732G16: Databaser; Design och programmering

Fö 6: Fysisk design av databasen

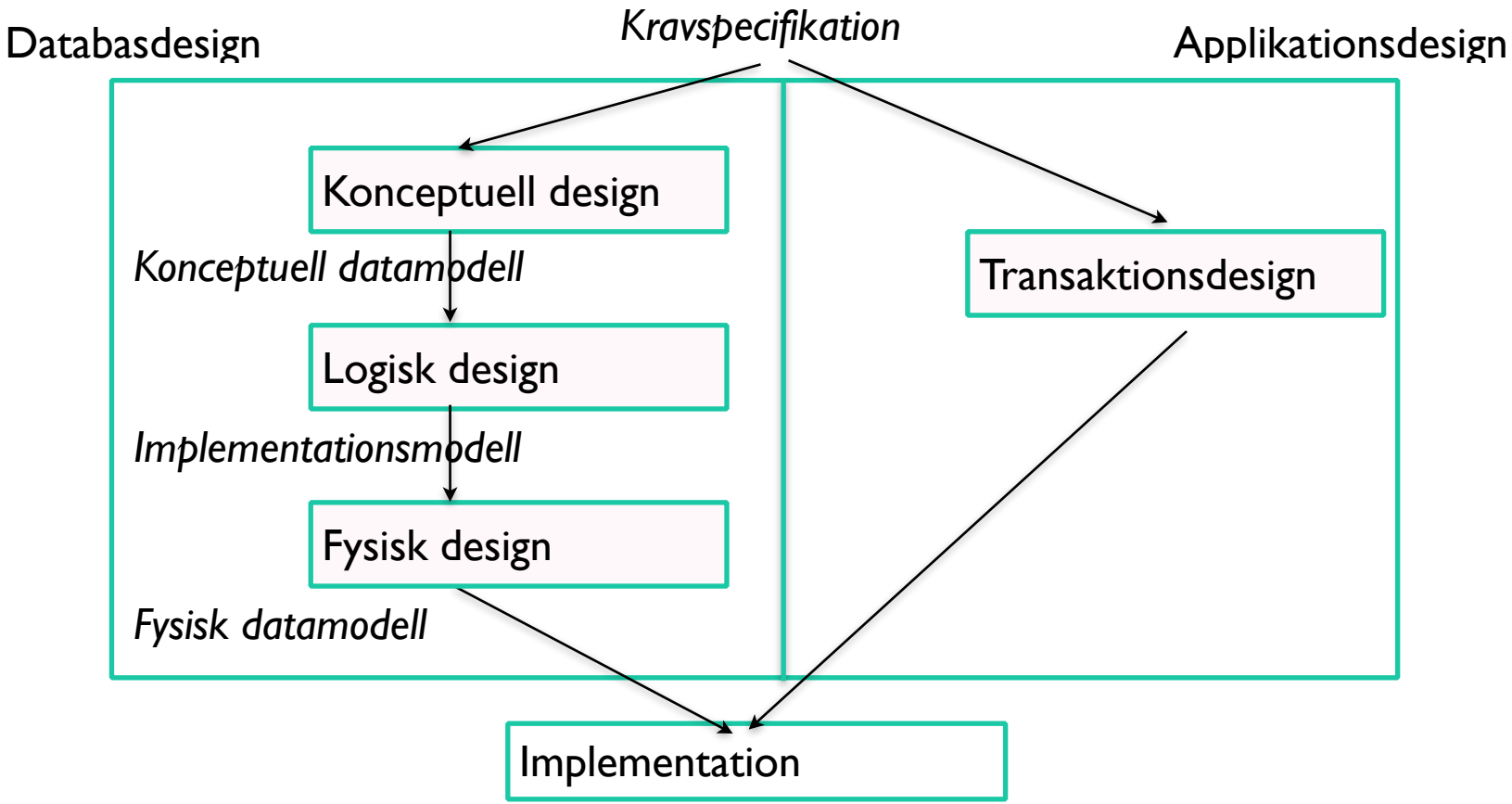
2024

LINKÖPINGS UNIVERSITET

Innehåll

- Minnesteknik
- Filorganisation
  - Osorterad
  - Sorterad
  - Hashtabell
  - Index

Programdesign, databasdesign



Fysisk datamodell

- Attributens datatyper
  - Char/Varchar, integer, float, boolean, date, …
- Integritetsvillkor
  - Primary Key, Constraint, Foreign Key, Not Null
- Lagring
  - filorganisation, indexering

Databasen lagras digitalt

Relation	*Tabell*	*Fil*
tupel	*rad*	*post*
attributvärde	*cell*	*fält*

- Datatyp = utrymme för respektive attributvärde

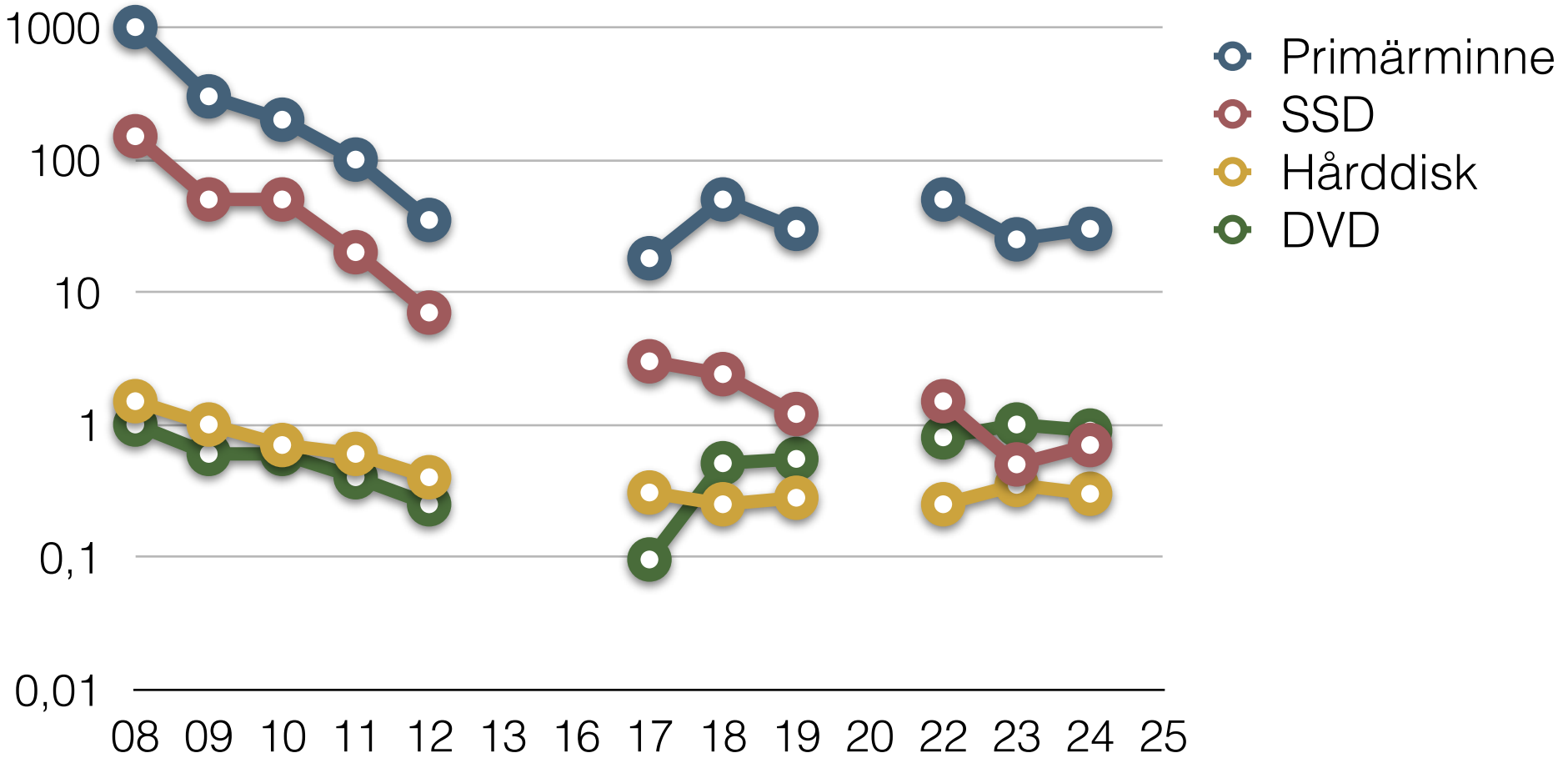
=>Varje tupel tar ett visst utrymme i filen

- Filer lagras i datorns minne … hur?

Lagra data - minnesteknik

- Primärminne/Arbetsminne (kretsteknik)
  - Flyktigt
  - Snabbt (direktaccess, nanosekunder)
  - Dyrt
- Sekundärminne (Olika typer: hårddisk (HDD, SSD), DVD, magnetband mm)
  - Permanent (relativt)
  - Långsamt (blockaccess, milli-mikro sekunder)
  - Billigt

Prisutvecklingen (kr/GB)



Hårddisk (skivbaserad)

- magnetiserbar beläggning
- åldras långsamt
- packningstäthet
- läs/skrivhuvud
- spår
- rotations-hastighet
- **Block**

Hårddisk, forts

- **Block** – minsta mängd data som överförs till PM
- 1/2kB till 4kB data

- Läsa/skriva = sök+rotation+läs/skriv = accesstid
- **Accesstid** = 5-10 millisekunder ($5\text{-}10*10^{-3}$).
- Att läsa flera block i samma spår går lite snabbare

SSD

- Kretsteknik, transistorer lagrar 1/0 (NAND)
- Organiserade i matriser, data läses ut några rader i taget. kallas **block** (ofta 4kb).
- **Block:** minsta mängd data som överförs till PM
- Läsning snabb, skrivning ofta lite långsammare, varierar: 200-2500 MB/s
- **Accesstid**: 2-20 mikrosekunder/block ($2\text{-}20*10^{-6}$)
- Slitage med ålder (sudda)

Filer och block

- HDD och SSD: block
- Filer lagras i en serie block
- Läsa in en tabell = hitta de block som filen ligger på och överför till primärminnet
- HDD:
  - Block i samma fil läggs efter varandra i samma och angränsande spår - så långt det går!
  - Med tiden uppstår fragmentering
- SSD
  - blocken direktadresserade

Insättning, sökning, (borttagning) i databasen

- `Insert into Person values ("030519-1234", "Stina", "Pettersson");`
- `Select * from Person where pnr = "030519-1234";`
- `(Delete from Person where pnr = "030519-1234";)`

Blockningsfaktor, bfr

Hur många av filens poster får plats i ett av hårddiskens block?

- R = postens storlek, B = blockstorleken
  ger bfr = ⌊B/R⌋
- en fil med r stycken poster tar filen upp
  b = ⌈r/bfr⌉ stycken block

- Vilka block som ingår lagras i filhuvudet eller i respektive block (pekare till nästa block)
- Hur stor del av filen måste vi läsa in? Beror av filorganisation!

Filorgansation

1. Hög (osorterat)
2. Ordnad sekvens (sorterat)
3. Hashtabell
4. Indexerad sekvens (index)

Hög (osorterad)

- Ny post - Hitta sista blocket (adress finns i filhuvudet). Läs in det, lägg till nya posten, skriv ut det. Om det inte får plats: länka in nytt block (ändra i filhuvudet).
  Antal blockaccesser: 2.
- Sökning - läs in första blocket, finns posten? Läs in nästa tills man hittar den sökta posten.
  Antal acesser: Medel=b/2, Max=b. (b=antal block)
- Borttagning - sök rätt block som ovan. Ta bort posten. Skriv tillbaks det reviderade blocket, som nu är delvis tomt. Flytta ”bak” resten av filen tar tid…
  Antal accesser: Medel=b/2 +1

Sökning i hög, räkne-exempel

Fil: Poststorlek, R = 100 byte
filstorlek, r = 30 000 st poster

Hårddisk: blockstorlek, B = 512 byte
Accesstid: 10 ms

Blockningsfaktorn blir: bfr=⌊B/R⌋=
=⌊512/100⌋= 5 poster per block

Filen tar upp b=⌈r/bfr⌉=⌈30000/5⌉=6000 st block

- För att hitta en viss post krävs i genomsnitt b/2= 3000 blocköverföringar = 3000*0,010s = 30 sekunder.

Hög: fördelar och nackdelar

+ Insättning går mycket fort.

+ filhuvudet litet: två länkar till block (första och sista)

- Många borttagningar leder till tomrum på skivan. (Kräver periodisk omorganisering av filen.)

- Sökning långsam

- Om posterna har variabel längd kan ändring i en sådan post få blocket att svämma över - posten måste då tas bort och sättas in på nytt.

Ordnad sekvens (sorterat)

- Posterna i filen sorteras enligt värdet på något fält i posterna. En lista över ingående block sparas i filhuvudet.
- Ny post - sök rätt block (se nedan), finns plats? = om plats i blocket, lägg in o skriv ut blocket, annars flytta ”resten av posterna” ett steg framåt (lägg in nytt block).
- Sökning - binärsökning hittar rätt block.
- Borttagning - som ny post om posten fysiskt ska tas bort. Alternativt markera posten som borttagen och organisera om då och då.

Sökning i ordnad sekvens, exempel

Samma fil som tidigare:
blockfaktor, bfr = 5 poster per block
antal block, b = 6000 block.
accesstid = 10 ms

Binärsökning, antal blocköverföringar:

$n=\lceil \log_2(b) \rceil = \lceil \log_2(6000) \rceil = 13$

- För att hitta en viss post krävs maximalt 13 blocköverföringar = 13*0,010 = 0.13 sekunder.

Ordnad sekvens, fördelar och nackdelar

- Insättning av ny post tar tid om plats skapas genom att förskjuta övriga poster i filen

- filhuvudet blir större (en länk per block)

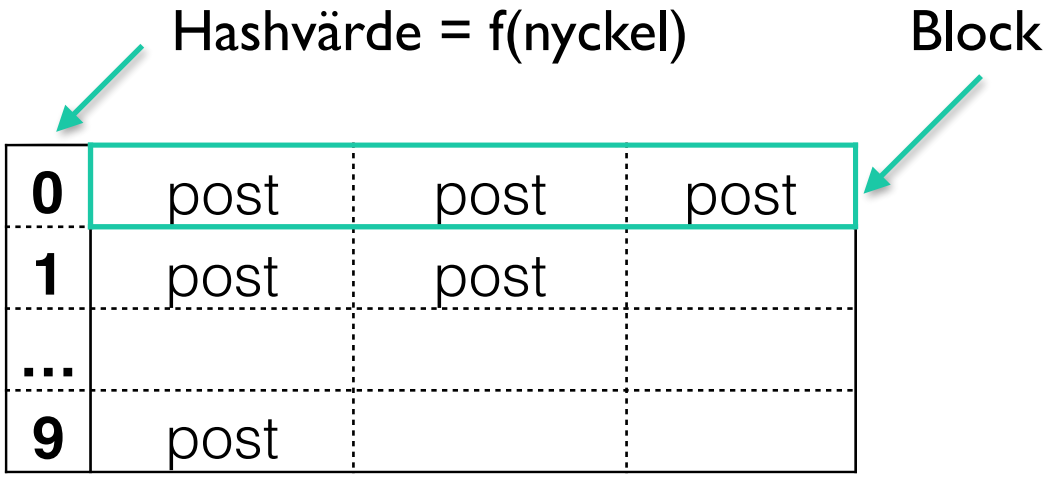
+ snabbare sökning än hög (om sökning på det filen är sorterad på, annnars funkar den som hög)

Hashstruktur

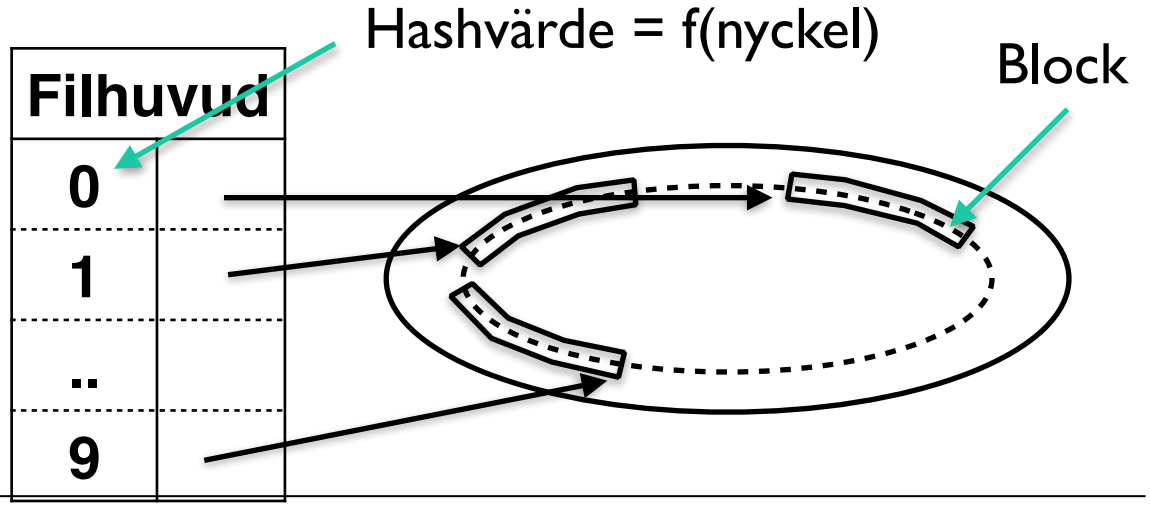
- För filer med nyckel.
- Posterna sprids över en ”hylla med fack” med hjälp av en hashfunktion.
- Varje hylla=ett block
- Kollisionshantering vid fullt block: spill-block

Hashstruktur

- Logiskt



- Fysiskt



Hashstruktur, forts

- Ny post - beräkna hashvärde, läs in blocket
  - om plats finns i blocket, sätt in
  - om inte länka in spillblock, skriv i det.
- Sökning på nyckel - beräkna hashvärde, läs in.
  - om inte rätt block följ länken till nästa tills funnen.
  - Tid för sökning: ett fåtal, men beror av hur många länkar man måste följa.
- Borttagning – hitta (som sökning ovan)
  - ta bort posten ur blocket och skriv tillbaks det ändrade blocket.

Hashstruktur, fördelar och nackdelar

+ snabb sökning (fåtal blockaccesser)

- avancerad algoritm för insättning och borttagning

- tar mer extra plats (filhuvud stort som sorterad fil plus länkfältet plus “luft”)

- kräver hela nyckeln vid utsökning och sökning på annat än nyckeln görs som om det vore en hög.

Indexerad sekvens

- Jfr innehållsförteckningen i en bok
- Indexpost pekar ut var i huvudfilen posten (med det indexvärdet) finns

- Primärindex, klustrat index (grupperade index), sekundärindex
- Glest index vs tätt index
- Not: Index lagras i en fil. Alltid sorterad.

Indexerad sekvens, forts

- Ny post - rätt block söks fram, läses in, ändras och skrivs tillbaks.
  - Vid översvämning i blocket skapas ett nytt block och en ny indexpost läggs in i indexfilen (som för sorterad fil).
- Sökning – sök nyckeln i indexfilen (som vid sökning i sorterad fil). Hämta aktuellt block från huvudfilen.
- Borttagning - sökning som ovan, ta bort(?)
  - Lämnar tomrum? Uppdatera indexfilen?

Sökning med primärindex, exempel

- Samma huvudfil som tidigare:
  antal block, b = 6000 block.
  tid för blocköverföring = 10 ms
- Antag att nyckelfältet är 9 bytes långt och adressen till block tar 6 bytes.
- Indexfilen är en sorterad fil - använd binärsökning.

Sökning i indexfilen

Indexfilens poststorlek iR blir då 15 bytes
Indexfilens blockfaktor ibfr =⌊512/15⌋= 34

Indexfilen har en post per block i huvudfilen, dvs ir=6000 poster
Indexfilen tar alltså upp ib = ⌈6000/34⌉=177 block

Hitta en viss indexpost (binärsökning i indexfilen): $\lceil \log_2(177) \rceil$ blocköverföringar =8 st.

Sedan ytterligare en för att läsa ur huvudfilen.

- Totalt 9 blocköverföringar, 0.09 sekunder.

Index, fördelar och nackdelar

- indexfilen tar extra plats i databasen (mer än hashtabell och sorterad fil).
- insättning och borttagning kräver uppdatering av indexfilen.
+ snabb sökning
+ relativt enkla algoritmer för insättning och borttagning

Sekundärindex

- index på unikt fält som filen inte är sorterad efter?
- Sekundärindexet får lika många poster som datafilen har poster.

- Detta index är **tätt** (en indexpost per post i huvudfilen, jfr primärindex: en indexpost per block = **glest**).

Sökning på sekundärindex, exempel

- samma fil som tidigare och indexposter som tidigare, men med tätt index:
  Indexfilens poststorlek iR = 15 bytes
  Indexfilens blockfaktor ibfr =⌊512/15⌋= 34
- Indexfilen tar ib =⌈30000/34⌉= 883 block
- Hitta en indexpost, binärsökning i indexfilen:
  $\lceil \log_2(883) \rceil$ blocköverföringar = 10 st
  sedan en blocköverföring ur huvudfilen.
- Totalt 11 blockaccesser, 0.11 sekunder.

Klustrat index

- Indexering på icke-unikt fält – går det?
- Sortera filen på indexfältet och gör index för första posten av varje indexvärde.

- Ny post – sortera in efter nyckeln
- Sökning – sök rätt index som tidigare, sök sedan rätt post i blocket, eller nästa block
- Borttagning – som för vanligt index.

Multipla index (indexnivåer)

- Index till indexfilen.
- För varje block i indexfilen en post i index-indexfilen…
- Kan ha godtyckligt många nivåer…

Sökning med multipelt index

Samma huvudfil och samma indexfil:
Indexfilens blockfaktor ibfr = 34
Indexfilen tar upp ib = 177 block
Index på indexfilen behöver i_2r= 177 poster.
Den tar 6 block.

Hitta index till indexfilen ($\lceil \log_2(6) \rceil$=3).
Hitta index till huvudfilen (1 överföring).
Läsa in det eftersökta blocket ur huvudfilen.

- Totalt 5 blockaccesser, 0.05 sekunder.

Typ	Ny post	Sökning	Borttagning	Extra plats
Hög	*snabb (2)*	*långsam (b/2)*	*långsam (b/2 +1)*	*övergivna poster**
Ordnad	långsam ($\log_2$(b)+1)	*snabb ($\log_2$(b))*	*långsam ($\log_2$(b)+1)*	*filhuvud + övergivna poster**
Hashstrukt	*snabb (2-3)*	*snabb (1-2)***	*snabb (2-3)*	*länkvärde + ”luft”*
Indexerad	*snabb ($\log_2$(ib) +2)*	*snabb ($\log_2$(ib) +1)*	*snabb ($\log_2$(ib) +2)*	*som ordnad + indexfilen*

Värdet anger antal accesser, b=antal block i filen, ib=antal block i indexfilen. *=Om ingen omorganisation sker. **= sökning på nyckelvärde, annan sökning som för Hög.

Filorganisationer

- Hög - sökning b/2
- Ordnad sekvens - sökning $(\log_2(b))$
- Hashstruktur - sökning konstant
- Indexstruktur - sökning i index (som Ordnad) + 1
  - primärindex (multipla nivåer)
  - sekundärindex
  - klustrat index

Fysisk design, sammanfattning

- designa databasen på fysisk nivå (datatyper, filorganisation)
- Ta med i beräkningen hur ofta tabellen används och hur (sökning/matchning mot vilket fält).
- Åtgärder:
  - Ordna datafilen - osorterad (hög) eller sorterad? På vad? Används alltid nyckel? Hashtabell!
  - Skapa index – vilka behövs? - de som söks på!

Frågor?

www.liu.se

LINKÖPINGS UNIVERSITET

Kuriosa: Johnsson Brothers Company

```
mysql> select * from parts;
+----------+--------------------+--------+--------+------+
| pk_parts | name               | color  | weight | qoh  |
+----------+--------------------+--------+--------+------+
|        1 | central processor  | pink   |     10 |    1 |
|        2 | memory             | gray   |     20 |   32 |
|        3 | disk drive         | black  |    685 |    2 |
|        4 | tape drive         | black  |    450 |    4 |
|        5 | tapes              | gray   |      1 |  250 |
|        6 | line printer       | yellow |    578 |    3 |
|        7 | l-p paper          | white  |     15 |   95 |
|        8 | terminals          | blue   |     19 |   15 |
|        9 | terminal paper     | white  |      2 |  350 |
|       10 | byte-soap          | clear  |      0 |  143 |
|       11 | card reader        | gray   |    327 |    0 |
|       12 | card punch         | gray   |    427 |    0 |
|       13 | paper tape reader  | black  |    107 |    0 |
|       14 | paper tape punch   | black  |    147 |    0 |
+----------+--------------------+--------+--------+------+
```

Ericsson serie 2000 (1983): två skivminnen (Disk drive) och en centralenhet (Central Processor).

Bildkälla: Datasaabs vänner http://www.datasaab.se/Bildarkiv/S2100/s2100.htm

Datasaab D21 1962 (Sveriges första serietillverkade dator)
Bakom: Bandstation (Tape Drive), Magnetband (Tapes), till höger Terminal, Terminalpapper

Kortstans (Card punch) c:a 1970-tal

Hålremseläsare (paper tape reader) c:a 1960-tal

Bild: TedColes - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=76086538