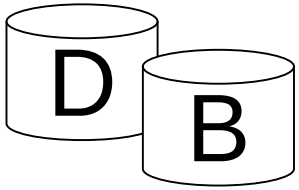
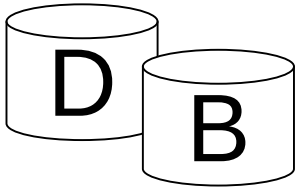


Transaktionhallinta - samanaikaisuus

- 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 (eristyvyys; välttämätön ominaisuus)



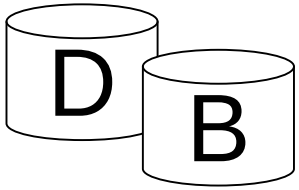
Transaktionhallinta - samanaikaisuus

- 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)

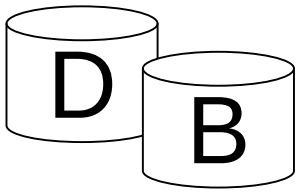
T2
read(X, u) u:=u-1000; write(X, u)

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



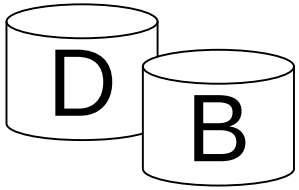
Transaktionhallinta - samanaikaisuus

- Transaktioiden T1,...,Tn ajoitusjärjestys (schedule, history) on näiden operaatioista (luku/kirjoitus) muodostuva jono, 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]



Transaktionhallinta - samanaikaisuus

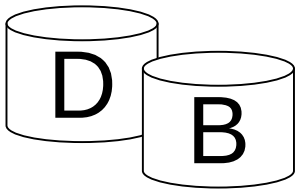
- Kontrolloitu suoritus olisi sarjallinen suoritus (serial schedule), jossa T1 joko suoritetaan kokonaan ennen kuin T2 aloitetaan tai päinvastoin.
  - Tällöin lopputulos molempien jälkeen on 500, mutta välitulos on joko 1000 tai 1500.
- Yleisesti sarjallinen suoritus aiheuttaa joidenkin transaktioiden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



Transaktionhallinta - samanaikaisuus

- Samanaikaisuuden hallinnan alijärjestelmän tehtävänä on lieventää hallitusti sarjallisuuden vaatimusta eli sallia rinnakkaisuutta, mutta eliminoida sen haitat.

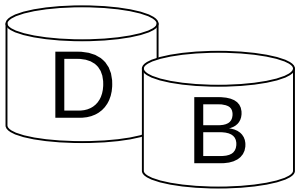
- Aiemmin määriteltiin transaktioiden T1,...,Tn ajoitusjärjestys (schedule, history) näiden operaatioista (luku/kirjoitus) muodostuvaksi jonoksi, jossa kunkin transaktion Ti operaatiot ovat samassa järjestyksessä kuin ne ovat transaktion Ti sisällä. Muiden transaktioiden operaatioita voidaan kuitenkin 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]



Transaktionhallinta - samanaikaisuus

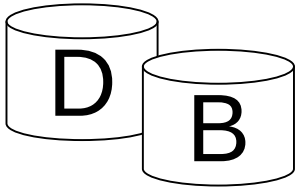
- Esim. sarjallinen ajoitusjärjestys:
  T1: read (x1, v1) :
  T1: write(x1, v2)
  T1: commit;
  T2: read(x2, u2)
  T2: read(x1, u1)
  T2: write(x1, u3)
  T2: commit;

- Transaktiojoukon jokainen sarjallinen ajoitusjärjestys on oikea; vrt. eristyvyyden määrittely.



Transaktionhallinta - samanaikaisuus

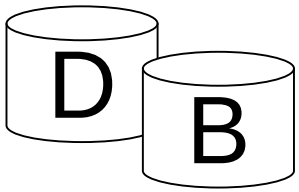
- Huom. Kahden sarjallisen ajoituksen tulos ei ole aina sama, jos tapahtumat eivät ole toisistaan riippumattomia
- Esim:
  - T1=Lyhennys(Laina,s): read(Laina,v), write(Laina,v-s)
  - T2=Lisääkorko(Laina,p): read(Laina,u), write(Laina,u+p*u)
  - Olkoon alkuarvo 2000, s=500, p=0.1
    - Lyhennys(Laina,500), Lisääkorko(Laina,0.1) antaa tuloksen 1650 ja
    - Lisääkorko(Laina,0.1), Lyhennys(Laina,500) antaa tuloksen 1700
  - molemmat ovat määritelmän mukaan oikein, jos transaktiot ovat erillisiä (jos halutaan kontrolloida järjestystä, pitää operaatiot laittaa samaan transaktioon)



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).

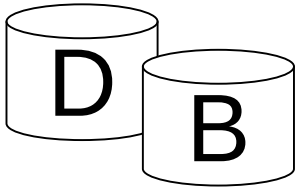
- Rinnakkainen ajoitus on oikea, jos se on ekvivalentti jonkin sarjallisen ajoituksen kanssa. Ajoitusta kutsutaan tällöin sarjallistuvaksi (serializable)



Transaktionhallinta - samanaikaisuus

- Ekvivalenssi voidaan määritellä eri tavoilla, mutta yleisesti käytetään konfliktiekvivalenssia, jossa keskenään konfliktoivien operaatioiden järjestys on sama kuin sarjallisessa ajoituksessa

- Operaatiot konfliktoivat, jos
  1) ne kuuluvat eri transaktioihin,
  2) ne kohdistuvat samaan tietoalkioon, ja
  3) ainakin toinen operaatio on write-operaatio.



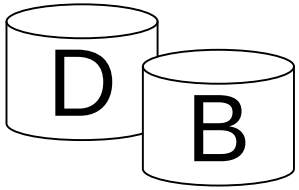
Transaktionhallinta - samanaikaisuus

- 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;

sarjallisessa ajoituksessa
T2;T1 pitäisi T2: write(X1,v5)
suorittaa ennen
kuin mikään T1:n operaatio

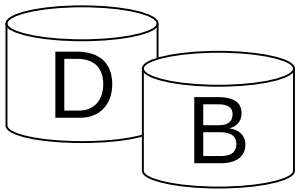
- on konfliktiekvivalentti sarjallisen ajoituksen (T1; T2), mutta ei ajoituksen (T2; T1) kanssa



Transaktionhallinta - samanaikaisuus

- Eristyvyysrikkomuksissa on kolme päätyyppiä eli eristyvyysanomaliaa:

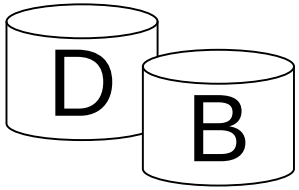
- 1° likainen kirjoitus (dirty write)
  - transaktio kirjoittaa toisen, sitoutumattoman, transaktion kirjoittaman tietoalkion päälle
- 2° likainen luku (dirty read)
  - transaktio lukee likaisen eli ei-pysyvän tietoalkion arvon (sitoutumattoman kirjoittaman arvon)
- 3° toistokelvoton luku (unrepeatable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



Transaktionhallinta - samanaikaisuus

- Likainen kirjoitus:
  - T2 kirjoittaa X:n päälle ja T1:n tekemä päivitys katoaa

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



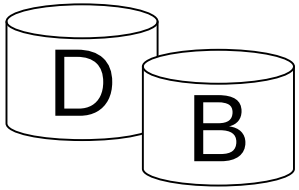
Transaktionhallinta - samanaikaisuus

- Likainen luku:

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

‘likainen luku’ (dirty read):
T2 lukee T1:n muuttaman arvon,
joka kuitenkin peruuntuu .

- (tulos = 500, vaikka vain T2 toteutuu)

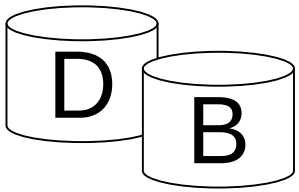


Transaktionhallinta - samanaikaisuus

- toistokelvoton luku:

```
T1          T2
 read(X, b1);
 ip=determine_interest(b1);
                read(X, u);
                u:=u-1000;
                write(X, u);
                commit;
 read(X,v);
 v:=v+ip;
 write(X, v);
 commit;
```

’toistokelvoton luku’
(unrepeatable read)
luetaan ensin saldo koron määräämistä varten sitten päivitystä varten, saadaan eri arvot.

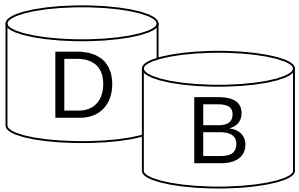


Transaktionhallinta - samanaikaisuus

- 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 tietoalkioita ja mitkä tekevät sen vasta myöhemmin.

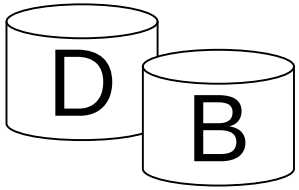
T1	T2	T3
s := 0		
s := s + u	u := u + x	v := v – y
s := s + v	missä kohdassa nämä ?	
s := s + w		

- Esimerkit rikkovat transaktioiden eristyvyyttä vastaan.



Transaktionhallinta - samanaikaisuus

- SQL:ssa on lause SET TRANSACTION, jolla sovellusohjelmassa voidaan valita aloitettavan transaktion eristyvyystaso (isolation level). Mahdolliset tasot vaativuudeltaan 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.



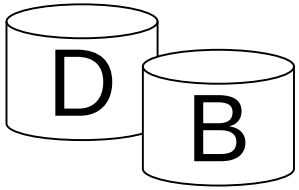
Transaktionhallinta - samanaikaisuus

- 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.

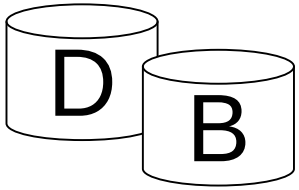


Transaktionhallinta - samanaikaisuus

- Eristyvyystaso on yhteydessä samanaikaisuuden hallinnan käytäntöön, esim. lukitusperiaatteeseen. Ankara (strict) kaksivaiheinen lukituskäytäntö takaa transaktioille eristyvyystason 3.

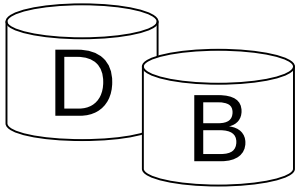
- Korkea eristyvyystaso rajoittaa samanaikaisia operaatioita. Transaktion eristyvyystaso voidaan asettaa oletusta alemmaksi, jos halutaan lisää samanaikaisuutta (riski korjausten tarpeelle kasvaa), tai toisinpäin ylemmäksi:

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



Transaktionhallinta - samanaikaisuus

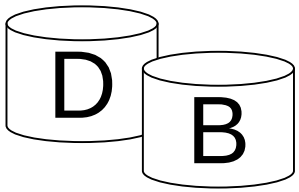
- Samanaikaisuuden hallinnan sisältämä kontrolli transaktioiden suoritukselle (eristyvyyden takaaminen) voidaan hoitaa:
  - asettamalla tietoalkioille **lukkoja** (locks):
    - operointi on sallittu vain transaktiolle, joka on saanut haltuunsa tietoalkion lukon (käyttöoikeuden)
  - seuraamalla transaktioiden ajoitusta niihin liittyvien **aikaleimojen** (timestamp) avulla
  - ylläpitämällä tietoalkioiden useita arvoja (‘vanha’ ja ‘uusi’): **moniversiotekniikalla**
  - **optimistisilla menetelmillä**: antamalla transaktioiden suorittaa varsinaiset operaationsa ja tarkistamalla sitten validointivaiheessa, ettei suoritukseen sisälly ristiriitaisia tilanteita
    - (operaatiot kohdistuvat tietoalkioiden tilapäisiin kopioihin, joten menetelmä on tavallaan moniversioinen)
  - Lukitus on yleisimmin käytössä.



Transaktionhallinta - samanaikaisuus

- Lukko (lock) on tietoalkion käyttöä valvova muuttuja. Transaktiolla pitää olla hallussaan operaation tarvittava lukko, jotta se voisi suorittaa operaation

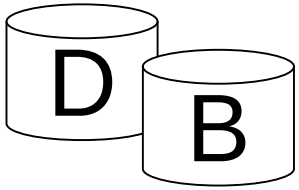
- Lukkoja on erityyppisiä:
  - lukulukko (lukulupa, read lock; shared lock) antaa oikeuden lukea tietoalkion, mutta ei kirjoittaa sitä
    - usealla transaktiolla voi samanaikaisesti olla lukulukko tiettyyn tietoalkioon (lukuoperaatiot eivät häiritse toisiaan, ‘shared’)
  - kirjoituslukko (kirjoituslupa, write lock, exclusive lock) antaa oikeuden kirjoittaa (ja lukea) tietoalkion arvon
    - kirjoituslukko on poissulkeva, ‘yksityinen’ (private):
    - vain yhdellä transaktiolla voi olla samanaikaisesti kirjoituslukko tietoalkioon X
    - muilla ei voi olla samanaikaisesti edes lukulukkoa tietoalkioon X



Transaktionhallinta - samanaikaisuus

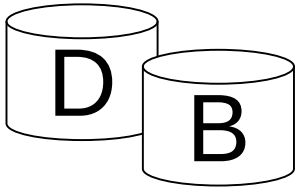
- Lukkojen käyttöön liittyviä operaatioita
  - read_lock(X): lukulukon pyyntö
  - write_lock(X): kirjoituslukon pyyntö
  - unlock(X): X:n lukon vapautus

- Lukkoja valvoo tkhj:n lukonhallitsin (lock manager).
- Transaktiolla on enintään yksi lukko tietoalkioon kerrallaan.
- Lukot vapautetaan viimeistään sitoutumisen yhteydessä



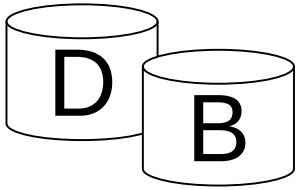
Transaktionhallinta - samanaikaisuus

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



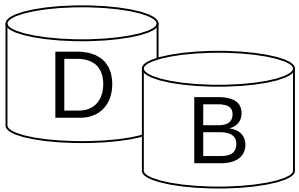
Transaktionhallinta - samanaikaisuus

- Lukitusoperaatioiden toiminta, transaktio T
- read_lock(X): (rl(X) = X:n lukulukkojen määrä)
  1. hae alkiota X lukkotaulusta
  2. jos X ei ole taulussa, lisää 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
  5. kirjaa lukkotauluun T:lle lukulukko X:ään (ja liitä X T:n
     lukkojen ketjuun transaktiotaulussa),
     aseta rl(X) := rl(X) + 1



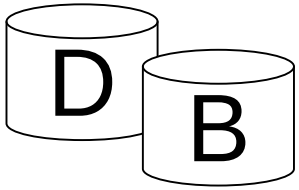
Transaktionhallinta - samanaikaisuus

- Lukitusoperaatioiden toiminta, transaktio T
- write_lock(X):
  1. hae tietoalkio X 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



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

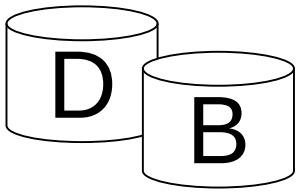


Transaktionhallinta - samanaikaisuus

- 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. Unlock(X) operaation suorituksen jälkeen.

- Ne eivät kuitenkaan automaattisesti takaa transaktion sarjallistuvuutta; tarvitaan hyvin määritelty lukituskäytäntö.



Transaktionhallinta - samanaikaisuus

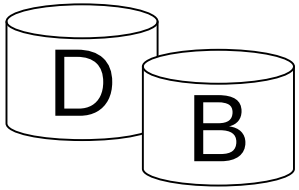
- Samanaikaisuuden hallinta kuuluu tkhj:n tehtäviin.
  - Tietokantasovelluksen ohjelmoijan (tai kyselyjä tekevän käyttäjän) ei siten tarvitse huolehtia tietoalkioiden lukituksesta.
  - Esim. SQL:
    - select -lausetta suoritettaessa varataan lukulukkoja hakemisto- ja tietoalkioille sen mukaan kuin on 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



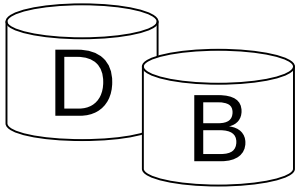
Transaktionhallinta - samanaikaisuus

- 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).

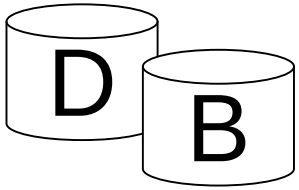
- Lukkojen rakeisuus (granulaarisuus, granularity) on tärkeä samanaikaisuuden asteeseen vaikuttava tekijä. Lukittava ‘tietoalkio’ voi olla yksittäinen kenttä, tietue, sivu, taulu tai jopa koko tietokanta. Hienojakoinen lukitus vaatii paljon lukkoja ja monimutkaista lukkojen hallintaa, mutta sallii maksimaalisen samanaikaisuuden.

- Käytännössä on yleistä rivi-, sivu- tai taulukohtainen lukinta.



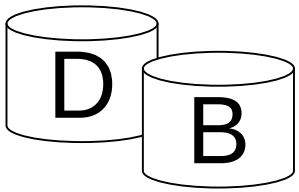
Transaktionhallinta - samanaikaisuus

- 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.



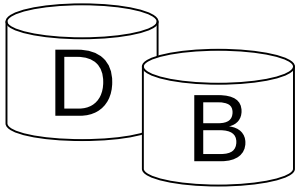
Transaktionhallinta - samanaikaisuus

- 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 heti transaktion alussa
    - tulee tietää tarvittavien tietoalkioiden joukot:
      - *read-set ja write-set*



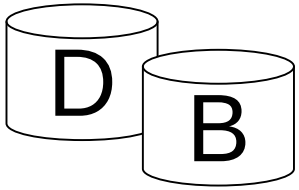
Transaktionhallinta - samanaikaisuus

- Konservatiivinen menetelmä:
  - voi olla liian varovainen: transaktion vaikea päästä alkuun
  - luku- ja kirjoitusjoukkojen määritys etukäteen hankalaa
  - plussaa: transaktiot eivät voi lukkiutua
    - lukkiutuminen: transaktio jää odottamaan, että toinen transaktio vapauttaisi lukon, mutta toinen ei voi edetä, koska se odottaa vastaavasti lukon vapautumista



Transaktionhallinta - samanaikaisuus

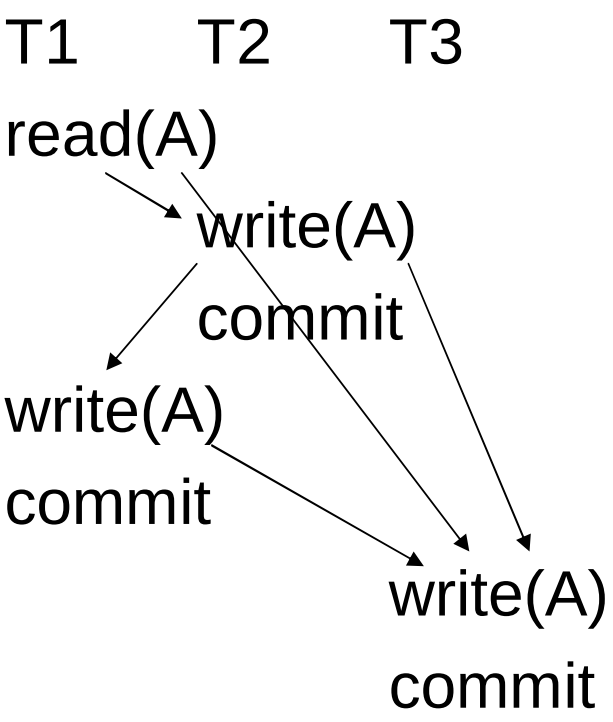
- Ankara kaksivaiheinen lukitus on käytännössä yleisin. Se takaa transaktion sarjallistuvuuden, jos kaikki transaktiot noudattavat samaa käytäntöä.
  - Käytännön merkitys: ei tarvitse tutkia erikseen ajoituksen sarjallistuvuutta (read/write-suhteiden verkko; verkon syklittömyys), lukkojen varaus- ja vapautusperiaate riittää.

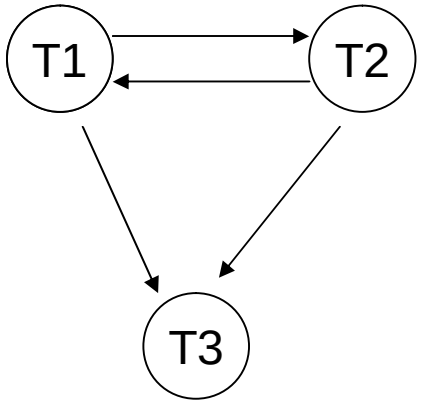


Transaktionhallinta - samanaikaisuus

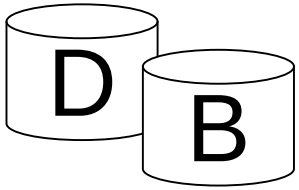
- Sarjallistuvuuden tutkiminen:
  - Transaktioista voidaan muodostaa suunnattu verkko, (ns. sarjallistuvuusverkko), jonka solmuina ovat transaktiot ja suunnattuina särminä transaktioita yhdistävät konfliktoivien operaatioiden (r/w, w/r, w/w) parit. Verkossa on särmä T:stä S:ään, jos T:n operaatio suoritetaan ennen sen kanssa konfliktoivaa S:n operaatiota. Jos verkossa on sykli, vastaava ajoitus ei ole sarjallistuva, muuten on.

Transaktionhallinta - samanaikaisuus





syklejä, ei sarjallistuva

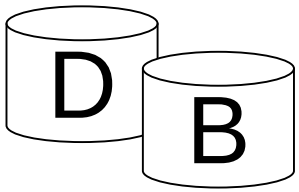


Transaktionhallinta - samanaikaisuus

- 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

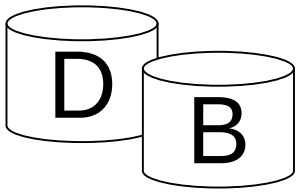


Transaktionhallinta - samanaikaisuus

- Kuinka ankara 2PL estää eristyvyysanomaliat?
- 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 kirjoituslukkoa eikä siten tehdä likaista kirjoitusta.
- Olennaista on, että T1:n kirjoituslukko on pitkäaikainen;

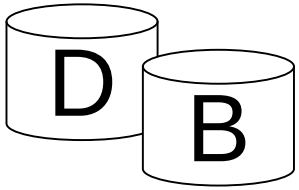


Transaktionhallinta - samanaikaisuus

- Kuinka ankara 2PL estää eristyvyysanomaliat?
- 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 tietoalkion lukko voitaisiin vapauttaa heti, kun siihen kohdistuvat operaatiot on tehty.
  - Tätä ‘hintaa’ pidetään järkevänä verrattuna kunkin ajoituksen sarjallistuvuuden selvittämiseen erikseen.



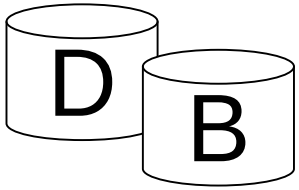
Transaktionhallinta - samanaikaisuus

- 3° toistokelvoton luku
-
- Esim.

```
T1:  read_lock(X);
 T1:  read(X, u);
    T2:  write_lock(X);
    T2:  write(X, v);
 T1:read(X,z)
 T1:  commit;    (tai: rollback;)
 T1:  unlock(X);
       . . .
```

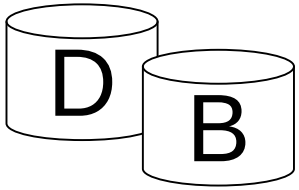
- Toistokelvoton luku estyy: T2 ei voi saada kirjoituslukkoa X:ään eli ei voi muuttaa X:n arvoa, kun T1:llä on pitkäaikainen lukulukko. (Lyhytaikainen lukulukko ei riitä.)

- On luontevaa ajatella, että sama lukituskäytäntö koskee kaikkia transaktioita. Transaktiokohtaisia muutoksia voidaan kuitenkin tehdä (eristyneisyystason asetus lauseella set transaction … - ei enää 2PL).



Transaktionhallinta - samanaikaisuus

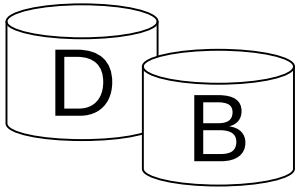
- Lukitus ja hakemistot
- Kaksivaiheinen lukitus ei sovellu hyvin hakemistojen käsittelyyn:
  - hierarkkisen hakemiston käsittely alkaa juuresta
  - alempien tasojen ja tietosivujen käsittelyssä tarvitaan lukkoja myöhemmin, jolloin ei enää välttämättä tarvita ylempien tasojen sivuja
    - lukkojen vapautus 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.



Transaktionhallinta - samanaikaisuus

- Esim. B+ -puu:
- konservatiivinen tapa:
  - varataan tasoittain kirjoituslukkoja; vapautetaan ne, kun on saatu lukko seuraavalle tasolle ja todettu, ettei tarvitse palata (sivulla on tilaa uudelle alkiolle)

- optimistinen tapa:
  - varataan lukulukkoja tasoja alas edettäessä; jos lehtitaso jaetaan ja ylempää tasoa pitää päivittää, korotetaan lukulukkoja kirjoituslukoiksi (tarpeen mukaan)



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 lisäys ei tapahdu kesken laskennan

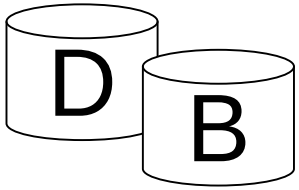
- Datasivujen lukot / haamutietueet?

  dno = 5 –tietueet monella datasivulla =>

  yksittäisen datasivun lukitus ei riitä

  dno = 5 –tietueet samalla datasivulla =>

  lukitus riittää

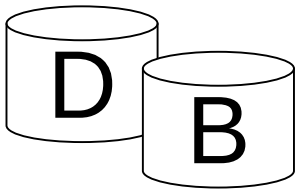


Transaktionhallinta - samanaikaisuus

- 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 35)

Mitä voidaan tehdä?

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



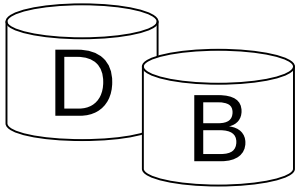
Transaktionhallinta - samanaikaisuus

- 2) tietoalkioiden järjestykseen perustuva varaaminen
  - estää ristikkäiset varaukset
  - mikä järjestys? järjestyksen ylläpito?

- 3) no wait –periaate: transaktio ei odota koskaan, vaan käynnistyy myöhemmin uudelleen
  - vuoro transaktiolle 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



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)