Rapport

HÖGSKOLAN I SKÖVDE

INLÄMNINGSUPPGIFT1

Frågespråk, Normalisering & Lagring

Databassystem G1N

Business Intelligence

Anton Karlsson

2021-03-12

Innehåll

Block 1 Frågespråk

Person(Pnr (PK), Namn)
Äger(personPnr(PK/FK), beteNamn(PK/FK))
Bete (Namn(PK), Färg, Typ)
Spö(Namn(PK), Längd, personPnr(FK))
Rulle(Namn(PK), Typ, spöNamn(PK/FK), personPnr(PK/FK))
Fisk(Namn(PK), Typ)
Fångst(Datum(PK), Väder, fiskNamn(FK), beteNamn(FK) , personPnr(PK/FK))
Ätit(personPnr(PK/FK), fiskNamn(PK/FK))

1. **Hämta personnumret för personen som heter ”Svartzonker”.**
   Π person.pnr σ person.namn = ’Svartzonker’ (Person)

   I relationsalgebra skrivar man först ut med hjälp av pi tecknet vad exakt man vill selecta sedan ” σ” för att ställa krav på att person.namn är likamed Svartzonker sedan skrivar man från vilken tabell i fråga detta gäller som alltså är Person.

2. **Vilken längd har spöet som tillhör personen som heter ”Evelina”?**
   Π spö.längd σ personPnr.spö=person.pnr and person.namn = ’Evelina’ (Spö x Person)

   Vi måste härmed joina person och spö tabellen för att korrekt kunna utvinna den information vi vill, alltså skriver vi ut personPnr.spö är lika med person.pnr för att joina de två tabellerna. Sedan skriver vi ut kravet att person.namn ska vara Evelina samt avslutar med vilka tabeller som används i parentesen.

3. **Hämta typen på rullen som tillhör personen som heter ”Svartzonker”**
   Π rulle.typ σ spöNamn.rulle=namn.spö and personPnr.spö=person.pnr and namn.person= ’Svartzonker’ (Rulle x Spö x Person)

   Vi följer en samma princip som förra frågan fast här får vi joina tre stycken tabeller rulle, spö och person genom deras främmande nycklar för att referera till tabellerna som ska slås ihop.

4. **Hämta alla datum för fångsterna som avser alla fiskar av typen ”Predator” och som togs på ett grönt bete av typen ”Spinnare”.**
   Π fångst.datum σ bete.namn=fångst.beteNamn and fisk.namn=fångst.fiskNamn and bete.färg=’grönt’ and bete.typ=’spinnare’ and fisk.typ=’predator’ (Fångst x Bete x Fisk)

   Återigen får vi slå ihop våra tabeller för att kunna utvinna den information vi behöver korrekt, vi slår ihop bete med fångst genom beteNamn, fisk med fångst genom fiskNamn sedan kravställningen bete.färg ska vara grönt, bete.typ ska vara spinnare och fisk.typ ska vara predator.

5. **Hämta namnet på de personer som fångat en fisk och som äger en rulle som heter ”Ambassadeur 5601 JB”.**

   Π person.namn σ person.pnr=spö.personPnr and spö.personPnr = rulle.personPnr and fångst.personPnr = person.pnr and rulle.namn =’Ambassadeur 5601JB’ and fångst.personPnr =’is not null’

6. **Hämta typen på de fiskar som aldrig ätits av någon person.**
   Alla_fiskar← Π fisk.namn (Fisk)
   Alla_ätnafiskar← Π ätit.fiskNamn (Ätit)
   Ej_ätna ← Alla_fiskar-Alla_ätnafiskar
   Resultat← Π fisk.typ σ fisk.namn = Ej_ätna.fiskNamn (Ej_ätna x Fisk)

   För att lösa denna uppgiften behöver vi skapa några temporära tabeller, vi vill ha reda på alla fiskar och sedan alla ätna fiskar för att kunna subtrahera alla ätna fiskar från alla fiskar för att få reda på vilka som EJ är ätna. Sedan så hämtar vi fisk.typ där kravet är att fisk.namn är lika med Ej_ätna.fiskNamn som är den nya temporära tabellen för att joina dessa två för att kunna få ut informationen om fisk.typ från dessa fiskar i Ej_ätna tabellen.

7. **Hämta namnet på personerna som äger alla beten.**
   Alla_beten ← Π bete.namn (bete)
   Alla_ägda_beten ← Π äger.personPnr, äger.beteNamn (äger)
   Äger_alla_beten← Alla_ägda_beten / Alla_beten
   Resultat← Π person.namn σ person.pnr=äger_alla_beten.pnr (person x Äger_alla_beten)

   Här följer vi en snarlik princip som i uppgift 6, vi tar alla beten i en temporär tabell, samt alla ägda beten, sedan i tabellen Äger alla beten så dividerar vi Alla_ägda_beten med Alla_beten för att få fram om någon då äger alla beten, eftersom personnummer attributet här är unionskompatibelt så funkar divisionen, precis som i föregående uppgift, fast då användes subtraktion.

8. **Hämta färgen på beten vid namn ”Westins Jätte” och som har fångat en fisk vid namn ”Gädda”.**

   Π bete.färg σ fångst.beteNamn = bete.namn and bete.namn = ’Westins Jätte’ and fångst.fiskNamn =’Gädda’ (bete x fångst)

   Här selectar vi bete.färg och joinar fångst med bete genom beteNamn och namn samt sätter krav att bete.namn ska vara Westins Jätte och att fångst.fiskNamn ska vara Gädda. Vi använder oss utav tabelllerna bete och fångst som skrivits ut i parentesen.

9. **Hämta namnet på personen som ätit alla fiskar som är av typen ”Predator”**

   Alla_personer ← Π person.pnr (person)
   Alla_ätna_fiskar ← Π ätit.personPnr, ätit.fiskNamn (ätit)
   Ätit_alla_fiskar← Alla_ätna_fiskar / Alla_personer
   Resultat← Π person.namn σ person.pnr=Ätit_alla_fiskar.personPnr and person.pnr = ätit.personPnr and fisk.namn = ätit.fiskNamn and fisk.typ = ’Predator’ (person x Ätit_alla_fiskar x fisk x ätit)

   En liknande princip som fråga 7 fast här ska vi ta fram personen som ätit alla fiskar av typen Predator, vi börjar med att skapa Alla_personer genom person.pnr och sedan Alla_ätna_fiskar med ätit.personPnr, ätit.fiskNamn. Vi måste ha med personPnr för att ha ett unionskompatibelt attribut för divisionen. Vi har även med fiskNamn för att kunna få med informationen om vilka fiskar som är ätna. Därefter använder vi oss utan den nya tabellen Ätit_alla_fiskar i den slutliga frågan. Där vi joinar person och ätit genom pnr och fisk med ätit genom fiskarnas namn. Slutligen ett krav på att fisk.typ ska vara ”Predator”.

10. **Hämta typen på betet som aldrig fångat någon fisk.**

    Alla_beten← Π bete.namn (bete)
    Alla_beten_vid_fångst← Π fångst.beteNamn (fångst)
    Alla_beten_utan_fångst ← Alla_beten-Alla_beten_vid_fångst
    Resultat← Π bete.typ σ bete.namn = Alla_beten_utan_fågnst.beteNamn and fångst.beteNamn = bete.namn (Alla_beten_utan_fågnst x bete x fångst)

    Här följer vi också en tidigare princip som användes i fråga 6, för att få reda på alla beten som inte använts vid fångst så måste vi ta alla beten minus de som använts vid fångst för att då få kvar de som inte använts vid fångst (utan fångst). Sedan selectar vi bete.typ för att besvara frågan, vi joinar tabellerna genom fångst.beteNamn = bete.namn och bete.namn = Alla_beten_utan_fångst.beteNamn, då när alla tabeller är joinade så kan vi ställa våran fråga och få ett korrekt svar.

Block 2 Normalisering

Uppgift 1

1. **Visa med ett eget exempel vad som är skillnaden mellan en supernyckel, en kandidatnyckel och en primärnyckel.**

   Exempel Medlem (<u>Medlemsnummer</u>, person_namn, mobilnr)
   Supernycklar får innehålla onödigt många attribut, det är med hjälp av en unik nyckel eller unik nyckel i kombination med andra attribut i relationstabellen som är garanterat unika tillsammans.
   I Medlem relationen finns det fyra stycken:
   Medlemsnummer

Medlemsnummer+person_namn
Medlemsnummer+mobilnr
Medlemsnummer+person_namn +mobilnr

Dessa är supernycklar så det garanterat inte kommer att finnas någon annan med ens medlemsnummer samt namn eller mobilnr, dessa blir då en supernyckel.
Det finns dock alltid minst en supernyckel och minst en kandidatnyckel i varje relation. Alla attribut tillsammans blir alltid en supernyckel. Alltså är alla attribut tillsammans en garanterat unik kombination, och då en supernyckel.

Kandidatnycklar är de nycklar som är eventuella ”kandidater” för att vara primärnyckel, i vissa fall kan man ha fler än ett unikt attribut som kan vara ens primära nyckel, men slutligen bör endast ett attribut väljas som blir den primära nyckeln i relationen.

Exempel Anställd (Personnr,anstlNr,namn)
En anställd har ett unikt Personnr och anställningsnummer, även om namn kan vara unikt, så bör först och främst Personnr och anställningsnummer vara kandidatnycklar, beroende på hur övriga relationer ser ut bör man välja den som passar in bäst för att enklast få unionskompatibla relationer. Så används Personnr i flera andra relationer så bör den väljas som den primära nyckeln.

2. **Vi har under kursen lyft att redundans vanligtvis inte är något att eftersträva. Dock finns det undantag där redundans kan vara något positivt. Förklara detta resonemang, alltså när redundans ska undvikas respektive när redundans kan introduceras i en databas.**

Redundans är vanligtvis onödig överskottsinformation som bör tas bort för att det tar extra plats i databasen och kan ställa till det om man tar bort information på ett ställe så ligger det kvar i en annan tabell. Men ibland kan det vara positivt då det förtydligar information i vissa sammanhang och säkrar informationen i databasen.

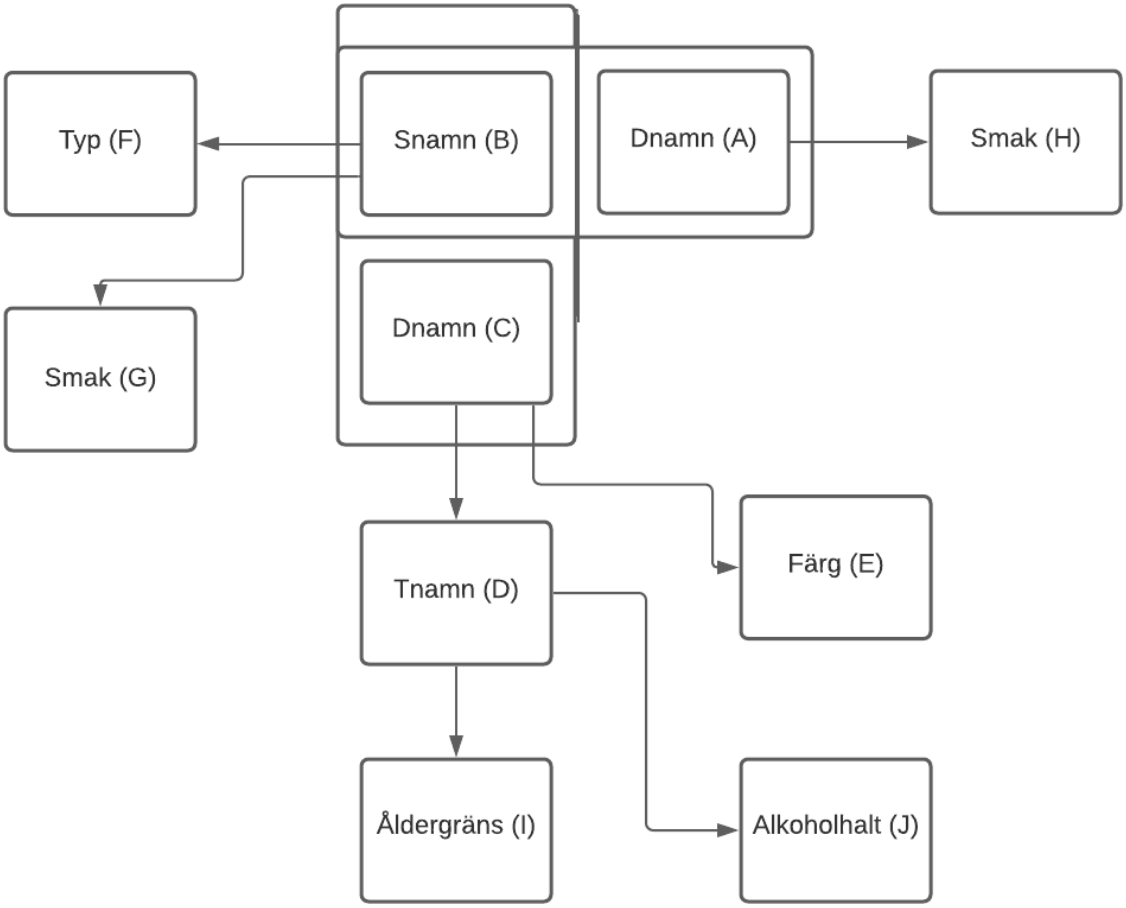
Så vill man ha förtydligande information i databasen bör kontrollerad redundans användas, då man har kvar redundans i syftet att sökningar då kommer ge överflödig information men den kommer hjälpa till att snabbt säkerställa att informationen är korrekt.
Har man redundans utan att veta om det eller gjort det på ett bra och strukturerat sätt så är det okontrollerad redundans.

Det är inte enbart i databaser redundans bör undvikas, till exempel i lagstiftning så försöker man undvika redundans då två lagar inte ska beskriva samma sak, de kan då vara en större risk för motsägelser, nackdelen är att det kan bli mer svårläst.

Redundans används mer flitigt i system som måste ha en hög tillförlitlighet, till exempel i tågklarering, om de datorerna som sköter kommunikationen inte visar samma resultat vid uträkningar så stoppas all trafik. Alltså sköter ett flertal datorer, exempelvis tre stycken samma uträkningar och kommunikation, vilket är hög redundans, men det är i detta fall något bra, då säkerheten att kommunikation är helt korrekt är extremt viktig.

Uppgift 2

1. **Beskriv följande domänbeskrivning med hjälp av en modell över funktionella beroenden**



Först och främst tar vi de attribut som är identifierare och visar med lådor och pilar vilka attribut de kan ge oss (peka på) enligt den angivna beskrivningen.

2. **Överför därefter modellen till relationer i 1NF, 2NF, 3NF och 4NF. Förklara för varje normalform vad den innebär.**
   Efter att modellen har ritats upp följer vi *Högskolan i Skövde* (2021) hänvisningar hur man räknar ut normalformerna, se nedanstående uträkningar.

   1NF: R (A, B, C, D, E, F, G, H, I, J) Relationen har en fungerande nyckel.

   2NF: R1(AH) Varje attribut är fullt funktionellt beroende av hela nyckeln i relationen.
   R2(BFG)
   R3(CEDIJ)
   R4(AB C)

3NF: R1(<u>A</u>H) Ingen transivitet.
R2(<u>B</u>FG)
R31(<u>C</u>ED)
R32(<u>D</u>IJ)
R4(<u>ABC</u>)

4NF: R1(<u>A</u>H) Inga icke triviala multivärda funktionella beroenden.
R21(<u>B</u>F)
R22(<u>B</u>G)
R31(<u>C</u>D)
R32 (<u>C</u>E)
R41(<u>D</u>I)
R42(<u>D</u>J)
R5(<u>ABCD)</u>

Uppgift 3 Beskriv i vilken normalform (1NF, 2NF, 3NF, BCNF, 4NF) följande relationer befinner sig. Var noga med att motivera i text varför en viss normalform gäller. Svar utan motivering ger per automatik U oavsett om svaret är korrekt eller ej.

A. R (<u>A,</u> B, C, D)
F = {A→B, B→C, C→D}
Svar: (2NF) Relationen uppfyller 1NF eftersom nyckeln är okej, den är även i 2NF eftersom nyckeln är minimal. Relationen är inte 3NF eftersom det finns transitivitet för att B pekar på C och C pekar sedan på D (Icke nyckel är beroende av icke nyckel).

B. R (<u>A</u>, <u>B</u>, C, D)
F = {A→B, B→CD}
Svar: (1NF) Uppfyller 1NF eftersom nyckel är okej, uppfyller inte 2NF eftersom nyckel inte är minimal.

C. R (<u>A</u>, <u>B</u>, C, D)
F = {A→C, A→D, AB→D)
Svar: (BCNF) uppfyller 1NF eftersom nycklar är okej, uppfyller 2NF då nyckel är minimal, 3NF uppfylls då inga icke-nycklar pekar på andra icke-nycklar. Uppfyller också BCNF eftersom inga nyckelattribut är beroende av ett icke-nyckelattribut. Uppfyller inte 4NF eftersom ett attribut är beroende av ett multivärdesattribut.

D. R (<u>A</u>, B, C, D)
F = {A→B, A→CD, B→A}
Svar: (2NF) Uppfyller 1NF eftersom nyckel är okej, 2NF eftersom nyckel är minimal, 3NF uppfylls inte då en icke nyckel pekar på en nyckel (Nyckel är beroende av en icke nyckel).

E. R (<u>A</u>, B, C, D, E)
F = {AB→C, B→E, C→D}
Svar: (UNF) UNF eftersom 1NF inte uppfylls, för att nyckel är ej okej.

Block 3 Lagring

1. **Låt säga att en databastabell består av en primärnyckel och 3st attribut. Om olika index hade använts på primärnyckel och alla attribut, vilka nackdelar kan det medföra?**

   Index har flera nackdelar, först så använder de utrymme. Detta kan vara obetydligt, men om ditt table är väldigt stort kan det ha en inverkan. För det andra, och ännu viktigare, måste du komma ihåg att index har en prestandaförlust när det gäller INSERT av nya rader, DELETE av gamla eller UPPDATE av befintliga värden i den indexerade kolumnen.

2. **Det finns många olika typer av index som alla har olika syfte och karaktär. Ange vilka typer av index som diskuterats under kursens föreläsningar och kurslitteratur. Beskriv även vad som kännetecknar dessa index.**
   Primärindex – sorterande i samma ordning som primärnyckeln i huvudfilen.
   Grupperande (clustered index) – sorterande i samma ordning som huvudfilen, men på ett eller flera fält som inte är en nyckel.
   Sekundärindex – som är sorterande i en annan ordning än huvudfilen. Kallas även oklustrade index.
   Tätt index (dense index) – Är ett index där varje datapost motsvaras av en indexpost.
   Glest index (sparse index) – Är ett index där varje datapost inte behöver motsvaras av en indexpost.

3. **Beskriv vad som kännetecknar dynamisk och statisk hashing. Ditt svar ska även inkludera vad som skiljer dynamisk och statisk hashing åt.**
   Statisk hashning: är en teknik som tillåter användare att utföra sökningar på ett färdigt uppslagsverk (alla objekt i uppslagsverket är färdiga och kommer ej att ändras). Resultatets dataskopans adress är därför alltid densamma. Statisk hashning är inte lika effektiv som dynamisk hashning.

   Dynamisk hashning: är en teknik där data buckets (dataskopor) blir adderade och borttagna dynamiskt och vid behov. Dataskoporna ändras beroende på records, dynamisk hashning är mera effekt än statisk hashning.

   I korthet är hashing en metod som använder matematiska funktioner som kallas hash-funktioner för att beräkna direkta platser för dataposter på skivan. Dessutom är statisk och dynamisk hashing två typer av hash. Huvudsakliga skillnaden mellan statisk och dynamisk hashning är att i den statiska hashen är den resulterande datatrådsadressen alltid densamma samtidigt som dynamisk hashning ökar eller krymper dataskoporna i enlighet med ökningen och minskningen av poster.

4. **Antag att det finns 4.000st rader lagrade på en disk med blockstorleken 1024 bytes. Varje rad har storleken 256 bytes. Hur många block måste då genomsnittligen genomsökas vid**

en linjärsökning samt en binärsökning för att hitta ett specifikt värde? Beskriv även kortfattat hur du kom fram till din lösning.

Uträkning BFR = 1024/256= 4 (4st rader per block.)
Uträkning antal block: 4000 / 4 = 1000 (1000st block.)
Linjärsökningar tar man hälften av alla block.
Uträkningen hälften av alla block: 1000/2 = 500st block accesser innan värdet är funnet.
Svaret på frågan är 500st i genomsnitt för att hitta ett specifikt värde med hjälp utav linjärsökning.

Svar: 2^{10}=1024, 10 st blockaccesser måste genomsnittligen sökas igenom för att hitta ett specifikt värde.

För att beräkna genomsnittet för hur många block som måste accessas används 2-logaritmen log2N. I praktiken innebär det att hitta hur många gånger man måste multiplicera 2 med sig själv för att resultatet ska komma upp i de antal block som ska sökas igenom (1000st). Det kan vara svårt att använda huvudräkning för att lösa detta. I matrisen (på uppgiftsbladet) letar vi efter de värde som ligger närmast över antal block som ska sökas igenom. I detta fall 1000st. 2^{10} = 1024, vilket innebär att 10st blockaccesser måste genomsnittligen sökas igenom för att hitta ett specifikt värde.

5. **I en databas med 14.000st rader lagrade på en disk med blockstorleken 1024 bytes där varje rad har storleken 128 bytes. Ett primärindex med radstorleken 32 bytes används. Hur många block måste du genomsnittligen genomsökas för att hitta ett specifikt värde? Använd gärna nedanstående matris som hjälp för att hitta svaret för din beräkning.**

BFR: 1024/32=32
Antal block: 14 000/32=437,5
2^9 = 512
Det innebär att det krävs 9st accesser i genomsnitt för att hitta det sökta värdet. Dessutom måste vi lägga till en blockaccess eftersom indexet pekar till ett block som innehåller alla rader. Resultatet på frågan är alltså 9+1 = 10st blockaccesser i genomsnitt för att hitta ett specifikt värde.

Referenser

Padron-McCharty, T. Risch, T (2005). *Databasteknik.* Lund: Studentlitteratur.