5. Tapahtumien hallinta

= transaction management (yleistä: E&N, Ch. 19)
- kaikkien tietokantajärjestelmien keskeinen osa
- erityisen tärkeä tapahtumapohjaisissa järjestelmissä
  - varausjärjestelmät, pankkitilijärjestelmät, kaupankäynti, varastonvalvonta, ...
  - monia käyttäjiä samanaikaisesti
  - tiukat vasteaikavaatimukset
  - jatkuva käytettävyysvaatimus

Tkhj:n tapahtumanhallinta takaa kokonaisuuden häiriöttömän toiminnan:
- **kontrolloi** samanaikaista käyttöä (→ 5.2)
- **elvyttää** järjestelmän tilapäisistä häiriöistä (→ 5.1)

- vrt. käyttöjärjestelmän prosessinhallinta, jonka päälle tkhj ja tapahtumanhallinta rakentuu

Tietokantatapahtuma (transaktio) on tietokantaa käsittelevä prosessin osa, jonka vaikutusten halutaan muodostavan yhden jakamattoman (atomisen) kokonaisuuden.
- tietojen hakuja, lisäyksiä, muutoksia, poistoja
- joko sovellusohjelman osa tai joukko peräkkäisiä SQL-operaatioita vuorovaikutteisessa käytössä

Esim. pankkitilisovelluksen proseduuri
tilisiirto(t1, t2, x),
joka siirtää x mk tililtä t1 tilille t2:

```
begin transaction
    update tili set saldo=saldo-x where tilinro=t1;
    update tili set saldo=saldo+x where tilinro=t2;
    insert into tilitapahtumat values
        (pvm, time, 'siirto', x, t1, t2, ...);
    commit;
end transaction;
```

Atomisuus tässä: kaikki kolme päivitystä toteutuvat ja jäävät voimaan (sitoutunut suoritus) tai mikään niistä ei jää voimaan (peruuntunut suoritus).
'commit' on transaktion päättävä lause, oikeastaan lopetuspyyntö
(vaatii tarkistuksia, onnistuuko)

Huom.
Levymuistia käytettäessä tietoja **ei välttämättä** viedä **levylle heti** operaation yhteydessä (levyliikenteen minimointi, varautuminen yhteiskäyttöön).

Pankkitilijärjestelmässä on voimassa eheysrajoite, jonka mukaan tilitapahtumat-relaatiossa tulee olla kirjaukset, jotka vastaavat tili-relaation tilien saldoja. (Tili-relaation kannalta tilisiirron oikeellisuus merkitsee sitä, että rahaa ei häviä eikä tule lisää.)

Mikä voi uhata?
- proseduurin suoritus voi jäädä kesken esim. update- ja insert-operaatioiden välissä (esim. ohjelmisto- tai laitteistovirhe)

Teknisesti kesken jääminen voi tarkoittaa erilaisia tilanteita:
- oletetaan, että tilin t1 ja tilin t2 tietueiden tallennuspaikat ovat eri sivuilla p1, p2 (levymuistissa)
  - jos kumpaakaan sivua (muutettua tietuetta) ei ehditty kirjoittaa levylle, mitään virhettä ei ole eikä siis tarvitse korjata
  - jos joko p1 tai p2 tai molemmat kirjoitettiin heti levylle, tietokanta jää epäeheäksi ja pitää korjata

(tietokanta voi jäädä epäeheäksi myös, vaikka häiriö sattuisi aivan transaktion lopussa, insert-lauseen jälkeen)

Transaktion käsite on siis looginen käsittelykokonaisuus, jolla on alku ja loppu. Ohjelma (tai välitön tietokannan käyttö) jakaantuu yleensä moneen transaktioon:

- ohjelmassa usein
  begin transaction ..... end transaction
- yleisesti: ... commit; ..... commit; ..... commit;
  (edellisestä commit-operaatiosta seuraavaan)

Transaktion yhteydessä otetaan yleensä huomioon vain ne transaktion operaatiot, jotka kohdistuvat välittömästi tietokantaan (sekä aloitus, lopetus ja eräät erityistoimet):

esim. tilisiirto (1234, 5678, 5000):

1. transaktion aloitus(pyyntö) begin
2. lukuoperaatioita tietohakemiston sivuihin
3. lukuoperaatioita tili-relaation hakemistoon
4. luetaan puskuriin b se tili-relaation sivu p,
   jossa on monikko 1234
   --- ohjelmallisesti varsinainen tililtäotto
   ja siihen mahdollisesti liittyvät
   tarkistukset + lisätoimet ---
5. kirjoitetaan muutetun monikon sisältö sivulle p
6.-9. vastaavat operaatiot tilin 5678 sivulle (tililleppano)
10. lukuoperaatioita tietohakemistoon
11. luetaan tilitapahtuma-relaation viimeinen sivu
    (oletusrakenne kasa)
12. kirjoitetaan muutettu sivu
13. transaktion sitoutumispyyntö (commit).

Tarkastelu yleistetään siten, että tietokantaan operoidaan vain kahdentyyppisillä operaatioilla:

read_item(X, v): lue tietoalkion X arvo ohjelman muuttujaan v

write_item(X, v): vie muuttujan v arvo tietoalkiolle X

(Huom. E&N: read_item(X), write_item(X))

X on tietoalkion suora osoite (vaihtelee riippuen alkiosta). Hakemistosivulle oma read_item( ) ...

Tietoalkio voi tässä olla
- tietueen kenttä; X on kentän osoite (tid, faddr)
- koko tietue; X on tid (tuple identifier, joka sisältää myös sivun tunnisteen)
- tietokannan sivu; X on sivun tunniste

Tietoalkion koko määrittelee tarkastelun (alkion) granulaarisuuden (hienojakoisuus). Sillä ei ole toistaiseksi merkitystä transaktiokäsittelyn logiikalle.

Kyselyä käsiteltäessä tarvitaan luku/kirjoitus-operaatioita mahdollisille hakemistoille ja perustiedostolle.

Esim. oletetaan, että relaatiolle R(A, B, C) on harva ISAM-hakemisto avaimella A. Tietokantaohjelman osa

```
begin transaction
insert into R values (a, b, c);
commit;
end transaction;
```

generoi seuraavan transaktion:

1. aloituspyyntö (begin)
2. tietohakemiston sivuihin p kohdistuvia operaatioita read_item(p, ...)
3. ISAM-hakemiston sivuihin kohdistuva operaatiojono read_item(p_1, v_1), ... , read_item(p_d, v_d), missä p_1, ... , p_d on A-attribuutin arvoon a johtava hakemistopolku ja d hakemiston korkeus
4. perustiedoston sivun p_a luku read_item(p_a, va)
5. operaatio write_item(p_a,va) (mikäli sivulla p_a on tilaa lisätylle riville; muuten lisätoimintoja)
6. transaktion sitoutumispyyntö (commit)

Varsinaisen toiminnallisen operaation ja puskurien kannalta luku ja kirjoitus tarkoittavat siis seuraavaa:

read_item(X, v):

1. kiinnitetään X:n sivu p_x puskuriin: B:= bufferfix(p_x);
2. v:= puskurisivulla B oleva X:n arvo;
3. vapautetaan puskurisivu B: bufferunfix(B); (vapautetaan puskurin kytkentä, ei välttämättä heti tilaa)

write_item(X, v):

1. kiinnitetään X:n sivu p_x puskuriin: B:= bufferfix(p_x);
2. X:=v (puskurisivulla B oleva X:n vastine);
3. merkitään puskurisivun p_x muuttuminen (puskurin kontrollitietoihin tms. tietorakenteeseen)
4. vapautetaan puskurisivu B: bufferunfix(B);

(lisäksi kontrollitoimia esim. samanaikaisuuden hallinnan ja elvytyksen toteuttamiseksi)

Huom. write_item() ei siis kirjoita välttämättä levylle asti.

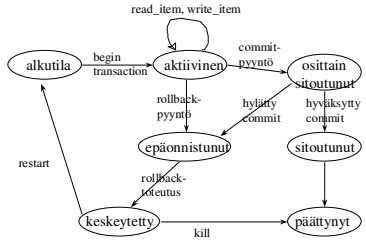
Transaktion sisäisiä vaiheita hallitaan määrittelemällä transaktion tilasiirtymämalli ja seuraavat tilat:

1. alkutila: transaktio syntyy (generoidaan) - oma tunniste jne
2. aktiivinen: transaktio suorittaa varsinaisia operaatioitaan (read_item(), write_item())
3. osittain sitoutunut (partially committed): transaktion ohjelmakoodi on suoritettu ja se on pyytänyt sitoutumista (commit-operaatiolla)
4. sitoutunut (committed): tkhj on vahvistanut transaktion tietokantaan tekemät muutokset pysyviksi eli sitoutuminen on onnistunut

   Tietokannan muutokset eivät ole enää peruttavissa (ilman uutta transaktiota). Tiedot eivät kuitenkaan ole välttämättä levyllä asti (vahvistus vain 'looginen'), mutta vastaava lokitieto on (vrt. elvytys, kohta 5.1).

5. epäonnistunut (failed): sitoutuminen on epäonnistunut (esim. samanaikaisuuden hallintaan liittyvien tarkistusten takia) tai transaktio itse on suorittanut keskeytysoperaation rollback (abort)
6. keskeytetty: epäonnistunut transaktio on peruutettu eli tietokanta on palautettu ennen transaktion aloitusta vallinneeseen tilaan
7. päättynyt (terminated): transaktion olemassaolo lakkaa

Keskeytetty transaktio voidaan vielä aloittaa uudelleen (restart), päättynyttä ei (tarvitaan uusi).
(E&N ei erota keskeytetty-tilaa)



SQL ja transaktiot

Transaktio päätetään normaalisti commit-lauseella ja se alkaa välittömästi edellisen transaktion loputtua. Upotettu SQL: transaktion rajat voidaan ilmaista eksplisiittisesti: begin transaction ... end transaction.

Vaihtoehtoinen tapa päättää transaktion suoritus on rollback-lause, jolla pyydetään, että tkhj peruuttaa transaktion tietokantaan tekemät muutokset.

Rollback-lauseen tarve seuraa jostakin ohjelmalogiikkaan (tai syötteisiin) liittyvästä virhetilanteesta, joka selviää vasta, kun tietokantaan on jo tehty muutoksia.
Rollback peruuttaa pääsääntöisesti koko transaktion (selkeä tapa, joskin voi olla aika järeä - erityisesti, jos transaktio on pitkä).

Huom. Tkhj voi itse aiheuttaa rollback-toiminnan (undo) häiriöistä elvytyksen yhteydessä.

- - - - -

Transaktion osittainen peruutus on mahdollista, jos sen sisälle voidaan määritellä ns. jatkoaloituskohtia (savepoint):

```
savepoint p1   ('tallenna tilanne, nimeä kohta = p1)

rollback to savepoint p1    (peruuta transaktiossa
                                pisteeseen p1)
```

Ohjelmoija voi periaatteessa kontrolloida transaktion suoritusta vaiheittain jatkoaloituskohtia määrittelemällä ja rollback-lauseilla. Se on käytännössä hankalaa ...

(if …. rollback to savepoint p1)

(yksi kohta on yksinkertainen hallita, yleisesti ei!)


```
SQL> create table t (x integer);
SQL> savepoint piste1;
SQL> insert into t values (111);

SQL> savepoint piste2;

SQL> insert into t values (222);
SQL> select * from t;

         X
-----------
       111
       222

SQL> savepoint piste3;

SQL> update t set x=444 where x=111;

SQL> select * from t;

         X
-----------
       444
       222
```



```
SQL> rollback to savepoint piste3;

Tapahtuma on peruutettu.    (pisteeseen piste3 asti)

SQL> select * from t;

         X
-----------
       111
       222

SQL> rollback to savepoint piste2;

Tapahtuma on peruutettu.    (pisteeseen piste2 asti)

SQL> select * from t;

         X
-----------
       111

SQL> commit;
Tapahtuma on vahvistettu.   (lopullisesti)

SQL> rollback to savepoint piste1;   (ei onnistu)

SQL> select * from t;

         X
-----------
       111
```

- - - - - -

Transaktion perusominaisuudet (ACID-ominaisuudet):

Tietokannan tila (state) = sen kaikkien tietoalkioiden arvojen yhdistelmä tietyllä ajanhetkellä. Tila muuttuu päivitysoperaatioilla; transaktio on yksittäistä päivitysoperaatiota tärkeämpi 'kontrolliyksikkö' tilanmuutosten kannalta.

Tietokannan tila on eheä ('konsistentti'), kun tietokannalle määritellyt eheysvaatimukset ovat voimassa.

1. Atomisuus (jakamattomuus, Atomicity)
   Transaktion aikaansaamat muutokset tietokannan tilaan muodostavat jakamattoman kokonaisuuden: joko kaikki toteutuvat tai ei mikään.
2. Oikeellisuus eli eheyden säilyminen (Consistency preservation)
   Transaktio säilyttää tietokannan eheyden eli 'vie tietokannan eheästä tilasta eheään tilaan'.
   Eheyden säilyminen vaatii, että tkhj:n eheyskontrolli on kunnossa ja että sovellusohjelman toiminnot eivät riko eheyttä ('ohi' tkhj:n tai jos tkhj:n eheyskontrolli on väljä). Oikeellisuuden säilymisessä ei oteta huomioon muiden transaktioiden vaikutusta eli riittää, että toiminta on oikeaa 'häiriöttömässä tilassa'.
   ("yksinään suorittamisen oikeellisuus")

3. Eristyvyys (Isolation)
   Transaktio suoritetaan ikäänkuin muita transaktioita ei olisi samanaikaisesti käynnissä. Kun niitä kuitenkin on, eristyvyys asettaa vaatimuksen, etteivät muiden transaktioiden operaatiot saa vaikuttaa tietyn transaktion suoritukseen (haitallisesti). Transaktion T kannalta näyttää, että jokainen muu transaktio T' olisi suoritettu kokonaisuudessaan ennen T:tä tai vasta sen jälkeen.

4. Pysyvyys (Durability, permanency)
   Kun transaktio on suoritettu onnistuneesti loppuun (sitoutunut), sen muutokset tietokannan tilaan jäävät pysyvästi voimaan. Pysyvyys on suhteessa valmiiseen transaktioon: mikään häiriö ei saa enää muuttaa tilaa; uusi transaktio voi tietysti muuttaa tietoalkioiden arvoja. - myös: "kompensoiva" transaktio

Tkhj:n elvytysalijärjestelmä (recovery subsystem) huolehtii transaktioiden atomisuudesta ja pysyvyydestä.

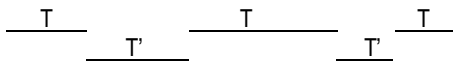
Tkhj:n samanaikaisuuden hallinnan alijärjestelmä (concurrency control subsystem) huolehtii eristyneisyydestä.

Transaktion oikeellisuuden säilyminen on tietokannan suunnittelijan ja tk-sovelluksen ohjelmoijan vastuulla.

Samanaikaisten toimintojen salliminen ja häiriöiden vaikutusten eliminointi muodostavat monimutkaisen toiminnallisen kokonaisuuden. 'Pelisääntönä' on karkeasti se, että samanaikaisuutta rajoitetaan vain niin paljon, ettei 'normaalitilanteessa' jouduta usein korjaamaan sen aiheuttamia (tulossa olevia) vaurioita.

Menetelmiä (5.2):
- lukitaan tietoalkioita muilta lukoilla (lock);
  pitkä lukinta-aika rajoittaa hankalasti muiden transaktioiden etenemistä;
  voi syntyä lukkiutuma, jolloin mikään transaktio ei pääse etenemään
- optimistisia menetelmiä: annetaan mennä;
  tarkistetaan; korjataan, jos tarpeen

Samanaikaiset toiminnot voivat perustua limittyneeseen eli vuorottelevaan (interleaved) tai (todella) samanaikaiseen (concurrent) suoritukseen. Yleensä tarkastellaan vuorottelevaa suoritusta: transaktioiden T ja T' suorituksen osat vuorottelevat, kumpikin on samanaikaisesti kesken.



Keskeisiä käsitteitä:

Transaktiohistoria eli transaktioiden ajoitusjärjestys (schedule) = eri transaktioiden luku- ja kirjoitusoperaatioiden jono

Yhden transaktion operaatioiden järjestys on kiinteä; suhteessa muihin transaktioihin on useita mahdollisia järjestyksiä ('vuorotteluja').
- käsitellään kohdassa 5.2

Transaktioiden todella suorittamat operaatiot kirjataan lokiin (log), johon mm. häiriöiden korjaus (tietokannan elvytys) perustuu.
- käsitellään kohdassa 5.1

5.1 Tietokannan elvytys (E&N, Ch. 21)

Transaktion atomisuus tai pysyvyys voi vaarantua monen häiriötilanteen takia:

1. Tietokonejärjestelmä 'romahtaa' (system crash) laitteisto-, ohjelmisto- tai tietoliikennevirheen takia. **Yleensä keskusmuistin (tietokantapuskurien) sisältöä menetetään**.

2. Yksittäisen transaktion suoritus keskeytyy ohjelman poikkeustilanteen (div by zero tms.) tai loogisen ohjelmavirheen takia. Käyttäjä voi myös keskeyttää kyselyn suorituksen 'väkivalloin'.

3. Transaktion suoritus keskeytetään hallitusti; esim. transaktion (proseduurin) koodissa suoritetaan jonkin ehdon seurauksena rollback-pyyntö. Esim. ei löydy transaktion tarvitsemaa syötettä tai se on virheellinen.

4. Samanaikaisuuden hallinnan alijärjestelmä joutuu keskeyttämään transaktion, jotta muut transaktiot voisivat edetä (lukkiutuma tai lievempi suoritusjärjestykseen liityvä häiriö).

5. Levyvirhe on turmellut levyn sisältöä.

6. Ulkopuolinen häiriötekijä (operointivirhe, … , sähkökatko) keskeyttää transaktion.

Tyypin 5 ja 6 häiriöt ovat harvinaisia, 'epänormaaleja'. Niiden kohdalla elvytys voi sisältää
- edellisen varmuuskopion (ajanhetkeltä t) käyttöönoton; lokin avulla voidaan mahdollisesti suorittaa uudelleen (redo) hetken t ja häiriöajankohdan välillä suoritetut toiminnot

  Varmuuskopion ja lokin tulisi olla esim. nauhalla tallessa (levyvirhe …).

Tyypin 1-4 häiriöt ovat normaaleja, ne hoidetaan tapahtumanhallinnan toimin.

Lokiin perustuvan elvytyksen periaatteet:
- peruutetaan (undo) keskeytyneiden transaktioiden suorittamien tietokantapäivitysten vaikutukset
- suoritetaan uudelleen (redo) sellaisten sitoutuneiden transaktioiden suorittamat tietokantapäivitykset, joita ei häiriön sattuessa ollut ehditty kirjoittaa levylle (vaan vasta puskurissa olevaan levyjaksoon)

"Normaalitilanteessa" undo- ja redo-toimet kohdistuvat siihen tietokannan tilaan, joka oli häiriön sattuessa voimassa (ei aikaisempaan varmuuskopioon). Lokin (ja mahdollisesti muiden tietojen) avulla etsitään, mitä pitää tehdä.

- varmuuskopioita harvemmin ...

Elvytys voi tapahtua monella tavalla riippuen siitä, mitä on tehty valmiiksi häiriöön mennessä:

ei mitään, viety levylle,
päivitetty vain puskuriin (pitää säilyttää tai ei)

Puskurin sisällön kirjoittamiseen levylle on useita vaihtoehtoja:
- välitön päivitys (immediate update): ennen transaktion sitoutumista
- viivästetty päivitys (deferred update): transaktion sitoutumisen jälkeen
  - pakotus levylle (force): välittömästi
  - no-force: vieläkin myöhemmin (käytännössä viimeistään sitten, kun puskuritilaa tarvitaan muille jaksoille)

No-force -vaihtoehto antaa uusille transaktioille mahdollisuuden käyttää puskurissa olevaa päivitettyä tietoa, jolloin levyhakujen tarve vähenee.

Kirjoitus puskurista levylle voi olla tarpeen puskuritilan käyttämiseksi muuhun tarkoitukseen.

Välittömien päivitysten käytäntö sallii tietokantasivun (sivun paikan) varastamisen (steal). Levylle viety tieto voi olla tällöin likaista (dirty); se muuttuu 'puhtaaksi' vasta, kun transaktio sitoutuu (tiedon oikeellisuus varmistuu). Likaisen tiedon käyttö voi olla mahdollista, mutta vain kontrolloidusti.

Loki on peräkkäistiedosto, johon kirjoitetaan tyypillisesti (aikajärjestyksessä) seuraavanlaisia tietueita:

1) (start, T) transaktion T aloituskirjaus

2) (write, T, x, v1, v2) muutoskirjaus

   Transaktio T on muuttanut tietoalkion x vanhan arvon v1 uudeksi arvoksi v2.
   v1 = arvon x alkukuva (before image)
   v2 = arvon x jälkikuva (after image)

3) (commit, T) transaktion T sitoutumiskirjaus

   Tkhj on vahvistanut T:n eli suorittanut commit-operaation.

4) (abort, T) transaktion T keskeytyskirjaus

Tkhj on peruuttanut T:n eli suorittanut abort-operaation (rollback).

5) (checkpoint) yksittäisestä transaktiosta riippumaton tarkistuspistekirjaus

(Lokiin voidaan tehdä myös lukukirjaukset (read, T, x), mutta niitä ei tarvita elvytyksessä.)

Esimerkki: s. 25.

Lokin muutoskirjauksessa (muutostietueessa) esiintyvä tietoalkio x voi olla periaatteessa mitä tahansa yksittäisestä merkistä koko sivuun. Relaation rivi on tässä yleinen alkio; lokitiedosto vie näin paljon vähemmän tilaa kuin kokonaisten muutettujen sivujen kirjaus.

Lokitiedostoa käsitellään kuten muitakin tiedostoja, ts. lokitietue viedään ensin puskurisivulle, sitten aikanaan esim. puskurisivun täyttyessä tai viimeistään transaktion sitoutuessa (jaksokohtaisesti) levylle.
Myös 'pakkokirjoitus'; vrt. sitoutumiskäytäntö (s. 22).

Häiriön sattuessa vain levyllä oleva lokin osa on varmuudella käytettävissä.
(Yleensä lokin käytettävyyttä varmistetaan vielä levyvirheiden varalta esimerkiksi useamman tiedoston vuorottelulla, ja varmistamalla levyllä oleva loki ajoittain nauhalle (tai toiselle levy-yksikölle)).

Lokin merkitys sitoutumisessa:
- transaktio T on sitoutunut silloin ja vain silloin, kun lokista löytyy tietue (commit, T)
- levyllä olevien lokitietueiden perusteella selviää,
  - mitkä transaktiot olivat häiriön sattumishetkellä sitoutuneita ja mitkä kesken: commit, abort (+ start)
  - mitä on kirjoitettu (+ vanha arvo): write-tietue

Commit-pyynnön toteuttaminen aiheuttaa tkhj:ssa joukon samanaikaisiin transaktioihin liittyviä tarkistuksia. Jos sitoutuminen voidaan tehdä, suoritetaan ns. sitoutumiskäytäntö (commit protocol):

1) kirjoitetaan lokiin tietue (commit, T)

2) pakkokirjoitetaan loki levylle
(kaikki lokin jaksot, jotka ovat toistaiseksi vasta puskureissa)

3) vapautetaan transaktion T mahdollisesti varaamat resurssit (mm. samanaikaisia toimintoja säätelevät lukot transaktion käsittelemiin tietoalkioihin).

Huom. Päivitettyjä datajaksoja ei pakoteta levylle, ts. lokia viedään levylle aina 'ennen dataa'. Tätä sanotaan WAL-käytännöksi (write-ahead-logging), ja sitä noudatetaan myös kirjoitettaessa tietosivuja levylle välittömien päivitysten yhteydessä.

(Mitä tapahtuisi, jos tehtäisiin toisin päin, loki datan jälkeen?)

Tarkistuspiste (checkpoint) lokissa:

tarkoituksena on viedä levylle asti kaikki puskureissa olevat tietokantasivujen päivitykset; seuraavien häiriöiden yhteydessä tarvittavat elvytystoimenpiteet vähenevät

Tarkistuspiste sisältää seuraavat toiminnot:

1. estetään väliaikaisesti transaktioiden suoritus
2. pakkokirjoitetaan kaikki transaktioiden päivittämät sivut puskurista levylle
3. kirjoitetaan lokiin tarkistuspistekirjaus ja pakkokirjoitetaan loki levylle
4. sallitaan transaktioiden jatkaa suoritustaan

Tarkistuspisteiden taajuus on tietokannan hoitajan päätettävissä. Järkevä taajuus riippuu tietokannan päivitystiheydestä ja häiriöalttiudesta:
- esim. 4 kertaa tunnissa tai kun tietty määrä transaktioita on sitoutunut edellisen tarkistuspisteen jälkeen.

Transaktioiden estäminen ja vaiheen 2 kesto voivat hidastaa normaalitoimintaa kiusallisen paljon.
Vaiheiden 2 ja (3, 4) järjestys voidaan vaihtaa, kun säilytetään edellinen loppuun suoritettu tarkistuspiste (viimeisenä virallisena), kunnes kaikki sivut ovat levyllä.

Lokiin perustuva elvytysalgoritmi:

(häiriöiden 1-4 (s. 16) aiheuttamat elvytystoimet)

1. luetaan lokia levyltä ja muodostetaan kaksi transaktiolistaa:

   L1 (aktiiviset) = transaktiot, joille on lokissa aloituskirjaus (start), mutta ei sitoutumiskirjausta eikä viimeistä tarkistuspistettä edeltävää keskeytyskirjausta

   L2 (sitoutuneet) = transaktiot, joille on lokissa sitoutumiskirjaus viimeisen tarkistuspisteen jälkeen

   - abort-kirjaus ennen tarkistuspistettä: muutosten peruutukset merkitty puskuriin, checkpoint huomannut

2. perutaan (undo) listan L1 transaktioiden write_item-operaatiot selaamalla lokia lopusta alkuun päin:
   - jokaista löytyvää muutoskirjausta (write,T,x,v1,v2) kohti suoritetaan operaatio write_item(x, v1) (palautetaan tietoalkion x alkukuva voimaan)

3. uusitaan (redo) listan L2 transaktioiden write_item-operaatiot selaamalla lokia alusta loppuun päin:
   - jokaista löytyvää muutoskirjausta (write,T,x,v1,v2) kohti suoritetaan operaatio write_item(x, v2) (saatetaan siis tietoalkion jälkikuva uudelleen voimaan)

Esimerkki. Olkoon levyllä oleva lokin sisältö häiriötilanteessa seuraava (järjestysnumero = 'aika'):

1: (start, T1)
2: (start, T2)
3: (write, T1, x1, 'AAA', 'BBB')
4: (commit, T1)
5: (write, T2, X1, 'BBB', 'CCC')
6: (checkpoint)
7: (write, T2, x2, '0000', '1111')
8: (start, T3)
9: (commit, T2)
10: (write, T3, x1, 'CCC', 'DDD')
11: (write, T3, x2, '1111', '2222')

Elvytysalijärjestelmä lukee lokia levyltä ja
- muodostaa listat L1 = <T3>, L2 = <T2>
- peruu transaktion T3 operaatiot suorittamalla:
  write_item(x2, '1111')
  write_item(x1, 'CCC')
- suorittaa uudelleen transaktion T2 operaatiot
  write_item(x1, 'CCC')
  write_item(x2, '1111')

Redo-operaatioissa tehdään edellä turhia kirjoituksia:
- ennen tarkistuspistettä suoritettua write_item-operaatiota ei tarvitse uusia
- ei tarvitse uusia myöskään sellaista tarkistuspisteen jälkeen tehtyä write_item-operaatiota, jonka tulos on ehtinyt levylle ennen häiriötilannetta

Jälkimmäinen tilanne saadaan selville, kun ylläpidetään tietosivun tunnustietueessa kenttää pageLSN: viimeistä ko. sivulle tehtyä päivitystä vastaavan lokitietueen järjestysnumero lokissa (log sequence number).

Myös undo voi olla turha:
ei tarvitse perua sellaisia kirjoituksia, joiden tulos ei ole ehtinyt levylle.

Redo/undo-tarve selviää vertaamalla järjestysnumeroita lokitietueessa (LSN) ja vastaavalla datasivulla (pageLSN):

- undo tarvitaan, jos LSN <= pageLSN
- redo tarvitaan, jos LSN > pageLSN

Esimerkissä voisi olla tilanne, että
- pageLSN(x1) = 10, pageLSN(x2) = 7

Tällöin tarvitaan x1:n peruminen (rivi 10) eikä muuta.

Entä häiriö kesken elvytyksen?

Redo-operaation tulee olla idempotentti eli sen peräkkäisten suoritusten tulee antaa sama tulos.
Koko elvytysprosessin tulee itse asiassa toimia näin;
kriittistä on write-operaatioiden suoritus.
write_item(x, arvo) on idempotentti
insert into T values ( ) ei ole

- - - - - -
Elvytyksen monimutkaisuuden tausta: pyrkimys puskuritilan joustavaan käyttöön.

E&N esittelee erikseen eri vaihtoehtoja:

- undo / no-redo (käytetään vain välittömiä päivityksiä)
- no-undo / redo (vain viivästettyjä päivityksiä)
- undo / redo (molemmat mahdollisia)

- E&N: LSN esitellään Aries-elvytyksen yhteydessä (21.5)

< tyhjä >