Samanaikaisuuden hallinta

tietokantapalvelimessa

Tiedonhallintaa

Alkuper. versio: Jaakko Rantanen

Pieniä korjauksia: Jouni Huotari

Transaktiot eli tapahtuma(sarja)t

- Transaktio (transaction) on DBMSn näkemä sovelluksen l. asiakasprosessin toimenpide (LUW, Logical Unit of Work)
  - atominen sarja luku- ja kirjoitusoperaatioita
- Useiden samanaikaisten asiakasprosessien hallinta on DBMSn keskeisin tehtävä
  - levyoperaatiot tiheitä mutta hitaita → CPUn kannattaa ajaa monia ohjelmia samanaikaisesti
- Huom: *tapahtuma*-sana esiintyy myös *event*-termin suomennoksena! Siis aivan eri asian.

Esimerkki

SQL API

```
select...update...select..
. update...   commit;
```

LUW = tilisiirto

Mitä DBMS tekee tämän pohjalta:

aloita

lue tilin tA saldo sA

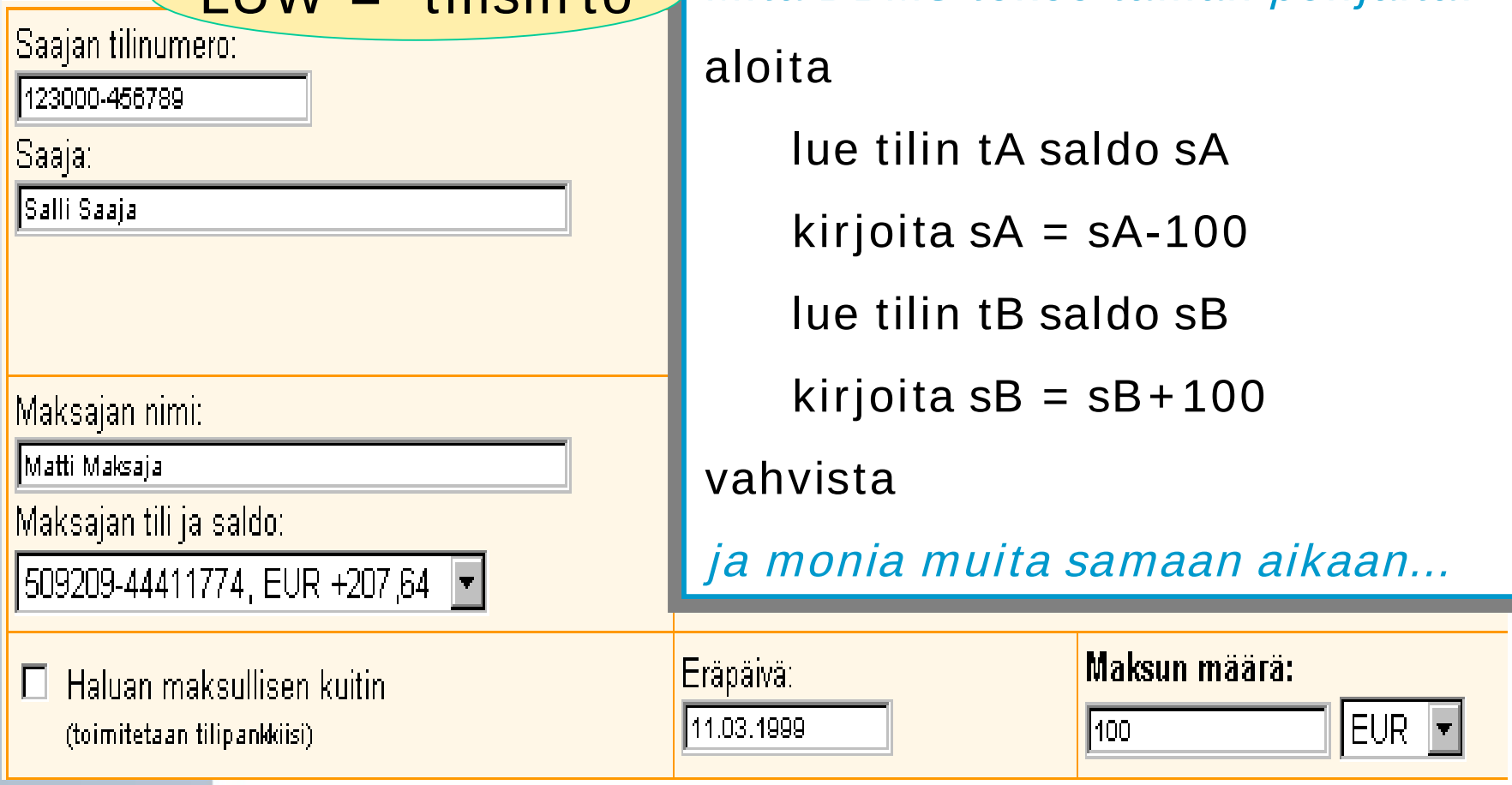
kirjoita sA = sA-100

lue tilin tB saldo sB

kirjoita sB = sB+100

vahvista

ja monia muita samaan aikaan...



Atomisuus

- Transaktio suoritetaan aina jakamattomana eli atomisena (atomic)

Kaikki tai ei mitään!

Samanaikaisuus (concurrency)

- Asiakasprosessi lähettää transaktion ja voi olettaa, että se ajetaan "kuin yksinäisenä"
- Samanaikaisten asiakasprosessien luku/kirjoitus-operaatiot lomitetaan:
  - LUW-tulosten oltava samat kuin yksin ajettuna
  - Tietokannan jokaisen eheysehdon pitää täyttyä
- Ydinasiat: lomittaminen ja vikasietoisuus
- Samanaikaisuusaste (kpl-määrä) on yksi tärkeä tietokantapalvelimen tehon mitta

Esimerkki

SQL API

```
select...update...select..
. update...   commit;
```

LUW = tilisiirto

Transaktio TX1

aloita	merkintä
lue tilin tA saldo sA	R(A)
kirjoita sA = sA-100	W(A)
lue tilin tB saldo sB	R(B)
kirjoita sB = sB+100	W(B)
vahvista	

Saajan tilinumero:
123000-456789
Saaja:
Salli Saaja

Maksajan nimi:
Matti Maksaja
Maksajan tili ja saldo:
509209-44411774, EUR +207,64

☐ Haluan maksullisen kuitin
(toimitetaan tilipankkiisi)

Eräpäivä:
11.03.1999

Maksun määrä:
100 EUR

Samanaikaiset tehtävät

- TX1 siirtää 100€ B:n tilitä A:n tilille.
- TX2 lisää molempiin 5% korkoja.

```
TX1: begin  A= A+100,  B= B-100   end
TX2: begin  A= 1.05*A,  B= 1.05*B  end
```

- Ei ole mitään tietoa tai takeita näiden transaktioiden suoritusjärjestyksestä. Lopputuloksen täytyy ehdottomasti olla riippumaton suoritusjärjestyksestä.

Ajoitus

- DBMS suorittaa nämä jollakin ajoituksella.
- Ajoituksen (scheduling) vaihtoehtoja:

1

TX1: A= A+100, B= B-100
TX2: A= 1.05*A, B= 1.05*B

Edellinen oli OK, mutta seuraava ajoitus ei:

2

TX1: A= A+100, B= B-100
TX2: A= 1.05*A, B= 1.05*B

Sarjallistuva ajoitus

Ajoitus 1 DBMSn näkemänä:

```
TX1: R(A),W(A),                R(B),W(B)
TX2:           R(A),W(A),                R(B),W(B)
```

Ajoitus 2 DBMSn näkemänä:

```
TX1: R(A),W(A),                          R(B),W(B)
TX2:           R(A),W(A), R(B),W(B)
```

Jälkimmäistä ajoitusta ei saa sallia.

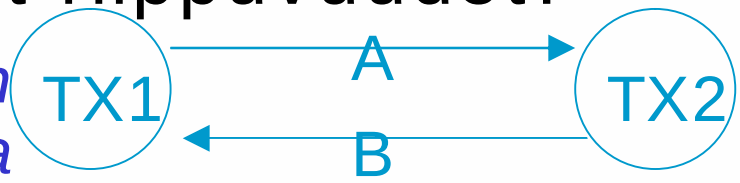
Ajoitukset jakautuvat siis sallittuihin eli sarjallistuviin ja kiellettyihin.

Ristiriidan välttäminen

Millaiset ajoitukset DBMSn on kiellettävä?

TX1: R(A),W(A), R(B),W(B)
TX2: R(A),W(A), R(B),W(B)

- Transaktioiden väliset riippuvuudet:



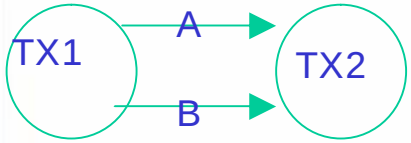
TX1:n tuloksen sanotaan riippuvan TX2:sta, koska TX1 käyttää tietoa (tieto B)*, jonka TX2 on viimeksi kirjoittanut*. (Esimerkissä on myös käänteinen riippuvuus TX1→TX2)

- Syklinen riippuvuus → ajoitus on kielletty.

Riippuvuudet ja ajoitus

Ajoitus 1 DBMSn näkemänä:

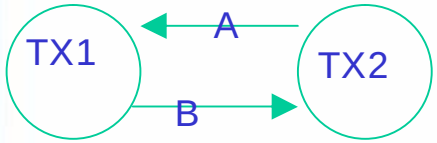
```
TX1: R(A),W(A),                R(B),W(B)
TX2:           R(A),W(A),                R(B),W(B)
```



TX2 riippuu TX1stä

Ajoitus 2 DBMSn näkemänä:

```
TX1: R(A),W(A),                          R(B),W(B)
TX2:           R(A),W(A), R(B),W(B)
```



Sykli: TX1 riippuu TX2sta
ja TX2 riippuu TX1stä

Sarjallistuvuus

TX1, TX2, TX3

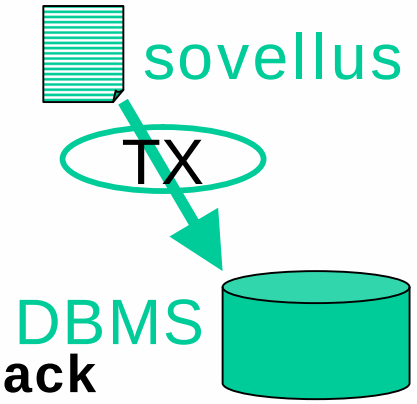
- Sarjallinen (serial) ajoitus toimii kuin "köyhän talon porsaat", mutta siinä on vain yksi samanaikainen asiakas.
- Tapahtumajoukon TX1, TX2,... ajoitus on sarjallistuva (serializable), jos se tuottaa saman eheän lopputuloksen kuin *jokin sarjallinen* ajoitus (kaikista eri järjestysvaihtoehdoista).

  Voi olla monta samanaikaista asiakasta.

  Tällä on vankka teoreettinen tausta, jota ei nyt tarkastella lähemmin.

Transaktio sovelluksessa

- Transaktion alku on joko
  - eksplisiittinen: komento **`begin transaction`**
  - implisiittinen: ei komentoa, vaan
    - ohjelman(osan) käynnistys tai
    - edellisen transaktion päättyminen
- Transaktio päättyy joko
  - vahvistukseen: komento **`commit`**
  - peruutukseen: komentoon **`rollback`**
    tai DBMSn suorittamaan automaattiseen peruutukseen, jos ei vahvistusta tule



Transaktioita käytettävä

- Mm. ODBC ja JDBC, Accessin Jet Engine,... käyttävät autocommit -tilamuuttujaa, jonka arvot ovat on tai off. Oletusarvo on on.
- Autocommit ei sovi normaaliin sovellukseen!
- Yleensä on annettava aluksi komento
  `set autocommit off`

Miten kävisi esimerkissä?

Saajan tilinumero:
123000-456789
Saaja:
Salli Saaja

Maksajan nimi:
Matti Maksaja
Maksajan tili ja saldo:
509209-44411774, EUR +207,64

Haluan maksullisen kuitin
(toimitetaan tilipankkiisi)

Eräpäivä:
11.03.1999

Maksun määrä:
100
EUR

set autocommit on !!!

lue tilin tA saldo sA

kirjoita sA = sA-100

lue tilin tB saldo sB

linjahäiriö/system crash/...

~~kirjoita sB = sB+100~~

Samanaikaisuuden hallinnan toteutustekniikoita

- DBMSien tarjoamat toteutustavat:
  - lukitus
  - versiointi
- Itse ohjelmoitu toteutustapa:
  - aikaleimojen hallinta
    - tarpeelliseksi katsotut tiedon osat (esim. rivi) varustetaan aikaleima-sarakkeella (on myös ylläpidettävä)
    - jokaisen luku- ja päivityskäskyn yhteydessä päätellään aikaleimoista onko joku muu ollut käsittelemässä ko. tietoja samaan aikaan

Tekniikkana lukitus

- DBMS voi hallita samanaikaisuutta esim. lukitsemalla tietoja (nk. pessimistinen menetelmä).
- Lukon piirteitä: aste (yksityinen/jaettu/...), rakeisuus (predikaatti/rivi/sivu/taulu), ym.
  - lukulukko on jaettu (S shared), muut saavat lukea,
  - kirjoituslukko yksityinen (X exclusive), poissulkeva.
- Käytännössä vanhin ja yleisin tekniikka
- Tuotteiden toteutuksissa merkittäviä eroja mm. tehokkuudessa ja muistin käytössä

Lukkojen hallinta

Eri DBMSt noudattavat erilaisia tekniikoita

- Lukkokirjanpito erikseen DBMSn muistialtaassa
  - n. 100B/lukko → syö paljon muistia, jos on rivikohtainen (on useilla DBMSilla sivu- tai taulukohtainen)
  - lukkojen laajentamisen riski (lock escalation)
- Lukkokirjanpito riviin vietynä (Oraclen tapa)
  - rivikohtainen, ei laajentamista

taulurivejä muistissa

[ots 1672] sarake1 sarake2 ...
[ots 0752] sarake1 sarake2 ...

pros. no
...
1009
0752
2137
...

Lukkiuma

- Lukkiuma eli lukkiutuma (deadlock) voi syntyä, jos nk. odotusverkossa on sykli:

nk. "wait-for-graph"

TX1 on lukinnut tiedon X ja haluaisi tiedon Y
TX2 on lukinnut tiedon Y ja haluaisi tiedon X

- Syklissä voi olla useitakin transaktioita
- Purku: DBMS valitsee uhri-transaktion ja pysäyttää sen

Kaksivaiheinen lukitseminen

- Kaksivaiheinen lukituskäytäntö (2PL = two phase locking) tarkoittaa lukkiumia torjuvaa ohjelman suoritustapaa
- Transaktio TX noudattaa 2PL-käytäntöä, jos kaikki lukitusoperaatiot (sekä S että X) TXssa edeltävät ensimmäistä lukon vapautusta
- Miten tätä noudatetaan käytännössä: kaikki lukot pidetään **`commit`**iin asti.
  Tämä on nk. ankara (strict) 2PL.

Tekniikkana versiointi

- Jokainen TX käsittelee omaa versiotaan (eli omia kopioita) taulusivuista
- Vasta commit-hetkellä DBMS ilmoittaa mahdollisen konfliktin: että joku muu on ehtinyt muuttaa "originaali"sivua
- Periaate: nopein voittakoon, muut aloittakoot alusta
- Versiointi on nk. optimistinen menetelmä ("ei kai nyt tule konfliktia"), lukinta pessimistinen ("kumminkin nyt tulee konflikti")

DBMS-toteutuksista

Lukinta on yleisintä
- Oracle
- DB2
- Sybase
- Informix
- SQL Server
- OpenIngres
- Interbase
- Progress
- ...

(Oracle tekee osittaista versiointia.)

- Versiointi on uutta ja vielä harvinaista
  - Solid Server
    Lukinta on valittavissa vaihtoehtoisesti. Solidissa voi taulukohtaisesti sanoa käytetäänkö lukkoja vai versiointia. On hyvin vaarallista sotkea näitä samaan tietokantaan!
  - Mimer (ruotsalainen)
  - ...

ACID-periaate

- Atomicity: Transaktio on jakamaton, joko kaikki komennot suoritetaan tai ei yhtään.
- Consistency: Jos tietokanta on aluksi ristiriidaton ja jokainen TX on sarjallistuva, tietokanta jää ristiriidattomaksi.
- Isolation: Transaktion suoritus eristetään muista samanaikaisista transaktioista.
- Durability: Jos transaktio vahvistetaan, sen tekemät päivitykset jäävät pysyviksi.

Muita seikkoja

- Lokin hallinta, Checkpoint
- Transaktioiden sisäkkäisyys ja ketjutus
- Pitkät tapahtumat
- Hajautetut transaktiot
- Miksi juuri SQL-palvelin
- Myös ODMG noudattaa SQL-standardin määrittelyjä, mm. eristyvyystasoja