoneeseen X**

20



Transaktionhallinta - samanaikaisuus

- Lukkojen käyttöön liittyviä operaatioita
 - read_lock(X)**: lukulukon pyyntö
 - write_lock(X)**: kirjoituslukon pyyntö
 - unlock(X)**: X:n lukon vapautus
- Lukkoja valvoo tkhj:n lukonhallitsin (lock manager).
- Transaktiolla on enintään yksi lukko tietokoneeseen kerrallaan.
- Lukot vapautetaan viimeistään sitoutumisen yhteydessä

21



Transaktionhallinta - samanaikaisuus

- Lukkojen hallinnan keskeiset tietorakenteet:
 - **lukkotaulu** (lock table), joka on organisoitu tietokonekohtaisesti:
 - tietokoneen X tunniste
 - X:n lukon haltijoiden tiedot: transaktion tunniste ja lukon tyyppi
 - X:n lukkoja haluavien transaktioiden jono
 - Lukkotaulusta saadaan nopeasti selville tietokoneen lukituksen tilanne. Organisoitina voi olla esim. hajautusrakenne
 - **transaktiotaulu**
 - jokaiselle transaktiolle tietue, josta alkaa transaktion hallussa olevien lukkojen (tietokoneiden tunnisteiden) ketju
 - taulun avulla löydetään transaktion sitoutuessa tai peruuntuessa sen hallussa mahdollisesti olevat lukot, jotka on vapautettava.

22



Transaktionhallinta - samanaikaisuus

- **Lukitusoperaatioiden toiminta, transaktio T**
- **read_lock(X)**: ($rl(X) = X:n$ lukulukkojen määrä)
 1. hae X:ää lukkotaulusta
 2. jos X ei ole taulussa, vie X tauluun, aseta $rl(X) := 0$ ja mene askeleeseen 5
 3. jos transaktiolla T on jo lukko X:ään, palaa
 4. jos jollakin toisella transaktiolla on jo kirjoituslukko X:ään, aseta T lukon vapautumista odottavien transaktioiden jonoon lukkotaulussa
 5. kirjaa lukkotauluun T:lle lukulukko X:ään (ja liitä X T:n lukkojen ketjuun transaktiotaulussa), aseta $rl(X) := rl(X) + 1$

23



Transaktionhallinta - samanaikaisuus

- **Lukitusoperaatioiden toiminta, transaktio T**
- **write_lock(X)**:
 1. hae X:ää lukkotaulusta
 2. jos X ei ole taulussa, vie X tauluun ja mene askeleeseen 5
 3. jos transaktiolla T on jo kirjoituslukko X:ään, palaa
 4. jos jollakin toisella transaktiolla on jo luku- tai kirjoituslukko X:ään, aseta T lukon vapautumista odottavien transaktioiden jonoon
 5. jos T:llä on jo lukulukko X:ään, korota (upgrade) se kirjoituslukoksi ja aseta $rl(X) := 0$; muuten kirjaa T:lle uusi lukko, kirjoituslukko, lukkotauluun

24



Transaktionhallinta - samanaikaisuus

- **Lukitusoperaatioiden toiminta, transaktio T**
- unlock(X):
 1. etsi X:ää vastaava tietue lukkotaulusta
 2. jos X:ään on kirjoituslukko transaktiolla T, poista T lukonhaltijain joukosta ja, jos jonossa on odottavia transaktioita, herätä niistä ensimmäinen muuten (T:llä on lukulukko)
 - aseta $rl(X) := rl(X) - 1$;
 - Jos $rl(X) = 0$, herätä ensimmäinen odottavista transaktioista, jos niitä on
 3. jos odottavien jono oli tyhjä, poista X:n tietue lukkotaulusta

25



Transaktionhallinta - samanaikaisuus

- Herätetty transaktio jatkaa siis suoritustaan read_lock- tai write_lock-operaationsa askeleesta 4.
- Lukitusoperaatiot (lukonpyynnöt) read_lock(X) ja write_lock(X) voidaan ajatella vastaavia luku- ja kirjoitusoperaatioita read(X,v), write(X,v) edeltäviksi operaatioiksi transaktion suorituksessa.
- Ne eivät kuitenkaan automaattisesti takaa transaktion sarjallisuutta; tarvitaan hyvin määritelty lukituskäytäntö.

26



Transaktionhallinta - samanaikaisuus

- Samanaikaisuuden hallinta kuuluu tkhj:n tehtäviin.
 - Tietokantasovelluksen ohjelmoijan (tai kyselyjä tekevän käyttäjän) ei siten tarvitse huolehtia tietoaikoiden lukituksesta.
 - Esim. SQL:
 - select -lausetta suoritettaessa varataan lukulukkoja hakemisto- ja tietoaikioille sen mukaan kuin on tarvetta lukea ko. alkioita
 - insert-, update- ja delete -operaatioille varataan vastaavasti kirjoituslukkoja
- Normaalisti lukot vapautetaan vasta transaktion päättyessä: sen sitouduttua tai peruunnuttua. Vapautus tehdään lauseella **unlock(all)**, joka vapauttaa kaikki transaktionsa lukot.
- Transaktion päättymiseen asti pidettävä lukko on **pitkäaikainen**, aikaisemmin eksplisiittisesti vapautettavat **lyhytaikaisia**. Lyhytaikainen lukko vapautetaan normaalisti heti operaation jälkeen

27



Transaktionhallinta - samanaikaisuus

- Esimerkiksi tietohakemistosivuihin ja indeksisivuihin (ISAM, B+ -puu) kohdistuvat lukot ovat yleensä lyhytaikaisia.
- Sovelluksen ohjelmoija vaikuttaa lukkojen varausaikaan yleensä vain epäsuorasti määrittelemällä transaktiot 'sopivan' pituisiksi (mahdollisimman lyhyiksi).
- Lukkojen **karkeus**, **rakeisuus** (**granulaarisuus**, **granularity**) on tärkeä samanaikaisuuden asteeseen vaikuttava tekijä. Lukittava 'tietoaikio' voi olla yksittäinen kenttä, tietue, sivu, taulu tai jopa koko tietokanta. Hienojakoinen lukitus vaatii paljon lukkoja ja monimutkaista lukkojen hallintaa, mutta sallii maksimaalisen samanaikaisuuden.
- Käytännössä on yleistä rivi-, sivu- tai taulukohtainen lukinta.

28



Transaktionhallinta - samanaikaisuus

- **Kaksivaiheinen lukituskäytäntö (2PL, two-phase locking)**
- Kaksivaiheisen lukituksen perusajatus:
 - mitään lukkoa ei vapauteta ennen kuin kaikki transaktion tarvitsemat lukot on varattu
- Transaktio jakaantuu kahteen vaiheeseen:
 - **Kasvuvaiheeseen**, jonka aikana kaikki lukot varataan,
 - Lukulukon korotus kirjoituslukoksi tulkitaan lukon varaukseksi eli on tehtävä kasvuvaiheen aikana.
 - **Kutistumisvaiheeseen**, jonka aikana lukot vapautetaan.

29



Transaktionhallinta - samanaikaisuus

- Kaksivaiheinen lukitus voi olla perusmuodon lisäksi mm.
 - **ankara** (**strict**, **E&N:rigorous**, **E&N:n strict** = kirjoituslukot sitoutumiseen asti): kaikki lukot vapautetaan vasta transaktion sitoutuessa (tai peruuntuessa)
 - **konservatiivinen** (**conservative**) kaikki lukot varataan transaktion alussa
 - tulee tietää tarvittavien tietoaikoiden joukot:
 - read-set ja write-set

30



Transaktionhallinta - samanaikaisuus

- Konservatiivinen menetelmä:
 - voi olla liian varovainen: transaktion vaikea päästä alkuun
 - luku- ja kirjoitusjoukkojen määrittäminen etukäteen hankalaa
 - plussaa: **transaktiot eivät voi lukkiutua**
- Ankara kaksivaiheinen lukitus on käytännössä yleisin. Se takaa transaktion sarjallisuuden, jos kaikki transaktiot noudattavat samaa käytäntöä.
 - Käytännön merkitys: ei tarvitse tutkia erikseen ajoituksen sarjallisuutta (read/write-suhteiden verkko; verkon syklittömyys), lukkojen varaus- ja vapautusperiaate riittää.
 - Transakzioista voidaan muodostaa verkko, jonka solmuina ovat transaktiot ja särminä transaktioita yhdistävät konfliktioivien operaatioiden (r/w, w/r, w/w) parit (sarjallisuusverkko). Jos verkossa on sykli, vastaava ajoitus ei ole sarjallistuva, muuten on.

31



Transaktionhallinta - samanaikaisuus

- Esim. seuraavat transaktiot toteuttavat 2PL-ehdon:

• T1'	T2'
• read_lock(Y);	read_lock(X);
• read(Y);	read(X);
• write_lock(X);	write_lock(Y);
• unlock(Y);	unlock(X);
• read(X);	read(Y);
• X:= X + Y;	Y:= X + Y;
• write(X);	write(Y);
• commit;	commit;
• unlock(X);	unlock(Y);
- Tässä tapauksessa lukkiutuma (deadlock) on mahdollinen!
 - T1' varaa Y:n lukulukon, T2' X:n lukulukon
 - lukkiutuman hoitamiseen on erilaisia menetelmiä
- lukulukon vapautus ei päästä toista transaktiota eteenpäin eli toiminta vastaa 2PL-käytäntöä (ei kuitenkaan ankaraa)
 - käytännössä tässä tulee kyseeseen jompikumpi sarjallinen suoritus

32



Transaktionhallinta - samanaikaisuus

- Kuinka ankara 2PL estää eristyvyysanomalia?
- 1° likainen kirjoitus
- Esim


```
T1: write_lock(X);
T1: write(X, u);
T2: write_lock(X);
T2: write(X, v);
T1: commit; (tai rollback);
T1: unlock(X);
```
- Ajoitus ei ole mahdollinen, koska T2 ei voi saada X:n kirjoituslukkua eikä siten tehdä liikaista kirjoitusta.
- Olennaista on, että T1:n kirjoituslukko on pitkäaikainen;

33



Transaktionhallinta - samanaikaisuus

- Kuinka ankara 2PL estää eristyvyysanomalia?
- 2° likainen luku
- Esim.


```
T1: write_lock(X);
T1: write(X, u);
T2: read_lock(X);
T2: read(X, v);
T1: commit; (tai rollback);
T1: unlock(X);
```
- T2 ei voi saada edes lukulukkoa, kun X:llä on pitkäaikainen kirjoituslukko.
- On helppo nähdä, että ankara 2PL rajoittaa usein 'liian paljon' samanaikaisuutta: tietyn tietokoneen lukko voitaisiin vapauttaa heti, kun siihen kohdistuvat operaatiot on tehty.
 - Tätä 'hintaa' pidetään järkevänä verrattuna kunkin ajoituksen sarjallisuuden selvittämiseen erikseen.

34



Transaktionhallinta - samanaikaisuus

- 3° toistokelvoton luku
- Esim.


```
T1: read_lock(X);
T1: read(X, u);
T2: write_lock(X);
T2: write(X, v);
T1: commit; (tai: rollback);
T1: unlock(X);
...
```
- Toistokelvoton luku estyy: T2 ei voi saada kirjoituslukkua X:ään eli ei voi muuttaa X:n arvoa, kun T1:llä on pitkäaikainen lukulukko. (Lyhytaikainen lukulukko ei riitä.)
- On luontevaa ajatella, että sama lukituskäytäntö koskee kaikkia transaktioita. Transaktiokohtaisia muutoksia voidaan kuitenkin tehdä (eristyisyystason asetus lauseella set transaction ... - ei enää 2PL).

35



Transaktionhallinta - samanaikaisuus

- **Lukitus ja hakemistot**
- Kaksivaiheinen lukitus **ei sovellu hyvin** hakemistojen käsittelyyn:
 - hierarkkisen hakemiston käsittely alkaa juuresta
 - alempien tasojen ja tietosivujen käsittelyssä tarvitaan lukkoja myöhemmin, jolloin ei enää välttämättä tarvita ylempien tasojen sivuja
 - lukkojen vapautus vasta kutistusvaiheessa rajoittaa huomattavasti samanaikaisuutta
 - Useimmiten isäsivun lukko voitaisiin vapauttaa, kun lapsisivuun on saatu lukulukko. Jos tulee tarvetta päivittää hakemistosivua, varataan päivitystä varten lukko uudelleen.

36



Transaktionhallinta - samanaikaisuus

- Esim. B+ -puu:
- **konservatiivinen tapa:**
 - varataan tasoitain kirjoituslukkoja; vapautetaan ne, kun on saatu lukko seuraavalle tasolle ja todettu, ettei tarvitse palata (sivulla on tilaa uudelle alkioille)
- **optimistinen tapa:**
 - varataan lukulukkoja tasojen alaspäin edettäessä; jos lehtitaso jaetaan ja ylempää tasoa pitää päivittää, korotetaan lukulukkoja kirjoituslukoiksi (tarpeen mukaan)

37



Transaktionhallinta - samanaikaisuus

- Hakemistojen lukituksella voidaan vaikuttaa myös ns. haamutietueiden havaitsemiseen: tiheän hakemiston tapauksessa hakemistosivun lukitus estää päivittävää ja lukevaa transaktiota pääsemästä samanaikaisesti tietosivulle
- esim (indeksi dno:n perusteella)
 - T: insert into employee values (... , dno=5)
 - S: select sum(salary) ... where dno=5
- jompikumpi varaa hakemistosivun lukon eli rivin lisäksi ei tapahdu kesken laskennan
- Datasivujen lukot / haamutietueet?
 - dno = 5 –tietueet monella datasivulla => yksittäisen datasivun lukitus ei riitä
 - dno = 5 –tietueet samalla datasivulla => lukitus riittää

38



Transaktionhallinta - samanaikaisuus

- Lukkiutuma (deadlock)
- **transaktio yrittää lukita jotakin toisen transaktion käytössä olevaa tietoalkiota ja tämä toinen transaktio odottaa lukitusta yrittävän vapauttavan jonkin lukon.** (esim. T1', T2' sivulla 32)

Mitä voidaan tehdä?

- 1) **lukkiutuman estävä käytäntö**
 - konservatiivinen 2PL: kaikki lukot alussa
 - varataan paljon lukkoja turhan aikaisin
 - samanaikaisuus vähäistä
 - vaikea tietää, mitä tietoalkioita tarvitaan

39



Transaktionhallinta - samanaikaisuus

- 2) tietoalkioiden järjestykseen perustuva varaaminen
 - estää ristikkäiset varaukset
 - järjestys? sen ylläpito?
- 3) no wait –periaate: transaktio ei odota koskaan, vaan käynnistyy myöhemmin uudelleen
 - vuoro transaktiolla taattava 'joskus' – tai sovellus luopuu transaktiosta kokonaan, 'ei enää tarpeen'
- 4) aikaleimoihin perustuva lukkiutuman ratkaisu:
 - esim. nuorempi peruutetaan ja aloitetaan myöhemmin uudelleen samalla aikaleimalla (**wait-die**)
- 5) lukkiutuman havaitseminen ja purku:
 - odotusverkko
 - odotusverkon ylläpito: verkon sykli merkitsee lukkiutuman syntymistä
 - **purku:** peruutetaan jokin transaktio ja aloitetaan se myöhemmin uudelleen

40



Transaktionhallinta - samanaikaisuus

- Transaktioiden odotus lukinnassa tai lukkiutumia purettaessa voi johtaa nälkiintymiseen (starvation):
 - **lukon odotus:**
 - vaikka ei synny lukkiutumaa, jokin transaktio häviää aina kilpailun lukitusvuorosta
 - ratkaisuja: FIFO, prioriteetin kasvatus odottaessa
 - **lukkiutuman purku:**
 - transaktio ei saa vuoroa uudelleenaloituksissa (vrt. wait-die: alkuperäisen aloitusajan säilyminen)

41