Databasteori

Övningar

Eva L. Ragnemalm

November 2009, senast reviderad april 2025

Lösningsförslag och facit till några av övningarna finns på kursens Lisam-rum. Du är också välkommen att visa dina lösningar och diskutera varianter med mig.

Boken som refereras till i texten är kursboken, Databasteknik, av T. Padron-McCarthy och T. Risch, Studentlitteratur, 2018.

Innehåll:

Övningar på ER-modellering

Skapa ER-diagram för nedanstående övningar (läs om ERmodeller i boken). Övningarna kräver inte EER-komponenter. Många övningar går att lösa på flera sätt (ibland kan man variera små detaljer, ibland radikalt olika strukturer). Fundera på vilka egenskaper de olika lösningssätten har (oftast får systemet olika begränsningar). För att bedömas som korrekt ska din lösning kunna representera allt som nämns i uppgiften.

1. Företaget

Antag att ett företag med s.k. matrisorganisation (personalen är anställd på avdelningar, som har avdelningschefer, men arbetet organiseras i projekt där man plockar in folk från olika avdelningar efter behov) behöver hjälp att hantera information om sina anställda för lönehantering och arbetsplanering, samt att hantera de löneförmåner i form av familjeförsäkringar som de anställda har. Avdelningschefers lön är delvis beroende av hur länge de varit chefer för sina avdelningar.

Datakrav:

Företaget består av ett antal avdelningar. Varje avdelning har ett namn, ett nummer, en chef och ett antal anställda. Startdatum för varje avdelningschef registreras. En avdelning kan ha flera lokaler. Lokaler identifieras endast med sina namn.

Varje avdelning finansierar ett antal projekt. Varje projekt har ett namn, ett nummer och en lokal där man arbetar.

För varje anställd lagras följande information: namn, personnummer, adress, lön och kön.

En anställd jobbar för endast en avdelning men kan jobba med flera projekt som kan tillhöra olika avdelningar. Information om antalet timmar (per vecka) som en anställd planeras jobba med ett projekt lagras. Facket har drivit igenom ett krav på att ingen får beläggas mer än 40 timmar i veckan. Information gällande den anställdes avdelningschef behöver också finnas.

För varje anställd lagras information om familjen av försäkringsskäl. För varje familjemedlem lagras förnamn, födelsedatum, kön samt typ av relation till den anställde (make/maka resp barn). Man kan inte anta att informationen om familjemedlemmarna är unik. Man antar att anställda inte är gifta med varandra (vilket problem skulle kunna uppstå om vi inte antog detta).

Exempel på funktioner (transaktioner) man sett behov av:

Lisa Ohlsson är sjuk idag, sök ut alla projekt där hon jobbar så att man kan skriva på lokalens whiteboard att hon är sjuk.

De anställda som har barn under 12 år ska få ett erbjudande om barnförsäkring (lista anställda med barn under 12)

Projekt X har drabbats av förseningar och behöver komma ikapp. Deltagarna behöver kunna lägga mer tid på projektet under den närmaste månaden. De avdelningschefer som har folk som jobbar på projekt X ska sammankallas för förhandlingar.

2. Varuhuset

Antag att ett större varuhus-företag med butiker över hela landet behöver hålla rätt på personalen och varorna, samt kunder som får hemleveranser och leverantörer som levererar varorna till varuhuset. Man behöver hålla rätt på personalens löner och för varorna gäller det lagersaldo och leveranser från olika leverantörer (en viss vara kan levereras av olika leverantörer).

Datakrav

Varuhusföretaget har anställda, med namn och lön, som arbetar på varuhusets olika avdelningar (som har namn och unika nummer), där man säljer olika varor (namn och varunummer). Varje avdelning har en chef, som är en av de anställda.

Varorna levereras av olika leverantörer (de har unika namn och också adress), och flera leverantörer kan leverera samma varor, men till olika priser, som också kan variera från gång till gång och som ska sparas (senaste leveranspriset på en vara från en leverantör).

Varuhuset har hemkörningsservice. Kunder som har konto hos varuhuset och anmält en adress för leveranser kan beställa varor och få dem levererade hem. Varje sådan beställning har ett ordernummer och ett orderdatum, utöver listan av ingående varor (naturligtvis kan man beställa mer än en av varje vara vid ett visst tillfälle).

Denna uppgift saknar ett till två data som rimligen behöver lagras.

3. Biobesök

Vi är några vänner som vill dokumentera vilka filmer vi sett på, och på vilken biograf vi såg dem.

För varje film vill vi lagra dess titel, produktionsår och regissör, samt namnet på en eller flera av de viktigaste skådespelarna som medverkar. Anta att titel och produktionsår tillsammans är unikt.

För mig och mina vänner lagrar vi namnet och telefonnumret. Anta att alla har mobiler, så telefonnumren är unika.

För varje biograf lagrar vi dess namn, namnet på salongen och vilken stad biografen finns i. Biografers namn är inte unika, men biografens namn och stad tillsammans är unika.

För varje biobesök lagrar vi filmenl, biografen och när det skedde (datum, vi går bara på en film per dag), samt vilka av vännerna som var med. Ett biobesök har alltså en film och en bio men flera vänner.

Detta går att lagra på två fundamentalt olika sätt. Det ena kommer att resultera i att om samma personer går på samma film på samma bio kan bara ett av de besöken lagras. Vilken? Varför?

4. Mäklarfirman

Antag att en mäklarfirma behöver hjälp att hålla ordning på sina försäljningsobjekt, kunder och budgivning. Man vill också hålla ordning på de banker kunderna har kontakt med.

Datakrav

För varje objekt (fastighet) lagrar man adress, område, beskrivning, bild, boyta samt vilken typ av fastighet det är (lägenhet, villa, kedjehus). En viss mäklare (en av de anställda på mäklarfirman) är huvudansvarig för varje objekt. Varje objekt tilldelas en unik kod för identifiering. Information om ägarna till objektet lagras också.

Varje mäklare har ett unikt ID, ett kontor och ett mobilnummer där hen alltid antas vara nåbar, och namn. Varje mäklare ansvarar för ett antal försäljningsobjekt.

Man lagrar också information om de banker eller andra låneinstitut som kunderna lånar pengar av. För varje långivare (som har unika namn men också ges ett ID) har man en kontaktperson och mobilnummer till den personen, och dessutom ett centralt telefonnumret till företaget. Man behöver ibland kontakta dessa långivare för att bekräfta budgivares lånelöfte.

Varje kund registreras med namn, adress och mobilnummer, samt hemtelefon och arbetstelefon. Man registrerar också alternativa kontaktpersoner (t.ex maka/make) med telefonnummer. En kund ges ett kundnummer för att kunna identifieras unikt.

Man måste också lagra information om budgivningen på de fastigheter man förmedlar. Ett bud på ett objekt har ett visst belopp och läggs vid en viss tidpunkt, som markeras av en unik tidsstämpel för att säkert kunna se i vilken ordning buden lagts på ett visst objekt. Ett bud på ett objekt kan inte vara lägre än ett tidigare bud. Ett bud läggs av en viss kund via en mäklare och stöds av en lånegivare. Ett bud måste förmedlas via en mäklare, men det behöver inte vara den mäklare som har huvudansvaret för försäljningsobjektet. Ett bud har oftast, men behöver inte ha, en stödjande lånegivare (ifall kunden inte behöver ta lån).

Exempel på funktioner(frågor):

Man vill kunna lista alla bud på ett visst objekt i tidsordning. Man vill kunna lista alla objekt en viss mäklare ansvarar för. Man vill kunna lista alla kunder som lagt bud på ett visst objekt. Man vill lista alla fastigheter som har högsta bud under ett visst belopp.

5. Symfoniorkestern

Symfoniorkestern DOTTER vill hålla koll på sina gamla och nu planerade konserter och deras innehåll. Man kan konstatera att varje konsert ges ett antal gånger under en säsong (en viss säsong har ett visst startdatum, ett tema och ett slutdatum. Bara en säsong är aktiv i taget.

En konsert identifieras för enkelhetens skull med ett unikt nummer, men har ett namn och ett antal datum då den framförs. En konsert ges bara under en säsong.

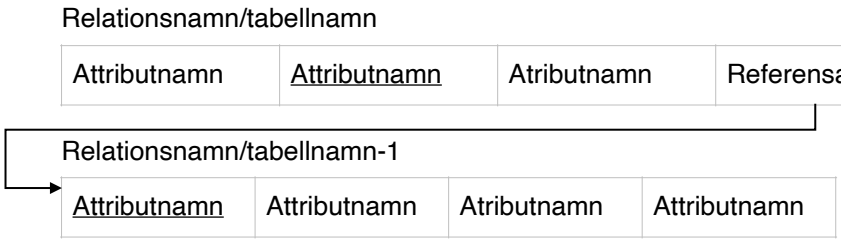
Vid en viss konsert framförs ett antal kompositioner. Varje komposition har en författare och ett namn och också ett unikt ID för att spara plats i databasen. En viss komposition kan förekomma vid flera konserter.

En konsert har också en dirigent (här behövs bara namnet lagras) och flera solister. En DOTTER-medlem är solist i en viss komposition vid en viss konsert, men en annan person kan vara solist i samma komposition vid en annan konsert.

Om solisterna lagras deras namn och vilket instrument de spelar, samt ett unikt ID. En solist kan spela på flera konserter, och kan vara solist på flera kompositioner i samma konsert. Vem som är solist på en viss komposition kan variera mellan konserter.

Övningar på Relationsmodeller

Dessa övningar går ut på att konvertera ER-diagrammen till Relationsmodeller (se Kokboksreceptet i boken eller föreläsningen om Relationsmodeller). I denna kurs ritar vi relationsscheman enligt nedanstående figur:



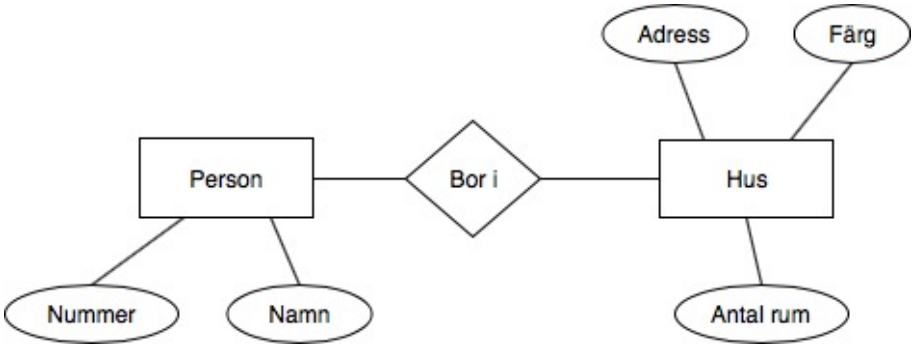
Figur 1. Relationsschema

Understrykning används för att identifiera primärnyckel (alla attribut som ingår i primärnyckeln stryks under) och pilar för främmande nycklar (se avsnittet Integritetsvillkor och Säkerhet). Observera att pilen är riktad från referensattributet till den relation där referensattributet utgör primärnyckel (för definition av begreppen, se boken eller ovannämnda föreläsning).

Observera också att när ER-diagrammet innehåller totalt deltagande i ett samband (dubbelstreckad linje) kommer det att resultera i någon typ av integritetsvillkor, då detta inte kan representeras i enbart relationen.

1. Personer bor i hus

Konvertera nedanstående ER-diagram till Relationer:



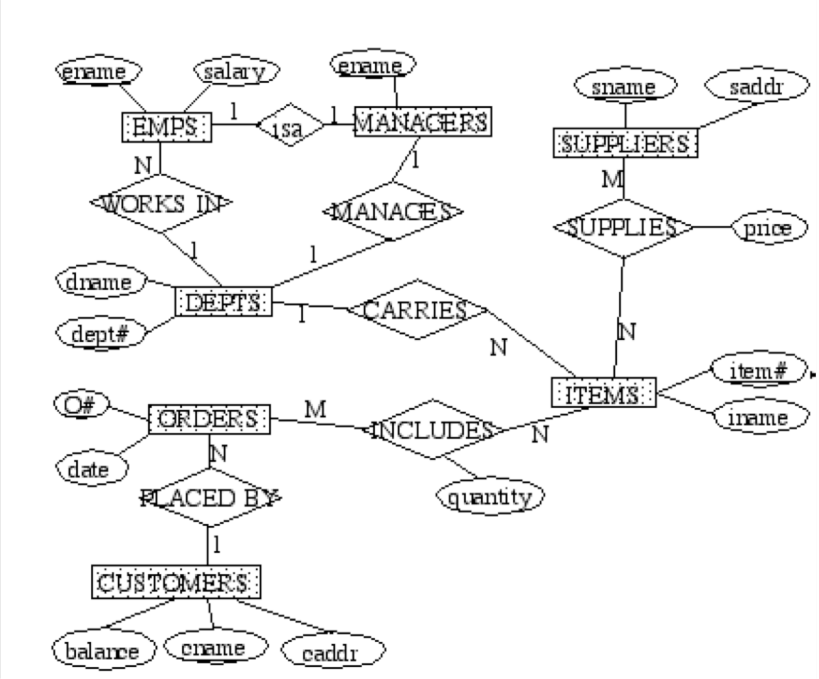
Figur 2. ER-diagram för personer som bor i hus

2. Företaget

Använd det ER-diagram du kom fram till i övning 1 i föregående avsnitt (eller titta på lösningsföreslagen på Lisam) och konvertera till relationsschema.

3. Varuhuset

Ett (ovanligt) ER-diagram för övning 2 i föregående avsnitt kan se ut så här (andra varianter finns, se Lisam):



Konvertera det till relationsschema. Vissa förenklingar kan göras, diskutera vilka fördelar och nackdelar som finns med dem.

4. Biobesök

Konvertera bägge versionerna av ER-diagrammet i övning 3 i föregående avsnitt (flervägssamband respektive central entitetstyp med binära samband till övriga entitetstyper) till relationsschema om du inte redan gjort det. Fundera på skillnaderna.

5. Mäklarfirman

Konvertera det ER-diagram du skapade i övning 4 i föregående avsnitt (eller titta på lösningsförslagen på Lisam) till relationsschema.

Integritetsvillkor och säkerhet

Här följer en kort sammanfattning av olika typer av integritetsvillkor och andra säkerhetsaspekter som är viktiga i en databas. Innehållet i en databas ska vara korrekt. Vilka tekniker finns för kontroller?

I den nedanstående texten används begreppet *relation* som samma sak som en tabell i en relationsdatabas och *attribut* vilket motsvarar en kolumn i en tabell. En *tupel* är en rad i en tabell. Ett *attributvärde* är innehållet i en cell i tabellen. Termerna relation/tupel/attribut används i första hand i teoretiska definitioner som kommer från relationsalgebra. Se avsnittet Normalisering för en fullständigare definition av begreppen.

Introduktion till Nycklar, nyckelintegritet

Vad som är en nyckel i ER-modellen är inte riktigt samma sak som en nyckel i relationsmodellen.

Nycklar i ER-modellen:

En nyckel till/för en entitetstyp definieras som: **det eller de attribut som kan identifiera de unika entitetsinstanserna**. Den identifieras vid skapandet av entitetstypen på basis av den verklighet entitetstypen representerar och markeras i ER-diagrammet genom att man stryker under namnet på attributet (eller attributen) med en heldragen linje. En nyckel kan bestå av ett eller flera attribut, om det är flera kan de också ritas som ett sammansatt attribut. En svag entitetestyp har inte en nyckel utan istället en partiell nyckel (som tillsammans med den identifierande entitetstypens nyckel används för att unikt identifiera instanser av den svaga entitetstypen). En partiell nyckel markeras i ER-diagrammet med streckad understrykning. Även en partiell nyckel kan vara sammansatt av flera attribut.

Nycklar i relationsmodellen

I relationsmodellen skiljer man på flera olika typer av nycklar men gemensamt för dem är att de är **en delmängd av attributen i en relation som kan användas för att unikt identifiera hela tupeln** i den relationen.

Relationsmodellen skiljer på tre olika typer av nycklar: *supernyckel*, *kandidatnyckel* samt *primärnyckel*. En **Supernyckel** är en attributmängd vilken som helst som uppfyller ovanstående nyckelvillkor (inklusive mängden av alla attribut i relationen, eftersom hela tupeln alltid kan användas för att hitta sig själv). En supernyckel får innehålla onödiga attribut, dvs attribut som egentligen inte behövs för att identifiera tupeln. En **Kandidatnyckel** är en minimal attributmängd som uppfyller ovanstående (dvs inget attribut går att ta bort ur attributmängden utan att tappa möjligheten att identifiera hela tupeln). **Primärnyckel** är den kandidatnyckel i en relation som väljs av databasadministratören (den som designar databasen) för att använda som referensattribut (se nedan) i andra relationer. Ofta används också primärnyckeln för att sortera tabellen och indexera på. Enligt relationsmodellen får en tupel inte sakna värde på primärnyckeln och i praktiken väljs ofta en primärnyckel som tar liten plats att lagra (orsaken till detta lämnas som övning!).

Man kan hitta kandidatnycklar (som alternativ till en markerad nyckel från ER-modellen) i en relation genom att ställa frågan: Vad representerar en tupel? Hur identifierar man den i verkligheten? (och finns det fler sätt att identifiera den?)

När vi lägger in en tupel i databasen får primärnyckeln inte ha värdet NULL, då skulle vi inte kunna använda den för att unikt hitta raden. Det får heller inte finnas två tupler med samma värde på primärnyckeln. Detta kallas **Nyckelintegritet**.

I SQL används kommandot: "`primary key (attributnamn)`" (i create table-kommandot) för att identifiera ett eller flera attribut som primärnyckel. Om man använder Engine=InnoDB i mysql kommer man då att få felmeddelanden om man försöker lägga in två tupler med samma värde på primärnyckeln eller en tupel med NULL som värde på den.

Referensintegritet

Antag att databasen innehåller de två tabellerna Kurs och Anställd som representerar relationerna Kurs och Anställd i figur 2. Tabellen/relationen Kurs representerar kurser som ges, tabellen Anställd de personer som ansvarar för kurserna. Attributet Ansvarig i relationen Kurs (dvs kolumnen Ansvarig i tabellen Kurs) innehåller anställningsnumret för den lärare som ansvarar för den kursen. Attributet Ansvarig i tabellen Kurs kallas då **Referensattribut**, vilket är samma sak som **främmande nycke**l. Referensattribut illustreras i ett relationsschema genom att en pil startar i (nedanför) referensattributet och slutar i den utpekade relationen, se figur 2. Om referensattributet är sammansatt av flera attribut (det händer om om den utpekade relationen har en sammansatt primärnyckel) har pilen flera ”bakändar”.

Figur 2 Främmande nyckel i relationsschema

Referensintegritet innebär att när information på detta sätt delas upp i flera tabeller/relationer med koppling emellan så måste information som refereras till från den ena faktiskt finnas i den andra. Den lärare som pekas ut som ansvarig i Kurs-tabellen måste faktiskt finnas (i databasen) för att databasens innehåll ska vara korrekt. Tekniskt sett innebär detta att om det på en viss rad i Kurs står 123 i kolumnen ansvarig, så måste det finnas en rad i Anställd-tabellen som har 123 som värde i AnstNr. (Notera att det finns ingen begränsning åt andra hållet, ingenting kräver att ett visst värde i Anställd(AnstNr) måste finnas i Kurs(Ansvarig)). Databashanteraren gör automatiskt kontroll av referensintegritet bland annat då data läggs till och tas bort.

I SQL definieras referensattribut med kommandot `constraint … foreign key` (som ingår i ett create table eller alter table kommando).

T.ex: `alter table Kurs add constraint fk-kurs-anstalld foreign key (kursansvarig) references Anstalld (anstnr);`

eller som del av Create table (vilket egentligen är snyggare):

`Create table Kurs (`... lista av attribut`), kursansvarig` (någon datatyp)`, foreign key (kursansvarig) references Anstalld (anstnr);`

Notera att datatypen för referensattributet och primärnyckeln måste vara exakt samma, samt att i det första exemplet är "fk-kurs-anställd" ett namn på själva villkoret, valt av databasadministratören med principen att "fk"= foreign key följt av de två tabellnamnen. Notera att detta namn måste vara unikt i din databas och du har bara en databas på LiU (tekniskt sett).

Notera också att för att få MySql att automatiskt utföra referensintegritetskontrollen måste tabellens "Engine" vara ”InnoDB” (Gäller mysql versioner 5.x och MariaDB)

SQL injektion - att skydda sig mot elak användning

Vad kan hända om man har ett formulär på nätet där man hämtar indata till en sql-sats? Antag att vi har en databas där vi lagrar personers namn och telefonnummer i en tabell med namnet Abonnent enligt nedan:

Abonnent

Namn	Telefonnr	hemligtNummer

Figur 1 En relation som lagrar namn, telefonnummer och ett booleskt värde som anger om personen har hemligt nummer.

Antag att vissa abonnenter har hemligt telefonnummer, dessa har värdet ”true” på hemligtNummer, övriga har värdet ”false”. Antag vidare att vi lägger upp ett formulär på nätet för att söka i denna tabell, men att de med hemligt nummer då inte ska presenteras. Vi har ett inmatningsfält i formuläret som levererar en sträng, som vi sedan sätter ihop med resten av ett fördefinierat SQL-uttryck och skickas till databashanteraren:

```
select namn, adress, telefonnummer from abonnent
where namn='$NAMN' and hemligtNummer=false;
```

Där $NAMN ersätts med den inmatade strängen. Om någon vill ställa till problem matar användaren in strängen ”Olle Karlsson' or 'a'='a' or ’a'='a"

sista raden i SQL-satsen blir då:

```
where namn= 'Olle Karlsson' or 'a'='a' or 'a'='a' and
hemligtNummer=false;
```

Den är alltid sann, och kommer att lista hela tabellen oavsett värde på attributet "hemligtNummer".

Läs mer i boken (kapitel 13.7) och googla också ”Little Bobby Tables” (XKCD är orginalet).

Introduktion till Normalisering

Normalisering är ett sätt att se till att en tabell (i relationsalgebra-termer en relation) inte innehåller redundant data, data som lagras på flera ställen i relationen. Man ser till att information inte dubbellagras i en relation genom att undersöka beroenden mellan attributen i en relation för att se om de uppfyller de s.k. Normaliseringsvillkoren. Det finns flera nivåer av normaliseringsvillkor (också kallade normaliseringsgrader eller normalformer), de mest använda är: 1:a normalformen (1NF), 2:a normalformen (2NF), 3:e normalformen (3NF) samt Boyce-Codds Normalform (BCNF). Det finns fler men de används sällan i praktiken eftersom det inte är givande.

De olika normalformerna beror av varandra i ordning, varför man, när man genomför normalisering bör kontrollera en relation mot de olika normalformerna i den ovanstående ordningen, alltså först 1NF, sedan 2NF, sedan 3NF till sist BCNF.

För risker och nackdelar med relationer som inte uppfyller BCNF, se boken (kap 11).

Normalisering som metod

När vi normaliserar i denna kurs går vi igenom relationerna (tabellerna) i databasen en och en, och för varje relation kontrollerar vi att relationen uppfyller de olika normaliseringsvillkoren. Om relationen inte gör det delar man upp den i två relationer genom s.k. informationsbevarande relationsschemauppdelning (se nedan). För varje uppdelning normaliserar man de två nya relationerna (oftast uppfyller de samma villkor som den ursprungliga redan gjorde, plus det man delade upp för).

Processen är alltså, för varje relation i vår databas:

1. Kontrollera om relationen uppfyller 1NF

Villkoret för 1NF är:

En relation är i 1:a normalform om:

alla attribut i relationen/tabellen är atomära, dvs odelbara.

När man använder ordet relation syftar man man på det matematiska begreppet relation, som definieras som ”en delmängd av kryssprodukten av ett antal domäner”, där en domän är en mängd attributvärden. Dessa attributvärden är per definition atomära. Alltså är attributen i en relation alltid atomära (läs mer om relationsalgebra i boken). När man talar om en tabell är frågan lite mer oklar, och man får undersöka om det finns anledning att tro att olika attribut eventuellt kan delas upp i flera i det specifika fallet. Ett attribut kan vara atomärt även om det består av en sammansatt datatyp som t.ex. en textsträng eller datumformat, det är hanteringen av det som avgör om det ska betraktas som atomärt. Om en sträng manipuleras i delar, t.ex. om den innehåller både gatunamn, postadress och postort och det görs sökningar efter geografiska områden mha postort antyder det att vi inte behandlar attributet som atomärt. Om det bara används i sin helhet för att skriva ut adresslistor eller adressera kuvert använder vi det atomärt.

Om en relation inte har atomära attribut måste man dela upp det aktuella attributet i de delar som behandlas som olika delar. Man delar alltså inte upp relationen utan bara attributet, det icke-atomära attributet ersätts av sina delar i samma relation.

När relationen uppfyller 1NF kan vi gå vidare och:

2. Kontrollera om relationen uppfyller 2NF

Villkoret för 2NF är:

En relation är i 2:a normalform om:

relationen är i 1 NF plus att alla icke-nyckelattribut i relationen är fullt funktionellt beroende av alla relationens kandidatnycklar.

Detta går också att formulera: det får inte finnas icke-nyckelattribut i relationen som är funktionellt beroende av *en del av* en kandidatnyckel.

För att kunna reda ut ifall relationen uppfyller detta behöver vi först definiera begreppen *nyckelattribut* och *funktionella beroenden*.

Funktionella beroenden, nyckelattribut och determinant

Definitionen av de högre normalformerna använder några begrepp som definieras här: funktionellt beroende, nyckelattribut och determinant:

Funktionellt beroende, Determinant

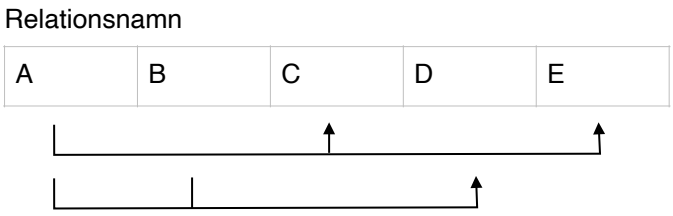
Givet att X och Y är delmängder av attributen i en relation R är definitionen av ett **Funktionellt Beroende (fb),** som skrivs X=>Y (utläses ”ett funktionellt beroende från X till Y”, eller ”Y beror av X”) att **X bestämmer värdet på Y**. Det betyder att ett visst värde på X alltid sammanfaller med ett visst värde på Y (men inte omvänt). Detta ska gälla i alla möjliga förekomster av databasinstanser, dvs det bestäms av vad relationen representerar, vad som faktiskt kan förekomma i databasen.

Ett exempel på funktionellt beroende är att i en relation som representerar personer, som innehåller namn, telefon och personnummer, finns det ett funktionellt beroende från personnummer till namn, alltså Pnr=>Namn. Namn beror av personnummer, men notera att det omvända inte gäller. Vi har också att Pnr=>Telefon, men normalt inte Namn=>Telefon eller Telefon=>Pnr.

Det eller de attribut som bestämmer, X, kallas **determinant** i beroendet. Det finns ett trivialt funktionellt beroende, nämligen att ett attribut bestämmer alltid sig själv, dvs X=>X. Determinanten kan ha med ”onödiga” attribut. Till exempel kan man säga att mängden av personnummer och telefon tillsammans bestämmer namn, alltså {Pnr, Telefon}=>Namn, fastän det räcker med personnummer.

Ett funktionellt beroende skrivs formellt i skrift som {Determinant} =>{attribut som bestäms} men när mängden består av ett enda attribut utelämnas mängdklamrarna. Notera att när mängden till vänster om pilen består av flera attribut betyder det att determinanten i det funktionella beroendet är en mängd, dvs att alla attribut i mängden bestämmer attributen till höger. När mängden på högersidan om pilen består av flera attribut betyder det däremot att de olika attributen till höger var för sig bestäms av determinanten.

Ett funktionellt beroende kan också illustreras visuellt i ett relationsschema, med hjälp av en pil från determinanten till det/de bestämda attributen i relationen, se nedan:



Figur 4 *En relation med attributen A..E . Här visas med pilar de funktionella beroende {A}=>{C,E}, samt {A,B}=>D. Lägg märke till var pilarna börjar och slutar.*

Ett **Fullt Funktionellt Beroende (ffb)** X=>Y är ett funktionellt beroende sådant att man kan inte ta bort något attribut ur determinanten (X) och fortfarande ha ett funktionellt beroende.

Funktionella beroenden kan beräknas utifrån varandra. Om vi har en relation med attributen A, B och C och vet att vi har fb A=>B och B=>C så gäller också att A=>C. Detta kallas ett transitivt beroende.

Man hittar de fulla funktionella beroendena i en relation genom att systematiskt gå igenom den attribut för attribut och ställa frågan: vilket eller vilka andra attribut bestämmer värdet på detta attribut? Det är vad relationen faktiskt representerar som avgör svaret.

Nycklar, nyckelattribut

De attribut i en relation som ingår i någon *kandidatnyckel* för relationen kallas för **nyckelattribut**. De attribut som **inte** ingår i någon kandidatnyckel kallas **icke-nyckelattribut**. Nyckelattribut markeras i relationen med asterisk (*) efter attributnamnet. Notera att det bara är när vi jobbar med normalisering som nyckelattribut ska markeras på detta sätt.

Det är viktigt att man identifierar alla kandidatnycklar och också de viktigaste funktionella beroendena i relationen, innan man fortsätter normaliseringen till 2:a och 3:e normalform, eftersom villkoren för dessa normalformer utgår från dem.

För att undersöka om en relation uppfyller 2:a normalform kan vi nu undersöka villkoret:

En relation är i 2:a normalform om:

relationen är i 1 NF plus att alla icke-nyckelattribut i relationen är fullt funktionellt beroende av alla relationens kandidatnycklar.

Detta går också att formulera: det får inte finnas icke-nyckelattribut i relationen som är funktionellt beroende av *en del av* en kandidatnyckel.

Det betyder bland annat att om relationen bara har enkla kandidatnycklar, dvs kandidatnycklar som består av enstaka attribut, så uppfyller den automatiskt 2NF. Notera också att här kollar man INTE efter attribut som är beroende på annat än kandidatnycklar, utan bara att de behöver hela kandidatnyckeln för att bestämmas. En relation som inte uppfyller detta villkor måste delas upp i två, se Informationsbevarande relationsschemauppdelning, se nedan.

När relationen uppfyller 2NF kan vi gå vidare och undersöka:

3. Kontrollera om relationen uppfyller 3NF

Villkoret för 3NF är:

En relation är i 3:e normalform om:

relationen är i 2 NF plus att inget icke-nyckelattribut är ffb av ett annat icke-nyckelattribut.

Notera att här avses egentligen "inget icke-nyckelattribut är ffb av en determinant som innehåller ett annat icke-nyckelattribut. Formuleringen i den löpande texten i boken i avsnitt 12.12: "Det får inte finnas några pilar som går mellan attribut utanför de olika kandidatnycklarna, bara (antingen) från kandidatnycklar till attributen utanför eller från attributen utanför in i kandidatnyckeln" visar att det är vad som avses.

Om vår relation inte uppfyller detta måste den delas i två, se informationsbevarande relationsschemauppdelning, nedan. När den uppfyller villkoret för 3NF kan vi gå vidare och undersöka::

4. Kontrollera om relationen uppfyller BCNF:

BCNF står för Boyce-Codds normalform, efter de två forskare som definierade det. Villkoret för BCNF är:

En relation är i BCNF om:

relationen är i 3NF plus att alla determinanter ska vara en kandidatnyckel.

Varianten som anges i boken, "1NF plus att alla determinanter ska vara en kandidatnyckel" är ekvivalent eftersom 3NF förutsätter att 2NF är uppfyllt, som i sin tur förutsätter att 1NF är uppfyllt. Den andra varianten som anges som "en krånglig definition" av BCNF, "3NF plus att inte heller nyckelattribut får vara beroende av ickenyckelattribut" är egentligen inte riktigt samma sak, eftersom den formuleringen tillåter att ett nyckelattribut är beroende av ett annat nyckelattribut, som enligt exemplet i boken (s 260-261) inte ska vara tillåtet. Det är därför bättre att använda den fetstilta definitionen ovan.

Om relationen inte uppfyller villkoret och man vill uppfylla BCNF måste den delas i två. Det finns tillfällen då man inte nödvändigtvis vill uppfylla BCNF för att det är svårt att dela relationen utan att

bryta funktionella beroenden. Det kan innebära att viss data går förlorad, och då kan det i vissa fall vara värt att behålla den grad av dubbellagring som 3NF medger.

Informationsbevarande relationsschemauppdelning

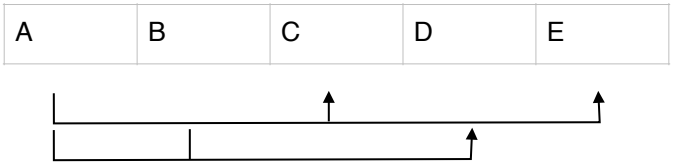
När man har redundans i sin relation (dvs bryter mot någon normalform) har man alltså någon form av dubbellagrad information. Då måste man bryta ut den information som dubbel-lagras. Det gör man genom att skapa två relationer av den ursprungliga relationen. Uppdelningen ska vara **informationsbevarande**, dvs det ska gå att återskapa den ursprungliga relationen (exakt) genom att göra en naturlig join[1] mellan de två nya. Man får inte bryta något fullt funktionellt beroende genom att sära på determinant och de attribut som bestäms av den i olika relationer.

Ett tips på hur man kan tänka här är att skapa en ny relation som representerar det funktionella beroende som gör att relationen bryter mot normalformen i fråga. Den nya relationen ska då innehålla determinanten och de attribut som bestäms av den. I den ”gamla relationen” låter man determinanten till det aktuella funktionella beroendet vara kvar och stryker de beroende attributen. Detta resulterar i att determinanten som är i den "gamla relationen" blir en främmande nyckel till den nya relationen, och då kan innehållet alltid återskapas.

Den ”gamla relationen” bör normalt behålla namnet från ursprungsrelationen (t.ex med ett ´efter, t.ex. Person´) medan den nya bör ges ett namn som beskriver vad den representerar.

Ett problem uppstår om något av de beroende attributen (som alltså ska strykas i den ”gamla relationen”) är eller ingår i en determinant i ett annat fullt funktionellt beroende än det aktuella. Det finns då ett transitivt funktionellt beroende som man ska beräkna och använda istället, dvs man lyfter ut allt som bestäms av determinanten i fråga.

När man har delat upp en relation i två på detta sätt bör man iterativt normalisera dem. Oftast uppfyller de nya relationerna samma normalformer som den gamla redan uppfyllde, plus den normalform man undersökte.



Figur 5: En relation med samma ffb som figur 4

Exempel: Relationen i figur 4 (och 5) uppfyller inte 2NF och behöver delas upp (anledningen lämnas som övning). Om vi nu skapar en ny relation med attributen A, C och E, så behöver vi lämna kvar A i ursprungsrelationen för att kunna återskapa ursprungstabellen. Däremot stryker vi C och E ur den gamla tabellen. Resultatet visas i figur 6.

Figur 6. Relationen i figur 5 efter uppdelning.

Dessa relationer har samma fulla funktionella beroenden som tidigare och det finns dessutom en främmande nyckel från A i den gamla tabellen (till höger) till den nya tabellen (till vänster).

[1] Läs om relationsalgebra-operationen join i boken (kap. 10.11-12)

Normaliseringsövningar

1. Universitetet

Antag att vi representerar information om program på ett universitet i nedanstående relation:

Program (Kod, Namn, Programansvarig, Studievägledare, Sektion)

Varje program har en kod som är unik, ett namn, namnet på en programansvarig, en studievägledare och en sektion som organiserar studenterna på programmet.

Antag att vi kommer på att vem som är programansvarig och vem som är studievägledare för ett program kan variera mellan olika år och vi behöver kunna lagra personerna som hade dessa roller respektive år. Vi måste då lägga till ett attribut År i relationen Program och definiera nyckeln som sammansättningen av Kod och År. Detta medför att vi får en instans av Program för varje år, vilket var vad vi behövde för att kunna lagra olika namn på olika år.

Innehåll i Program skulle då kunna vara:

Kod	År	Namn	Programansv	Studievägledare	Sektion
LIU500007	2019	Kognitionsvetenskapliga programmet	Mattias Arvola	Katarina Isotalo	KogVet
LIU500007	2020	Kognitionsvetenskapliga programmet	Erik Prytz	Katarina Isotalo	KogVet
LiU50009	2020	Statistik och dataanalys	Linda Wänström	Isak Hietala	StatLin
LiU50009	2019	Statistik och dataanalys	Linda Wänström	Annika W	StatLin

Notera att ett programs kod, namn och sektion är oförändrad år från år medan Programansvarig och Studievägledare alltså kan variera.

Identifera fulla funktionella beroenden i Program och normalisera relationen. Notera att det bara finns den kandidatnyckel som diskuteras ovan.

2. Biblioteket

Linköpings oberoende bibliotek har en databas som håller reda på alla deras böcker. De är inte så många, men biblioteket är väldigt oberoende. Dessutom innehåller databasen registrerade låntagare och aktuella lån. Tabellerna, med innehåll, finns nedan. Tabellinnehållet kan användas för att motbevisa teorier om funktionella beroenden (men inte bevisa).

Dock är bibliotekets databasdesign kass. Analysera den existerande databasen, förklara vilka problem den har och föreslå en bättre design. (Dvs normalisera respektive tabell). Om du vill kan du därefter göra en helt ny design genom att börja från början med ER-diagram, konvertera till relationsschema och normalisera. Blev det någon skillnad?)

Det finns tre tabeller:

- En tabell som heter BOOK som innehåller data om bibliotekets böcker. Den har attributen TitleNr vilket är ett löpnummer som biblioteket ger till varje bok som tas in, ISBN (olika utgåvor, olika år, av en bok har olika ISBN, och får också olika TitleNr men har samma Title och författare), CopyNr som används för att skilja på olika exemplar av samma bok och börjar på 1 för varje TitleNr, Title som är bokens namn, PublYear är publikationsår, Author är författarens namn (man lägger in en rad (tupel) per författare för böcker med flera författare - varje rad har alltså bara en författare), AuthorNat är

författarens land samt Condition vilket är just det exemplarets (fysiska) skick. Primärnyckel är satt till kombinationen av TitleNr, CopyNr och Author.

- En tabell som heter CUSTOMER, och som innehåller data om låntagarna. Den har attributen CustomerNr som är ett unikt nummer som biblioteket ger varje låntagare, PersonNr som är personens personnummer, Name, Address och Tel som är information om låntagaren. CustomerNr är primärnyckeln.
- En tabell som heter LOAN, där information om lån är lagrad. Den har attributen TitleNr, CopyNr, CustomerNr, Date (vilket är datumet då boken lånades ut), samt BorrowerName som är låntagarens namn. Primärnyckel är kombinationen av TitleNr och CopyNr.

Databasinnehållet

Tabellerna ser för närvarande ut såhär (notera att en teori om ett funktionellt beroende aldrig kan bevisas av ett databasinnehåll, bara motbevisas):

BOOK

TitleNr	ISBN	CopyNr	Title	PubYear	Author	AuthNat	Cond
1	7114810	1	Database system concepts	1997	A. Silberschatz	USA	Good
1	7114810	1	Database system concepts	1997	Henry F. Korth	USA	Good
1	7114810	1	Database system concepts	1997	S. Sudarshan	India	Good
2	805317538	1	Fundamentals of database systems	1994	Ramaz Elmasri	USA	Mint
2	805317538	1	Fundamentals of database systems	1994	S. B. Navathe	USA	Mint
2	805317538	2	Fundamentals of database systems	1994	Ramaz Elmasri	USA	Prist. Mint
2	805317538	2	Fundamentals of database systems	1994	S. B. Navathe	USA	Prist. Mint
3	198642253	1	Mord	1996	Jan Guillou	Sweden	Poor
3	198642253	2	Mord	1996	Jan Guillou	Sweden	Good
4	3411021764	1	Våld	1998	Jan Guillou	Sweden	Poor

Notera att denna tabell representerar 4 böcker där biblioteket har två av dem i två exemplar.

CUSTOMER

1. CustomerNr	1. PersonNr	1. Name	1. Address	1. TelNr
1	6312111658	Thomas Padron-McCarthy	Vägen 7	282627
2	4403149001	Lena Strömbäck	Gatan 6	123456
3	5901239023	Eva Ragnemalm	Vägen 3	987654
4	5612067832	Arnold Scwarzenegger	Stigen 8	345634

LOAN

1. TitleNr	1. CopyNr	1. CustomerNr	1. Date	1. BorrowerName
1	1	3	23 nov. 2009	Eva Ragnemalm
3	2	1	1 nov. 2009	Thomas Padron-McCarthy
2	1	2	1 dec. 2009	Lena Strömbäck

OBS: om man följer ”algoritmen” att undersöka normalformerna i tur och ordning och gör informationsbevarande relationsschemauppdelningar på korrekt sätt kommer man att ha en konstig Book-tabell kvar på slutet som man inte blir av med. Då har man gjort rätt... Det är inte alltid man kan rädda en dålig design genom att normalisera på detta sätt.

3. Andrahandsuthyrningsfirma

En firma som administrerar andrahandsuthyrning av lägenheter vill hålla reda på kontraktisinformationen.

Man vill hålla reda på vem hyr vad (kundnamn, kundnummer, lägenhetsnummer, lägenhetsadress) när (start och slutdatum) samt till vilken hyra (som är olika för varje lägenhet). Man vill att gamla kontrakt ska gå att hitta, men om samma person hyr samma lägenhet en gång till ersätts detta kontrakt med det nya. En lägenhet kan bara hyras av en person åt gången.

De lagrar också information om vem som egentligen äger lägenheten.

För ”tills vidare”-kontrakt registreras slutdatum som null.

Varje person antas bara hyra varje lägenhet en gång och kan bara hyra en lägenhet åt gången. En ägare kan dock låta firman hantera flera lägenheter.

Möjligt (icke-optimalt) relationsschema:

Kontrakt(kundNr, lghNr, kNamn, lghAdr, start, slut, hyra, ägarNr, äNamn)

KundNr	Lgh Nr	Kund Namn	Lgh Adr	Start	Slut	Hyra	Ägar Nr	Ägar Namn
CR76	PG4	J.Kay	Lagv 4	81112	90312	3200	CO40	T.Moe
CR76	PG16	J.Kay	Nyv 12	90313	null	3500	CO93	U.Sin
CR56	PG4	A.Son	Lagv 4	90313	null	3200	CO40	T.Moe
CR56	PG35	A.Son	Husg 1	71112	90310	3000	CO93	U.Sin
CR56	PG16	A.Son	Nyv 12	60801	71110	3500	CO93	U.Sin

4. Inspektion

Antag att uthyrningsfirman inspekterar varje lägenhet mellan uthyrningarna och noterar brister och problem. När inspektion ska göras bokar inspektören en bil som används under dagens inspektioner. En bil kan bokas av två personer under samma dag eftersom inspektörerna ofta är ute enbart förmiddag eller eftermiddag, men en inspektör byter inte bil under dagen (kan alltså bara boka en bil per dag). En inspektör kan inspektera flera lägenheter under samma dag, men varje lägenhet inspekteras endast en gång en viss dag.

Det finns alltså ett (eller flera) rapportformulär som nedanstående per lägenhet, dvs information om lägenhetsnummer och lägenhetsadress ska också lagras i databasen:

Lägenhetsnummer: PG4
Lägenhetsadress Studentv 8 Nollköping

Inspektionsdatum	Tid	Kommentar	InspektörNr	Inspektör Namn	BilNr
90228	10.00	Trasigt proslin	SG37	Ann Beech	ABC123
81130	09.00	Fint	SG14	David Ford	DEF098
71130	13.00	Mögel i badrum	SG14	David Ford	GHI987

Normalisera relationen.

5. Bilar

Övning ur kursboken, sid 234.

Man vill representera ett antal medlemmar i en förening (bilklubb) och deras bilar. Personerna har medlemsnummer (unikt) och namn samt ett antal bilar var. Man kan tänka sig att representera detta på följande olika sätt:

Medlem 1

Nummer	Namn	Bilar
1	Olle	RFN540
2	Stina	null
3	Saddam	SQL123, DBA456, WCA912

Medlem 2

Nummer	Namn	Bil 1	Bil 2	Bil 3
1	Olle	RFN540	null	null
2	Stina	null	null	null
3	Saddam	SQL123	DBA456	WCA912

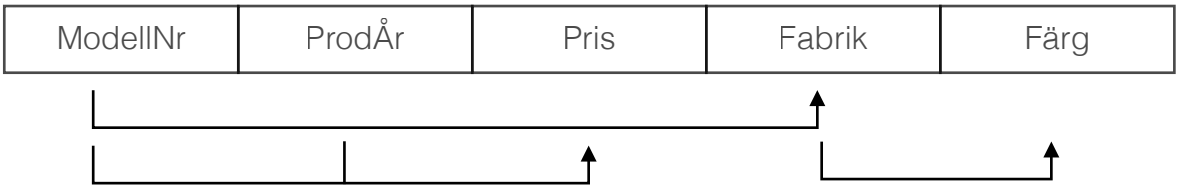
Medlem 3

Nummer	Namn	Bil
1	Olle	RFN540
3	Saddam	SQL123
3	Saddam	DBA456
3	Saddam	WCA912

Vilka problem finns med dessa lösningar? Finns det något bättre sätt? Och vad har detta med normalformer att göra?

6. Kylskåp

Antag att du har en relation Kylskåp enligt nedan, med de fulla funktionella beroenden som finns utritade (de skrivs: ModellNr=>Fabrik, ModellNr, ProdÅr=>Pris, Fabrik=>Färg)



Fungerar följande attribut/attributmängder som kandidatnyckel? Motivera.

a) {ModellNr}

b) {ModellNr, ProdÅr}

c) {ModellNr, Färg}

Baserat på den/de kandidatnycklar du identifierar, ange vilken normalform Kylskåp uppfyller och varför.