

5.2 Samanaikaisuuden hallinta

Tietokannalla on tyypillisesti useita samanaikaisia käyttäjiä (ohjelmia/ihmisiä).

On toivottavaa, että

- yhdenkään käyttäjän toiminta ei hidastuisi kohtuuttomasti, vaikka muita käyttäjiä olisi runsaastikin (ainakaan toiminta ei saisi estyä kokonaan)
- tietokanta säilyy eheänä eli että jokaisen transaktion toiminnot pysyvät periaatteessa loogisesti erillään muiden transaktioiden toiminnoista (eristyvyys; välttämätön ominaisuus)

Esim. nosto pankkitililtä:

- useat transaktiot generoituvat samasta proseduurista tilinosto(X, summa):
(X = tietyn tilin saldo tili-relaatiossa, alussa esim. 2000)

T1	T2
read_item(X, v)	read_item(X, u)
v:=v-500;	u:=u-1000;
write_item(X, v)	write_item(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 markkaa

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. Ekvivalenssi voidaan määrittellä eri tavoilla.

Esimerkiksi ajoitus

```
T1: read_item(x1, v1)
T2: read_item(x2, u2)
T1: write_item(x1, v1)
T1: commit;
T2: read_item(x1, u1)
T2: write_item(x1, u3)
T2: commit;
```

on ekvivalentti ajoituksen (T1; T2), mutta ei ajoituksen (T2; T1) kanssa.

- tulosekvivalenssi: sama tulos
- konfliktiekvivalenssi: konfliktioivat operaatiot samassa järjestyksessä
- näkemusekvivalenssi: luetaan kummassakin saman write-lauseen tulos,

Hyvä kontrolli olisi esimerkiksi se, että T1 suoritetaan kokonaan ennen transaktiota T2 tai päinvastoin.

Yleistettynä tämä sarjallinen suoritus aiheuttaa kuitenkin joidenkin transaktioiden huomattavan viivästymisen (pahimmillaan suorituksen estymisen, jos käynnissä oleva ei pääse ollenkaan loppuun).

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

Transaktiohistoria eli ajoitus (schedule) = se, järjestys, jossa eri transaktioiden luku- ja kirjoitusoperaatiot suoritetaan.

Esim. sarjallinen historia:

```
T1: read_item(x1, v1)          tai:
T1: write_item(x1, v1)        T2, T2, T2, T2, T1, T1, T1
T1: commit;
T2: read_item(x2, u2)
T2: read_item(x1, u1)
T2: write_item(x1, u3)
T2: commit;
```

Transaktiojoukon jokainen sarjallinen ajoitus on oikea; vrt. eristyvyyden määrittely.
(oikeellisuus eli C-ominaisuus vaaditaan tietysti myös)

Huom. Kahden sarjallisen ajoituksen tulos ei ole aina sama, jos transaktiot eivät ole riippumattomia.

kummassakin sama viimeinen write-lause Ajoitus on sarjallistuva (serializable), jos se on ekvivalentti jonkin sarjallisen suorituksen kanssa. Ekvivalenssi määrittellään yleensä konfliktiekvivalenssina: siis keskenään konfliktioivien eli 'vaarallisten' operaatioiden järjestys säilyy ajoituksissa samana.

Operaatiot konfliktioivat, jos

- 1) ne kuuluvat eri transaktioihin,
- 2) ne kohdistuvat samaan tietokalkioon, ja
- 3) ainakin toinen operaatio on write-operaatio.

Esimerkkejä samanaikaisuusongelmista:

1° 'lost update' -ongelma

Tilinosto-esimerkissä esim. seuraava ajoitus:

T1	T2
read_item(X, v);	
v:=v-500;	
	read_item(X, u);
	u:=u-1000;
write_item(X, v);	
commit;	
	write_item(X, u);
	commit;

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

Ongelma kuuluu luokkaan 'toistokelvoton luku':

read_item(X,u) tuottaa T2:ssa ennen T1:n tekemää päivitystä eri arvon kuin se tuottaisi päivityksen jälkeen suoritettuna.

2° tilapäisen päivityksen ongelma

```

T1                T2
read_item(X, v);
v:=v-500;
write_item(X, v);

                read_item(X, u);
                u:=u-1000;
                write_item(X, u);
                commit;

abort;
```

Tässä on kysymyksessä 'likainen luku' (dirty read): T2 lukee T1:n 'tilapäisesti' päivittämän arvon, joka kuitenkin peruuntuu T1:n päättyessä abort-operaatioon.

(tulos = 500, vaikka vain T2 toteutuu)

3° Koosteoperaation suoritus muiden transaktioiden rinnalla

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.

Esimerkit rikkovat transaktioiden eristyvyyttä vastaan.

Eristyvyyserikkomuksissa on kolme päätyyppiä eli eristyvyysanomalia:

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

3° toistokelvoton luku (unrepeatable read)

toinen transaktio kirjoittaa T1:n lukeman tietoalkion ennenkuin T1 sitoutuu

Likainen kirjoitus:

```

T1: write-item(X, u);
...
T2: write_item(X, v);
...
T1: commit;
```

T2:n write_item on tässä likainen. Jos T1 ei sitoudu, vaan suorittaa rollback-toiminnon, write_item on silti likainen.

Esim. Tietokannan eheysrajoite: $X = Y$
T1 asettaa $X := Y := 1$; T2 asettaa $X := Y := 2$
(kumpikin transaktio on siis selvästi oikeellinen)

Seuraava ajoitus rikkoo tietokannan eheyden (lopputulos: $X = 2, Y = 1$):

```

T1: u:=1;
T1: write_item(X, u);
    T2: v:=2;
    T2: write_item(X, v);
    T2: write_item(Y, v);
    T2: commit;
T1: write_item(Y, u);
T1: commit;
```

T2:n operaatio write_item(X, v) on likainen kirjoitus (muut kirjoitusoperaatiot ovat puhtaita).

Likainen luku:

transaktio T1 lukee toisen transaktion T2 kirjoittaman tietoalkion ennenkuin T2 sitoutuu tai peruuntuu

```

T1: write_item(X, u);
...
T2: read_item(X, v);
...
T1: commit;      (tai T1: rollback)
```

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_item(X, u);
    T2: read_item(X, v);
    T2: write_item(Y, v);
    T2: commit;
T1: u := 1;
T1: write_item(X, u);
T1: commit;
```

Tässä T2:n lukuoperaatio on likainen, koska T1 ei ole lukuhetkellä sitoutunut (ja on kirjoittanut T2:n lukeman arvon).

Myös seuraava ajoitus rikkoo tietokannan eheyden:

```
T1: u := 0;
T1: write_item(X, u);
   T2: read_item(X, v);
T1: rollback;
   T2: write_item(X, v);
   T2: commit;
```

Tulos: X = 0, transaktiot yksinään ovat oikeellisia.

Toistokelvoton luku:
ks. tilinostoesimerkin menetetty päivitys

SQL2:ssa on lause SET TRANSACTION, jolla sovellusohjelmassa voidaan valita aloitettavan transaktion eristyvyystaso (isolation level). Mahdolliset tasot vaativuudeltaan nousevassa järjestyksessä:

- 1) lue sitoutumatonta (read uncommitted):
transaktio saattaa lukea liikaista tietoa tai toistokelvottomasti, mutta ei kirjoita liikaista
- 2) lue sitoutunutta (read committed)
transaktio saattaa lukea toistokelvottomasti, mutta ei kirjoita eikä lue liikaista
- 3) toistokelpoinen luku (repeatable read)
transaktio ei kirjoita eikä lue liikaista eikä lue toistokelvottomasti
- 4) sarjallistuva (serializable)
kuten 3; lisäksi ns. haamuilmiöiden esiintyminen on kielletty

Oletustaso on standardissa sarjallistuvuus; käytännössä esimerkiksi Oraclessa taso 2.

Eristyvyystaso on yhteydessä samanaikaisuuden hallinnan käytäntöön, esim. lukitusperiaatteeseen. Ankar (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):

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

(Oraclessa asetus siis kireämpi kuin oletus)

Haamuilmiö: erikoistapaus, joka syntyy, kun tietokantaan lisätään transaktiossa T rivi, joka täyttää toisen transaktion T' käsittelemien rivien valintaehdon. Esim.

```
T: insert into employee values (... , dno=5)
T': select sum(salary) ... where dno=5
```

Ajoitus (T, T') ottaa mukaan myös uuden työntekijän palkan, ajoitus (T',T) sitävastoin ei. Jos T' ehtii ottaa relaation käyttöönsä ennen lisäystä, kesken laskennan ilmestyvä uusi rivi on ns. haamutietue.

Lukituskäytäntö

Samanaikaisuuden hallinnan sisältämä kontrolli transaktioiden suoritukselle (eristyvyys takaanen) voidaan hoitaa useilla menetelmillä:

- asettamalla tietoalkioille lukkoja (locks):
operointi on sallitu 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)

Lukitusmenetelmä on yleisimmin käytössä.

Lukko (lock) on tietokannan käyttöä valvova muuttuja.

Lukkoja on erityyppisiä:

- lukulukko (read lock; shared lock) antaa oikeuden lukea tietokannan, mutta ei kirjoittaa sitä
- lukulukko tiettyyn tietokantaan voi samanaikaisesti olla usealla transaktiolla (lukuoperaatiot eivät häiritse toisiaan, 'shared')
- kirjoituslukko (write lock, exclusive lock) antaa oikeuden kirjoittaa (ja lukea) tietokannan arvon
- kirjoituslukko on poissulkeva, 'yksityinen': vain yhdellä transaktiolla voi olla samanaikaisesti kirjoituslukko tietokantaan X
- muilla ei voi olla edes lukulukkoa tietokantaan X

(muitakin lukkotyyppisiä on)

Lukkojen käyttöön liittyviä operaatioita

read_lock(X): lukulukon pyyntö
 write_lock(X): kirjoituslukon pyyntö
 unlock(X): X:n lukon vapautus

valvoo tkh:n lukonhallitsin (lock manager).

Transaktiolla on enintään yksi lukko tietokantaan kerrallaan.

Keskeiset tietorakenteet:

- lukkotaulu (lock table), joka on organisoitu tietokantaan kohtaisesti:
 - tietokannan X tunniste
 - X:n lukon haltijoiden tiedot: transaktio (tunniste) ja lukon tyyppi
 - X:n lukkoa haluavien transaktioiden jono
- transaktiotaulu: jokaiselle transaktiolle tietue, josta alkaa transaktion hallussa olevien lukkojen (tietokantojen tunnisteiden) ketju

Lukkotaulusta saadaan nopeasti selville tietokannan lukituksen tilanne. Organisoitina voi olla esim. hajautusrakenne.

Transaktiotaulun avulla löydetään transaktion sitoutuessa tai peruuntuessa sen hallussa mahdollisesti olevat lukot, jotka on vapautettava.

Lukko-operaatioiden toiminta (suorittajana 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 $rl(X) := rl(X) + 1$

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 $rl(X) := rl(X) - 1$; muuten kirjaa T:lle uusi lukko, kirjoituslukko, lukkotauluun

Askeleeseen 4 sisältyy siis mahdollinen lukon odotus, joka päättyy, kun lukonhallitsin vapauttaa vastaavan lukon ja 'herättää' transaktion.

unlock(X):

1. etsi X:ää vastaava tietue lukkotaulusta
2. jos X:ään on kirjoituslukko transaktiolla T, poista T lukonhaltijain joukosta ja herätä ensimmäinen odottavista transaktioista, jos niitä on; 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

Herätetty transaktio jatkaa siis suoritustaan read_lock- tai write_lock-operaationsa askeleesta 4.

Lukon konversio voitaisiin periaatteessa suorittaa myös 'alaspäin': muuttamalla (downgrade) kirjoituslukko lukulukoksi.

Lukitusoperaatiot (lukonpyynnöt) read_lock(X) ja write_lock(X) voidaan ajatella vastaavia luku- ja kirjoitusoperaatioita read_item(X,v), write_item(X,v) edeltäviksi operaatioiksi transaktion suorituksessa. Ne eivät kuitenkaan automaattisesti takaa transaktion sarjallisuutta; tarvitaan hyvin määritelty lukituskäytäntö.

Vastaesimerkki: Olkoon alussa $X = 20$, $Y = 30$ ja seuraava ajoitus:

T1	T2
read_lock(Y);	
read_item(Y);	
unlock(Y);	
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);
write_lock(X);	
read_item(X);	
$X := X + Y$;	
write_item(X);	
unlock(X);	(tulos: $X = 50$, $Y = 50$)

Sarjallisen suorituksen tulos on joko

$X = 50$, $Y = 80$ (järjestys T1, T2)
 $X = 70$, $Y = 50$ (järjestys T2, T1)

(Sarjalliset suoritukset antavat tässä eri tulokset, koska transaktiot eivät ole toisistaan riippumattomia.)

Kaksivaiheinen lukituskäytäntö korjaa tilanteen (s. 49).

Lukkojen varaaminen ja vapauttaminen käytännössä:

Samanaikaisuuden hallinta kuuluu (joko järjestelmän oletuksiin tai määritteleviin asetuksiin perustuen) 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. (Ne löytyvät transaktiotaulun kautta.)

Transaktion päättymiseen asti pidettävä lukko on pitkäaikainen, aikaisemmin eksplisiittisesti vapautettavat lyhytaikaisia. Lyhytaikainen lukko vapautetaan normaalisti heti operaation jälkeen (vrt. edellinen esimerkki).

Esimerkiksi tietohakemistosivuihin ja varsinaisiin hakemistosivuihin (ISAM, B+ -puu) kohdistuvat lukot ovat yleensä lyhytaikaisia.

< tyhjä >

Sovelluksen ohjelmoija vaikuttaa lukkojen varausaikaan yleensä vain epäsuorasti määrittelemällä transaktiot 'sopivan' pituisiksi (mahdollisimman lyhyiksi).

Lukkojen granulaarisuus on tärkeä samanaikaisuuden asteeseen liittyvä tekijä. Lukittava 'tietoalkio' voi olla yksittäinen kenttä, tietue, sivu, taulu tai jopa koko tietokanta. Hienojakoinen granulaarisuus vaatii paljon lukkoja ja monimutkaista lukkojen hallintaa, mutta sallii maksimaalisen samanaikaisuuden. Käytännössä on yleistä rivi-, sivu- tai taulukohtainen lukinta.

Oracle: rivilukkoja ja taululukkoja, paljon erilaisia tyyppejä.

- oletus: rivilukot
- lauseella lock table voidaan säädellä lukinnan laajuutta
row share, row exclusively, share, exclusive, ...
- lukitusalgoritmiin kuuluva odotus voidaan myös estää
(nowait)