Svarsförslag tenta da127a 2018-03-15

Jesper Larsson

1.

```
select namn from rätt where vego
```

Det funkar med `where vego=true`, men det är onödigt. Det är inte rätt att jämföra med `'ja'` eller annat strängvärde (men jag har förstås inte underkänt för det).

2.

```
insert into prod_betyg (person, prod, betyg)
values (54, 'Nestlé', 4)
```

Jag har godkänt om man istället gjort en `update` som ändrar betyget för befintlig person/podukt.

3.

```
select person.namn
from (person join rätt on favoriträtt=r_id)
where vego
```

4. Tillägg:

   - i person:

     ```
     primary key (p_id),
     foreign key (favoriträtt) references rätt(r_id)
     ```

   - i rätt:

     ```
     primary key (r_id)
     ```

   - i prod_betyg:

     ```
     primary key (person, prod),
     foreign key (person) references person(p_id)
     ```

5. Här insåg jag först vid rättningen att jag tänkte fel när jag ställde frågan, vilket gjort att den blev svårare än avsett. Eftersom nyckeln i prod_betyg innehåller en del (kolumnen person) som är en främmande nyckel, och en del som inte är det (kolumnen prod) ska prod_betyg vara en *svag entitet* i E/R-diagrammet. Det är en konstruktion jag bara visat i förbigående på föreläsningen, och att behärska det ingår inte i lägsta nivån jag kräver för att godkänna. Därför har jag fått vara

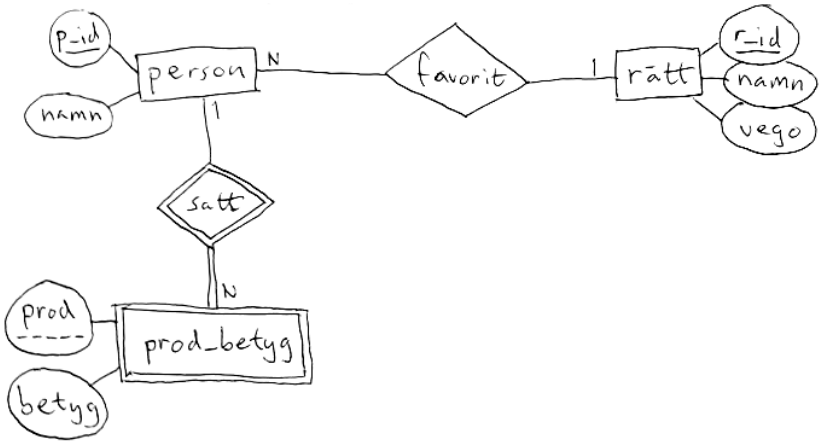
väldigt generös i rättningen av denna uppgift, och har godkänt bara man ritat ett någotsånär vettigt diagram, där någon av delarna finns med på rätt sätt.

Men oavsett det tycker jag att det är märkligt att många uppenbarligen inte satt sig in i E/R alls, inte heller tittat hur de sur ut i boken, utan ritat saker som inte följer notationen. Jag tycker att jag varit tydlig med att det *kommer* att ingå E/R bland de frågor man *ska* klara för godkänt. På omtentan kommer jag att vara noga med att formulera en fråga som inte är svårare än ni ska klara, och kan därmed vara hårdare i bedömningen.

Notera följande, som många gjort fel på:

- Samband ska ritas som en romb med en text som beskriver vad sambandet uttrycker, inte bara streck eller rektangel.
- Ett 1:N-samband motsvarar att en tabell har en främmande nyckel till en annan tabell. Då ska den främmande nyckeln som uttrycker sambandet *inte* synas som attribut också, det är redundant! Exempelvis har person-entiteten *inte* något attribut för favoriträtt, utan favoriträtt uttrycks som ett samband.

Gör man diagrammet i uppgiften helt rätt blir det så här:



6. Denna fråga tycks också ha blivit svårare än avsett, och jag har varit hyfsat generös i bedömningen.

   a) Rätt. Vi har korrekt gjort prod_betyg till ett många-till-många-samband mellan person och rätt på detta sätt.

   b) Fel. Innebär att antingen bara en person kan ha en viss rätt som favorit eller att vi får redundans i tabellen (beroende på vad vi tar som nyckel).

   c) Fel. Detta ger en massa redundans i persontabellen, och bryter mot andra normalformen.

7.

```
select namn, count(betyg)
from (person left join prod_betyg on p_id=prod_betyg.person)
group by p_id
```

Huvudpoängen här är aggregatet: count och group by. Har man missat det (om inte det bara brister i en liten del, och man är på väg mot att få det rätt) har jag satt U.

Att det står `left` före `join` innebär att det är en *outer join*, vilket behövs för att man ska få med de personer som satt 0 betyg. Har man gjort en inner join har man bara kunnat få G.

Har man helt förbisett man måste joina ihop två tabeller för att få ut de data man vill ha har jag satt G–.

8.

```
select rätt.namn
from ((person as p1 join person as p2 on p1.favoriträtt=p2.favoriträtt)
      join rätt on p1.favoriträtt=rätt.r_id)
where p1.p_id=19 and p2.p_id=42
```

Huvudpoängen här är att man behöver joina *person* med sig själv, för att få två p_id-kolumner att jämföra med 19 respektive 42 i de rader som har samma favoriträtt. Likande exempel fanns både på Blerims föreläsning och på datorövning 2 (fast vi här har komplikationen att vi behöver joina med en annan tabell också, för att få fram rättnamnet). *Ingen* som skrev tentan har klarat detta. Att jämföra samma kolumn i samma rad med två olika värden (`p_id = 19 and p_id = 42` eller `p_id = 19 or p_id = 42`) ger aldrig rätt resultat: *and* ger inga svar alls (den kan inte ha två värden samtidigt) och *or* ger favoriträtt för både person 19 och 42 utan koll om de är samma, så det ger obönhörligen U, vilket alltså alla fått.

9. Delfrågor:

   a) FFB:
      - kundnr ⟶ { namn, adress }
      - { datum, kundnr } ⟶ { arbete, pris, fakt, bet }
      - arbete ⟶ pris (detta framgår inte helt tydligt i uppgiften)

   b) Beroendet från kundnr är ett brott mot 2NF, och beroendet från arbete är brott mot 3NF/BCNF

   c) och d. Tabeller:
      - kund: (<u>kundnr</u>, namn, adress }
      - uppdrag: (<u>kundnr</u>, <u>datum</u>, arb_id, fakt, bet)
      - arbete: (<u>arb_id</u>, pris)

      Det finns visst utrymme får variation och tolkningar, man kan få VG även om man inte gjort exakt så här, och har man bara klarat av det viktigaste får man G.

10. Tabeller:

    - användare (anv_namn, namn, epost)
    - post: (id, anv [FK till användare], text, länk, tidpunkt)
    - delning: (delare [FK till användare], post_id [FK till post])
    - följer: (följare [FK till användare], följd [FK till användare])

      Även här gäller att det finns visst utrymme får variation och tolkningar, att man kan få VG även om man inte gjort exakt så här, och att man har man får man G om man bara klarat det viktigaste. Eftersom svaret som krävs är så pass litet krävs det dock inte att mycket saknas för att man ska få U.

11. Man behöver inte ge exakt nedanstående, men ska resonera rimligt kring indexen för full poäng.

    I. Det finns redan primärnyckelindex på person(p_id), så inget behöver tillföras.

    II. Tillfört index på person(favoriträtt) kan hjälpa eftersom det används för urval.

    III. Man skulle kunna tänka sig index på prod_betyd(betyg), men det är tveksamt om det gör så stor nytta, eftersom det finns så få betyg, så det är rimligt att det går snabbare att läsa hela tabellen och filtrera ut de som har betyg 4, än att hoppa in på varje plats som har 4 med hjälp av index. (Om det inte är ovanligt att man sätter betyg 4, men det är svårt att tänka sig att det skulle vara i denna kontext.)

    IV. Om det finns många olika producenter (som Findus) hjälper ett index på prod_betyg(prod) för att snabbt hämta fram rätt rader i prod_betyg. Om detta blir ett litet antal rader hjälper sedan ett index på person(favoriträtt) systemet att joina dem med person.

    Finns det få producenter lär indexet på prod_betyg inte hjälpa så mycket. Då kan man tänka sig att systemet utför joinen först. Joinen kan då göras genom att man går igenom alla personer, slår upp deras favoriträttsbetyg och filtrerar ut de rader som har rätt producent. Att slå upp personerna i prod_betyg kan *kanske* göras med befintligt primärnyckelindex, om det är ett B-träd och har kolumnerna på rätt håll (sorterat på person i första hand och prod i andra hand), annars kan det hjälpa att tillföra ett index på prod_betyg(person).

12. Delfrågor:

    a) Det förefaller rimligast att göra det som en transaktion tillsammans, för att utesluta utnyttjande av tillfälliga anomalier som beror på att delar av en uppdatering gått igenom. Men det kan samtidigt innebära att det tar längre tid innan en uppdatering slår igenom, vilket också är dåligt, så helt hundra procent säkert är det inte att det är att föredra.

    b) En högre isoleringsnivå (serializable är den högsta) gör att risken minskar för att någon ser delar av uppdateringar, men kan samtidigt göra att transaktionerna tar längre tid att få igenom. Fördelar och nackdelar analogt med (a). Varje update som en egen transaktion kan ju betraktas som ekvivalent med lägsta möjliga isoleringsnivå.

c) Ren låsning innebär att man ofta kan få vänta på att andra transaktioner ska bli klara, alltså sämre *throughput* med oönskade effekter. Å andra sidan kan flerversionsmetod innebära att en transaktion kan tillåtas använda ett ganska gammalt värde (istället för att den låses och väntar), vilket också kan vara farligt.