Oppgaver INF3100

Dette heftet inneholder først og fremst løsningsforslag til oppgaver fra læreboken, men også noen ekstraoppgaver. Ekstraoppgavene er gitt navn etter hvilket kapittel de tilhører, og løsningsforslag til ekstraoppgavene er derfor plassert under det tilhørende kapittelet.

Oversikt over innholdet

Ekstraoppgaver INF3100

Ekstraoppgavene er gitt navn etter hvilket kapittel de tilhører.

Oppgave 3.x.1

Betrakt følgende to mengder med FDer:

$$F = \{A \rightarrow C,\ AC \rightarrow D,\ E \rightarrow AD,\ E \rightarrow H\}$$
$$G = \{A \rightarrow CD,\ E \rightarrow AH\}$$

Sjekk om de to mengdene er ekvivalente.

Oppgave 3.x.2

Gitt relasjonen *R(A,B,C,D)* med FD-ene $AB \rightarrow CD$ og $B \rightarrow D$.

- Hvilke kandidatnøkler har R?
- Hvilken normalform har R?
- Hvordan skal R dekomponeres for å få en høyere normalform?
- Vis at dekomponeringen er tapsfri.
- Vi innfører så MVDen $A \twoheadrightarrow D$. Vis at da holder $A \rightarrow D$.

Oppgave 3.x.3

Gitt en relasjon R(A, B, C, D, E) og FD-ene

$$F = \{DE \rightarrow AD,\ AB \rightarrow BC,\ BC \rightarrow CD,\ CD \rightarrow ABE\}$$

som gjelder for R.

1. Finn en minimal overdekning av F
2. Finn alle elementære FDer i R
3. Hvilken normalform har R?
4. Dekomponer R til BCNF. Avgjør om dekomposisjonen er FD-bevarende.
5. Finnes det noen FD-bevarende dekomposisjon til BCNF?

Oppgave 3.x.4

Vis at hvis $X \twoheadrightarrow Y$, så $X \twoheadrightarrow W$ der W er resten av attributtene.

Oppgave 3.x.5

Avgjør for hvert av følgende problemer om den angitte (tapsfrie) dekomposisjonen kan ha støyinstanser. Hvis den kan det, gi et eksempel på støyinstanser.

(a) R(A,B,C) med FDer A→C og B→C, dekomposisjonen {AB, BC, AC}.

(b) S(A,B,C,D) med FDer A→B, B→C og C→D, dekomposisjonen {AB, BC, CD}.

(c) T(A,B,C,D) med FDer AB→D og AC→D, dekomposisjonen {ABC, ABD, ACD}.

Oppgave 3.x.6

For hvert av punktene (a)-(d), gjør følgende: (i) Dekomponer relasjonen tapsfritt til BCNF. (ii) Avgjør om dekomposisjonen kan ha støyinstanser.

(a) Relasjonen R(A,B,C,D,E,F,G) med FDene ABC → E og AB → DF.
(b) Relasjonen S(A,B,C,D,E,F) med FDene BCDE → F og AD → B.
(c) Relasjonen T(A,B,C,D,E,F,G) med FDene ABD → F og CDE → G.
(d) Relasjonen U(A,B,C,D,E,F) med FDene ABC → E, C → D og CF → A.

Oppgave 6.x.1

Gitt følgende skjema for å registrere selskaper med meny, gjester og hvem som var arrangør/vert:

Person(<u>pid</u>, navn)
Selskap(<u>dato</u>, meny, vert)
Gjest(<u>pid, dato</u>)

I Gjest er pid fremmednøkkel til Person(pid) og dato til Selskap(dato). Primærnøkler er understreket.

Oppgave: Finn navnet på de gjestene som har vært med i samtlige selskaper arrangert i tidsrommet 2000-2009 av Aschehoug.

Oppgave 10.x.1

Følgende er et skjema over studentforeninger:

Studentforening(<u>forening, verv, person</u>)

Verv er 'leder', 'nestleder', 'kasserer', 'arrangementssjef', mm. Informasjonen kan tolkes som en graf der hver node representerer en forening eller en person, med kanter som viser hvilke verv de enkelte personene har.

(a) Finn ut om det er en sti mellom Heidi Bø og Eirik Mo der stien inneholder 5 eller færre personer (når vi teller med Heidi og Eirik i antall personer). Skriv ut personstien (eller stiene, hvis det er flere).

(b) Finn ut om grafen inneholder noen sykler som starter med 'Siv Dahl'.

Oppgave 13.x.1

Anta at vi har 7 disker organisert i RAID6, med diskene d3, d5, d6 og d7 som datadisker og diskene d1, d2 og d4 som paritetsdisker. Anta at datadiskene initielt inneholder følgende blokker:

d3: 00001111
d5: 00000100
d6: 10011011
d7: 11000101

a) Beregn innholdet i d1, d2 og d4.

b) Anta at d6 endrer sitt innhold til

   d6: 10000100

   Angi hvilke øvrige endringer som må foretas, og hvilke disker som må involveres i oppdateringene.

c) Anta at d5 kræsjer. Hvordan rekonstruerer vi innholdet?
d) Anta at både d6 og d7 kræsjer (etter at d5 er rekonstruert). Hvordan rekonstruerer vi innholdet?

Oppgave 16.x.1

Optimaliser treet i 16.3.1 (a).

Oppgave 16.x.2

Gitt en hotelldatabase med relasjonene

Kunde(knr, navn, adr)
Hotellrom(knr, ankdato, avrdato, rom ,pris)
Service(knr, dato, stype, prid)

Kunde og **Hotellrom** burde være rimelig selvforklarende. **Kunde** har primærnøkkel (knr) og **Hotellrom** primærnøkkel (knr, ankdato). **Service** inneholder informasjon om servicetilbud som hotellgjestene har benyttet seg av, som spa, frisør, restaurantbesøk mm., med prisinformasjon (som blir lagt til den endelige hotellregningen ved avreise). **Service** har primærnøkkel (knr, dato, stype).

I relasjonsalgebrauttrykket under er **K**, **H** og **S** forkortelser for henholdsvis **Kunde**, **Hotellrom** og **Service**.

$$
\begin{aligned}
&\delta\,(\\
&\quad \pi_{\mathbf{K}.\text{navn},\ \mathbf{S}.\text{stype}}\,(\\
&\qquad \sigma_{\mathbf{S}.\text{dato} \geq \mathbf{H}.\text{ankdato} \textbf{ and } \mathbf{S}.\text{dato} \leq \mathbf{H}.\text{avrdato}}\,(\\
&\qquad\quad \sigma_{\mathbf{K}.\text{knr} = \mathbf{H}.\text{knr}}\,(\\
&\qquad\qquad \mathbf{K} \times \sigma_{\mathbf{H}.\text{ankdato} \geq \text{2016-01-01} \textbf{ and } \mathbf{H}.\text{avrdato} \leq \text{2016-01-15}}\,(\sigma_{\mathbf{S}.\text{knr} = \mathbf{H}.\text{knr}}\,(\mathbf{S} \times \mathbf{H})))))
\end{aligned}
$$

(a) Hva uttrykker spørringen?

(b) Optimaliser uttrykket.

Oppgave 18.x.1

For hvert av tilfellene (a), (b) og (c): Beskriv hva som skjer hvis vi forsøker å eksekvere dem med SI-protokollen FUW (First Updater Wins). Sett selv inn operasjoner av formen

$l_i(x)$	- T_i ber om eksklusiv lås på element x
$u_i\,(x)$	- T_i frigir låsen på x
c_i	- commit
a_i	- abort

(a) $r_1(a); r_2(a); r_1(b); w_2(a); w_1(a); r_2(b); w_2(b)$

(b) $r_1(a); r_2(b); w_2(b); r_1(b); w_1(a); w_1(b)$

(c) $r_1(a); r_2(a); r_3(a); w_1(b); r_3(b); w_2(a); w_3(a); w_2(b)$

Oppgave 20.x.1

Anta at det i Paxos consensus-protokollen er 3 noder og at node 1 vil fremme dekretet "NEI" og de andre nodene dekretet "JA" hvis de får muligheten til det.

(a) Forklar forløpet i Paxos consensus hvis node 1 er utpekt til å starte protokollen og det bare trengs én avstemningsrunde for å avslutte protokollen.

(b) Forklar forløpet hvis node 1 og node 2 starter protokollen samtidig og en av de to lykkes med sin avstemningsrunde.

(c) Anta at node 1 og node 2 starter protokollen samtidig. Gi et eksempel på et forløp der Paxos consensus ikke terminerer.

(d) Anta at det ikke er noe i veien med nettet eller nodene slik at alle meldinger kommer frem i rimelig tid. Hvis alle nodene ønsker å fremme samme forslag til dekret (f.eks. "JA"), vil Paxos consensus da alltid terminere? Forklar.

(e) Betrakt følgende situasjon: Node 1 er utpekt til å starte protokollen, og "overbooker" ved å sende fase 2a-meldinger til både node 2 og 3. Den planlegger å benytte enten {1,2} eller {1,3} som det egentlige kvorumet i fase 2, avhengig av hvilken som svarer først; node 1 trenger bare svar fra en av dem for å kunne fortsette med fase 3. Anta at det bare trengs én avstemningsrunde fra node 1 for å avslutte protokollen fordi node 2 besvarer alle meldingene den får fra node 1. Imidlertid er nettverket til node 3 veldig tregt, så avstemningsrunden til node 1 avsluttes uten at node 3 mottar noen meldinger. Først etterpå oppfører nettet seg normalt igjen, og node 3 mottar en svært forsinket fase 2a-melding fra node 1, men ingen fase 3-melding. Forklar hvordan node 3 kan benytte protokollen til å få resultatet av avstemningen.

Oppgave 20.x.2

Anta at det i Paxos commit-protokollen er 5 noder som initielt har følgende prosesser:
Node 1: En ressurshåndterer, en akseptor og lederen
Node 2: En ressurshåndterer og en akseptor
Node 3: En ressurshåndterer og en akseptor
Node 4: En ressurshåndterer
Node 5: En ressurshåndterer

(a) Forklar forløpet i Paxos commit hvis alle ressurshåndtererne kan committe sin deltransaksjon og hver av Paxos consensus-instansene bare trenger én avstemningsrunde.

(b) Forklar forløpet hvis ressurshåndtererne på node 1-4 kan committe, mens node 5 må abortere sin deltransaksjon, og hver av Paxos consensus-instansene bare trenger én avstemningsrunde.

(c) Forklar forløpet hvis alle ressurshåndtererne kan committe sin deltransaksjon, men nettverket til node 5 er veldig tregt, så det drøyer med å komme noen respons fra ressurshåndtereren på node 5.

(d) Forklar forløpet hvis alle ressurshåndtererne kan committe sin deltransaksjon og alle consensus-instansene har påbegynt avstemningsrunde 0, men deretter blir nettverket til node 3 veldig tregt slik at det drøyer med å komme respons fra akseptorprosessen på denne noden.

Anta at vi er i en situasjon der ressurshåndtereren på node 5 har fremmet dekretet "prepared" og lederen (på node 1) har fremmet dekretet "aborted" i samme instans av Paxos consensus (altså den instansen som ble initiert av ressurshåndtereren på node 5).

(e) Gi et eksempel på et forløp der dette kan skje.

(f) Gi et eksempel på et videre forløp der det i instansen blir fullført en avstemningsrunde med dekretet "aborted".

(g) Gi et eksempel på et videre forløp der det i instansen blir fullført en avstemningsrunde med dekretet "prepared".

Oppgave 2007.x.1

Ekstraoppgave knyttet opp mot eksamen INF3100 våren 2007.

A. La oss si at vi slår sammen Hendelse og Epikrise i én tabell.
   i. Hvordan vil tabellen se ut?
   ii. Hvilke FDer gjelder i den nye tabellen?
   iii. Hvilke kandidatnøkler har den?
   iv. Hvilken normalform er den på?

B. Vi skal se på en liten utvidelse av akvariebutikkdatabasen.

   Butikken har flere rabattordninger - bl.a. en for dem som har handlet for med enn en viss sum foregående år, og en for dem som er medlem i Pirajafiskens venner.

   En kunde kan være med i flere rabattordninger, men kan bare bruke én rabattordning i forbindelse med hvert kjøp (hver salgsID).

   Til å håndtere dette, har databasen tabellen

   Rabatt(kundeID, rabattnavn, salgsID, prosent)

   der rabattnavn er navnet på en rabatt kunden har, salgsID er et kjøp/salg der kunden har benyttet denne rabatten og prosent er hvor stor rabatt denne kunden har for akkurat denne rabattypen.

   i. Hvilke FDer gjelder i tabellen?
   ii. Hvilke kandidatnøkler har Rabatt?
   iii. Dekomponer Rabatt tapsfritt til BCNF.

2.4.1

Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)

a) De som selger printere, men ikke PCer:

$\cancel{\pi_{maker}(\sigma_{type='printer'}(Product) - \sigma_{type='pc'}(Product))}$

$$\pi_{maker}(\sigma_{type='printer'}(Product)) - \pi_{maker}(\sigma_{type='pc'}(Product))$$

b) PCer med hastighet ≥ 2.50:

$$\pi_{model}(\sigma_{speed>=2.50}(PC))$$

c) De som selger laptoper med harddisk minst 120GB:

$$\pi_{maker}(\sigma_{hd>=120}(Product \bowtie Laptop))$$

Det er antakelig unødvendig med $\sigma_{type='laptop'}$, her, for Laptop skal bare matche de i Product med type='laptop'.

d) Modell og pris på produkter fra C:

$$\pi_{model,price}(\sigma_{maker='C'}(Product) \bowtie PC)$$
$$\cup$$
$$\pi_{model,price}(\sigma_{maker='C'}(Product) \bowtie Laptop)$$
$$\cup$$
$$\pi_{model,price}(\rho_{Product(maker, model, ptype)}(\sigma_{maker='C'}(Product)) \bowtie Printer)$$

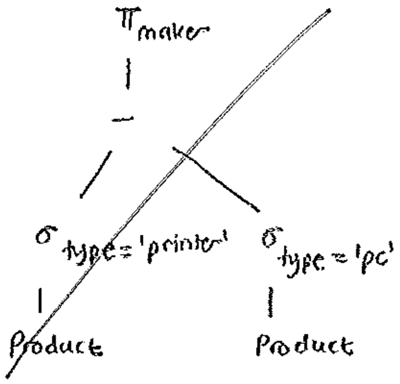
NB! Må renavne fordi type i Product ikke må matches mot type i Printer!

e) Alle sort-hvit laserskrivere:

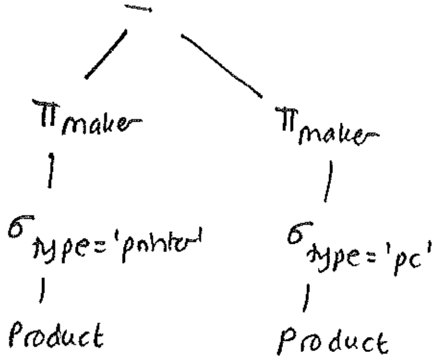
$$\pi_{model}(\sigma_{\text{not color and } type='laser'}(Printer))$$

2.4.2

a)

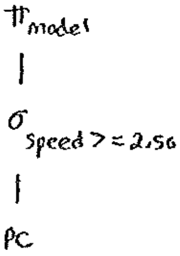


π_{maker}
$-$
$\sigma_{type='printer'}$
$\sigma_{type='pc'}$
Product
Product



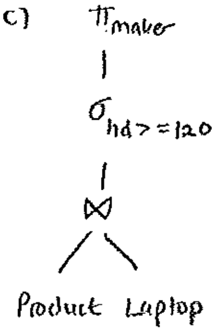
$-$
π_{maker}
π_{maker}
$\sigma_{type='printer'}$
$\sigma_{type='pc'}$
Product
Product

b)



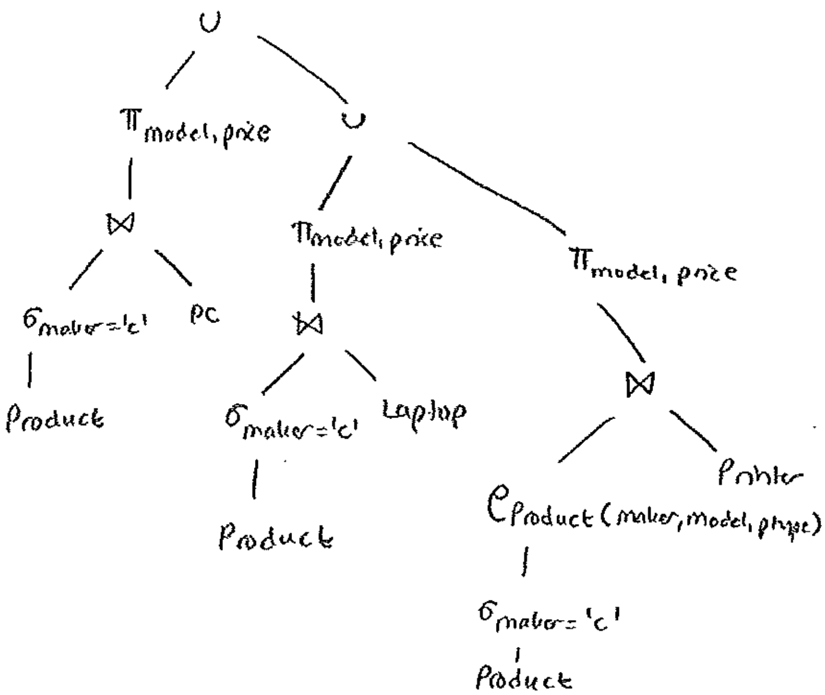
π_{model}
$\sigma_{speed>=2.56}$
PC

c)



π_{maker}
$\sigma_{hd>=120}$
$\bowtie$
Product
Laptop

d)



$\cup$
$\pi_{model, price}$
$\cup$
$\bowtie$
$\pi_{model, price}$
$\pi_{model, price}$
$\sigma_{maker='c'}$
PC
$\bowtie$
Product
$\sigma_{maker='c'}$
Laptop
$\bowtie$
Product
Printer
$\rho_{Product(maker, model, ptype)}$
$\sigma_{maker='c'}$
Product

2.5.1

a) PC med hastighet < 3.00 må være priset ≤ $800.

$$\sigma_{speed < 3.00 \text{ and } price > 800}(PC) = \emptyset$$

[Brytes av eksempeldataene, se modell 1001: speed = 2.66, pris $2114.]

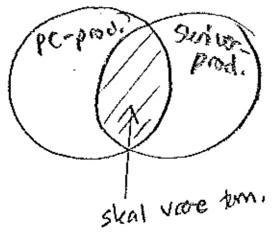
b) Laptop med skjerm mindre enn 15,4" må ha ≥ 120 Gb harddisk eller pris < $1000.

Det betyr at mengden av laptops med skjerm < 15,4", harddisk < 120Gb og pris ≥ $1000, skal være tom:

$$\sigma_{screen < 15.4 \text{ and } hd < 120 \text{ and } price >= 1000}(Laptop) = \emptyset$$

[Brytes av eksempeldataene, se modell 2004: screen = 13.3, hd = 60, price = 1150]

c) Ingen PC-produsenter får lage skrivere.



$$\pi_{maker}\left(\sigma_{type='pc'}(Product)\right) \cap \pi_{maker}\left(\sigma_{type='printer'}(Product)\right) = \emptyset$$

[Brytes av eksempeldataene; produsent D lager begge typer.]

2.5.1

d) Hvis laptop med større memory enn en PC, skal prisen også være høyere.

Idé: Finn par av laptoper og PCer hvor laptopen har størst memory. Finn par hvor laptopen har høyest pris. Førstnevnte mengde skal være inneholdt i sistnevnte.

$L(m, s, r, h, sc, p) :=$ Laptop; (renaming)

Answer(...) :=

$$\sigma_{r > ram}(L \times PC) \subseteq \sigma_{p > price}(L \times PC)$$

[Moteksempel i eksempeldataene:

Laptop 2002 har ram = 1024 og price = 949
PC 1002 har ram = 512 og price = 995]

2.5.1

e) En PC-produsent må også produsere en laptop med minst like stor speed.

laptop med større enn PCen

Kan lage par av PCer og laptoper med hensyn på hastighet for samme produsent, projiser på PC, skal da ha noe som omfatter samtlige PCer

Laptoper $\rightarrow$ $R(m,s) := \pi_{maker, speed}(Product \bowtie Laptop)$

PCer $\rightarrow$ $S(maker, speed) := \pi_{maker, speed}(Product \bowtie PC)$

PCer med en Laptop m/høyere speed fra samme produsent $\rightarrow$ $T(maker) := \pi_{maker}\left(S \underset{\substack{maker=m \\ \text{and} \\ speed<=s}}{\bowtie} R\right)$

Svar: $\underbrace{\pi_{maker}(S)}_{\text{Alle PCer}} \subseteq \underbrace{T}_{\text{Alle PCer hvor det er en Laptop med høyere hastighet}}$

2.5.2

Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)

a) Ingen klasse med 'bore' > 18":

$$\pi_{class}(\sigma_{bore>18}(Classes)) \subseteq \emptyset$$ ($\pi_{class}(\dots)$ er unødvendig)

b) Hvis mer enn 10 våpen, må 'bore' <= 15":

$$\pi_{class}(\sigma_{numGuns>10}(Classes)) \subseteq \pi_{class}(\sigma_{bore<=15}(Classes))$$

Alternativt:

$$\pi_{class}(\sigma_{numGuns>10 \text{ and } bore>15}(Classes)) \subseteq \emptyset$$

($\pi_{class}(\dots)$ er i begge tilfellene unødvendig)

c) Ingen klasse mer enn tre skip:

$$\sigma_{noships>3}(\gamma_{class,\, count(name) \rightarrow noships}(Ships \bowtie Classes)) \subseteq \emptyset$$

d) Intet land både battleship og battlecruisers:

$$\pi_{country}(\sigma_{type='bb'}(Classes)) \cap \pi_{country}(\sigma_{type='bc'}(Classes)) \subseteq \emptyset$$

e) Intet skip med mer enn 10 våpen i slag med skip med færre enn 9 våpen og som ble senket:

Skip med mer enn 10 våpen: $\rho_{S10(name1)}(\pi_{name}(Ships \bowtie \sigma_{numGuns>10}(Classes)))$

$\bowtie$ ← ② begrenser første skip til de med mer enn 10 våpen

slag: $\rho_{Out10(name1,battle,result1)}(Outcomes)$

$\bowtie$ ← ① kobler sammen to og to skip i samme slag hvor ett skip er senket

senkede skip i slag: $\rho_{Out9(name2,battle,result2)}(\sigma_{result='sunk'}(Outcomes))$

$\bowtie$ ← ③ begrenser andre skip til de med mindre enn 9 våpen

skip med mindre enn 9 våpen: $\rho_{S9(name2)}(\pi_{name}(Ships \bowtie \sigma_{numGuns<9}(Classes))) \subseteq \emptyset$ (ingen slike)

Løsningsforslag

NB Merk at "key" på engelsk / i lærebøker er det samme som kandidatnøkkel !

3.1.1

Oppgaven er løst med tanke på norske forhold

person(fdato, pnr, navn, husnr, gate, postnr, poststed, tlf)

FDer

Antakelser

fdato, pnr → navn

Navn er ikke entydig; flere personer kan ha identisk navn. Derfor har vi ikke navn på venstresiden i noen FD.

fdato, pnr → husnr, gate, postnr, poststed

Én person har bare én adresse

tlf → fdato, pnr

Hver telefon er registrert på én person. Én person kan ha flere telefonnumre.

postnr → poststed

Et postnummer har entydig poststed, men et poststed (f.eks. Oslo) kan ha mange postnumre

husnr, gate, poststed → postnr

Imidlertid kan flere poststeder omfatte identiske gatenavn, og en gate på et gitt poststed kan dekke flere postnumre (f.eks. Trondheimsveien i Oslo, som dekker postnumre 0560, 0565, ..., 0964 – ialt 11 forskjellige postnumre).

Det kan være flere med likt navn på samme adresse, f.eks. far og sønn med likelydende for- og etternavn.

tlf^+ = tlf, fdato, pnr, navn, husnr, gate, postnr, poststed

tlf er eneste kandidatnøkkel.

3.1.2 a)

Supernøkler er alle delmengder av A1.. An som inneholder A1.
Det er 2^{n-1} slike delmengder fordi hvert av attributtene A2.. An har to mulige "valg": Å være med i delmengden eller ikke ;
det gir $\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{n-1 \text{ stk}} = 2^{n-1}$.

3.2.1

R(A, B, C, D)

BC → D, D → A, A → B

a) Se på alle delmengder av ABCD og tillukninger av disse:

$A^+ = AB$ $B^+ = B$ $C^+ = C$ $D^+ = DAB$

Eneste nye er D → B (fra $D^+ = DAB$)

$AB^+ = AB$ – ingen nye

$AC^+ = ACBD$ – nye: AC → B, AC → D

$AD^+ = ADB$ – ny: AD → B

$BC^+ = BCDA$ – ny: BC → A

$BD^+ = BDA$ – ny: BD → A

$CD^+ = CDAB$ – nye: CD → A, CD → B

$ABC^+ = ABCD$ – ny: ABC → D

$ABD^+ = ABD$ – ingen nye

$ACD^+ = ACDB$ – ny: ACD → B

$BCD^+ = BCDA$ – ny: BCD → A

$ABCD^+ = ABCD$ – ingen nye

3.2.1

b) Ser på hvilke tillukninger i a) som gir ABCD.
De som utgjør kandidatnøkler, er

AC, BC, CD

(men ikke ABC, for AC $\subsetneq$ ABC, og ikke BCD, for CD $\subsetneq$ BCD, så verken ABC eller BCD er minimale supernøkler, og derfor er de ikke kandidatnøkler).

c) Supernøkler som ikke er kandidatnøkler, er alle de ikke nevnt i b), hvor lukningen er ABCD:

ABC, ACD, BCD, ABCD

$\boxed{3.2.2}$ i) $S(A,B,C,D)$ Løsningsforslag

$A \to B,\ A \to C,\ C \to D$

a) Ikketrivielle FDer (ett attributt i hs), i tillegg til de angitte

$A \to D$ $AB \to C$ $AC \to B$ $AD \to B$ $ABC \to D$

$AB \to D$ $AC \to D$ $AD \to C$ $ABD \to C$

$BC \to D$ $ACD \to B$

b) Kandidatnøkler:

$A^+ = ABCD$

Ingen andre bestemmer A, så A må være med i alle supernøkler. A er eneste minimale supernøkkel og dermed eneste kandidatnøkkel.

c) Supernøkler som ikke er kandidatnøkler:

Alle mengder som inneholder A, utenom A alene:

AB ABC $ABCD$

AC ABD

AD ACD

ii) $T(A,B,C,D)$ $A \to B,\ B \to C,\ C \to D,\ D \to A$

a) $A^+ = ABCD$ så $A \to C,\ A \to D$ og $AB \to C,\ AD \to C;\ AB \to D,\ AC \to D,\ ABD \to C,\ ABC \to D$

$B^+ = BCDA$ så $B \to D,\ B \to A$ osv.

$C^+ = CDAB$ så $C \to A,\ C \to B$

$D^+ = ABCD$ så $D \to B,\ D \to C$

b) A, B, C, D

c) alle 2-, 3- og 4-elements delmgd av $\{A,B,C,D\}$

iii) $U(A,B,C,D)$

$AB \to C,\ BC \to D,\ CD \to A,\ AD \to B$

a) $A^+ = A,\ B^+ = B,\ C^+ = C,\ D^+ = D$ [Regel: For å sjekke om $X \to Y$ er med, beregn X^+ og se om $Y \subseteq X^+$]

$AB^+ = ABCD$ så $\underline{AB \to D}$ (har alt $AB \to C$) (og dafor også $ABC \to D$)

$AC^+ = AC$

$AD^+ = ADBC$ (har alt $AD \to B$) så $\underline{AD \to C}$

$BC^+ = BCDA$ så $\underline{BC \to A}$ (og dafor også $BCD \to A$)

$BD^+ = BD$

$CD^+ = CDAB$ så $\underline{CD \to B}$ (og dafor også $ACD \to B$)

$ABC^+ = ABCD$ så $\underline{ABC \to D}$

$ABD^+ = ABCD$ så $\underline{ABD \to C}$

$BCD^+ = ABCD$ så $\underline{BCD \to A}$

$ACD^+ = ABCD$ så $\underline{ACD \to B}$

b) $AB,\ BC,\ CD,\ AD$

c) $ABC,\ ABD,\ BCD,\ ABCD,\ ACD$

~~3.5.6~~ 3.2.4

Vis: Hvis $X \subseteq Y$, så $X^+ \subseteq Y^+$.

Anta $X \subseteq Y$. Vi skal vise at ved hvert steg i tillukningsalgoritmen, kan vi bygge opp en $\tilde{Y}$ hvor $\tilde{X} \subseteq \tilde{Y}$, der $\tilde{X}$ og $\tilde{Y}$ er de (delvis) tillukkede mengdene som bygges opp underveis i algoritmen.

I utgangspunktet (basis) er $\tilde{X} = X$ og $\tilde{Y} = Y$, så siden $X \subseteq Y$, er $\tilde{X} \subseteq \tilde{Y}$.

Betrakt et steg i algoritmen, og anta at $\tilde{X} \subseteq \tilde{Y}$ (induksjonshypotese). Ett steg går ut på å finne en $W \to Z$ hvor W er med i den delvis tillukkede mengden. Så se på FD-ene om det er noen slik $W \to Z$ for enten $\tilde{X}$ eller $\tilde{Y}$. (I) Hvis det er en for $\tilde{Y}$, men ikke $\tilde{X}$, vil etter steget

$$\tilde{Y} \leftarrow \tilde{Y} \cup Z$$

Så $\tilde{Y}$ blir desto større, og derfor holder $\tilde{X} \subseteq \tilde{Y}$ også etterpå. (II) Hvis det er en $W \to Z$ hvor $W \subseteq \tilde{X}$, må $W \subseteq \tilde{Y}$ også siden $\tilde{X} \subseteq \tilde{Y}$. Men da vil etter steget

$$\tilde{Y} \leftarrow \tilde{Y} \cup Z$$
og $$\tilde{X} \leftarrow \tilde{X} \cup Z$$

og da holder fortsatt $\tilde{X} \subseteq \tilde{Y}$ etterpå.

Så når vi gjennomfører algoritmen, vil vi alltid bygge opp en $\tilde{Y}$ som er minst like stor som $\tilde{X}$, så når algoritmen er ferdig, ender vi opp med en $X^+ \subseteq Y^+$ også.

~~3.5.7~~ 3.2.5

Vis at $(X^+)^+ = X^+$.

Gitt $W = X^+$. Vi skal lukke W. Se etter en $Y \to Z$ hvor $Y \subseteq W$. Hvis det fins en slik hvor $Z \nsubseteq W$, er ikke X^+ lukket, så det kan ikke finnes ytterligere slike, altså kan vi ikke øke W ytterligere.

– for kriteriet på at X^+ er ferdig beregnet, var at det ikke var noen slike.

3.2.6

a) $AB \to C$

Vis at det ikke er slik at $A \to C$ eller $B \to C$

b) $A \to B$

Vis at det ikke er slik at $B \to A$

c) $AB \to C$, $A \to C$

Vis at det ikke er slik at $B \to C$

Metode 1: Tillukning

$A \to C$? $A^+ = A$, så svaret er nei.

$B \to C$? $B^+ = B$, så svaret er nei.

$B \to A$? $B^+ = B$, så svaret er nei

$B \to C$? $B^+ = B$, så svaret er nei

Metode 2: Chase

A	B	C
a	b	c
a	b_2	c_2

$\} A \to C$?

$AB \to C$ kan ikke benyttes til å endre noe. Siden vi har at de to radene er like i A, men forskjellige i C, har vi at $A \to C$ ikke gjelder.

A	B	C
a	b	c
a_2	b	c_2

$\} B \to C$?

Samme her: $AB \to C$ gir ingen endringer. Siden de to radene er like i B, men forskjellige i C, har vi at $B \to C$ ikke gjelder.

A	B
a	b
a_2	b

$\} B \to A$?

$A \to B$ gir ingen endringer. Siden de to radene er like i B, men forskjellige i A etter at chase-algoritmen er fullført, har vi at $B \to A$ ikke gjelder.

A	B	C
a	b	c
a_2	b	c_2

$\} B \to C$?

$AB \to C$ og $A \to C$ kan ikke benyttes til å gjøre noen endringer. (Så etter at chase-algoritmen har fullført, er instansen fortsatt som over.) Siden de to radene er like i B, men forskjellige i C, er det slik at $B \to C$ ikke gjelder.

Metode 3: Moteksempel

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_3

Lovlig instans ($AB \to C$ er oppfylt), men hverken $A \to C$ eller $B \to C$ er oppfylt.

A	B
a_1	b_1
a_2	b_1

$A \to B$ er oppfylt, men ikke $B \to A$

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

$AB \to C$ og $A \to C$ er oppfylt, men ikke $B \to C$.

3.2.9

R(A, B, C, D, E) + FDer, projisert på S(A, B, C) – hvilke FDer i S?

b) $BC \to DE$, $A \to E$, $D \to A$, $E \to B$

Finn alle ikketrivielle (minimale) avledede FDer som involverer A, B, C:

$A^+ = AEB$, så $A \to B$

$B^+ = B$

$C^+ = C$

$AB^+ = A^+ = AEB$

$AC^+ = ACEBD$, så $AC \to B$ – men har alt $A \to B$

$BC^+ = BCDEA$, så $BC \to A$

Så FDene som gjelder internt i S, er

$A \to B$, $BC \to A$ (så AC og BC er kandidatnøkler for S)

c) $C \to D$, $AD \to E$, $BC \to E$, $DE \to A$

$C^+ = CD$

$BC^+ = BCDEA$, så $BC \to A$

$AC^+ = ACDE$

$AB^+ = AB$

Så FDene som gjelder internt i S, er

$BC \to A$ (så BC er kandidatnøkkel for S)

d) $AB \to E$, $AC \to D$, $BC \to E$, $E \to A$, $D \to B$

$AB^+ = ABE$

$AC^+ = ACDBE$, så $AC \to B$

$BC^+ = BCEAD$, så $BC \to A$

Så FDene som gjelder internt i S, er

$AC \to B$, $BC \to A$ (så AC og BC er kandidatnøkler for S)

3.2.9

d)

$R(A,B,C,D,E)$

$AB \to E$, $AC \to D$, $BC \to E$, $E \to A$, $D \to B$

Projisere R på $S(A,B,C)$. FDer som holder i S:
Ta alle tillukninger av ikketomme, ekte delmengder av ABC:

$A^+ = A$

$B^+ = B$

$C^+ = C$

$AB^+ = ABE$

$AC^+ = ACDBE$ gir. $AC \to B$ som berører relasjon S

$BC^+ = BCEAD$ gir $BC \to A$ "

Så $AC \to B$ og $BC \to A$ holder for S.

$\boxed{3.3.1}$ a) R(A, B, C, D)

Løsningsforslag

Oppgavene er løst for BCNF og EKNF

$B \to A$, $C \to B$, $D \to C$, $A \to D$

i) $A^+ = ABCD$

$B^+ = ABCD$

$C^+ = ABCD$

$D^+ = ABCD$

Kandidatnøkler: A, B, C, D

Brudd på BCNF: ingen

(og dermed heller ingen brudd på EKNF.)

3.3.1 b)

$R(A,B,C,D)$

$BC \to D$, $D \to A$, $A \to B$

Kandidatnøkler: Må ha med C siden C ikke finns i noen høyreside

$C^+ = C$

$$\left.\begin{array}{l} AC^+ = ACBD \\ BC^+ = BCDA \\ CD^+ = CDAB \end{array}\right\} \Rightarrow AC, BC, CD \text{ er kandidatnøkler}$$

Ikke-trivielle FDer (med minimal venstreside):

$BC \to D$

$D \to A$

$A \to B$

$D \to B$ $(D^+ = DAB)$

$AC \to D$ $(AC^+ = ACBD)$

$BC \to A$ $(BC^+ = BCDA)$

$\Rightarrow$ AC og BC er elementære kand.nkl. (CD er ikke elementær)

$AC \to D$ er elementær

$BC \to A$ er elementær

$D \to A$, $D \to B$

Brudd på BCNF: (Tilstrekkelig å se på den opprinnelige mengden FDer)

$D \to A$

$A \to B$

BCNF dekomposisjon: $ABCD \to ABD$, CD ($D \to A$, $D^+ = ABD$); $ABD \to AB$, AD ($A \to B$, $A^+ = AB$)

Brudd på EKNF: Ingen.

3.3.1

c) $R(A,B,C,D)$ $\quad A \to B, A \to C$

Kandidatnøkler: Må ha med A, D siden de ikke fins i noen høyreside.

$AD^+ = ADBC \quad \Rightarrow \quad AD$ er kandidatnøkkel

Ikketrivielle FDer (med minimal venstreside):

$A \to B$
$A \to C$ $\quad \Rightarrow \quad AD$ er ikke elementær kand.nkl.

Brudd på BCNF:

$A \to B$
$A \to C$

BCNF dekomposisjon:

$$\begin{array}{ccc} & R(A,B,C,D) & \\ \swarrow & & \searrow \\ R1(A,B,C) & & R2(A,D) \end{array} \qquad A \to B, \; A^+ = ABC$$

Brudd på EKNF: (faktisk bryter de 3NF og 2NF også)

$A \to B$
$A \to C$

EKNF dekomposisjon:

1. Minimal overdekning: $A \to B, A \to C$ er minimal allerede
2. $R_A(A,B,C)$

   (samler $A \to B$ og $A \to C$ til $A \to BC$)
3. $R_0(D)$
4. Verken ABC eller D er supernøkler, utvid $R_0(A,D)$

Dekomposisjonen blir $R_A(A,B,C)$, $R_0(A,D)$
(så samme som BCNF-dekomposisjonen).

3.3.1 d) $R(A, B, C, D)$

$AB \to D$, $BD \to C$, $CD \to A$, $AC \to B$

Kandidatnøkler:

$$\left.\begin{aligned} AB^+ &= ABDC \\ BD^+ &= BDCA \\ CD^+ &= CDAB \\ AC^+ &= ACBD \end{aligned}\right\} \Rightarrow AB, BD, CD, AC \text{ er kandidatnøkler}$$

Ikke-trivielle FDer med minimal venstreside:

$$\left.\begin{aligned} AB &\to D \\ BD &\to C \\ CD &\to A \\ AC &\to B \end{aligned}\right\} \Rightarrow AB, BD, CD, AC \text{ er alle elementære kandidatnøkler}$$

$AB \to C$ ($AB^+ = ABCD$)

$BD \to A$ ($BD^+ = ABCD$)

$CD \to B$ ($CD^+ = ABCD$)

$AC \to D$ ($AC^+ = ABCD$)

Brudd på BCNF: (Ser bare på opprinnelige FDer)

Ingen

3.3.1

e) $R(A,B,C,D,E)$

$AB \to C,\ C \to E,\ E \to A,\ E \to D$

Kandidatnøkler: B må være med i alle slike fordi ikke i noen høyreside.

$$\left.\begin{array}{l} AB^+ = ABCED \\ BC^+ = BCEAD \\ BD^+ = BD \\ BE^+ = BEADC \end{array}\right\} \Rightarrow AB, BC, BE \text{ kandidatnøkler}$$

Ikketrivielle FDer med minimal venstreside:

$AB \to C$ $\Rightarrow$ AB elementær kand.nk.

$C \to E$ $\Rightarrow$ BC ikke elementær kand.nk.

$E \to A$

$E \to D$

$\left.\begin{array}{l} C \to A \\ C \to D \end{array}\right\} (C^+ = CEAD)$

$\left.\begin{array}{l} AB \to D \\ AB \to E \end{array}\right\} (AB^+ = ABCDE)$

$BE \to C$ $(BE^+ = ABCDE)$ $\Rightarrow$ BE elementær kand.nk.

Brudd på BCNF (bare opprinnelige FDer):

$C \to E$

$E \to A$

$E \to D$

BCNF dekomposisjon:

$ABCDE \to ACDE,\ CB$

CB: $C \to E$, $C^+ = ACDE$

$ACDE \to ADE,\ CE$

$E \to A$, $E^+ = ADE$

Brudd på EKNF:

$E \to D,\ C \to D$ (disse bryter også 2NF)

Dekomponere til EKNF:

1. Minimal overdekning:
   1. $AB \to C,\ C \to E,\ E \to A,\ E \to D$
   2. Ingen splitting nødvendig
   3. VS allerede minimale
   4. Ingen FD er overflødig
2. $R_{AB}(A,B,C),\ R_C(C,E),\ R_E(E,A,D)$
3. Alle attributter er med, $R_0 = \emptyset$
4. $R_{AB}(A,B,C)$ er en supernøkkel (AB er kandidatnøkkel!)

Dekomp. blir $R_{A,B}(A,B,C),\ R_C(C,E),\ R_E(E,A,D)$

3.3.1

f) $R(A,B,C,D,E)$

$AB \to C$, $DE \to C$, $B \to E$

Kandidatnøkler: A, B, D må være med fordi ikke i noen H.S.

$ABD^+ = ABDCE \Rightarrow ABD$ er (eneste) kandidatnøkkel

Ikketrivielle FDer: (minimal V.S.)

$AB \to C$ $\Rightarrow$ ABD er ikke elementær kandidatnøkkel
$DE \to C$
$B \to E$

Brudd på BCNF:

$AB \to C$
$DE \to C$
$B \to E$

BCNF dekomposisjon:

$R(A,B,C,D,E)$

$AB \to C$, $AB^+ = ABCE$

$R1(A,B,C,E)$ $R2(A,B,D)$

$B \to E$, $B^+ = BE$

$R3(B,E)$ $R4(A,B,C)$

Brudd på ECNF:

$AB \to C$ (også brudd på 2NF)
$DE \to C$ (også brudd på 3NF)
$B \to E$ (også brudd på 2NF)

ECNF dekomposisjon:

1. Minimal overdekning: 1. $AB \to C$, $DE \to C$, $B \to E$
   2. Ingen splitting nødvendig
   3. VS er allerede minimale
   4. Ingen FDer overflødige
2. $R_{AB}(A,B,C)$, $R_{DE}(D,E,C)$, $R_B(B,E)$
3. Alle attributter er med, $R_0 = \emptyset$
4. Ingen supernøkler i noen av R_{AB}, R_{DE}, R_B: $R_0(A,B,D)$

Dekomposisjonen blir altså $R_{AB}(A,B,C)$, $R_{DE}(D,E,C)$, $R_B(B,E)$, $R_0(A,B,D)$.

$\boxed{3.3.3}$

R(A, B, C, D)

F = { A → B, A → C }

Kandidatnøkkel: AD

A → B, A → C bryter BCNF og 3NF

Dekomposisjon ihht. A → B:

R_1(A, C, D) R_2(A, B)

F_1 = {A → C} F_2 = {A → B}

Kand. nkl: AD Kand. nkl: A

A → C bryter BCNF, dekomponer til

R_{11}(A, D) R_{12}(A, C)

F_{11} = { } F_{12} = {A → C}

Resultat:

R_{11}(A, D) F_{11} = { }

R_{12}(A, C) F_{12} = {A → C}

R_2(A, B) F_2 = {A → B}

(FD-bevarende dekomposisjon)

Dekomposisjon ihht. A → BC:

R_1'(A, D) R_2'(A, B, C)

F_1' = { } F_2' = { A → BC }

(FD-bevarende)

Forskjellen er at vi ender opp med tre og to tabeller, henholdsvis. To tabeller sparer noe lagringsplass.

3.4.1

$R(A, B, C, D, E)$

$\mathcal{D} = \{ABC, BCD, ACE\}$

a) $BC \to D$, $AC \to E$

Er $\mathcal{D}$ tapsfri?

	A	B	C	D	E
ABC	a	b	c	d_1	e_1
BCD	a_2	b	c	d	e_2
ACE	a	b_3	c	d_3	e

$BC \to D$ og $AC \to E$ gir

	A	B	C	D	E
ABC	a	b	c	$\cancel{d_1}\ d$	$\cancel{e_1}\ e$
BCD	a_2	b	c	d	e_2
ACE	a	b_3	c	d_3	e

← uten indekser, derfor er $\mathcal{D}$ tapsfri

3.4.1

$R(A, B, C, D, E)$

$\mathcal{D} = \{ABC, BCD, ACE\}$

c) $B \to E,\ CE \to D,\ D \to E$

Tapsfri $\mathcal{D}$?

Chase:

	A	B	C	D	E
ABC	a	b	c	$\cancel{d_1}\ d$	e_1
BCD	a_2	b	c	d	$\cancel{e_2}\ e_1$
ACE	a	b_3	c	d_3	e

Ingen rad er uten indekser, så $\mathcal{D}$ er ikke tapsfri.

Eksempel på instans som gir falske tupler: Kan alltid ta utgangspunkt i resultatet av chasen; dette vil alltid være på en form som gir falske tupler!

I chasen representerer bokstaver uten subscripts konstanter, mens bokstaver med subscripts er variable. Bytt ut variablene med (nye) konstanter, da får vi f.eks. følgende instans:

A	B	C	D	E
a	b	c	d	f
g	b	c	d	f
a	h	c	k	e

(Vi har satt inn f for begge e_1-ene, g for a_2, h for b_3 og k for d_3.)

projisér

A	B	C
a	b	c
g	b	c
a	h	c

B	C	D
b	c	d
h	c	k

A	C	E
a	c	f
g	c	f
a	c	e

⋈ (join)

A	B	C	D
a	b	c	d
g	b	c	d
a	h	c	k

⋈ (join)

A	B	C	D	E
a	b	c	d	f
a	b	c	d	e
g	b	c	d	f
a	h	c	k	f
a	h	c	k	e

Falske tupler!

3.5.2

R(ABCDE) $AB \to C$, $C \to B$, $A \to D$

D = {ABC, AD, ABE}

	A	B	C	D	E
ABC	a	b	c	$\cancel{d_1}$ d	e_1
AD	a	$\cancel{b_2}$ b	$\cancel{c_2}$ c	d	e_2
ABE	a	b	$\cancel{c_3}$ c	$\cancel{d_3}$ d	e

← uten indekser, altså tapsfri

3.6.1

Løsningsforslag

$R(A, B, C)$

$B \twoheadrightarrow C$

(a_1, b, c_1) og (a_2, b, c_2) med i R betyr at også
(a_1, b, c_2) og (a_2, b, c_1) må være med i R, ved å
bytte om C-verdiene siden B-verdiene er
like og $B \twoheadrightarrow C$.

Tilsvarende:

(a_1, b, c_1) og (a_3, b, c_3) med i R gir
(a_1, b, c_3) og (a_3, b, c_1) med i R

(a_2, b, c_2) og (a_3, b, c_3) med i R gir
(a_2, b, c_3) og (a_3, b, c_2) med i R.

3.6.3

R(n, s, b, cn, cs, cb, as, am)

a) FDer:

s → n, b
cs → cn, cb
} social security number bestemmer navn og fødselsdato, dette gjelder både forelder og barn

as → am } bilnummer bestemmer merke

MVDer:

Det er jo slik at hvis XYZ er alle attributtene og $X \twoheadrightarrow Y$, så $X \twoheadrightarrow Z$. At $X \twoheadrightarrow Y$ uttrykker at opplysningene i Y og Z er urelaterte. Derfor kan det være en angrepsvinkel å prøve å identifisere tre grupper av attributter: De som gjelder personen som person (X – n, s, b), de som gjelder personens barn (Y – cn, cs, cb) og de som gjelder personens biler (Z – as, am). Opplysningene om personens barn og de om bilene, er urelaterte (de relaterer seg til personen, men ikke til hverandre). I såfall er det snakk om MVDene

n, s, b $\twoheadrightarrow$ cn, cs, cb

n, s, b $\twoheadrightarrow$ as, am

(det er nok å angi én av dem; den andre følger automatisk).

(En annen måte, er å teste ut noen instanser og se hvordan de bør se ut for å være lovlige.)

3.6.3

b) Sjekk av brudd på 4NF:

Kandidatnøkkel er (s, cs, as): Disse forekommer ikke i noen høyreside i FDene, og

$(s, cs, as)^+ = s, cs, as, n, b, cn, cb, am$

$n, s, b \twoheadrightarrow cn, cs, cb$: bryter 4NF fordi venstresiden ikke er en supernøkkel.

(det samme gjelder $n, s, b \twoheadrightarrow as, am$)

Dekomponer i henhold til MVDer som bryter 4NF:

R(n, s, b, cn, cs, cb, as, am)

$n, s, b \twoheadrightarrow cn, cs, cb$

R1 (n, s, b, cn, cs, cb)

FDer: $s \rightarrow n, b$

$cs \rightarrow cn, cb$

R2 (n, s, b, as, am)

$s \rightarrow n, b$

$as \rightarrow am$

Sjekk av brudd på normalformer opp til BCNF:

Kandidatnøkkel R1:
s, cs forekommer ikke i noen høyreside og må være med.

$(s, cs)^+ = s, n, b, cs, cn, cb$

så eneste kand.nk. er (s, cs).

$s \rightarrow n, b$ bryter BCNF
bryter 3NF
(og EKNF)
bryter 2NF

$cs \rightarrow cn, cb$ bryter BCNF
bryter 3NF
(og EKNF)
bryter 2NF

Kandidatnøkkel R2:
(s, as) (samme argumentasjon som R1)

$s \rightarrow n, b$ bryter BCNF, 3NF og 2NF

$as \rightarrow am$ —"—

3.6.3

b) (forts.)

Dekomponerer R1 og R2 i henhold til FDer som bryter 2NF:

R1(n, s, b, cn, cs, cb)

$s \rightarrow n, b$ bryter 2NF
($s^+ = s, n, b$)

S1(s, n, b)   S2(s, cn, cs, cb)

$cs \rightarrow cn, cb$
bryter 2NF
($cs^+ = cs, cn, cb$)

T1(cs, cn, cb)   T2(s, cs)

R2(n, s, b, as, an)

$s \rightarrow n, b$ bryter 2NF

~~S3(s, n, b)~~   S4(s, as, an)

denne har
vi alt, den
er lik S1

$as \rightarrow an$
bryter 2NF
($as^+ = as, an$)

T3(as, an)   T4(s, as)

Totalt: Dekomponert til

S1(s, n, b)

T1(cs, cn, cb)

T2(s, cs)

T3(as, an)

T4(s, as)

disse opplysningene bør jo egentlig slås sammen i én relasjon, men det klarer ikke dekomposisjonen å avsløre – men vi som databasemennesker ser at informasjonen i de to semantisk er den samme: det er opplysninger om social security number, navn og fødselsdato for personer.

3.6.3

Alternativ til a)

Alternativ MVD under a):

$$s \twoheadrightarrow cn, cs, cb$$

Dvs. opplysninger om barn og deres ssn, navn og fødselsdato relaterer seg til personen (ved vedkommendes ssn), men er urelatert til øvrige opplysninger.

Faktisk er det slik at hvis $s \to n, b$ $cs \to cn, cb$ $as \to am$ $n, s, b \twoheadrightarrow cn, cs, cb$ så holder $s \twoheadrightarrow cn, cs, cb$. Vi kan vise dette ved en variant av Chasealgoritmen:

s	n	b	cs	cn	cb	as	am
s	n_1	b_1	cs	cn	cb	as_1	am_1
s	n	b	cs_1	cn_1	cb_1	as	am

holder $s \twoheadrightarrow cn, cs, cb$? Sett opp to linjer der den ene er uten indekser på s, cn, cs, cb og den andre på s og resten. Målet er å få en rad uten indekser.

Bruk FDene: (Får bare brukt $s \to n, b$)

s	n	b	cs	cn	cb	as	am
s	$\cancel{n_1}$, n	$\cancel{b_1}$, b	cs	cn	cb	as_1	am_1
s	n	b	cs_1	cn_1	cb_1	as	am

Legg så på to tupler hvor de to ovenstående tuplene er "krysset" i henhold til MVDen $n, s, b \twoheadrightarrow cn, cs, cb$:

s	n	b	cs	cn	cb	as	am
s	n	b	cs_1	cn_1	cb_1	as_1	am_1

← uten indekser, så påstanden $s \twoheadrightarrow cn, cs, cb$ holder.

(Omvendt er det også slik at hvis $s \to n, b$ $cs \to cn, cb$ $as \to am$ $s \twoheadrightarrow cn, cs, cb$, så holder $n, s, b \twoheadrightarrow cn, cs, cb$. Det vises på tilsvarende måte.)

3.6.3

Alternativ til b)

Med MVDen $s \twoheadrightarrow cn, cs, cb$ (og FDene $s \rightarrow n, b$ $cs \rightarrow cn, cb$ $as \rightarrow an$) blir dekomposisjonen slik: (Fortsatt må man finne kandidatnøkler og bredd på normalformene først)

R(n, s, b, cn, cs, cb, as, an)

$s \twoheadrightarrow cn, cs, cb$

R1'(s, cn, cs, cb) R2'(n, s, b, as, an)

$cs \rightarrow cn, cb$ $s \rightarrow n, b$

S1'(cs, cn, cb) S2'(s, cs) S3'(s, n, b) S4'(s, as, an)

$as \rightarrow an$

T1'(as, an) T2'(s, as)

Totalt:

S1'(cs, cn, cb)

S2'(s, cs)

S3'(s, n, b)

T1'(as, an)

T2'(s, as)

3.x.1

$F = \{A \to C, AC \to D, E \to AD, E \to H\}$

$G = \{A \to CD, E \to AH\}$

Ekvivalens hvis FDene i den ene følger fra den andre og omvendt.

F gir $A \to CD$?

$$\begin{array}{cccccc} A & B & C & D & E & H \\ \hline A & B & C & D & E & H \\ A & b_2 & \cancel{c_2}^{C} & \cancel{d_2}^{D} & e_2 & h_2 \end{array} \Bigg\} \; A \to CD?$$

Bruker FDene i F ($A \to C, AC \to D$), får at andre rad er uten indekser i CD. Så F gir $A \to CD$.
Alternativt: Lukk A i hht. F:

$A^+ = ACD$

Så $A \to CD$ holder i F.

F gir $E \to AH$?

$$\begin{array}{cccccc} A & B & C & D & E & H \\ \hline A & B & C & D & E & H \\ \cancel{a_2}^{A} & b_2 & c_2 & \cancel{d_2}^{D} & E & \cancel{h_2}^{H} \end{array} \Bigg\} \; E \to AH?$$

Ja. ($E \to AD, E \to H$)
Alternativt:

$E^+ = EADH$ i F

Så G følger av F.

G gir $A \to C$?

$$\begin{array}{cccccc} A & B & C & D & E & H \\ \hline A & B & C & D & E & H \\ A & b_2 & \cancel{c_2}^{C} & \cancel{d_2}^{D} & e_2 & h_2 \end{array} \Bigg\} \; A \to C?$$

Ja. ($A \to CD$)
Alternativt: $A^+ = ACD$ i G.

G gir $AC \to D$?

$$\begin{array}{cccccc} A & B & C & D & E & H \\ \hline A & B & C & D & E & H \\ A & b_2 & C & \cancel{d_2}^{D} & e_2 & h_2 \end{array} \Bigg\} \; AC \to D?$$

Ja. ($A \to CD$)
Alternativt: $AC^+ = ACD$ i G.

G gir $E \to AD$?

$$\begin{array}{cccccc} A & B & C & D & E & H \\ \hline A & B & C & D & E & H \\ \cancel{a_2}^{A} & b_2 & \cancel{c_2}^{C} & \cancel{d_2}^{D} & E & \cancel{h_2}^{H} \end{array} \Bigg\} \; E \to AD?$$

Ja ($E \to AH, A \to CD$)
Alternativt: $E^+ = EAHCD$ i G.

G gir $E \to H$?

$$\begin{array}{cccccc} A & B & C & D & E & H \\ \hline A & B & C & D & E & H \\ \cancel{a_2}^{A} & b_2 & c_2 & d_2 & E & \cancel{h_2}^{H} \end{array} \Bigg\} \; E \to H?$$

Ja ($E \to AH$)
Alternativt: $E^+ = EAHCD$ i G.

Så alle FDer i G følger av FDene i F, og alle FDene i F følger fra FDene i G.
Altså er F og G ekvivalente.

3.x.2

$AB \to CD,\ B \to D$

A må være med i alle kandidatnøkler fordi den ikke er i noen høyreside.
C og D kan ikke være med i noen kandidatnøkkel fordi de bare er i høyresider.

$A^+ = A$

$AB^+ = ABCD$ — AB er kandidatnøkkel (den eneste)

Normalformer:

$AB \to C$: AB er kandidatnøkkel (og supernøkkel), så BCNF.
$AB \to D$ —ıı—
$B \to D$: B er ikke en supernøkkel. Så ikke BCNF.
D er ikke et nøkkelattributt. Så ikke 3NF.
B er ekte inneholdt i kandidatnøkkelen AB. Så ikke 2NF
Så 1NF, men bryter 2NF.

Relasjonen er på 1NF, men bryter 2NF.

Dekomposisjon i henhold til FDen som bryter 2NF:

R(A,B,C,D)
↙ ↘
S(B,D) T(A,B,C)

S(B,D)	T(A,B,C)
$B \to D$	$AB \to C$
På BCNF	På BCNF

Kan for sikkerhets skyld sjekke at tapsfri:

	A	B	C	D
BD	a_1	B	c_1 C	D
ABC	A	B	C	d_2 D

Tapsfri fordi første rad er uten indekserte forekomster.

$A \twoheadrightarrow D$ innføres. Holder $A \to D$?

A	B	C	D	
A	B	C	D	$A \to D$?
A	b_2	c_2	d_2 D	
A	b_2	c_2	D	$A \twoheadrightarrow D$: kryss rad 1 og 2 på D og BC
A	B	C	d_2 D	

Ja, for D er uten indekser i rad 2.

~~Oppgave 2~~ 3.x.3

$F = \{ DE \to AD,\ AB \to BC,\ BC \to CD,\ CD \to ABE \}$

1. Minimal overdekning:

a Splitter høyresidene (og fjerner trivielle FDer)

$DE \to A$

trivell $\cancel{DE \to D}$

–"– $\cancel{AB \to B}$

$AB \to C$

–"– $\cancel{BC \to C}$

$BC \to D$

$CD \to A$

$CD \to B$

$CD \to E$

b Minimale venstresider:

Alle venstresider er minimale: siden det ikke er noen FD med bare ett attributt i venstresiden, F.eks. for $DE \to A$:

$D^+ = D,\ E^+ = E,$

så verken $D \to A$ eller $E \to A$ holder.

c Fjern overflødige FDer:

$DE^+ = DE$ i $F - \{DE \to A\}$

$AB^+ = AB$ i $F - \{AB \to C\}$

$BC^+ = BC$ i $F - \{BC \to D\}$

$CD^+ = CDBEA$ i $F - \{CD \to A\}$ $\quad$ $CD \to A$ er overflødig

$CD^+ = CDAE$ i $F - \{CD \to B\}$

$CD^+ = CDAB$ i $F - \{CD \to E\}$

Resultat:

$DE \to A$

$AB \to C$

$BC \to D$

$CD \to B$

$CD \to E$

2.

Elementær FD: $X \to A$ der

- A er ett attributt
- $X \to A$ er ikketriviell (dvs. $A \notin X$)
- X minimal (dvs. har ikke $Y \to A$ for noen ekte delmengde Y av X)

Alle FDene i den minimale overdekningen er elementære:

$DE \to A$

$AB \to C$

$BC \to D$

$CD \to B$

$CD \to E$

($A^+ = A$, $B^+ = B$, $C^+ = C$, $D^+ = D$, $E^+ = E$, så ingen venstreside kan gjøres mindre og fortsatt utgjøre en FD.)

3. Kandidatnøkler: Enhver kandidatnøkkel må bestå av minst to attributter siden alle venstresider har to attributter.

$DE^+ = DEA$

$AB^+ = ABCDE$ Kandidatnøkkel: AB

$BC^+ = BCDEA$ Kandidatnøkkel: BC

$CD^+ = CDBEA$ Kandidatnøkkel: CD

$AC^+ = AC$, $AD^+ = AD$, $AE^+ = AE$, $BD^+ = BD$, $BE^+ = BE$,
$CE^+ = CE$, $ACE^+ = ACE$

Så

$AB \to C$, $BC \to D$, $CD \to B$, $CD \to E$

oppfyller kravet til BCNF. For

$DE \to A$

er venstresiden ikke en supernøkkel, så kravet til BCNF er brutt. Høyreside er et nøkkelattributt (nøkkelattributtene er ABCD), så kravet til 3NF er oppfylt. Nå er $AB \to C$ en elementær FD, så AB er en elementær kandidatnøkkel. Følgelig oppfyller $DE \to A$ kravet til EKNF.

Totalt er R derfor på EKNF.

4. Tapsfri dekomposisjon til BCNF:

$R(A, B, C, D, E)$

DE → A bryter BCNF

$R1(A, D, E)$ $\quad$ $R2(B, C, D, E)$

5. Det fins ingen FD-bevarende dekomposisjon til BCNF.
Hvis vi bruker algoritmen for FD-bevarende dekomposisjon (til EKNF eller bedre), får vi følgende relasjoner:

$R_{AB} = \{\underline{A, B}, C\}$ fra $AB \rightarrow C$

$R_{BC} = \{\underline{B, C}, D\}$ fra $BC \rightarrow D$

$R_{CD} = \{B, \underline{C, D}, E\}$ fra $CD \rightarrow B$, $CD \rightarrow E$. R_{CD} inneholder også FDen $BC \rightarrow D$.

$R_{DE} = \{A, \underline{D, E}\}$ fra $DE \rightarrow A$

$BC \rightarrow D$ i R_{CD} bryter BCNF. Så denne er på EKNF, men ikke BCNF. Hvis det hadde vært mulig å dekomponere tapsfritt og FD-bevarende til BCNF, ville denne algoritmen som vi her har brukt, gitt en slik dekomposisjon.

~~3.?.6 Ekstraoppgave~~ 3.x.4

Vis: Hvis $X \twoheadrightarrow Y$, så $X \twoheadrightarrow W$ hvor W er resten av attributtene.

Anta $X \twoheadrightarrow Y$. Anta en instans med to tupler t_1 og t_2 som er like på X; da fins u_1 og u_2 som er lik t_1 og t_2 på X, og hvor u_1 er lik t_1 på Y og t_2 på W, og u_2 er lik t_2 på Y og t_1 på W:

	X	$Y-X$	W
t_1	x_1	y_1	w_1
t_2	x_1	y_2	w_2
u_1	x_1	y_1	w_2
u_2	x_1	y_2	w_1

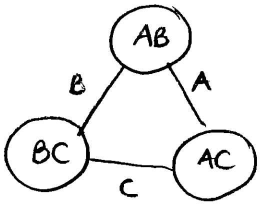
Hvis $X \twoheadrightarrow W$ skal holde, må det bety at det fins to tupler v_1 og v_2 hvor v_1 er lik t_1 på W og lik t_2 på $Y-X$, og hvor v_2 er lik t_2 på W og lik t_1 på $Y-X$:

v_1	x_1	y_2	w_1
v_2	x_1	y_1	w_2

Men to slike tupler har vi, hvis vi velger $v_1 = u_2$ og $v_2 = u_1$! Så kravet til $X \twoheadrightarrow W$ holder, og følgelig holder påstanden.

3.x.5

(a) $R(ABC)$, $A \to C$, $B \to C$, $\{AB, BC, AC\}$:



Sykel, så vi kan ikke utelukke at dekomposisjonen kan gi støyinstanser

Støyinstanseksemplene er konstruert ved å tenke ut hvordan tupler kan "kansellere hverandre" under naturlig join.

Eksempel på støyinstanser:

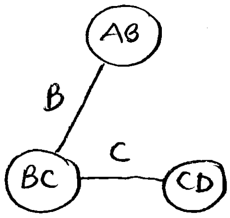
A	B
a_1	b_1
a_2	b_2

B	C
b_1	c_1
b_2	c_2

A	C
a_1	c_2
a_2	c_1

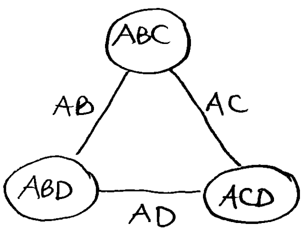
Hvis vi joiner de tre, får vi den tomme mengden.

(b) $S(ABCD)$, $A \to B$, $B \to C$, $C \to D$, $\{AB, BC, CD\}$



Ikke sykel, så dekomposisjonen vil aldri gi støyinstanser.

(c) $T(ABCD)$, $AB \to D$, $AC \to D$, $\{ABC, ABD, ACD\}$



Sykel, så vi kan ikke utelukke at dekomposisjonen kan gi støyinstanser

Eksempel på støyinstanser:

A	B	C
a	b_1	c_1
a	b_2	c_2

A	B	D
a	b_1	d_1
a	b_2	d_2

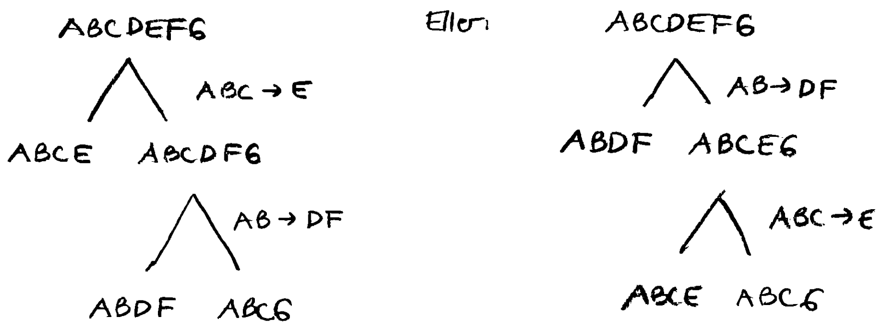
A	C	D
a	c_1	d_2
a	c_2	d_1

Hvis vi joiner de tre, får vi den tomme mengden.

3.x.6

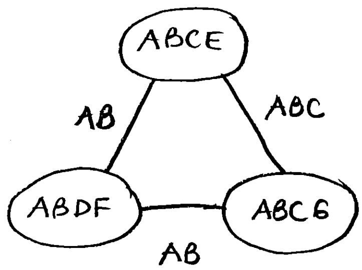
(a) R(A, B, C, D, E, F, G) ABC→E, AB→DF

(i) Kandidatnøkkel: ABCG



Begge gir dekomposisjonen {ABCE, ABDF, ABCG}.

(ii)

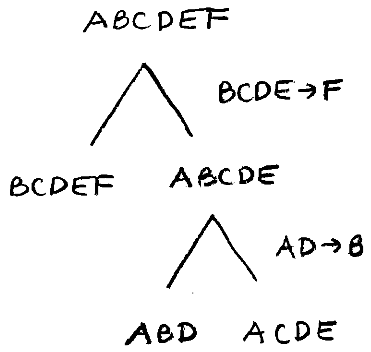


Snittgrafen har en sykel, men siden ABDF ∩ ABCG = AB ⊆ ABCE, kan dekomposisjonen likevel <u>ikke</u> ha støyinstanser.

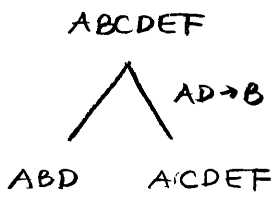
3.x6

(b) S(A, B, C, D, E, F)   BCDE → F, AD → B

(i) Kandidatnøkkel: ACDE.



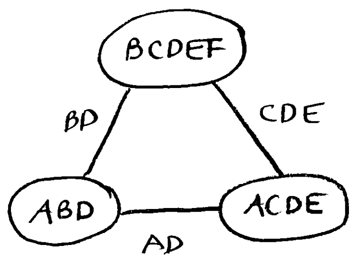
Eller



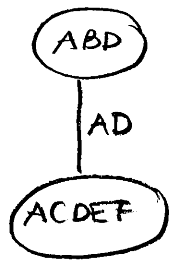
Dekomposisjon: {ABD, ACDEF}

Dekomposisjon: {BCDEF, ABD, ACDE}

(ii)



Eller



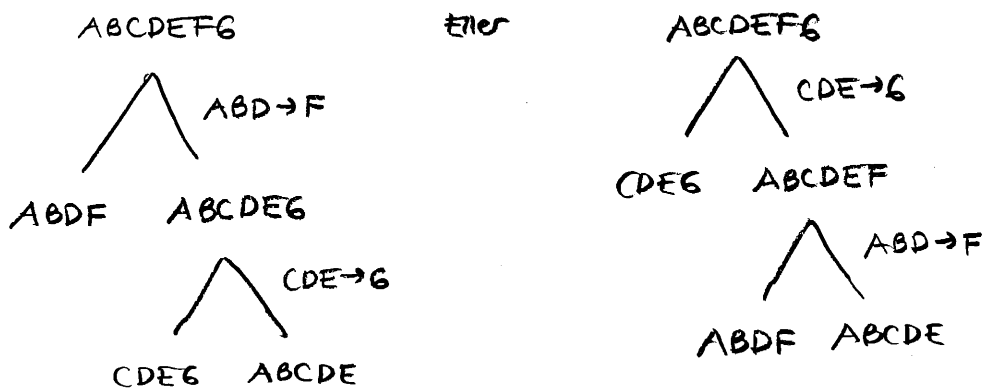
Syklisk, kan ha støyinstanser.

Ikke syklisk, kan ikke ha støyinstanser

3.x.6

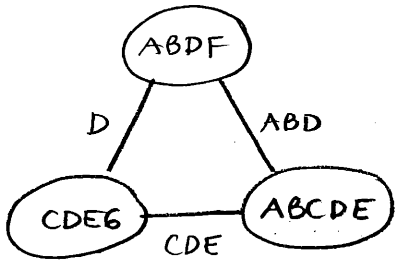
(c) $T(A,B,C,D,E,F,G)$ $ABD \to F$, $CDE \to G$

(i) Kandidatnøkkel: ABCDE



Begge gir dekomposisjonen {ABDF, CDEG, ABCDE}

(ii)



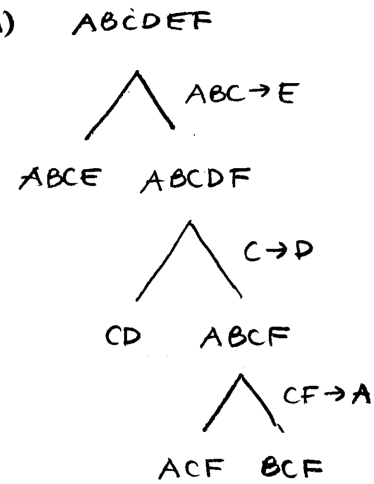
Sykliske, men siden $ABDF \cap CDEG = D \subseteq ABCDE$, kan dekomposisjonen likevel ikke ha støyinstanser.

3.x 6

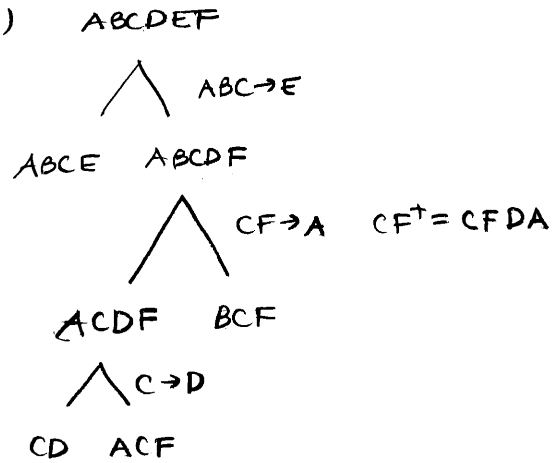
(d) U(A, B, C, D, E, F)   ABC → E, C → D, CF → A

(i) Kandidatnøkkel: BCF. Det er fem mulige måter å dekomponere på, (i-A), (i-B), (i-C), (i-D), (i-E).

(i-A)

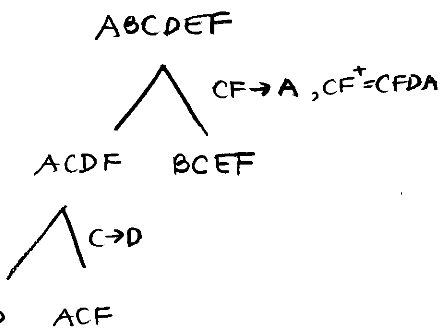


(i-B)

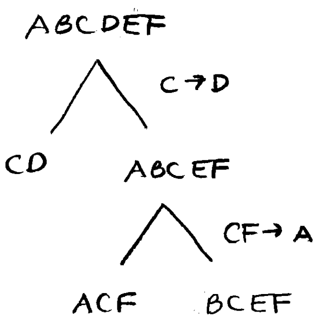


Dekomposisjon: { ABCE, CD, ACF, BCF }

(i-C)

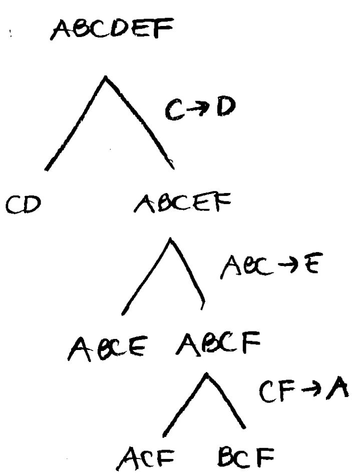


(i-D)



Dekomposisjon: { CD, ACF, BCEF }

(i-E)

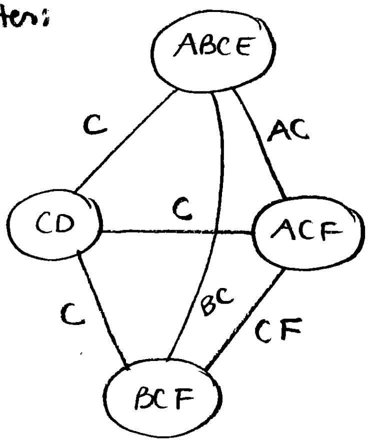


Dekomposisjon: { ABCE, CD, ACF, BCF }
(Samme som (i-A) og (i-B).)

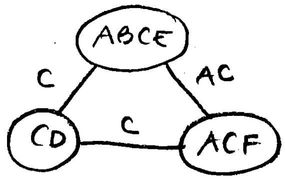
3.x.6

(d) (ii)

Enten:

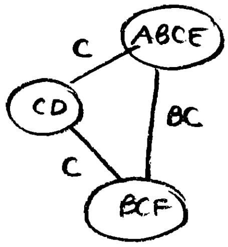


Det er <u>fire</u> sykler:



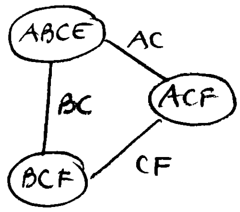
Denne gir ikke støyinstansproblemer fordi:

$C = CD \cap ABCE \subseteq ACF$

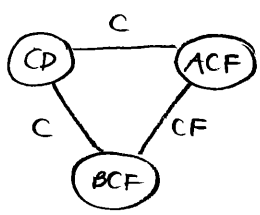


Denne gir ikke støyinstansproblemer fordi:

$C = CD \cap ABCE \subseteq BCF$



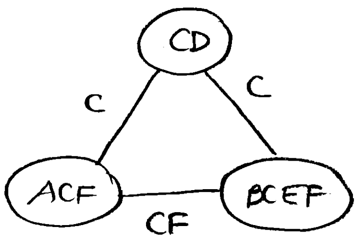
<u>Denne kan forårsake støyinstanser!</u>



Denne gir ikke støyinstansproblemer fordi:

$C = CD \cap BCF \subseteq ACF$

Eller:



Sykliske, men siden $C = CD \cap ACF \subseteq BCEF$,
kan dekomposisjonen likevel <u>ikke</u> ha støyinstanser.

5.2.1

R:

A	B
1	2
3	4
1	2
3	5
4	5

S:

B	C
1	2
3	5
3	6
4	5
1	3
4	5

a) $\pi_{A^2, B^2, A+B}(R)$:

A^2	B^2	$A+B$
1	4	3
9	16	7
1	4	3
9	25	8
16	25	9

b) $\pi_{B-1,\ C+1}(S)$:

$B-1$	$C+1$
0 ~~2~~	3
2 ~~4~~	6
2 ~~4~~	7
3 ~~5~~	6
0 ~~2~~	4
3 ~~5~~	6

c) $\tau_{A,B}(R)$: [(1,2), (1,2), (3,4), (3,5), (4,5)]

d) $\tau_{C,B}(S)$: [(1,2), (1,3), (3,5), (4,5), (4,5), (3,6)]

e) $\delta(R)$:

A	B
1	2
3	4
3	5
4	5

Løsningsforslag

5.2.1 f) $\delta(S)$:

B	C
1	2
3	5
3	6
4	5
1	3

g) $\gamma_{A, AVG(B)}(R)$:

A	AVG(B)
1	2
3	4,5
4	5

h) $\gamma_{B, SUM(C)}(S)$:

B	SUM(C)
1	5
3	11
4	10

5.2.1

i) $\gamma_A(R)$: Gruppering leverer en <u>mengde</u> som svar.

A
1
3
4

j) $\gamma_{A,max(c)}(R \bowtie S)$:

$R \bowtie S$

A	B	C
3	4	5
3	4	5

$\gamma_{A,max(c)}(R \bowtie S)$

A	max(c)
3	5

k) $R \overset{\circ}{\bowtie}_R S$

A	B	C
3	4	5
3	4	5
⊥	1	2
⊥	3	5
⊥	3	6
⊥	1	3

l) $R \overset{\circ}{\bowtie}_L S$

A	B	C
3	4	5
3	4	5
1	2	⊥
1	2	⊥
3	5	⊥
4	5	⊥

m) $R \overset{\circ}{\bowtie} S$

A	B	C
3	4	5
3	4	5
1	2	⊥
1	2	⊥
3	5	⊥
4	5	⊥
⊥	1	2
⊥	3	5
⊥	3	6
⊥	1	3

n) $R \overset{\circ}{\bowtie}_{R.B<S.B} S$

A	R.B	S.B	C
1	2	3	5
1	2	3	6
1	2	4	5
1	2	4	5
1	2	3	5
1	2	3	6
1	2	4	5
1	2	4	5
3	4	⊥	⊥
3	5	⊥	⊥
4	5	⊥	⊥
⊥	⊥	1	2
⊥	⊥	1	3

5.2.2

a) $\pi_L(\pi_L(R)) = \pi_L(R)$?

For vanlig projeksjon stemmer dette; å projisere to ganger på samme attributtliste gir samme resultat som én projeksjon.

For utvidet projeksjon stemmer ikke dette lenger:

$$\pi_{A+1 \to A}(\pi_{A+1 \to A}(R)) \neq \pi_{A+1 \to A}(R)$$

Eks. R:

A
1
2
3

$\pi_{A+1 \to A}(R)$:

A
2
3
4

$\pi_{A+1 \to A}(\pi_{A+1 \to A}(R))$:

A
3
4
5

b) $\sigma_C(\sigma_C(R)) = \sigma_C(R)$?

Ja. Om jeg velger ut tupler etter et kriterium, så vil jeg ikke få noe endret utvalg om jeg bruker samme kriterium en gang til.

c) $\gamma_L(\gamma_L(R)) = \gamma_L(R)$?

Nei. Problemet er at aggregeringene gir annet resultat ved andre anvendelse av grupperingsoperatorene.

$$\gamma_{A,\,count(B) \to B}(\gamma_{A,\,count(B) \to B}(R)) \neq \gamma_{A,\,count(B) \to B}(R)$$

Eks. R:

A	B
1	1
1	2
2	3
2	4

$\gamma_{A,\,count(B) \to B}(R)$:

A	B
1	2
2	2

$\gamma_{A,\,count(B) \to B}(\gamma_{A,\,count(B) \to B}(R))$:

A	B
1	1
2	1

5.2.2

d) $\tau_L(\tau_L(R)) = \tau_L(R)$?

Ja, Om man sorterer to ganger etter samme kriteser, skal man ikke få noe annet enn om man sorterer én gang.

e) $\delta(\delta(R)) = \delta(R)$?

Ja. Om man har eliminert duplikater én gang, vil man ikke fjerne noe mer ved å gjøre prosessen en gang til.

6.1.1

select X Y
from ...
} Y er et alias for X

select X as Y
from ...
} Y er et alias for X

select X, Y
from ...
} X og Y er to attributter

6.1.5

a, b integer, muligens null i noen tupler.

a) $a < 50$ or $a >= 50$: Alle tupler hvor $a \neq$ null

b) $a = 0$ or $b = 10$: Alle tupler hvor $a = 0$ og b enten er null eller har en verdi
&
alle tupler hvor $b = 10$ og a enten er null eller har en verdi

c) $a = 20$ and $b = 10$: Hvis a og b er de eneste attributtene, er tuppelet $(a, b) = (20, 10)$ eneste mulige kandidat.

d) $a = b$: Hvis en av a eller b har verdien null, er resultatet av beregningen av $a = b$ lik unknown. Denne verdien er $\neq$ true.
Eneste tupler som tilfredsstiller denne, er derfor de hvor a og b er ikke-null og har samme verdi.

e) $a > b$: Tilsvarende. For at verdien skal kunne bli sann (true), må både a og b være ulik null. Dessuten må verdien i a være større enn den i b.

6.1.6

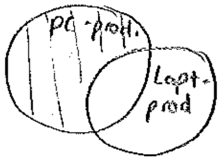
```
select *
from Movies
where length <= 120 or length > 120;
```

samme som :

```
select *
from Movies
where length is not null;
```

6.2.2

a) Produsenter av PCer, men ikke Laptops:




```
(select maker
 from Product
 where type = 'pc')
except
(select maker
 from Product
 where type = 'laptop');
```

← må ha mengdeoperator her (altså except, ikke except all) eller ha select distinct kombinert med except all

maker
C
D

b) Produsent, hastighet for laptops med hd >= 100 GB:

```
select distinct p.maker, l.speed
from Product p, Laptop l
where p.model = l.model and l.hd >= 100;
```

maker	speed
E	2,00
A	2,16
B	1,83
F	1,60
G	2,00

6.2.2

c) Modell, pris for produkter fra C:

```
(select q.model, q.price
 from Product p, PC q
 where p.model = q.model and p.maker = 'C')
 union all
 (select q.model, q.price
 from Product p, Laptop q
 where p.model = q.model and p.maker = 'C')
 union all
 (select q.model, q.price
 from Product p, Printer q
 where p.model = q.model and p.maker = 'C');
```

union all gir en mengde som svar fordi hver av selectene gir en mengde ut (og ikke en bag). Det skyldes at model er primærnøkkel i hver av relasjonene PC, Laptop og Printer.

model	price
1007	510

6.2.2

d) Par av PCer med samme RAM og hd.

```
select p.model, q.model
from PC p, PC q
where p.ram = q.ram and p.hd = q.hd and p.model < q.model;
```

p.model	q.model
1001	1004
1001	1009
1002	1005
1003	1013
1004	1009

e) Prosessorhastigheter som forekommer i to eller flere PCer:

```
select speed
from PC
group by speed
having count(model) >= 2;
```

speed
2.80
3.20
2,20

6.2.2

f) Produsenter av to forskjellige PCer og/eller Laptoper med hastigheter minst 2.0:

```
select distinct p.maker
from ((select model from PC where speed >= 2.0)
      union
      (select model from Laptop where speed >= 2.0)) as m,
      Product p
where p.model = m.model
group by p.maker
having count(p.model) >= 2;
```

maker
A
B
D
E

6.2.3

a) Navn, displacement, antall våpen for skip i North Cape:

```
Select s.name, c.displacement, c.numGuns
From Outcomes o, Ships s, Classes c
Where o.battle = 'North Cape' and o.ship = s.name and s.class = c.class;
```

b) Skip tyngre enn 37000 tonn:

```
Select s.name
from Ships s, Classes c
where s.class = c.class and c.displacement >= 37000;
```

c) Alle skip:

```
(Select name from ships)
union
(select ship as name from Outcomes);
```

6.2.4

$\pi_L(\sigma_C(R_1 \times R_2 \times .. \times R_k))$

hvor L attributtliste, C betingelse, $R_1 .. R_k$ kan gjenta en relasjon flere ganger, (I såfall renavnet på passende måte). Uttrykt ved SQL:

select L
from $R_1, R_2, \ldots, R_k$
where C;

6.3.1 [Subqueries, minst to ekte forskjellige fremgangsmåter:]

a) Alle som lager laptop med hastighet minst 2.0:

```
select distinct p.maker
from Product p
where p.model in (select L.model
                  from Laptop L
                  where L.speed >= 2.0);
```

```
select distinct p.maker
from Product p
where exists (select *
              from Laptop L
              where L.model = p.model and L.speed >= 2.0);
```

b) Skriverne med høyest pris:

```
select p.model
from Printer p
where p.price >= all (select q.price
                      from Printer q);
```

```
select p.model
from Printer p
where p.price = (select max(q.price)
                 from Printer q);
```

6.3.1

c) Laptops med lavere hastighet enn den raskeste PCen:

```
select L.model
from Laptop L
where L.speed < ( select max(p.speed)
                  from PC p );
```

```
select L.model
from Laptop L
where L.speed < any (select p.speed
                     from PC p);
```

d) Modellene med lavest pris:

```
select u.model
from ((select model, price from PC)
      union all
      (select model, price from Laptop)
      union all
      (select model, price from Printer)) as u
where u.price >= all ((select price from PC)
                      union all
                      (select price from Laptop)
                      union all
                      (select price from Printer));
```

6.3.1

d) (2. variant) (Ikke særlig elegant, men....)

```
(select model
 from PC
 where price <= all ( (select min(price) from PC)
                      union all
                      (select min(price) from Laptop)
                      union all
                      (select min(price) from Printer)))
union all
(select model
 from Laptop
 where price <= all ( (select min(price) from PC)
                      union all
                      (select min(price) from Laptop)
                      union all
                      (select min(price) from Printer)))
union all
(select model
 from Printer
 where price <= all ((select min(price) from PC)
                      union all
                      (select min(price) from Laptop)
                      union all
                      (select min(price) from Printer)))
```

6.3.1

e) Fargeskriveren med høyest pris (produsenten av ..) :

```
       distinct
select ˅ p.maker
from Product p
where p.model in ( select q.model
                   from Printer q
                   where q.price >= (select max(r.price)
                                     from Printer r
                                     where r.color));
```

```
       distinct
select ˅ p.maker
from Product p, Printer q
where p.model = q.model and q.color and
      not exists ( select *
                   from Printer r
                   where r.color and r.price > q.price );
```

6.3.1

f) Produsent av PCen med raskeste prosessor blant de med størst ram:

```
Select p.maker
from  (select s.model
       from (select r.model, r.speed, r.ram
             from PC r
             where r.ram = (select max(ram) from PC)) as s
       where
             r.speed =
               (select max(speed)
                from PC t
                where t.ram = (select max(ram) from PC))) as q,
       Product as p
where p.model = q.model;
```

6.3.8

Informasjon om alle produkter, inklusive produsent hvis tilgjengelig.

```
(select model, speed, ram, hd, rd, null as screen, null as color, null as type, price, maker
from PC natural left outer join Product)
union all
(select model, speed, ram, hd, null as rd, screen, null as color, null as type, price, maker
from Laptop natural left outer join Product)
union all
(select model, null as speed, null as ram, null as hd, null as rd, null as screen, color, type, price, maker
from Printer natural left outer join Product));
```

Ideen er at alt skal med, så for hver klasse av produkter hvor en betegnelse ikke gir mening, opprettes et attributt med verdien null for dette.

6.3.9

```
select *
from Ships natural join Classes;
```

eller

```
select s.name, s.class, s.launched, c.type, c.country, c.numGuns, c.bore, c.displacem.
from Ships s, Classes c
where s.class = c.class;
```

6.4.6

a) Gjennomsnittlig hd for PCer:

```
select avg(hd)
from PC;
```

b) Gjennomsnittlig pris Laptops med speed ≥ 3.0:

```
select avg(price)
from Laptop
where speed >= 3.0;
```

c) Gjennomsnittlig pris PCer fra A:

```
select avg(q.price)
from Product p, PC q
where p.maker = 'A' and p.model = q.model;
```

6.4.7

a) Antall bc-klasser:

```
select count(*)
from Classes
where type = 'bc';
```

b) Gjennomsnittlig bore for bb:

```
select sum(bore) / count(bore) as avgbore
from Classes
where type = 'bb';
```

(Denne går bare bra hvis count(bore) ≠ 0.)

c) Som b, men vekter med antall skip i klassen:

```
select sum(weight) / sum(noships) as avgweightedbore
from ( select c.class, count(s.name) as noships, count(s.name) * max(bore) as weight
       from Ships s, Classes c
       where c.type = 'bb' and s.class = c.class
       group by c.class ) as d;
```

NB! Det er ikke lov å ha beregning inni en aggregeringsfunksjon!
Ellers kunne vi gjort det slik (men det går altså ikke):

```
select sum(noships * bore) / sum(noships)
from (select c.class, count(s.name) as noships, max(bore) as bore
      from Ships s, Classes c
      where c.type = 'bb' and s.class = c.class
      group by c.class) as d;
```

Mange DBMSer tillater beregning inni aggregering, derfor tillater vi det også.

NB2! Det er ikke lov å referere 'bore' uten å aggregere i en group by, selv om alle i hver gruppe har samme bore.
Et triks er å bruke max(bore) eller min(bore) for å få ut verdien.

6.4.7

d) Antall skip pr. klasse senket i slag:

```
select s.class, count(s.name) as nosunk
from Ships s, Outcomes o
where o.result = 'sunk' and o.ship = s.name
group by s.class;
```

e) Siste lanseringsår pr. klasse:

```
select class, max(launched) as lastlaunch
from Ships
group by class;
```

f) For hver klasse med minst to skip, antall senkede skip:

```
select s.class, count(s.name) as nosunk
from ( select class
       from Ships
       group by class
       having count(name) >=2 ) as c,
       Ships s, Outcomes o
where  o.result = 'sunk' and o.ship = s.name and s.class = c.class
group by s.class;
```

(Burde hatt med 0 for de klassene med minst to skip og ingen senkede; det blir litt mer komplisert.)

6.4.7

g) Hvert våpen har en granat som veier (i pund) ca. halvparten av diameteren (i tommer) opphøyd i tredje.

Gjennomsnittsvekt på granatene til hvert lands skip:

```
select country, sum(weight)/sum(numGuns) as avgShellWeight
from ( select s.name, s.class, c.country, c.numguns, c.numGuns*((c.bore)**3)/2 as weight
       from Classes c, Ships s
       where s.class = c.class ) as t
group by country;
```

Her er det, litt tilsvarende c), gjort slik at antall våpen spiller inn.
Så hvis skip s_i har n_i våpen, hver med en granat av vekt w_i,
skal vi beregne for hvert land

$$\left(\sum_i n_i \cdot w_i\right) / \left(\sum_i n_i\right)$$

i for vedkommende land

Derfor starter vi med (i indre select) å beregne $n_i \times w_i$ for hvert skip.
Deretter grupperer vi på land og gjør den avsluttende beregningen.

~~Oppgave 3~~ 6.x.1

Person (pid , navn)

Selskap (dato, meny, vert)

Gjest (pid, dato)

Gjester som har vært med i samtlige selskaper arrangert i tidsrommet 2000-2009 av Aschehoug:

Forslag 1)

```
select p.navn
from Person p
where not exists ( select s.dato
                   from Selskap s
                   where vert = 'Aschehoug' and
                         dato >= date '2000-01-01' and
                         dato <= date '2009-12-31' )
                 except all
                 (select g.dato
                  from Gjest g
                  where g.pid = p.pid));
```

det er ingen av de aktuelle selskapene som p ikke har vært i

Alle de aktuelle selskapene

gjør bra fordi select s.dato from.. blir en mengde (dato er primærnøkkel)

Alle selskaper som p har vært i

Forslag 2)

```
select p.navn
from Person p
where (select count(s.dato)
       from Selskap s, Gjest g
       where g.pid = p.pid and g.dato = s.dato and
             s.dato >= date '2000-01-01' and
             s.dato <= date '2009-12-31' and
             s.vert = 'Aschehoug')
      =
      (select count(t.dato)
       from Selskap t
       where t.dato >= date '2000-01-01' and
             t.dato <= date '2009-12-31' and
             t.vert = 'Aschehoug');
```

Antall av de aktuelle selskapene som p har vært i

Antall aktuelle selskaper.

trenger ikke count(distinct ..) fordi grunnlags-relasjonene i dette tilfellet blir mengder.

7.2.2

a) type må være 'pc', 'laptop' eller 'printer':

```
create table Product (
    maker char(15),
    model integer,
    type varchar(7) check (type = 'pc' or type = 'laptop' or type = 'printer'),
    primary key (maker, model)
);
```

b) speed for laptop må være minst 2,2:

```
create table Laptop (
    model integer primary key,
    speed real check (speed >= 2.2),
    ram integer,
    hd integer,
    screen real,
    price integer
);
```

c) Printere er kun laser og ink-jet:

```
create table Printer (
    model integer primary key,
    color boolean,
    type varchar(7) check (type = 'laser' or type = 'ink-jet')
);
```

7.2.4

a)
```
create table pc (
    model  integer primary key,
    speed  real,
    ram    integer,
    hd     integer,
    price  integer,
    check (speed >= 2.0 or price <= 600)
);
```

(dvs. speed < 2.0 ⇒ price <= 600)

b)
```
create table laptop (
    model  integer primary key,
    speed  real,
    ram    integer,
    hd     integer,
    screen real,
    price  integer,
    check (screen >= 15 or hd >= 40 or price < 1000)
);
```

(dvs. screen < 15 ⇒
(hd >= 40 v. price < 1000))

7.4.2

a)
```
create assertion max3ships
  check ( 3 >= all ( select count(*)
                     from Classes c, Ships s
                     where s.class = c.class
                     group by s.class ));
```

Hvis vi skal ta alvorlig at group by-attributtene alltid skal være med i select, må vi gjøre det feks. slik:

```
create assertion max3ships
  check ( 3 >= all ( select nships
                     from ( select s.class, count(*) as nships
                            from Classes c, Ships s
                            where s.class = c.class
                            group by s.class ) as t ));
```

b)
```
create assertion shipwithclassname
  check ( not exists ( ( select class as name from Classes )
                       except
                       ( select name from Ship where name = class )));
```

(Kunne vært løst uten check assertion ved en enkel fremmednøkkel)

10.2.1

Flights(airline, frm, to, departs, arrives)

a) Reiser der det er minst én times opphold mellom flybytter:
For hvilke par av byer (x,y) er det mulig å komme fra x til y med en eller flere flights?

```
with recursive Reaches(frm, to, arrives) as (
    select frm, to, arrives from Flights
  union
    select r.frm, f.to, f.arrives
    from Reaches r, Flights f
    where r.to = f.frm and f.departs >= r.arrives + 100
)
select frm, to from Reaches;
```

Har her antatt i beregningen "f.departs >= r.arrives + 100" at tidspunktene rett og slett er i form av heltall, dvs. kl. 9:00 er representert ved tallet 900 osv. Da er det enkelt å angi "en time senere" ved å simpelthen legge til 100.

Rekursjonen terminerer fordi det er et endelig antall byer og alle tidspunkter er ≤ 2400.

10.2.3

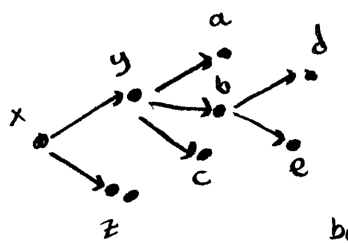
SequelOf(movie, sequel)

En sequel til en film er en oppfølgerfilm. SequelOf inneholder en oversikt over direkte oppfølgere. Oppfølgerfilmer kan igjen ha sine oppfølgerfilmer, så SequelOf inneholder i realiteten data om en mengde trær.

a) FollowOn(movie, sequel) skal inneholde direkte eller indirekte oppfølgerfilmer. Vi skriver FollowOn i form av et rekursivt view:

```
create recursive view FollowOn(movie, sequel) as (
    select movie, sequel from SequelOf          } <ikke-rekursiv term>
    union all
    select f.movie, s.sequel                    }
    from FollowOn f, SequelOf s                 } <rekursiv term>
    where f.sequel = s.movie                    }
);
```

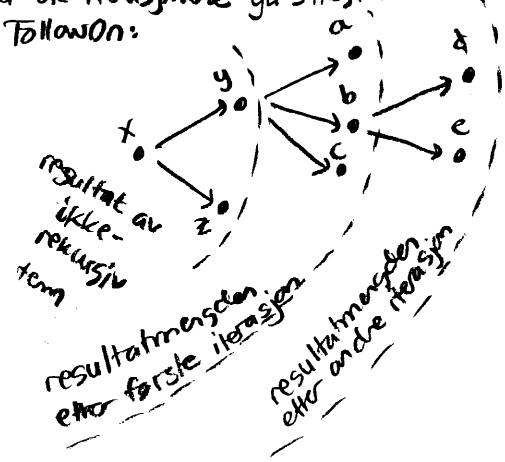
Det burde gå bra med union all her fordi rekursjonen systematisk beveger seg utover trærne, så vi får stadig med nye biter av et tre i resultatmengden: Hvis SequelOf inneholder et tre



SequelOf

x	y
x	z
y	a
y	b
y	c
⋮	

Så vil iterasjonene gå slik, hvis vi bare fokuserer på tupler på formen (x, ..) i FollowOn:



FollowOn

x	y	(beregning av)
x	z	etter ikke-rek. term
x	a	
x	b	
x	c	etter første iterasjon
x	d	
x	e	etter andre iterasjon

10.2.3

c) Finn (x,y) der y er en oppfølger til x, men ikke en direkte oppfølger.

```
select * from FollowOn
except all
select * from SequelOf;
```

Det står egentlig i oppgaven at man skal skrive en rekursiv spørring som gjør dette, men FollowOn inneholder jo det vi trenger.

Alternativt kunne det kanskje vært lurt å ta med i FollowOn hvor mange "kanter" det er mellom hvert par av filmer

```
create recursive view FollowOnD(movie, sequel, dist) as (
    select movie, sequel, 1 from SequelOf
    union all
    select f.movie, s.sequel, f.dist+1
    from FollowOnD f, SequelOf s
    where f.sequel = s.movie
);

select movie, sequel
from FollowOnD
where dist >1;
```

d) De der y hverken er en direkte oppfølger eller en direkte oppfølger av en direkte oppfølger.

```
select movie, sequel
from FollowOnD
where dist >2;
```

10.2.3

e) (x,y) der y er en oppfølger av x, og y har høyst én oppfølger.

```
select f.movie, f.sequel
from FollowOn f
where f.sequel in (select g.movie
                   from FollowOn g
                   where g.movie = f.sequel
                   group by g.movie
                   having count(g.sequel) = 1)
      or
      f.sequel not in (select s.movie
                       from SequelOf s);
```

f) De filmene som har to eller flere oppfølgere:

```
select movie
from FollowOn
group by movie
having count(sequel) >= 2;
```

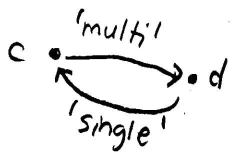
10.2.4

Rel(class, rclass, mult)

beskriver en rettet graf der kantene er merket 'multi' eller 'single'.

Rel beskriver relasjoner mellom klasser og om relasjonene er en-til-mange, mange-til-mange eller en-til-en:

Hvis c,d er en-til-mange (da kan hvert element i klassen c være knyttet til mange elementer i d, men et element i d kan bare være knyttet til ett element i c), så inneholder Rel to tupler



class	rclass	mult
:	:	
c	d	'multi'
d	c	'single'
:	:	

a) De parene (c,d) der det finnes en sti fra c til d:

Vi må ha med stier som går fra en klasse tilbake til klassen selv, men vi er ikke interessert i stier som inneholder løkker utover dette. Her er det lurt å ha med innholdet i stiene som del av attributtene i den rekursive spørringen.

```
create recursive view Path(class, rclass, path, pathlabels) as (
    select class, rclass, array[class, rclass], array[mult] from Rel
    union all
    select p.class, r.rclass,
           p.path || r.rclass,
           p.pathlabels || r.mult
    from Path p, Rel r
    where p.class <> p.rclass and p.rclass = r.class
);
```

(Det viste seg at jeg ikke trengte attributtet path noe sted i spørringene, jeg trengte bare pathlabels.)

```
select distinct class, rclass from Path;
```

10.2.4

b) De parene (c,d) der det finnes en sti fra c til d der alle kantene er merket 'multi':

```
select distinct class, rclass
from Path
where 'multi' = all(pathlabels);
```

c) De parene (c,d) der det finnes en sti fra c til d der minst én kant er merket 'single':

```
select distinct class, rclass
from Path
where 'single' = any (pathlabels);
```

d) Det er minst én sti fra c til d, men ingen av stiene har bare 'multi' på kantene:

```
select distinct p.class, p.rclass
from Path p
where (p.class, p.rclass) not in
      (select class, rclass
       from Path q
       where 'multi' = all (q.pathlabels);
```

10.2.4

e) Det er en sti der lablene er annenhver 'single' og 'multi':

```
with recursive Annenhver(class, rclass, lastLbl) as (
    select class, rclass, mult from Rel
    union all
    select a.class, r.rclass, r.mult
    from Annenhver a, Rel r
    where a.class <> a.rclass and a.rclass = r.class
          and a.lastLbl <> r.mult
)
select distinct class, rclass from Annenhver;
```

f) Det er stier fra c til d og tilbake til c der alle kantene er merket 'multi':

```
select distinct p1.class, p1.rclass
from Path p1, Path p2
where p1.rclass = p2.class and p2.rclass = p1.class
      and p1.class <> p1.rclass
      and 'multi' = all (p1.pathlabels)
      and 'multi' = all (p2.pathlabels);
```

Løsningsforslag

10.4.3 PostgreSQL:

10.4.3 løst med PostgreSQL

```
create type classtp as (
    class varchar(20),
    classtype char(2),
    country varchar(15),
    numGuns integer,
    bore integer,
    displacement integer
);
```

← lager som type for å se hvordan det funker. Får da ikke noen primary key (det brukes bare for tabeller). Burde kunnet skrive „ check (classtype = 'bb' or ...) her, men det protesterer systemet på. Så det kommer her, i stedet.

```
create table ship (
    name varchar(20) primary key,
    class classtp check (classtype = 'bb' or classtype = 'bc'),
    launched integer
);
```

← bruker altså en record som verdi her, så bryter 1NF

```
create table battle (
    name varchar(30) primary key,
    duration interval
);
```

← interval kan brukes for å angi fra-til

```
create table outcome (
    ship varchar(20) references ship(name),
    battle varchar(30) references battle(name),
    result varchar(7)
        check (result = 'sunk' or result = 'ok' or result = 'damaged'),
    primary key (ship, battle)
);
```

← antar at et skip kan være med i flere slag

Eks: Ship

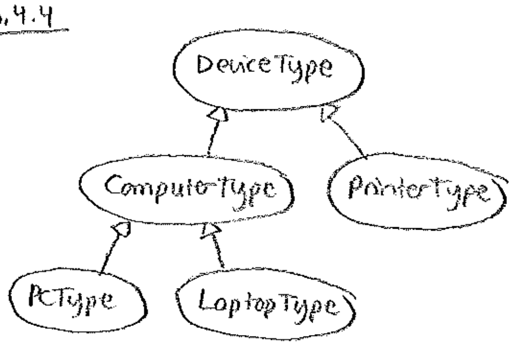
name	class	launched
Haruna	(Kongo, bc, Japan, 8, 14, 32000)	1915
Hiei	(Kongo, bc, Japan, 8, 14, 32000)	1914
Kongo	(Kongo, bc, Japan, 8, $\underline{\underline{15}}$, 32000)	1913

feil – men modellen hindrer ikke dette(*)

(*) Ulempen med modellen er at det er ingen kontroll på at klassebetingelsen brukes "riktig": Info/record med klasseopplysninger legges inn direkte under hvert skip, så vi får duplisering av data og fare for å skrive feil i noen av recordene (kan f.eks. ha info om class Bismarck hvor f.eks. 'numGuns' eller 'bore' på tvers av skipstupler har forskjellige verdier selv om det gjelder samme klasse og altså burde hatt identiske verdier). På den annen side: Oppgavene under 10.5.3 blir enkle; man har direkte tilgang til klassedata for hvert skip uten joinoperasjoner.

10.4.4 løst med SQL:1999

10.4.4




```
create type ModelNameType as varchar(10);
create type SpeedType as real;
create type RamType as integer;
create type HdType as integer;
create type ScreenType as real;
create type PriceType as real;
create type ColorType as varchar(5);
create type PrinterType as varchar(10);

create type DeviceType as (
    model  ModelNameType,
    price  PriceType
)
ref is system generated;

create type ComputerType under DeviceType as (
    speed  SpeedType,
    ram    RamType,
    hd     HdType
);

create type PCType under ComputerType as (
);

create type LaptopType under ComputerType as (
    screen  ScreenType
);

create type PrinterType under DeviceType as (
    color  ColorType,
    type   PrinterType
);

create table Device of DeviceType (
    ref is deviceid system generated,
    primary key (model)
);
```

For at skal kunne tilordne objektidentifikator til relasjoner som er typet med DeviceType

subtype

Tabellen kan da ha objekter av alle subtyper av DeviceType, det er mer elegant enn den opprinnelige strukturen med tre tabeller (en for hver av pc, laptop, printer)

10.4.4 (forts.)

```
create type MakerNameType as varchar(20);
create

create type ProductType as (
    maker MakerNameType,
    model ref (DeviceType)
);
```

Trenger ikke lenger et attributt "type" som brukes til å angjøre hvilken tabell opplysninger om produktet fins i, det kan vi gjøre ved å se på hvilken subtype objektene i tabellen Device har.

```
create table Product of ProductType (
    primary key (maker, model),
    model with options scope Device
);
```

Restrikerer til å referere DeviceType-objekter i Device-relasjonen

Burde kanskje valgt et annet navn for ikke å blande sammen med model i DeviceType...

10.4.4 løst med PostgreSQL

10.4.4 PostgreSQL: Har ikke reftype. Reformulert oppgave:

"Redesign produktdatabasen fra oppgave 2.4.1. Bruk typedeklarasjoner der dette passer. Utnytt muligheten for arv til å strukturere databasen bedre."

```
create table product (                 ← (lager automatisk en tilhørende
    maker varchar(20),                     type product)
    model integer,
    type varchar(10),
    primary key (maker, model)
);

create table PC (                      ← (lager automatisk typen PC)
    speed numeric,
    ram integer,
    hd integer,
    price integer,
    primary key (maker, model)         ← i PostgreSQL 8.4 arves ikke nøkler
) inherits (product);                  ← arver attributtene fra product

create table Laptop (
    speed numeric,
    ram integer,
    hd integer,
    screen numeric,
    price integer,
    primary key (maker, model)
) inherits (product);

create table Printer (
    color boolean,
    printertype varchar(10),           ← kan ikke gjenbruke 'type' som
    price integer,                         attributtnavn (product har et attributt type)
    primary key (maker, model)
) inherits (product);
```

NB Bedre forslag neste side!

10.4.4 (forts).

Bedre:

```
create table product (
    maker varchar(20),
    model integer,
    type varchar(10),          ← kan vurdere å droppe dette attributtet
    price integer,             ← alle har en pris, så ha den heller her
    primary key (maker, model)
);

create table computer (        ← felles for PC og Laptop
    speed numeric,
    ram integer,
    hd integer,
    primary key (maker, model)
) inherits (product);

create table PC (
    primary key (maker, model)     ← intet ekstra i forhold
) inherits (computer);               til computer
                                     (ingen nye attributter)
                                     Logisk likevel ryddigere/ryddigere
                                     å ha en egen tabell
                                     og ikke simpelthen lagre
                                     alle PCer i computer-tabellen

create table Laptop (
    screen numeric,
    primary key (maker, model)
) inherits (computer);

create table Printer (
    color boolean,
    printertype varchar(10),
    primary key (maker, model)
) inherits (product);
```

Antakelig skal ingen tupler legges inn i table product, og heller ikke i table computer; disse er å anse som 'abstrakte' i SQL:1999-terminologi, men jeg tror ikke PostgreSQL har noen mulighet for å definere "abstrakte tabeller".

10.5.2

10.5.2 løst med SQL:1999

a) Produsenter av PCer med hd > 80 GB:

```
select distinct p.maker
from Product p
where deref(p.model) is of (PCType) and
      p.model -> hd > 80;
```

} restriksjon til objekter av rett subtype i Device-tabellen

peker til objekt av PCType

b) Produsenter av laserprintere:

```
select distinct p.maker
from Product p
where deref(p.model) is of (PrinterType) and
      p.model -> type = 'laser';
```

c) ?? rar oppgavebeskrivelse .. Mones for hver produsent (manufacturer) den/de laptopene som har høyest ram? I såfall:

```
select q.maker, q.model->model
from   (select p.maker, max(p.model->ram) as maxram
        from Product p
        where deref(p.model) is of (LaptopType)
        group by p.maker) mp,
        Product q
where q.maker = mp.maker and
      q.model -> ram = mp.maxram;
```

10.5.2 Løst med PostgreSQL

10.5.2 PostgreSQL: (bruker 2. versjon av løsningsforslagene i 10.4.4)

a) Produsenter av PCer med harddisk over 80 Gb:

```
select distinct maker
from PC
where hd > 80;
```

b) Produsenter av laserskrivere:

```
select maker
from Printer
where printertype = 'laser';
```

c) For hver laptopmodell den laptopmodellen som har størst RAM fra samme forhandler:

```
select L.maker, L.model, K.model
from (select maker, max(ram) as maxram
      from Laptop
      group by maker) as m,        } de med mest RAM pr. forhandler
      Laptop L, Laptop K
where L.maker = m.maker and
      K.maker = m.maker and K.ram = m.maxram;
```

L er alle laptoper, hver
matches med de (angitt av K)
som har maks ram –
men kan ikke bare bruke m
fordi jeg ikke kan få ut
modellen også under grupperingen.

10.5.3 PostgreSQL:

10.5.3 løst med PostgreSQL

a) Slag hvor minst ett skip ble ødelagt:

```
select distinct battle
from outcome
where result = 'damaged';
```

b) Skip med minst 40000 tonn slagvolum (displacement):

```
select name
from ship
where (class).displacement >= 40 000;
```

c) Klasser med skip som ble lansert etter 1926:

```
select distinct (class).class
from ship
where launched > 1926;
```

d) Slag hvor minst ett engelsk skip ble ødelagt:

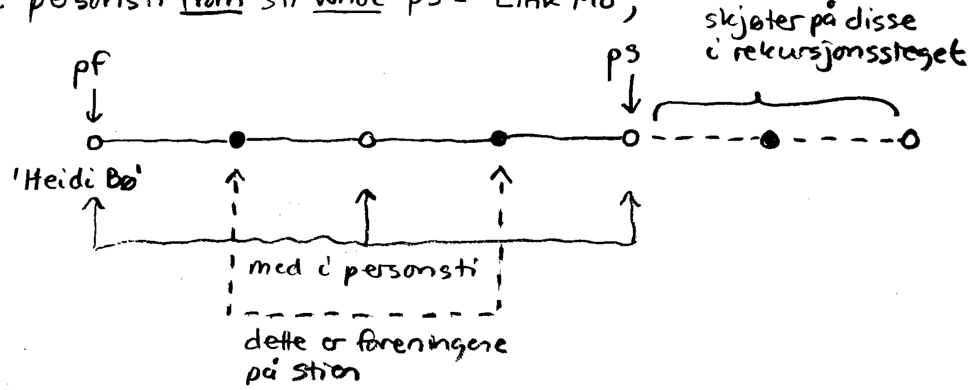
```
select distinct b.battle
from outcome b, ship s
where b.ship = s.ship and b.result = 'damaged' and
      (s.class).country = 'Gt. Britain';
```

10.x.1

Studentforening(forening, verv, person)

a) Sti mellom 'Heidi Bø' og 'Eirik Mo' med 5 eller færre personer:

```
with recursive sti(pf, ps, personsti) as (
    -- pf er første person i stien (i dette tilfellet er alltid pf = 'Heidi Bø')
    -- ps er siste person i stien
    -- personsti er alle personene på stien
  select s1.person, s2.person, array[s1.person, s2.person]
  from studentforening s1, studentforening s2
  where s1.person = 'Heidi Bø'
    and s1.forening = s2.forening
    and s1.person <> s2.person
  union all
  select s.pf, s2.person, s.personsti || s2.person
  from sti s, studentforening s1, studentforening s2
  where cardinality(s.personsti) < 5  -- avskjærer hvis stien er for lang
    and s.ps <> 'Eirik Mo' -- avskjærer hvis vi allerede har funnet
                           -- en sti til Eirik Mo
    and s.ps = s1.person
    and s1.forening = s2.forening
    and s1.person <> s2.person
    and s2.person <> all(s.personsti) -- avskjærer sykler
)
select personsti from sti where ps = 'Eirik Mo';
```



Forbedringspunkt: Ha med en sti som inneholder alle foreningene og test at s2.forening <> all(s.foreningssti). Da unngår man sykler av foreninger også.

Det er selvfølgelig unødvendig å ha med pf og ps siden pf = personsti[1] og ps = personsti[cardinality(personsti)], men det er litt lettere å lese.

10.x.1

b) Finn ut om grafen inneholder noen sykler som starter med 'Siv Dahl'.

```
with recursive sykel(pf, ps, personer, foreninger) as (
    -- pf er første person i en sti (her: alltid lik 'Siv Dahl')
    -- ps er siste person i stien
    -- personer er alle personer i stien unntatt den første
    -- foreninger er alle foreninger i stien
  select s1.person, s2.person, array[s2.person], array[s2.forening]
  from studentforening s1, studentforening s2
  where s1.person = 'Siv Dahl'
        and s1.forening = s2.forening
        and s1.person <> s2.person
  union all
  select c.pf, s2.person, c.personer || s2.person, c.foreninger || s2.forening
  from sykel c, studentforening s1, studentforening s2
  where c.ps <> 'Siv Dahl' -- avskjærer hvis vi alt har funnet en sykel
      and c.ps = s1.person
      and s1.forening = s2.forening
      and s1.person <> s2.person
      and s2.forening <> all(c.foreninger) -- avskjærer "indre" sykler
      and s2.person <> all(c.personer) -- avskjærer "indre" sykler
        -- Dette avskjærer ikke at s2.person = 'Siv Dahl'
        -- fordi 'Siv Dahl' ikke initielt er med i c.personer.
        -- Vi skal fortsette til c.personer inkluderer 'Siv Dahl'.
)
select count(*) from sykel where ps='Siv Dahl';
```

12.2.1

a) Finn alle Laptop-elementer med pris mindre enn 800.

```
let $products := doc("products.xml")
for $laptop in $products//Laptop[@price <"800"]
return $laptop
```

b) Finn alle Laptop-elementer med pris mindre enn 800, og produser sekvensen av disse elementene omgitt av en tag <CheapLaptops>.

```
let $laptopSeq := (
    let $products := doc("products.xml")
    for $laptop in $products//Laptop[@price <"800"]
    return $laptop
)
return <CheapLaptops>{$laptopSeq}</CheapLaptops>
```

Fra forrige oppg.

c) Finn alle produsenter hvor alle laptops har en pris på max 1000.

```
let $products := doc("products.xml")
for $maker in $products//Maker
where every $laptop in $maker/Laptop satisfies
      $laptop/@price <="1000"
return $maker
```

d) Finn navn på alle produsenter som lager både PCer og laptops

```
let $products := doc("products.xml")
for $maker in $products//Maker
where count($maker/Laptop) > 0 and
      count($maker/PC) > 0
return $maker/@name
```

13.2.1

a) Diskkapasitet:

- 8 overflater
- 100 000 spor pr. overflate
- hvert spor i snitt 2000 sektorer à 1024 bytes

Så $8 \cdot 100\,000 = 800\,000$ spor, hvert i snitt $2000 \cdot 1024 = 2\,048\,000$ bytes, totalt $800\,000 \cdot 2\,048\,000$ bytes $= 1,6$ terabytes.

b) Maksimal søketid: Det er når diskhodene må flyttes på tvers av samtlige spor. Det er oppgitt at tiden det tar å flytte hodene n spor, er

$$1 + 0,0003n \text{ millisekunder}$$

Her er det 100 000 spor, så må flytte hodene 99999 spor; det gir

$$1 + 0,0003 \cdot 99999 = 31 \text{ ms}$$

c) Maksimal rotasjonsforsinkelse: Det er hvis platene må rotere en gang før rett sektor er under diskhodet (hodet mistet akkurat såvidt den aktuelle sektoren).

Det er oppgitt at disken roterer 6000 rpm, så tiden for én runde er

$$60/6000 = 0,01 \text{ s} = 10 \text{ ms}$$

mente de $65536 = 1024 \cdot 64$?

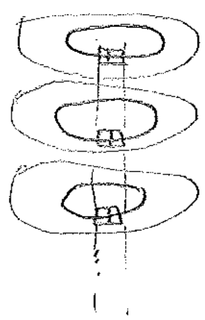
d) Én blokk er 64 sektorer eller 65546 bytes. Overføringstid er tiden det tar før de ønskede data å passere under diskhodet.

Hvis sektorene ligger samlet langs ett spor, utgjør de i snitt

$$64/2000$$

av det totale sporet, så tiden det tar er

$$(64/2000) \cdot 10 \text{ ms} = 0,32 \text{ ms}$$



Hvis sektorene ligger fordelt på samme sylinder, men forøvrig så tett som mulig, kan vi lese fra alle 8 overflatene i parallell. På hvert enkelt spor ligger i såfall $64/8 = 8$ sektorer, og tiden det tar er

$$(8/2000) \cdot 10 \text{ ms} = 0,04 \text{ ms}$$

Alternativ fremgangsmåte, alle bytene ligger "tett" langs ett spor: Kan lese

$$\underbrace{2000 \cdot \overbrace{1024}^{=1\text{Kb}}}_{\text{antall bytes pr. spor}} \cdot \underbrace{(6000/60)}_{\text{antall runder pr. sekund}} = 200\,000 \text{ Kb/s} = 200 \text{ Mb/s}.$$

Siden datamengden i det gitte eksempelet er $65546/1024 = 64$ Kb, tar det $64/200\,000 = 0,00032$ s å lese blokken.

13.3.1

cylinder of request	first time available (ms)
8000	0
48000	1
4000	10
40000	20

Hodet initielt på spor 16000.

Megatron 747

8 plater, 16 overflater

$2^{16} = 65536$ spor/overflate

Gjennomsnittlig $2^8 = 256$ sektorer/spor

$2^{12} = 4096$ B/sektor

7200 rpm

1ms for å starte og stoppe diskhodet

1ms for å passere 4000 sylindre

Gaps opptar 10% av plassen

a) Elevatoralgoritmen:

$t = 0$: Beveger seg mot spor 8000 (eneste request på dette tidspunktet

$$\underbrace{1}_{\text{start og stopp av hodet}} + \underbrace{0{,}00025}_{1/4000} \cdot (16000 - 8000) = 3\,\text{ms}$$

$t = 3$: Er kommet til spor 8000 og betjener requesten. Med tallene i boken tar det

$$\underbrace{8{,}33/2}_{\text{halv rotasjon}} + \underbrace{0{,}13}_{\text{lese en blokk}} = 4{,}3\ \text{ms}$$

$t = 7{,}3$: Request for spor 48000 er eneste nye request. Går mot denne:

$$1 + 0{,}00025 \cdot (48000 - 8000) = 11\,\text{ms}$$

$t = 18{,}3$: Ferdiggjør requesten:

4,3ms

$t = 22{,}6$: Nærmeste av de to gjenværende (4000 og 40000) er 40 000. Går mot denne:

$$1 + 0{,}00025 \cdot (48\,000 - 40\,000) = 3\,\text{ms}$$

$t = 25{,}6$: Ferdiggjør requesten:

4,3ms

$t = 29{,}9$: Spor 4000 gjenstår, går mot denne:

$$1 + 0{,}00025 \cdot (40000 - 4000) = 10\,\text{ms}$$

$t = 39{,}9$: Ferdiggjør requesten:

4,3 ms

$t = 44{,}2$: Ferdig.

13.3.1

b) First-come-first-served:

t=0: Mot 8000,

3 + 4,3 = 7,3 ms

t=7,3: Mot 48000,

11 + 4,3 = 15,3 ms

t=22,6: Mot 4000,

1 + 0,00025 · (48000 − 4000) = 12 ms
4,3 ms for å behandle

t=38,9: Mot 40 000

10 + 4,3 = 14,3 ms

t= 53,2 : Ferdig.

13.3.2

To speilede disker, den ene leser fra indre halvdel av sylindrene, den andre fra de ytre.

Anta at leseforespørslene er på tilfeldige spor og at det ikke er skriveforespørsler.

7200 rpm ~ 1 rotasjon pr. 8,33 ms

Flytte hodet: 1ms start+stopp) + 1ms pr. 4000 sylindre i forflytning

$65536 = 2^{16}$ spor pr. overflate, 8 plater med 16 overflater

$256 = 2^8$ sektorer i snitt pr. spor

$2^{12} = 4096$ bytes pr. sektor

1 blokk = 16384 bytes

gaps mellom sektorene utgjør 10% av sirkelen/sporet, sektorene utgjør 90%.

a) Gjennomsnittlig tid for lesing av en blokk:

Én blokk går over 16384 : 4096 = 4 sektorer

256 sektorer og 256 gaps pr. spor

4 sektorer og de 3 mellomliggende gapene utgjør

$$360 \cdot \frac{90}{100} \cdot \frac{4}{256} + 360 \cdot \frac{10}{100} \cdot \frac{3}{256} = 5{,}48 \text{ grader av en sirkel}$$

Overføringstiden for blokken er følgelig

$$\frac{5{,}48}{360} \cdot 8{,}33 \text{ ms} = 0{,}13 \text{ millisekunder}$$ (se forøvrig eksempel 13.2 i boka)

Gjennomsnittlig rotasjonsforsinkelse er 1/2 runde, dvs.

$$\frac{8{,}33}{2} \text{ ms} = 4{,}17 \text{ ms}$$

Gjennomsnittlig må hodet passere 1/3 av radius for å finne rett spor; tiden dette tar er (med start og stopp av hodet på 1 ms)

$$1 + \frac{65536}{2} \cdot \frac{1}{3} \cdot \frac{1}{4000} \text{ ms} = 3{,}73 \text{ ms}$$

For hver disk er altså gjennomsnittlig totaltid

$$3{,}73 + 4{,}17 + 0{,}13 \text{ ms} = 8{,}03 \text{ ms}$$ dvs. 1000/8,03 = 124 blokker pr. sekund (124,5..)

Siden de to diskene jobber i parallell, overfører de to blokker på denne tiden, så totalt

$$2000/8{,}03 = 249$$

diskblokker pr. sekund.

b) Gjennomsnittlig tid for hver disk hvis alle spor gjennomsøkes på begge, er (iflg. eksempel i boka, regnet ut som over) 10,76 ms, så totalt

$$2 \cdot 1000/10{,}76 = 185$$ diskblokker pr. sekund

a) gir en økning på 249/185 = 1,35, dvs. 35% mer enn b).

13.3.2

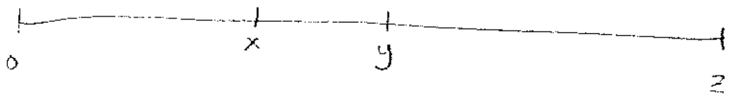
c) Kan ikke velge fra hvilken disk blokkene skal leses, så mindre fleksibilitet for scheduleren, Hvis det ikke er fullstendig randomisert hvor lesingen skjer, er man fort i en situasjon hvor én disk står for mesteparten av aksessene, og da er resultatet verre enn om begge disker leser fra alle spor: Da overføres (bare fra én disk)

124 blokker pr. sekund

, i forhold til b) er dette 185/124 = 1,49, dvs. nesten 50% dårligere enn b).

Mer om beregningene i 13.3.2 a)

Hvor langt fra hverandre i antall spor vil to tilfeldige blokker være?



Anta først at blokk x ligger fast.

a) Hvis y befinner seg i [0..x], vil avstanden til x i snitt være $\frac{x}{2}$. Dette kan vi også regne ut ved å summere over alle avstander og dele på antall spor i dette intervallet – eller vi kan få en tilnærming ved å anta at y løper over alle reelle tall i [0..x] og integrere:

$$\frac{1}{x}\int_0^x (x-y)\,dy = \frac{1}{x}\left[xy-\frac{1}{2}y^2\right]_0^x = \frac{1}{x}\left(x^2-\frac{1}{2}x^2\right) = \frac{x}{2}$$

b) Tilsvarende vil avstanden i snitt være $\frac{z-x}{2}$ hvis y befinner seg i [x..z]:

$$\frac{1}{z-x}\int_x^z (y-x)\,dy = \frac{1}{z-x}\left[\frac{1}{2}y^2-xy\right]_x^z$$

$$= \frac{1}{z-x}\left(\frac{1}{2}z^2-xz-\frac{1}{2}x^2+x^2\right) = \frac{1}{z-x}\left(\frac{1}{2}z^2-xz+\frac{1}{2}x^2\right)$$

$$= \frac{(z-x)^2}{2(z-x)} = \frac{z-x}{2}$$

c) Totalt vil vi få gjennomsnittlig avstand ved å vekte a) og b) med sannsynligheten for at y havner i [0..x] og [x..z]:

$$\frac{x}{z}\cdot\frac{x}{2} + \frac{z-x}{z}\cdot\frac{z-x}{2} = \frac{2x^2-2xz+z^2}{2z}$$

Når vi så lar x variere, får vi på tilsvarende måte snittet ved å integrere denne og dele på antall spor totalt:

$$\frac{1}{z}\int_0^z \frac{2x^2-2xz+z^2}{2z}\,dx = \frac{1}{2z^2}\left[\frac{2}{3}x^3-x^2z+xz^2\right]_0^z = \frac{1}{2z^2}\left(\frac{2}{3}z^3-z^3+z^3\right) = \underline{\underline{\frac{1}{3}z}}$$

Eksakte beregninger:

$$\frac{1}{x}\sum_{y=0}^{x}(x-y) = \frac{1}{x}\left(x\cdot x - \frac{x(x+1)}{2}\right) = \frac{x}{2} - \frac{1}{2}$$

$$\frac{1}{z-x}\sum_{y=x}^{z}(y-x) = \frac{1}{z-x}\left(\frac{z(z+1)}{2} - \frac{(x-1)x}{2} - (z-x+1)\cdot x\right) = \frac{z-x}{2} + \frac{1}{2}$$

$$\frac{x}{z}\cdot\left(\frac{x}{2}-\frac{1}{2}\right) + \frac{z-x}{z}\left(\frac{z-x}{2}+\frac{1}{2}\right) = \frac{2x^2 - 2xz + z^2 - 2x + z}{2z}$$

$$\frac{1}{z}\sum_{x=0}^{z}\frac{2x^2-2xz+z^2-2x+z}{2z} =$$

$$\frac{1}{2z^2}\left(2\cdot\frac{z(z+1)(2z+1)}{6} - 2z\cdot\frac{z(z+1)}{2} + z\cdot z^2 - 2\cdot\frac{z(z+1)}{2} + z\cdot z\right) =$$

$$\underline{\underline{\frac{1}{3}z}} - \frac{1}{3z}$$

dominerende ledd

13.4.5

RAID 4

a)

$$
\begin{array}{r}
01010110 \\
\oplus\ 11000000 \\
\oplus\ 00101011 \\
\oplus\ 10111011 \\
\hline
=\ 00000110
\end{array}
$$

b)

b)

$$
\begin{array}{r}
11110000 \\
\oplus\ 11111000 \\
\oplus\ 00111100 \\
\oplus\ 01000001 \\
\hline
=\ 01110101
\end{array}
$$

13.5.1

felt 1: char(23) DVS 23 bytes

2: integer (2 bytes) 2 bytes

3: SQL date – er på formen YYYY-MM-DD, trenger strengt talt bare 3 bytes, men hvis lagres som tegn, blir det 7 bytes

4: SQL time, uten desimalpunktum
– er på formen hh:mm:ss, trenger hvis lagret som tegn, 6 bytes

a) Felter kan starte hvor som helst:

23 + 2 + 7 + 6 = 38 bytes

b) Felter må ha et multiplum av 8:

24 + 8 + 8 + 8 = 48 bytes

c) Felter må ha et multiplum av 4:

24 + 4 + 8 + 8 = 44 bytes

13.5.2

Feller som i 13.5.1.

Recordhode: To 4-byte-pekere
Ett tegn

a)
$2 \cdot 4 + 1 + 38 = 47$ bytes

b)
$2 \cdot 8 + 8 + 48 = 72$ bytes

c) $2 \cdot 4 + 4 + 44 = 56$ bytes

13.6.1

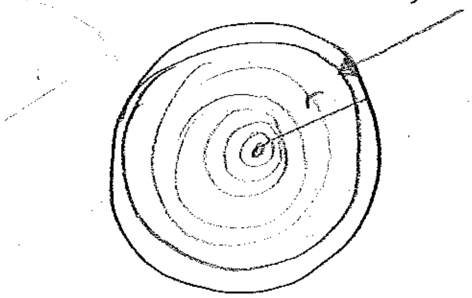
Fysiske diskadresser: Representert ved

- sylindernummer
- spor i sylinder
- blokk innen spor

Antall sylindre: 2^{16}, dvs. trenger 2 bytes for å lagre sylindernummer

Antall spor pr. sylinder: 16 overflater, så 16 spor. Trenger 1 byte

Blokk: Her trenger man å identifisere hvilken sektor som er den første i blokken. Det er gjennomsnittlig 256 sektorer pr. spor, men disken er sonet, så det er flere sektorer i de ytre sporene.
Hvor mange soner skal vi regne med i ytterste spor?



Anta at antall sektorer i et spor er proporsjonalt med sporets lengde (denne er $2\pi m$ hvis sporet ligger i radius m).
Spor ligger i radier $k \cdot d$ hvor $d = \frac{r}{2^{16}}$ og $k = 1..2^{16}$. (r er totalradius)
Hvert spor har lengde $2\pi k d$. Totallengde er derfor

$$\sum_{k=1}^{2^{16}} 2\pi k d = \cancel{2}\pi d \, \frac{2^{16}\cdot(2^{16}+1)}{\cancel{2}} \approx 2^{32}\pi d \qquad \left(\sum_{k=1}^{n} k = \frac{n(n+1)}{2}\right)$$

På denne totallengden befinner det seg $\underbrace{256}_{2^8}\cdot 2^{16}$ sektorer, dvs.

$$\frac{256\cdot 2^{16}}{2^{32}\pi d} = \frac{1}{2^{8}\pi d} \text{ sektorer pr. lengdeenhet}$$

Ytterste radius har lengde $2\pi\cdot 2^{16}d$ (her er $k=2^{16}$) og har derfor

$$\frac{2\pi\cdot 2^{16}d}{2^{8}\pi d} = 2^{9} \text{ sektorer}$$ – dvs. trenger 9 bytes for å representere blokk innen spor.

det dobbelte av gjennomsnittet – det burde man kanskje ha gjettet?

Totalt: $2 + 1 + 9 = 12$ bytes.

~~Oppgave 2~~ 13.x.1

d1	0	0	1
d2	0	1	0
d4	1	0	0
d3	0	1	1
d5	1	0	1
d6	1	1	0
d7	1	1	1

a) $d1 = d3 \oplus d5 \oplus d7 =$

$$\begin{array}{r} 00001111 \\ \oplus\ 00000100 \\ \oplus\ 11000101 \\ \hline = 11001110 \end{array}$$

$d2 = d3 \oplus d6 \oplus d7 =$

$$\begin{array}{r} 00001111 \\ \oplus\ 10011011 \\ \oplus\ 11000101 \\ \hline = 01010001 \end{array}$$

$d4 = d5 \oplus d6 \oplus d7 =$

$$\begin{array}{r} 00000100 \\ \oplus\ 10011011 \\ \oplus\ 11000101 \\ \hline = 01011010 \end{array}$$

b) Siden d2 og d4 er konstruert på grunnlag av d6, må – i tillegg til endringen på d6 – disse diskene endres. Nytt innhold:

ny d2 = gammel d2 $\oplus$ gammel d6 $\oplus$ ny d6 =

$$\begin{array}{r} 01010001 \\ \oplus\ 10011011 \\ \oplus\ 10000100 \\ \hline = 01001110 \end{array}$$

ny d4 = gammel d4 $\oplus$ gammel d6 $\oplus$ ny d6 =

$$\begin{array}{r} 01011010 \\ \oplus\ 10011011 \\ \oplus\ 10000100 \\ \hline = 01000101 \end{array}$$

I praksis kan man først beregne

gammel d6 $\oplus$ ny d6

og legge denne til både d2 og d4.

~~Oppgave 2 (forts)~~ 13.x.1 (forts.)

c) d5 kan rekonstrueres på ett av følgende vis:

d4 ⊕ d6 ⊕ d7

d1 ⊕ d3 ⊕ d7

d) d6 kan rekonstrueres fra d4 ⊕ d5 ⊕ d7 eller fra d2 ⊕ d3 ⊕ d7.
Men siden d7 også er krasjet, går ikke dette.

d7 kan rekonstrueres fra d4 ⊕ d5 ⊕ d6, d2 ⊕ d3 ⊕ d6 eller d1 ⊕ d3 ⊕ d5.
Siden d6 er krasjet, bruker vi den siste: d1 ⊕ d3 ⊕ d5.

Når d7 er rekonstruert, kan vi rekonstruere også d6, fra
f. eks. d4 ⊕ d5 ⊕ d7.

14.1.1

Blokk: 5 records

eller

20 nøkkel/peker-par

n records.

a) Dense index : Ett nøkkel/peker-par pr. record. Krever

$$\lceil n/20 \rceil \approx n/20$$

blokker i indeksen.

b) Sparse index: Ett nøkkel/peker-par pr. blokk. Krever

$$\lceil \lceil n/5 \rceil / 20 \rceil \approx n/100$$

blokker i indeksen.

I tillegg trengs $\lceil n/5 \rceil \approx n/5$ blokker til å holde selve dataene/recordene.

14.2.1

Løsningsforslag

Merk: I innledningen og i oppgave e brukes 10 records per blokk mens oppgave a-d bruker 20 records per blokk. (Trykkfeil i boken, men jeg har valgt å beholde det slik.)

Blokk: 20 records
eller
99 nøkler og 100 pekere

B-tre node snitt 70% full, dvs. 69 nøkler og 70 pekere.

a) 100 000 records, snitt 70 pekere per løvnode, gir
100 000/70 ≈ 1429 løvnoder.
Nivået over krever 1/70-del av dette, dvs 1429/70 ≈ 21 noder
Nivået over der igjen består bare av rotnoden.
Totalt $\underbrace{100\,000/20}_{\text{selve recordene}} + 1429 + 21 + 1 = 6451$ blokker, og 4 disk I/O.

b) Samme svar som i a)

c) 100 000/20 = 5000 blokker som skal ha en indeks-peker hver.
5000/70 ≈ 72 løvnoder.
Disse får alle plass i rotnoden.
Totalt 5000 + 72 + 1 = 5073 blokker, og 3 disk I/O.

d) Primærblokk + overflyt har i snitt 30 recorder tilsammen.
100 000/30 ≈ 3334 "dobbeltblokker"
3334/70 ≈ 48 løvnoder
1 rotnode.
Totalt 3334 · 2 + 48 + 1 = 6717 blokker, 3⅓ disk I/O (må lete i overflytsblokken i snitt 1/3 av gangene)

e) 7 recorder per løvnode, totalt ⌈100 000/7⌉ = 14286 løvnoder.
Nivået over krever 14286/70 ≈ 204 blokker/noder,
deretter 204/70 ≈ 3 blokker,
deretter 1 rotnode.
Totalt 14286 + 204 + 3 + 1 = 14494 blokker, og 4 disk I/O.

(For sammenligning med oppgaven over:
20 recorder per blokk gir 14 recorder per løvnode, totalt 7143 løvnoder
7143/70 ≈ 102, 102/70 ≈ 2, totalt 7143+102+2+1 = 7248 blokker, 4 disk I/O.)

15.2.1

a) Distinct: δ

```
Open() {
    b := the first block of R;
    t := the first tuple of b;
    D := Ø;                        // set of tuples seen so far
}

GetNext() { noNew := true;
  While noNew {
    If (t is past last tuple of b) {
        increment b to next block;
        If (no such block)
            return NotFound;
        else
            t := first tuple of b;
    }
    if (not t in D)
       noNew := false;
  }
  add t to D;
  return t;
}

Close() {
}
```

15.2.1

b) Grouping : γ_L

```
Open() {
    b := the first block of R;
    t := the first tuple of b;

    //Vi må sette opp hele strukturen her, og poppe ut en og
     en gruppe i GetNext();

    G := ∅; more := true;        // G er en struktur som inneholder ett
                                    element pr. gruppe;
    while more {

        if (t is past last tuple of b) {
            increment b to next block;
            if (no such block)
                more := false;
            else
                t := first tuple of G
        }

        if (more) {

            g := grouping attributes of t;
            if (not g in G)
                create new group for g in G;

            gr := G.g;  // gr is a structure containing grouping attributes' values,
                            non-grouping values for each group member, and aggregation
            add non-grouping attributes of t to gr;                        variables;

            if (min/max aggregations in L) //represent by gr.min/gr.max;
                compare t's attribute value to min/max in gr and adjust;
                                          // gr.min := min(gr.min, t.a);
            if (count aggregates in L) //represent by gr.count;
                increase count in gr by 1;    // gr.count++;

            if (sum aggregates in L)    // represent by gr.sum;
                add t's attribute value to sum in gr;   // gr.sum := gr.sum + t.a;

            if (avg aggregates in L)    // need to represent by pair (sum,count);
                add t's attribute value to sum in gr and increase count by 1
                                  // gr.sum := gr.sum + t.a; gr.count++;

        }
    }
    gr := G.first[];
}
```

15.2.1

b) (cont.)

```
GetNext() {
    if (gr is empty) {
        return NotFound;
    }
    oldgr := gr;
    gr := G.next[];
    prepare output tuple:
        if avg, compute sum/count.
    return tuple consisting of gr, grouping attributes and aggregation values;
}

Close() {
}
```

15.2.1

c) set union R ∪ S (not assuming R and S sets)

```
Open() {
    b := first block of R;
    t := first tuple of b;
    D := ∅;                  // set of tuples seen so far
    firstRel := true;
}

GetNext() {
    noNew := true;
    while noNew {
        if (t past last tuple of b) {
            increment b to next block;
            if (no such block) {
                if (firstRel) {
                    b := firstBlock of S;
                    t := first tuple of b;
                    if (t is past last tuple of b) {
                        return NotFound
                    }
                else
                    return NotFound
            }
            else
                t := first tuple of b;
        }
        if (not t in D) noNew := false;
    }
    add t to D;
    return t;
}

Close() {
}
```

15.2.3

a) $R \overset{\circ}{\bowtie}_L S$: Left outerjoin, tilsvarer naturlig join med hengende tupler fra R lagt til (med $\perp$ for de spesielle S-attributter

R plass i minnet. (Antar det er plass til minst en blokk fra S også.

Ett-pass algoritme:

```
Les R inn i minnet.
For hver S-blokk b {
    Les b inn i minnet.
    For hvert tuppel t i b {
        For hvert tuppel u i R som matcher t {
            Join t og u til output.
            Merk u som "brukt".
        }
    }
}
```

For hvert umerkede tuppel i R, legg til $\perp$ for de spesielle S-attributtene og output.

b) $R \overset{\circ}{\bowtie}_L S$

S plass i minnet. (Antar plass til minst en R-blokk også.)

Ett-pass algoritme:

```
Les S inn i minnet.
For hver R-blokk b {
    Les b inn i minnet.
    For hvert tuppel t i b {
        For hvert tuppel u i S som matcher t, join t og u til output
        Hvis ingen slik u finnes, legg ⊥-verdier til t og output
    }
}
```

Løsningsforslag

15.4.4

$B(R) = B(S) = 10.000$
$M = 500$

a) enkel sort-join (seksjon 15.4.6)

Tabellen i figur 15.11 gir at denne krever $5(B(R)+B(S)) =$
$5(10\,000 + 10\,000) = 100.000$ disk I/O.

Merk at minnekravet er $\sqrt{10\,000} = 100$, slik at algoritmen kan brukes.

b) mer effektiv sort-join (seksjon 15.4.8)

Tabell 15.11 gir nå $3(B(R)+B(S)) = 3(10.000+10\,000) = 60.000$ disk I/O.

Minnekravet er $\sqrt{10.000+10.000} \approx 142$, dvs. algoritmen kan brukes.

c) Sett-union

For to-pass sort-basert union (seksjon 15.4.4), gir tabell 15.11
samme krav som for den effektive sort-join i punkt b),
dvs. 60 000 disk I/O med minnekrav $M \geq 142$.

15.6.1

$B(R) = 10\ 000$

$T(R) = 500.000$

Indeks på R a

$V(R, a) = k$.

Hva blir kostnaden til $\sigma_{a=0}(R)$ som en funksjon av k, sett bort fra disk I/O for å aksessere selve indeksen.

a) Ikke-clustering index.

Må hente anslagsvis 500.000/k tupler, som i verste fall ligger på hver sin blokk.

Dvs. kostnaden blir 500.000/k disk I/O.

(Eventuelt max(500.000/k, 10.000) disk I/O.)

b) Clustering index.

De aktuelle tuplene er fordelt på anslagsvis 10.000/k blokker, dvs. kostnaden blir 10.000/k disk I/O.

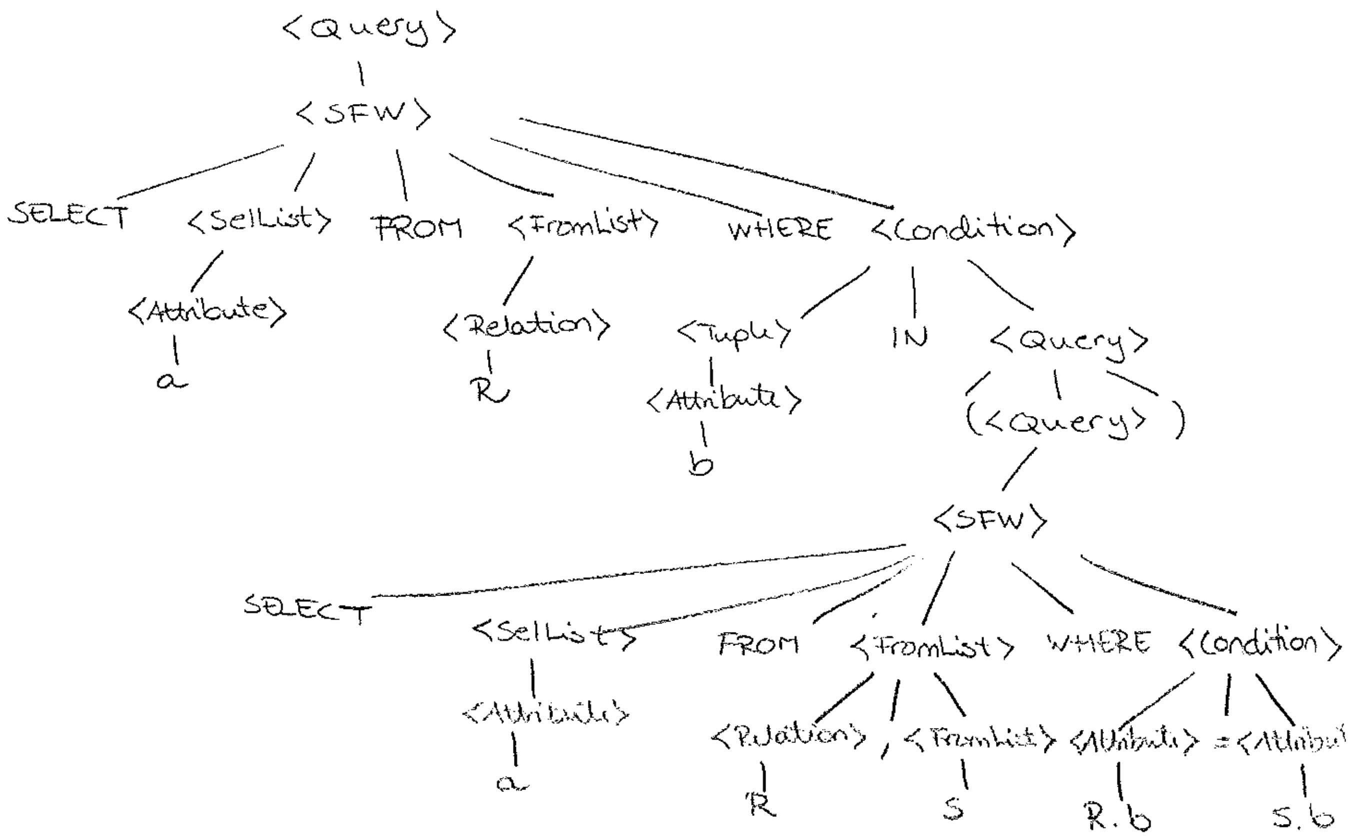
c) R er clustered, indeksen brukes ikke.

Må hente alle blokkene, dvs. 10.000 disk I/O.

16.1.3

(Bruker grammatikken på tavla.)

a)



16.2.1

a) $\delta(\pi_L(R)) \neq \pi_L(\delta(R))$:

$R = \{(a_1, b_1), (a_1, b_2)\}$

$L = A$

$\pi_A(R) = \{a_1, a_1\}$

$\delta(\pi_A(R)) = \{a_1\}$

$\delta(R) = R$

$\pi_A(\delta(R)) = \{a_1, a_1\}$

b) $\delta(R \cup_B S) \neq \delta(R) \cup_B \delta(S)$:

$R = \{a\}$

$S = \{a\}$

$R \cup_B S = \{a, a\}$

$\delta(R \cup_B S) = \{a\}$

$\delta(R) = R$

$\delta(S) = S$

$\delta(R) \cup_B \delta(S) = \{a, a\}$

$\delta(R \setminus_B S) \neq \delta(R) \setminus_B \delta(S)$:

$R = \{a, a\}$

$S = \{a\}$

$R \setminus_B S = \{a\}$

$\delta(R \setminus_B S) = \{a\}$

$\delta(R) = \{a\}$

$\delta(S) = \{a\}$

$\delta(R) \setminus_B \delta(S) = \emptyset \quad (\{\})$

1621

c) $\pi_L(R \cup_s S) \neq \pi_L(R) \cup_s \pi_L(S)$:

R:

A	B
a_1	b_1
a_2	b_2

S:

A	B
a_2	b_1

$R \cup_s S$:

A	B
a_1	b_1
a_2	b_1
a_2	b_2

$\pi_A(R)$:

A
a_1
a_2

$\pi_A(S)$:

A
a_2

$\pi_A(R \cup_s S)$:

A
a_1
a_2
a_2

$\neq$

$\pi_A(R) \cup_s \pi_A(S)$:

A
a_1
a_2

16.2.1

d) $\pi_L(R \setminus S) \neq \pi_L(R) \setminus \pi_L(S)$:

Set - differanse

Kan bruke samme eksempel for begge. Da skal $\pi_A(R)$ og $\pi_A(S)$ gjøres om til mengder før vi tar differensen.

Bag - differanse

$R = \{(a_1, b_1), (a_1, b_3)\}$

$S = \{(a_1, b_1), (a_1, b_2)\}$

$L = A$

$\pi_A(R) = \{a_1, a_1\}$

$\pi_A(S) = \{a_1, a_1\}$

$\pi_A(R) \setminus_B \pi_A(S) = \emptyset$ (den tomme mengden/bagen)

$R \setminus_B S = \{(a_1, b_3)\}$

$\pi_A(R \setminus_B S) = \{a_1\}$

16.2.6

Vurder om $\sigma_C(R \cap S)$ skal skyves til en eller begge av R og S når det er en indeks på S og ikke på R.

Merk at $\sigma_C(R \cap S) = \sigma_C(R) \cap S = R \cap \sigma_C(S) = \sigma_C(R) \cap \sigma_C(S)$. Grunnen til at det holder å skyve til det ene argumentet, er at snittet vil garantere at bare C-tupler fra det andre argumentet velges ut. Derfor vil det alltid spare én operasjon å bare skyve til det ene argumentet.

Hvis vi skyver σ_C til R, må vi gjennom hele R og velge ut, og deretter bruke resultatmengden mot (indeksen i?) S til å velge ut den endelige resultatmengden.

(NEI, må gjennom hele S, se lenger ned)

Hvis vi skyver σ_C til S, kan vi bruke indeksen til effektivt å plukke ut disse. Deretter må vi gjennom hele R på jakt etter matchende tupler.

Spørsmålet er hvilken av de to som indeksen i S gir størst gevinst på.

$\sigma_C(R) \cap S$: #R operasjoner for å teste C på hvert tuppel. $\sigma_C(R) \cap S$: algoritmen for $\cap$ utnytter ingen indeks på S; den bygger opp en struktur som involverer <u>hele</u> tupler.

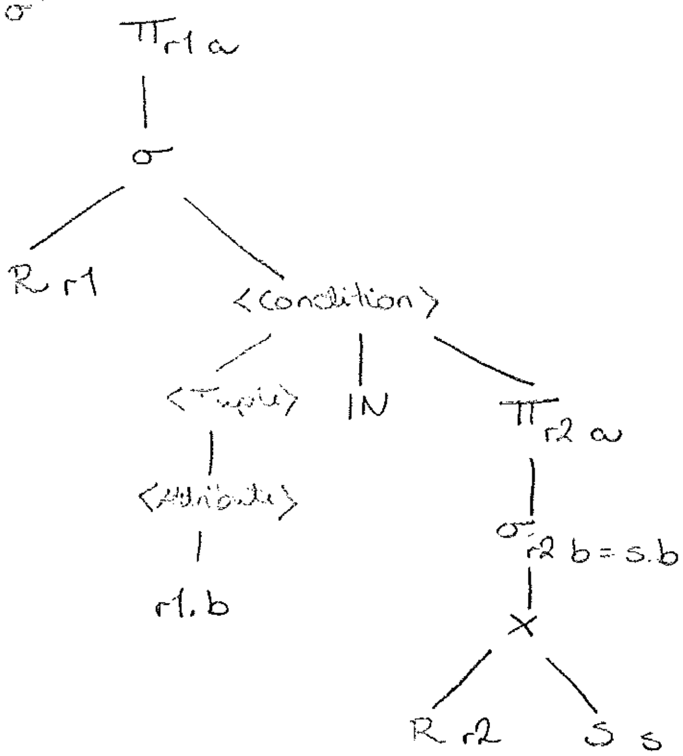
$R \cap \sigma_C(S)$: Avhengig av utseendet på C, kan indeksen på S benyttes. Hvis elementene i C bare gjelder indekserte attributter i S, vil $\sigma_C(S)$ være effektivt beregnbar og kreve vesentlig færre enn #S operasjoner. Deretter må vi gjennom alle tuplene i R og alle i utvalget $\sigma_C(S)$ for å beregne snittet.

Altså er det generelt gunstig å pushe seleksjon til den <u>indekserte</u> grenen. Hvis begge har indekser, vil det være lønnsomt å pushe til begge grener. Forutsetningen er at C-tuplene kan velges ut via indeksen(e).

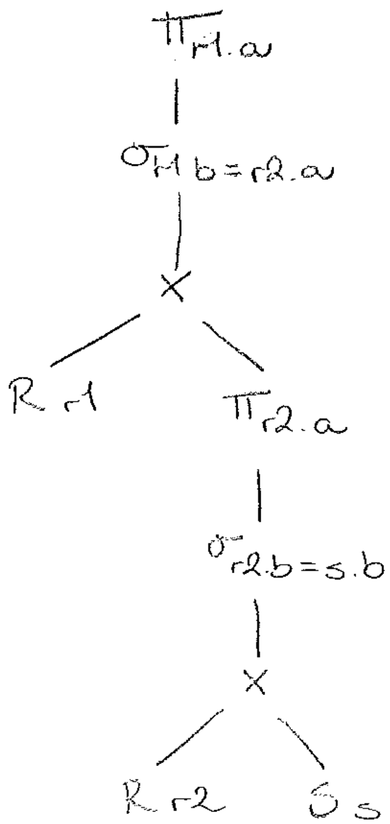
16.3.1

Løsningsforslag

a) To-argument σ:



Konvertert til vanlig σ:



Merk at denne kan optimaliseres bl.a ved å erstatte produkt med join

Løsningsforslag

16.3.2

a) $(R(a,b) \bowtie S(b,c)) \bowtie (T(c,d) \bowtie U(d,e))$

Kan skrives om til

$$\pi_{R.a, R.b, S.c}(R(a,b) \bowtie_{R.b=S.b} S(b,c)) \bowtie \pi_{T.c, T.d, U.e}(T(c,d) \bowtie_{T.d=U.d} U(d,e))$$

Som igjen kan skrives om til

$$\pi_{R.a, R.b, S.c, T.d, U.e} \Big(\pi_{R.a, R.b, S.c}(R(a,b) \bowtie_{R.b=S.b} S(b,c)) \bowtie_{S.c=T.c} \pi_{T.c, T.d, U.e}(T(c,d) \bowtie_{T.d=U.d} U(d,e)) \Big)$$

Som igjen kan skrives om til

$$\pi_{R.a, R.b, S.c, T.d, U.e}((R(a,b) \bowtie_{R.b=S.b} S(b,c)) \bowtie_{S.c=T.c} (T(c,d) \bowtie_{T.d=U.d} U(d,e)))$$

Som er assosiativ og dermed kan skrives

$$\pi_{R.a, R.b, S.c, T.d, U.e}(R(a,b) \bowtie_{R.b=S.b} S(b,c) \bowtie_{S.c=T.c} T(c,d) \bowtie_{T.d=U.d} U(d,e))$$

16.3.2

Løsningsforslag

b) $(R(a,b) \bowtie S(b,c)) \bowtie (T(c,d) \bowtie U(a,d))$

Kan skrives om til

$$\pi_{R.a, R.b, S.c, T.d} \Big(\pi_{R.a, R.b, S.c} (R(a,b) \bowtie_{R.b = S.b} S(b,c)) \bowtie_{S.c = T.c \wedge R.a = U.a} \pi_{T.c, T.d, U.a} (T(c,d) \bowtie_{T.d = U.d} U(a,d)) \Big)$$

Som igjen kan skrives om til

$$\pi_{R.a, R.b, S.c, T.d} \Big((R(a,b) \bowtie_{R.b = S.b} S(b,c)) \bowtie_{S.c = T.c \wedge R.a = U.a} (T(c,d) \bowtie_{T.d = U.d} U(a,d)) \Big)$$

Som ikke er assosiativ siden $S(b,c) \bowtie_{S.c = T.c \wedge R.a = U.a} T(c,d)$ ikke gir mening.

(Og det gjør heller ikke de to uttrykkene

$(R(a,b) \bowtie_{R.b = S.b} S(b,c)) \bowtie_{S.c = T.c \wedge R.a = U.a} T(c,d)$

$S(b,c) \bowtie_{S.c = T.c \wedge R.a = U.a} (T(c,d) \bowtie_{T.d = U.d} U(a,d))$.)

16.3.2

c) $(R(a,b) \bowtie S(b,c)) \bowtie_{R.a<T.c} T(c,d)$

Kan skrives om til

$\pi_{R.a, R.b, S.c}(R(a,b) \bowtie_{R.b=S.b} S(b,c)) \bowtie_{R.a<T.c} T(c,d)$

Denne er ikke assosiativ, da $S(b,c) \bowtie_{R.a<T.c} T(c,d)$ ikke gir mening.

16.6.4

R(a,b) S(b,c) T(c,d) U(a,d)

$T(R) = T(U) = 1000$

$T(S) = T(T) = 200$

$V(R,a) = V(R,b) = V(S,b) = V(T,d) = V(U,a) = V(U,d) = 200$

$V(S,c) = V(T,c) = 20$

a) $T(R \bowtie S) = T(R) * T(S) / \max[V(R,b), V(S,b)]$
$= 1000 * 200 / \max(200,200) = 1000$

$T(R \bowtie T)$ er ikke aktuell (ingen felles attributter)

$T(R \bowtie U) = T(R) * T(U) / \max[V(R,a), V(U,a)]$
$= 1000 * 1000 / \max(200,200) = 5000$

$T(S \bowtie T) = T(S) * T(T) / \max[V(S,c), V(T,c)]$
$= 200 * 200 / \max(20,20) = 2000$

$T(S \bowtie U)$ ikke aktuell

$T(T \bowtie U) = T(T) * T(U) / \max[V(T,d), V(U,d)]$
$= 200 * 1000 / \max(200, 200) = 1000$

En grådig algoritme vil starte med enten $R \bowtie S$ eller $T \bowtie U$, begge med 1000 tupler. Anta $R \bowtie S$.

$T((R \bowtie S) \bowtie T) = 1000 * 200 / \max(V(S,c); V(T,c))$
$= 1000 * 200 / \max(20,20) = 10000$

$T((R \bowtie S) \bowtie U) = 1000 * 1000 / \max(V(R,a), V(U,a))$
$= 1000 * 1000 / \max(200,200) = 5000.$

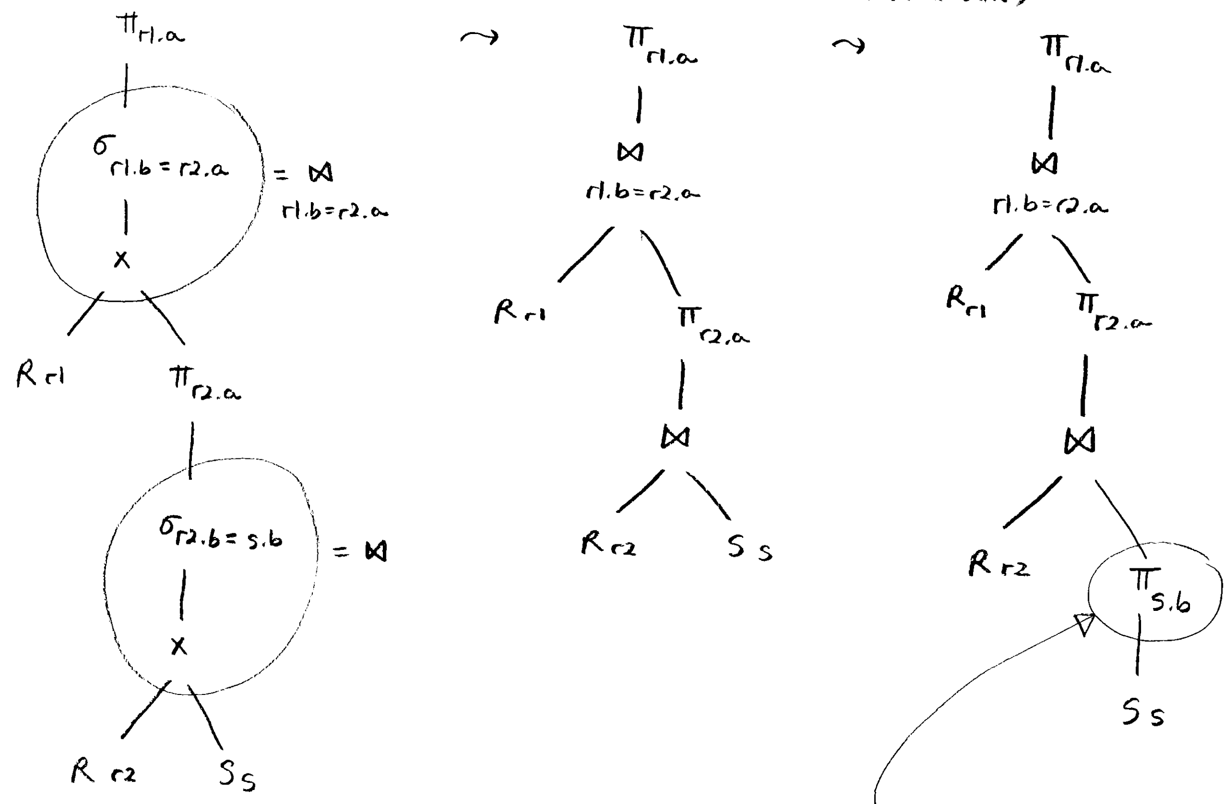
Deretter vil $(R \bowtie S) \bowtie U$ velges, med 5000 tupler.
Totalt $1000 + 5000 = 6000$ tupler til mellomresultatene.

b) Den optimale rekkefølgen er imidlertid $(R \bowtie S) \bowtie (T \bowtie U)$, med totalt $1000 + 1000 = 2000$ tupler i mellomresultatene.

16.x.1

$R(a,b), S(b,c)$

Kan innføre en π over S (men se kommentar)



Lønner seg ikke hvis det fins en indeks på S.b, for da kan ikke indeksen utnyttes i algoritmen som beregner $\bowtie$. (Indeksen gjelder den opprinnelige S, men det lages ingen indeks på mellomresultatet $\pi_b(S)$.)

16.x.2

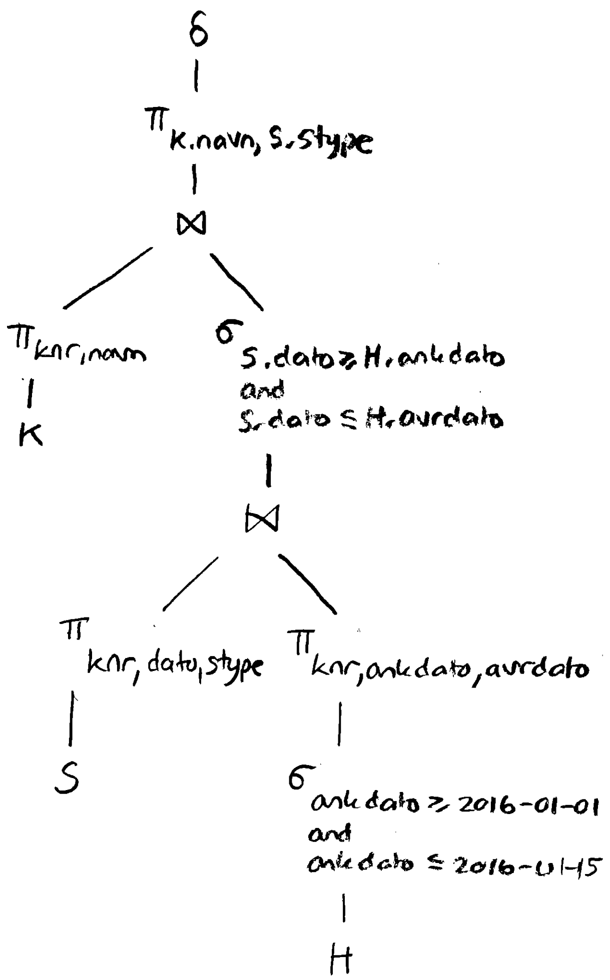
a) Spørringen finner hvilke servicetilbud de hotellgjestene som har bodd på hotellet i perioden 1.–15.1.2016 har benyttet. For hver gjest skrives ut gjestens navn og servicetilbudene (flerforekomster er fjernet).

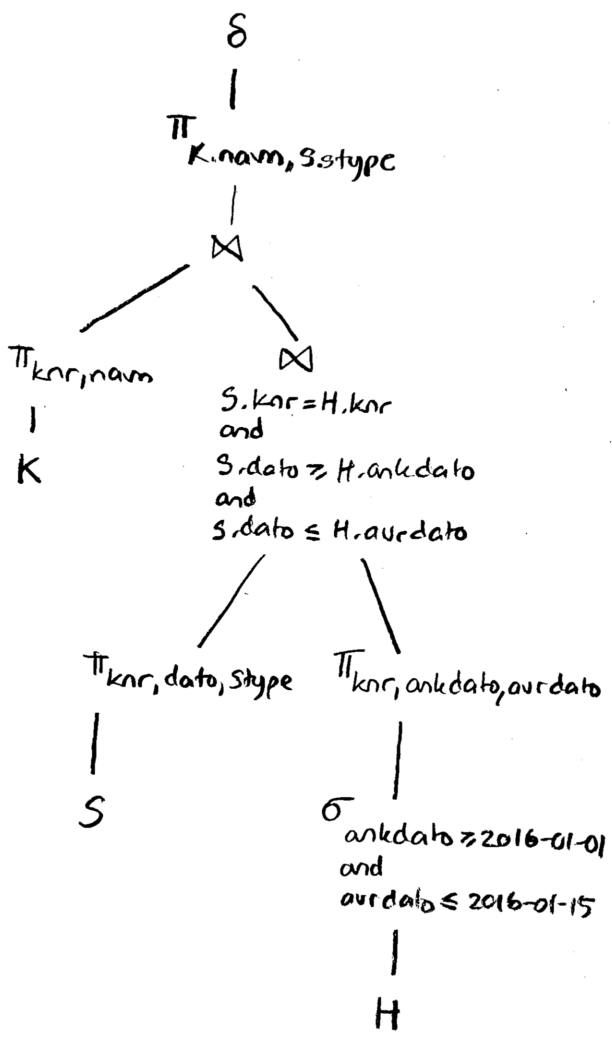
b) Noen løsningsforslag der vi har

(i) erstattet kombinasjoner av seleksjon og kartesisk produkt med join av passende type

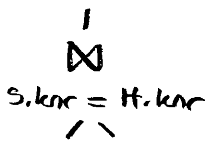
(ii) skjøvet gjenstående seleksjoner så langt ned som mulig

(iii) skjøvet projeksjon ned så langt som mulig (husk at den opprinnelige projeksjonen må beholdes hvis de nye projeksjonene har flere attributter i listen enn den opprinnelige). Vi har under ikke skjøvet projeksjonene forbi seleksjoner som skjer direkte på grunnlagstabellen fordi vi regner med at seleksjon før projeksjon gir mindre mellomresultater.





Merk at hvis vi ikke projiserer bort 'pris' fra S og H før den nederste joinen, så må denne være en equijoin



fordi en naturlig join vil joine på 'pris' også.

17.4.1

a) <start T>; <T, A, 10, 11>; <T, B, 20, 21>; <commit T>

I undo/redo-logging er kravet at loggrecordene må være på disk før endringene er på datadisken. Hvis

LX : Loggrecord skrives til disk (dvs. <T, X, v, w>)

X : Ny verdi av X skrives til disk

C : Commitrecord skrives til disk

er alle lovlige rekkefølger disse:

LA, A, LB, B, C

LA, A, LB, C, B

LA, LB, A, B, C

LA, LB, A, C, B

LA, LB, C, A, B

LA, LB, B, A, C

LA, LB, B, C, A

b) <start T>; <T, A, 10, 21>; <T, B, 20, 21>; <T, C, 30, 31>; <commit T>

Tilsvarende her (la CC stå for commit til disk)

LA, A, LB, B, LC, C, CC

LA, A, LB, B, LC, CC, C

⋮ (orker ikke skrive alle)

LA, LB, LC, CC, C, B, A

17.4.2

<start U>
<U, A, 10, 11>
<start T>
<T, B, 20, 21>
<U, C, 30, 31>
<T, D, 40, 41>
<commit T>
<U, E, 50, 51>
<commit U>

a) Crash etter at <start T> kom på loggdisken:

Verken T eller U er fullført. Begge skal rulles tilbake.

Disk etter recovery:

A	10
B	
C	
D	
E	

B–E: endres ikke

Loggdisk etter recovery:

<start U>
<U, A, 10, 11>
<start T>
<abort U>
<abort T>

b) Crash etter at <commit T> kom på loggdisken:
Løper gjennom loggen bakfra og ruller tilbake ikkecommittede (U), løper deretter gjennom loggen forover og skriver committede på nytt (T):

Disk etter recovery:

A	10
B	21
C	30
D	41
E	

E: endres ikke

Loggdisk etter recovery:

<start U>
<U, A, 10, 11>
<start T>
<T, B, 20, 21>
<U, C, 30, 31>
<T, D, 40, 41>
<commit T>
<abort U>

17.4.2

c) Crash etter at <U,E,50,51> kom på loggdisken:

A	10
B	21
C	30
D	41
E	50

<start U>
⋮
<U,E,50,51>
<abort U>

d) Crash etter at <commit U> på loggdisk:

A	11
B	21
C	31
D	41
E	51

<start U>
⋮
<commit U>

18.2.1

a) Eksempel på en serialiserbar plan:

$S = r_1(A); w_1(A); r_2(B); w_2(B); r_1(B); w_1(B); r_2(A); w_2(A)$

Bevis: Den er ekvivalent med $T_1; T_2$:

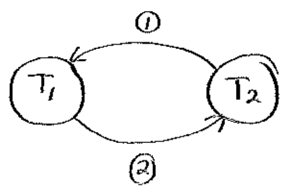
T1;T2:

T_1	T_2	A	B
		a	b
READ(A,t)			
t:=t+2			
WRITE(A,t)		a+2	
READ(B,t)			
t:=t*3			
WRITE(B,t)			3b
	READ(B,s)		
	s:=s*2		
	WRITE(B,s)		6b
	READ(A,s)		
	s:=s+3		
	WRITE(A,s)	a+5	

S:

T_1	T_2	A	B
		a	b
READ(A,t)			
t:=t+2			
WRITE(A,t)		a+2	
	READ(B,s)		
	s:=s*2		
	WRITE(B,s)		2b
READ(B,t)			
t:=t*3			
WRITE(B,t)			6b
	READ(A,s)		
	s:=s+3		
	WRITE(A,s)	a+5	

Merk at S ikke er konfliktserialiserbar:



① $S = \ldots w_2(B); r_1(B) \ldots$

② $S = \ldots w_1(A) \ldots r_2(A) \ldots$

Grafen har en sykel.

18.2.1

a) (forts.)

Eksempel på en ikke-serialiserbar plan:

$R = r_1(A); w_1(A); r_1(B); r_2(B); w_2(B); w_1(B); r_2(A); w_2(A)$

Bevis:

R:

T_1	T_2	A	B
		a	b
READ(A,t)			
t:=t+2			
WRITE(A,t)		a+2	
READ(B,t)			b
t:=t*3			
	READ(B,s)		b
	s:=s*2		
	WRITE(B,s)		2b
WRITE(B,t)			3b
	READ(A,s)		
	s:=s+3		
	WRITE(A,s)	a+5	

Det endelige resultatet av R er ikke likt $T_1; T_2$ (se forrige side), og heller ikke likt $T_2; T_1$:

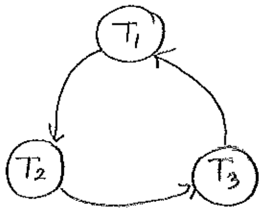
$T_2; T_1$:

T_1	T_2	A	B
		a	b
	READ(B,s)		
	s:=s*2		
	WRITE(B,s)		2b
	READ(A,s)		
	s:=s+3		
	WRITE(A,s)	a+3	
READ(A,t)			
t:=t+2			
WRITE(A,t)		a+5	
READ(B,t)			
t:=t*3			
WRITE(B,t)			6b

d) Se under a) hvor vi har vist $T_1;T_2$ og $T_2;T_1$ fra en vilkårlig initialtilstand (a,b).

18.2.5

a) $w_3(A); r_1(A); w_1(B); r_2(B); w_2(C); r_3(C)$ konflikter



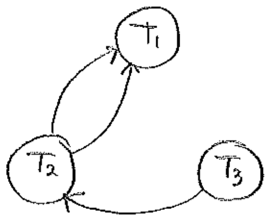
Sykel, så planen er ikke konfliktserialiserbar

Er den likevel ekvivalent med en av $T_1; T_2; T_3$ | $T_1; T_3; T_2$ | $T_2; T_3; T_1$ | ...
uansett hva som skjer med dataene?

Hvis T_1 beregner ny verdi i B på grunnlag av hva den leser i A, er det ikke likegyldig om T_3 har skrevet ny verdi i A før eller eller T_1 leser den. Tilsvarende argumentasjon gjelder for T_2 og T_3.
Så, nei: Denne planen er generelt ikke ekvivalent med noen seriell plan.

18.2.5

d) $r_1(A); r_2(A); r_3(B); w_1(A); r_2(C); r_2(B); w_2(B); w_1(C)$



Grafen er sykelfri, så planen er konfliktserialiserbar
(Planen er ekvivalent med $T_3; T_2; T_1$)

Siden T_3 bare <u>leser</u> verdier, har den ingen innvirkning på databasetilstanden. Så faktisk vil $T_2; T_1; T_3$ ha samme effekt på <u>databasen</u>, ergo er den <u>ekvivalent</u>, men den er <u>ikke</u> <u>konflikt</u>ekvivalent med planen over (Men applikasjonen som initierte T_3, vil generelt få en annen avlesning fra planen $T_2; T_1; T_3$ enn den som den opprinnelige planen gav)

18.3.2

a) T_1 : $l_1(A); r_1(A); l_1(B); w_1(B); u_1(A); u_1(B)$

T_2 : $l_2(B); r_2(B); l_2(C); w_2(C); u_2(B); u_2(C)$

T_3 : $l_3(A); w_3(A); l_3(C); r_3(C); u_3(A); u_3(C)$

T_1	T_2	T_3
		$l_3(A)$
		$w_3(A)$
$l_1(A)$ – vente		
	$l_2(B)$	
	$r_2(B)$	
	$l_2(C)$	
	$w_2(C)$	
	$u_2(B)$	
	$u_2(C)$	
		$l_3(C)$
		$r_3(C)$
		$u_3(A)$
$l_1(A)$ – innvilget		
		$u_3(C)$
$r_1(A)$		
$l_1(B)$		
$w_1(B)$		
$u_1(A)$		
$u_1(B)$		

18.3.2

d)

T_1: $l_1(A); r_1(A); w_1(A); l_1(C); w_1(C); u_1(A); u_1(C)$

T_2: $l_2(A); r_2(A); l_2(C); r_2(C); l_2(B); r_2(B); w_2(B); u_2(A); u_2(C); u_2(B)$

T_3: $l_3(B); r_3(B); u_3(B)$

T_1	T_2	T_3
$l_1(A)$		
$r_1(A)$		
	$l_2(A)$ - vent	
		$l_3(B)$
		$r_3(B)$
		$u_3(B)$
$w_1(A)$		
$l_1(C)$		
$w_1(C)$		
$u_1(A)$		
$u_1(C)$		
	$l_2(A)$ - tildelt	
	$r_2(A)$	
	$\vdots$	
	$u_2(B)$	

18.4.1

b)

(i) SL hvis kun leseaksjon på et element, XL ellers, u til slutt:

$sl_1(A); r_1(A); sl_2(B); r_2(B); sl_3(C); r_3(C);$

$sl_1(B); r_1(B); sl_2(C); r_2(C); sl_3(D); r_3(D);$

$xl_1(C); w_1(C); u_1(A); u_1(B); u_1(C); xl_2(D); w_2(D)$

$u_2(B); u_2(C); u_2(D); xl_3(E); w_3(E) u_3(C); u_3(D); u_3(E);$

(ii)

T_1	T_2	T_3
$sl_1(A)$		
$r_1(A)$		
	$sl_2(B)$	
	$r_2(B)$	
		$sl_3(C)$
		$r_3(C)$
$sl_1(B)$		
$r_1(B)$		
	$sl_2(C)$	
	$r_2(C)$	
		$sl_3(D)$
		$r_3(D)$
$xl_1(C)$ - vent		
	$xl_2(D)$ - vent	
		$xl_3(E)$
		$w_3(E)$
		$u_3(C)$
		$u_3(D)$
		$u_3(E)$
	$xl_2(D)$	
	$w_2(D)$	
	$u_2(B)$	
	$u_2(C)$	
	$u_2(D)$	
$xl_1(C)$		
$w_1(C)$		
$u_1(A)$		
$u_1(B)$		
$u_1(C)$		

(iii)-(vi): Ikke aktuelle; intet behov for oppgradering av låser.

18.4.1

c)

(i) $xl_1(A); r_1(A); xl_2(B); r_2(B); xl_3(C); r_3(C);$
$sl_1(B); r_1(B); sl_2(C); r_2(C); sl_3(A); r_3(A);$
$w_1(A); u_1(A); u_1(B); w_2(B); u_2(B); u_2(C);$
$w_3(C); u_3(C); u_3(A);$

(ii)

T_1	T_2	T_3
$xl_1(A)$		
$r_1(A)$		
	$xl_2(B)$	
	$r_2(B)$	
		$xl_3(C)$
		$r_3(C)$
$sl_1(B)$ –vent		
	$sl_2(C)$ –vent	
		$sl_3(A)$ –deadlock

(iii) $sl_1(A); r_1(A); sl_2(B); r_2(B); sl_3(C); r_3(C);$
$sl_1(B); r_1(B); sl_2(C); r_2(C); sl_3(\overset{A}{\cancel{D}}); r_3(\overset{A}{\cancel{D}});$
$xl_1(A); w_1(A); u_1(A); u_1(B); xl_2(B); w_2(B);$
$u_2(B); u_2(C); xl_3(C); w_3(C); u_3(C); u_3(\cancel{D}/A)$

18.4.1

c) (forts.)

(iv)

T_1	T_2	T_3
$sl_1(A)$		
$r_1(A)$		
	$sl_2(B)$	
	$r_2(B)$	
		$sl_3(C)$
		$r_3(C)$
$sl_1(B)$		
$r_1(B)$		
	$sl_2(C)$	
	$r_2(C)$	
		$sl_3(A)$
		$r_3(A)$
$xl_1(A)$ - vent		
	$xl_2(B)$ - vent	
		$xl_3(C)$ - vranglås

18.4.1

c) (forts.)

(v) $ul_1(A); r_1(A); ul_2(B); r_2(B); ul_3(C); r_3(C);$
$sl_1(B); r_1(B); sl_2(C); r_2(C); sl_3(A); r_3(A);$
$xl_1(A); w_1(A); u_1(A); u_1(B); xl_2(B); w_2(B);$
$u_2(B); u_2(C); xl_3(C); w_3(C); u_3(C); u_3(A);$

(vi)

Anta prioritet til skrivere; da får ingen ta sl etter at ul er tatt

T_1	T_2	T_3
$ul_1(A)$		
$r_1(A)$		
	$ul_2(B)$	
	$r_2(B)$	
		$ul_3(C)$
		$r_3(C)$
$sl_1(B)$ – vent		
	$sl_2(C)$ – vent	
		$sl_3(A)$ – vranglås

Alternativt: prioritet til lesere

T_1	T_2	T_3
$sl_1(B)$		
$r_1(B)$		
	$sl_2(C)$	
	$r_2(C)$	
		$sl_3(A)$
		$r_3(A)$
$xl_1(A)$ – vent		
	$xl_2(B)$ – vent	
		$xl_3(C)$ – vranglås

18.4.1

e)

(i) $xl_1(A)$; $r_1(A)$; $xl_2(B)$; $r_2(B)$; $xl_3(C)$; $r_3(C)$;
$sl_1(B)$; $r_1(B)$; $sl_2(C)$; $r_2(C)$; $sl_3(D)$; $r_3(D)$;
$w_1(A)$; $u_1(A)$; $u_1(B)$; $w_2(B)$; $u_2(B)$; $u_2(C)$;
$w_3(C)$; $u_3(C)$; $u_3(D)$;

(ii)

T_1	T_2	T_3
$xl_1(A)$		
$r_1(A)$		
	$xl_2(B)$	
	$r_2(B)$	
		$xl_3(C)$
		$r_3(C)$
$sl_1(B)$ – vent		
	$sl_2(C)$ – vent	
		$sl_3(D)$
		$r_3(D)$
		$w_3(C)$
		$u_3(C)$
		$u_3(D)$
	$sl_2(C)$	
	$r_2(C)$	
	$u_2(B)$	
	$u_2(C)$	
$sl_1(B)$		
$r_1(B)$		
$w_1(A)$		
$u_1(A)$		
$u_1(B)$		

18.4.1

e) (forts.)

(iii) $sl_1(A)$; $r_1(A)$; $sl_2(B)$; $r_2(B)$; $sl_3(C)$; $r_3(C)$;
$sl_1(B)$; $r_1(B)$; $sl_2(C)$; $r_2(C)$; $sl_3(D)$; $r_3(D)$;
$xl_1(A)$; $w_1(A)$; $u_1(A)$; $u_1(B)$; $xl_2(B)$; $w_2(B)$;
$u_2(B)$; $u_2(C)$; $xl_3(C)$; $u_3(C)$; $u_3(D)$

(iv)

T_1	T_2	T_3
$sl_1(A)$		
$r_1(A)$		
	$sl_2(B)$	
	$r_2(B)$	
		$sl_3(C)$
		$r_3(C)$
$sl_1(B)$		
$r_1(B)$		
	$sl_2(C)$	
	$r_2(C)$	
		$sl_3(D)$
		$r_3(D)$
$xl_1(A)$		
$w_1(A)$		
$u_1(A)$		
$u_1(B)$		
	$xl_2(B)$	
	$w_2(B)$	
	$u_2(B)$	
	$u_2(C)$	
		$xl_3(C)$
		$u_3(C)$
		$u_3(D)$

18.4.1

e) (forts.)

(v) $ul_1(A)$; $r_1(A)$; $ul_2(B)$; $r_2(B)$; $ul_3(C)$; $r_3(C)$;
$sl_1(B)$; $r_1(B)$; $sl_2(C)$; $r_2(C)$; $sl_3(D)$; $r_3(D)$;
$xl_1(A)$; $w_1(A)$; $u_1(A)$; $u_1(B)$; $xl_2(B)$; $w_2(B)$;
$u_2(B)$; $u_2(C)$; $xl_3(C)$; $u_3(C)$; $u_3(D)$

(vi)

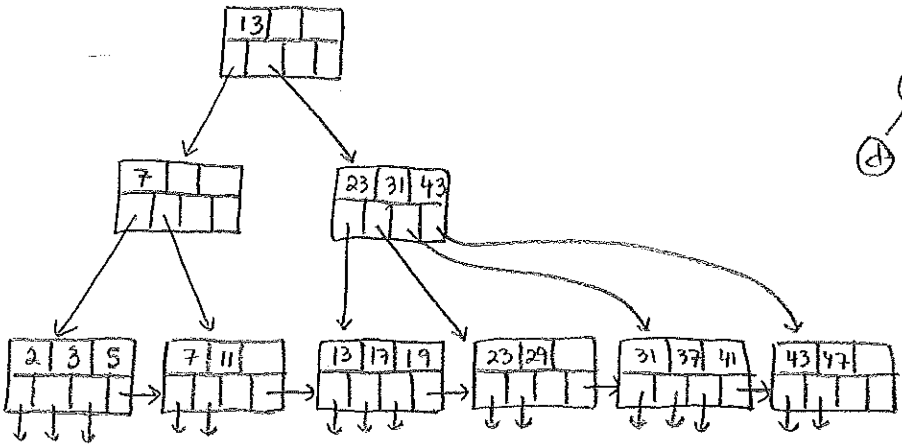
Antar prioritet til skrivere; da får ingen ta sl etter at ul er tatt →

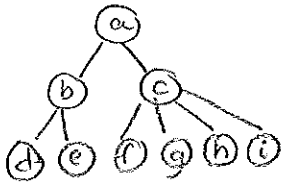
T_1	T_2	T_3
$ul_1(A)$		
$r_1(A)$		
	$ul_2(B)$	
	$r_2(B)$	
		$ul_3(C)$
		$r_3(C)$
$sl_1(B)$		
- vent		
	$sl_2(C)$	
	- vent	
		$sl_3(D)$
		$r_3(D)$
		$xl_3(C)$
		$u_3(C)$
		$u_3(D)$
	$sl_2(C)$	
	$r_2(C)$	
	$xl_2(B)$	
	$w_2(B)$	
	$u_2(B)$	
	$u_2(C)$	
$sl_1(B)$		
$r_1(B)$		
$xl_1(A)$		
$w_1(A)$		
$u_1(A)$		
$u_1(B)$		

Alternativt: Prioritet til lesere

T_1	T_2	T_3 (forts)
$sl_1(B)$		
$r_1(B)$		
	$sl_2(C)$	
	$r_2(C)$	
		$sl_3(D)$
		$r_3(D)$
$xl_1(A)$		
$w_1(A)$		
$u_1(A)$		
$u_1(B)$		
	$xl_2(B)$	
	$w_2(B)$	
	$u_2(B)$	
	$u_2(C)$	
		$xl_3(C)$
		$u_3(C)$
		$u_3(D)$

18.7.1





Treprotokollen:

1. I B-trær settes første lås på roten
2. Videre låser kan tildeles noder hvor foreldrenoden er låst av transaksjonen
3. Låser kan frigis når som helst
4. Låser som er frigitt, kan ikke relåses av samme transaksjon

a) Innsetting av 4: Låser a, b. Ser i b at uansett hva som skjer under b, er det plass i b til en ny peker. Så låser opp a. Låser d. Splitter i to noder og fordeler 2,3 på første og 4,5 på andre. Låser opp d. Justere innholdet i b, låser opp b.

b) Innsetting av 30: Låser a, c, g. Her kan ikke a låses opp ennå fordi vi ikke vet før vi kommer videre ned om innsettingen vil kreve en splitting av et løv og påfølgende splitting av c. Når node g nås og vi ser at det er plass, kan a og c låses opp før g oppdateres. Så oppdateres g og g låses opp.

c) Slett 37: Låser a, c. Ser i c at uansett hva som skjer under c, vil ikke slettingen medføre at c skal slås sammen med en av sine søsken. Så låser opp a. Låser h. Ser at slettingen ikke krever påfølgende sammenslåing av noder, så låser opp c. Endrer i h, låser opp h.

Tekst forholder seg til det opprinnelige treet, ikke slik det ser ut etter pkt. a)

d) Slett 7: Låser a, b. Her kan vi ikke frigi låsen i a ennå, for b inneholder minimalt antall pekere og kan derfor bli slått sammen med c, hvilket påvirker også innholdet i a. Låser e. Ser at e kommer under kritisk grense og låser derfor d for å se om en peker i d kan flyttes til e. Det kan den siden d er over nedre grense. Straks vi vet dette, vet vi også at det ikke blir noen sammenslåinger av noder, så låsen på a kan frigis. b må imidlertid også justeres, så vi holder låsen i denne. Innholdet i d justeres, låsen til d frigis. Innholdet i b justeres, låsen til b frigis. Innholdet i e justeres, låsen frigis.

18.8.1

a)

T_1	T_2	RT(A)	WT(A)	C(A)	RT(B)	WT(B)	C(B)
		t_0	u_0	true	v_0	w_0	true
st_1							
$r_1(A)$		st_1					
	st_2						
	$w_2(B)$					st_2	false
	$r_2(A)$	st_2					
Skriver for sent $w_1(B)$							
	c_2						true

– Vent til C(B) er sann, sjekk situasjonen på nytt.

– Velkes opp, bruk Thomas' skriveregel og gjør ingen endringer.

Eller c2 gjøres umiddelbart etter r2(A). Da vil fortsatt w1(B) komme for sent. Anvend Thomas' skriveregel: La T1 fortsette uten å skrive B.

b)

T_1	T_2	RT(A)	WT(A)	C(A)	RT(B)	WT(B)	C(B)
		t_0	u_0	true	v_0	w_0	true
st_1							
	st_2						
$r_1(A)$		st_1					
	$r_2(B)$				st_2		
	$w_2(A)$		st_2	false			
$w_1(B)$							

– rull tilbake; T_2 har alt lest B.

18.8.1 , men SI-protokollen FUW: Legger inn skrivelåser i forkant av skriveoperasjoner. Noterer på leseoperasjonene hvilken versjon som blir lest, og på skriveoperasjonene hvilken versjon som ble skrevet (navngis etter den transaksjonen som skrev).

a)

T_1	T_2	A	B	commit(A)	commit(B)
		a_0	b_0		
st_1					
$r_1(a_0)$					
	st_2				
	$l_2(B)$				
	$w_2(b_2)$				
	$r_2(a_0)$				
	c_2		b_2		T_2
	$u_2(B)$				

$l_1(B)$ –
avslått: T_2 var samtidig ($T_2 \in$ commit(B) og c_2 skjedde etter st_1)
a_1 —Merk at låsen avslås og T_2 må rulles tilbake selv om låsen var ledig.

b)

T_1	T_2	A	B	commit(A)	commit(B)
		a_0	b_0		
st_1					
	st_2				
$r_1(a_0)$					
	$r_2(b_0)$				
	$l_2(A)$				
	$w_2(a_2)$				
	c_2	a_2		T_2	
	$u_2(A)$				
$l_1(B)$					

innvilget: T_2 var samtidig, men har ikke overlappende skrivemengde med T_1 ($T_2 \notin$ commit(B))

$w_1(b_1)$					
c_1			b_1		T_1
$u_1(B)$					

18.8.1

c)

T_1	T_2	T_3	A	B	C	commit(A)	commit(B)	commit(C)
st_1			a_0	b_0	c_0			
		st_3						
	st_2							
$r_1(a_0)$								
		$r_3(b_0)$						
$L_1(C)$								
$w_1(c_1)$,								
c_1					c_1			T_1
$u_1(C)$	$r_2(b_0)$							
	$r_2(c_0)$							
		$L_3(B)$						
		$w_3(b_3)$						
		c_3		b_3			T_3	
		$u_3(B)$						
	$L_2(A)$							
	$w_2(a_2)$							
	c_2		a_2			T_2		
	$u_2(A)$							

Leser c_0 selv om c_1 er tilgjengelig fordi c_0 var siste committede verdi på det tidspunktet T_2 startet (st_2)

T_1, T_2 og T_3 har ikke-overlappende skrivemengder

Valideringsprotokollen:

- Hvis T vil validere, må vi være sikre på at T ikke har lest for gamle verdier. Dette kan skje hvis en eldre transaksjon som ikke var avsluttet da T begynte, har skrivemengde som overlapper med Ts lesemengde.
- Hvis T vil validere, må vi være sikre på at dersom T ihnulges dette, kan vi ikke risikere at T skriver for tidlig. Dette kan skje hvis en eldre transaksjon har en skrivemengde som overlapper med Ts skrivemengde.
- Transaksjoner fjernes fra FIN når alle i START har starttidspunkt etter deres sluttidspunkt.

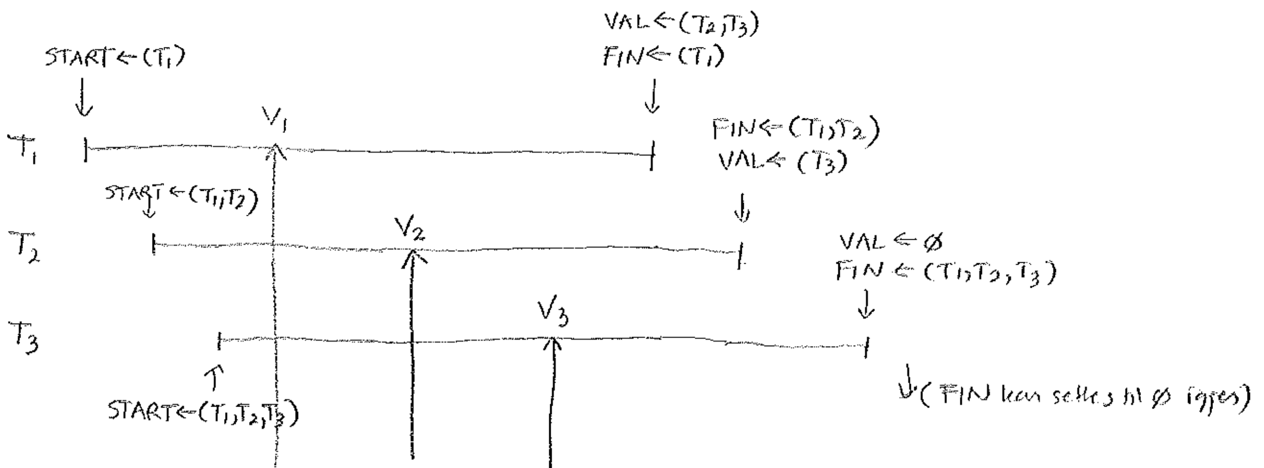
Mer presist: Fins en U hvor

$U \in VAL$ og $RS(T) \cap WS(U) \neq \emptyset$
eller
$U \in FIN$ og $RS(T) \cap WS(U) \neq \emptyset$

$U \in VAL$ og $WS(T) \cap WS(U) \neq \emptyset$
(Hvis $U \in FIN$, er den ferdig med skrivingen.)

18.9.1

a)



$START = (T_1, T_2, T_3)$
$VAL = \emptyset$
$FIN = \emptyset$

Ingen i VAL eller FIN, T_1 kan validere.

$START \leftarrow (T_2, T_3)$
$VAL \leftarrow (T_1)$

$START = (T_2, T_3)$
$VAL = (T_1)$

$RS(T_1) = (A, B)$
$WS(T_1) = (A)$
$RS(T_2) = (B, C)$
$WS(T_2) = (B)$
$RS(T_3) = (C)$
$WS(T_3) = (C)$

T_1 er validert, men ikke avsluttet. Må sjekke T_2s lese- og skrivemengder mot T_1s skrivemengde:

$RS(T_2) \cap WS(T_1) = \emptyset$
$WS(T_2) \cap WS(T_1) = \emptyset$

T_2 validerer

$START = (T_3)$
$VAL = (T_1, T_2)$

T_1 og T_2 er begge validert, men ikke avsluttet. Må sjekke T_3s lese- og skrivemengder mot T_1 og T_2s skrivemengder:

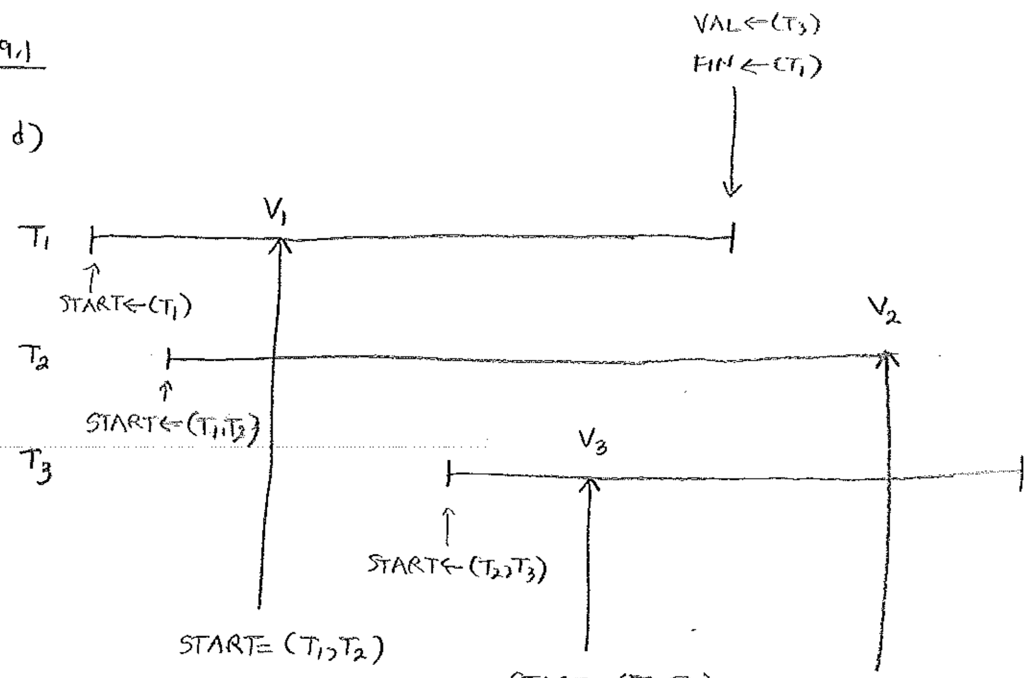
$RS(T_3) \cap WS(T_1) = \emptyset$
$WS(T_3) \cap WS(T_1) = \emptyset$
$RS(T_3) \cap WS(T_2) = \emptyset$
$WS(T_3) \cap WS(T_2) = \emptyset$

T_3 validerer

$START \leftarrow \emptyset$
$VAL \leftarrow (T_1, T_2, T_3)$

18.9.1

d)



START= (T_1, T_2)

VAL= $\emptyset$

FIN= $\emptyset$

$RS(T_1) = (A, B)$

$WS(T_1) = (A)$

Det er ingen i valideringsfasen og ingen nylig avsluttede.

T_1 valideres.

START ← (T_2)

VAL ← (T_1)

START = (T_2, T_3)

VAL = (T_1)

FIN = $\emptyset$

$RS(T_3) = (C, D)$

$WS(T_3) = (B)$

T_1 er validert, men ikke avsluttet. Må sjekke T_3 mot T_1s skrivemengde:

$RS(T_3) \cap WS(T_1) = \emptyset$

$WS(T_3) \cap WS(T_1) = \emptyset$

Ingen konflikter,

T_3 kan validere.

START ← (T_2)

VAL ← (T_1, T_3)

START = (T_2)

VAL = (T_3)

FIN = (T_1)

$RS(T_2) = (B, C)$

$WS(T_2) = A$

T_3 er validert, men ikke avsluttet. T_1 er avsluttet. Må sjekke T_2s lese- og skrivemengder mot T_3s skrivemengde.

Må sjekke T_2s lesemengde mot T_1s skrivemengde.

$RS(T_2) \cap WS(T_3) = (B) \neq \emptyset$

$WS(T_2) \cap WS(T_3) = \emptyset$

$RS(T_2) \cap WS(T_1) = \emptyset$

Konflikt mellom T_2 og T_3: Siden T_3 ikke er ferdig med all skriving, kan det være at T_2 leser B før T_3 skriver den.

T_2 må rulles tilbake og startes på nytt.

18.x.1

a)

T_1	T_2	a	b	commit(a)	commit(b)
		a_0	b_0		
$r_1(a_0)$					
	$r_2(a_0)$				
$r_1(b_0)$					
	$l_2(a)$				
	$w_2(a_2)$				
$l_1(a)$ - vent					
	$r_2(b_0)$				
	$l_2(b)$				
	$w_2(b_2)$				
	c_2	a_2	b_2	T_2	T_2
	$u_2(a)$				
$l_1(a)$ avslås: a_1					
	$u_2(b)$				

18.x.1

b)

T_1	T_2	a	b	commit(a)	commit(b)
		a_0	b_0		
$r_1(a_0)$					
	$r_2(b_0)$				
	$l_2(b)$				
	$w_2(b_2)$				
	c_2		b_2		T_2
	$u_2(b)$				
$r_1(b_0)$					
$l_1(a)$					
$w_1(a_1)$					
$l_1(b)$ -					
avslås					
a_1					
$u_1(a)$					

18.x.1

c)

T_1	T_2	T_3	a	b	commit(a)	commit(b)
			a_0	b_0		
$r_1(a_0)$						
	$r_2(a_0)$					
		$r_3(a_0)$				
$l_1(b)$						
$w_1(b_1)$						
c_1				b_1		T_1
$u_1(b)$						
		$r_3(b_0)$				
	$l_2(a)$					
	$w_2(a_2)$					
		$l_3(a)$ - vent				
	$l_2(b)$ - avslått					
	$\cancel{c_2}$ a_2					
	$u_2(a)$					
		fortsett:				
		$w_3(a_3)$				
		c_3	a_3		T_3	
		$u_3(a)$				

19.1.2

a)

T_1	T_2	T_3
$r_1(A)$		
$w_1(B)$		
		$r_3(B)$ (dirty)
		$w_3(C)$
	$r_2(C)$ (dirty)	
	$w_2(D)$	
a_1		

Alle tre må rulles tilbake.

d)

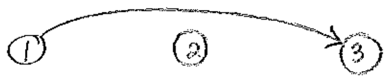
T_1	T_2	T_3
$r_1(A)$		
		$r_3(B)$
$w_1(B)$		
		$w_3(C)$
	$r_2(B)$ (dirty)	
	$r_2(C)$ (dirty)	
	$w_2(D)$	
a_1		

T_1 og T_2 må rulles tilbake.

19.2.1

a)

T_1	T_2	T_3
$sl_1(A)$		
$r_1(A)$		
		$sl_3(B)$
		$r_3(B)$
	$sl_2(C)$	
	$r_2(C)$	
$xl_1(B)$ (avslått)		
		$xl_3(C)$ (avslått)
	$xl_2(D)$	
	$w_2(D)$	
	$u_2(C)$	
	$u_2(D)$	
	C_2	$xl_3(C)$
		$w_3(C)$
		$u_3(B)$
		$u_3(C)$
$xl_1(B)$		C_3
$w_1(B)$		
$u_1(A)$		
$u_1(B)$		
C_1		

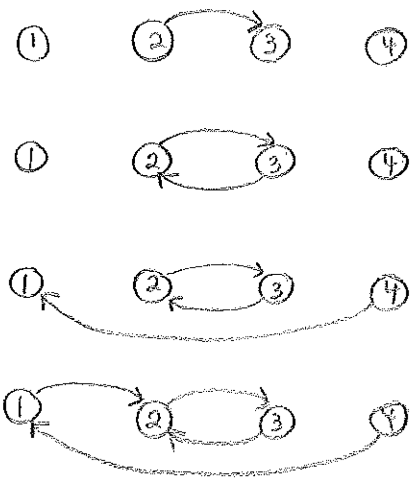


Ingen vranglås.

A.2.1

d)

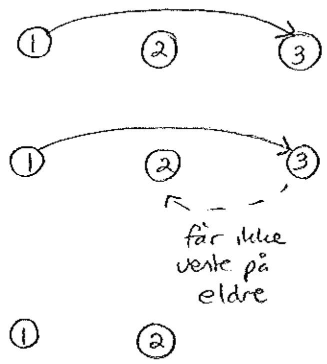
	T_1	T_2	T_3	T_4
	$sl_1(A)$ $r_1(A)$			
			$sl_3(B)$ $r_3(B)$	
		$xl_2(C)$ $w_2(C)$ $sl_2(D)$ $r_2(D)$		
				$sl_4(E)$ $r_4(E)$
		$xl_2(B)$ (avslått)		
Vranglås mellom T_2 og T_3 →			$xl_3(C)$ (avslått)	
				$xl_4(A)$ (avslått)
T_1 og T_4 er også involvert →	$xl_1(D)$ (avslått)			
Abortere T_3 →			a_3 $u_3(B)$ $u_3(C)$	
		$xl_2(B)$ $w_2(B)$ $u_2(C)$ $u_2(D)$ $u_2(B)$ c_2	(må startes på nytt)	
	$xl_1(D)$ $w_1(D)$ $u_1(A)$ $u_1(D)$ c_1			
				$xl_4(A)$ $w_4(A)$ $u_4(E)$ $u_4(A)$ c_4



19.2.2

a)

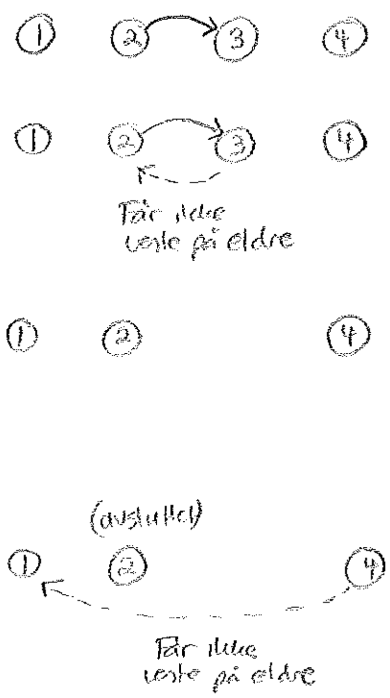
T_1	T_2	T_3
$sl_1(A)$		
$r_1(A)$		
		$sl_3(B)$
		$r_3(B)$
	$sl_2(C)$	
	$r_2(C)$	
$xl_1(B)$ (avslått)		
		$xl_3(C)$
		a_3
		$u_3(B)$
$xl_1(B)$		(star på nytt)
$w_1(B)$		
$u_1(A)$		
$u_1(B)$		
c_1		
	$xl_2(D)$	
	$w_2(D)$	
	$u_2(C)$	
	$u_2(D)$	
	c_2	



19.2.2

d)

T_1	T_2	T_3	T_4
$sl_1(A)$			
$r_1(A)$			
		$sl_3(B)$	
		$r_3(B)$	
	$xl_2(C)$		
	$w_2(C)$		
	$sl_2(D)$		
	$r_2(D)$		
			$sl_4(E)$
			$r_4(E)$
	$xl_2(B)$ (avslått)		
		$xl_3(C)$	
		a_3	
		$u_3(B)$	
		$u_3(C)$	
	$xl_2(B)$ (start på nytt)		
	$w_2(B)$		
	$u_2(C)$		
	$u_2(D)$		
	$u_2(B)$		
	c_2		
			$xl_4(A)$
			a_4
			$u_4(E)$ (start på nytt)
$xl_1(D)$			
$w_1(D)$			
$u_1(A)$			
$u_1(D)$			
c_1			



19.2.3

a)

T_1	T_2	T_3
$sl_1(A)$		
$r_1(A)$		
		$sl_3(B)$
		$r_3(B)$
	$sl_2(C)$	
	$r_2(C)$	
$xl_1(B)$ (sårer T_3)		
		a_3
		$u_3(B)$
$xl_1(B)$		(start på nytt)
$w_1(B)$		
$u_1(A)$		
$u_1(B)$		
c_1		
	$xl_2(D)$	
	$w_2(D)$	
	$u_2(C)$	
	$u_2(D)$	
	c_2	

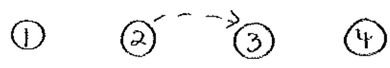
T_1 "sårer" T_3, T_3 må avbryte.

(T_3 slipper å avbryte hvis den i realiteten er ferdig, men det er ikke tilfellet her.)

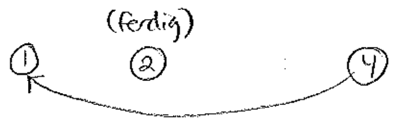
19.2.3

d)

T_1	T_2	T_3	T_4
$sl_1(A)$			
$r_1(A)$			
		$sl_3(B)$	
		$r_3(B)$	
	$xl_2(C)$		
	$w_2(C)$		
	$sl_2(D)$		
	$r_2(D)$		
			$sl_4(E)$
			$r_4(E)$
	$xl_2(B)$ (sårer T_3)		
		a_3	
		$u_3(B)$	
	$xl_2(B)$ (start på nytt)		
	$w_2(B)$		
	$u_2(C)$		
	$u_2(D)$		
	$u_2(B)$		
	c_2		$xl_4(A)$ (avslått)
$xl_1(D)$			
$w_1(D)$			
$u_1(A)$			
$u_1(D)$			
c_1			$xl_4(A)$
			$w_4(A)$
			$u_4(E)$
			$u_4(A)$
			c_4



T_2 "sårer" T_3, T_3 må avbryte
(T_3 slipper å avbryte hvis den i realiteten er ferdig, men det er ikke tilfellet her)



yngre kan vente på eldre

20.5.2

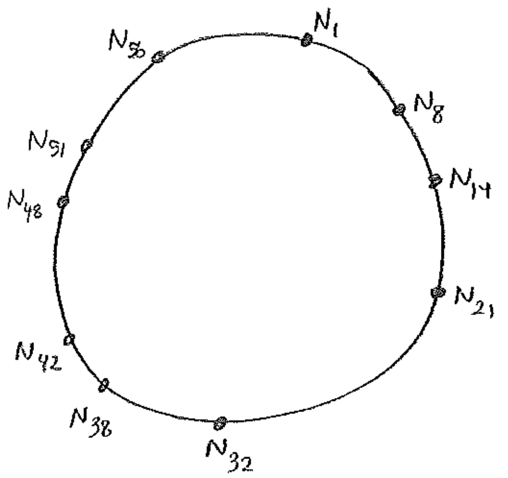
(i, j, M): node i sender M til node j, M ∈ {P, R, D, C, A}

Node 0 er koordinator, global transaksjon med subtransaksjoner på node 1 og 2.

a) Eksempel på sekvens av meldinger der node 1 vil committe og node 2 abortere:

(0, 1, P)
(0, 2, P)
(1, 0, R)
(2, 0, D)
(0, 1, A)
(0, 2, A)

20.7.1



a) (K,V) hvor $h(k) = 35$ havner i N_{38}

b) –"– $= 20$ –"– N_{21}

c) –"– $= 60$ –"– N_1

20.7.2

a) Fingertabell for N_{14}:

avstand	1	2	4	8	16	32
node	N_{21}	N_{21}	N_{21}	N_{32}	N_{32}	N_{48}

b) Fingertabell for N_{51}:

avstand	1	2	4	8	16	32
node	N_{56}	N_{56}	N_{56}	N_1	N_1	N_{21}

20.7.3

a) N_{14} ønsker et par (k,v) hvor $h(k) = 27$:

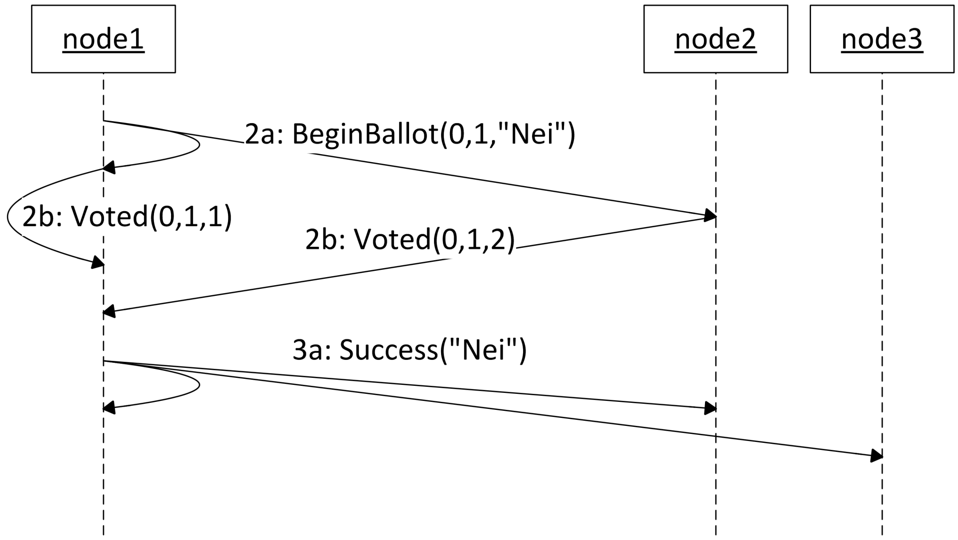
$N_{14} \rightarrow N_{21}$
$N_{21} \rightarrow N_{32}$
$N_{32} \rightarrow N_{14}$

b) N_8 ønsker (k,v) hvor $h(k) = 5$:

Algoritmen som er oppgitt, sier egentlig at $N_8 \rightarrow N_{42} \rightarrow N_1 \rightarrow N_8$, men selvfølgelig vil den implementerte algoritmen først se på forgjenger og etterfølgers id'er og konkludere med at N_8 har verdien selv. Så ingen meldinger trengs.

Løsningsforslag 20.x.1

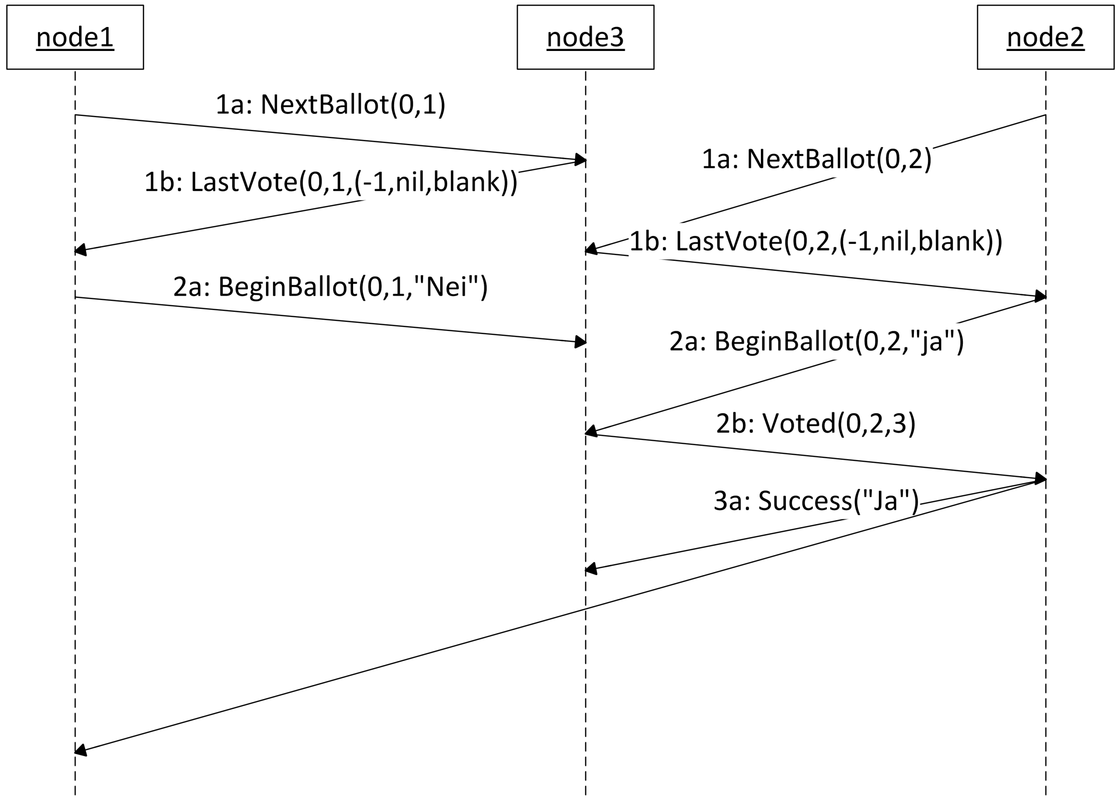
Oppgave a.

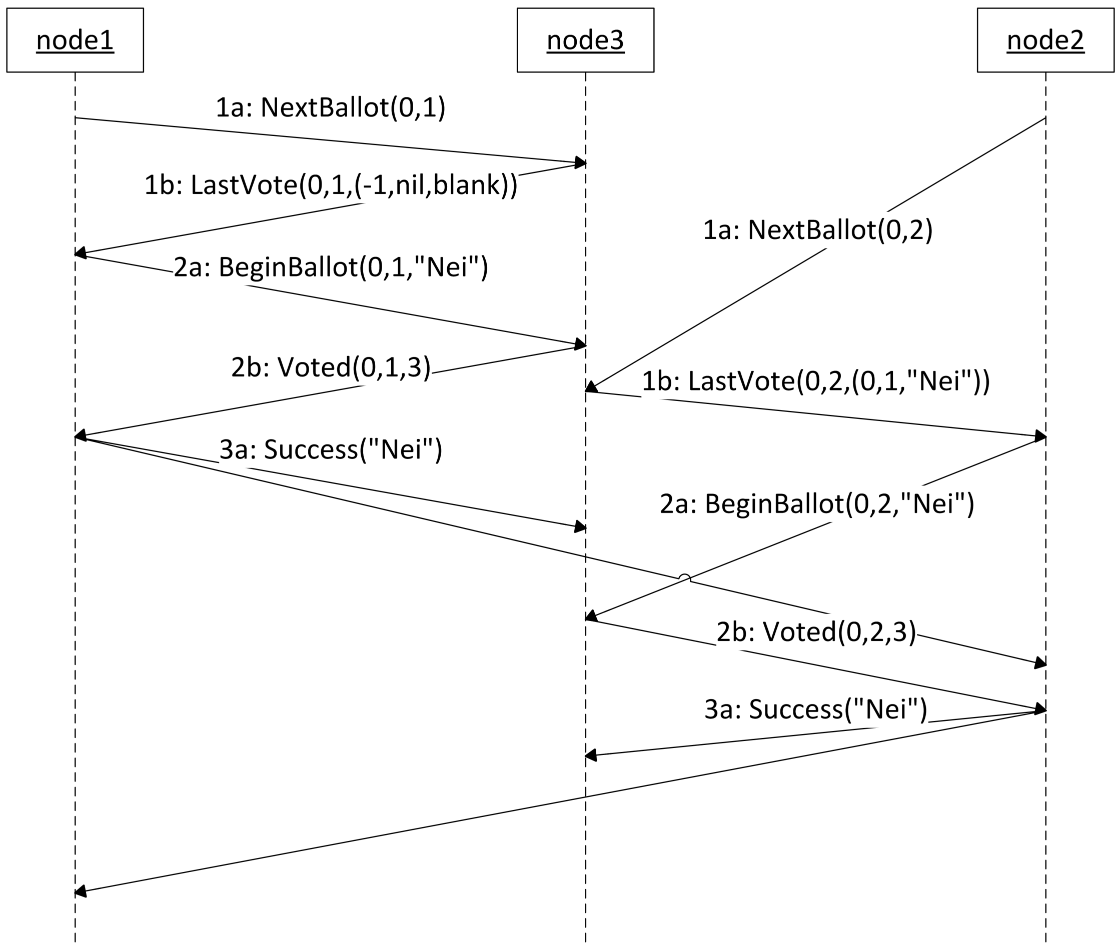


Kommentarer:

- Når en bestemt node er utpekt til å starte protokollen, så starter den protokollen med rundenummer 0 i fase 2a.
- I figuren over har node 1 valgt Q = {1,2} som den majoriteten den velger å benytte i fase 2. Vi har tegnet inn at node 1 sender meldinger også til seg selv, men i praksis vil det jo ikke gå noen meldinger internt i en node - node 1 vil simpelthen justere innholdet av sine lokale variable til det innholdet de skal ha etter hver fase, som om det ble sendt meldinger internt.
- Protokollen sier at det skal sendes fase 2a-meldinger til en majoritet av nodene. Hvis antall noder er 2F+1, betyr dette at det skal sendes melding til minst F+1 noder. Det er ikke noe i veien for å sende meldingen til langt flere enn F+1 noder. Da kan fase 3 påbegynnes straks F+1 svar er kommet inn. Så i det gitte tilfellet med 3 noder kunne det vært like lurt å sende fase 2a-meldinger til både node 2 og 3. Da kan node 1 fortsette med fase 3 i det øyeblikket den får en 2b-melding fra en av de andre.

Oppgave b.

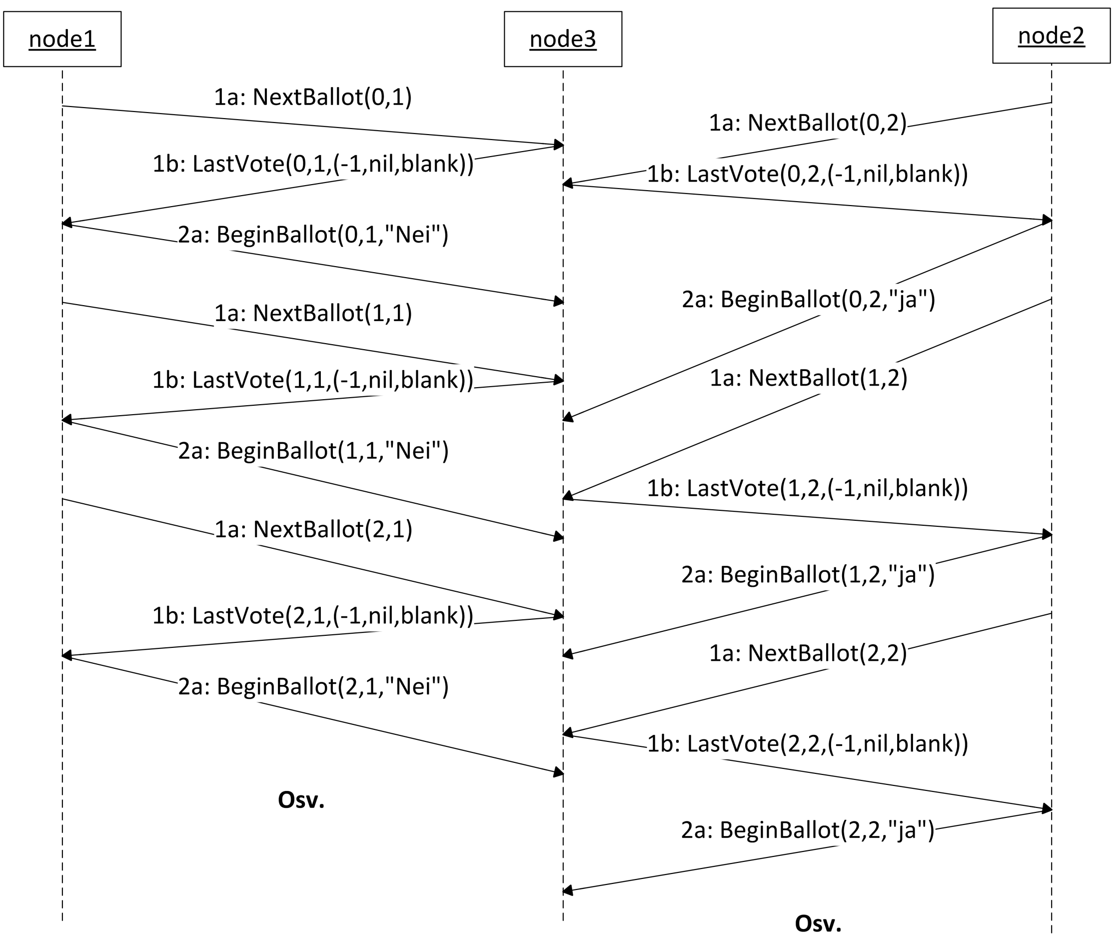

node1
node3
node2
1a: NextBallot(0,1)
1a: NextBallot(0,2)
1b: LastVote(0,1,(-1,nil,blank))
1b: LastVote(0,2,(-1,nil,blank))
2a: BeginBallot(0,1,"Nei")
2a: BeginBallot(0,2,"ja")
2b: Voted(0,2,3)
3a: Success("Ja")


node1
node3
node2
1a: NextBallot(0,1)
1b: LastVote(0,1,(-1,nil,blank))
1a: NextBallot(0,2)
2a: BeginBallot(0,1,"Nei")
2b: Voted(0,1,3)
1b: LastVote(0,2,(0,1,"Nei"))
3a: Success("Nei")
2a: BeginBallot(0,2,"Nei")
2b: Voted(0,2,3)
3a: Success("Nei")

Kommentarer:

- Det er flere mulige forløp; over vises to av dem. I begge benytter node 1 mengden {1,3} og node 2 mengden {2,3} i fase 1 og 2. I begge mottar dessuten node 3 en fase 1a-melding fra node 1 før den mottar en fase 1a-melding fra node 2. Node 1 og 2 starter begge med rundenummer 0. Da er det den noden som har høyest id som har høyest rundenummer, dvs. node 2. (Vi har i figurene utelatt meldinger som går fra en node til noden selv.)
- Tilfelle 1: Node 3 får fase 1a-meldingen fra node 2 før den får fase 2a-meldingen fra node 1. Da vil node 3 følge opp avstemningsrunden til node 2, så ytterligere meldinger fra node 1 vil bli ignorert. Node 3 besvarer 2a-meldingen fra node 2. Når node 2 får 2b-svaret fra node 3, har den et flertall (node 3 og seg selv) med dekretet "Ja", og avrunder med fase 3.
- Tilfelle 2: Node 3 får fase 1a-meldingen fra node 2 etter at den har fått (og besvart) fase 2a-meldingen fra node 1. Da vil node 3 følge opp avstemningsrunden til node 2, men node 1 har alt fått det flertallet den trenger (node 3 og seg selv) med dekretet "Nei" og avrunder med fase 3. Node 2 får via 1b-svaret fra node 3 vite at dekretet "Nei" allerede er blitt fremmet, og fortsetter derfor med å fremme dette dekretet i fase 2. Når node 2 får 2b-svaret fra node 3, avrunder den med dekretet "Nei" i fase 3.

Oppgave c.



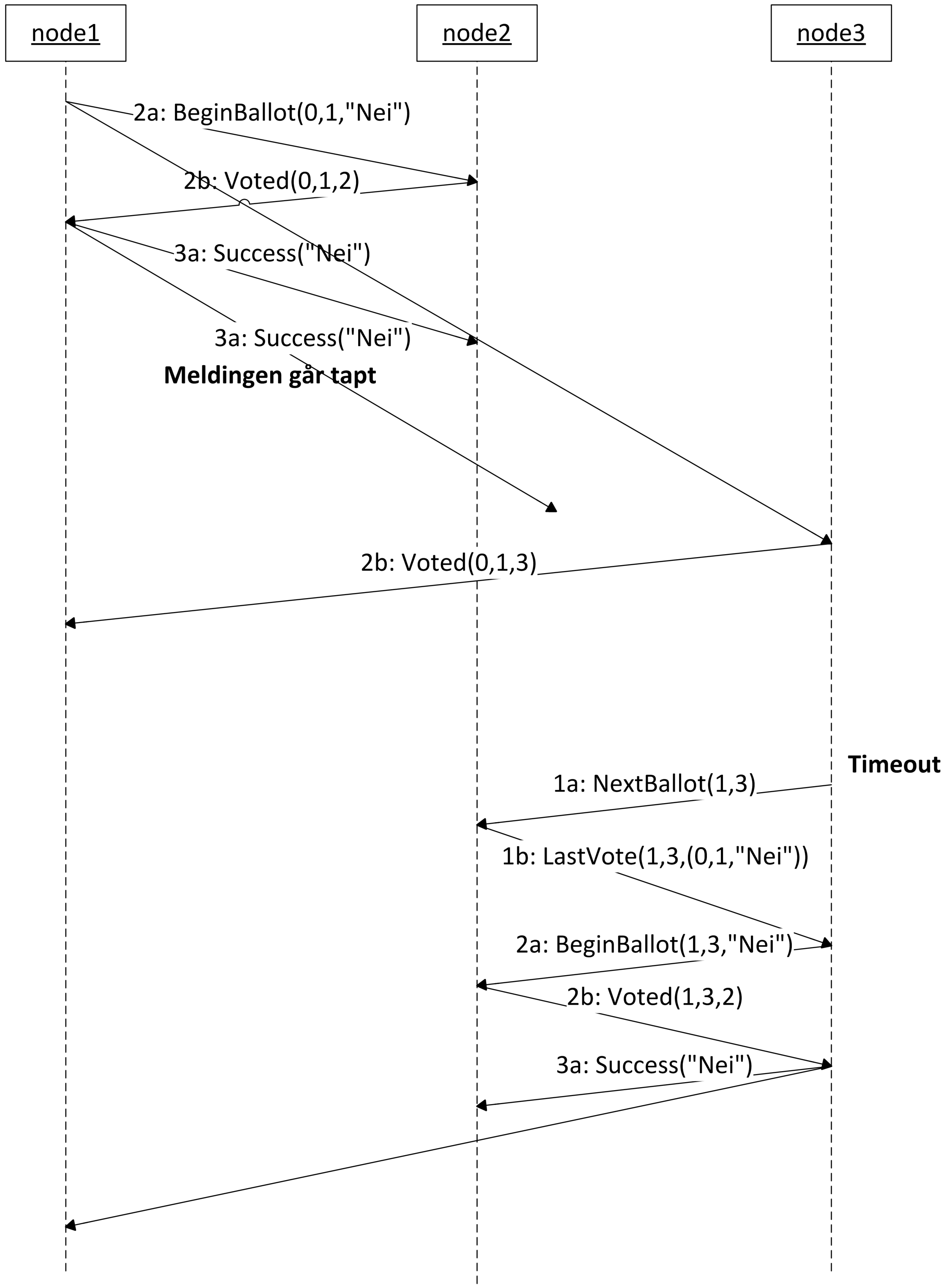
Kommentarer:

- Node 1 og node 2 starter på nye avstemningsrunder når de ikke får svar på sine 2a-meldinger innen en viss tidsfrist. Men denne tidsfristen er satt så kort at det stadig påbegynnes nye runder. Hver gang node 3 ser en ny fase 1a-melding, har den et høyere rundenummer enn det noden har sett til nå, og noden skal da følge opp denne avstemningsrunden og droppe meldinger for andre (lavererangerte) runder. Dermed besvarer ikke node 3 noen av de fase 2a-meldingene den mottar. (Figuren viser ikke meldinger fra en node til noden selv.)
- En node kan også ødelegge for sine egne avstemningsrunder hvis tidsfristene for å få svar er for korte. F.eks. kan node 1 stadig starte på nye avstemnings-runder umiddelbart etter at den har sendt fase 2a-meldingene sine og før den får svar på disse.

Oppgave d.

Nei, terminering har ikke noe å gjøre med hva dekretet inneholder. Forløpet i eksempelet i oppgave 3 er uavhengig av dekretene.

Oppgave e.

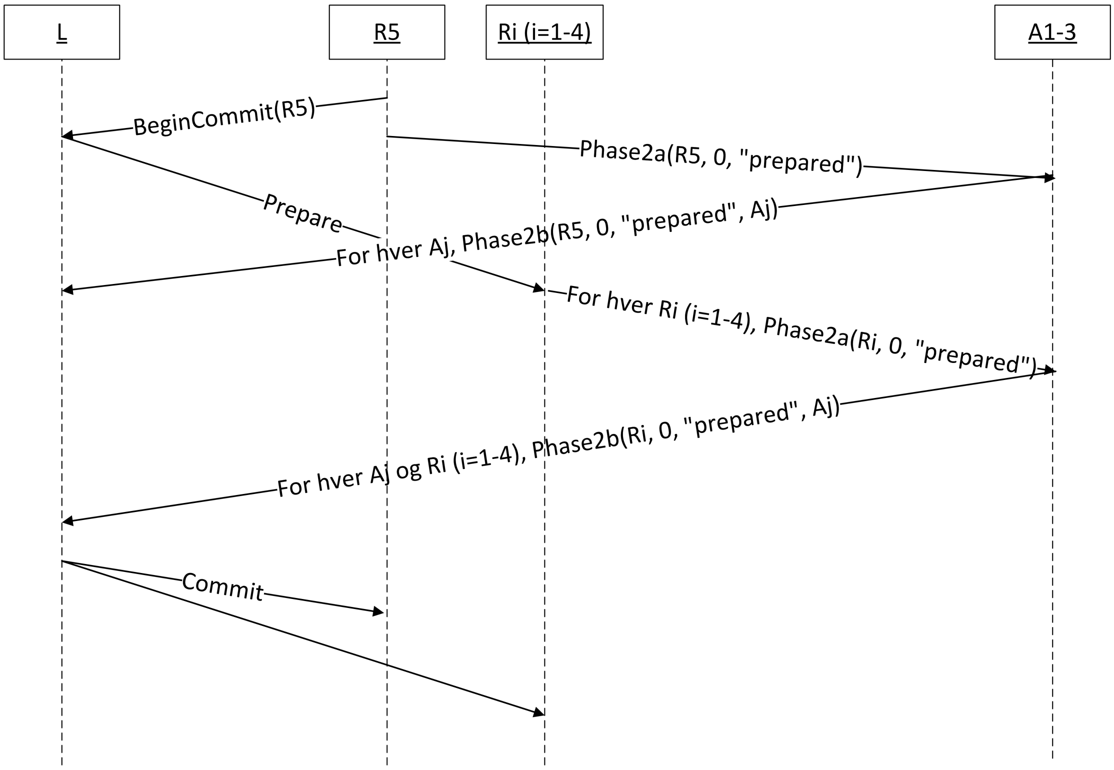

node1
node2
node3
2a: BeginBallot(0,1,"Nei")
2b: Voted(0,1,2)
3a: Success("Nei")
3a: Success("Nei")
Meldingen går tapt
2b: Voted(0,1,3)
Timeout
1a: NextBallot(1,3)
1b: LastVote(1,3,(0,1,"Nei"))
2a: BeginBallot(1,3,"Nei")
2b: Voted(1,3,2)
3a: Success("Nei")

Kommentarer:

- Noder kan starte ytterligere avstemningsrunder for å få vite resultatet av protokollen.
- I oppgaven var en av ideene bak formuleringene at node 3 skjønner at protokollen er påstartet ved at den mottar fase 2a-meldingen fra node 1. Denne meldingen er svært forsinket, men protokollen sier ikke noe om at node 3 må være klar over dette; node 3 besvarer i utgangspunktet alle meldinger på vanlig vis. Imidlertid er det viktig at den på ett eller annet tidspunkt tar initiativ til egne avstemningsrunder for å få fremdrift i protokollen. Til dette kan det benyttes en timeout. I forløpet over velger node 3 ved timeout å benytte {2,3} som majoritetsmengde i fase 1 og 2. Node 2 vil besvare ytterligere avstemningsrunder (med høyere rundenummere) selv om den har fått resultatet av avstemningen. Via svaret på fase 1 får node 3 så vite at dekretet "Nei" har vært oppe til avstemning, og benytter derfor dette dekretet i fase 2. Node 2 følger lojalt opp også denne fasen, og node 3 kan konkludere med at dekretet "Nei" er vedtatt (fordi den har fått svar fra alle i sitt kvorum) og sende fase 3-melding til alle. Det er ikke noe problem for en node å motta flere fase 3-meldinger; de vil alltid inneholde samme dekret. (Figuren viser ikke meldinger fra en node til noden selv.)
- Generelt bør/må protokollen implementeres slik at alle noder som er involvert i protokollen, vet om dette slik at de kan ta et selvstendig ansvar for protokollen ved behov. Nodene kan benytte timeouts for å avgjøre om de skal gjøre noe for å drive protokollen fremover. Hvis f.eks. node 3 ikke hører noe som helst fra noen av de andre nodene innen en viss tidsfrist, bør den selv starte på en avstemningsrunde. Den starter da en avstemningsrunde med rundenummer 1 i fase 1a. (Siden node 1 var forhåndsutpekt til å starte protokollen i dette tilfellet, bør rundenummer 0 reserveres for node 1; de øvrige nodene starter sine avstemningsrunder med 1 eller høyere.) Hvis rundenummer 1 er for lavt i forhold til de rundene andre noder har iverksatt og sagt seg villig til å delta i, vil det ikke komme noen svar. Da vil noden etter en ny timeout forsøke med rundenummer 2, osv.

Løsningsforslag 20.x.2

Oppgave a.



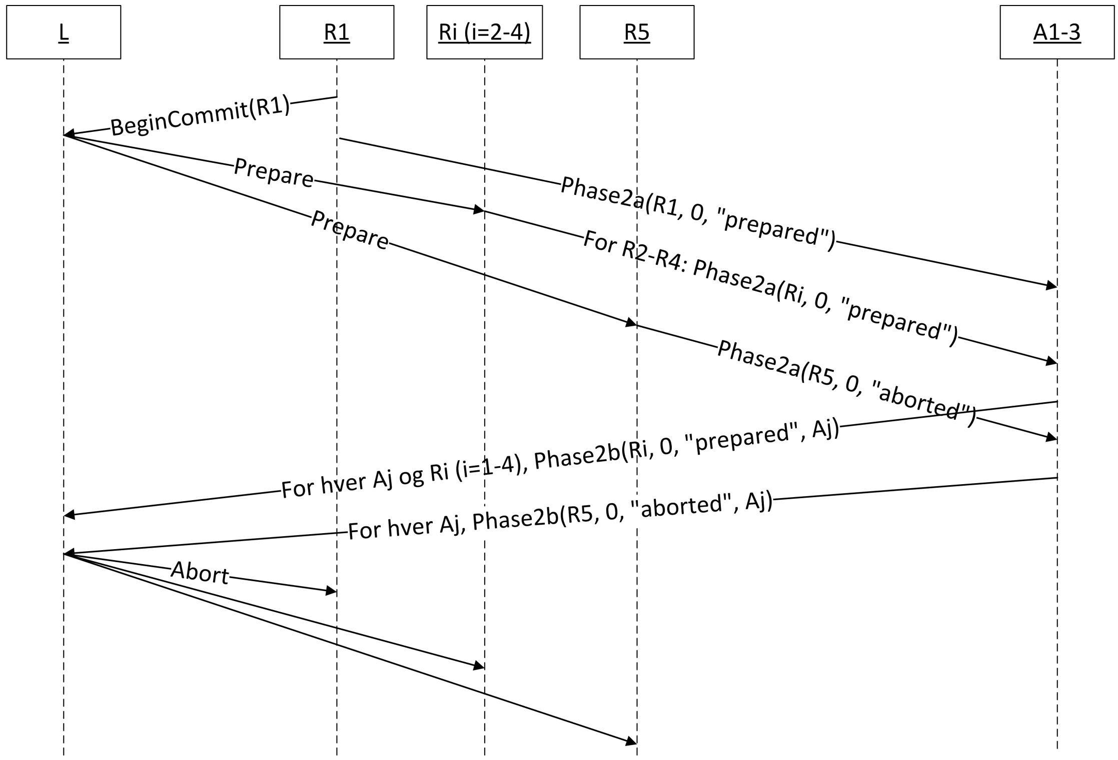
- L er lederprosessen. Ri er ressurshåndterer nr. i, der i=1,2,3,4,5. Aj er akseptorprosess nr. j, der j=1,2,3. Her har vi antatt at det er R5 som blir først ferdig med sin deltransaksjon. Figuren viser R1 til R4 som én felles "stolpe", og det samme for A1-3, for å forenkle fremstillingen litt.

- Når R5 er klar til å committe (og ikke har hørt noe fra de andre ennå), sender den BeginCommit til L og starter sin instans av Paxos consensus ved å sende Phase2a-meldinger med rundenummer 0 og dekretet "prepared" til alle akseptorene. Akseptorene besvarer med Phase2b-meldinger til L.

- Når L mottar BeginCommit-meldingen fra R5, sender den en Prepare-melding til hver av de andre Ri-ene. Når de andre Ri-ene mottar denne meldingen, gjør de på samme måte som R5: Hver starter sin instans med en Phase2a-melding med rundenummer 0 og dekretet "prepared", og akseptorene besvarer disse direkte til lederen.

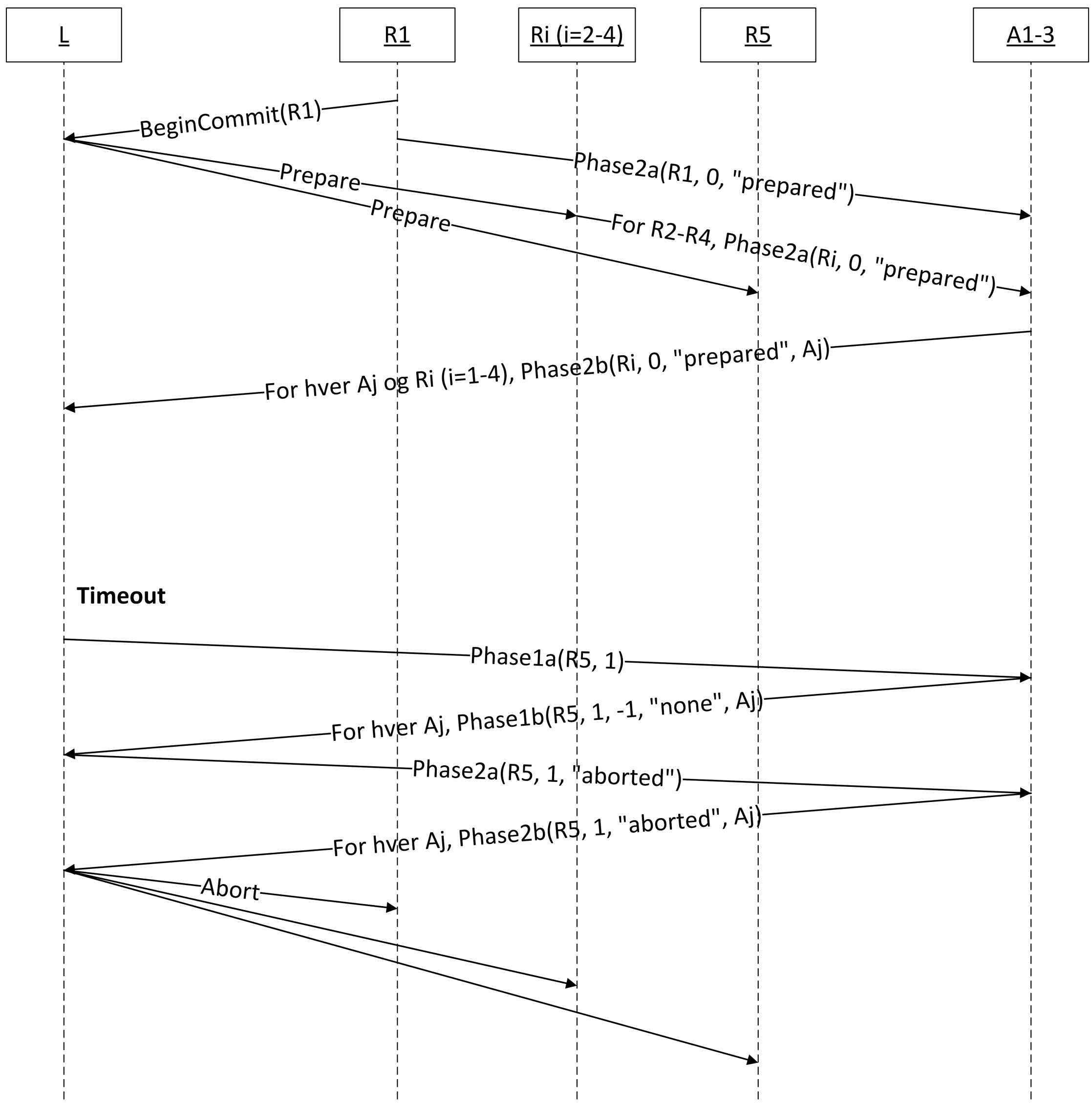
- Når L har mottatt Phase2b-meldinger med dekretet "prepared" fra et flertall av akseptorene for hver Paxos consensus-instans, har den oversikt over det endelige resultatet: Samtlige instanser hadde dekretet "prepared", så lederen sender Commit til alle Ri-ene.

Oppgave b.



- Her har vi antatt at R1 er den som først blir ferdig med sin deltransaksjon og varsler L. Deretter sender L en Prepare-melding til de andre Ri-ene. Alle Ri-ene melder inn resultatet av sin deltransaksjon i form av Phase2a-meldinger til akseptorene: R1-4 med dekretet "prepared", R5 med dekretet "aborted". Hver akseptor besvarer med Phase2b-meldinger til L. L kan sende en Abort-melding til alle Ri-ene i det øyeblikket den har fått inn Phase2b-meldinger med dekretet "aborted" fra en majoritet av akseptorene. (Faktisk kan L sende en Abort-melding straks den mottar den første av Phase2b(R5, 0, "aborted", Aj)-meldingene. Dette gjelder imidlertid bare for runder med rundenummer 0.)

Oppgave c.

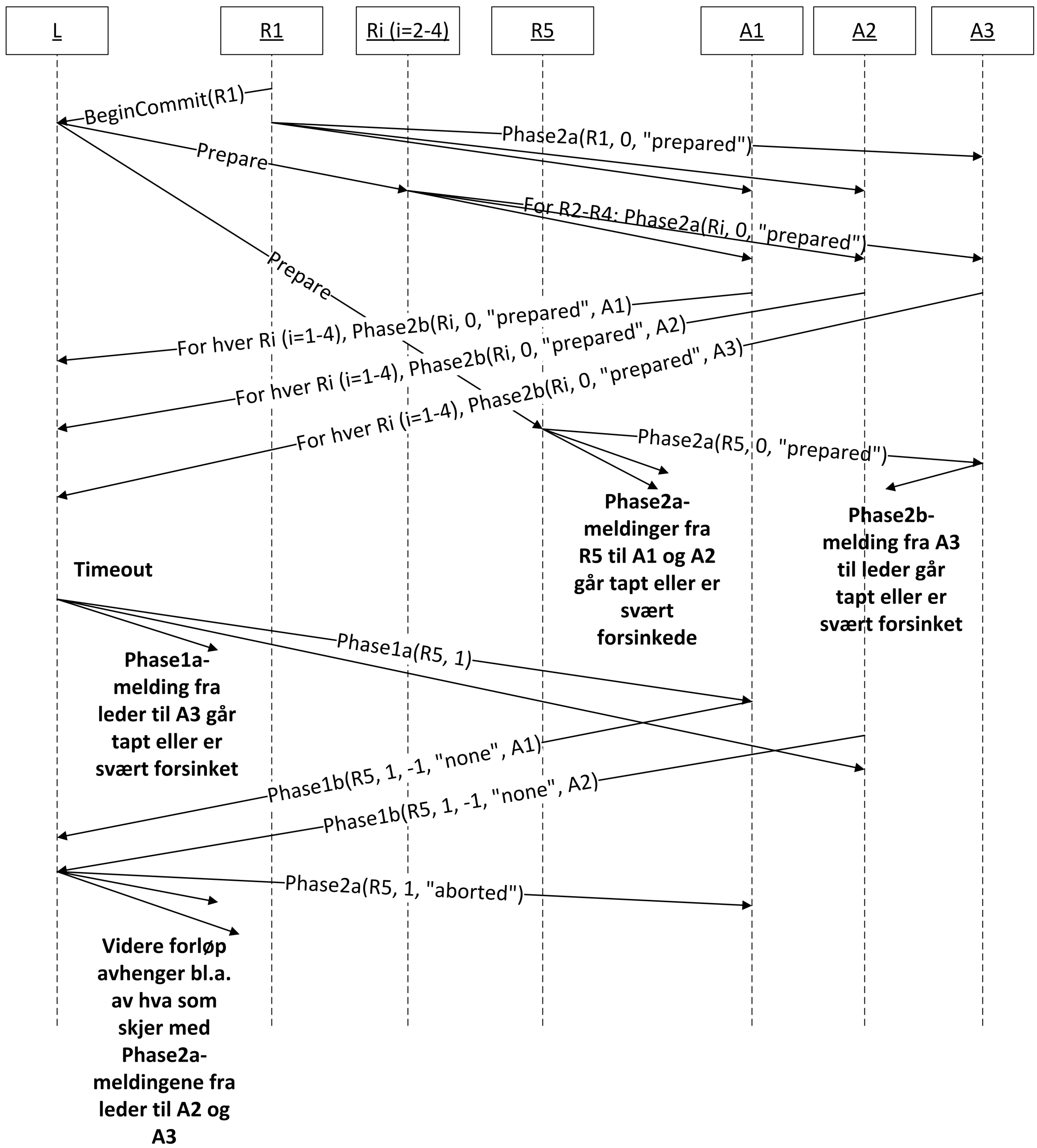


- L sender Prepare til alle, men R5 er treg med å svare (eller nettet til den er nede). Aj-ene sender Phase2b-meldinger for de Phase2a-meldingene de mottar, men det kommer aldri noen Phase2a-meldinger til dem fra R5.
- Etter en timeout påstarter L en ny avstemningsrunde for de instansene der Phase2b-meldinger ikke foreligger; i dette tilfellet gjelder det R5. Nye runder startes som vanlig med Phase1a-meldinger. Når L har mottatt svar fra Aj-ene (eller en majoritet av dem) og får vite at intet dekret er kjent, sender L en Phase2a-melding med dekretet "aborted", som så blir vedtatt.

Oppgave d.

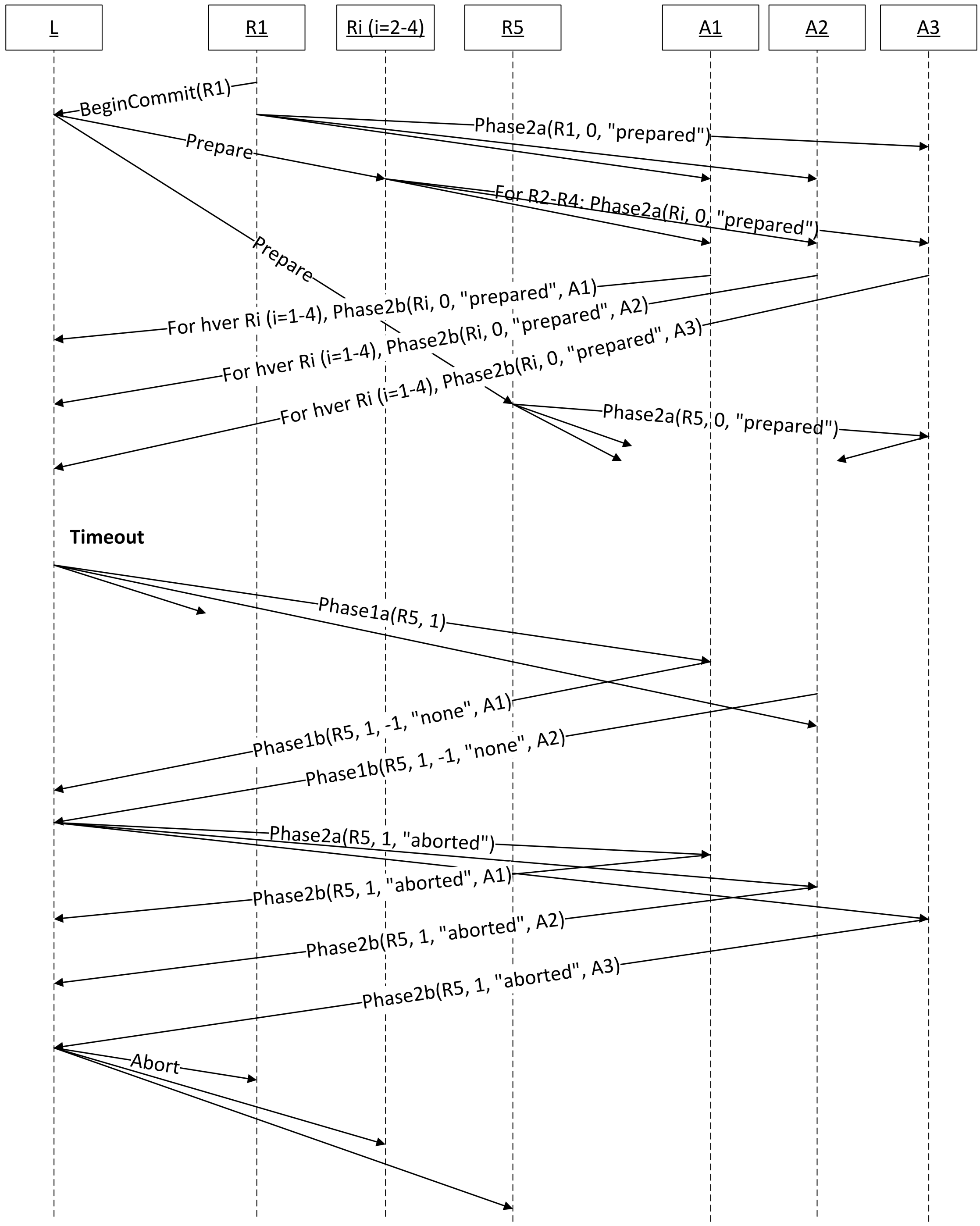
Her vil forløpet være omtrent som i oppgave a, med unntak av A3 som ikke svarer. Det spiller ingen rolle, for så lenge en majoritet av akseptorene svarer, vil protokollen ha progresjon.

Oppgave e.



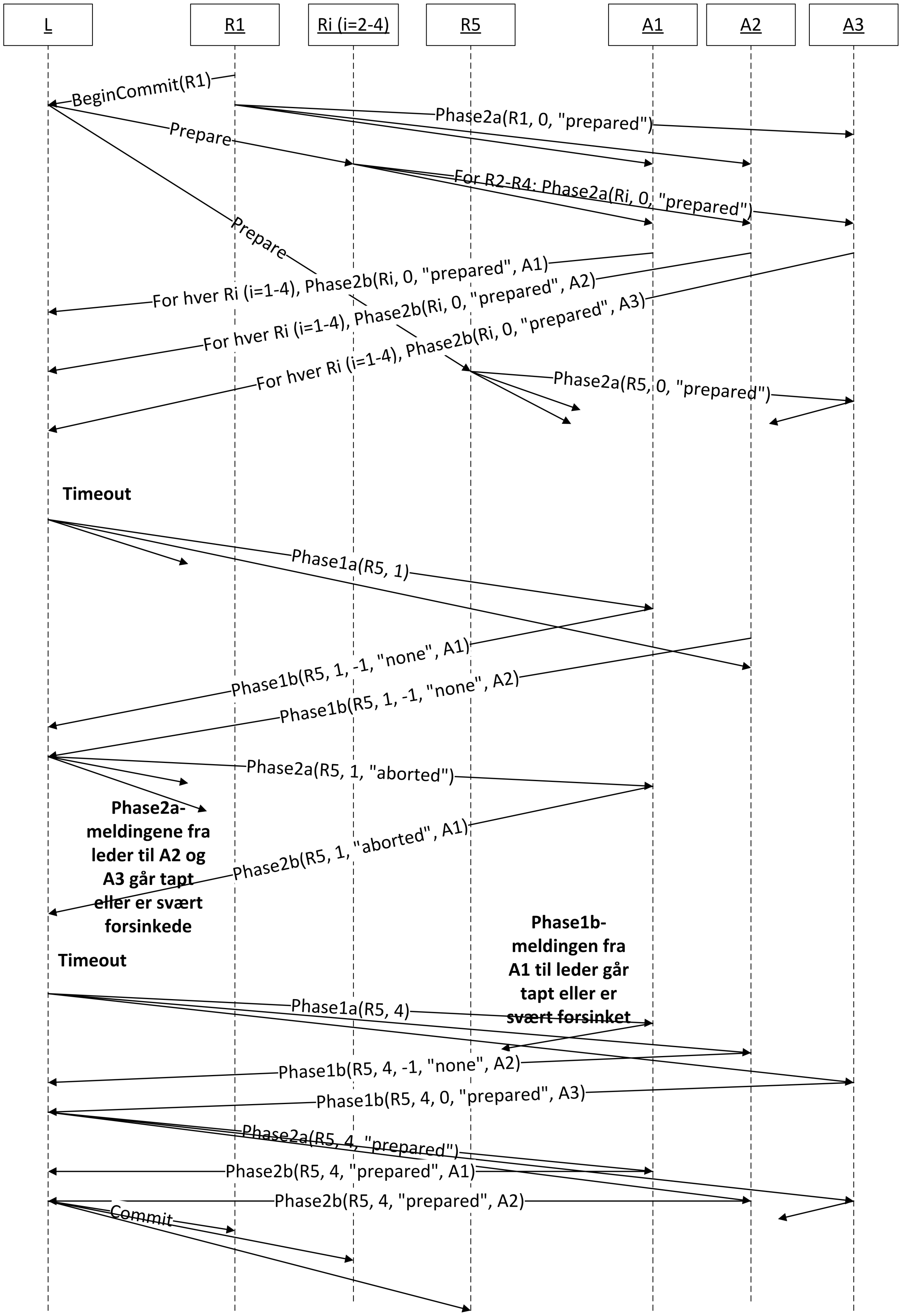
- Protokollen starter som vanlig, men i eksempelet over skjer det deretter noe med nettet mellom R5 og A1-A2 slik at det drøyer før Phase2a-meldingene kommer frem (eller meldingene går tapt). A3 mottar Phase2a-meldingen fra R5, men dens Phase2b-svar til L går tapt.
- I mellomtiden har L timet ut og starter en ny avstemningsrunde for R5 ved å sende Phase1a-meldinger til alle akseptorene. A1 og A2 får disse meldingene, men ikke A3. A1 og A2 returnerer Phase1b-meldinger til L.
- A1 og A2 utgjør en majoritet, så L kan fortsette. Siden A1 og A2 ikke visste om noe forslag til dekret, foreslår L dekretet "aborted" og sender dette til alle akseptorene. A1 mottar denne Phase2a-meldingen. Så nå foreligger to forslag til dekret for R5: "prepared" fra R5 selv (A3 vet om dekretet "prepared"), og "aborted" fra L (A1 vet om dekretet "aborted").

Oppgave f.



- Her mottar og besvarer alle (eller en majoritet av) akseptorene Phase2a-meldingen fra L.
- Når L mottar svarene på den nye avstemningsrunden, vet den at dekretet for R5 er "aborted", så L sender Abort til samtlige ressurshåndterere.

Oppgave g.


L
R1
Ri (i=2-4)
R5
A1
A2
A3
BeginCommit(R1)
Prepare
Phase2a(R1, 0, "prepared")
For R2-R4: Phase2a(Ri, 0, "prepared")
Prepare
For hver Ri (i=1-4), Phase2b(Ri, 0, "prepared", A1)
For hver Ri (i=1-4), Phase2b(Ri, 0, "prepared", A2)
For hver Ri (i=1-4), Phase2b(Ri, 0, "prepared", A3)
Phase2a(R5, 0, "prepared")
Timeout
Phase1a(R5, 1)
Phase1b(R5, 1, -1, "none", A1)
Phase1b(R5, 1, -1, "none", A2)
Phase2a(R5, 1, "aborted")
Phase2a-meldingene fra leder til A2 og A3 går tapt eller er svært forsinkede
Phase2b(R5, 1, "aborted", A1)
Timeout
Phase1b-meldingen fra A1 til leder går tapt eller er svært forsinket
Phase1a(R5, 4)
Phase1b(R5, 4, -1, "none", A2)
Phase1b(R5, 4, 0, "prepared", A3)
Phase2a(R5, 4, "prepared")
Phase2b(R5, 4, "prepared", A1)
Phase2b(R5, 4, "prepared", A2)
Commit

- I dette tilfellet er det bare A1 som mottar Phase2a-meldingen med dekretet "aborted" fra L. Etter en ny timeout prøver L nok en avstemningsrunde (denne gangen med rundenummer 4, siden dette er neste rundenummer som er reservert for node 1). A2 og A3 svarer på denne Phase1a-meldingen. A2 har ikke sett noe dekret hittil, mens A3 har sett dekretet "prepared". Når L får Phase1b-meldingene fra A2 og A3, vil L fremme dekretet "prepared" i Phase2a. Denne meldingen blir besvart av en majoritet, og dermed har alle Paxos consensus-instansene besvart med "prepared", og L kan meddele samtlige ressurshåndterere om at de kan committe.
- Eksempelet er ikke helt i tråd med den beskrivelsen vi har gitt av algoritmen: Vi har forutsatt at A1 er på samme node som L. Det betyr at A1 umulig kan gå glipp av Phase1a(R5, 4)-meldingen fra L, og også at L alltid vil få svar fra A1. Eneste tilfelle der L vil fremme dekretet "prepared" i fase 2, er når minst en av aktuatorene i fase 1 rapporterer dekretet "prepared" og ingen av dem rapporterer det nyere dekretet "aborted". Siden lederen er den eneste som får initiere nye avstemningsrunder og derfor alltid vil motta en Phase1b(R5, .., .., "aborted", A1)-melding fra sin lokale akseptor A1, vil "prepared" derfor bare kunne bli fremmet i fase 2 i ytterligere avstemningsrunder hvis en annen node overtar lederansvaret, f.eks. noden som har A3.

Løsningsforslag

Eksamen Inf3100 2007

I

Salg(salgsID, kundeID, artsnavn, dato, antall)
Hendelse(hendID, salgsID)
Epikrise(hendID, dato, status, info)

a)

```
create table Salg (
    salgsID int primary key,
    kundeID int not null,
    artsnavn varchar(30) not null,
    dato date not null,
    antall int not null,
    unique (kundeID, artsnavn, dato)
);
```

int eller noe annet passende

varchar eller char med en passende parameter

Jeg finner det rimelig at alle attributtene skal ha en verdi.

kandidatnøkkel

```
create table Hendelse (
    hendID int primary key,
    salgsID int not null references Salg(salgsID)
);
```

b)

```
select count(distinct kundeID)
from Salg
where artsnavn = 'Black Molly' and dato >= date '2006-01-01'
                               and dato <= date '2006-12-31';
```

c)

```
select s.salgsID, s.artsnavn, count(h.hendID) as ant
from Salg s, Hendelse h
where s.salgsID = h.salgsID and
      s.dato >= '2006-01-01' and
      s.dato <= '2006-12-31'
group by s.salgsID, s.artsnavn
having count(h.hendID) > 2;
```

Tar med artsnavn for enkelt å kunne få det med i select-klausulen.

Alternativt kan man ta group by s.salgsID og ha select s.salgsID, max(s.artsnavn), count(..)

Inf3100 2007

I d)

Idé: Tell opp hvor mange hendelser det er for hver salgsID og sammenlikn med antall hendelser for salgsID-en der minst én av epikrisene har status 'diagnose'.

```
select s.salgsID, s.dato
from Salg s
where
    s.dato >= date '2007-01-01' and
    s.dato <= date '2007-01-31' and
    ( select count(*)
      from Hendelse h1
      where h1.salgsID = s.salgsID)
    =
    (select count(distinct h.hendID)
     from Hendelse h2, Epikrise e
     where h2.salgsID = s.salgsID and
           h2.hendID = e.hendID and
           e.status = 'diagnose')
order by s.dato;
```

antall hendelser knyttet til dette salget (s.salgsID)

antall forskjellige hendelser hvor det er minst én epikrise med status diagnose (må ha distinct i count fordi vi ellers kan komme til å telle en hendID flere ganger)

(Det er mange måter å besvare denne oppgaven på, f.eks. ved bruk av exists ...)

e) $\sigma_{ant>2}(\gamma_{salgsID, ortsnavn, count(hendID) \to ant}(\sigma_{2006\text{-}01\text{-}01 \le dato \le 2006\text{-}12\text{-}31}(Salg \bowtie Hendelse)))$

Inf3100 2007

II

Adresse(by, postnr, gate)

by, gate $\rightarrow$ postnr
postnr $\rightarrow$ by

a) gate forekommer ikke i noen høyreside og må derfor være med i alle kandidatnøkler.

$\text{gate}^+ = \text{gate}$

$(\text{by, gate})^+ = \text{by, gate, postnr}$

$(\text{gate, postnr})^+ = \text{gate, postnr, by}$

Kandidatnøklene er derfor (by, gate) og (gate, postnr).

b) I by, gate $\rightarrow$ postnr er venstresiden en kandidatnøkkel (og derfor supernøkkel), altså er denne FDen på BCNF.

I postnr $\rightarrow$ by er venstresiden ikke supernøkkel, men høyresiden er et nøkkelattributt, så den er på 3NF, men bryter BCNF.

c) Adresse(by, postnr, gate)

dekomponerer i henhold til bruddet i postnr $\rightarrow$ by

R1(postnr, by) R2(postnr, gate)

Fortsatt gjelder FDen by, gate $\rightarrow$ postnr, men den går på tvers av de to nye relasjonene, så dekomposisjonen er ikke FD-bevarende.

d) I dette tilfellet er det rimelig å anta at det sjelden er oppdateringer av Adresse, mens det ved queries svært ofte ("alltid"?) ville vært behov for å joine R1 og R2 ~~for~~ å få ut "hele" adressen. Da kan det lønne seg å beholde tabellen Adresse: Ved eventuelle oppdateringer må det foretas en sjekk internt i tabellen for å se at FDene bevares, mens det ved queries kan hentes ut fullstendig adresseinformasjon fra denne ene tabellen.

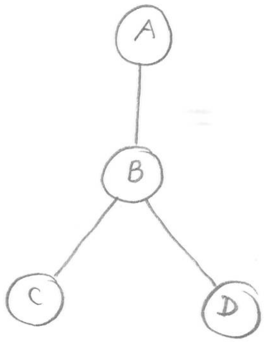
Inf 3100 2007

III

a) Treprotokollen benyttes når den underliggende datastrukturen er et tre (dataelementene som inngår i transaksjonene, er organisert i et tre).

b) Første lås settes på en vilkårlig node i treet.
Deretter kan transaksjonen bare låse videre nedover i treet, på følgende måte: For å ta lås på en node, må man ha lås på foreldrenoden. Låser kan slippes når man vil, men hvis en node har hatt en lås og deretter sluppet den igjen, får den ikke ta låsen på nytt (selv om foreldrenoden fortsatt har sin lås).

c)



T_1: $l_1(A); r_1(A); l_1(B); u_1(A); r_1(B); l_1(C); u_1(B); r_1(C); w_1(C); u_1(C)$

T_2: $l_2(A); r_2(A); l_2(B); u_2(A); r_2(B); l_2(D); r_2(D); w_2(D); u_2(D); w_2(B); u_2(B)$

Planen blir:

$l_2(A); r_2(A); l_2(B); u_2(A); r_2(B); l_1(A); r_1(A); l_2(D); r_2(D); l_1(B); u_1(A); r_1(B); l_1(C); u_1(B); r_1(C); w_2(D); u_2(D); w_1(C); u_1(C); w_2(B); u_2(B)$

Inf 3100 2007

III c) (forts.)

	T_1	T_2
		$l_2(A)$
		$r_2(A)$
		$l_2(B)$
		$u_2(A)$
		$r_2(B)$
	$l_1(A)$	
	$r_1(A)$	
		$l_2(D)$
		$r_2(D)$
vent	$l_1(B)$	
		$w_2(D)$
		$u_2(D)$
		$w_2(B)$
		$u_2(B)$
fortsett	$l_1(B)$	
	$u_1(A)$	
	$r_1(B)$	
	$l_1(C)$	
	$u_1(B)$	
	$r_1(C)$	
	$w_1(C)$	
	$u_1(C)$	

Inf 3100 2007

IV

a) TPMMS: Formålet er sortering når en relasjon er for stor til å få plass i primærminnet.

Fase 1: Sorterer så store biter som man kan få plass til i minnet av gangen, og skriver hver delsortering til disk.

Etter fase 1 har vi grupper av blokker som er fullstendig sortert hver for seg. /"sublister"

Fase 2: Henter inn en blokk fra hver subliste til minnet. Deretter fletter man sammen postene fra disse blokkene: Nye blokker fra hver subliste hentes inn ved behov, fulle (ferdig sortete) blokker skyfles videre til den/de prosessene som bestilte sorteringen.

Kostnad: Hver blokk leses til minnet og skrives tilbake til disk i fase 1, dvs. 2 I/O pr. blokk.

I fase 2 leses hver blokk til minnet én gang. Hva de ferdig-sortete blokkene deretter skal brukes til, varierer med situasjonen: kanskje er dette (TPMMS) del av en større prosess, i såfall går disse blokkene videre til en ny algoritme (pipelining el.l.), eller de skal simpelthen lagres på disk. Uansett regner vi ikke denne siste I/O-en som en del av algoritmens kostnad.

Totalt: 3 I/O pr. blokk.

b) Det som avgjør bruk av TPMMS kontra andre sorteringsalgoritmer, er størrelsen på det som skal sorteres:

Hvis hele relasjonen kan rommes i minnet samtidig, brukes quicksort eller noe annet tilsvarende.

Hvis ikke, brukes TPMMS eller en annen algoritme som er beregnet brukt i denne situasjonen. Hvis antall dellister under TPMMS er så høyt at ikke alle listene kan representeres ved én blokk i minnet samtidig, må det flere faser til – da brukes trefase MMS etc.

Løsningsforslag

Ekstraoppgaver

2007.x.1 - Ekstraoppgave til eksamen 2007

A. La oss si at vi slår sammen Hendelse og Epikrise i én tabell.

(i) Hvordan vil tabellen se ut?

(ii) Hvilke FDer gjelder?

(iii) Hvilke kandidatnøkler har den?

(iv) Hvilken normalform er den på?

(i) HE(hendID, salgsID, dato, status, info)

(ii) hendID → salgsID (fra primærnøkkelen i Hendelse)

hendID, dato → status, info (-"- i Epikrise)

(iii) hendID og dato må være med i alle kandidatnøkler fordi de ikke forekommer i noen høyreside. Siden

(hendID, dato)$^+$ = hendID, dato, salgsID, status, info

er (hendID, dato) kandidatnøkkel (den eneste slike).

(iv) hendID → salgsID : hendID er ikke supernøkkel, så bryter BCNF.
Salgs ID er ikke nøkkelattributt, så bryter 3NF.
hendID er ekte delmengde av kandidatnøkkelen, så bryter 2NF.

Totalt på 1NF, bryter 2NF

Behøver da strengt tatt ikke vurdere den siste FDen, men gjør det likevel:

hendID, dato → status : Venstresiden er supernøkkel, så er på BCNF.

hendID, dato → info : Samme her.

Totalt blir HE på 1NF, men bryter 2NF.

Ekstraoppgaver

B. Vi skal se på en liten utvidelse av akvariebutikkdatabasen. Butikken har flere rabattordninger - bl.a. en for de som har handlet for mer enn en viss sum foregående år, og en for de som er medlem i Pirajafiskens venner. En kunde kan være med i flere rabattordninger, men kan bare bruke én rabattordning i forbindelse med hvert kjøp.

Til å håndtere dette, har databasen tabellen

Rabatt(kundeID, rabattnavn, salgsID, prosent)

der rabattnavn er navnet på en rabatt kunden har, salgsID er et salg der kunden har benyttet denne rabatten og prosent er hvor stor rabatt denne kunden har for akkurat denne rabattypen.

(i) Hvilke FDer gjelder?

(ii) Hvilke kandidatnøkler har Rabatt?

(iii) Dekomponer Rabatt til BCNF

Ekstraoppgaver

Løsningsforslag B.

(i) Fra før har vi at

salgsID → kundeID

Dette bør jo gjelde i den nye tabellen også.

Siden det er maksimalt én rabattordning som kan brukes pr. kjøp, har vi dessuten at

salgsID → rabattnavn

(Men vi har ikke kundeID → rabattnavn, for en kunde skulle kunne ha flere rabatter.)

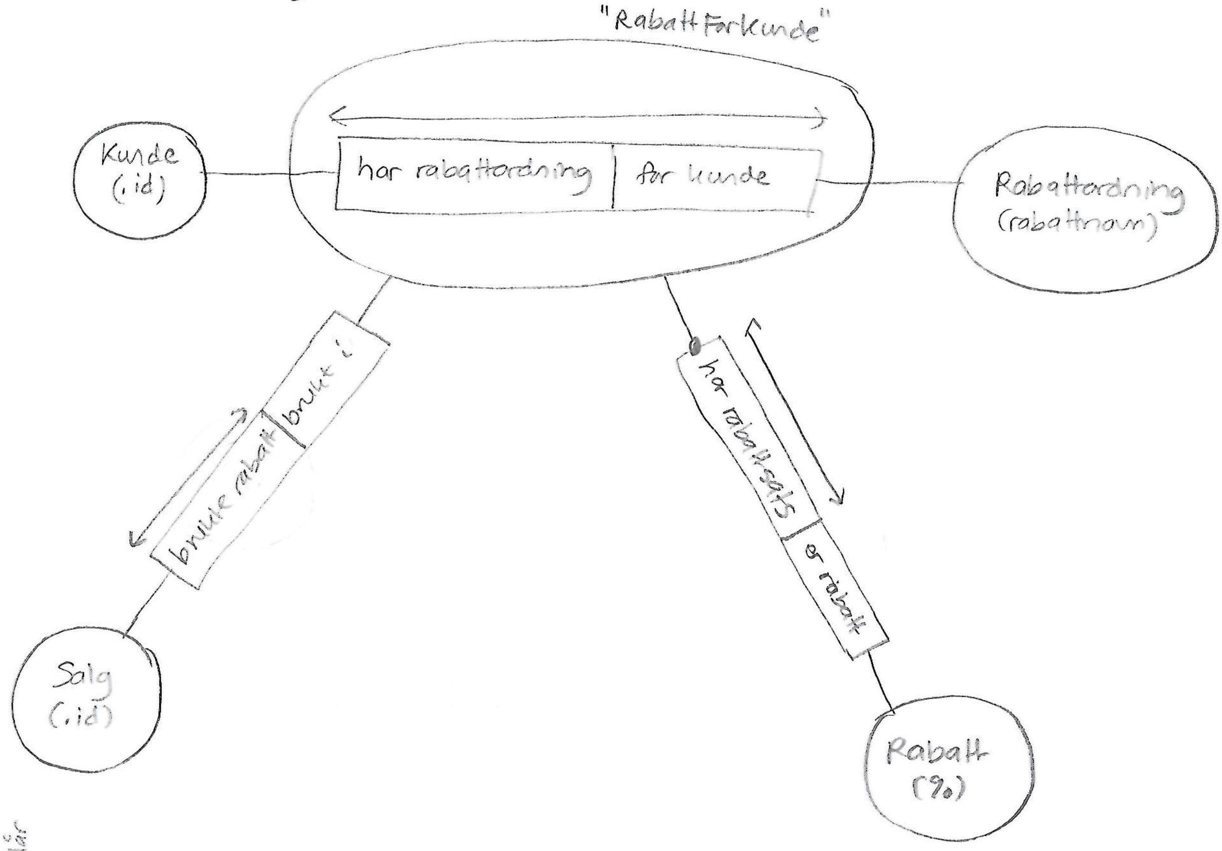
Dessuten avhenger størrelsen på rabatten på hvilket rabattprogram og hvilken kunde det er snakk om, så

kundeID, rabattnavn → prosent

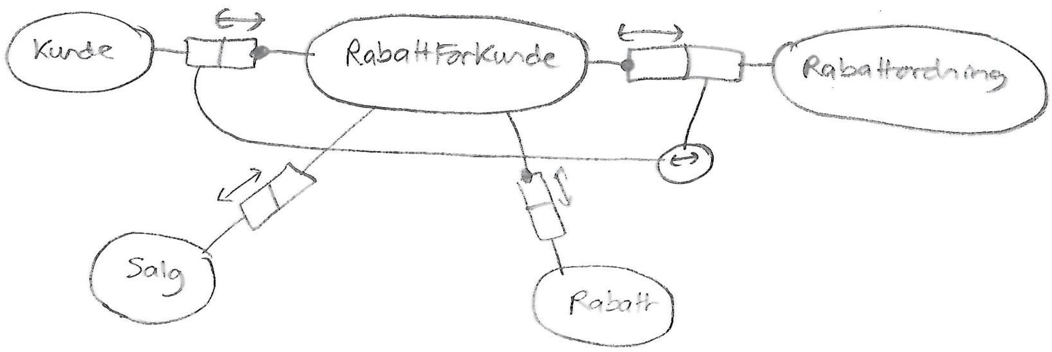
(Men ikke rabattnavn → prosent, for da kan man ikke ha forskjellige prosenter for forskjellige kunder innen en rabattype.)

Til B (i)

De som har Inf1300, vil kunne ha nytte av å modellere og gruppere, da vil FDene fremkomme nærmest av seg selv hvis modelleringen er riktig:



eller, skrevet fullt ut (droppet rollenavn og referansemåter):



Gruppert:

RabattForkunde(<u>kundeID, rabattnavn</u>, prosent)

RabattSalg(<u>salgsID</u>, kundeID, rabattnavn)(*)

Dermed:

kundeID, rabattnavn → prosent

salgsID → kundeID, rabattnavn

(*) Egentlig skulle RabattSalg vært slått sammen med den gamle Salg, og vi skulle i tillegg ha angitt ekvivalente stier mellom Salg – Kunde og Salg – RabattForKunde – Kunde slik at de to kundeID'ene fra gammel og ny versjon blitt slått sammen til én under grupperingen. Men for å tydeliggjøre denne delen av oppgaven, innfører jeg i stedet en ny relasjon RabattSalg. (Og underslår ekvivalente stier.)

B (forts.)

(ii) salgsID må være med i alle kandidatnøkler fordi den ikke er i noen høyreside.

salgsID$^+$ = salgsID, kundeID, rabattnavn, prosent

Så (salgsID) er kandidatnøkkel (eneste slike).

(iii) Må først vurdere hvilke brudd vi har:

salgsID → kundeID er på BCNF; venstresiden er kandidatnøkkel og derfor supernøkkel

salgsID → rabattnavn —"—

kundeID, rabattnavn → prosent er på 2NF, men bryter 3NF: venstresiden er ikke supernøkkel, høyresiden er ikke nøkkelattributt, venstresiden er ikke en ekte delmengde av en kandidatnøkkel.

Rabatt(kundeID, rabattnavn, salgsID, prosent)

splitter etter kundeID, rabattnavn → prosent

RabattForKunde(kundeID, rabattnavn, prosent)

FDer i denne:

kundeID, rabattnavn → prosent

Er på BCNF, lokal kandidatnøkkel er (kundeID, rabattnavn)

RabattSalg(kundeId, rabattnavn, salgsId)

FDer i denne:

salgsId → kundeId, rabattnavn

Er på BCNF, lokal kandidatnøkkel er (salgsID).