NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informatikk

Løsning til

Eksamensoppgave i TDT4145 Datamodellering og databasesystemer

Faglig kontakt under eksamen:

Roger Midtstraum: 995 72 420

Svein Erik Bratsberg: 995 39 963

Eksamensdato: 7. august 2017

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

Hjelpemiddelkode/Tillatte hjelpemidler:

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

Annen informasjon:

Målform/språk: Norsk bokmål

Antall sider (uten forside):

Antall sider vedlegg:

Informasjon om trykking av eksamensoppgave

Originalen er:

1-sidig ☐ **2-sidig** ☐

sort/hvit ☐ **farger** ☐

skal ha flervalgskjema ☐

Kontrollert av:

Dato Sign

Oppgave 1 – Datamodeller og relasjonsdatabaser (15 %)

a) Alternativ 1 er enklest og viser tydelig at R representerer en relasjonsklasse mellom attributter i entitetsklassene A og B. Mellom en bestemt entitet i A og en bestemt entitet i B kan det bare finnes en relasjon i relasjonsklassen R.

   Alternativ 2 er mer omfattende, bruker flere symboler og det er ikke like entydig at entitetsklassen R representerer en relasjon mellom entiteter i A og B. Det vil ofte være nødvendig å legge til et ekstra nøkkelattributt. Fordeler med dette alternativet er at det kan finnes flere ulike relasjoner mellom samme entiteter i A og B. R kan også inngå i egne relasjonsklasser, dette er ikke mulig i alternativ 1.

b) Siden spesialiseringen av A er total, trenger vi ikke en egen tabell for A. Hvis vi velger en slik løsning blir relasjonsdatabase-skjemaet:

   B(a1, b1, b2, e1) – e1 er fremmednøkkel mot E, kan være NULL.

   C(a1, c1, c2, e1) – e1 er fremmednøkkel mot E, kan være NULL.

   E(e1, e2)

   F(f1, f2)

   CF(a1,f1) – a1 er fremmednøkkel mot C, kan ikke være NULL. f1 er fremmednøkkel mot F, kan ikke være NULL.

Oppgave 2 – Teori (10 %)

a) Redundans vil i denne sammenhengen bety at vi har et databaseskjema der vi lagrer den samme informasjonen flere ganger. Dette åpner i alminnelighet for inkonsistens i databasen. I tillegg kan vi oppleve såkalte anomalier knyttet til innsetting, oppdatering og sletting av data i databasen. I en normaliseringsprosess vil vi fjerne redundans gjennom å splitte dataene over flere tabeller. Gjennom en slik prosess søker vi å vi oppnå et relasjonsdatabaseskjema der vi har eliminert muligheten for inkonsistens og blitt kvitt anomaliene. Ulempene med en slik tilnærming er at vi kan ende opp med et databaseskjema med mange tabeller og stort behov for å joine tabeller når vi skal finne de dataene vi trenger. Ved å tillate (noe) redundans kan vi noen ganger få et enklere relasjonsskjema som gir enklere og mer effektive spørringer, men ulempen vil være at vi må håndtere redundansens bivirkninger.

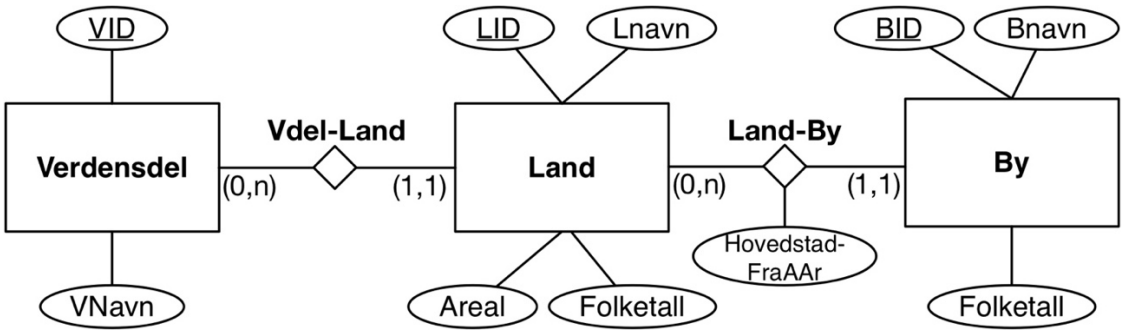
   I tillegg til ”skjema-basert”-redundans som vi har diskutert over, kan redundant lagring av de samme dataelementene i noen sammenhenger være hensiktsmessig for å oppnå tilgjengelighet, ytelse og sikkerhet.

b) En NULL-verdi er en spesiell verdi som kommer i tillegg til en verdimengdes (datatypes) ordinære verdier. Den brukes når et attributt ikke har en verdi eller denne verdien er ukjent. Uten NULL-verdien ville vi måtte tilordne denne spesielle oppgaven til en av verdiene i verdimengden – dette ville vært mindre standardisert, krevd mer dokumentasjon og forutsatt kjennskap til denne tilordningen blant alle som bruker

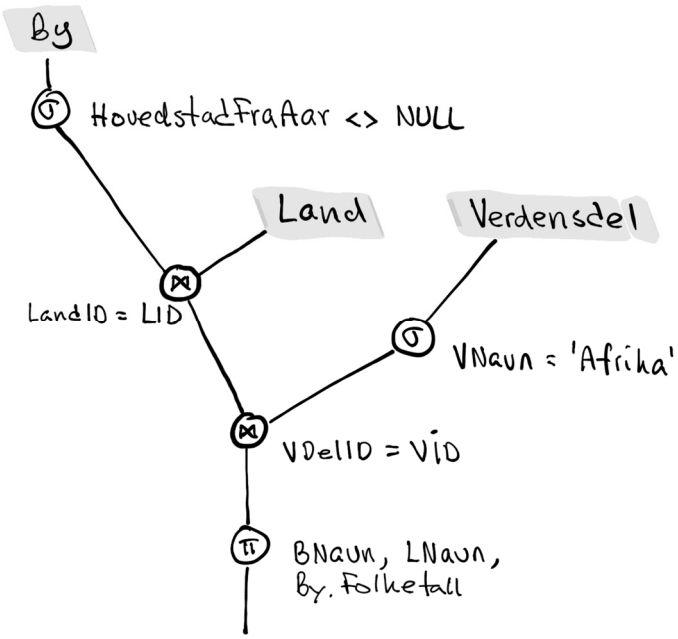
databasen. I tillegg ville den verdien som ble valgt være utilgjengelig for sitt opprinnelige formål.

Oppgave 3 – Relasjonsdatabaser, ER-modeller, Relasjonsalgebra og SQL (20 %)

a) ER-modell:



b) Relasjonsalgebra:



c) SELECT VID, VNavn, count(LID), sum(Areal), sum(Folketall)
FROM Verdensdel INNER JOIN Land On VID = VDelID
GROUP BY VID, Vnavn

d) SELECT BNavn, By.Folketall, LNavn, VNavn
FROM By INNER JOIN Land ON LandID = LID
INNER JOIN Verdensdel ON VDelID = VID
WHERE By.Folketall > 5000000
ORDER BY By.Folketall DESC, BNavn ASC

e)
```
CREATE TABLE Land (
    LID INTEGER NOT NULL,
    LNavn VARCHAR(30),
    Areal DOUBLE,
    Folketall INTEGER,
    VDelID INTEGER NOT NULL,
    CONSTRAINT pk1 PRIMARY KEY (LID),
    CONSTRAINT fk1 FOREIGN KEY (VDelID) REFERENCES Verdendel(VID)
      ON DELETE CASCADE
      ON UPDATE CASCADE)
```

Oppgave 4 - Normaliseringsteori (15 %)

a) S må ha verdien 2. T kan ha alle verdier, unntatt 4 og kan bare være 5 hvis V er 1. U kan ikke ha verdien 2. Paret (V,W) kan ikke ha verdiene (1,2) eller (6,1). Vi kan heller ikke ha (V,W) = (1,U).

b) Kandidatnøklene er: AC og EC. Ikke-nøkkelattributtene er dermed: BD.

Andre normalform er oppfylt fordi det ikke finnes funksjonelle avhengigheter fra en del av en kandidatnøkkel til et ikke-nøkkelattributt.

Tredje normalform er oppfylt siden det for alle funksjonelle avhengigheter er slik at venstresiden er en supernøkkel (AC→BD og EC→BD) eller høyresiden er nøkkelattributter (A→E og E→A).

Boyce-Codd Normalform er ikke oppfylt siden det finnes funksjonelle avhengigheter (A→E og E→A) der venstreside-attributtene ikke er en supernøkkel.

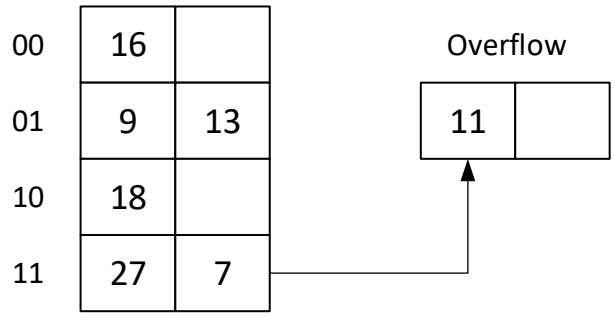
c) Når F = Φ har vi ingen restriksjoner mellom radene. Vi kan da ha så mange ulike rader som verdimengdene tillater, i dette tilfellet 3x3x3, altså 27 ulike rader. Når F = {A→BC} gjelder må alle rader med samme A-verdi ha samme verdier for B og C. Vi har tre mulige verdier for A og kan derfor ikke ha flere enn 3 rader i tabellen.

d) En dekomponering av en tabell, R, i to (komponent-)tabeller, R_1 og R_2, har *tapsløst-join egenskapen* når vi for alle mulige forekomster av R, kan fordele dataene på R_1 og R_2, og sammenstilling (join) av R_1 og R_2 vil gi den tabellforekomsten vi startet med. Dersom en dekomponering ikke har tapsløst-join-egenskapen, vil sammenstillingen kunne generere (nye) rader som ikke var med i den opprinnelige tabellforekomsten. De nye radene vil være ”søppeldata” og viser at dekomponeringen ikke er ekvivalent med utgangstabellen.

e) Den funksjonelle avhengigheten A→B representerer en restriksjon om at (alle) rader med samme verdi for A, må ha samme verdi for B. Hvis vi for eksempel har Personnr→Fornavn betyr dette at en person kan ha bare en verdi for fornavn, det betyr ikke at personnummeret fører til eller bestemmer en persons fornavn.

Oppgave 5 – Statisk hashing (5 %)

27 MOD 4 = 3 = 11 (binært)
18 MOD 4 = 2 = 10
 9 MOD 4 = 1 = 01
 7 MOD 4 = 3 = 11
16 MOD 4 = 0 = 00
13 MOD 4 = 1 = 01
11 MOD 4 = 3 = 11



Oppgave 6 – Lagring, indeksering og queries (10 %)

Vi har en database som lagrer webclicks i en tabell:

Click (<u>clickId</u>, user, url, time)

Hver post (record) i tabellen er 100 byte lang og hver blokk er 8 KB (8192 bytes). Vi har registrert 300 000 klikk i databasen.

a) Vi ønsker å lagre tabellen i et clustered B+-tree med clickId (8 byte) som søkenøkkel. Hvor mange blokker er det på hvert nivå i B+-treet? Beskriv eventuelle antagelser du tar.
Løsning:
Level 0: Plass til 81 poster per blokk, men 2/3 fyllgrad gir 54 poster per blokk. Da er det 300 000 / 54 = 5556 blokker.
Level 1: Hver post er 8 byte (ClickId) + 4 byte (BlockId) = 12 byte. 2/3 fyllgrad gir 455 poster per blokk. Dvs. 13 blokker til sammen (5556 poster).
Level 2: 1 blokk (rotblokk)

b) Vi ønsker å utføre følgende query:

```
SELECT url, count(*) AS clickcount
FROM Click
GROUP BY url
ORDER BY clickcount DESC;
```

Hvordan vil du indeksere tabellen for å kunne svare på queriet?
Begrunn svaret ditt.

Løsning: *Bør ha indeks på URL, med en liste av ClickId per URL (variabel lengde post). Kjører Index-only query på denne og teller opp antall ClickId per URL og lagrer i temp-fil. Sorterer denne etterpå på antall ClickId. Vi trenger ikke en clustered indeks her, det holder med unclustered. Da kan selve klikkene lagres et annet sted, f.eks. i B+-treet.*

Oppgave 8 – Transaksjoner (15 %)

a) Forklar de fire begrepene som forkortes ACID.
Løsning: *Se kap 20.3 i E&N.*

b) Vi har en historie:
H_1: $r_1(A)$; $w_2(A)$; $w_2(B)$; $w_3(B)$; $w_1(B)$; C_1; C_2; C_3;
Vi innfører tofaselåsing (rigorous 2PL). Skriv om historien slik at den bruker 2PL. Innfør operasjonene wl(X) – write_lock(X) og rl(X) – read_lock(X).
Løsning:
(burde vært spesifisert at unlock også skulle være med)

T1	T2	T3
rl1(A)		
r1(A)		
		wl3(B)
		w3(B)
		C3; unlock(B)
wl1(B)		
w1(B)		
C1; unlock(A,B)		
	wl2(A)	
	w2(A)	
	wl2(B)	
	w2(B)	
	C2; unlock(A,B)	

c) For hver klasse av samtidighetsproblemer gitt under, hvilke ser du i historien H_1 i b) og hvorfor / hvorfor ikke?
- Dirty read: *Nei, ingen lesning av «skitne» verdier her.*
- Lost update*: Ja, r1(A) ….. w1(B). Her er det sannsynligvis en sammenheng. w2(A); w2(B) har også en sammenheng. Dermed «overlever» verdien til A, mens T2s B-verdi blir overskrevet.*
- Unrepeatable read: *Nei.*
- Incorrect summary: *Nei.*

Oppgave 9 – Transaksjoner: Recovery – ARIES (10 %)

a) Beskriv og forklar hva som skjer i de tre fasene av recovery etter krasj i ARIES.
Løsning: *Se kap 22.5 i E&N.*

b) Hva er PageLSN og hvorfor er dette en viktig oppfinnelse?
Løsning: *I hver datablokk/page (både i buffer og på disk) lagres det et felt som kalles PageLSN, som viser hvilken loggpost som sist ble utført på blokka. PageLSN er en svært viktig oppfinnelse fordi den forenkler testing av om REDO er nødvendig. Hvis en logget operasjon er allerede gjort mot ei blokk, vil dette være reflektert i blokkas PageLSN. Da slipper man å undersøke blokka for å finne ut om operasjonen er allerede gjort ved å se på data i blokka.*