Databasesystemer

IT Universitetet i København

7. juni 2005

Eksamenssættet består af 5 opgaver med 13 spørgsmål, fordelt på 6 sider (inklusiv denne side).

Vægten af hver opgave er angivet. Du har 4 timer til at besvare alle spørgsmål. Hvis du ikke er i stand til at give et fuldt svar på et spørgsmål, så prøv at give et delvist svar. Du kan vælge at skrive på engelsk eller dansk (evt. med engelske termer).

Siderne i besvarelsen skal være numererede, og forsynet med navn, CPR nummer og kursuskode (DBS). Skriv kun på forsiden af arkene, og sortér dem inden numereringen, så opgaverne forekommer i nummerrækkefølge.

"MDM" refererer i sættet til kursusbogen "Modern Database Management 7th edition" af Jeffery A. Hoffer, Mary B. Prescott and Fred R. McFadden.

Alle skriftlige hjælpemidler er tilladt.

1 Datamodellering (30%)

a) Udarbejd en datamodel for blodbanken i Sengeløse Sygehus ved brug af EER notationen beskrevet i MDM kapitel 3 og 4. Nedenfor er en beskrivelse af det, som systemet skal dække. I det omfang beskrivelsen ikke er fyldestgørende, må du selv gøre dig nogle antagelser. (Sådanne antagelser skal fremgå af besvarelsen.) Ved vurderingen lægges der vægt på, at der er mindst mulig redundans i databasen, og at attributter med samme indhold og betydning ikke unødigt bruges på flere entitetstyper.

b) Konvertér din EER model fra spørgsmål a) til en relationel datamodel. Du skal blot angive attributnavne for hver relation, og f.eks. *ikke* specificere datatyper, fremmednøgler, etc. Angiv hvilken metode, du anvender til konverteringen (fra MDM eller fra kursets forelæsningsslides).

Casebeskrivelse:
Sengeløse Sygehus er et mindre lokalt sygehus, der har skadestue, tilbyder ambulante behandlinger, og desuden driver en blodbank, der tapper lokale donorer og er leverandør af blodportioner til regionens større sygehuse. Administrationen af blodbanken er hidtil foregået ved hjælp af et papirbaseret journalsystem. I forbindelse med pensioneringen af blodbankens mangeårige sekretær, er det besluttet at erstatte systemet med et moderne IT system, der desuden skal have en række nye funktioner.

For hver donor findes et donoroplysningskort med personinformation samt information om tapninger. Flg. er et eksempel på et sådant donorkort:

DONOROPLYSNINGSKORT		
CPR: 060275-1133		
Navn: Rasmus Berg		
Adresse: Bakketoppen 1		
Postnr: 1313 Bjergby		
Tlf.: 1223 3344		
Blodtype: A pos.		
Tapninger (dato, tapning/kontrol):		
1.	11/2-2001	ABP/AG
2.	13/5-2001	PR/AG
3.	29/9-2001	AG/ABP
...		

Oplysningerne om tapning og kontrol handler om, hvem fra personalet, der har foretaget selve tapningen, og hvem der har lavet den obligatoriske kontrol af oplysningerne. Det er tanken, at den information om tapninger, der findes i det eksisterende journalsystem *ikke* skal overføres. Dog ønskes det, at information om, hvor mange gange en donor er tappet i alt, gemmes i det nye system. Ud over disse oplysninger skal det for hver donor registreres,

hvornår denne senest er blevet indkaldt til tapning (dette sker ved brev), og hvornår det evt. er aftalt, at tapningen skal foretages.

For personalet skal gemmes standardoplysninger om CPR nummer, navn, adresse, og telefon. Desuden skal det registreres, hvornår en person er ansat (og evt. ophørt med ansættelsen), hvilken uddannelse vedkommende har, samt hvad personens unikke initialer er. (Det er disse initialer, der anvendes på donoroplysningskortene.) Det er *ikke* nødvendigt at gemme oplysninger om tidligere ansættelser, hvis personen har været ansat flere gange. For bioanalytikere skal der yderligere gemmes information om, hvorvidt de har uddannelsesfunktion eller ej. For læger skal der yderligere gemmes information om, hvilke specialer de har (f.eks. genetik eller mikrobiologi). For sygeplejersker skal der ikke gemmes yderligere information.

Den sidste type af information, der skal gemmes i systemet, omhandler blodportionerne. For hver blodportion skal det registreres, hvilken tapning portionen stammer fra, hvornår tapningen fandt sted, hvilken blodtype det drejer sig om, og hvad blodprocenten er. Når en blodportion sendes til en afdeling på et andet sygehus, skal leverancen registreres, så man efterfølgende kan se hvor portionerne blev sendt hen, og hvornår det skete. Til brug ved leverancerne skal systemet registrere navn og adresse for alle sygehuse, der aftager blod fra Sengeløse Blodbank, samt navn på den person på hver afdeling, der er ansvarlig for modtagelse af blod.

2 Normalisering (10%)

I denne opgave betragter vi relationen `Behandler`, der indeholder information om medicinske behandlere, og hvilke sygdomme de behandler:

```
Behandler(id,adresse,postnr,by,speciale,sygdom)
```

Hver behandler har en unik `id`, og desuden præcis én registreret adresse. Til ethvert postnummer svarer netop ét bynavn. En behandler kan have mere end ét speciale, men i hver by er der højst én behandler med et givet speciale. En sygdom hører til præcis ét speciale, men et speciale kan dække mange sygdomme.

a) Angiv alle kandidatnøgler i `Behandler`. Redegør for eventuelle antagelser om, hvordan data kan se ud.

b) Foretag normalisering af `Behandler` til 3. normalform. Angiv det resulterende relationsskema, samt hvilke funktionelle afhængigheder, der er anvendt ved normaliseringen. Eventuelle antagelser om, hvordan data kan se ud, skal også angives.

3 SQL (30 %)

Denne opgave omhandler SQL forespørgsler på `Behandler` relationen fra opgave 2. (Bemærk at forespørgslerne skal være på `Behandler` og altså *ikke* på de normaliserede relationer fra spørgsmål 2.b.) Ved løsningen skal du bruge oplysningerne fra opgave 2 om indholdet i `Behandler`. Betragt flg. SQL forespørgsler:

```
1:      SELECT speciale
        FROM Behandler
        WHERE postnr=1000 AND sygdom='hundegalskab';

2:      SELECT speciale
        FROM Behandler
        WHERE postnr=1000 AND id IN (SELECT id
                                     FROM Behandler
                                     WHERE sygdom='hundegalskab');
```

a) Giv en kort forklaring i ord på, hvad de to forespørgsler returnerer – herunder hvad der er forskellen mellem dem.

I kurset har vi stiftet bekendtskab med SQLs `COUNT(*)` funktion, der bruges til at tælle tupler. SQL har desuden funktionen `COUNT (DISTINCT <attribut>)`, der bruges til at tælle antallet af *forskellige* værdier af en attribut.

b) Skriv en SQL forespørgsel, der returnerer en relation med tre attributter: Der skal være et tupel for hvert speciale i `Behandler` med angivelse af antallet af behandlere og antallet af forskellige sygdomme indenfor specialet.

Antag at vi nu yderligere har relationen `Patient(id,sygdom,behandler_id)`, der indeholder information om patienter, hvilke(n) sygdom(me) de har, og for hver sygdom hvilken behandler (fra `Behandler` relationen), de er henvist til.

c) Skriv en SQL forespørgsel der returnerer `id` for de patienter, der (fejlagtigt) er henvist til en behandler, som ikke ifølge `Behandler` kan behandle deres sygdom.

For at forebygge fejlagtige referencer ønskes det at erstatte `Patient` med relationen `Patient2(id,sygdom,behandler_by)`. Idéen er, at {`sygdom,behandler_by`} bruges som fremmednøgle. (Da der kun er én behandler for hver sygdom i hver by, kan man referere til en unik behandler på denne måde.)

d) Skriv en SQL kommando, der opretter relationen `Patient2`, og erklærer ovennævnte fremmednøgle. Skriv herefter en eller flere SQL kommandoer, der indsætter informationen fra `Patient` i `Patient2`, således at alle patienter beholder deres behandler(e). Patienter og sygdomme, der ikke er henvist til en kvalificeret behandler, indsættes med værdien `NULL` på attributten `behandler_by`.

4 Transaktioner (15 %)

Relationen `Seats(seatID,class,reserved)` bruges til at håndtere sædereservationer i et fly. Den indeholder er tupel for hvert sæde, og attributten `reserved` er 0 eller 1 afhængigt af, om sædet er ledigt eller reserveret. 15 minutter før afgang lukkes der for reservationer til business class ved at eventuelle ledige sæder på business class bliver overført til kunder på "economy plus". Nedenstående transaktion (skrevet i Oracle SQL) foretager overførslen, ved hjælp af to relationer `FreeBusinessSeats(seatID,reserved)` og `Upgrades(seatID)`, der bruges til at gemme mellemresultater.

```
1. DELETE FROM FreeBusinessSeats;
2. DELETE FROM Upgrades;
3. INSERT INTO FreeBusinessSeats (SELECT seatID, reserved FROM Seats
                                  WHERE class='business' AND reserved=0);
4. INSERT INTO Upgrades (SELECT *
                         FROM (SELECT seatID FROM Seats
                               WHERE class='economy plus' AND reserved=1)
                         WHERE rownum<=(SELECT COUNT(*) FROM FreeBusinessSeats));
5. UPDATE FreeBusinessSeats SET reserved=1
   WHERE rownum<=(SELECT COUNT(*) FROM Upgrades);
6. UPDATE Seats SET reserved=0 WHERE seatID IN (SELECT * FROM Upgrades);
7. UPDATE Seats SET reserved=1
   WHERE (seatID,1) IN (SELECT seatID,reserved FROM FreebusinessSeats);
```

Forklaring: De først to SQL sætninger sletter evt. gamle mellemresultater. Linie 3 indsætter de ledige business class sæder i relationen `FreeBusinessSeats`. I linie 4 udvælges reservationer på "economy plus", som skal opgraderes. Antallet af opgraderinger holdes under antallet af ledige pladser ved brug af `rownum` variablen, der returnerer nummeret på den aktuelle række. Linie 5 markerer det rette antal ledige sæder i `FreeBusinessSeats` som reserverede. I linie 6 og 7 overføres informationen om de nye reservationer til `Seats` relationen.

a) Antag at ovenstående transaktion kører på SQL isoleringsniveau `READ COMMITTED`. Argumentér for, at hvis der samtidig foretages en reservation på en business class plads (dvs. en transaktion, der ændrer en `reserved` værdi fra 0 til 1), kan der forekomme en dobbeltbooking, dvs. at antallet af reserverede sæder er mindre end antallet af passagerer.

På grund af ovenstående problem er det oplagt at overveje et højere isoleringsniveau. Vi betragter SQLs `REPEATABLE READ` og `SERIALIZABLE`, samt "snapshot isolation" beskrevet i artiklen *A Critique of ANSI SQL Isolation Levels*.

b) Overvej for hver af de tre ovennævnte isoleringsniveauer, hvorvidt der kan forekomme en dobbeltbooking. Argumentér for dine svar.

5 Indeksering (15 %)

De to forespørgsler angivet i opgave 3 refererer til attributterne `speciale`, `postnr`, `sygdom` og `id`. For at forbedre forespørgslernes køretid kan det overvejes at oprette et eller flere indekser. Denne opgave går ud på at vurdere forskellige muligheder for indeksering.

a) Vurdér for hver af de fire ovennævnte attributter effekten af at have et indeks på denne attribut som *eneste* indeks. Det skal angives hvilke af de to forespørsgler (om nogen), indekset vil gøre hurtigere. Hvad en den eller de bedste af disse muligheder, med henblik på at gøre begge forespørgsler hurtige? Argumentér for dit svar.

b) Betragt følgende fire muligheder for at lave et "composite" indeks:

- `CREATE INDEX idx1 ON Behandler(postnr,sygdom);`
- `CREATE INDEX idx2 ON Behandler(sygdom,postnr);`
- `CREATE INDEX idx3 ON Behandler(id,postnr);`
- `CREATE INDEX idx4 ON Behandler(postnr,id);`

Angiv hvilke af de to forespørgsler (om nogen), hvert indeks vil gøre hurtigere, hvis det oprettes som eneste indeks.

c) Foreslå en kombination af to indekser, der får forespørgslerne til at køre så hurtigt som muligt, samlet set. Argumentér for dit valg.