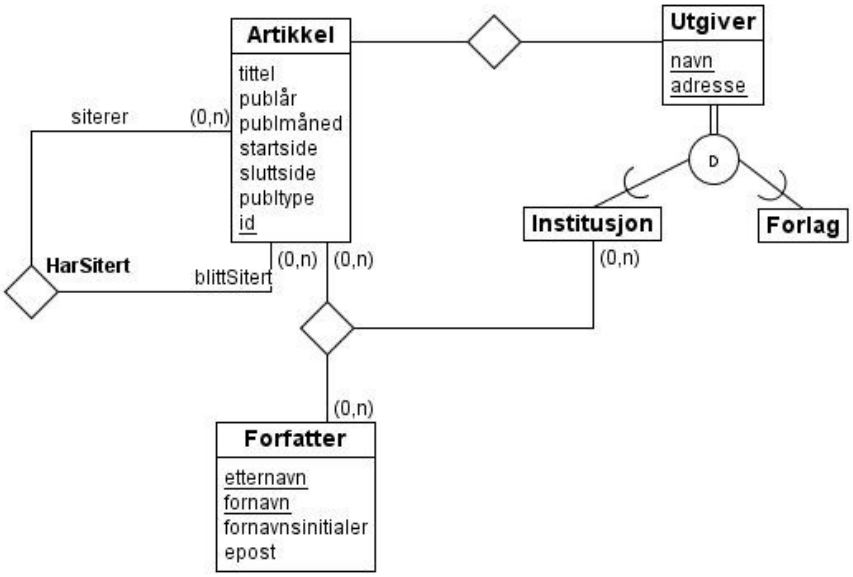
Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

LØSNINGSFORSLAG TIL
EKSAMENSOPPGAVE I FAG TDT4145 – DATAMODELLERING OG DATABASESYSTEMER
10. august 2009, **ver. 16.mai 2010**

Oppgave 1 – Datamodellering – 20 %

Oppgaven er relativt enkel. Det som er utfordringen er relasjonen mellom Artikkel, Forfatter og Institusjon. Den kan også modelleres som er sett med tovegsrelasjoner. I denne løsningen har vi ikke lagt vekt på å lage noe hierarki av artikkeltyper.



Oppgave 2 – Relasjonsalgebra og SQL – 20 %

a) *PROJECT_fødeland(SELECT_filmtittel='Cinema paradiso' (SpillerI) JOIN Skuespiller)*

b) *PROJECT_filmtittel(Film)-PROJECT_filmtittel(SELECT_(navn='Jim Carrey' AND fødeår=1962)(SpillerI)))*

c)
```
SELECT filmtittel, filmaar
FROM  film
ORDER BY filmaar ASC;
```

d)
```
SELECT rollenavn
FROM Film, SpillerI
WHERE filmkategori='komedie' AND
    Film.filmtittel=SpillerI.filmtittel AND
    Film.filmaar=SpillerI.filmaar;
```

e)
```
SELECT filmkategori, SUM(lengde) AS sumlengde
FROM film
GROUP BY filmkategori
ORDER BY sumlengde DESC;
```

f)
```
SELECT navn,fødselsår
FROM SpillerI
GROUP BY navn,fødselsår
HAVING count(*)>= ALL (SELECT count(*)
                       FROM SpillerI
                       GROUP BY navn,fødselsår);
```

Oppgave 3 – Lagring og indekser – 20%

a) To enforce the constraint that *eid* is a key, all we need to do is make the clustered index on *eid unique* and *dense*. That is, there is at least one data entry for each *eid* value that appears in an Emp record (because the index is dense). Further, there should be exactly one data entry for each such *eid* value (because the index is unique), and this can be enforced on inserts and updates.

b) If we want to change the salaries of employees whose *eid*'s are in a particular range, it would be sped up by the index on *eid*. Since we could access the records that we want much quicker and we wouldn't have to change any of the indexes.

c) If we were to add 1 to the ages of all employees then we would be slowed down, since we would have to update the index on age.

d) Hashfila inneholder postene. Hver post kan være 16 byte lang hvis postene er lagret tett med dictionarybeskrivelse. 20000*16/(0.8*8192) blir 49 blokker. B+-treet inneholder 50 poster med 20000/50 pekere. Hver post i B+-treindeksen blir da 1604 byte. Da det sannsynligvis er variabelt antall pekere per post bør kanskje posten være litt lengre. Med 67% fyllgrad blir det ca . 4 poster per blokk. Da trenger vi 13 blokker på løvnivå og en rotblokk. Dvs til sammen 49+14 = **63** blokker.

Oppgave 4 – Transaksjoner – 20%

a) Write-ahead logging (WAL) benytter seg av to forskjellige teknikker:

1) Den skriver loggposten før den skriver den berørte datablokka, slik at det alltid er mulig å gjøre undo av loggposten.
2) Den tvinger (redo-)loggen til disk før commit (force-log at commit). Dette gjør at det er mulig å slippe å skrive datablokkene (dvs. NO-FORCE) før commit. I stedet skrives den sekvensielle loggen lett til disk.

b)
S_1: Cascadeless (ACA)
S_2: Recoverable
S_3: Strict

c) De to datastrukturene som lagres i hvert sjekkpunkt er dirty page table (DPT) og transaksjonstabellen. DPT brukes for å vite hvilke sider som var skitne i bufferet når sjekkpunktet

startet. For hver side blir det lagret recoveryLSN, dvs LSN til eldste loggpost som gjorde siden skitten. Dette brukes for å "screene" hvilke loggposter som trenger REDO. Hvis loggpostens LSN er mindre enn RecoveryLSN, trenger ikke loggposten redoes, og man slipper å lese blokka inn. Det samme gjelder hvis blokka loggposten refererer til ikke finnes i DPT. Transaksjonstabellen brukes for å finne hvilke transaksjoner som var aktive ved starten av sjekkpunktet. Analysefasen består i å lese i transaksjonstabellen og skanne loggen framover for å finne ut hvilke transaksjoner som var aktive ved krasjtidspunktet. Disse transaksjonene blir rullet tilbake i UNDO-fasen av recovery.

d)

a. Serialiserbarhet: I - isolasjon
b. Gjenopprettbarhet: A - atomiskhet
c. ACA - Avoiding Cascading Aborts: A - atomiskhet
d. Restriksjoner: C - konsistens
e. Strict historie: I - isolasjon
f. Rollback: A - atomiskhet
g. Loggpost: A – atomiskhet, D - durability

Oppgave 6 – Normalisering – 20%

a) Navn -> Snr
Snr -> Navn
Emnenr -> Eksamensdato
Snr,Emnenr ->Kandidatnr,Karakter
Kandidatnr,Emnenr->Karakter
Navn,Emnenr->Karakter

b) **Innsettingsanomalier**. Det går ikke å sette inn en navn og snr uten å sette inn eksamensresultater. **Slettingsanomalier**. Hvis du sletter informasjon om en karakter, kan du slette all informasjon om studenten. **Oppdateringsanomalier**. Siden informasjon om emnenr og eksamensdato er redundant lagret, kan det bli inkonsistents ved oppdatering. **Redundans**. Dobbeltlagring av snr og eksamensdato.

c) Kandidatnøkler er **snr, emnenr** og **navn, emnenr**. Eksamensdato er delvis avhengig av nøkkel (snr). Da er ikke tabellen på 2. normalform. Vi antar da at tabellen er på 1.normalform.

d) Det er noen FAer som bryter med BCNF:
navn->snr
snr->navn
emnenr->eksamensdato
Vi tar ut disse i to separate tabeller: Student og Eksamensdato.
Vi lagrer resten av tabellen i Eksamen. Da mister vi FAen emnenr,kandidatenr->karakter.
Student(navn, snr)
Eksamen(snr,emnenr,kandidatnr,karakter)
Eksamensdato(emnenr,eksamensdato)

.