2. Tietokannan tallennusrakenteet

- tallennusrakenne = ”säilytysrakenne”

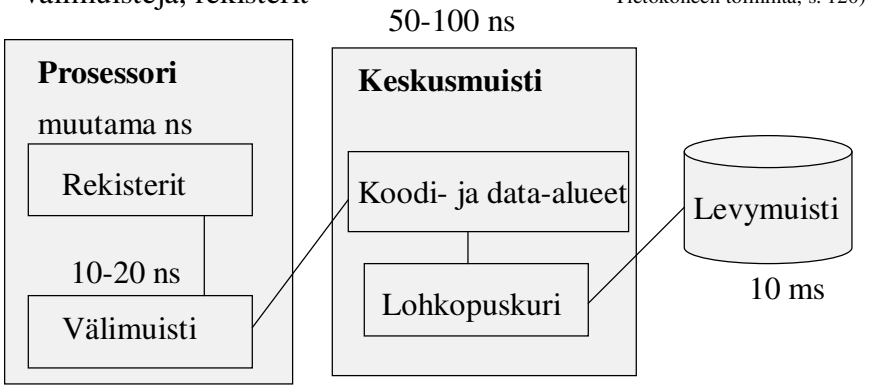
2.1 Levymuisti ja sen käyttö
2.2 Puskurointi
2.3 Tietokannan tiedostorakenne
2.4 Järjestämätön peräkkäistiedosto (kasa)
2.5 Järjestetty peräkkäistiedosto
2.6 Hajautukseen perustuvat tiedostorakenteet

E&N: luku 5 (ei 5.3)

2.1 Levymuisti ja sen käyttö

Muistilaitteiden hierarkia:
- tukimuistit (**levymuisti**)
- **keskusmuisti**
- välimuisteja, rekisterit

(Häkkinen, Tietokoneen toiminta, s. 120)



2.1 Levymuisti ja sen käyttö - 2

Tietokannalle levymuisti keskeinen:

- suuri kapasiteetti
- tietojen pysyvyys (yli keskusmuistihäiriöiden)
- riittävä nopeus
- halvempaa kuin keskusmuisti

Tietokantojen koko vaihtelee todella paljon: MB .. GB ..

Tietokantaperiaate perustuu normaalisti levymuistin käyttöön, vaikka kanta mahtuisikin keskusmuistiin.
- koko ei ole ainoa tekijä!
- keskusmuistin nopeutta hyödynnetään epäsuorasti: puskuroinnin välityksellä

Erikoistapaus: keskusmuistitietokanta (pysyvyys backup-kopioin)

Mikä on “very large database” nykyään?

2.1 Levymuisti ja sen käyttö - 3

Tietokannan käsittely:
- kaikki tietoihin kohdistuvat operaatiot (ohjelmallisten käskyjen suoritus) keskusmuistissa
- siirto levyltä ja levylle on hidasta

→ pyritään minimoimaan siirtojen määrä
- sopivilla tiedosto- ja hakemistorakenteilla
- muokkaamalla kyselyt tehokkaiksi

Tietokanta levyllä
- joukko tiedostoja (tauluja)
- joukko sivuja (yleensä yksi tiedosto = monta sivua)

Osoittaminen yleensä joko sivunumerolla tai parilla (sivunumero, tietuenumero).

Siirto levyltä/ levylle: koko sivu

2.1 Levymuisti ja sen käyttö - 4

Peruskäsitteitä:

Fyysinen tietokanta = levyllä oleva joukko sivuja,
jotka sisältävä tietueita.

Sivu (page) = vakiokokoinen alue, joka on sijoitettu levyjaksoon.
(termejä sivu, jakso (block), lohko käytetään vaihtoehtoina)

Tietue (record) vastaa relaation monikkoa (tuple),
jakaantuu kenttiin.

Kenttä (field) sisältää attribuuttiarvoja (= merkkijonoja, lukuja).
Kentän arvon merkitys voi olla myös osoitin; hakemistoissa tai joissakin (muissa kuin relaatio-) tietokannoissa.

2.1 Levymuisti ja sen käyttö - 5

Tietokannan sisältöön viitataan eri tasoilla:

- tiedoston (taulun) nimi tai muu tunniste
- sivunumero
- tietuenumero
- kentän tunniste (tai paikka)

Levymuistiin viitattaessa sivunumero on tärkein,
tietuenumero tulee käyttöön ja kentän löytäminen tapahtuu keskusmuistin puskurissa.

2.1 Levymuisti ja sen käyttö - 6

Levymuistin rakenne:
levykkö, levypinta, sylinteri, ura, sektori (,merkki)

Levyjakso t. lohko (disk block; ”sivun paikka”) = yhdestä tai useammasta peräkkäisestä sektorista koostuva alue levypinnan uralla.

Sektori = pienin osoitettavissa oleva yksikkö;
sektorin fyysinen osoite = (levypinta, ura, sektori).

Levyhaku (disk access) = levymuistin käytön perussuure:
joko jakson luku (kopiointi keskusmuistin puskuriin)
tai jakson kirjoitus puskurista levylle.

2.1 Levymuisti ja sen käyttö - 7

Jaksot määräytyvät levyn formatoinnin yhteydessä: kiinteä koko, tyypillisesti 1 KB - 16 KB.

Siirto jaksonpituisina kokonaisuuksina (sivuina) on kompromissi:

- levyn rakenteesta johtuen ei kannata siirtää esimerkiksi kenttiä eikä edes tietueita,
- jakson pidentämistä taas rajoittaa tarvittavan keskusmuistitilan (puskurin) tarve ja mahdollisesti tarpeettomien osien siirron vaatima aika

2.1 Levymuisti ja sen käyttö - 8

Levyhaun vaatima aika: **saantiaika** eli hakuaika (access time) =
kohdistusaika + pyörähdysviive + siirtoaika

Kohdistusaika s (seek time): luku/kirjoituspää viedään oikealle uralle (oikealle sylinterille)

Sylintereitä esimerkiksi 8000, kohdistusaika välillä 0-s_{max};
esimerkiksi keskimäärin 5 ms, **välillä 0 - 15 ms**.

Pyörähdysviive rd (rotational delay, latency): odotetaan levyn pyörähdystä jakson alkukohdalle.

- kiinteä pyörimisnopeus esim. 10000 rpm
- keskimääräinen odotus puoli kierrosta

$\rightarrow$ viive rd = 60000 / 2*10000 ms = 3 ms
(**yleensä** luokkaa **3 - 8 ms**)

2.1 Levymuisti ja sen käyttö - 9

Jakson siirtoaika bt (block transfer time):
yhden jakson kopiointi puskuriin tai päinvastoin.

- riippuu jakson ja uran koosta sekä pyörimisnopeudesta:

esim. jakso 4KB, ura 128KB:
bt = (4/128) * (60/10000) s = 0.19 ms
(= yleensä pienin komponenteista)

2.1 Levymuisti ja sen käyttö - 10

Todellinen hakuaika on monimutkainen laskea (sijoitteluvaihtoehdot, sektori– l. jaksovälit ym. kontrollitiedot); yleensä riittää jakson

keskimääräinen hakuaika s + rd + bt

Siis k jaksolle k* (s + rd + bt) ?

Suurten komponenttien s ja rd toistuminen pyritään eliminoimaan:

s = 0, jos luetaan samalta sylinteriltä kuin edellinen jakso (ja pieni, jos luetaan viereiseltä)
rd = 0, jos luetaan peräkkäisiä jaksoja (samalta uralta)

eli k jakson luku: s + k * (rd + bt) tai s + rd + k * bt

2.2 Puskurointi

Puskuri = keskusmuistialue, jossa on tilaa joukolle jaksoja (sivuja; page frame). Tkhj:lla on yleensä oma puskurinhallitsin, joka tietää (kontrollitiedoista)

- mitkä tietokannan sivut ovat puskurissa,
- onko puskurissa olevan sivun sisältöä muutettu.

Puskurissa olevaa sivua voi käyttää moni operaatio (transaktio) - ilman moninkertaista levyltä lukua (ja välillä tapahtuvaa kirjoitusta). (Tapahtumanhallinta hoitaa, levyhakuja säästyy ...)

- puskurista haku esim. vain 10-100 ns
- levyhaun aika jakaantuu monelle operaatiolle

2.2 Puskurointi - 2

Täysi hyöty peräkkäisten jaksojen lukemisesta vaatii, että keskus-muistikäsittely voi tapahtua päällekkäin levyoperaation kanssa.

Kaksoispuskurointi:

luetaan levyltä puskuriin A;
käsitellään puskuria A, luetaan levyltä puskuriin B;
käsitellään puskuria B; luetaan levyltä puskuriin A;
(jne)

(tulee olla käsittelyaika < jakson saantiaika, ja
erillinen i/o-prosessori)

Puskurin käyttö on erityisen tehokasta, jos monet transaktiot tarvitsevat samoja sivuja.
Esim. kyselyt samoihin ajankohtaisiin asioihin, ilmoittautumiset suosituille kursseille.

2.2 Puskurointi - 3

Puskurien käyttö loogisesti:

- operaatiot
  bufferfix(P): tuo sivu P puskuriin
  bufferunfix(B): vapauta puskuritila (sivu) B
  - jaksoa ei välttämättä siirretä heti levylle eli puretaan vain kytkentä B:n ja P:n väliltä (siirto voi tapahtua esimerkiksi, kun tilaa tarvitaan muuhun)

Esim.
```
update employee
set salary = salary + 500;
```

- käydään läpi kaikki relaation employee monikot
- on edullista, jos ne ovat levyllä niin peräkkäin kuin mahdollista

2.2 Puskurointi - 4

Kyselyn yksinkertaistettu toteutus:

```
for jokainen employee-taulun sivu P do
  B:=bufferfix(P);
  for jokainen puskurisivulla B oleva
        employee-monikko e do
          e.salary := e.salary + 500;
  aseta B:n päivitysbitti ('muutettu');
  bufferunfix(B);
end.
```

- Jos on puskuritilaa usealle sivulle B1, B2, …, puskuria ei siis tarvitse vapauttaa heti.

(hyöty tälle / muille kyselyille?)

2.3 Tietokannan tiedostorakenne

Tiedosto = tietuejoukon muodostama kokonaisuus
(loogisesti: tietueita, fyysisesti: tietokantasivuja).

Yleensä tiedosto muodostetaan samantyyppisistä tietueista, esim. relaatiosta (taulusta).
Erikoistapaus: myös erityyppisiä (eri taulujen) tietueita voidaan säilyttää yhdessä, lähekkäin.

Tiedoston alue levyllä: joukko varausyksikköjä (segment, cluster, extent).
Varausyksikkö = peräkkäisten jaksojen muodostama yhtenäinen alue (vrt. peräkkäin luku).

Tilanvaraus riippuu tiedoston koosta ja järjestelmän piirteistä.

2.3 Tietokannan tiedostorakenne - 2

Esim. Oracle:
tablespace: tietokannalle varataan (loogisesti) 1 tai useampia
'taulutiloja', jotka jaetaan erityyppisiin segmentteihin:
datasegmentti: taulun data (create table ...)
indeksisegmentti: hakemiston tiedot (create index ...)
työtilasegmentit: välituloksille
peruutussegmentit: elvytystiedoille

varausyksiköt (extent): segmenttien osia;
tilaa otetaan käyttöön tarpeen mukaan alue kerrallaan

- tilankäyttöä säädellään tietokannanhoitajan toimesta tai
create-lauseiden parametreilla
(yksityiskohdilla ei ole tietokannan käytön
logiikan kannalta merkitystä)

2.3 Tietokannan tiedostorakenne - 3

Tiedostokuvaaja (file header, file descriptor) edustaa tiedostoa ohjelmille:
- tiedoston levyjaksojen osoitteet (esim. luettelona)
- tietueiden rakennetiedot
- muuta hallintatietoa

Esimerkki monimutkaisesta organisoinnista: Unixin i-node

- 10 suoraa levyosoitetta
- 0-3 epäsuoraa osoitetta (osoitelohkoihin)

i-node kiinteänkokoinen
osoitelohkoihin voidaan pakata 'riittävästi' jakso-osoitteita

2.3 Tietokannan tiedostorakenne - 4

Sivun sisäinen organisointi

Sivulla on **otsikkotietue** (page header):
hallintatietoa, esim. ensimmäisen ja viimeisen tietueen osoite,
sivun viimeisin käyttö- ja päivitysaika

Tietuerakenteen päävaihtoehdot:

1. **kiinteänmittaiset** tietueet:
   jaksossa on tilaa f tietueelle, kiinteät tietuepaikat
   f = **jaksotuskerroin,** $f = \lfloor B/R \rfloor$, B = jakson ja R tietueen pituus
   - yleensä jaksossa on useita kokonaisia tietueita
2. **vaihtuvanmittaiset** tietueet: tietueiden paikkojen hallinta monimutkaistuu

2.3 Tietokannan tiedostorakenne - 5

Normaalisti tietueita ei jaeta useaan jaksoon. Voi olla myös **jatkettu tietue** (spanned record): tietue jatkuu jaksorajan yli, periaatteessa moneenkin jaksoon:

jakso i	tietue j-2	tietue j-1	tietue j (alkuosa)

jakso i+1	tietue j (loppuosa)	tietue j+1	.....

- tila käytetään tarkemmin
- käsittely monimutkaistuu
- joskus välttämätön (blob-tyyppi)

2.3 Tietokannan tiedostorakenne - 6

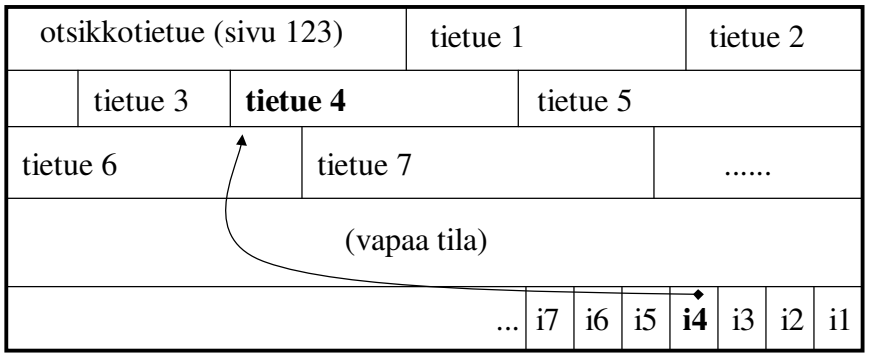
Kiinteänmittaisten tietueiden paikat voidaan laskea suoraan tietueen järjestysnumerosta sivulla.

Vaihtuvanmittaisten tietueiden paikat voidaan hallita
esim. seuraavalla organisaatiolla:

- otsikkotietue (page header)
- tietueet (monikot) peräkkäin luontijärjestyksessä
- tietuehakemisto: tietueiden (suhteelliset) osoitteet
  (loogisesti tavallaan otsikkotietueen laajennus)

Sijoittamalla tietuehakemisto kasvamaan sivun lopusta alkua kohti voidaan sivun tila käyttää optimaalisesti hyväksi.

2.3 Tietokannan tiedostorakenne - 7



tietuetunniste (monikkotunniste):

node_id	file_id	page_no	tuple_index
		123	6

2.3 Tietokannan tiedostorakenne - 8

Yleisesti tiedoston tietueen osoite = (p, i)

p = sivun numero

i = tietuehakemiston indeksi (tai tietueen järjestysnumero tai tietueen osoite, jos ei ole tietuehakemistoa)

Vielä yleisempi muoto on moniosainen tietuetunniste (edell. kuva):

tietoverkon pisteen (site) tunniste

tiedoston tunniste

osoite (p, i).

Huom. Relaatiotietokannassa ei ole osoitteita datasivuilla, kylläkin hakemistosivuilla. Olio- tai verkkotietokannoissa oliot (tietueet) voivat sisältää toisten olioiden (tietueiden) osoitteita (osoittimia).

2.3 ..... Tietueen esitysmuoto

Tietueen esitysmuoto

- kiinteänmittaisia / vaihtuvanmittaisia kenttiä
  kiinteä pituus: tilaa ehkä tuhlataan
  vaihtuva pituus: minimoi tilankäytön
  Yksikin vaihtuvanmittainen kenttä → tietue vaihtuvanmittainen

- tiedostossa on tyypillisesti yhden relaation monikkoja
  → tietueet samantyyppisiä (samat kentät)
- voi myös olla erityyppisiä tietueita (esim. klusteroitu tiedosto, jossa on monen relaation monikkoja)

Seuraavassa esitysmuodon perusvaihtoehdot esimerkein:

2.3 ..... Tietueen esitysmuoto - 2

Relaation employee(name, ssn, salary, address) monikon
('Smith, John', '010263-189F', 17000, 'Park Avenue ...')
esittäminen:

1. kiinteänmittaiset kentät

Smith, John	010263-189F		17000	Park Avenue ...
20 tavua	11 tavua		4 tavua (lukuna)	30 tavua

Kokonaisluku ehkä aloitetaan sanarajalta; mahdollisesti edessä tyhjä tavu.

2.3 ..... Tietueen esitysmuoto - 3

Vaihtuvanmittaiselle kentälle erilaisia esitystapoja:

2. pituuskentän avulla

010263-189F	17000	11	Smith, John	26	Park Avenue ...

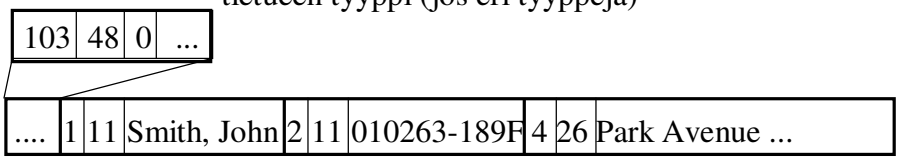
- kiinteänmittaiset alussa: näillä kiinteät paikat,
  vaihtuvanmittaisen osan alkukohta samoin
- kenttien järjestys (vähintään) kuvattava
  esim. sivun otsikkotietueessa

3. erotinmerkkien avulla

010263-189F	#	17000	#	Smith, John	#	Park Avenue ...	#

2.3 ..... Tietueen esitysmuoto - 4

2a. Oraclen esitysmuoto:
- kaikki tiedot periaatteessa vaihtuvanmittaisia (sama esitystapa)
- attribuutin tunniste (järjestysnumero), pituus ja arvo
- tietueen otsikossa
  - järjestysnumero (relaatioon lisäämisen järjestyksessä)
  - tietueen pituus
  - hallintatietoa,
    - esim. poistomerkintä,
    - tietueen tyyppi (jos eri tyyppejä)



Huom. Tässä palkka on NULL eli puuttuu esityksestä.

2.3 ..... Tietokantasovelluksen tehokkuus

Tehokkuuden arviointi esim. kyselyä kohti
- karkeasti levyhakujen lukumäärällä
  - hajautetussa tapauksessa verkon yli siirrettävän tiedon määrällä
- tarkemmin ottamalla huomioon myös
  - puskurien koot
  - levymuistin ominaisuudet
  - verkkoyhteyksien nopeus jne

Levyhakujen määrä on yksinkertainen mitta ja
- laitteistosta riippumaton
- riippuu tiedostorakenteesta

Levyhakujen maksimimäärä vai keskiarvo?
vrt. tietorakennealgoritmien analyysi …

2.3 ..... Tietokantasovelluksen tehokkuus - 2

Tiedostorakenteen tasolla vaikuttavia asioita:
- missä järjestyksessä tietueet ovat levyllä
- mitkä tietueet ovat lähekkäin (erityisesti: samalla sivulla)
- mitä yhteyksiä tietueiden ja tiedostojen välillä ylläpidetään osoittimilla (relaatiotietokanta: ei)

Tiedoston (perustietojen eli tietueiden) organisointitavan lisäksi käytetään apurakenteita (hakemistoja eli indeksejä, yleisesti saantimenetelmiä), joilla pyritään vähentämään 'raakaa' tiedon hakua esim. tiedostoa läpi lukemalla:
- nopeuttavat operaatioita
- vievät lisää levytilaa
- vaativat (omat?) puskuritilansa
- vaativat ylläpitoa ("ylimääräisiä" operaatioita)
- tekevät tehokkuuden analysoinnin monimutkaisemmaksi

näihin palataan!

2.3 ..... Tietokantasovelluksen tehokkuus - 3

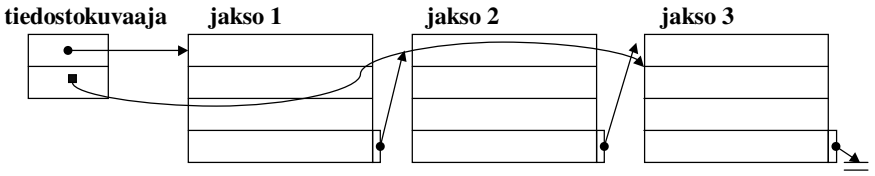
'Lopullinen' vaikeus tiedostorakenteen suunnittelussa ja suorituskyvyn arvioinnissa: sovellusten (yleensä useita!) erilaiset käyttötavat ja niiden keskinäinen tärkeys
- eri kyselyjen frekvenssit?
- ketä/mitä operaatiotyyppiä palvellaan eniten?
  esim. kyselyjä vai lisäyksiä (tai päivityksiä)
- sovellusten ristiriitaiset tarpeet
- tilanteen muuttuminen (+ käyttötietojen puute alunperin)

Esim. hae henkilön N.N. tiedot
muuta henkilön P.P. osoite
korota osaston 9 työntekijöiden palkkoja 5 %
korota kaikkia palkkoja 5 %

2.4 Järjestämätön peräkkäistiedosto (kasa)

Tietueet sijoitetaan peräkkäisille sivuille siinä järjestyksessä kuin ne muodostetaan.

Tiedostoon kuuluvien sivujen hallinta: esimerkiksi ketjurakenteena:



Relaatiotietokannan relaation oletusrakenne on yleensä kasa.
- vrt. relaation määritelmä (ei järjestystä)
- entä yleiset käsittelytavat? (nimen aakkosjärjestys, osastoittain jne)

2.4 Järjestämätön peräkkäistiedosto (kasa) - 2

Esim. Oracle:
create table employee (...)
luo tyhjän kasan.

insert into employee values (...)
sijoittaa uuden monikon kasan viimeiseen jaksoon.

Insert-operaatio on kasaan sovellettuna tehokas: tarvitaan yleensä vain kaksi (korkeintaan 3) levyhakua, kasan koosta riippumatta. Aikavaativuus on siis vakiokertaluokkaa: O(1).

Avainrajoitteen tarkistus vaatii kuitenkin pahimmassa tapauksessa kaikkien jaksojen lukemisen eli jopa N+2 levyhakua – eli tehokkuus osittain näennäistä!

Tietueen haku vaatii keskimäärin N/2 levyhakua (kun tietue löytyy).

2.4 Järjestämätön peräkkäistiedosto (kasa) - 3

Tietueen poisto avaimen avulla: tietueen paikannus (haku), poisto (tai ainakin poistomerkintä) ja mahdollisesti muutetun sivun kirjoitus takaisin levylle.

Poisto-operaatio yleisemmässä tapauksessa, esim.

delete from employee where dno=5:

```
for jokainen employee-taulun sivu P do
        B:=bufferfix(P);
        for jokainen B:n monikko e do
                if e.dno = 5 then
                        merkitse e poistetuksi;
                        aseta sivun B päivitysbitti ← 1
                endif;
        bufferunfix(B);
end.
```


2.4 Järjestämätön peräkkäistiedosto (kasa) - 4

Poisto vaatii pahimmassa tapauksessa 2N levyhakua: jokaisen sivun haun ja kirjoituksen.

Poistomerkintää käytettäessä tuhlataan tilaa - ja hakualgoritmin tulee ottaa merkinnät huomioon.

Vaihtoehto poistomerkinnöille:
- poistetaan tietue, vapautetaan tietueen paikka
  → jaksoihin jää tyhjiä paikkoja
  uudelleen käyttö mahdollista, mutta monimutkaista
  (jakson tiivistäminenkin mahdollista, mutta toistuvana työlästä – vai onko?)

Tietueen päivitys: haetaan sivu puskuriin, muutetaan, kirjoitetaan (ei välttämättä heti) levylle samaan paikkaan.

2.4 Järjestämätön peräkkäistiedosto (kasa) - 5

Vaihtuvanmittainen tietue?
- ei mahdu välttämättä entiselle paikalle
  → poistetaan vanha, lisätään uusi kasan loppuun

Siis on mahdollista, että kasa muuttuu harvaksi, jolloin se on (joskus) organisoitava uudelleen (reorganisation):
luetaan kasan tietueet, viedään ne uuteen tyhjään kasaan ja hävitetään vanha kasa.

Uudelleenorganisointi on normaali toiminto tietokannan ylläpidossa (ei vain kasalle, vaan monelle muulle rakenteelle!):
- kallis (pyritään välttämään)
- ajankohtaa ei aina ole helppo löytää

2.4 Järjestämätön peräkkäistiedosto (kasa) - 6

Esimerkkejä.

1) Kyselyn select * from employee where ssn ='123456789'
toteutus kasarakenteessa:

```
for jokainen employee-taulun sivu P do
        B:=bufferfix(P);
        for jokainen B:n monikko e do
                if e.ssn = '123456789' then tulosta e;
        bufferunfix(B);
end.
```

Kyselyn aikavaativuus on N levyhakua, jos ei ole ssn-avainrajoitetta.

Jos on ssn-avainrajoite, keskimäärin N/2; pahin tapaus edelleen N.

2.4 Järjestämätön peräkkäistiedosto (kasa) - 7

2) Kasarakenteisen tiedoston käytön ”todellinen” suoritusaika:

Olkoon tiedostossa n = 1 milj. tietuetta
á 200 tavua, ja jakson koko 4KB (4096 tavua).

Jaksotuskerroin on f = $\lfloor 4096/200 \rfloor$ = 20.

Jaksoon jää ‘ylimääräistä’ tilaa 96 tavua, mikä ehkä riittää jakson sisäiseen organisointiin (sivuotsikko, linkit ym.)

Jaksoja on N = 1000 000 / 20 = 50 000 kpl. Jakson saantiajalla 10 ms tietueen haku vie enintään 50 000 * 10 ms = 8.3 min!

Huom. Saantiaika 10 ms sisältää muutakin kuin siirtoajan.
Esim. pelkällä siirtoajalla 0.2 ms tietueen haun maksimiaika on 50 s.
(Ts. kasa sellaisenaan ei ole yleensä hyvä ratkaisu.)

2.4 Järjestetty peräkkäistiedosto

(ordered file, sequential file)

Tietueet säilytetään tiedostossa (jaksoissa) jonkin järjestyskentän (järjestysavaimen) mukaan (nousevassa) järjestyksessä.

Järjestys saattaa vastata intuitiivisesti joitakin käsittely- tai tulostustarpeita - mutta ei kaikkia …
Tiedosto voidaan järjestää vain yhden kentän mukaan kerrallaan.

Kasan tehokkuusongelmat – ratkaiseeko järjestys ne?

2.4 Järjestetty peräkkäistiedosto - 2

Esim. employee-relaatio järjestysavaimena name:

	NAME	SSN	BDATE	SALARY
jakso 1	Aaron, Ed	…	…	…
	Abbott, Diane	…	…	…
	. . .			
	Acosta, Marc	…	…	…
jakso 2	Adams, John	…	…	…
	Adams, Robin	…	…	…
	. . .			
jakso 3	Allen, Sam	…	…	…
	Andre, Marc	…	…	…
	. . .			
jakso N	Wright, Pam	…	…	…
	. . .			
	Zimmer, Byron	…	…	…

2.4 Järjestetty peräkkäistiedosto - 3

Operaatioiden aikavaativuudesta:

- Koko tiedoston läpikäynti järjestysavaimen mukaisessa järjestyksessä hyvin nopeaa. Seuraavaan tietueeseen siirtyminen vaatii uuden levyhaun vain 1/f tapauksessa (f = jaksotuskerroin).

- Tietueen haku järjestysavaimen mukaan:
  1) tiedostoa läpilukemalla keskimäärin N/2 levyhakua, maksimi N (vrt. kasa)
  2) voidaan käyttää binäärihakua (‘haarukointia’), kustannus enintään enintään $\lceil \log_2 N \rceil$ levyhakua (kun järjestysavain on yksikäsitteinen).

     Esim. N = 50 000 (n = 1000 000) , $\lceil \log_2 N \rceil$ = 16 levyhakua, saantiajalla 10 ms aikaa kuluu siis 0.16 s.

2.4 Järjestetty peräkkäistiedosto - 4

Binäärihaun algoritmi:

1. Etsi tietuetta keskimmäiseltä sivulta $\lfloor N/2 \rfloor$.
2. Jos ei löydy, etsi samalla periaatteella sivuilta 1, 2, … , $\lfloor N/2 \rfloor$ -1
   (kun tutkitun sivun avaimet ovat suurempia kuin haettava avain),
   tai sivuilta $\lfloor N/2 \rfloor$ +1, … ,N
   (kun tutkitun sivun avaimet ovat pienempiä kuin haettava avain).
3. Haku onnistuu, kun tietue löytyy tutkittavalta sivulta, ja
   päättyy tuloksettomana, kun
   (1) tutkittava tiedoston osa on yksi sivu eikä
   haettua tietuetta ole tällä sivulla, tai
   (2) tutkittava tiedoston osa on tyhjä.

2.4 Järjestetty peräkkäistiedosto - 5

Muita järjestetyn tiedoston ominaisuuksia:

- Haku erisuuruusehdon <, >, =>, <= perusteella on suhteellisen tehokasta: ehdon täyttävät tietueet ovat lähekkäin ja haku voidaan keskeyttää selaamatta koko tiedostoa läpi.
  (myös muunnettu binäärihaku mahdollinen?)

Esim.
```
select name from employee
where salary > 10000 and salary <= 12000;
```

- Haku muun kuin järjestysavaimen mukaan ei nopeudu kasaan verrattuna yhtään. (yksittäisen tietueen haku: mahdollisesti läpiluku; kaikki järjestyksessä → ensin lajittelu hakuavaimen mukaan)

2.4 Järjestetty peräkkäistiedosto - 6

Lisäys, poisto?
Tiedoston tulee olla operaation jälkeen järjestyksessä.

Lisäys:
1. On löydettävä oikea lisäyskohta.
2. Jos jaksossa ei ole tilaa, on lisäyksen jälkeen
   siirrettävä keskimäärin puolet tietueista (yhden tietuepaikan verran eteenpäin), tai
   linkitettävä toiseen jaksoon lisätty tietue järjestyksen määräämälle paikalleen (tietue- tai sivulinkki)

Poisto: Poistettavan paikannus kuten lisäyksessä,
poistomerkintää käytettäessä siirtoja ei tarvita.

2.4 Järjestetty peräkkäistiedosto - 7

Vaihtoehtoja lisäyksen vaatimille siirroille:
- Jätetään jaksoihin systemaattisesti varatilaa lisäyksiä varten (muutamalle tietueelle, esim. 20-30 %).
- Linkitetään jaksoihin ylivuotojaksoja (vrt. ed.).

E&N: myös järjestämätön ylivuototiedosto mahdollinen (uudelleenorganisointi ajoittain lomittamalla varsinainen ja ylivuototiedosto yhteen; lomitus, ks. seuraava kalvo)

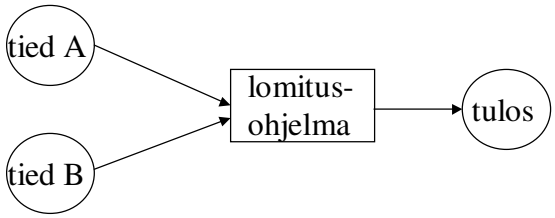
Vaihtoehdot lisäävät monimutkaisuutta. Uudelleenorganisointi tulee joka tapauksessa jossain vaiheessa tarpeelliseksi.

Järjestetty peräkkäistiedosto ei ole sellaisenaan yleinen tietokannan hallinnassa. Lisättäessä järjestetyn perustiedoston ‘päälle’ hakemistorakenne saadaan ns. IS-tiedosto, joka on hyvin tehokas myös yksittäisten hakuoperaatioiden kannalta (luku 3).

2.4 Järjestetty peräkkäistiedosto - 8

Järjestykseen perustuva käsittely on hyvin tehokasta silloin, kun se on mahdollista. Tyypillisiä tilanteita ovat esim.

- laajan aineiston läpikäynti yleensä
- usean järjestyksessä olevan tiedoston lomitus (merge) eli
  'tahdistettu käsittely'
  (sovelluskohteita mm. liitosoperaation toteutus, duplikaattitietueiden poisto, eräät joukko-operaatiot)



2.6 Hajautukseen perustuvat tiedostorakenteet

- tavoite: saada yksittäinen tietue käsittelyyn nopeasti
→ tarvitaan suorasaantitiedosto (random access file)
  - hajatiedosto, hajautettu tiedostorakenne

- suoraan levyosoitteilla? nopein, mutta joustamaton

Hajautuksessa (hashing) tietue paikannetaan
'suoraan' (= selaamatta ympäristöä)
'epäsuorasti': ei levyosoitteella, vaan

säilyttämällä levyjaksojen osoitteita hajautusalueella.

Keskusmuistirakenteissa hajautuksen tavoite sama: selaamisen välttäminen. Levymuistia käytettäessä pyritään minimoimaan sivujen (levyjaksojen) lukemista; jakson sisältä voidaan etsiä selaamallakin …

2.6 Hajautukseen perustuvat tiedostorakenteet - 2

Hajatiedosto jakaantuu loogisesti soluihin - oikeastaan solutiloihin (sankoihin; bucket, cell, slot).

Solu = yksi tai useampi jakso, rakenne yleensä kasa.

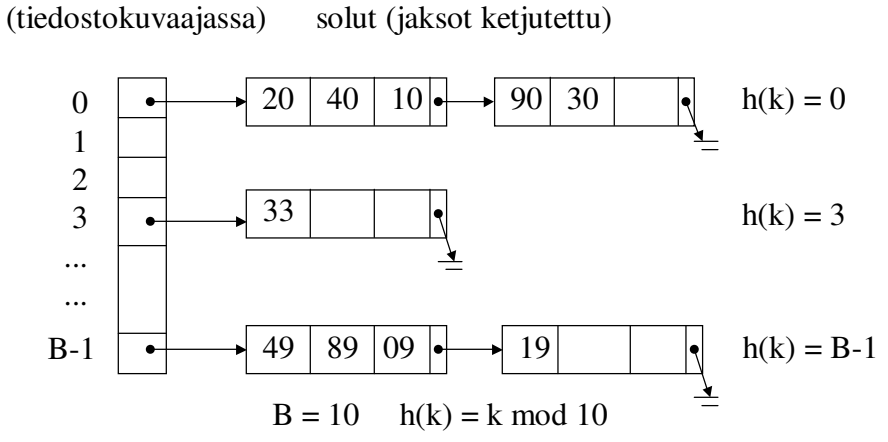
Tietueet jaetaan soluihin **hajauttimen** l. hajautusfunktion h avulla: solun **kotiosoite** ha saadaan kaavasta

ha = h(k),

missä k on **hajautusavain**, yleensä tietueen tunniste (tai sen osa).

Hajautusavaimen arvoalue on yleensä paljon laajempi kuin hajautusalueen koko, ts. monella tietueella on sama kotiosoite (”yhteentörmäys”; yhtä solua vastaavan tiedoston tietueet).

2.6 Hajautukseen perustuvat tiedostorakenteet - 3



(soluhakemistossa yleensä myös osoitin kasan viimeiseen jaksoon)

2.6 Hajautukseen perustuvat tiedostorakenteet - 4

Kotiosoitetta vastaavaan levyjaksoon mahtuu jokin määrä tietueita. Sen perään voidaan ketjuttaa ylivuotojaksoja (syntyy kasa, jaksot ketjutettuina).

Jaksorakenteen takia yhteentörmäyksellä ei ole aivan samaa merkitystä kuin keskusmuistihajautuksessa.

(Keskusmuistirakenteissa tästä aiheutuvat yhteentörmäykset vaativat tavallaan välitöntä ratkaisua (ylivuotoalue, ketjutus jne). - ei tässä.)

Hajautusavaimen arvolla k varustetun tietueen **haku**:
lasketaan h(k) ja etsitään tietuetta kotiosoitteesta alkavasta jaksoketjusta kuten kasasta.

2.6 Hajautukseen perustuvat tiedostorakenteet - 5

Hajautuksen optimitilanne: jokaisen ketjun pituus = 1.

Jos on aito ketju (enemmän kuin yksi jakso / kotiosoite), tässä **ei** toteudu aivan suorasaantitiedoston idea löytää tietue suoraan yhdellä levyhaulla.
Toisaalta ero kasan tai järjestetyn tiedoston etsintään on selvä:
hajautuksessa ehkä keskimäärin 1.x levyhakua.

Lisäys hajautettuun tiedostoon:
- lasketaan lisättävän tietueen kotiosoite kuten haussa
- lisätään tietue vastaavaan soluun kuten kasaan

Poisto: paikannus kuten haussa; poistetaan tietue jaksosta
loogisesti (poistomerkinnällä), tai
tiivistämällä samalla kasaa

2.6 Hajautukseen perustuvat tiedostorakenteet - 6

Hajauttimella on levymuistiin perustuvassa rakenteessa samat ominaisuudet kuin keskusmuistirakenteissa:

- avainarvot (tietueet) tulee hajauttaa mahdollisimman tasaisesti kotiosoitteisiin (osoiteavaruuteen) (tärkeä)
- h(k) tulee voida laskea nopeasti (tässä vähemmän tärkeä)

Yleinen hajauttimen muoto: $h(k) = k \bmod B$.

Jos k ei ole kokonaisluku, siitä voidaan muodostaa yksinkertaisella muokkauksella kokonaisluku (merkkien koodiarvojen summa, merkkien tai bittien poiminta arvon keskeltä tms.). (Eikä mod-operaatio ole ainoa tapa normeerata tulos välille (0,...,B-1).

2.6 Hajautukseen perustuvat tiedostorakenteet - 7

Hajautusrakenteen tehokkuus:

- soluhakemisto pyritään pitämään keskusmuistissa (muuten yksi levyhaku oikean soluosoitteen saamiseksi)

Oletetaan, että N-sivuinen tiedosto on hajautettu tasaisesti B soluun.

Tietueen **lisäys** vie enintään 3 levyhakua (kun ei ole avainrajoitetta).

Tietueen **haku**: enintään $\lceil N/B \rceil$ levyhakua.
Tietueen **poisto**: enintään $\lceil N/B \rceil + 1$ levyhakua.
- poisto-operaatio muulla kuin hajautusavaimeen liittyvällä ehdolla voi vaatia $2\lceil N/B \rceil$ levyhakua.

Esim. N = 50000, B = 5000, saantiaika 10 ms
- operaatiot maksimissaan 0.1-0.2 s

2.6 Hajautukseen perustuvat tiedostorakenteet - 8

Päivitysoperaatio: jos muutetaan muuta kuin avainkentän arvoa, muutettu tietue sijoitetaan samaan jaksoon (tai ainakin kasaan); muutettaessa avainkentän arvoa tietue poistetaan kotisolustaan ja lisätään uuteen kotisoluunsa.

Parametrien yhteys:

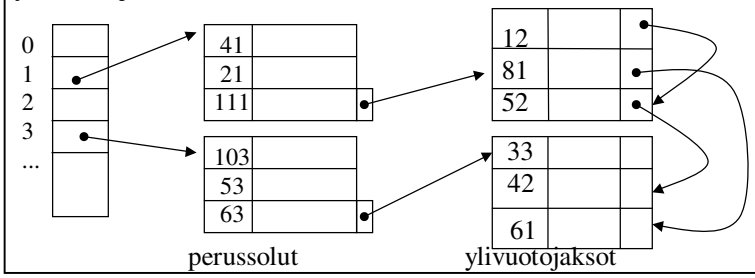
- suuri hajautusalue (ja hyvä hajautin) → hyvä hajautus (lyhyet ketjut); voi olla paljon vajaita jaksoja
- pieni hajautusalue tai huono hajautin → pitkiä ketjuja, voi silti olla paljon vajaita jaksoja

Parametrien valinnassa tarvittaisiin tietoa tiedoston dynaamisuudesta ja operaatioista. Pyritään siihen, että tiedoston luontivaiheessa
- jokainen solu olisi yksisivuinen (eikä aivan täynnä).

2.6 Hajautukseen perustuvat tiedostorakenteet - 9

(Perus)hajautusrakenne on luonteeltaan staattinen: solujen lukumäärä on kiinteä. Usein solujen tasapaino heikkenee vähitellen - koko tiedosto on organisoitava uudelleen.
(B:n muutos? hajauttimen muutos?).

Joskus käytetään yhteisiä ylivuotojaksoja useille (t. kaikille) soluille. Tällöin on luontevaa käyttää jakso-osoitteiden sijasta tietueosoitteita ylivuotojaksoissa:



2.6 Hajautukseen perustuvat tiedostorakenteet - 10

Hajautusrakenteen käyttö kyselyissä:

- (taulun) avaimen tarkkaan arvoon perustuva haku nopea
  `select * from employee where ssn = '123456789';`

- muuhun arvoon perustuva haku yhtä tehoton kuin kasasta
  (kaikkien kotisolujen kaikki ketjut …)
  `select * from employee where name = 'Smith';`

- likimääräiseen arvoon perustuva haku kuten kasasta
  `select * from employee where name like 'Sm%';`
  (vaikka name olisi avain; siis tehoton)

- järjestysehto ... `name < 'S%'` : tehoton

2.6 Hajautukseen perustuvat tiedostorakenteet - 11

Hajautuksen erikoistapauksia:

- täydellinen hajautin: ei samoja soluosoitteita eri avaimille
  (ei ole mielekäs, kun jaksoon mahtuu useita tietueita; tasainen jakauma riittää)

- järjestyksen säilyttävä hajautin
  sovellusten kannalta toivottava; saataisiin sekä nopea yksittäin haku että järjestyksessä tapahtuva käsittely samasta tiedostosta

  Esim. (juoksevan) tilausnumeron alkuosa hajautusavaimena

2.6 Hajautukseen perustuvat tiedostorakenteet - 12

Oracle:
- hajautus liittyy klusterointiin: yhden tai useamman taulun rivit säilytetään yhdessä
- hajautusfunktio voidaan antaa; oletus '… mod alkuluku'
- datalohkojen ketjutusta kehotetaan välttämään (yksi levyhaku!)
-sopivuus:
    - yhtäsuuruuskyselyt (koko avaimelle)
    - tiedosto staattinen tai maksimikoko ennustettavissa