Fysisk databasedesign

- Fysiske lagringsstrukturer
  - Lagringsmedier
    - Egenskaper og bruksområder
    - Oppbygging av en harddisk
  - Filstrukturer
    - Representasjon av tabeller: Fil, post, felt
    - Blokker
    - Heap-filer, sekvensielle filer, hashing
    - Beregning av plassforbruk
- Aksessteknikker
  - Sekvensielt søk, binærsøk
  - Indekser
    - Tette og ikke-tette indekser
    - Flernivå-indekser og B^+-trær
    - Lage indekser med SQL
- Valg under fysisk databasedesign

Pensum: Kapittel 9

Introduksjon

- Fysisk databasedesign innebærer å tilpasse logisk skjema til et konkret DBHS.
  - En database vil fysisk være lagret på et antall filer.
  - DBA kan velge "filstrukturer".
  - DBHS bruker forskjellige "aksessteknikker" for lagring og gjenfinning av data.
  - DBA kan styre hvilke teknikker som DBHS benytter.
  - DBA kan overvåke databasen og endre fysisk design for å oppnå bedre ytelse.
- Valg av maskiner, nettverk, antall disker, lagring av databasen på forskjellige disker og så videre er relevant for god ytelse, men er ikke noe vi ser på her.

Lagringsmedier

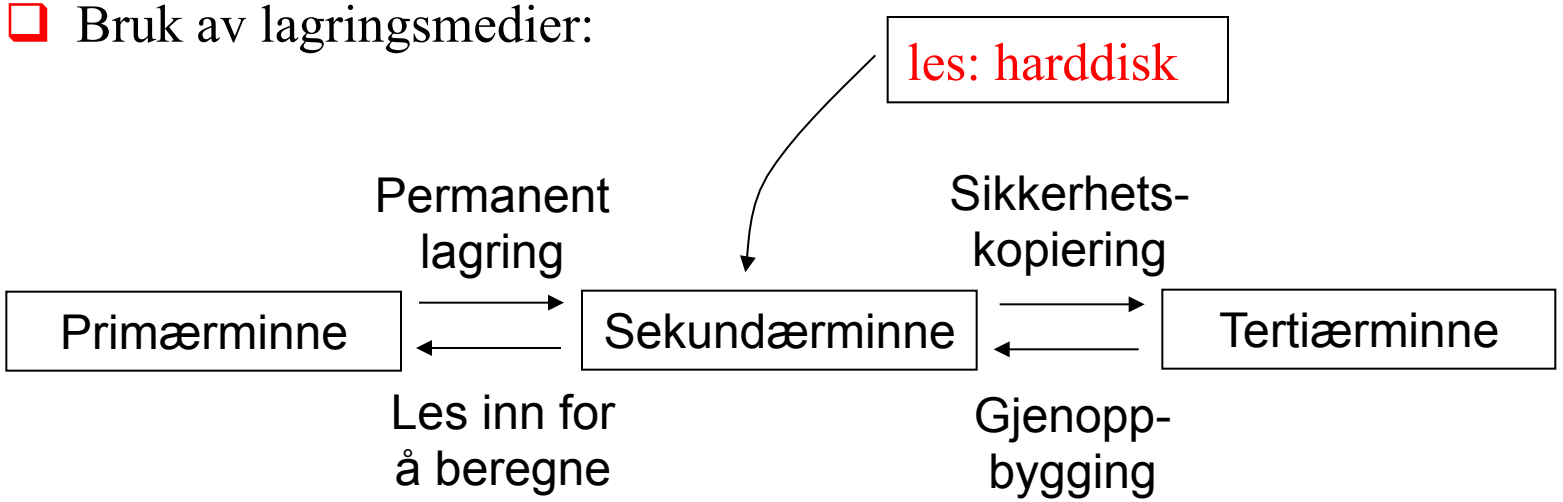
- Noen lagringsmedier:
  - Internhukommelse (RAM, ROM, …)
  - Flash-memory
  - Magnetplatelager (harddisk), diskett
  - Optiske medier (CD-ROM, …)
  - Magnetbånd
  - Robotstyrt magnetbånd-lager, CD-jukebox

- Egenskaper:
  - Aksesshastighet
  - Pris pr. byte
  - Permanent / midlertidig (avhengig av strømtilførsel)
  - Sekvensielt / direkteaksess

Høy hastighet = høy pris

Bruk av lagringsmedier

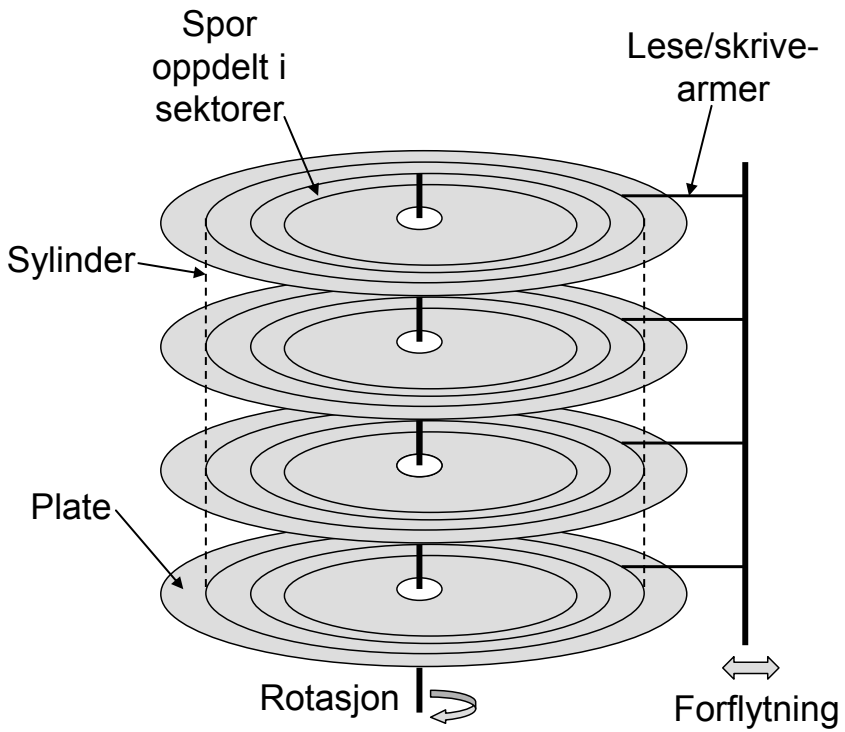
- Internhukommelsen/primærminnet er (som regel) ikke stor nok til å holde hele databasen.
  - Data blir lest inn fra ytre lager når man skal utføre beregninger.
  - På grunn av mulighet for strømbrudd må oppdaterte data skrives tilbake til ytre lager.
- Bruk av lagringsmedier:



- Observasjon: Et DBHS "skyfler" data kontinuerlig mellom internhukommelse og harddisk.

Oppbygging av en harddisk

- En harddisk består av et antall sirkulære magnetplater montert på en roterende "akse".
- Hver plate er delt inn i et antall spor, og hvert spor er delt inn i sektorer (typisk størrelse er 512 byter).
- Armer med lese/skrive-hoder kan posisjoneres over et spor.
- Sporene som dekkes av lese/skrive-hodene i en bestemt posisjon kalles en sylinder.
- Tiden det tar å lese/skrive:
  - Posisjonere lese/skrive-armene +
  - Vente til riktig sektor passerer +
  - Overføre data fra/til internhukommelse.



Flytting av mekaniske enheter tar tid!

I/O-operasjoner

- Relativ forskjell i aksesstid:
  - Internhukommelse: 100 nanosekunder
  - Harddisk: 10 millisekunder
  - 1 nanosekund=10^{-9} sekunder, 1 millisekund=10^{-3} sekunder

- Vi kan gjøre mange operasjoner i internhukommelsen for hver aksess av ytre lager (harddisk) !

- I/O-operasjoner:
  - Innlesing fra harddisk kalles en Input-operasjon.
  - Lagring til harddisk kalles en Output-operasjon.
  - I/O-operasjon er samlebetegnelse på innlesing/lagring.

- Strategi: Minimaliser antall I/O-operasjoner !

Representasjon av tabeller

- I prinsippet kan vi tenke på ethvert (?) lagringsmedium som en sekvens av byter. Hvordan lagrer vi en databasetabell (2-dimensjonal) som en byte-sekvens på platelageret?
  - Rad-for-rad eller kolonne-for-kolonne?
  - Dessuten: Hver enkelt verdi krever gjerne flere byter.

- En database lagret på ytre lager er organisert i en eller flere filer.
  - En fil består av poster som igjen består av felt.

- Forenkling:
  - Hver tabell lagres på en (egen) fil.
  - Hver rad lagres i en post, og hver verdi i et felt.

Terminologi

- Fra SQL til relasjonsmodellen til E/R til fysiske filer:

SQL	Rel.mod.	E/R	Filer
tabell	relasjon	entitetstype	fil
rad	tuppel	entitetsforekomst	post
kolonne	attributt	attributt	felt

- Merk: Begrepene betyr ikke helt det samme.

Fast og variabel postlengde

- Datatypen VARCHAR(n) gir felt av variabel lengde.
  - Hvis ett eller flere felt har variabel lengde får hele posten også variabel lengde.
  - Lagring av flere posttyper i samme fil kan også gi variabel postlengde.

- Anta at alle felt har datatyper som gir fast lengde.
  - I en fil med postlengde n kan post 1 lagres fra adresse 1, post 2 fra adresse n+1, post 3 fra adresse 2n+1, og så videre.
  - Enkelt og effektivt å gjenfinne en post.

- Med variabel postlengde er det ikke like enkelt.
  - Kan legge inn skilletegn mellom felt og poster.
  - Gjenfinning krever i så fall søking.

Blokker

- En harddisk er svært mye tregere enn internhukommelsen.
- Når vi leser fra / skriver til harddisken tar vi like godt en ”jafs” av gangen.

- En blokk er den minste enheten for overføring av data mellom internhukommelsen og ytre lager.
  - Typisk blokkstørrelse: 4KB (4 096 byte).
  - En blokk vil som regel inneholde mange poster (rader).

- DBHS holder oversikten over hvilke data som ligger i hvilke blokker. For et platelager vil blokkadressen være på formen:

  PlatelagerID+platenummer+spornummer+sektornummer

Fil delt inn i blokker

	KNr	Fornavn	Etternavn	Telefon	
Post 1	1	Elias	Hansen	99 88 77 66	Blokk 1
Post 2	2	Hulda	Akselsen	31 45 88 21	
Post 3	5	John	Boine	23 94 53 18	
	…	…	…	…	
Post 58	63	Mari	Gygre	55 66 77 88	
Post 59	64	Michael	Svensen	19 82 37 64	Blokk 2
Post 60	69	Robert	Romman	91 28 73 46	
Post 61	72	Laura	Eika	64 37 82 19	
	…	…	…	…	

- En fil blir lagret i et antall blokker.
- Det settes gjerne av litt ledig plass i hver blokk.
  - Gjør det lettere å sette inn / slette poster.
  - Fyllingsgraden (for eksempel 80%) kan styres (i noen DBHS).

Beregning av plassforbruk

- Vi bør vite ca. hvor mye plass en database krever.
- Hvor stor plass krever varetabellen til Hobbyhuset?

  Vare(VNr, Betegnelse, Pris, KatNr, Antall, Hylle)

  - Anslår antall varer til 3000.
  - Hvert tegn i CHAR/VARCHAR opptar 2 byter (Unicode).
  - Anslår at VARCHAR(n) krever 2×n byter.
  - Postlengde (se vedlegg B for datatyper i Vare):

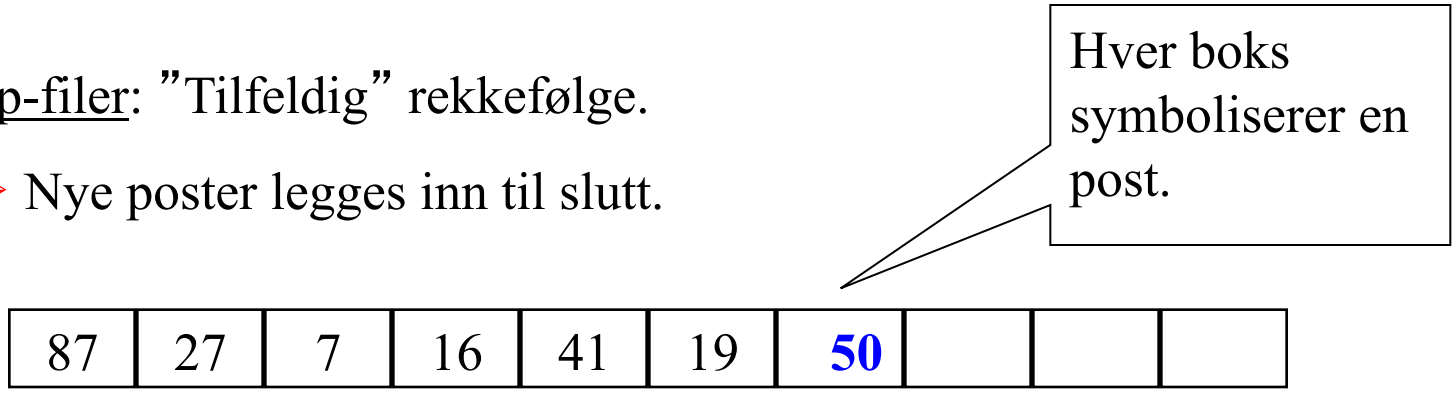
    2×5 + 2×30 + 8 + 2 + 4 + 2×3 = 90 byter.
  - Antar blokkstørrelse= 4KB og fyllingsgrad 80%.
  - Poster pr. blokk: 4000×0.8/90 = 35 (avrunder 4096 til 4000)
  - Antall blokker: 3000/35 = 86.
  - Plassbehov: 86×4000 = 340 400
  - Varetabellen krever altså ca. 340 KB.

Forenkling:

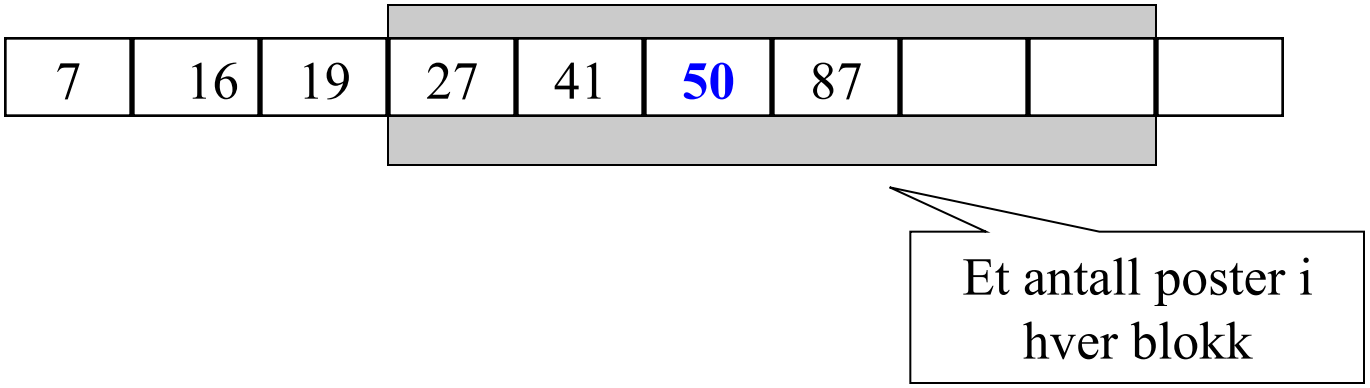
3000 × 90 / 0.8 = 337 500

Filstrukturer

- Heap-filer: ”Tilfeldig” rekkefølge.
  - Nye poster legges inn til slutt.



- Sekvensielle filer: Sortert, f.eks. med hensyn på VNr.



Sekvensielt søk og binærsøk

- Anta vi skal finne en bestemt verdi i en fil med 100.000 poster.
  - Anta vi legger 50 poster i hver blokk.
  - 100.000 poster kan lagres i **2000** blokker.
  - Det er kun antall I/O-operasjoner (blokk-innlesinger) som teller!
- Sekvensielt søk:
  - Start med blokk 1 og fortsett til verdien er funnet, eller det er slutt på filen.
  - I gjennomsnitt trenger vi **2000**/2=**1000** lese-operasjoner.
- Hvis filen er sortert (på feltet vi søker i) kan vi bruke binærsøk.
  - Undersøk den midterste blokken, og gjenta deretter binærsøk på enten venstre eller høyre "halvdel".
  - I gjennomsnitt trenger vi $\log_2$ **2000** lese-operasjoner (2^{11}=2048).

- **11** leseoperasjoner er mye bedre enn **1000**, men ikke godt nok!
  - Dessuten: En fil kan bare være fysisk sortert på **ett** felt.
  - Løsning: Indekser …

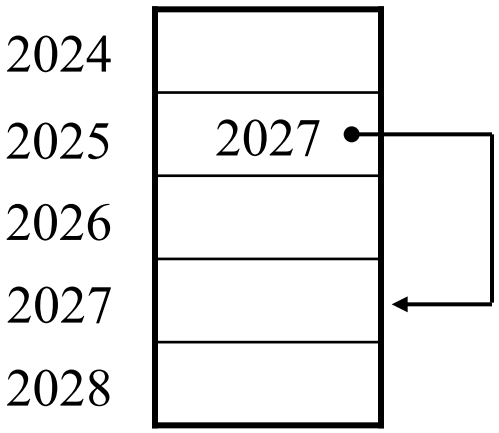
Innsetting, sletting, oppdatering

- Sletting og oppdatering:
  - Først må vi søke !
- Innsetting:
  - Heap-filer: Sett inn bakerst !
  - Sekvensielle filer: Krever søk og reorganisering.
    - Med ledig plass i blokkene kan vi (en stund) reorganisere innenfor blokkene!
- Holde sortert eller ikke ?
  - Mer effektive søk, men merarbeid ved ajourhold.
  - Slike problemstillinger dukker ofte opp (man får sjelden noe helt gratis …)

Referanser (pekere)

- Et lagringsmedium er i prinsippet en byte-sekvens.

- Enhver celle har en adresse (et tall).
  - Hvis en celle inneholder adressen til en annen sier vi den er en referanse (den referer / peker på den andre).
  - Vi tegner ofte referanser som piler.

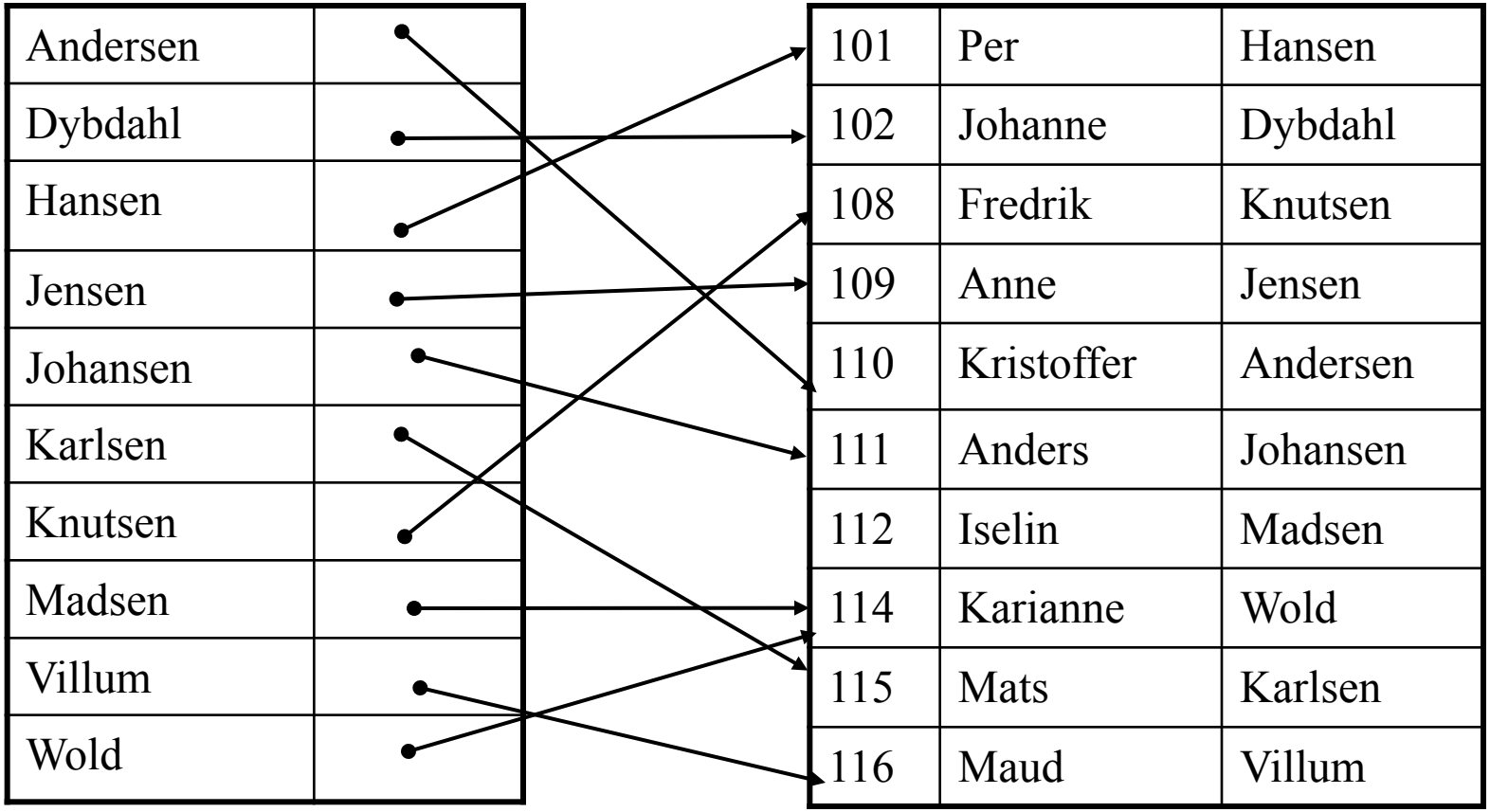


- Celle på adresse 2025 inneholder adressen (tallet) 2027.
- Kan symboliseres ved en pil fra celle 2025 til celle 2027.

Indekser

- En indeks er en datastruktur som kan gjøre aksess av data mer effektiv.
- Tenk på en indeks som en referanseliste bakerst i en bok.
  - Inneholder søkeord + referanser (=sidetall i en bok).
  - Listen er sortert med hensyn på søkeord.
  - I databaseindekser vil referansene være diskadresser.
- Indekser reduserer søketid på bekostning av større plassforbruk og merarbeid i forbindelse med innsetting, oppdatering og sletting.
  - Vi må vurdere hvilke kolonner det er hensiktsmessig å opprette indekser på.

Visualisering av indekser



Lage indekser med SQL

- Den enkleste varianten:

```
CREATE INDEX VarenavnIdx
ON Vare ( Varenavn )
```

Access: Kan også gå på Vis/Indekser.

- Indekser kan hindre repetisjoner:

```
CREATE UNIQUE INDEX EtternavnIdx
ON Ansatt ( Etternavn )
```

- Indekser kan være sammensatte:

```
CREATE INDEX NavnIdx
ON Ansatt ( Etternavn, Fornavn )
```

(Får her indeks på etternavn "på kjøpet" !)

Tette og ikke-tette indekser

- En tett indeks (engelsk: dense) inneholder en post for hver eneste post i filen.
- En ikke-tett indeks (engelsk: sparse) inneholder en post for hver blokk i filen.

- Ikke-tette indekser er mye mindre enn tette.
  - Med poststørrelse = 100 byter og blokkstørrelse = 4 KB kan vi ha 40 fil-poster i hver blokk.

- Det er kun mulig å lage ikke-tette indekser for feltet filen er fysisk sortert med hensyn på.
  - Det kan altså være maksimalt èn ikke-tett indeks for hver fil.

Antall diskaksesser ved bruk av indekser

- Eksempel: Tabellen Vare inneholder 3000 rader som hver krever 90 byter og er sortert mhp. VNr. Vi lagrer 35 poster pr. blokk, og trenger 86 blokker (med blokkstørrelse = 4 KB).

- Oppretter ikke-tett indeks mhp. VNr.
  - 10+8 byter pr. post i indeksen (VNr + referanse).
  - Indeksen vil ha en 1 post pr. blokk i varefilen (dvs. 86 poster), som kan lagres i 1 blokk.
  - Kan dermed aksessere en vilkårlig post med 2 diskaksesser!

- Oppretter en tett indeks mhp. Betegnelse.
  - 60+8 byter pr. post i indeksen (Betegnelse + referanse).
  - Det er plass til 58 indeksposter i en blokk.
  - Indeksen vil ha 1 post pr. post i varefilen (3000) = ca. 51 blokker.
  - 7 diskaksesser for å lokalisere en vare mhp. Betegnelse: 6 for binærsøk i indeksen + 1 oppslag i varefilen.

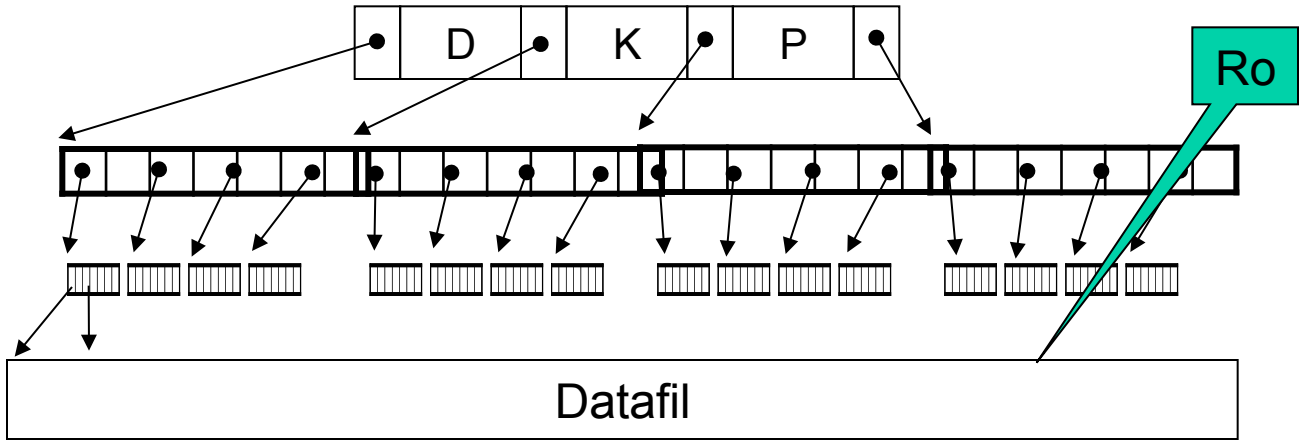
B^+-trær

- Flernivå-indekser:
  - For store indekser er det interessant å opprette "indekser-på-indekser".
  - En indeks er alltid sortert, slik at indeks på nivå 2 (3, 4, …) kan være ikke-tett.
  - Ved å gjenta dette i flere nivåer får vi en tre-struktur.
  - Søking skjer fra "roten" i treet mot "bladene".
  - Antall diskaksesser = dybden i treet.

- B^+-trær:
  - En spesiell variant av flernivå-indekser som gir like mange diskaksesser for alle søkeverdier; treet er **B**alansert.
  - Mange DBHS bruker kun B^+-trær for indeksering.

B$^+$-trær

Antall lese-operasjoner:

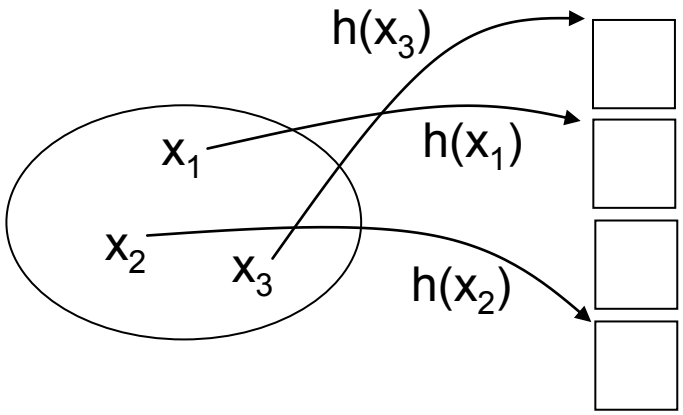


Anta vi skal finne post der Etternavn = ”Roheim”.

- Les inn rot-blokken.
- Søk i rot-blokken: Her skal vi følge pekeren helt til høyre.
- Gjenta søk på ”mellomliggende” nivåer …
- … Les inn riktig blokk fra datafilen.

Hashing

- En hashfunksjon avbilder verdier på blokk-adresser.
  - Setter av n antall blokker til filen.
  - Leke-eksempel: h(x) = x MOD 17 (rest ved divisjon)
- Velger et felt som hash-nøkkel.
  - Lagring av post: Anvend h på hash-nøkkelen.
  - Gjenfinning: Anvend h på hash-nøkkelen.
- Hva om h sender for mange poster til samme blokk?
  - Bruk "overflyt-blokker" (men dette medfører litt søking).
  - Ønsker å minimalisere både plassforbruk og antall kollisjoner.



Vurdering av hashing

- Aksess med hensyn på hash-nøkkelen:
  - Hvis vi ikke har overflyt, så får vi 1 diskaksess både ved lagring og gjenfinning (bedre kan det ikke gjøres) !

- Aksess med hensyn på andre felt:
  - Filen må nå betraktes som en heap-fil, og vi må i utgangspunktet bruke sekvensielt søk.
  - Vi kan imidlertid kombinere hashing (som filstruktur) og indekser på andre felt.

- Søk på intervaller er problematisk:
  - Hash-funksjoner bevarer generelt ikke rekkefølge: Selv om $a < b$ så er ikke $h(a) < h(b)$.

Valg av datatyper

- Tekst
  - Bruk CHAR der du vet sikkert at dataene har fast lengde (personnr, regnr for biler), og VARCHAR ellers.
  - Og LONGCHAR (Notat) for lange tekster.
- Tall
  - Velg så "liten" datatype som mulig, men slik at alle tenkelige verdier får plass.
  - Er det viktig med absolutt nøyaktighet når det gjelder desimaler?
  - Autonummererte verdier blir lagret som lange heltall (Access), fremmednøkler mot slike verdier skal også være lange heltall.
- Tekst eller tall?
  - Skal du regne med verdiene?
  - Ønsker du å vise ledende nuller (kommunenr 0812, ikke 812)?
- Beløp
  - Bruk Currency i Access
  - Bruke Decimal/Number med nøyaktig 2 desimaler i andre DBHS.

Valg av indekser

- Følgende kolonner kan med fordel indekseres:
  - Kolonner man ofte søker i, eller sorterer på.
  - Primærnøkler og fremmednøkler. Noen DBHS lager automatisk indeks på primærnøkler.
  - Kolonner som ofte brukes som koblingskolonner i spørringer.
- Følgende kolonner bør ikke/kan ikke indekseres:
  - Kolonner som kun inneholder noen få ulike verdier/mange like verdier, for eksempel Ja/Nei-kolonner. (Kan bruke såkalte bitmap-indekser…)
  - Bilder, video, lydklipp
  - Lange dokumenter (finnes spesielle teknikker, jmf søkemotorer)

- Indekser krever maskinressurser:
  - Indekser tar opp plass.
  - Indekser må vedlikeholdes ved oppdatering i tabellene.