DATAMODELLERING

Förord

Syftet med denna anvisning är att den ska vara enkel och lätt att förstå för de som vill skapa en hållbar och bra datamodell som underlag för en relationsdatabas.

”En grafisk modell säger mer än tusen Kinesiska ordspråk.”

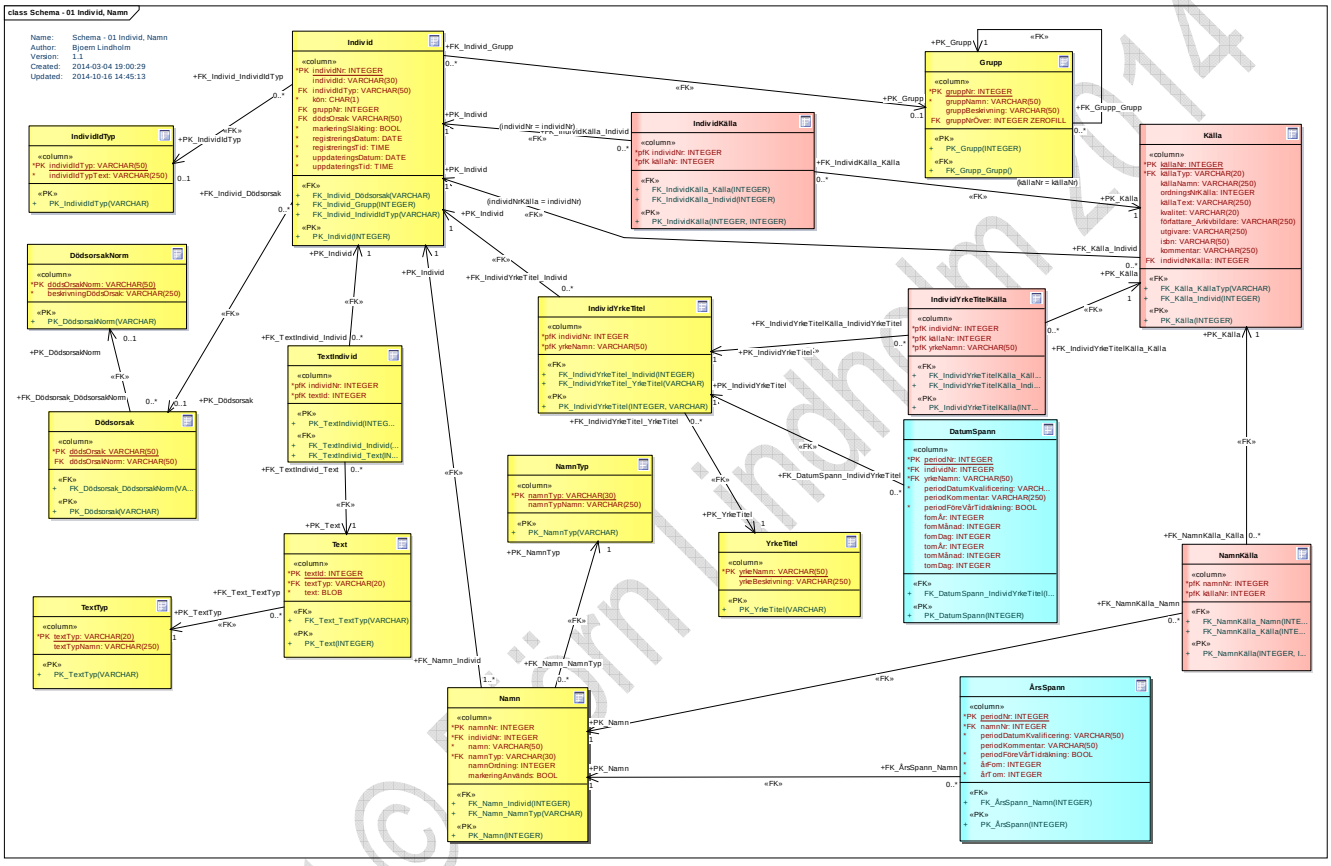
Innehållsförteckning

Vad en datamodell är

En datamodell är en modell med tabeller som ska ingå i en relationsdatabas.

Tabellerna är många gånger information av olika natur som vi vill hålla reda på. Det kan även vara fysiska saker. Vad som ingår bestäms av den verksamhet vi ska lagra information om i våra tabeller.

Nedan ser vi ett exempel på en del av en datamodell för verksamheten genealogi (släktforskning).



I modellen ovan är Individ och Källa exempel på fysiska saker, medan DatumSpann och NamnTyp är ren information.

Vi ska senare se på detaljerna i modellen.

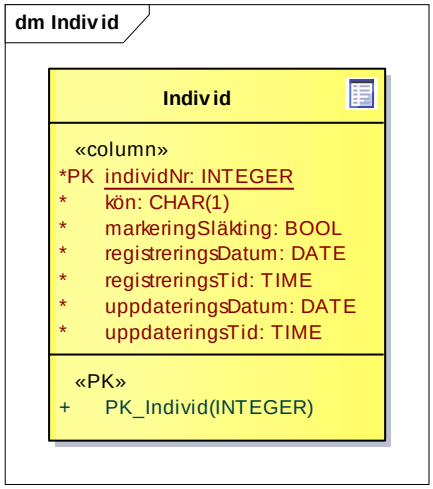
Beståndsdelar i en datamodell

En datamodell består enkelt uttryckt av tre delar;

- **Tabeller** med sina attribut (kolumner)
- **Relationer** (kopplingar) mellan tabellerna
- **Diagram** (eller grafer)

Primärnyckel

Varje tabell måste ha minst en identifierare / primärnyckel, PK. En identifierare är ett attribut som **unikt** identifierar varje förekomst av rader i tabellen.



Tabellen ”Individ” i grafen ovan har som identifierare ”PK” attributet individNr. Detta betyder att varje förekomst av en individ har ett unikt nummer.

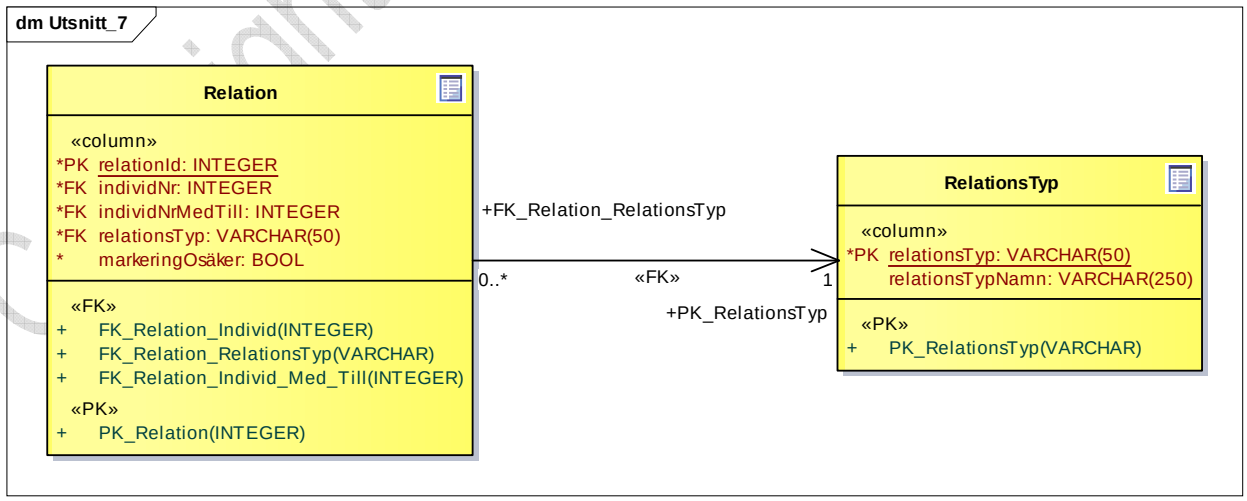
En primärnyckel för en tabell kan bestå av ett eller flera attribut om det behövs.

Främmande nyckel

En främmande nyckel ”FK” är ett attribut som finns i en tabell och som pekar ut en viss förekomst (rad) i en annan tabell (där den är primärnyckel). Dessa är mycket bra för att hålla ordning och reda i en databas. En rad i en tabell kan inte tas bort ifall det i en annan tabell finns en främmande nyckel som pekar på just den raden. I så fal måste den raden i tabellen med den främmande nyckel tas bort först. Detta minskar risken för felaktig bearbetning och alltså risken för en korrupt databas.

Klassificerade eller typade egenskaper

I många sammanhang kan det vara bra att lyfta ut egenskaper till egna tabeller. Det handlar om s k typade egenskaper som vi vill hålla reda på och kunna förändra över tid. Ett sådant exempel syns nedan.



Till tabellen ”Relation” finns kopplad en tabell ”RelationsTyp”. Attributet relationsTyp har exempelvis värdeförrådet, gift sambo, biologiskt barn, adoptivbarn, fadder etc.

Eftersom attributet relationsTyp är främmande nyckel FK i tabellen Relation, kan en relationsTyp, exempelvis ”sambo” inte tas bort ur tabellen RelationsTyp med mindre än att alla rader i tabellen Relation som har värdet ”sambo” i relationsTyp först tas bort eller ändras till någon annan relationsTyp.

På liknande sätt kan en rad i tabellen Relation inte skapas med en relationsTyp som inte finns i tabellen RelationsTyp.

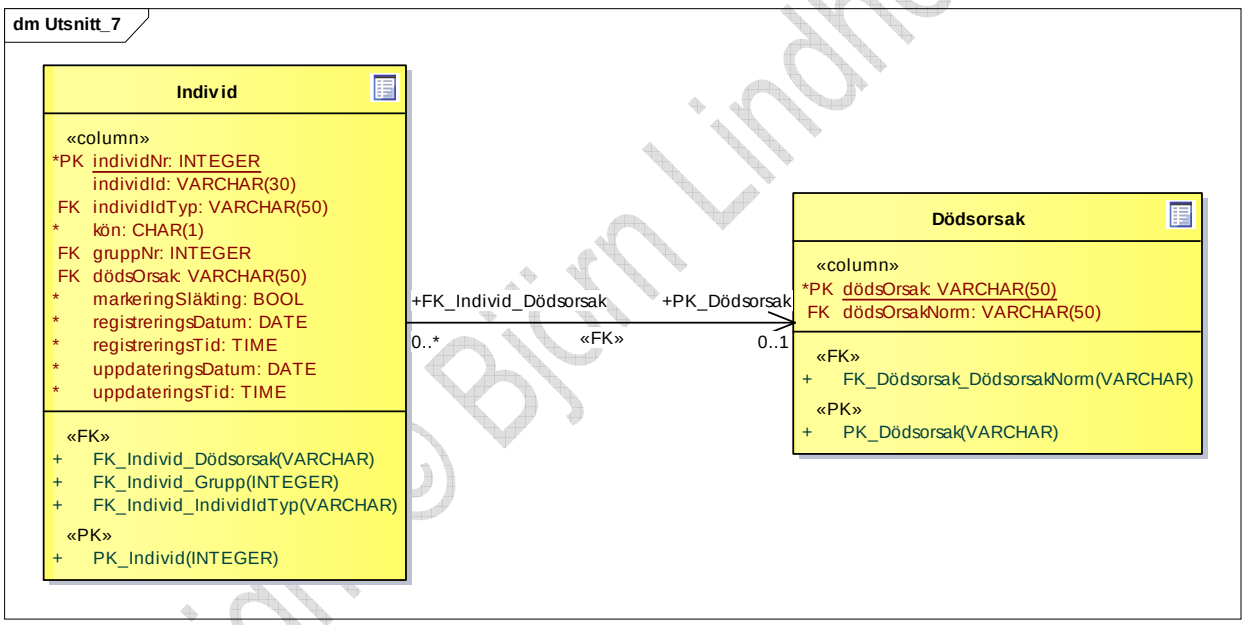
Detta gör att det blir enkelt att hålla koncistens i databasen.

Relationer kopplingar

Relationerna beskriver hur olika tabeller hänger samman med sina nycklar och främmande nycklar..

Multiplicitet

Multiplicitet är ett ord för att beskriva hur många av en tabell som relaterar till hur många av an annan tabell.



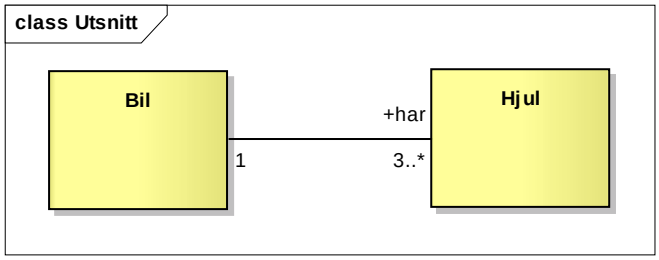
I diagrammet ovan ser vi att för associationen finns texten ”0..*” i ena ändan och ”0..1”. Detta utläses ”noll till många” Individer relaterar till ”noll till en” Dödsorsak.

Detta betyder alltså att en Individ inte behöver ha en dödsorsak knutet till sig, men om det finns en så får det bara finnas en. En beskriven dödsoras kan å andra sidan finnas angiven för många individer eller ingen.

De vanligaste förekommande typerna av multiplicitet är följande;

0..1
0..*
1
1..*

Multipliciteten ska beskriva så noga som möjligt och det ska även här vara den verksamhet som vi beskriver som styr. Titta på exemplet nedan.

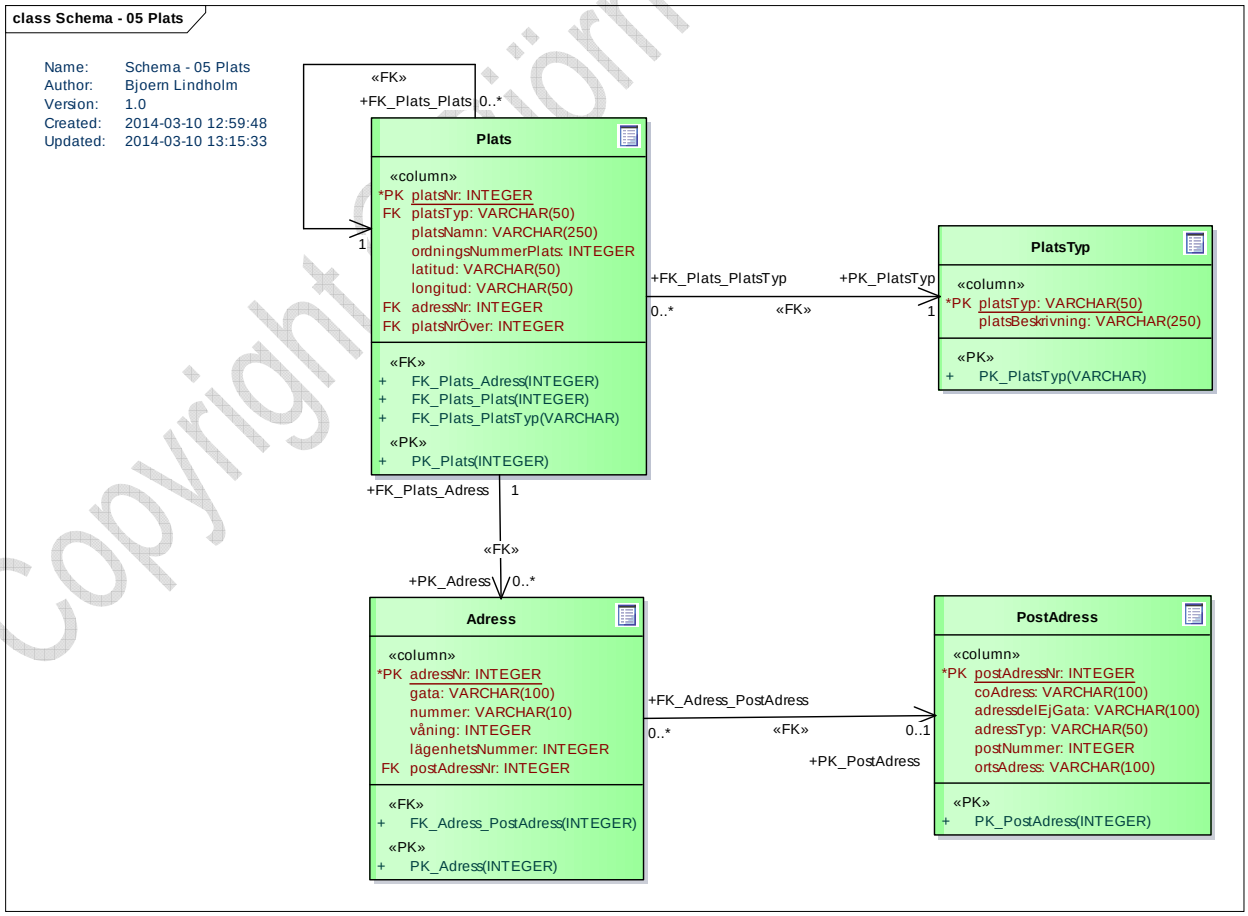


Av diagrammet kan vi utläsa att, **en** bil har **3 eller flera** (många) hjul. Vi har därmed klargjort att det inte finns några bilar som har färre än 3 hjul. Samtidigt har vi inte satt några gränser uppåt för hur många hjul en bil får ha i vår verksamhet.

Diagram

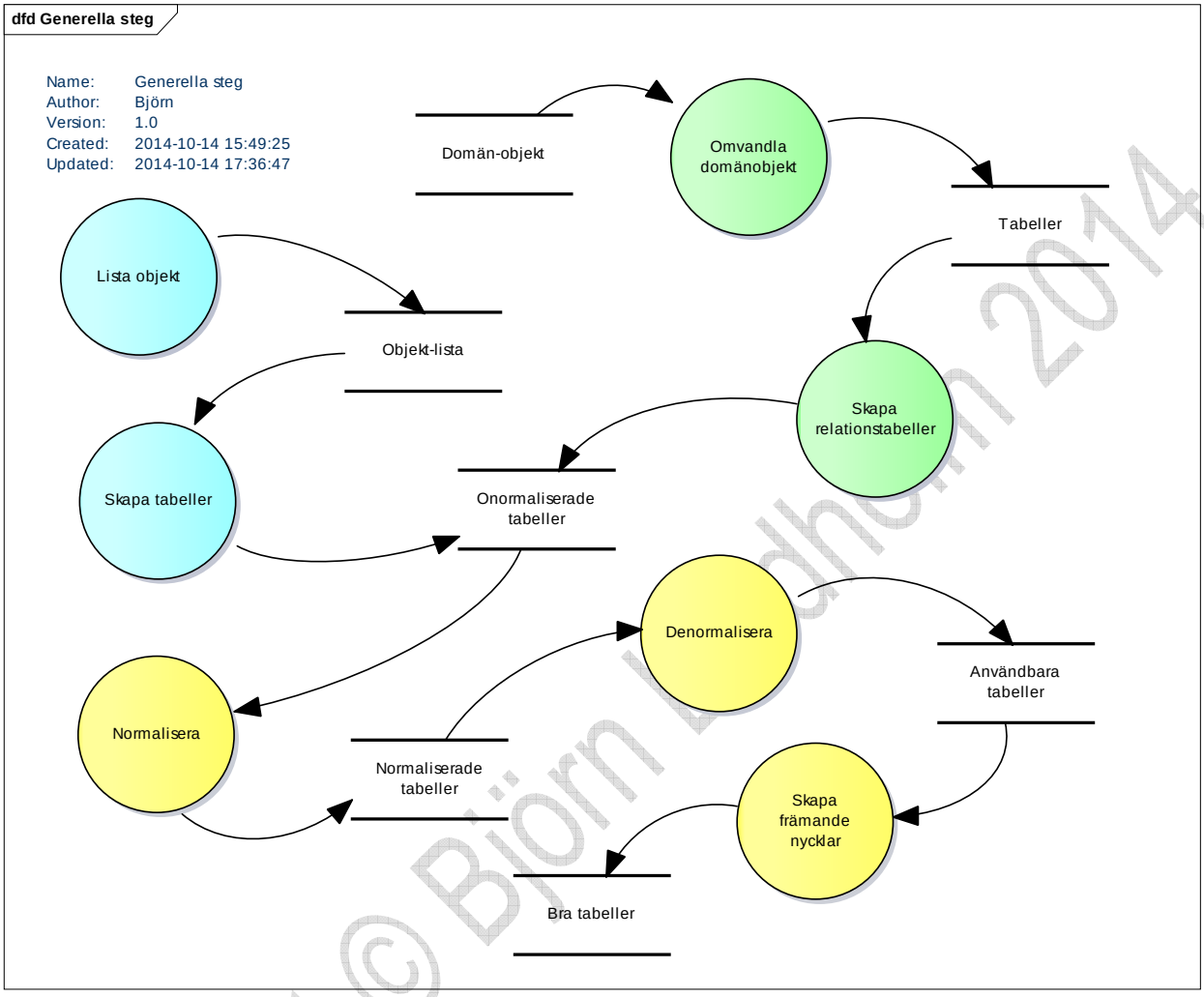
En datamodell kan beskrivas i ett jättediagram innehållande alla tabeller för verksamheten och alla relationer som finns mellan tabellerna. Det kan vara trevligt att ha som en totalbild. Mer praktiskt är dock att ha ett antal diagram som beskriver olika delar av den totala modellen, olika utsnitt. Det gör det lätta för diskussioner olika människor emellan. Fokus sätts på ett bestämt område och inget annat distraherar.

I exemplet nedan från släktforskningsmodellen syns ett diagram som endast fokuserar på ”Plats” och hur det hanteras.



De olika stegen i datamodellering

De generella stegen i datamodellering är enligt följande bild.



Utgångspunkten vid modelleringen kan vara olika. Endera så har vi en domänmodell att börja med eller så har vi inte det. Arbetet med att skapa en datamodell kommer att gå betydligt fortare om vi har en domänmodell.

I bilden ovan finns två blå aktiviteter, ”Lista objekt” och ”Skapa tabeller”. Dessa aktiviteter börjar vi med när vi inte har någon domänmodell att utgå från. Det två gröna aktiviteterna, ”Omvandla domänobjekt” och ”Skapa relationstabeller” är de begynnande aktiviteterna som vi utför i de fall vi har en domänmodell att utgå ifrån. De tre sista gula aktiviteterna är gemensamma oavsett vad vår utgångspunkt är. Det är dock troligen så att normaliseringen är mer omfattande i de fall vi inte har en domänmodell att börja med.

De första stegen när en befintlig domänmodell finns

Vi har alltså en modell med ett antal domänobjekt och relationer mellan dessa att utgå ifrån.

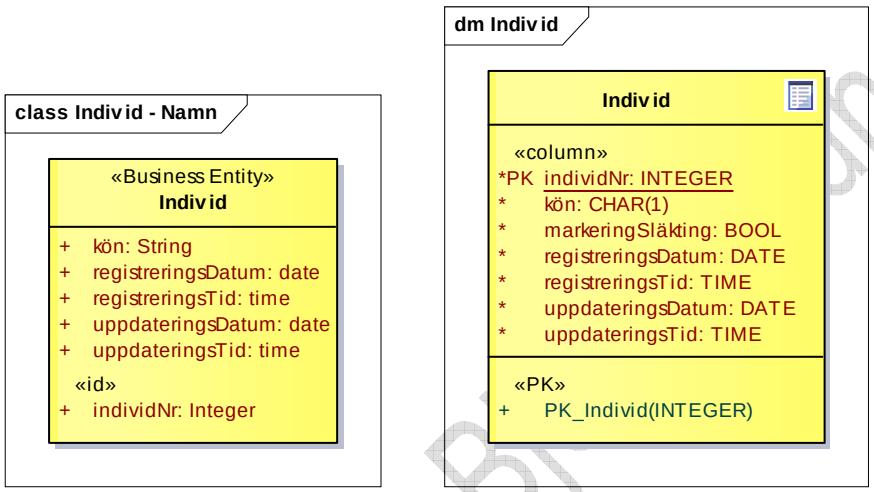
De två stegen är;

- Omvandla domänobjekt
- Skapa relationstabeller

Mycket av arbetet i dessa två steg kan automatiseras om vi andänder ett bra modelleringsverktyg. Ett sådant kan klara det mesta av omvandlingen av domänobjekt till tabeller och även skapandet av relationstabeller.

Omvandla domänobjekt

Detta går enklast till så att vi för varje domänobjekt skapar en tabell. Det kan se ut enligt följande.



Till vänster är domänobjektet och till höger ser vi tabellen. IndividNr var id i domänobjektet och ska följaktligen vara primärnyckel, PK, i tabellen.

Vi kan även se att kolumnerna (attributen i tabellen) har fått två egenskaper. Dels har alla en asterisk framför sig. Vidare har individNr blivit understruken. Asterisken betyder att vi har definierat att attributet inte får vara ”null”, d v s det måste alltid finnas och ha ett godkänt värde enligt vår deklaration av värdeförråd. Understrukningen markerar att attributets värde måste vara unikt.

Att vi gör dessa definitioner ger en stabilare databas i slutändan.

Till sist så har vi omvandlat attributens typning från Java i domänobjekten till en typning för SQL. Det betyder framför allt att alla String måste omvandlas till CHAR eller VARCHAR. Vi måste då också bestämma hur långa dessa attribut ska vara. För att göra detta tar vi en titt på de definierade värdeförråden, d v s domänobjektens domänvärden.

Observera dock att det inte räcker med att endast gå igenom domänvärdena. Vi måste även tänka på framtiden och ta höjd för eventuella kommande förändringar och tillägg.

Den uppmärksamme har säkert noterat att det har tillkommit en kolumn, markeringSläkting. Mer om den nedan.

Hantering av generalisering specialisering

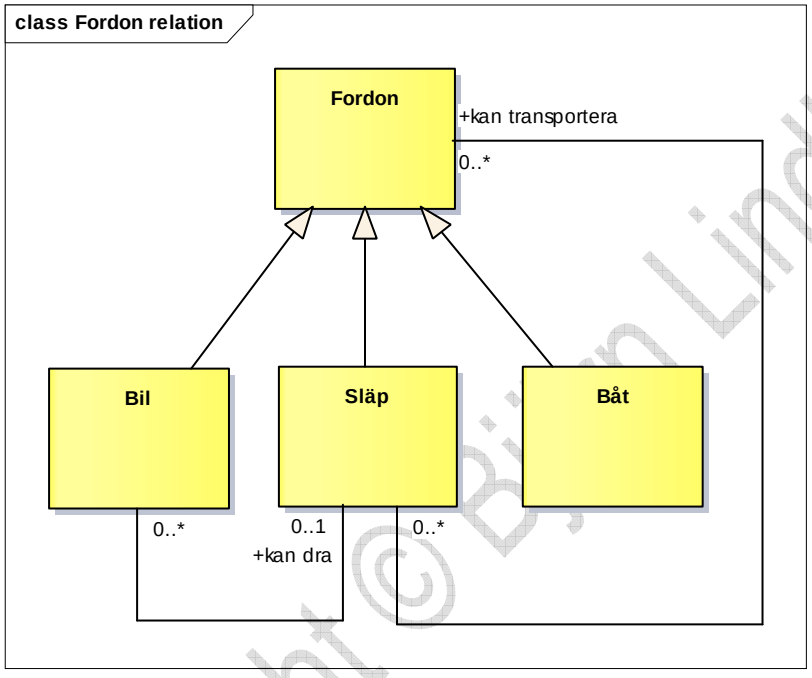
I en datamodell kan vi inte ha denna typ av relation (arv). Vi måste alltså lösa upp det på något sätt. Alternativen är följande.

- Skapa en tabell för varje domänobjekt
- Skapa tabeller endast för de specialiserade domänobjekten
- Skapa tabell endast för det generella domänobjektet

Vilket alternativ vi bör välja beror på domänobjektens innehåll och relationer.

Detta arbeta kan normalt inte klaras av ett modelleringsverktyg automatiskt utan måste ske manuellt efter en eventuell generering.

En tabell för varje domänobjekt är nödvändigt om de olika domänobjekten har olika identifierare. De är då troligen även ganska olika beträffande egenskaperna. I diagramet nedan finns ett sådant exempel.

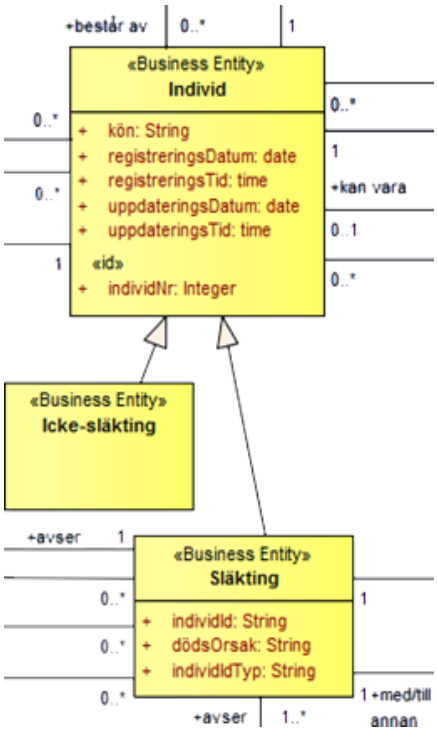


Det är säkert så att både fordon, bil och båt har olika identifierare. Dessutom har sannolikt bil och båt en hel del olika egenskaper.

En tabell endast för specialiserade domänobjekten passar bra om de har mycket olika relationer till andra domänobjekt.

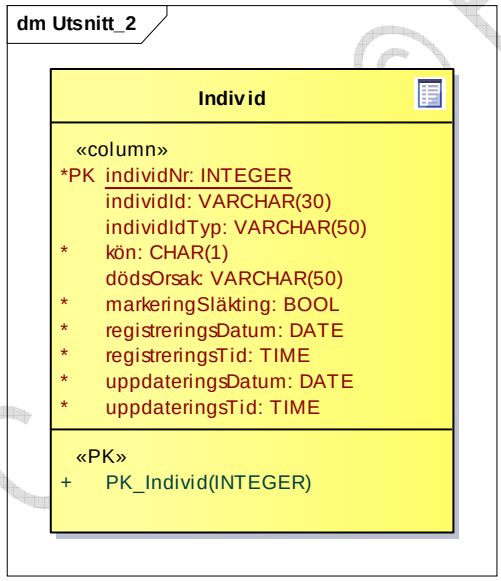
En tabell endast för det generella domänobjektet passar bra om ett av de specialiserade saknar egna egenskaper eller har mycket få och det samtidigt inte har några egna relationer till andra domänobjekt.

I domänmodellen för genealogi har vi följande generalisering/specialisering.



Av diagrammet framgår att Individ har specialiserats med Släkting och Icke-släkting. Vi kan notera att vi för Icke-släktingar inte har något behov av att lagra annan information än för individ. Vidare ser vi att släkting har betydligt fler relationer än icke-släkting som endast har de relationer som individ har.

I detta fall är det lämpligt att endast skapa en tabell för individ och ingen tabell för de specialiserade domänobjekten. Tabellen kommer att se ut enligt följande.

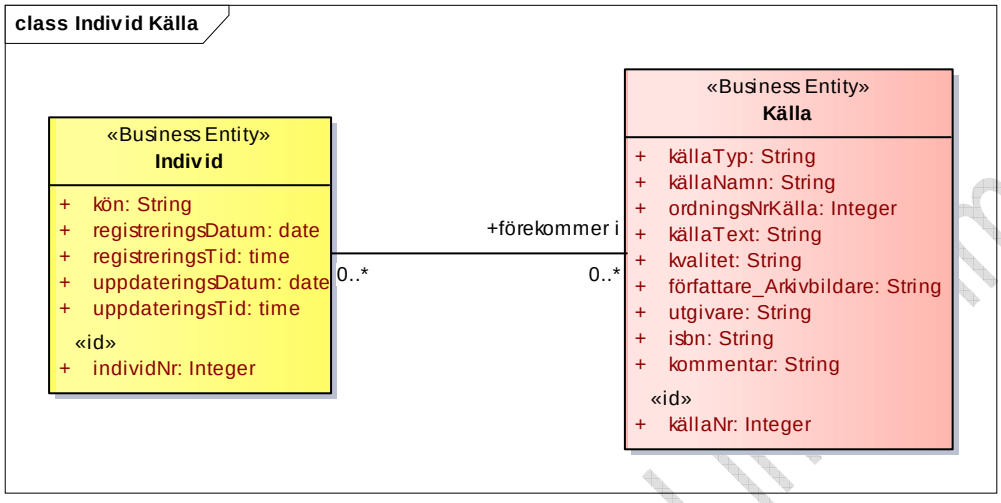


Vi kan se att tabellen för individ nu har utökats med de tre attributen från domänobjektet släkting. Dessa har ingen asterisk framför sig, vilket betyder att de kan vara tomma (null). Det kommer de att vara (null) för Icke-släktingar bl a. En boolesk variabel har också tillkommit, markeringSläkting. Den är sann om individen är en släkting och falsk i annat fall.

Datamodellen kommer nu inte att se likadan ut som domänmodellen. Det är då viktigt att ha domänmodellen kvar för att kunna förstå verksamheten och vad som gäller.

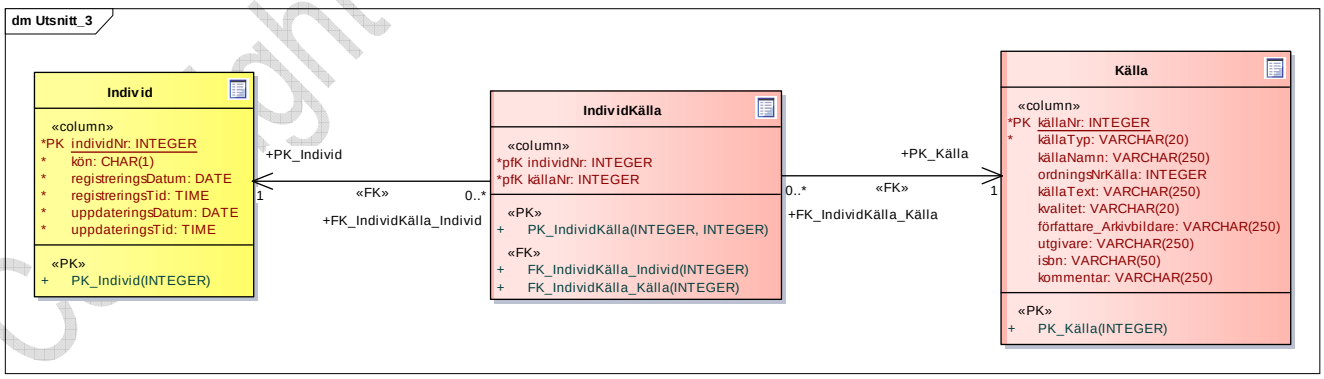
Skapa relationstabeller

I domänmodellen kan det finnas ett antal relationer som är av multipliciteten många till många, d v s n..* - m..* , där n och m är heltal (även noll). I en relationsdatabas kan vi inte hantera detta utan de måste lösas upp genom att vi skapar en relationstabell emellan de båda tabellerna. Studera följande exempel.



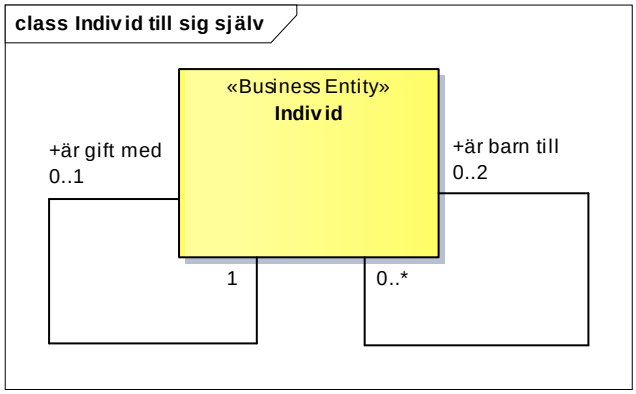
Här har vi ett många till många exempel. En individ kan förekomma i många källor, exempelvis husförhörslängder födelseböcker m fl. Å andra sida så innehåller en husförhörsläng många individer. Nollorna säger att relationen inte behöver finnas. Vi kan alltså behöva lagra uppgifter om en källa utan att den har någon koppling till några av våra lagrade individer. På samma sätt kan vi lagra uppgifter om individer utan att vi har en koppling till någon källa.

I datamodellen kommer vi att ha tabeller för både individ och källa. För att lösa upp vår många till många relation skapar vi en tabell emellan som ges det samansatta namnet av de två andra tabellerna. I detta fall IndividKälla.



Vi kan här se att många till många multipliciteten hamnat på relationstabellen. Och multipliciteten till de ursprungliga tabellerna blir alltid 1. Relationstabellen får som både primär nyckel PK och främmande nyckel FK de båda andra tabellernas primära nycklar.

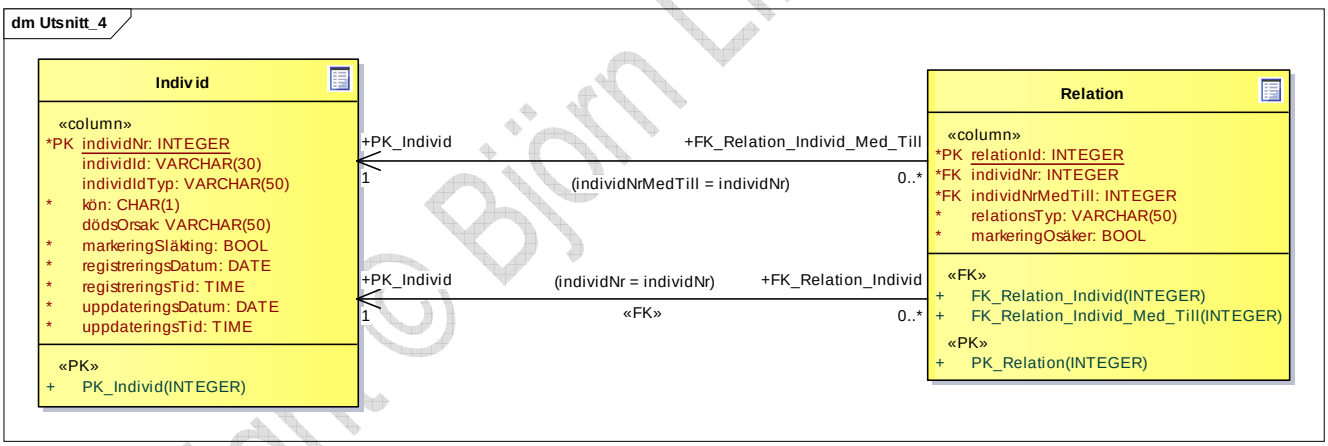
Domänobjekt kan ha relationer till sig själva. För dessa måste vi också skapa relationstabeller.¨ Relationerna kan vara en eller flera. Nedan finns ett exempel för domänobjektet Individ.



Från den vänstra associationen kan vi utläsa att, **en** Släkting är gift med **noll eller en** Släkting. En släkting behöver alltså inte vara gift med någon annan släkting (0), men om den är det så kan det endast vara med en (1). Vår verksamhet kan alltså inte hantera mångiften.

Den andra associationen säger att, **noll till många** Släktingar är barn till **noll till 2** Släktingar. Här hanterar vi endast blodsband. Varför står det då noll till 2? Måste inte alltid ett barn ha 2 biologiska föräldrar? Jo det är förvisso sant, men i våra modeller kommer det ofta att stå noll (till något) eftersom vi kan sakna den fulla informationen och måste kunna hantera delmängder av information. Det betyder exempelvis i fallet ovan att vi kanske bara känner till den ene föräldrarna eller **inte någon** av dem.

I datamodellen kommer det att se ut enligt nedan.



Observera relationstabellen skulle kunna ha **två** primära nycklar, IndividNr och individNrMedTill för de **två** individer som ingår i relationen. Så är det inte i detta fall för den genealogiska modellen.
Detta beror på att vi i ett släktforsknings-sammanhang måste kunna hantera att **två** individer har flera olika typer av relationer till varandra. För att tabellen ska ha unika nycklar har de fått ett eget attribut ”realtionsId” som primärnyckel. På detta sätt får vi inga dupletter.

Efter detta steg fortsätter vi med steget ”Normalisera”, se vidare avsnitt nedan.

De första stegen utan en befintlig domänmodell

Vi har alltså ingen domänmodell att utgå ifrån.

De två stegen är;

- Lista objekt
- Skapa tabeller

Lista Objekt

Det första blir alltså att försöka lista vilka objekt vi har som kan bli tabeller. Med objekt menas saker, information mm. Steget utförs med fördel som en gruppövning med ett flertal personer som är insatta i ämnet. Viktigt under arbetet är att inte ifrågasätta de förslag som kommer upp från olika personer utan att bara lista dem. Det kan vara objekt, men det kan även vara egenskaper eller andra saker. Den bedömningen gör vi senare när vi behandlar listan i detalj. Att inte ifrågasätta de förslag som kommer upp gör att kreativiteten hålls vid liv och inte hämmas.

Resultatet kan bli en ganska lång lista med olika saker som kan bli tabeller eller innehåll i tabeller.

Skapa Tabeller

Nu startar arbetet med att gå igenom listan rad för rad.

Hitta tabeller

Värdera under diskussion i gruppen om det som listats på en rad är en tabell eller något annat. Det skulle exempelvis kunna vara en egenskap hos något, d v s ett attribut eller något annat (kanske ett värdeförråd).

Om det inte är en tabell, lämna då den raden tills vidare och gå till nästa rad i listan.

När en rad plockats ut, rita då upp en tabell för denna. Försök sedan hitta vad som är identifierare för denna tabell (sak), d v s vad som unikt identifierar (är nyckel till) varje rad i tabellen. Det kan vara en eller flera attribut (egenskaper). Tänk på att nyckeln ska hålla över tiden.

Låt oss exempelvis tänka att vi har en tabell som heter ”Bil”. Vad är då nyckel till bil? Det skulle kunna vara registreringsnummer, men vissa registreringsnummer följer ägaren och inte bilen. En taxi som upphör att vara taxi och övergår i privat ägo får ett nytt registreringsnummer.

För att få en unik nyckel som håller över tiden använder vi istället bilmodell + chassinummer. Det blir alltså en nyckel som består av två attribut.
Ett exempel på värde är: Volvo V70 + YV1SW59G241400818.

Fortsätt sedan med nästa rad i objektlistan på samma sätt och gå vidare tills hela listan gåtts igenom.

Vi har då fått upp ett antal tabeller.

Behandla resterande rader i objektlistan

Gå sedan igenom objektlistan igen och ta hand om de rader som inte är avbockade. Dessa kan nu exempelvis vara egenskaper, d v s attribut i tabeller eller exempelvis värdeförråd.

Placera ut funna attribut på ritade tabeller eller värdeförråd på attribut. Om ett attribut inte passar på någon tabell kanske en tabell saknas.

Komplettera tabellerna

Till sist är det dags att komplettera tabellerna. Gå igenom alla ritade tabeller och diskutera fram vilka ytterligare attribut de ska innehålla.

Rita relationer

Till sist är det dags att beskriva relationerna mellan tabellerna. Rita associationer och beskriv multipliciteten (se avsnitt om multiplicitet i ”Beståndsdelar i en datamodell” ovan). Om ett många till många förhållande hittas, måste det brytas upp genom att en relationstabell skapas emellan.

Navigering sker alltid från många till ett vilket betyder att de attribut som är nyckel i den tabell som har multipliciteten 1 eller 0..1, även måste finnas i den andra tabellen.

Normalisering

Normalisering uppfanns av den Brittiska forskaren Edgar Codd i början av 1970-talet.

Normalisering är ett steg som genomförs för att se till att tabellerna är bra ur ett design-perspektiv och att de ska gå enkelt att använda i ett datasystem. Det ska även se till att tabellerna är bra ur ett förvaltnings-perspektiv, d v s att de enkelt ska kunna förändras och byggas ut.

VARNING! På nätet finns det många som beskriver hur normalisering går till och hur mormalformerna ska tolkas. Här är det på plats med en varning. Tyvärr finns det ett antal både dåliga och rent felaktiga beskrivningar. De bästa och mest tillförlitliga är enligt min åsikt de som finns på så kallade wikies. De är ofta också granskade och påverkade av ett flertal personer.

Första normalfomen, 1NF

En tabell är i första normalformen, om varje attribut endast innehåller atomära värden och värdeinnehållet för varje attribut endast innehåller ett singelvärde.

Med atomära värden menas att det inte får vara ett sammansatt innehåll.

Exempel på sammansatta innehåll, d v s icke atomära värden;
namn = ”Karl Svenson”
postadress = ”122 25 Stockholm”

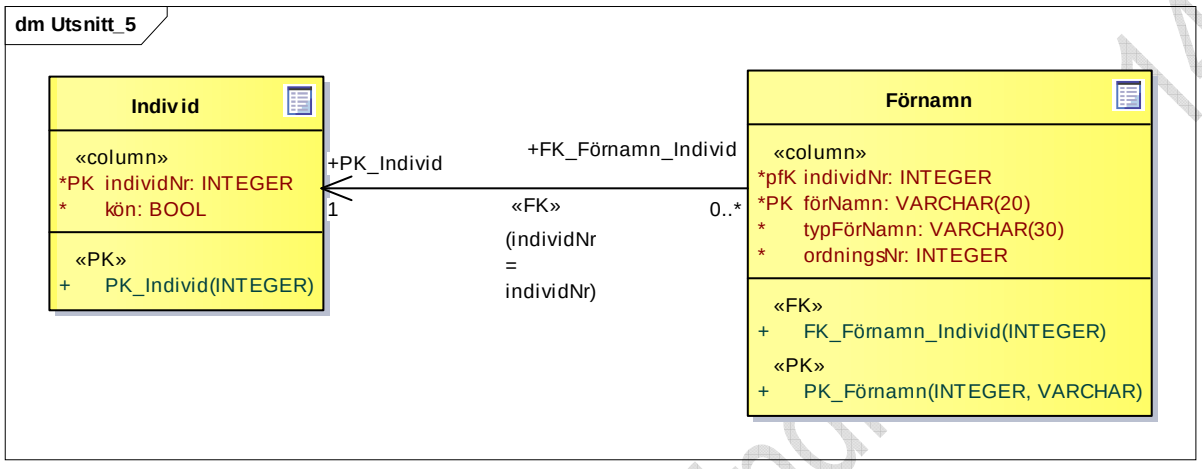
I det första fallet för- och efternamn. I det andra postnummer och ortsadress. Lösningen är att dela upp dessa attribut i flera.

Med singelvärde menas att attributet inte får innehålla flera värden.

Exempel på attribut som inte innehåller singelvärde;
förnamn = ”Karl Sven Oskar”

Förnamn innehåller i exemplet ovan 3 värden, 3 förnamn. Lösningen är att skapa en egen tabell för förnamn enligt exempel nedan.

Tabellen förnamn har den sammansatta nyckeln bestående av IndividNr och förNamn. Detta beror på att flera individer kan ha samma förnamn. IndividNr är också främmande nyckel till tabellen Individ. Vidare finns i tabellen Förnamn även ett attribut som berättare vad det är för typ av förnamn samt ett attribut som beskriver ordningen bland förnamnen.



Andra normalformen, 2NF

En tabell är i andra normalformen, om den är i första samt inga icke nyckelattribut inte är beroende av endast delar av primärnyckeln.

Alla attribut ska alltså vara beroende av hela primärnyckeln.

Exempel:

turnering PK	**år PK**	**vinnare**	**underlag**
Franska öppna	2012	Rafael Nadal	Grus
Franska öppna	2013	Rafael Nadal	Grus
Wimledon	2012	Roger Federer	Gräs
Wimledon	2013	Andy Murray	Gräs
Australian open	2012	Novak Djokovic	Hard court
Australian open	2013	Novak Djokovic	Hard court

Vinnare i tabellen ovan är beroende av båda primärnycklarna. Underlag är däremot endast beroende av primärnyckeln turnering. Underlag bryts därför ut till en egen tabell för turnering enligt exempel nedan.

turnering PK	**år PK**	**vinnare**
Franska öppna	2012	Rafael Nadal
Franska öppna	2013	Rafael Nadal
Wimledon	2012	Roger Federer
Wimledon	2013	Andy Murray
Australian open	2012	Novak Djokovic
Australian open	2013	Novak Djokovic

turnering PK	**underlag**
Franska öppna	Grus
Wimledon	Gräs
Australian open	Hard court

Tredje normalformen, 3NF

En tabell är i tredje normalformen, om den är i andra och alla attribut är beroende av primärnyckeln och endast primärnyckel (så hjälp mig Codd).

Detta betyder att inga beroenden får finna mellan icke nyckelattribut.

Exempel:

turnering PK	**år PK**	**vinnare**	**födelseårVinnare**
Franska öppna	2012	Rafael Nadal	1986
Franska öppna	2013	Rafael Nadal	1986
Wimledon	2012	Roger Federer	1981
Wimledon	2013	Andy Murray	1987
Australian open	2012	Novak Djokovic	1987
Australian open	2013	Novak Djokovic	1987

Attributet vinnare ovan är beroende av primärnycklarna turnering och år, men födelseårVinnare är däremot beroende av vinnare som inte är primärnyckel.
Lösningen är att bryta ut födelseår vinnare till en egen tabell enligt nedan. I den tabellen blir då vinnare primärnyckel.

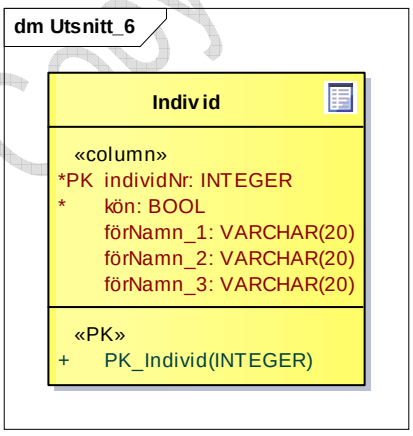
turnering PK	**år PK**	**vinnare**
Franska öppna	2012	Rafael Nadal
Franska öppna	2013	Rafael Nadal
Wimledon	2012	Roger Federer
Wimledon	2013	Andy Murray
Australian open	2012	Novak Djokovic
Australian open	2013	Novak Djokovic

Vinnare PK	**födelseårVinnare**
Rafael Nadal	1986
Roger Federer	1981
Andy Murray	1987
Novak Djokovic	1987

Utöver tredje normalformen

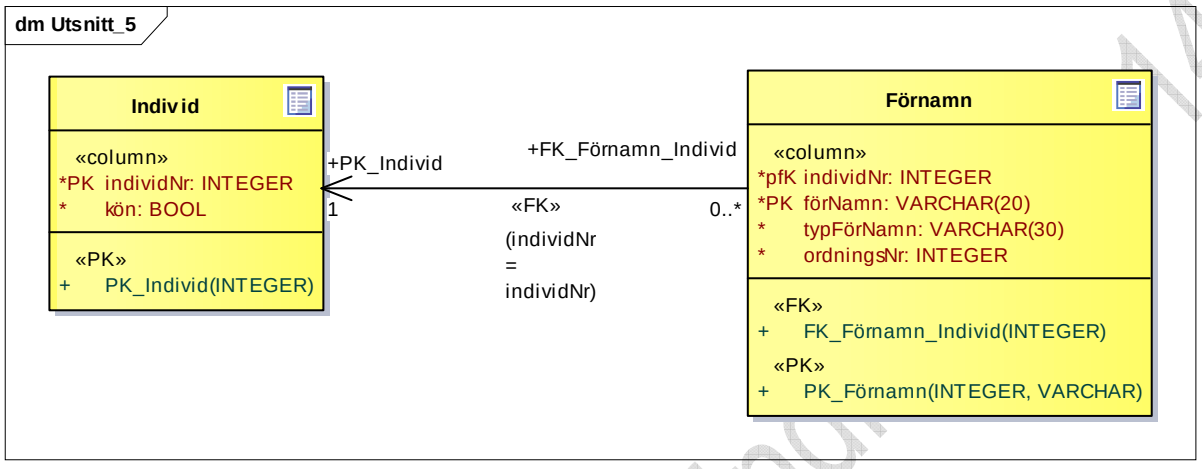
Det finns förutom de 3 första normalformerna, ytterligare ett antal definierade, men de 3 första är de viktigaste och som det räcker mycket bra med. Däremot finns det några andra bra regler som är mycket användbara. Dessa följer här.

Undvik upprepningar. Tabellen nedan är exempel på upprepningar.



Upprepningar är ingen bra lösning. I exemplet ovan finns plats för 3 förnamn. Detta betyder att ifall de flesta har endast ett, så har vi en mängd tomma förnamnsfält i onödan. Dessutom får vi problem när vi upptäcker att vi behöver lagra fler än 3 stycken. Om vi exempelvis behöver lagra ytterligare 2 leder detta till ändringar i databasen. Det stannar dock inte där. Vi kommer även att behöva ändra i vår applikation.

Lösningen är att bryta ut förnamn enligt exemplet nedan som vi sett tidigare. Det skapar en flexibel lösning.



Bygg inte in egenskaper i attributen.

Egenskaper kan ges i attribut explicit eller implicit. Båda varianterna är mycket dåliga lösningar, där den implicita varianten är farligast för den kan kanske vara svår att upptäcka.

Ett exempel på ett attribut med **explicita** egenskaper:

Vi har i tabellen nedan attributet förnamn.

individNr PK	förnamnNr PK	förnamn
22	1	Karl
22	2	Sven
22	3	”Kalle”

Vi har då exempelvis ha följande värden för en individs förnamn:

Karl
Sven
”Kalle”

En understrykning markerar tilltalsnamn. Namn inom citattecken markerar smeknamn.

Denna typ av markeringar kan användas i in- och utdata, men hör inte hemma i en databas.

Lösningen är att införa ytterligare ett attribut som är en typning av förnamn. Ett exempel syns nedan.

individNr PK	förnamnNr PK	förnamn	förnamnTyp
22	1	Karl	Tilltalsnamn

22	2	Sven	Förnamn
22	3	Kalle	Smeknamn

En typning skapar en flexiblare lösning. Vi kan nu enkelt byta utseende på hur vi vill visa exempelvis tilltalsnamn i ett gränssnitt utan att det påverkar databasen och tabellernas innehåll. Det kan ju dessutom vara så att vi vill visa det på olika sätt i olika sammanhang.
Ett exempel på ett attribut med **implicita** (inbyggda) egenskaper:

individNr PK	**personNr**
1	6805262349
2	7112051235

Personnummer är 10 siffror. Den näst sista siffran (den sista i födelsenumret) beskriver kön. En udda siffra står för en man och en jämn står för en kvinna. Individ nummer 1 är alltså en kvinna och individ nummer 2 en man.

Ovanstående är ingen bra lösning. En bra lösning är att skapa ett eget attribut för kön. Det kan då se ut enligt följande:

individNr PK	**personNr**	**kön**
1	6805262349	kvinna
2	7112051235	man

Denna lösning fungerar även bättre ur ett internationellt perspektiv, eftersom det endast är i Sverige vi har personnummer. Lösningen är även framåtriktad. Vad gör skatteverket om födelsenumren inte skulle räcka till en dag?

Tänk objekt och relationer

Här är ett exempel med objekt och relationer.

Utgångspunkten är att vi vill hålla reda på Butiker, deras Varor och deras Leveransområden. Vi har då dessa 3 objekt.

Butik har ett många till många förhållande till Vara, eftersom samma vara kan säljas av många butiker och en butik kan sälja många varor.

Vi får då skapa en relationstabell för att hålla reda på detta enligt nedan där både butik och vara är både PK och FK.

ButikVara	
Butik pfK	Vara pfK

Vi ger tabellen det sammansatta namnet ”ButikVara”.

På liknande sätt är det mellan objekten Butik och Leveransområde som även det är ett många till många förhållande. Vi skapar då ytterligare en relationstabell enligt nedan, med det sammansatta namnet ”ButikLeveransområde.

ButikLeveransområde	
Butik pfK	Leveransområde pfK

Vi kan nu enkelt hålla reda på de båda relationerna utan redundans.

Men här gömmer sig en fara!

Med detta utseende på tabellerna måste alla varor från en butik kunna levereras till den butikens samtliga leveransområden. Tänk om det inte är så. Tänk om ett antal varor är undantagna från vissa leveransområden, kanske leveransområden som ligger långt bort.

Om så är fallet är leveransområde inte kopplat till butik utan till en butiks vara, d v s i vår modell till relationsobjektet ButikVara.

Vi får då en ny relationstabell som speglar relationen mellan ButikVara och Leveransområde. Vi kallar den för det sammansatta namnet, d v s ButikVaraLeveransområde och den kommer att se ut enligt nedan.

ButikVaraLeveransområde		
Butik pfK	Vara pfK	Leveransområde pfK

Denormalisering

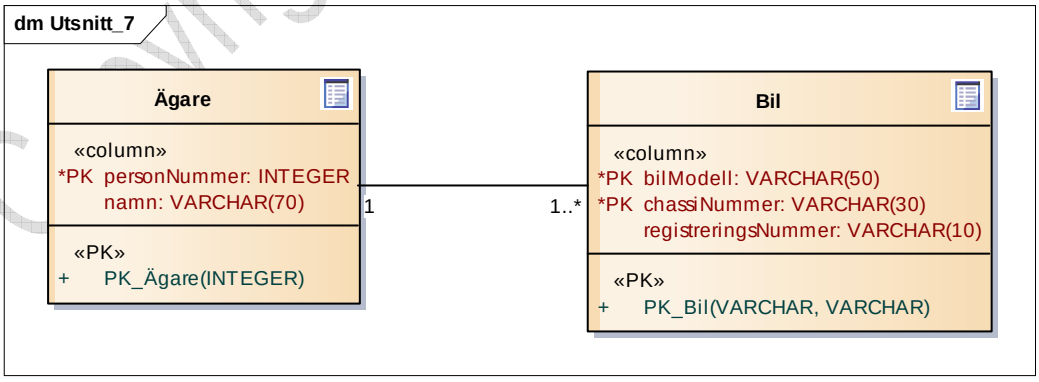
Denormalisering är en aktivitet där vi medvetet frångår normaliseringsreglerna. Detta gör vi för att:

- öka prestanda
- skapa enkelhet
- optimera sökvägar

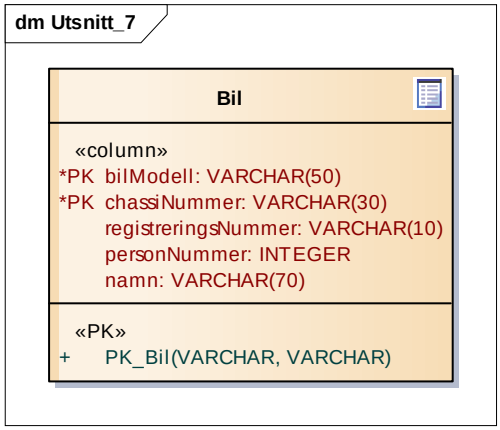
Några exempel på denormalisering.

Vi kanske vill skapa en tabell för ”Kund”, och för varje kund vill vi lagra telefonnummer. En kund kan dock ha många telefonnummer, men vi kanske begränsar oss till att lagra två stycken. Vi skapar då två attribut. Exempelvis telefonNr_1 och telefonNr_2 på tabellen Kund i st f att skapa en egen tabell för telefonnummer.

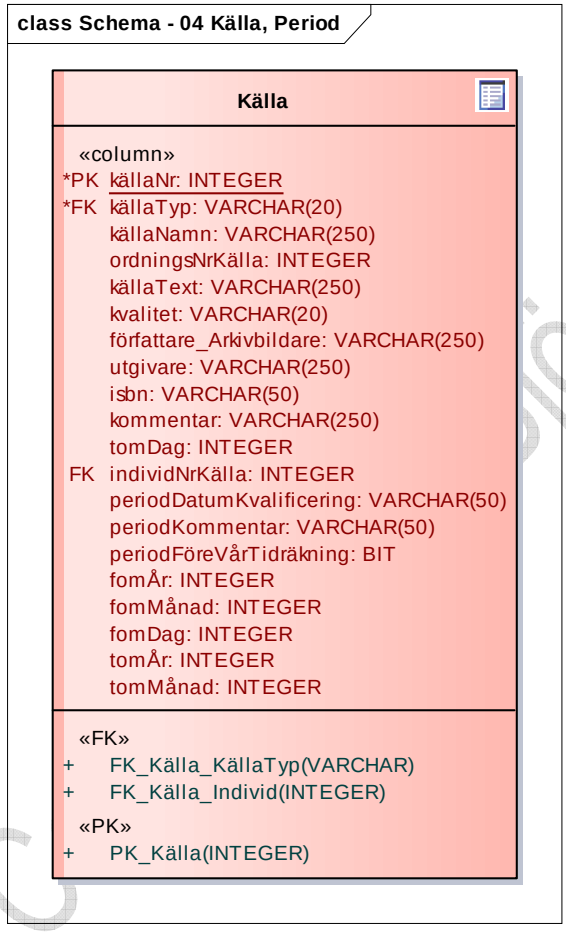
Ett anat exempel med bil och ägare. Det kanske ser ut enligt följande:



Men nu är det kanske så att vi inte är intresserade av vilka olika bilar en person äger. Vi önskar endast hålla reda på vem ägaren är till en bil. Därför lägger vi dessa uppgifter på bil även om det inte är egenskaper på en bil. Det blir då enligt följande.



Ytterligare ett exempel, denna gång från modellen för släktforskning. Där finns en tabell som heter källa. Den ser ut enligt följande.



Som vi ser innehåller tabellen attribut som ”isbn” ”utgivare” och ”författare”. Det är attribut som hör till böcker och dyl. Det är inte attribut som hör till kyrkböcker, mantalslängder mm. Rent logiskt skulle vi kanske ha skiljt på olika källor. I detta fall har vi för enkelhetens skull valt att inte göra så även om vi kommer att få ett antal tomma attribut i vissa samanhang. En viktig anledning till att vi har gjort detta är att det finns 10 relationer till källa. Det skulle ha kunnat inneburit att vi för varje utbruten typ av källa kanske fått 10 nya relationer med ytterligare attribut i de tabeller som relationerna går till.

Det sista exemplet på denormalisering är skapandet av hjälptabeller för olika ändamål. Några exempel är följande:

- Samlingstabell med sammansatta uppgifter. Kan kanske även lösas med skapande av vyer.
- Indextabeller av olika slag. Används ofta för att snabba upp sökningar.
- Nya relationstabeller för kortare sökvägar. Detta betyder att vi exempelvis i st för att navigera över 6 olika tabeller skapar en tabell som direkt knyter samman den första och den sista i kedjan.

Ofta är denna typ av denormalisering inte beroende på hur saker hänger samman i verkligheten utan istället på hur vi behöver behandla dem i system och gränssnitt.

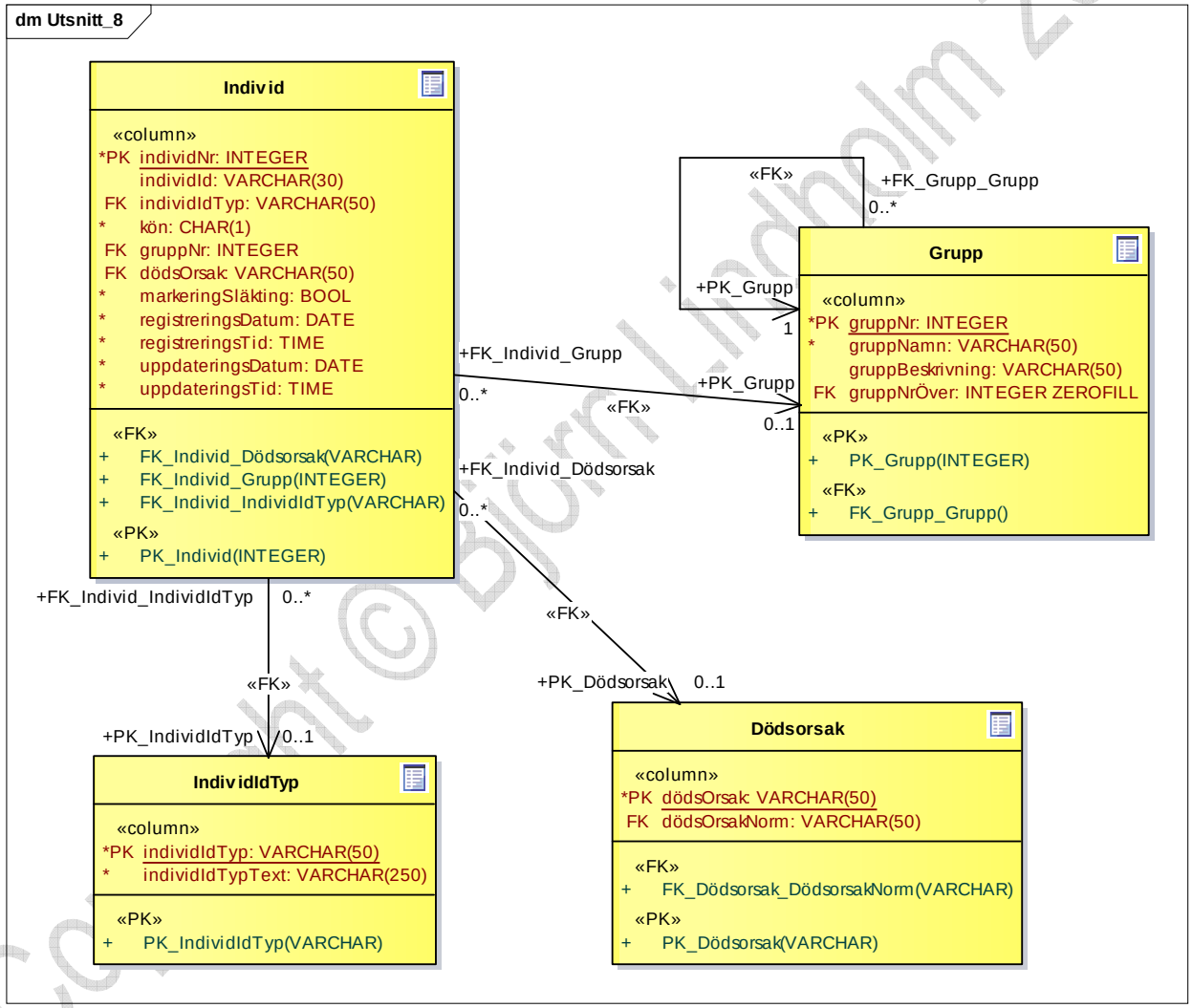
Skapa främmande nycklar

Det sista steget är att skapa främmande nycklar.

All navigering i datamodellen sker i pilens riktning från ett ”många” förhållande emot ett ”ett” förhållande. Exempelvis 0..* till 1 eller 1..* till 0..1. För att navigeringen tekniskt ska vara möjlig måste den tabell som navigering sker ifrån, d v s den som ha mångaförekomsten ha ett eller flera attribut som i princip är detsamma som nyckeln eller nycklarna i den tabell som navigering ska ske till.

Attributen behöver inte ha samma namn men de ska ha samma format och värdeförråd.

I exemplet nedan från släktforskningsmodellen ser några exempel.



Tabellen individ har tre främmande nycklar, FK för att navigering ska kunna ske till de olika tabellerna grupp, individidtyp och dödsorsak.