NTNU
Norges teknisk-naturvitenskapelige Universitet

Institutt for datateknikk og informasjonsvitenskap

Fag TDT4145 Datamodellering og databasesystemer

Løsningsforslag til øving 4: Normalisering, B-trær, hashing og fysisk design.

Oppgave 1: Normaliseringsteori

a) En nøkkel for en relasjon, R, er en mengde attributter, K ⊆ R, som har følgende egenskaper:

1. Hvis t1 og t2 er to forskjellige tupler i en forekomst av R så er t1[K] ≠ t2[K].
2. Det finnes ikke noen K' ⊂ K som har egenskap 1.

En nøkkel er med andre ord en entydig og minimal indentifikator for tupler i relasjonen.
En supernøkkel for en relasjon, R, er en mengde attributter, SK ⊆ R, med følgende egenskap:

1. Hvis t1 og t2 er to forskjellige tupler i en forekomst av R så er t1[SK] ≠ t2[SK].

En supernøkkel er en mengde attributter som entydig identifiserer ett og bare ett tuppel i en forekomst (instans) av en relasjon. For supernøkler er det ikke noe krav om minimalitet. En supernøkkel kan altså inneholde attributter som er overflødige med hensyn på supernøkkelens egenskap som entydig identifikator. Nøkler er supernøkler som er minimale i den forstand at vi ikke kan ta bort ett eller flere attributter og stå igjen med en supernøkkel.

En funksjonell avhengighet, X → Y, er en restriksjon (eng: constraint) mellom to mengder attributter, X og Y. Restriksjonen er at alle par av tupler, t1 og t2, som har samme verdier for attributtene i X (t1[X] = t2[X]), må ha samme verdier for attributtene i Y (t1[Y] = t2[Y]).

b) Tillukningen til X, X+ , er mengden av alle attributter som er funksjonelt avhengige av X.
Algoritme som beregner X+ med hensyn på en mengde funksjonelle avhengigheter, F:

```
X+ := X;
REPEAT
      oldX+ := X+;
FOR each functional dependency Y → Z in F DO
IF Y Í X+ THEN X+ := X+ ∪ Z;
UNTIL (X+ = oldX+);
```

c) a+ = ae, ab+ = abcde og e+ = e.

d) En mengde attributter er en supernøkkel for en tabell når tillukningen til attributtene i mengden inneholder alle attributtene i tabellen. Vi kan derfor undersøke om en mengde attributter er en supernøkkel ved å beregne tillukningen og sjekke om den inneholder alle attributtene i tabellen. En supernøkkel er en nøkkel dersom den er minimal . Det vil si at vi ikke kan fjerne ett eller flere attributter fra en nøkkel og stå igjen med noe som er en supernøkkel. Vi sjekker om en supernøkkel er minimal ved å ta bort ett og ett attributt og for hver restmengde undersøke om den er en supernøkkel.

e) Kaller mengden av funksjonelle avhengigheter i R for F. For å ha tapsløs-join-egenskapen må det da være slik at enten (($R_1 \cap R_2 \rightarrow (R_1 - R_2)$) eller (($R_1 \cap R_2 \rightarrow (R_2 - R_1)$) er med i F^+. Dette er det samme som å se om snittet av projeksjonene er supernøkkel i en av projeksjonene.

f) 1. R_1(a, b, c) og R_2(b, c, d)

$R_1 \cap R_2$ er (b,c). $R_1 - R_2$ er (a). $R_2 - R_1$ er (d). Ikke tapsløs join.

2. R_1(a, b, d) og R_2(b, c, d)

$R_1 \cap R_2$ er (b,d). $R_1 - R_2$ er (a). $R_2 - R_1$ er (c). Tapsløs join

3. R_1(a, b, d) og R_2(b, c)

$R_1 \cap R_2$ er (b). $R_1 - R_2$ er (a, d). $R_2 - R_1$ er (c). Tapsløs join

g) En tabell er på tredje normalform (3NF) når det for alle funksjonelle avhengigheter, $X \rightarrow A$, som gjelder for tabellen, er slik at

1) X er en supernøkkel i tabellen, eller
2) A er et nøkkelattributt i tabellen.

h) For å avgjøre om dekomponeringen er en god løsning, må vi undersøke fire forhold:

1. Attributtbevaring. Vi har attributtbevaring siden alle attributter i R er med i minst en komponenttabell.
2. FD-bevaring. Ut fra de funksjonelle avhengighetene som gjelder i komponenttabellene, kan vi utlede alle funksjonelle avhengigheter som gjelder i R. Dekomponeringen oppfyller dermed kravet om bevaring av de funksjonelle avhengighetene.
3. Tapsløst-join-egenskapen. R1 og R3 kan joines tapsløst siden det felles attributtet (A) er en (super-)nøkkel i R1. Resultatet av dette joinet kan joines tapsløst med R2 siden det felles attributtet (C) er en (super-)nøkkel i R2. Vi er da (tapsløst) tilbake tilbake til utgangspunktet og kan konkludere at dekomponeringen har tapsløst-join-egenskapen.
4. Normalform. De funksjonelle avhengighetene som gjelder i komponenttabellene, har alle en supernøkkel som venstresideattributt. De tre komponenttabellene oppfyller dermed kravet til Boyce-Codd normalform (BCNF).

Siden alle fire kravene er oppfylt, er denne dekomponeringen en god løsning.

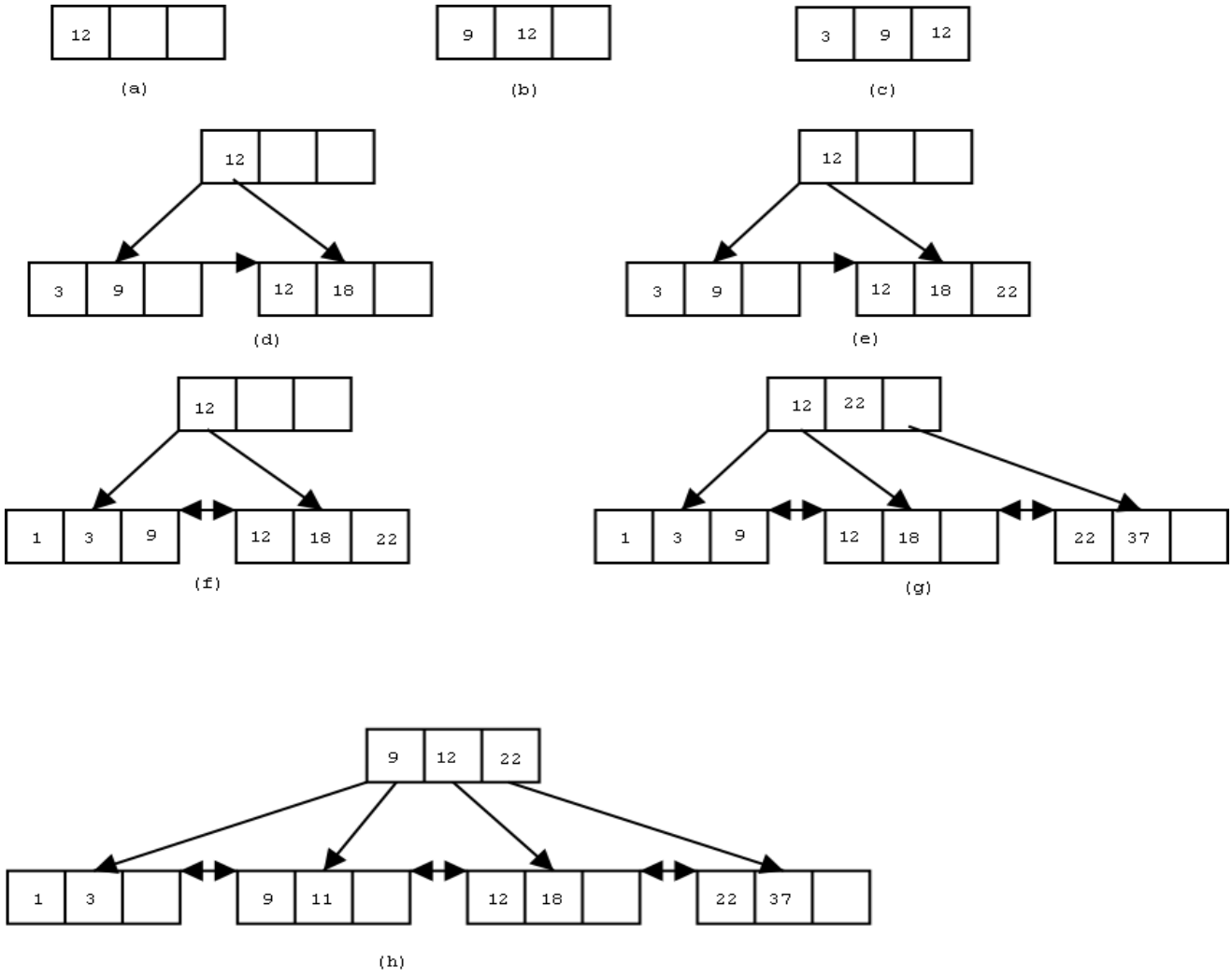
Oppgave 2: B-trær

(a) Alle postene i dette B-treet er indeksposter. Vi antar hver indekspost er 8 bytes = 4 byte (primærnøkkel) + 4 byte (blokkidentifikator). Hvis vi antar ingen overhead i blokkene (dvs. det er ingen administrasjonsfelt i blokkene) og 67% fyllgrad, får vi plass til 4096/8 *67/100 = 343 poster i gjennomsnitt i hver blokk. Da trenger vi 200000/343 = 584 blokker på nederste nivå. Da får vi plass til alle 584 indekspostene i to blokker på nivået over. Indekspostene til de to i sin turn får plass i en rotblokk.

(b) Rotblokka og en av to blokker på det neste nivået må leses inn fra disk. Løvnodeblokka må leses inn fra disk, oppdateres og skrives til disk. Datalagringsblokka må leses inn, oppdateres og skrives til disk. Til sammen 6 diskaksesser. Vi antar da at ingen av blokkene ligger i buffer på forhånd.

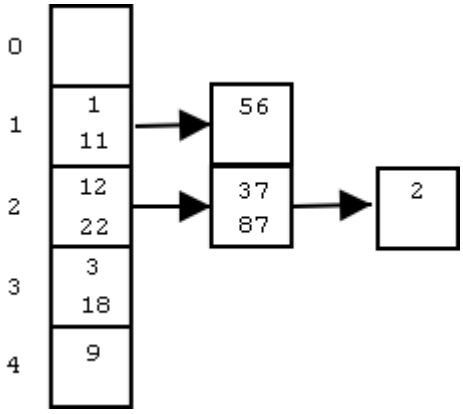
(c) Vi antar 10 millisekunder for å lese inn eller skrive blokka. Vi får fire diskaksesser, dvs. 40 millisekunder. Sannsynligvis tar det mindre tid, da den siste blokka som leses inn også er den som skal skrivest ut. Dvs. søketida kan være liten for den siste diskaksessen.

(d) **Se vedlagte figur**. Det er vist ett B-tre etter hver innsetting.



Oppgave 3: Hashing

Se vedlagte figur. Vi får tre overløpsblokker.



Oppgave 4:

For å løse denne oppgaven må man gjøre ein analyse som følger. Trenger 400*100/4096 blokker=10 blokker for å lagre Ansatte-relasjonen. Typisk vil man uten indekser trenge en table-scan for å finne en bestemt post i relasjonen. Selv om man i utgangspunktet i gjennomsnitt kun skulle trenge å gå gjennom halvparten av blokkene, vil sannsynligvis hele relasjonen bli lest pga. at man ikkje man angi til SQL at ”ett svar er nok”. De mest fornuftige alternativene til indeksering med utgangspunkt i spørringene er 1) ingen indeks, 2) indeks på persnr, 3) indeks på kombinasjonen navn og avdnavn, og

4) to indekser, en på persnr, og en på navn/avdnavn. Tidsforbruk med de forskjellige indeksalternativene er:

Operasjon	Ingen IDX	Pnr IDX	Navn/avdnavn IDX	PNR og navn/avdnavn IDX
Q1	10	2	10	2
Q2	10	10	2	2
I1	2	4	4	6
	10p1+10p2+2(1-p1-p2)= 8p1+8p2+2	10p2+2p1+4(1-p1-p2)= 10p2+2p1+4-4p1-4p2= 6p2-2p1+4	6p1-2p2+4	2p1+2p2+6(1-p1-p2)= 2p1+2p2+6-6p1-6p2= 6-4p1-4p2
p1=0.1, p2=0.5	6.8	6.8	3.6	3.6

Man ser da at man får best gjennomsnittlig ytelse ved bruk av *enten* en indeks på navn/avdnavn alene, eller to indekser, en på PNR, og en på navn/avdnavn.

Legg merke til at ved en feiltagelse var oppgaven upresis på kostnad ved oppdatering av indeks, vi antar det vil vere studentar som kan tolke oppgaven slik at kostnaden ved oppdatering av indeks som (sekvensiell) lesing av to diskblokker i *en* diskoperasjon. Dette vil gi følgende resultat:

Operasjon	Ingen IDX	Pnr IDX	Navn/avdnavn IDX	PNR og navn/avdnavn IDX
Q1	10	2	10	2
Q2	10	10	2	2
I1	2	3	3	4
	10p1+10p2+2(1-p1-p2)= 8p1+8p2+2	10p2+2p1+3(1-p1-p2)= 10p2+2p1+3-3p1-3p2= 7p2-p1+3	7p1-p2+3	2p1+2p2+4(1-p1-p2)= 2p1+2p2+4-4p1-4p2= 4-2p1-2p2
p1=0.1, p2=0.5	6.8	6.4	3.2	2.8

I dette tilfellet oppnår man best gjennomsnittlig ytelse ved bruk av to indekser, en på PNR, og en på navn/avdnavn.