NTNU
Det skapende universitet

Institutt for datateknikk og informasjonsvitenskap

Løsningsskisse til Eksamensoppgave i TDT4145 Datamodellering og databasesystemer

Eksamensdato: 23. mai 2013

Eksamenstid (fra-til): 09:00 - 13:00

Hjelpemiddelkode/Tillatte hjelpemidler:

D – Ingen trykte eller håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.

Oppgave 1 – Datamodeller (20 %)

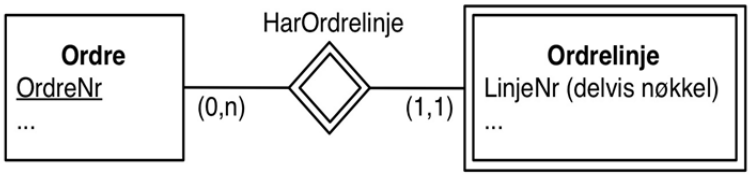
a) Anta en entitetsklasse (A) med disse egenskapene:

- Det finnes ikke noe attributt eller mengde av attributter som kan tjene som entydig identifikator for entitetsklassen.
- Entitetsklassen A har en relasjonsklasse (R) til en annen entitetsklasse (B). Klassen A er eksistensavhengig av relasjonsklassen R og kan ha denne typen relasjon til maksimalt en entitet i entitetsklassen B. A har altså (1,1)-kardinalitet i forhold til relasjonsklassen R.

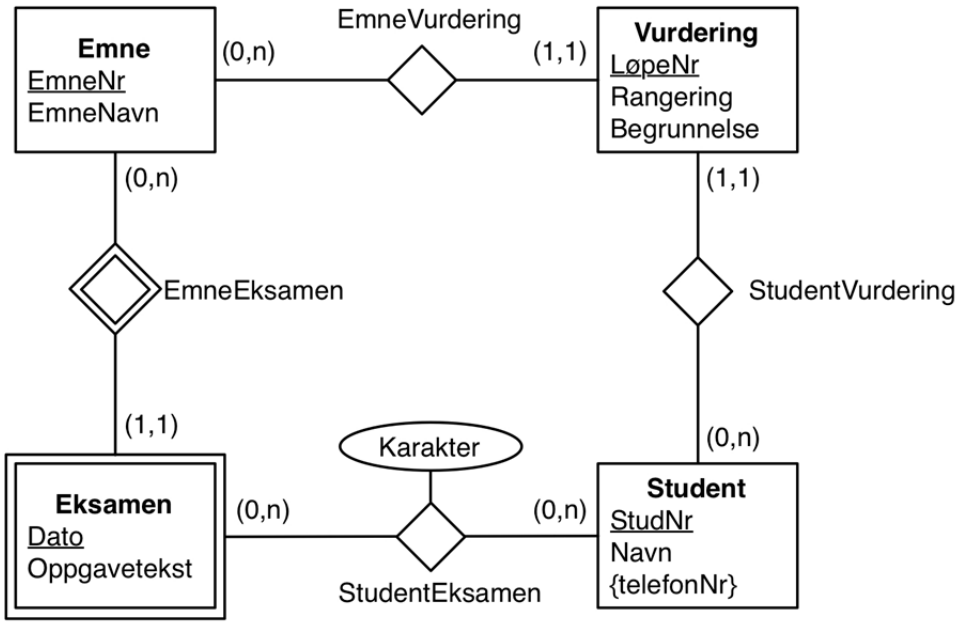
Dersom det finnes et attributt eller en mengde attributter i A som alltid er entydig blant de A-entitetene som har en R-relasjon til samme B-entitet, kaller vi dette en *delvis nøkkel*. I slike tilfeller kan vi modellere A som en svak entitetsklasse. Relasjonsklassen R fungerer da som en *identifiserende relasjonsklasse* for entitetsklassen A.

En globalt entydig identifikator for A vil bestå av attributtene i den delvise nøkkelen pluss attributtene i identifikatoren til B. Alternativet til å modellere A som en svak entitetsklasse, vil være å legge til et ekstra attributt i A, ofte kalt en surrogatnøkkel, som vil fungere som identifikator entitetsklassen.

I figuren under har vi vist et eksempel på en svak entitetsklasse. Her vil LinjeNr være delvis nøkkel for Ordrelinje og HarOrdrelinje være identifiserende relasjonsklasse. I dette eksempelet anser vi at det er mer hensiktsmessig å modellere Ordrelinje som en svak entitetsklasse, enn å innføre en globalt entydig nøkkelattributt i entitetsklassen.



b) ER-diagrammet under viser en mulig ER-modell.



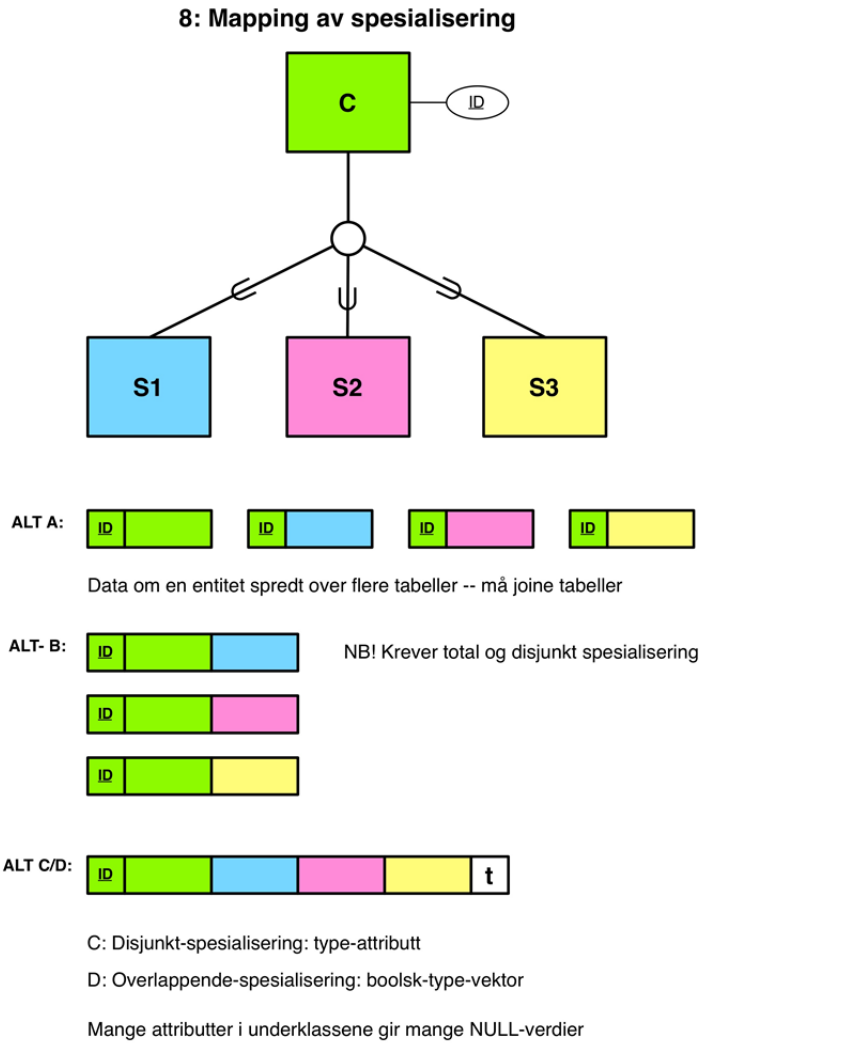
c) Figuren på neste side viser de 4 alternativene vi har når vi skal oversette en spesialisering av en entitetsklasse til et relasjonsdatabaseskjema.

Alternativ A brukes når vi ønsker å beholde både superklassen og sub-klassene. Løsningen kan brukes for både delvis og totale spesialiseringer og for disjunkte og overlappende spesialiseringer. Denne løsningen gir lite NULL-verdier i databasen, som er en fordel, men krever join for å finne alle data for en entitet.

Alternativ B brukes når vi velger å beholde bare subklassene. Løsningen gir få NULL-verdier og trenger ikke join for å finne alle data om en entitet, men kan bare brukes når spesialiseringen er både total og disjunkt.

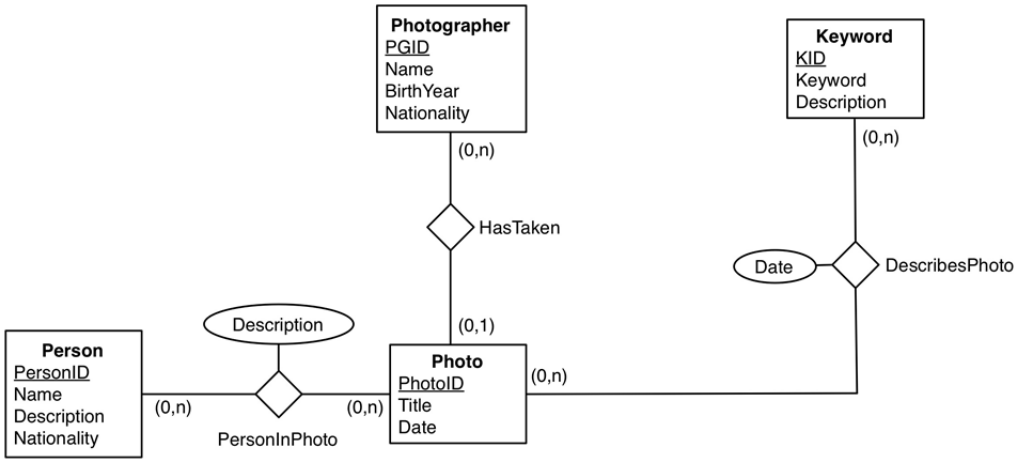
Alternativ C og D brukes når vi velger å beholde bare superklassen. Løsningen kan gi mange NULL-verdier, trenger ikke join for å finne alle data om en entitet, kan brukes for både delvise og totale spesialiseringer og fungerer for disjunkte (C) og overlappende (D) spesialiseringer. Løsningen gir det minste antall tabeller.

Generelt vil vi ønske å beholde de klassene som gjør at vi får en mest mulig hensiktsmessig relasjonsdatabase-modell for den virkeligheten databasen skal modellere.



Oppgave 2 – Relasjonsalgebra og SQL (20 %)

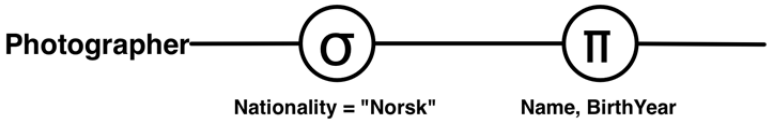
a) ER-diagrammet under viser en ER-modell for relasjonsskjemaet.



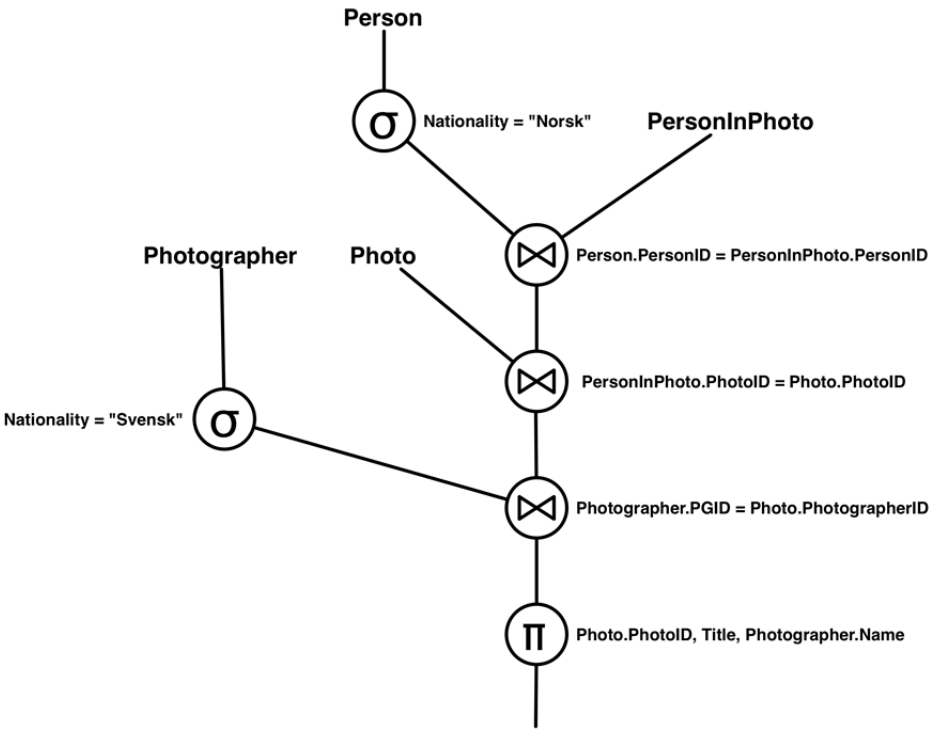
Forutsetninger:

- Photo trenger ikke å ha registrert en Photographer.
- Photo trenger ikke å være beskrevet av noe Keyword.
- Photo trenger ikke å vise noen (kjent) Person.
- Photographer trenger ikke å ha tatt noe Photo.
- Person trenger ikke være avbildet i noe Photo.

b) Relasjonsalgebra:



c) Relasjonsalgebra:



d)
```
SELECT PhotoID, Title
FROM Photo
WHERE PhotoID NOT IN (SELECT PhotoID
                      FROM KeywordPhoto)
```

e)
```
SELECT DISTINCT P.Name
FROM Person AS P, PersonInPhoto AS PIP, KeywordPhoto AS KP,
     Keyword AS K
WHERE P.PersonID = PIP.PersonID
  AND PIP.PhotoID = KP.PhotoID
  AND KP.KID = K.KID
  AND K.Keyword = ”terrengsykling”
```

f)
```
SELECT P.PersonID, P.Name, count(PIP.PhotoID)
FROM Person AS P, PersonInPhoto AS PIP
WHERE P.PersonID = PIP.PersonID
GROUP BY P.PersonID, P.Name
HAVING count(PIP.PhotoID) > 100
ORDER BY count(PIP.PhotoID) DESC
```

Oppgave 3 – Teori (20 %)

a) For eksempel F = {A -> CD}.

b) Mengden attributter må være en av kandidatnøklene for tabellen (= en minimal supernøkkel) og valgt blant disse som primærnøkkel for tabellen.

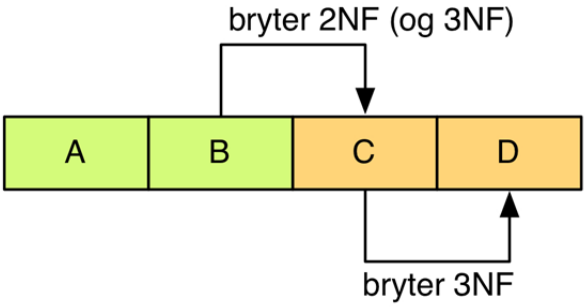
c) Dekomponeringen er tapsløs hvis de resulterende tabellenes felles attributter, i dette tilfellet BC, er en supernøkkel for en eller begge tabellene. En av mange mulige løsninger er F = {BC -> D}.

d) Dekomponeringer vurderes etter fire forhold:

- **Attributtbevaring:** $R = R_1 \cup R_2 \cup R_3$ så det er ivaretatt.
- **FD-bevaring:** C -> D ivaretas i R_2, D -> E ivaretas i R_3 og E -> F ivaretas i R_3. Det betyr at vi har FD-bevaring.
- **Tapsløst-join:** R_1 og R_2 joiner tapsløst fordi det felles attributtet C er (super-)nøkkel i R_2. Resultatet joiner tapsløst med R_3 til utgangspunktet (R) siden det felles attributtet (D) er en nøkkel i R_3. Vi har med andre ord oppfylt kravene til tapsløst-join-egenskapen.
- **Normalform:** R_1 er på BCNF siden $G_1 = \emptyset$. R_2 har G_2 = { C -> D } og er på BCNF siden C er en supernøkkel i R_2. R_3 har G_3 = { D -> E, E -> F } og oppfyller kravene til 2NF siden det ikke finnes noen delvis avhengigheter av nøkkelen D. Tabellen er *ikke* på 3NF på grunn av E -> F, der E ikke er en supernøkkel og F ikke er et nøkkelattributt.

Dekomponeringen har god kvalitet med unntak av tabellen R_3 som burde vært splittet i to tabeller, R_{31}(D, E) og R_{32}(E, F).

e) I figuren på neste side har vi illustrert situasjonen. Nøkkelattributtene (A og B) er markert med grønn bakgrunnsfarge, de øvrige attributtene med oransje bakgrunnsfarge. Avhengigheten B -> C gir en delvis avhengighet av nøkkelen og bryter derfor kravene til 2NF. Avhengigheten C -> D gir ikke noen delvis avhengighet av nøkkelen, men bryter kravene til 3NF siden C ikke er en supernøkkel og D ikke noe nøkkelattributt.



Oppgave 4 – Lagring, indekser, queryutføring (15 %)

Vi har en tabell for å lagre sensur. Nøkkelen er understreket:

Grading(studno, courseno, grade)

Det er gitt følgende alternative måter å lagre og indeksere denne tabellen:

i) *Heapfil*: Den består av 200 000/200 = 1000 blokker.
ii) *Clustered B+-tre med søkenøkkel (studno, courseno)*. Fordi blokkene i B+-treet har 67% fyllgrad får vi her 50% flere blokker på løvnivå enn heapfila, dvs. 1500 blokker.
iii) *Heapfil og unclustered B+-tre med søkenøkkel (studno, courseno):* Heapfila har her 1000 blokker og B+-treet blir like stort som i ii) (dvs. 1500 blokker) fordi "grade" og "RecordId" (peker til heapfila) antas like store.

a) SELECT grade FROM Grading WHERE courseno='TDT4145' AND studno=123456;

b) INSERT INTO Grading VALUES (123456, 'TDT4145', 'A');

c) SELECT studno, grade FROM Grading WHERE studno < 10000;

	i)	ii)	iii)
a)	500	3	4
b)	2	4	6
c)	1000	75 + 2 (climb) = 77	75 + 2 (climb) + 10,000 = 10,077

Kommentarer til tabellen:

a) Svarene her antar at posten finnes og at den i heapfila vil finnes i midten gjennomsnittlig. Hvis posten ikke finnes ville svaret vært (1000, 3, 3).
b) Vi antar at for heapfila må vi lese og skrive siste blokk. For B+-trærne må vi lese alle blokkene nedover og skrive løvblokka. Når vi i tillegg har en heapfil, må lese og skrive ei blokk i denne også. Vi har ikke tatt hensyn til blokksplitt i disse beregningene. Hver 133. innsetting vil i gjennomsnitt gi en blokksplitt. For heapfila (i) kunne noen også tenkes å sjekke etter at det ikke finnes andre poster med samme nøkkel, dvs. de må lese hele fila.
c) 5% av 1500 blokker er 75 blokker. Nedoverscan er to blokker og bortoverscan er 75 blokker. I tillegg for iii) må vi lese ei blokk fra heapfila per post som tilfredstiller queriet.

Oppgave 5 – Transaksjoner - gjenopprettbarhet (10 %)

H1: r1(A);r2(A);w1(A);r1(B);w2(B);c2;c1;

strict da det ikke er noen lesing eller skriving basert på ikkecommittede endringer.

H2: w1(A);w2(A);c1;c2;

ACA da T2 skriver A som ennå ikke er committet av T1.

H3: r1(A); r2(C); r1(C); r3(A); r3(B); w1(A); w3(B); r2(B); w2(C); w2(B); c1; c2; c3;

unrecoverable da T2 leser B som T3 har skrevet og T2 committer før T3.

Oppgave 6 – Transaksjoner - låser (5 %)

Vi innfører egne operatorer for låsing og opplåsing: readlock – **rl**, writelock – **wl** og unlock – **ul**. Vi låser opp alle låsene ved/etter commit. Det som er problemet med H4 i forhold til låser er når T1 prøver å skrivelåse B, må den vente fordi T2 allerede har en leselås på B. Da vil T2 kjøre ferdig og slippe sine låser, så kan T1 kjøre resten av sine operasjoner. H4 blir da til H5

H5: **rl**1(A);r1(A); **rl**2(B);r2(B); **rl**2(A);r2(A); c2; **ul**2(B);**ul**2(A); **wl**1(B);w1(B);
wl1(C);w1(C); c1; **ul**1(A);**ul**1(B);**ul**1(C)

Her er H5 når låseoperatorene ikke er vist:

H5: r1(A);r2(B);r2(A);c2;w1(B);w1(C);c1;

Oppgave 7 – Transaksjoner - recovery (10 %)

a) En transaksjon har oppdatert en tabell og gjort ei blokk i bufferet "skitten" (dirty). Hvilke metoder finnes for å la transaksjonen oppfylle D-en (durability) i ACID-egenskapene?

   Her er det FORCE eller NO-FORCE som er alternativene:

   1) NO-FORCE: (Redo-)loggen må skrives til disk ved commit.
   2) FORCE: De skitne datablokkene må skrives til disk ved commit.

b) Viktig at det "loopes" over hver loggpost (ikke hver blokk eller hver transaksjon). Man kan starte på 105, da det er de eldste recLSN.

   (*102: Trenger ikke REDO fordi loggpost.LSN < DPT(B).recLSN*)
   (*103: Trenger ikke REDO fordi loggpostens dataelement (C) ikke finnes i DPT.*)
   105: Trenger ikke REDO. B finnes i DPT og DPT(B).recLSN ==loggpost.LSN. Må derfor lese inn B fra disk og ser da at B.pageLSN==loggpost.LSN.
   107: Trenger ikke REDO fordi loggpost.LSN < DPT(D).recLSN.
   108: Trenger REDO fordi loggpost.LSN >= DPT(D).recLSN og loggpost.LSN > D.pageLSN.