Tentamen i DATABASER

Svar:

Obs! Lärarversion, med lösningar

DAG: 18 December, 2001 **TID:** kl. 14.15 – 18.15 **PLATS:** V-huset

Ansvarig:	Patrik Jansson
Förfrågningar:	Patrik Jansson, ankn. 5415
Resultat:	anslås den 15 Januari, 2002
Poängantal:	sammanlagt maximalt 60 poäng.
Betygsgränser:	CTH: 3:a 24 p., 4:a 36 p., 5:a 48 p. GU: Godkänd 28 p., Väl godkänd 48 p. Doktorander: Godkänd 28 p.
Hjälpmedel:	Ett A4-blad (båda sidor får användas) med valfritt innehåll får medföras och skall i så fall lämnas in med tentan. (Skriv då namn och personnummer även på detta blad.)

Observera:

- Skriv tydligt och disponera pappret på ett lämpligt sätt.
- Börja varje uppgift på nytt blad. Skriv endast på en sida av pappret.
- Alla svar skall **motiveras väl** och ej vara onödigt komplicerade!
- Ange på tentan om du går på GU eller Chalmers och vilken linje/program du går.
- Det är denna gång samma tentamen för GU och för Chalmers.

Lycka till!

Uppgift 1. Transaktioner och distribuerade databaser.
14p

(a) Ange två valfria transaktioner och en serialiserbar operationsföljd för dessa två transaktioner (inklusive `begin`, `commit` och alla lås) som leder till deadlock med vanlig tvåfaslåsning.

Svar: Det finns många rätta lösningar (och ännu fler felaktiga). För att det skall bli deadlock måste minst två objekt vara inblandade i båda transaktionerna — låt oss kalla dem A och B och låt båda transaktionerna skriva till båda dessa:

```
1: B, WA, WB, C
2: B, WB, WA, C
```

Att få operationsföljden serialiserbar kan exempelvis lösas genom att bara fläta samman låsningarna och låta resten av operationerna utföras seriellt. (Ett annat sätt att garantera serialiserbarhet är att inte ta med några skrivoperationer i transaktionerna.)

Med konservativ tvåfaslåsning (lås allt i *en* operation i början) kan man inte få deadlock, men om vi låser A och B i två separata operationer och i olika ordning så får vi deadlock med exempelvis denna operationsföljd:

```
1B, 2B, 1LA, 2LB, 1LB, 2LA -- deadlock
1WA, 1WB, 1UA, 1UB, 1C,
2WB, 2WA, 2UB, 2UA, 2C
```

(b) Beskriv vertikal fragmentering med uttryck i relationsalgebra + förklaring samt ge två orsaker till varför man vill använda det.

Svar: Dela upp en relation i olika projektioner. $\pi_M(r) \bowtie \pi_N(r) = r$

Hemlighet: vissa (hemliga) attribut lagras på en nod, offentliga attribut lagras på en annan nod.

Lokalitet: attribut som ändras ofta från en viss nod lagras där. (Lön lagras hos personalavdelningen, kurs-lärare-information hos studievägledaren.)

Normalformer: undvik redundans genom att dela upp till normalform.

mm.

(c) Beskriv horisontell fragmentering med uttryck i relationsalgebra + förklaring samt ge två orsaker till varför man vill använda det.

Svar: Dela upp en relation i olika selektioner. $\sigma_v(r) \cup \sigma_{\texttt{not } v}(r) = r$

Utrymme: hela relationen får inte plats på en nod.

Lokal information: lagra humanistiska kurser på humanistisk fakultet, tekniska kurser på teknisk fakultet.

Effektivitet: balansera last mellan olika noder.

mm.

Ett homogent distribuerat databassystem har replikerat samma data på 11 mindre noder med vikt 1 (Hob1 till Hob11) och på två större noder: Frodo med vikt 4 och Gandalf med vikt 5.

(d) Kvorum-konsensus: Man vill att systemet skall kunna uppdateras även om Frodo är nere. Ange lämpliga läs- och skrivkvorum för att ge systemet så stor lästillgänglighet som möjligt med detta krav uppfyllt. Kan systemet då läsas om bara Gandalf är uppe?

Svar: Stor lästillgänglighet betyder lågt läskvorum, så vi vill minimera Q_r. Första kravet kan skrivas som $Q_w < 1 - 4/20$. Allmänt måste gälla $Q_w + Q_w > 1$ och $Q_w + Q_r > 1$. Efter förenkling får vi $1/2 < Q_w < 4/5$ och $Q_r > 1 - Q_w$. Att minimera Q_r är samma som att

maximera Q_w. Välj exempelvis $Q_w = 0.78$ och $Q_r = 0.22$. Då är $Q_r < 5/20$ så system kan läsas även om alla hober (inklusive Frodo) är nere, bara Gandalf är uppe.

(e) Två-fas-commit: Gandalf, Frodo, Hob3 och Hob7 är inblandade i en distribuerad transaktion där Gandalf är koordinator. (Det hade visst något med en magisk ring att göra ;–) Beskriv den kommunikation som behövs när transaktionen skall avslutas med `commit` i tre olika fall: om allt går bra, om Frodo kraschar direkt efter fas 1, och om Frodo inte svarar alls.

Svar: Gandalf skickar `PREPARE` till de övriga (agenterna). Agenterna skickar `READY` till Gandalf. När Gandalf fått in alla svar skickar han `COMMIT` till alla. Agenterna svarar med `ACK`.

Gandalf skickar `PREPARE` till agenterna. Alla agenter skickar `READY` till Gandalf. Frodo kraschar. När Gandalf fått in alla svar skickar han `COMMIT` till alla. Agenterna svarar med `ACK` när de är klara. När Frodo vaknar igen frågar han Gandalf vad som hände, Gandalf säger `COMMIT` och Frodo lyder och avslutar med `ACK`.

Gandalf skickar `PREPARE` till agenterna. Agenterna skickar `READY` till Gandalf, men Frodo svarar ej. När Gandalfs tålamod tar slut skickar han `ROLLBACK` till alla. Agenterna svarar med `ACK` när de är klara.

Uppgift 2. Representera och söka i släktträd.

10p

- Varje Person har ett unikt ID-nummer, ett namn och ett födelsedatum.
- En länk i sambandet Född kopplar samman tre personer: ett barn, dess biologiska mamma och pappa.
- Ett barn har högst en mamma och högst en pappa.
- Vissa personer är kvinnor, andra är män.

Det sista påståendet kan representeras i ER-diagram på (minst) två sätt: (a) med hjälp av ett attribut Kön och (b) med hjälp av undertyper Kvinna och Man till Person.

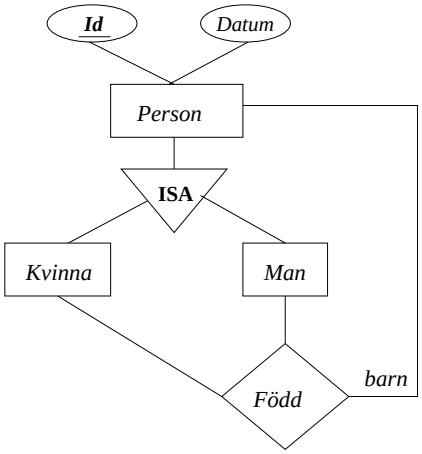
(a) Rita ER-diagram enligt (a) och översätt det till relationer med nycklar och referenser i samma stil som i appendix.

(b) Rita ER-diagram enligt (b) och översätt det till relationer med nycklar och referenser i samma stil som i appendix.

(c) Definiera (i SQL eller relationsalgebra) en vy *kusin(Barn, Kusin)* som ger varje kusinpar en gång. Två olika personer anses vara kusiner om de är barnbarn till samma person och inte är syskon. Se till att ta bort dubbletter och att bara ta med en av *(x,y)* och *(y,x)* i resultatet. Tips: definiera först vyerna *barn(Barn, Förälder)* samt *syskon(Barn, Syskon)*.

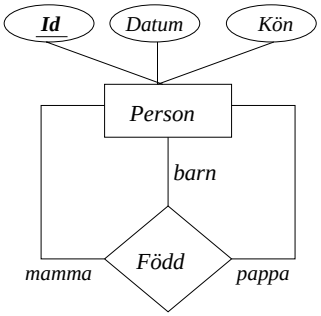
Svar:

a.



född(Barn, Mamma, Pappa)
född.Barn → person.Id
född.Mamma → person.Id
född.Pappa → person.Id
person(Id, Namn, Datum, Kön)

b.



född(Barn, Mamma, Pappa)
född.Barn → person.Id
född.Mamma → kvinna.Id
född.Pappa → man.Id
person(Id, Namn, Datum)
kvinna(Id)
kvinna.Id → person.Id
man(Id)
man.Id → person.Id

```
create view barn(B, F) as
  select Barn, Mamma from född union
  select Barn, Pappa from född;
create view syskon(B, S) as
  select distinct x.B, y.B
    from barn x  join  barn y  using (F)
   where x.B < y.B;
create view kusin(B, K)
  as select distinct x.B, z.B
       from barn x, syskon y, barn z
      where x.F=y.B and y.S=z.F;
```

Uppgift 3. Normalisering av skolregister

20p

För att underlätta administrationen vid en skola ska ett register upprättas. I registret ska finnas uppgifter om Betyg, Elever, Kurser, Lärare, Rum och Tider (alla förkortas till sin första bokstav). Ett sätt är att lagra all information i en enda universell tabell: BEKLRT. (Så här långt bör alla känna igen sig från laboration 1 men här börjar olikheterna.)

Följande (mer universitetslika) funktionella beroenden förutsätts gälla:

EK $\rightarrow$ B	En elev har högst ett slutbetyg på en viss kurs.
LT $\rightarrow$ RK	Varken lärare eller elever kan befinna sig i mer än
ET $\rightarrow$ RK	ett rum (eller gå mer än en kurs) vid samma tidpunkt.
RT $\rightarrow$ KL	Högst en lärare (och en kurs) samtidigt i samma rum

Det finns ingen närvaroplikt så elever kan få betyg på två kurser som har krockande lektioner.

Några beroenden som inte antas gälla är

K $\not\rightarrow$ L	Många kurser har mer än en lärare
KT $\not\rightarrow$ L	Flera lärare på en kurs kan köra parallella
KT $\not\rightarrow$ R	sessioner i olika rum vid samma tidpunkt.

Uppgifter:

(a) Ange alla kandidatnycklar för BEKLRT.

(b) Ange en förlustfri, beroendebevarande uppdelning av BELKRT till tabeller i BCNF. För varje deltabell skall du ange alla kandidatnycklar, alla främmande nycklar och alla icke-triviala funktionella beroenden. Det är OK att bryta en referens, men inte OK att bryta något enda beroende.

Svar:

```
BEKLRT

1. EK -> B      -- ej BCNF, 3NF
2. LT -> KR     -- ej BCNF, 3NF
3. ET -> KR     -- BCNF
4. RT -> KL     -- ej BCNF, 3NF

Dela upp enligt 1:

  ELKRT, ET nyckel, 2+3+4, fr. n. EK till EKB(EK)
  EKB,   EK nyckel, 1

  Fortfarande är 2,4 ej normala, men 3 är BCNF.

Dela upp enligt 2: (bryter fr.n.)

  ETL,   ET nyckel, 5=ET->L, fr.n. LT till LTKR(LT)
  LTKR,  LT nyckel, RT unik, 2+4
  EKB,   EK nyckel, 1

Alla tre är på BCNF.

Beroenden:
  direkt synliga:
    5. ET->L
```

```
    2. LT->KR
    4. RT->KL
    1. EK->B
  frågan är då endast om 3. ET->KR följer från dessa.
  Men detta följer av pseudotransitivitet med 5 + 2.
```

(c) Implementera dina tabeller inklusive nycklar och referenser i SQL (med `create table`).

Svar:

```
create table LTKR
  ( L varchar(15)
  , T varchar(8)  not null
  , K varchar(15)
  , R varchar(2)  not null
  , primary key (L,T)
  , unique (R,T)
  );
create table ETL
  ( E varchar(20)
  , T varchar(8)  not null
  , L varchar(15)
  , primary key (E,T)
  , foreign key (L,T) references LTKR(L,T)
  );
create table EKB
  ( E varchar(20)
  , K varchar(15)
  , B number(1)
  , primary key (E,K)
  );
```

(d) Definiera (i relationsalgebra eller i SQL) en vy *krock(E,T,K1,K2)* som ger alla elever e och tider t då e deltar i en lektion (i kursen $k1$) som krockar (i tid) med en annan lektion (i kursen $k2$). Eleven skall ha betyg i båda kurserna.

Dela upp uppgiften i lämpliga mindre delar (vyer) och lös (definiera) dessa var för sig samt kombinera delarna till svaret.

Svar:

```
-- vilka elever har betyg i två olika kurser
create view tvåkurser(E,K,K2) as
  select E, x.K, y.K from EKB x join EKB y using (E);

-- vilka elever går en viss tid på en viss kurs
create view ETK(E,T,K) as
  select x.E, T, y.K from ETL x natural join LTKR y;

-- vilka tider är det lektioner i en viss kurs
create view KT(K2, T) as
  select K, T from LTKR;

-- vilka tider går en elev (som också går på kursen K2) på lektioner i K
```

```
create view gårpåen as
  select E, T, K, K2 from tvåkurser natural join ETK;

-- vilka tider krockar?
create view krock(E,T,K1,K2) as
  select gårpåen.E, T, gårpåen.K, K2 from gårpåen natural join KT where K <> K2;
```

Uppgift 4. ER ↔ relationer

10p Översätt "baklänges" från ESC-databasen (se appendix) till ett ER-diagram. Kontrollera att ditt diagram översätts till precis de scheman, nycklar och referenser som beskrivs i appendix. Tips: *participation* blir i ER-diagrammet ett samband mellan *tre* olika entitetstyper.

Uppgift 5. Sökfrågor

6p Använd ESC-databasen (se appendix) och skriv sökfrågor i SQL eller relationsalgebra som visar

(a) titel, land och år för alla sånger som har tävlat på hemmaplan.

(b) alla länder som någonsin tävlat, samt deras bästa, deras genomsnittsliga och deras sämsta placeringar.

(c) titel, land och år för alla sånger som har vunnit när de har startat som nummer 1.

(d) titel, land och år för alla sånger som har vunnit när de har startat sist.

Svar:

```
(a)
select Title, Country, Year
  from song natural join year;

(b)
select Country, min(Place), avg(Place), max(Place)
  from song
 group by Country;

(c)
select Title, Country, Year
  from Song
 where Place = 1 and Startno = 1;

(d)
select Title, Country, Year
  from song natural join
       (select Year, max(Startno) No_of_songs
          from song
         group by Year)
 where Startno = No_of_songs and Place = 1;
```

Appendix: Eurovision Song Contest (ESC)

ESC-databasen har följande struktur (relationer, nycklar och referenser). Notera att det skiljer sig lite från beskrivningen för laboration 3.

country(Country)	Alla länder som deltagit i ESC.
person(Name, Sortname)	Alla personer som varit involverade i ESC.
task(Task)	Alla uppgifter: textförfattare, kompositör, sångare, osv.
host(Year, Name) *host.Year* → *year.Year* *host.Name* → *person.Name*	Alla programledare och när de var med och ledde ESC.
song(Year, Startno, Title, Place, Points, Country) *song.Year* → *year.Year* *song.Country* → *country.Country*	Alla sånger som har varit med i ESC, med data om hur det gick och vilket land som framförde den.
participation(Year, Startno, Name, Task) *participation.*{*Year, Startno*} → *song.*{*Year, Startno*} *participation.Name* → *person.Name* *participation.Task* → *task.Task*	Alla personer som medverkat med någon uppgift i någon sång.
year(Year, Date, City, Venue, Country) *year.Country* → *country.Country*	Alla år som ESC har gått samt när och var.