Data øving 4

## Oppg.1

a)

* Nøkkel: Identifiserende data i en database for et sett med annen data. Eks. BookID, Personnr.
* SuperNøkkel: Kan bestå av en eller flere felt som til sammen bestemmer en rad i en tabell. En supernøkkel kan miste deler av nøkkelen sin uten å miste viktig informasjon (redundancy). En nøkkel som ikke er redundant kalles candidate key.
* Funksjonell avhengighet: et attributt i en relasjon bestemmes av et annet attributt.I en relasjon vil den ene verdien bestemmes av den andre.

b)

Closure X+ er en samling av alle attributtene vi kan nå/bestemmes via X.

Algoritme: ???

c)

**a+:** a, e

**ab+:** a+: (a, e) b+: (b, c) --> a, b, c, d, e

**e+:** e

d)

Hvis tilslutningen av attributtene inneholder alle attributtene i tabellen, er mengden med attributter en supernøkkel.

En supernøkkel er også en nøkkel hvis supernøkkelen ikke inneholder flere attributter enn det som er nødvendig for å nå/definere alle attributtene i entiteten.

e)

Dersom: ( R1 ∩ R2 ) → R1 eller ( R1 ∩ R2 ) → R2, Har oppdelingen i R1 og R2 tapløs-join-egenskapen.

f)

1. R1(a, b, c) og R2(b, c, d) - NEI

2. R1(a, b, d) og R2(b, c, d) - JA

3. R1(a, b, d) og R2(b, c) - JA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | Full linje |
| R1 | a | a | a |  | x |
| R2 |  | a | a | a | x |
| ----------------- | ---------------- | ----------------- | ----------------- | ---------------- | ----------------- |
| R1 | a | a | a | a | JA |
| R2 |  | a | a | a | x |
| ----------------- | ----------------- | ----------------- | ----------------- | ----------------- | ----------------- |
| R1 | a | a | a | a | JA |
| R2 |  | a | a |  | x |

g)

* Krever at relasjonen også er 2NF
* En relasjon R er på tredje normalform hvis alle ikke-trivielle funksjonelle avhengigheter i R på formen X→A tilfredsstiller minst ett av følgende
  + X er en supernøkkel i R
  + A er et nøkkelattributt i R

h)

Alle attributtene i R er med i en eller flere av dekomponeringene.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | full linje |
| R1 | a | a |  |  | x |
| R2 |  |  | a | a | x |
| R3 | a | a | a | a | JA |

Det virker som at denne dekomponeringen er en god løsning.

## Oppg.2

nøkler: 2369, 3760, 4692, 4871, 5659, 1821, 1074, 7115, 1620, 2428, 3943, 4750, 6975, 4981 og 9208

Hash funksjon: h(K) = K mod 8

|  |  |
| --- | --- |
| blokker | Part# |
| 0 | 3760, 9208 |
| 1 | 2369 |
| 2 | 1074 |
| 3 | 5659, 7115 |
| 4 | 4692, 1620, 2428 |
| 5 | 1821, 4981 |
| 6 | 4750 |
| 7 | 4871, 3943, 6975 |

vet at hver blokk bare kan ha to post#, så vi har overflyt(hvis man kan si det) i blokk 4 og 7. for å løse dette må vi bruke overflow-chaining

Block-access: 2 \*(2/15) + 1\*13/15 = (2\*2 + 1\*13) / 15 = 17/15

## Oppg.3

hashfunksjon: h(K) = K mod 128

dybde: 2

blokker: 4

2^d = 2^2 = 4 adresser

|  |  |  |
| --- | --- | --- |
| Blokk | rest | part# |
| 0 | 0-31 | 4871, 5659, 1821, 4750 |
| 1 | 32-63 | 3760, 1074 |
| 2 | 64-95 | 2369, 4692, 7115, 1620, 6975 |
| 3 | 96-127 | 2428, 3943, 4981, 9208 |

Ser at det blir overflyt(overflow) i blokk 0,2 og 3, og at siden dybden skal være 2, vil blokk 2 ikke kunne løses ved bare en iterasjon av splittelse. gjør mellom operasjonen og får 2 blokker med d\*=3

|  |  |  |
| --- | --- | --- |
| blokk | d\* | part# |
| 0 | 3 | 4871, 5659 |
| 1 | 3 | 1821, 4750 |
| 2 | 2 | 1074 |
| 3 | 3 | 2369, 4692 |
| 4 | 3 | 7115, 1620, 6975 |
| 5 | 3 | 2428, 3943 |
| 6 | 3 | 4981, 9208 |

som nevnt tidligere, ser vi at det blir overflytt på blokk 4, så vi splitter på nytt, med d\*=4

|  |  |  |
| --- | --- | --- |
| blokk | d\* | part# |
| 0 | 3 | 4871, 5659 |
| 1 | 3 | 1821, 4750 |
| 2 | 2 | 1074 |
| 3 | 3 | 2369, 4692 |
| 4 | 4 | 7115, 1620 |
| 5 | 4 | 6975 |
| 6 | 3 | 2428, 3943 |
| 7 | 3 | 4981, 9208 |

## Oppg.4

a)

antall studenter: 400 000

hver index: 4+4 = 8 bytes (pointer og id)

Vi antar at blokkene vil være tilnærmet 2/3 utfylt, da det vil være usannsynlig at det fylles helt opp, og det vil bli en del skriverom + tomrom på blokken.

Dermed får vi plass til 2/3\*4096/8=341 indeksposter pr blokk

dette gir oss med 400 000 studenter; 400000/341 = 1173,020527859238 = 1174 ≈ 1175 blokker på nederste nivå for å få plass til alle 1174 indekspostene i den øverste blokken

b)

Vi antar at ingen av blokkene ligger i buffer.

1 Dersom vi skal legge til student nr. 200.001 i databasen, må først rotblokken leses for å finne ut hvilken løvnode som må endres.

2+3 Deretter må løvnoden leses og skrives.

4+5 Tilslutt må datalagringsblokken leses og skrives for å oppdatere B+-treet. Dermed har vi i alt 5 diskaksesser

c)

har fra wikipedia: <https://en.wikipedia.org/wiki/Hard_disk_drive>, at lesetiden er mellom 2,5 og 10 ms for en harddisk i 2017. får da for lesetider 2,5ms, 5ms, og 10ms; med antagelse om at lese/skrive er to operasjoner som tar like lang tid

3\*2(2,5ms) = 15 ms

3\*2(5ms) = 30 ms

3\*2(10ms) = 60 ms

d) tall som skal inn: 12, 7, 9, 3, 18, 22, 1, 37, 11

1.

|  |  |  |
| --- | --- | --- |
| 12 |  |  |

2.

|  |  |  |
| --- | --- | --- |
| 7 | 12 |  |

3.

|  |  |  |
| --- | --- | --- |
| 7 | 9 | 12 |

4.



5.



6.



7.



8.



9.



## Oppg.5

ansatte = 100000

blokker = 1250

100 000 ansatte, fordelt på 1250 blokker gir 100000/1250 = 80 personer per blokk

søkenøkkel = ssn

gjennomsnittlig aksessering = 1.2 blokker.

1. har samtlig informasjon, så aksesserer direkte inn i lagringsdatablokken, da den ikke trenger å lete etter entiteten. 1
2. 1.2, da det søkes på ssn, evt 2. da man ikke kan aksessere 0.2 blokker
3. 3 nivåer, pluss tilgang til lagringsblokken, gir 4 aksesser
4. 0.1\*100000 = 10000 brukere, er 80 ansatte per blokk, så 10000/80 gir 125 blokksøk. siden det brukes 1,2 søk på ssn, får vi 125\*1.2 = 150 blokker antar jeg er et godt estimat
5. ingen anelse, tipper 3 for å finne ssn til hver instans av distinct lname, og uvisst antall lname som matcher gir 3\*1,2\*n, der n er antall lname som matcher