NTNU
Det skapende universitet

Institutt for datateknikk og informasjonsvitenskap

Løsningsskisse til Eksamensoppgave i TDT4145 Datamodellering og databasesystemer

Eksamensdato: 4. august 2015

Eksamenstid (fra-til): 15:00 - 19:00

Hjelpemiddelkode/Tillatte hjelpemidler:

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

Versjon 5. mai 2016.

Oppgave 1 – Datamodeller (20 %)

a) **Foretrukket løsning:**

Person(PID, Name)

Phone(PersonID, PhoneNo)

-- PersonID er fremmednøkkel mot Person-tabellen.

Dog(RegNo, Name, Breed, OwnerID)

-- OwnerID er fremmednøkkel mot Person-tabellen.

UnwantedIncident(DogRegNo, VictimPersonID, WalkerPersonID, Severity)

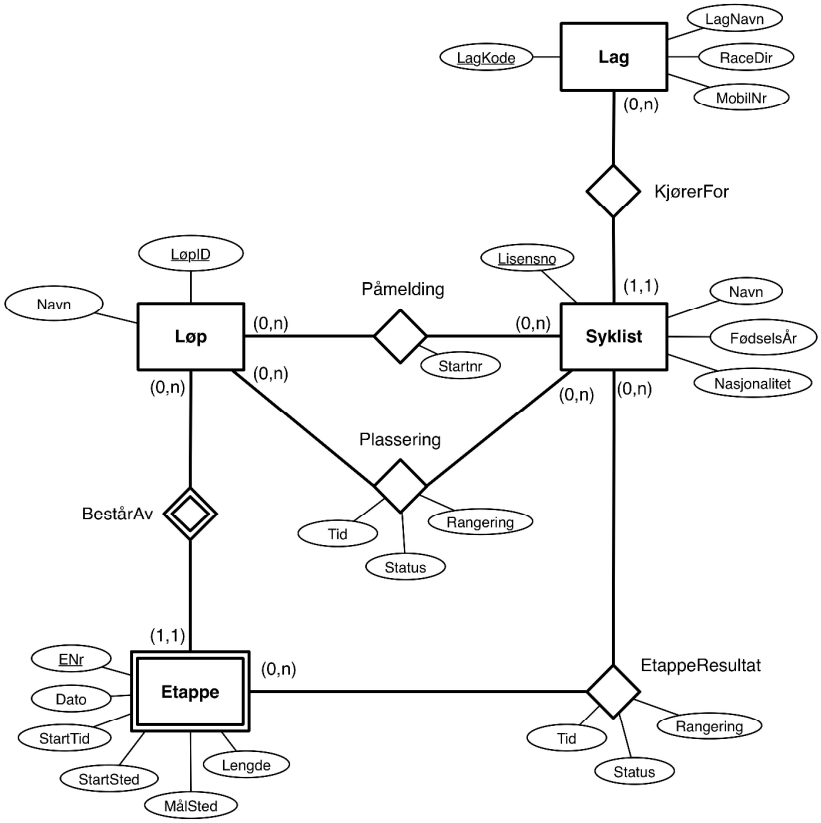
-- DogRegNo er fremmednøkkel mot Dog-tabellen. VictimPersonID og WalkerPersonID er fremmednøkler mot Person-tabellen.

RegisteredDogWalker(DogRegNo, WalkerPersonID**)**

-- DogRegNo er fremmednøkkel mot Dog-tabellen. WalkerPersonID er fremmednøkkel mot Person-tabellen.

Siden spesialiseringen av Person er *delvis*, må vi ha en tabell for superklassen. Det som kan være et alternativ er å ha egne tabeller for subklassene. Gevinsten ved en slik løsning vil være liten i forhold til den økte kompleksiteten i relasjonsskjemaet.

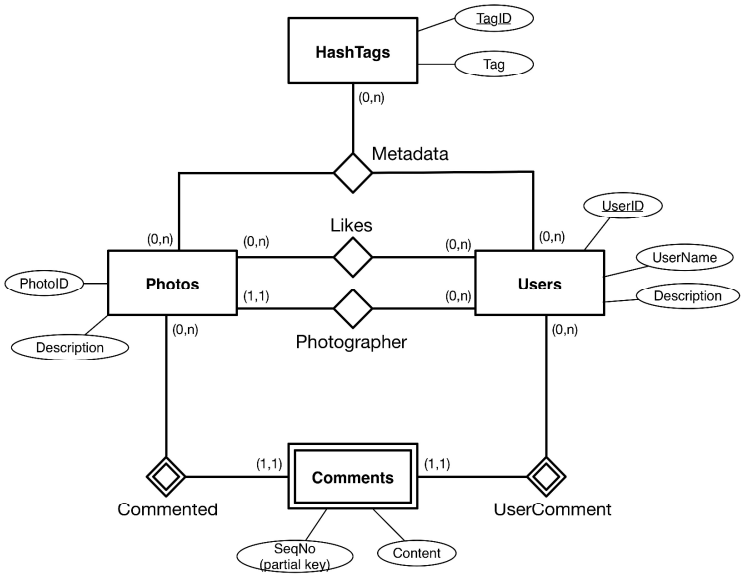
b) ER-diagrammet under viser en mulig datamodell. Det vil være like riktig å spesifisere attributtene inne i boksene.



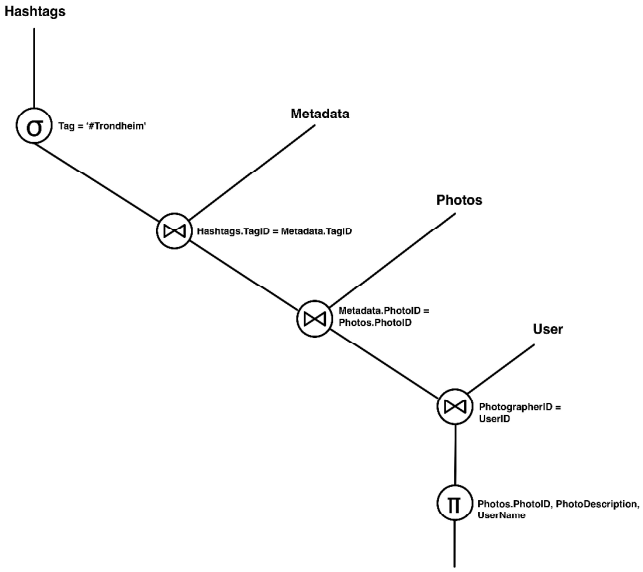
Oppgave 2 – Relasjonsalgebra og SQL (21 %)

I SQL-oppgavene er svarene gitt med join spesifisert i FROM-delen. Det skal ikke trekkes for løsninger som spesifiserer join-betingelser i WHERE-delen.

a) ER-diagram



b) Relasjonsalgebra:



c)
```
SELECT TagID, Tag
FROM Hashtags
WHERE Tag LIKE '%Trondheim%'
ORDER BY Tag ASC;
```

d)
```
DELETE FROM Likes
WHERE UserID = 'xyz00' AND
      PhotoID IN (SELECT PhotoID
                  FROM Photos
                  WHERE PhotographerID = 'abc11')
```

e)
```
SELECT UserID
FROM Comments
WHERE PhotoID IN (SELECT PhotoID
                  FROM Metadata NATURAL JOIN Hashtags
                  WHERE Tag = '#Beiarn')
```

Oppgave 3 – Teori (19 %)

a) En funksjonell avhengighet, X –> Y, gjelder hvis det for alle tupler (rader) som har samme verdier for attributtene i X, må være slik at verdiene for attributtene i Y er de samme.

Tabelleksempel: Person(PersonNr, Navn, Fødselsår). PersonNr -> Navn vil gjelde når PersonNr er unikt for en person og hver person kan ha bare ett navn. FødselsÅr -> Navn vil ikke gjelder. Det ville i så fall forutsatt at alle med samme fødselsår har samme navn.

b) Tapsløst-join-egenskapen garanterer at man ved joining av komponent-tabellene til utgangspunkt-tabellen, alltid vil få samme innhold som i utgangspunkt-tabellen. Dersom man ikke har tapsløst-join-egenskapen, vil en sammenjoining av komponent-tabellene kunne gi tupler som ikke finnes i utgangspunkt-tabellen.

En dekomponering i 2 komponenttabeller har tapsløst-join-egenskapen hvis komponenttabellenes felles attributt(er) er en supernøkkel i en eller begge komponenttabellene.

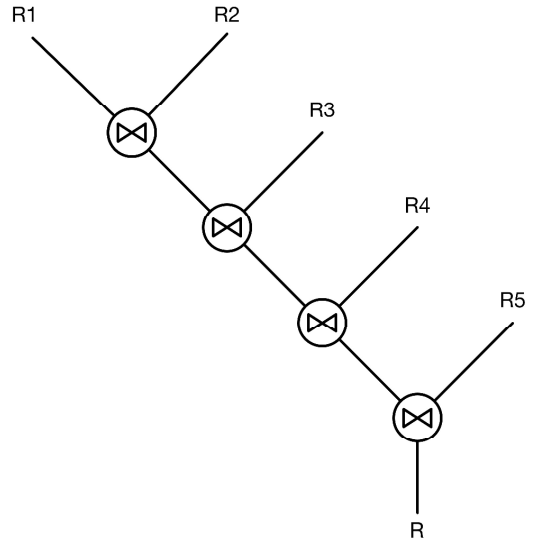
R(a, b, c) med F = { b -> c } kan splittes tapsløst i R_1(a b) og R_2(b c) siden R_1 og R_2 sitt felles attributt (b) er en supernøkkel i R_2.

R(a, b, c) med F = { b -> c } kan ikke splittes tapsløst i R_1(a b) og R_2(a c) siden R_1 og R_2 sitt felles attributt (a) ikke er en supernøkkel i verken R_1 eller R_2.

c) Tabellens eneste kandidatnøkkel er abe. Vi har ikke-nøkkel-attributter som er delvis avhengig av kandidatnøkkelen (b -> c og e -> d). Dette bryter kravet til andre normalform. Den høyeste normalformen som oppfylles er derfor 1NF.

d) En mulig dekomponering er: R_1(A, B, C, E), R_2(A, D), R_2(B, D) og R_4(C, D) og R_5(D, E). ABCE er primærnøkkel i R_1. A er primærnøkkel i R_2. B er primærnøkkel i R_3, C er primærnøkkel i R_4 og E er primærnøkkel i R_5. Dekomponeringer vurderes etter fire forhold:

- **Attributtbevaring:** $R = R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5$ så det er ivaretatt.
- **FD-bevaring:** A -> D ivaretas i R_2, B - > D ivaretas i R_3, C -> D ivaretas i R_4 og E -> D ivaretas i R_5 Det betyr at alle FD-er i F er ivaretatt og vi har FD-bevaring.
- **Tapsløst-join:** Figuren under vises hvordan R_1, R_2, R_3, R_4 og R_5 kan joines tapsløst til R (i hver av joinene er komponent-tabellenes felles attributter en supernøkkel i minst en av komponenttabellene). Egenskapen kan alternativt vises ved å bruke matrisemetoden (algoritme 15.3 i læreboken).



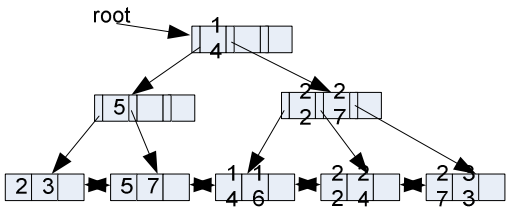
- **Normalform:** R_1 har $F_1 = \varnothing$ og er på BCNF siden R_1 ikke har noen ikke-trivielle funksjonelle avhengigheter. R_2 har F_2 = {A -> D} og er på BCNF siden A er en (super-)nøkkel i R_2. R_3 har F_3 = {B -> D} og er på BCNF siden B er (super-)nøkkel i R_3. R_4 har F_4 = {C -> D} og er på BCNF siden C er (super-)nøkkel i R_4. R_5 har F_5 = {E -> D} og er på BCNF siden E er (super-)nøkkel i R_5.

Dekomponeringen har alle ønskede egenskaper.

Oppgave 4 – Lagring og skalering (10 %)

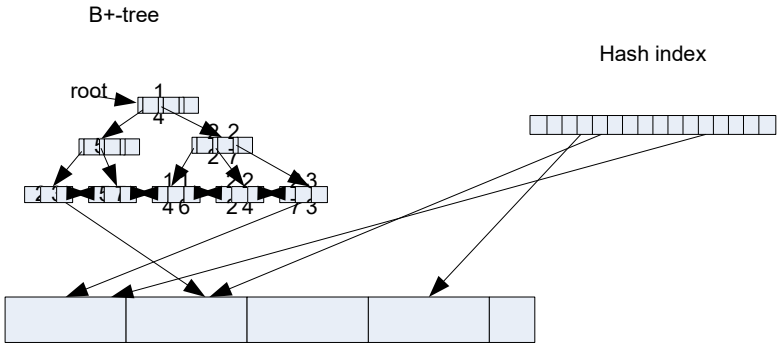
Dette er hentet fra pensumnotatet «Storage definitions» og viser typiske lagrings- og indekseringsalternativer brukt i systemer. Det finnes andre alternativer også.

- *Clustered B+-tree/clustered index*. There is a B+-tree index on the primary key. The leaf level in the B+-tree stores the actual records/rows of the table. Thus, it is a clustered index.



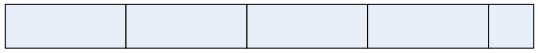
  This alternative is used within MySQL's InnoDB storage engine. It is also the storage alternative used in MicroSoft's SQL server when a primary key is defined for a table. There, it is called a *clustered index*.
  This alternative is good for most cases: Direct access on key, range scans, insertions, …

- *Heap file and B+-tree*. The table is stored in a heap file. There is a B+-tree index on the primary key. There may be a hash index on some fields of the records.



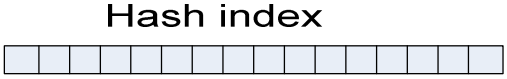
  Heap file with a B+-tree index is used in MySQL's MyISAM storage engine.
  Heap files are good for insertions and table scans. The B+-tree is useful for access on the indexed attribute. The hash index is perfect for direct access on the indexed attribute.

- *Heap file*.



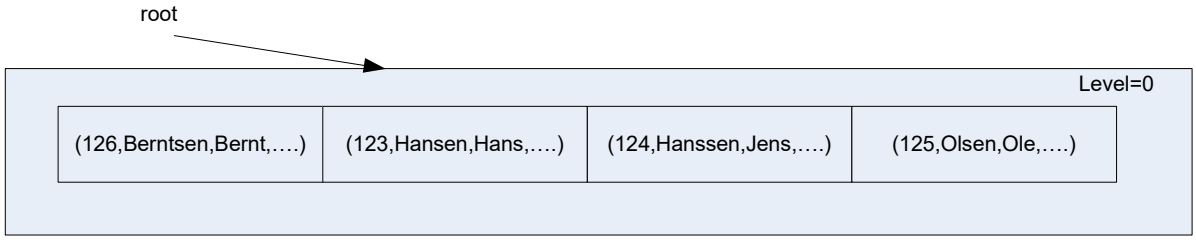
  In MicroSoft's SQL Server you get a heap file when no primary key is defined for a table.
  The heap file is good for insertions and full table scans, but not good for others accesses.

- *Clustered hash index*. Hash index on the primary key. Clustered index storing the actual records/rows of the table, i.e. a clustered hash index. It is called a *hash cluster* in Oracle.

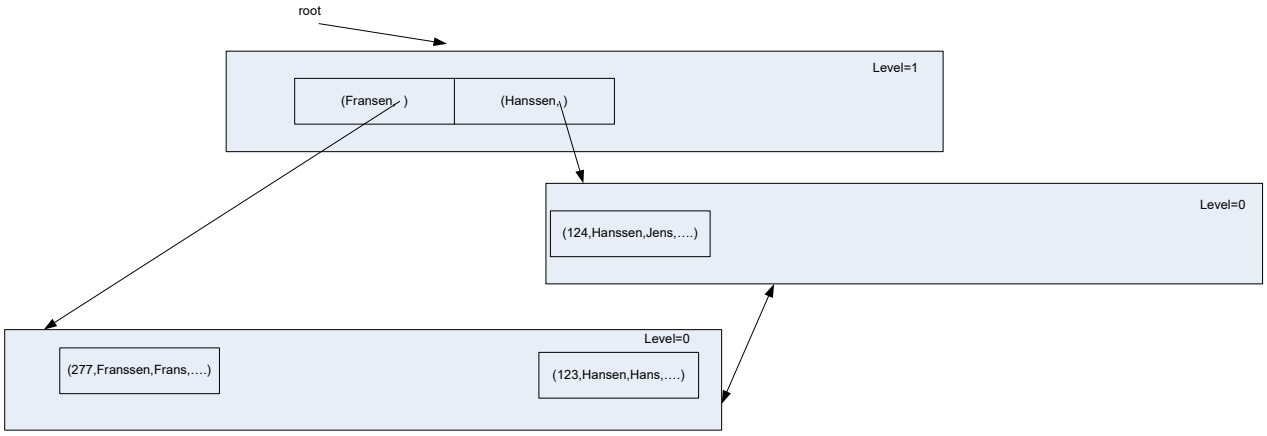


Oppgave 5 – B+-trær (10 %)

a) Her får alle postene plass i ei blokk, dermed består B+-treet av ei blokk på nivå 0 og som også er rota til treet. Det er viktig at studenten viser at postene er sortert på søkenøkkel. I dette tilfellet er etternavnet søkenøkkel.



b) Etter splitting får vi (minst) to nivåer. Det kan være flere nivåer over de som er vist. Rota her peker på level=1, men den kan være en peker til level=2 eller level=3 også, avhengig av hvor mange poster som er satt inn. Vi har ikke vist sidevegspekere annet enn mellom de to aktuelle blokkene på nivå 0. (277,Fransen,Frans, …) er vist for å kunne illustrere den forrige pekeren i blokka på level=1.



Oppgave 6 – Transaksjoner - serialiserbarhet (5 %)

H1: r1(A); w1(A); r2(B); r2(A); w2(A); 1 -> 2. Konflikserialiserbar

H2: r1(A); w1(A); w2(A); 1-> 2. Konfliktserialiserbar.

H3: r1(A); r2(C); r1(C); r3(A); r3(B); w1(A); w3(B); r2(B); w2(C); w2(B); 3->1->2 og 3->2. Konfliktserialiserbar.

Oppgave 7 – Transaksjoner - recovery (10 %)

a) PageLSN er et felt i datablokkene som forteller hvilken loggpost som sist endret blokka.
   1. Det brukes for å kunne avgjøre om en loggpost trenger redo mot denne blokka.
   2. Det brukes også til å sjekke at loggen er skrevet før datablokka, dvs. når datablokka skal skrives, sjekkes det om loggen er skrevet ut fram til og med PageLSN. Hvis nei, må loggen skrives først (WAL).

b) NO-FORCE har den fordelen at ved Commit må man skrive loggen og ikke de skitne blokkene.
   1. Loggen er sekvensiell og kan skrives raskt ut til disk. FORCE krever utskrift av datablokker og det kan være mange blokker og de kan være spredd utover på disken.
   2. NO-FORCE (redo-logging) gjør skriving av datablokker kan slås sammen, f.eks. ved oppdateringer til samme blokk. I tillegg kan vi f.eks. sortere alle writes på diskadresser slik at roterende disker kan få bedre throughput.

c) Man må sjekke at loggen er skrevet ut først ved å se på PageLSN og sjekke mot FlushedLSN (nyeste skrevne loggpost). Hvis loggen ikke er skrevet ut, så må man skrive ut denne først.