Kapitel 2

ER-modellering

2.1 Konceptuella datamodeller

Om man ska skapa en databas som beskriver en del av verkligheten, t ex ett företag, brukar man börja med att göra en beskrivning av hur den delen av verkligheten ser ut och fungerar. Denna beskrivning på hög nivå kallas en *konceptuell* eller *begreppsmässig* beskrivning. I stället för "beskrivning" säger man ofta *schema*.

Det konceptuella schemat behöver egentligen inte ha någonting med datorer att göra, utan är bara en beskrivning av verkligheten som lika gärna skulle kunna användas t ex av någon som vill analysera hur företaget fungerar. Om man vill skapa en databas måste det konceptuella schemat översättas till ett schema som går att mata in i en databashanterare. Om man använder en relationsdatabashanterare så består det schemat av en eller flera tabeller.

> **Repetition av kapitel 1:** Ett *schema* beskriver vilka data man kan lagra i en viss databas, medan en *datamodell* beskriver hur scheman kan se ut. Som exempel kan vi ta en relationsdatabas, dvs en databas där data lagras i tabeller. Databasens *schema* utgörs av vilka tabeller som finns i databasen, och vilka kolumner de har, medan *datamodellen* är själva det faktum att databasen byggs upp av tabeller.
> De data som man sen verkligen lagrar i tabellerna kallas, förstås, *data*.
> Tänk också på att det finns en del språkförbistring: Ibland används ordet "datamodell" för att beteckna det som vi här kallar för "schema".

Den konceptuella beskrivningen görs med hjälp av en *konceptuell datamodell*, och består av ett schema uttryckt i den modellen.

En vanlig konceptuell datamodell är den så kallade *ER-modellen*, och det är den vi ska använda i det här kapitlet. "ER" står för "Entity-Relationship", dvs ungefär "saker" och "samband". En beskrivning som man skapar med hjälp av ER-modellen ritas upp som en bild med olika figurer, och den kallas *ER-schema* eller *ER-diagram*. .

2.2 Varför ska jag lära mig det här?

Idag använder man för det mesta relationsmodellen när man arbetar med databaser. När man ska skapa en databas skulle man därför kunna tycka att det vore enklast att direkt bestämma tabellernas utseende, i stället för att gå omvägen med att först göra ett konceptuellt schema.

Ett schema med tabeller kräver dock en del speciallösningar som visserligen är enkla när man kan dem, men som ändå gör att man inte kan koncentrera sig helt på den del av världen som man vill avbilda i databasen, och hur den fungerar. I stället måste man tänka på "tabelldetaljer", som till exempel att man ibland behöver införa extra hjälptabeller. Tabeller kan också fort bli oöverskådliga, med långa rader av kolumner (!) och med mer eller mindre kryptiska kopplingar mellan de olika tabellerna.

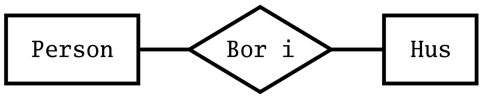
ER-diagram är därför lättare att jobba med, i synnerhet för den som är ovan vid tabeller. Det brukar leda till en bättre design i slutänden om man börjar med att rita ER-diagram, och sedan översätter till tabeller, i stället för att försöka konstruera tabeller direkt.

2.3 Grunderna för hur man ritar ER-diagram

Gör så här:

Rita upp de *typer av saker* som ska finnas i databasen. De kallas *entitetstyper*, och ritas som fyrkantiga lådor:

Rita sen upp de *samband* som finns mellan de olika typerna av saker. De kallas *sambandstyper*, och ritas som diamanter mellan de fyrkantiga lådorna:

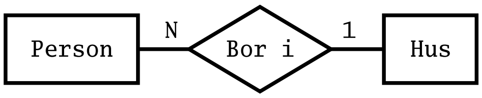


Det här ER-diagrammet betyder alltså att det finns personer, det finns hus, och personerna bor i husen. (ER-diagrammet säger dock inget om hur många personer och hus det är, eller vilka, och vilka personer som bor i vilka hus. Allt det där är data, och ER-diagrammet är ett schema.)

"Sambandstyp" heter "relationship type" på engelska, och "samband" heter "relationship", men undvik att kalla dem för "relation" eller "relationstyp" på svenska. I databassammanhang är en relation en tabell i relationsmodellen.

2.4 Hur många personer bor i varje hus?

Man kan också rita ut mer exakt vad det är för sorts sambandstyp:



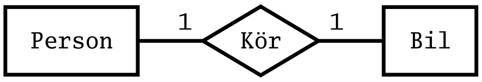
"**N**" betyder att det kan bo flera personer i varje hus, och "**1**" betyder att varje person bara kan bo i ett hus. Det kallas ett *många-till-ett-samband*.

"Många" betyder i det här sammanhanget noll, ett eller flera. Det kan finnas hus som det inte bor någon i, det kan finnas hus som det bara bor en person i, och det kan finnas hus som det bor en miljon personer i.

Sambandstyper kan vara av tre olika slag, så kallade *kardinalitetsförhållanden*:

- *Ett-till-ett-samband*[1]

  eller *1:1-samband*. En sambandstyp där en sak av något slag kan höra ihop med en sak av ett annat slag, och varje sak av det andra slaget kan höra ihop med en sak av det första slaget. Exempel: En person kan vid ett och samma tillfälle bara köra en bil, och varje bil kan bara köras av en person.[2]

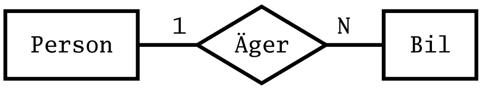


- *Ett-till-många-samband* eller *1:N-samband*. En sambandstyp där en sak av något slag kan höra ihop med flera saker av ett annat slag, men varje sak av det andra slaget kan bara höra ihop med en sak av det första slaget. Exempel: En person kan äga flera bilar, men varje bil kan bara ägas av en person. Om man vänder på det och börjar med bilarna blir det i stället ett *många-till-ett-samband* (*N:1-samband*).

---

[1] På svenska ska sammansatta ord inte skrivas isär. Alltså heter det exempelvis "ett-till-ett-samband", och inte "ett-till-ett samband" eller "ett till ett samband". Det heter "ER-diagram" och inte "ER diagram". Det heter "databas" och inte "data bas". Alla sammansatta ord skrivs ihop. Varför göra fel när det är så lätt att göra rätt? Läs mer på **http://www.skrivihop.nu/**.

[2] Jaja, *kan* kan man väl. Men det brukar inte bli så bra.



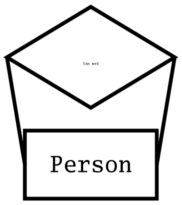
- *Många-till-många-samband* eller *N:M-samband*. En sambandstyp där en sak av något slag kan höra ihop med flera saker av ett annat slag, och varje sak av det andra slaget kan höra ihop med flera saker av det första slaget. Exempel: En person kan äga flera hus, och varje hus kan ägas gemensamt av flera personer.

Faktaruta: ER-diagram kan ritas på flera olika sätt. Det som brukar skilja mest är just hur man ritar sambandstyper, och hur man då anger kardinalitetsförhållandena (till exempel ett-till-många). Några exempel visas i avsnitt 2.21 på sidan 49.

2.5 Skilj på schema och data

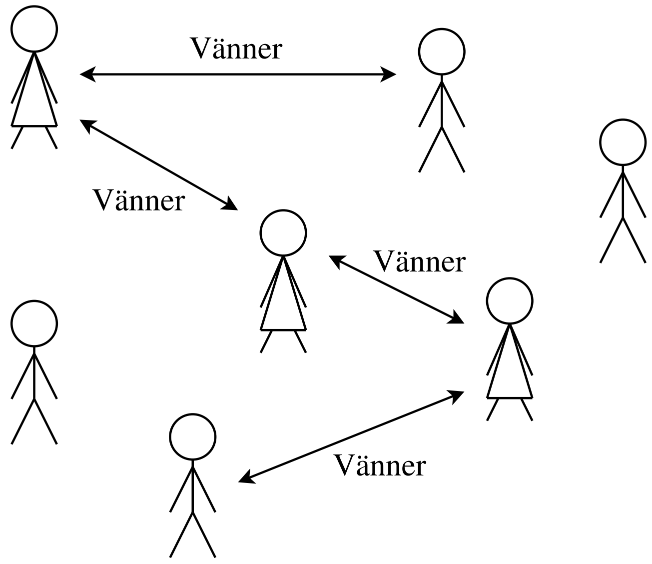
ER-diagrammet är ett schema, och beskriver vilka data som kan lagras. Det är inte en avbildning av datat.

Det här ER-diagrammet säger alltså inte att det finns en person som är vän med sig själv, utan det säger att personer kan vara vänner med andra personer:



Om **Person** alltså är en *entitetstyp*, så brukar man kalla de enskilda personerna för *entitetsinstanser*. (Ja, egentligen är det ju inte personerna, utan deras representation i databasen.) Man kan också kalla personerna för *entiteter*, men det kan vara förvirrande eftersom den benämningen ibland används även för entitetstyperna. På samma sätt talar man om *sambandsinstanser*.

Så här skulle det kunna se ut om man ritade upp *instanserna* i en databas med ER-schemat ovan:



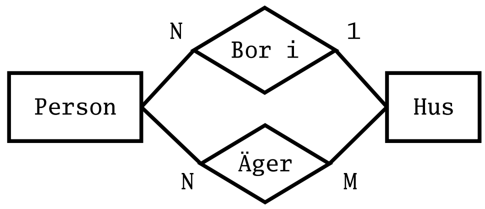
2.6 Skilj på ER-diagram och tabeller

Även om man ofta översätter ER-diagram till tabeller, för att kunna lagra sin databas i en relationsdatabashanterare, *så har ER-diagrammet i sig inget med tabeller att göra*.

Det man ritar ut i ett ER-diagram är alltså *inte* tabeller och kopplingarna mellan dem, även om en del (men inte alla) av entitetstyperna sen kan översättas till tabeller, och en del (men inte alla) av sambandstyperna kan översättas till kopplingar mellan tabeller.

2.7 Flera sambandstyper mellan samma entitetstyper

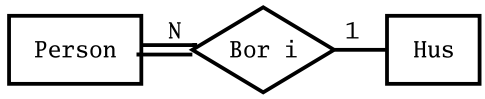
Det går bra att ha flera olika sambandstyper som binder ihop samma entitetstyper. Till exempel kan personer kanske bo i husen precis som tidigare, men de kan också äga husen:



Hus kan ägas gemensamt av flera personer, och varje person kan vara med och äga flera hus.

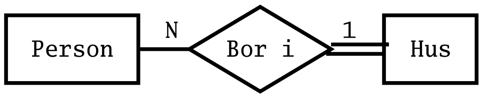
2.8 Fullständigt deltagande

Om varje person som är med i databasen måste bo i ett hus, kan man rita ut det villkoret med ett dubbelstreck mellan **Person** och **Bor i**:



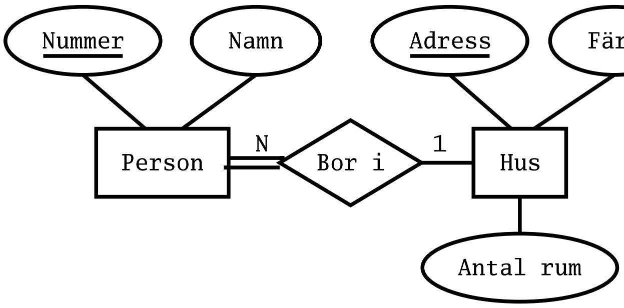
Det kallas *fullständigt deltagande*: alla personer som finns med i databasen måste delta i ett boendesamband. Motsatsen, dvs att det kan finnas personer i databasen som inte bor någonstans, brukar kallas *partiellt deltagande*. En sambandstyp kan ha fullständigt deltagande på ena sidan, på båda sidorna, eller inte på någon sida.

Ett annat exempel: Genom att ändra lite i figuren, säger vi i stället att en person kan bo i flera olika hus, men alla personer behöver inte bo någonstans, och i varje hus bor det bara en person. Det finns inga hus som det inte bor någon person i.



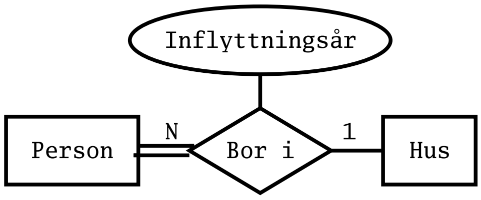
2.9 Attribut

Om sakerna eller sambanden har egenskaper, så kallar man dem *attribut*, och ritar dem som ovaler:



Att **Nummer** på **Person** är understruket betyder att personernas nummer är unika, dvs två personer kan inte ha samma personnummer. Personnumret kan alltså användas som nyckel. Samma sak med husens adresser.

Även sambandstyper kan ha egenskaper:



Inflyttningsår anger vilket år en person flyttade in i det hus hon bor i. I det här fallet skulle man också kunna sätta attributet **Inflyttningsår** på personen, eftersom varje person bor i exakt ett hus, och alltså har exakt ett inflyttningsår, men det är nog naturligare att låta inflyttningsåret höra till **Bor i**-sambandet.

Det finns också fall när det är nödvändigt att ha attributet på sambandstypen:

- Antag att vi även lagrar vilket hus personen bodde i *innan* det hus som hon bor i nu. Vi vill lagra inflyttningsåret även för det huset. Då kan det finnas två olika inflyttningsår för en person.

- Antag att vi ändrar N:1-sambandet till ett N:M-samband, och alltså låter en person bo i flera olika hus samtidigt. Då kan det finnas flera olika inflyttningsår för en person.

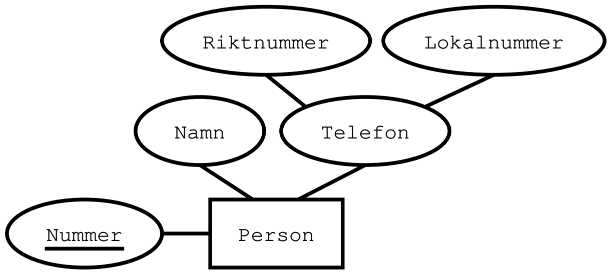
2.10 Ett vanligt fel

Nybörjare på ER-diagram gör ibland följande fel. Om de får uppgiften att rita ett ER-diagram för sin skola, så börjar de med att rita en ruta:

Det här är förstås galet, eftersom det här ER-diagrammet säger att det finns en entitetstyp som heter "skola", och alltså kommer databasen att innehålla ett antal skolor (entitetsinstanser av entitetstypen "skola"). "Skola" hade passat som rubrik till hela ER-diagrammet, men inte som en entitetstyp.

2.11 Sammansatta attribut

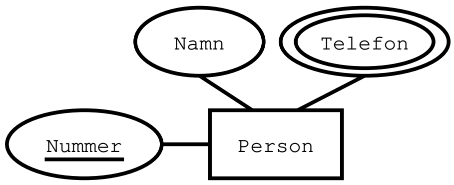
Ibland kan ett attribut vara sammansatt av flera delar, som hör ihop men som man även vill behandla var för sig. Då kan man rita det som ett *sammansatt attribut*:



Telefonnummer kan ju behandlas som en enhet (**0707-347013**), men om man ofta vill dela upp personerna i databasen efter riktnummer är det kanske bättre att dela upp numren i två delar (**0707** och **347013**).

2.12 Flervärda attribut

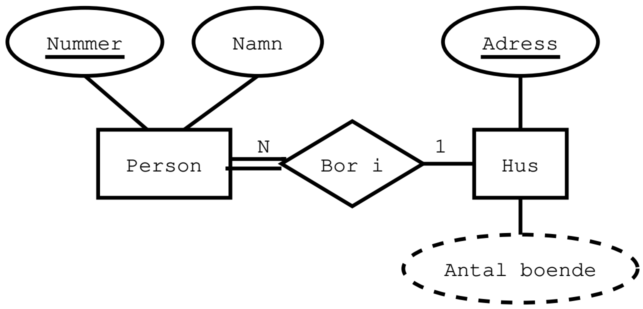
Attributen ovan är "enkla" i betydelsen att varje enskild entitet ("entitetsinstans") har högst *ett* värde på attributet. Varje person har ett namn, ett personnummer och ett telefonnummer. Ibland räcker inte det. Till exempel kanske man vill kunna lagra flera telefonnummer som hör till en person. Då kan man rita det som ett *flervärt attribut* (även kallat multipelt attribut) genom att använda en dubbelellips runt attributnamnet:



Att telefonnummer är ett flervärt attribut innebär att en person kan ha hur många telefonnummer som helst: noll, ett, två eller kanske en miljard.

2.13 Härledda attribut

En del attribut vill man kanske inte lagra i databasen, utan man kan räkna ut dem utifrån andra data som redan finns i databasen. Sådana *härledda attribut* markeras med en streckad oval:



I det här fallet ska man alltså inte lagra antal boende för varje hus, utan man räknar ut det på nytt varje gång man behöver värdet, helt enkelt genom att räkna hur många instanser det finns av "bor i"-sambandet som hör ihop med just det huset.

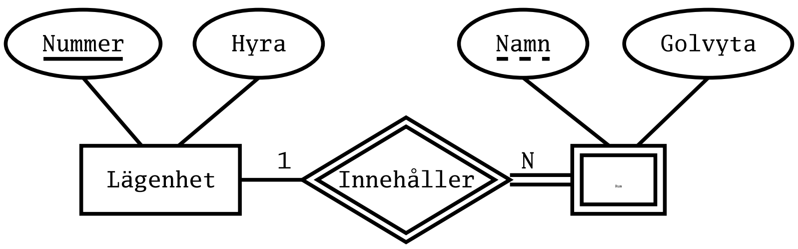
2.14 Svaga entitetstyper

Ibland vill man ha med saker som det är svårt att prata om som självständiga saker, utan de hör alltid ihop med en annan sak. Ta till exempel rummen i en lägenhet: köket, vardagsrummet badrummet, med flera. Om vi bara ska lagra data om en enda lägenhet i databasen så räcker det att säga "köket", för då finns det bara ett kök att välja på. Men om vi har många lägenheter, till exempel i en databas för ett bostadsföretag, så räcker det inte med "köket". Man vet inte vilket kök man menar, utan man måste alltid precisera sig med till exempel "köket i min lägenhet" eller "köket i lägenhet nummer 86".

Bostadsföretagets olika rum har alltså *ingen egen nyckel*. En nyckel i databassammanhang är något som man kan använda för att identifiera en viss sak, till exempel personnummer för personer. Personnumret **631211-1658** identifierar en viss person, men rumsnamnet **köket** räcker inte för att veta vilket kök man menar.

Om man ändå vill lagra rum i databasen, så kan man förstås hitta på en särskild nyckel för varje rum. Köket i min lägenhet kan till exempel heta **kök 17**, och köket i din lägenhet kan heta **kök 42**. Badrummet i min lägenhet kanske heter **badrum 203**, och ditt badrum heter **badrum 65**. Eller så ger man bara alla rum varsitt nummer: mitt kök är rum nummer **828**, och mitt badrum är rum nummer **703**.

Det kan vara lite opraktiskt, och det vore nog naturligare att fortfarande säga **köket** och **badrummet**, men dessutom tala om vilken lägenhet som det hör till. Då kan man använda en *svag entitetstyp*:



Det här ER-diagrammet säger att det finns lägenheter, som var och en har ett eget nummer. Lägenheterna innehåller rum, men rummen har bara namn, som inte är unika för hela databasen. Däremot är de unika inom en viss lägenhet: om man vet ett rumsnamn, och numret på den lägenhet som rummet hör till, så räcker det för att unikt identifiera rummet i hela databasen.

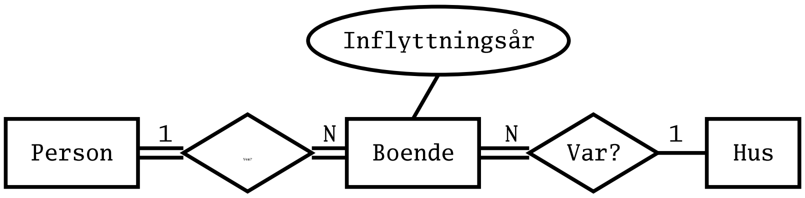
Vi har ritat **Rum** med en dubbel fyrkant, för att ange att det är en svag entitetstyp. Rumsnamnet **Namn** kallas *partiell nyckel*, och ritas understruket med ett (hmmm...) streckat streck. Sambandstypen **Innehåller**, som används för att identifiera vilken lägenhet ett rum tillhör, kallas *identifierande sambandstyp*, och ritas som en dubbel diamantbox. Entitetstypen **Lägenhet** kallas *identifierande entitetstyp*. Naturligtvis kan vi inte ha några lösa rum, som inte tillhör någon lägenhet, så vi ritar ut ett fullständigt deltagande med hjälp av ett dubbelstreck mellan **Rum** och **Innehåller**. Sambandstypen blir **1:N**, eftersom varje rum hör till en lägenhet, och en lägenhet kan innehålla flera rum.

Ibland säger man att den identifierande entitetstypen *äger* den svaga: en lägenhet "äger" de rum som den innehåller. Om man tar bort en viss lägenhet i databasen, vill man också ta bort rummen som den innehåller.

2.15 Objektifiering av sambandstyper

Ibland kan man välja mellan att använda sig av en sambandstyp eller en entitetstyp. Att använda en entitetstyp i stället för en sambandstyp betyder att man betraktar kopplingen som en egen sak, och inte bara som en koppling mellan två andra saker. Det kallas ibland *objektifiering*.

Till exempel kan vi göra om **Bor i**-sambandet till en egen entitetstyp, som vi kallar **Boende**:



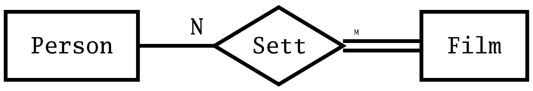
Nu ser vi inte längre ett boende som en sambandstyp som knyter ihop en person och ett hus, utan nu är ett boende en egen sak. Varje boende-entitet hör ihop med en person (den som bor) och med ett hus (huset hon bor i).

Det finns två anledningar till att man ibland vill objektifiera en sambandstyp:

- Det känns naturligare att betrakta kopplingen som en egen sak, än bara som en koppling mellan två andra saker. Det beror oftast på att vi är extra intresserade av just den kopplingen, och att vi vill spara extra mycket information om den.
- Man vill kunna ha flera instanser av samma sambandstyp mellan samma entiteter, till exempel att en person har sett en film flera gånger.

En sambandstyp som **Bor i**-sambandet i de tidigare exemplen är en koppling mellan en person och ett hus. *Man kan bara ha en sådan koppling mellan en viss person och ett visst hus*. Antingen bor man i huset, eller också inte. Det gäller alla sambandstyper. Det går inte att ha flera instanser av samma sambandstyp mellan samma entitetstyper.

Antag att vi ska rita ett ER-diagram som modellerar att personer har sett filmer:



N:M-sambandet betyder att en person kan ha sett flera filmer, och att en film kan ha setts av flera personer. Men vi kan inte ange att en person sett samma film flera gånger. (Jo, vi skulle kunna lägga på ett attribut **Antal** på sambandstypen, eller ett flervärt attribut **Datum** som anger alla datum som personen sett filmen. Men som det ser ut nu så har personen antingen sett filmen, eller också har hon inte sett den.)

Om vi i stället betraktar en filmtittning som en egen sak, och inte bara en typ av samband mellan personer och filmer, så kan vi ha flera olika tittningar där en viss person sett en viss film:

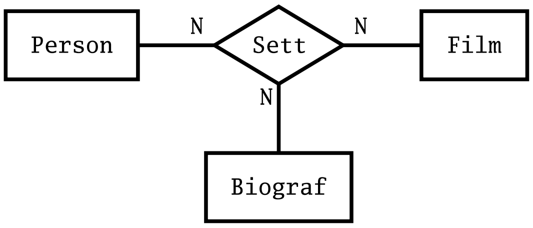
Nu har vi inte ritat ut några attribut, men vi måste hitta på en nyckel till den nya entitetstypen, ett "tittningsnummer" som inte behövdes för sambandet.

Om man har ett flervägssamband, dvs ett samband som binder ihop fler än två entitetstyper, är det ofta bättre att objektifiera det (se nedan).

2.16 Flervägssamband

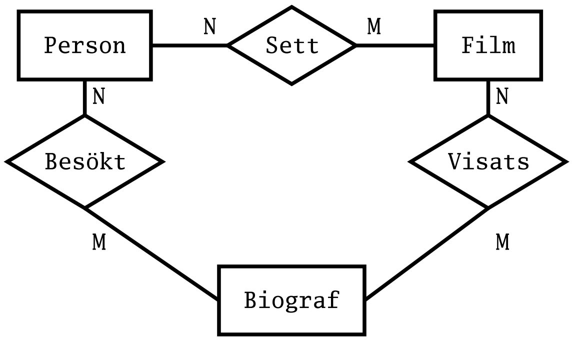
Bor i-sambandet mellan personer och hus ovan kallas ett *tvåvägssamband*, eftersom det binder ihop *två* entitetstyper. Man kan också ha sambandstyper som binder ihop fler entitetstyper, till exempel tre.

Kanske vill vi än en gång hålla reda på vilka personer som har sett vilka filmer, men vi vill dessutom veta vilka biografer de sett filmerna på. Jag har kanske sett **Spindelmannen** på **Rigoletto** och på **Filmstaden**, så jag kan uttala mig om hur upplevelsen är av att se just den filmen där. Däremot kan jag inte säga något om hur bra **Spindelmannen** passar ihop med ljudsystemet på **Röda kvarn**, för jag har inte sett den filmen på den biografen.



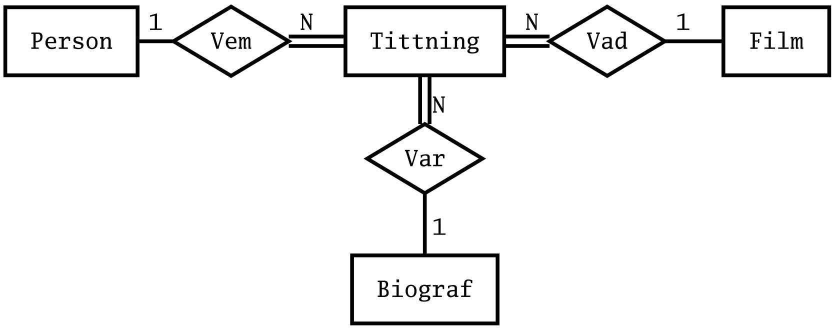
Det här ER-diagrammet visar att personer har sett filmer på biografer. Ett **Sett**-samband ("en instans av sambandstypen **Sett**") binder alltså ihop en person, en film och en biograf.

Ett trevägssamband går normalt *inte* att byta ut mot tre tvåvägssamband. Jämför med det här ER-diagrammet:



I en databas med det här schemat kan vi lagra att jag har sett **Spindelmannen**, att jag har besökt **Rigoletto**, och att **Spindelmannen** visats på **Rigoletto**, men inte att jag såg **Spindelmannen** just på **Rigoletto**. Jag kanske såg någon annan film när jag var på den biografen.

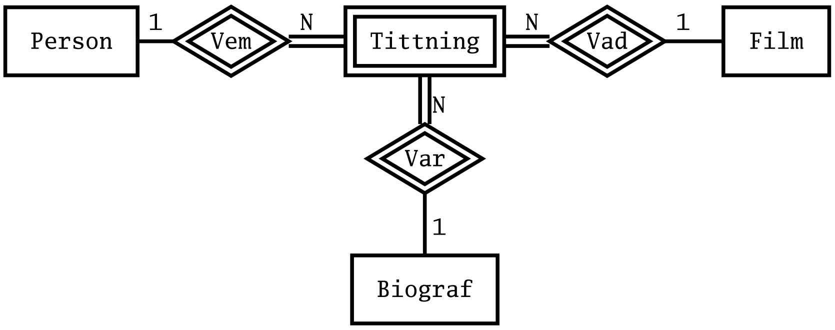
Det kan vara lite lurigt att rita ut kardinalitetsförhållandena i ett flervägssamband. Det är lättare om man låter **Sett** vara en entitetstyp i stället för en sambandstyp, dvs att man objektifierar **Sett**-sambandet:



Här ser vi alltså inte filmseende som en sambandstyp som knyter ihop en person, en film och en biograf, utan nu är filmseendet en egen sak. Varje filmtittningsentitet hör ihop med en person (vem), en film (vad) och en biograf (var).

ER-diagrammet med den nya entitetstypen **Tittning** säger inte rktigt samma sak som trevägssambandet, för nu kan vi lagra samma kombination av person, film och biograf flera gånger. Dessutom ska vi hitta på en nyckel till den nya entitetstypen. I stället kan man skapa en svag entitetstyp med tre identifierande samband. Då säger man samma sak som med trevägssambandet:[3]

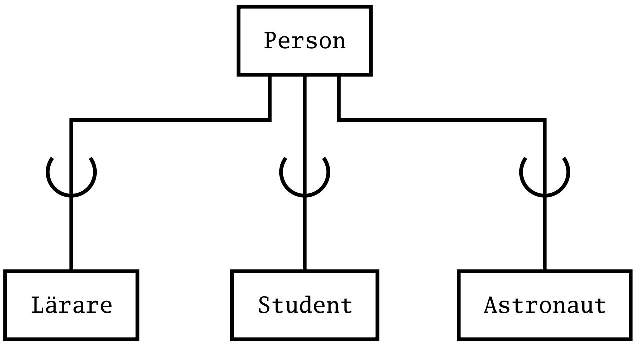
[3]Förutom att den svaga entitetstypen kan ha en partiell nyckel, och att man därför kan ha mer än en tittning som hör till en och samma kombination av person, film och biograf.



2.17 Utvidgade ER-diagram (EER)

Erfarenhet från objektorienterad[4] datamodellering visar att det ofta är praktiskt att låta en klass ärva egenskaper från en annan. Därför har man konstruerat en utvidgad ER-modell, den så kallade *EER-modellen*. "EER" står för "Enhanced[5] Entity-Relationship".

EER-modellen har en notation med flera olika typer av arv, men vi börjar med en enda: enkelt klass/subklass-samband:



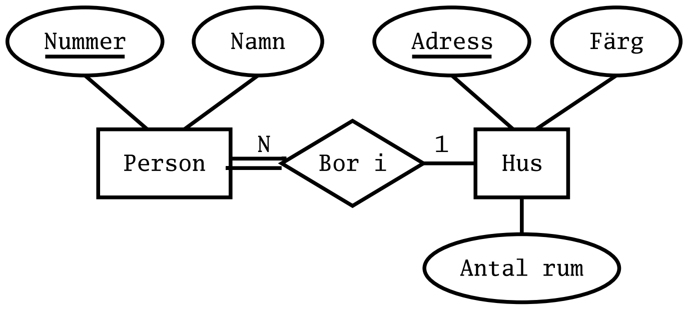
Det här säger att databasen innehåller personer. En del av de personerna är (förutom att de är personer) lärare, en del är studenter,

[4] En introduktion till objektorientering finns på bokens webbplats.
[5] Det engelska ordet "enhanced" betyder ungefär "förbättrad".

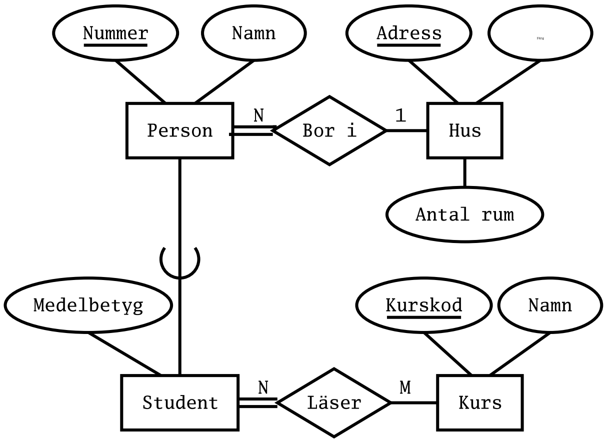
och en del är astronauter.

En student har alla egenskaper som en person har, till exempel att hon har ett personnummer och bor i ett hus, för en student *är* ju en person. Man säger att entitetstypen **Student** *ärver* från entitetstypen **Person**. Sen kan studenter ha ytterligare egenskaper, förutom personegenskaperna, till exempel att de kan läsa kurser. Precis som i objektorienterade sammanhang kallar vi **Student** för en *subklass* eller *härledd klass*, och **Person** för en *superklass* eller *basklass*. Strecket med "gaffelsymbolen" anger att **Student** är en subklass till **Person**, och att **Student** alltså ärver alla egenskaper som **Person** har. Man kan också se det som en delmängdssymbol: mängden studenter är en delmängd av mängden personer.

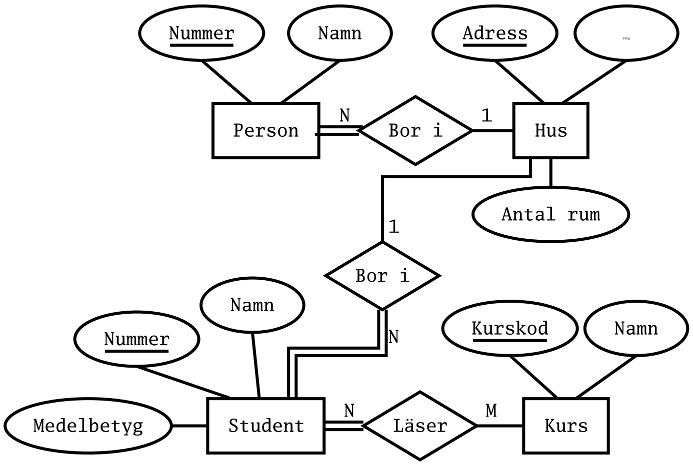
Låt oss gå tillbaka till vårt tidigare exempel med personer som bor i hus:



Vi antar nu att en del av personerna är studenter. Precis som alla andra personer har en student ett namn och ett personnummer, och hon bor i ett hus. Men en student har dessutom ett medelbetyg, och hon läser kurser:



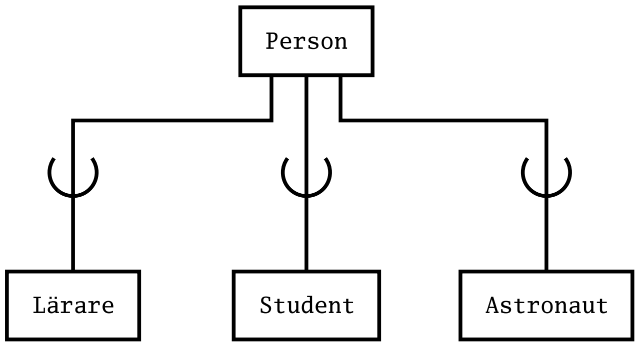
Om vi vill ange samma sak *utan* arv, skulle vi bli tvungna att rita något i den här stilen:



Notera att entitetstypen **Student** har samma attribut och deltar i samma sambandstyper som **Person**, plus egna attribut och sambandstyper. De gemensamma attributen och sambandstyperna skulle vi bli tvungna att rita två gånger. Dessutom har vi tappat bort informationen om att alla studenter är personer.

2.18 Mer om arv i EER-diagram

Om vi går tillbaka till det första exemplet på EER-diagram så visar det ju, som vi kanske minns, att det finns **personer**, och en **person** kan vara **lärare**, **student** eller **astronaut**:



Om personer har någon viss egenskap, till exempel namn, så har även studenter, lärare och astronauter det, för de är ju allihop personer.

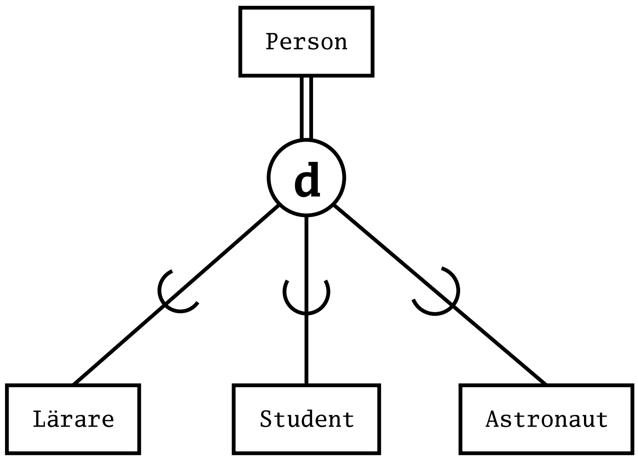
Men det finns några frågor som det här EER-diagrammet inte ger svar på:

- Måste alla personer vara studenter, lärare eller astronauter, eller kan det finnas personer som bara är personer, och inte tillhör någon av de tre subklasserna?

- Kan en person samtidigt vara både till exempel lärare och student, eller kan en person bara tillhöra en enda av subklasserna?

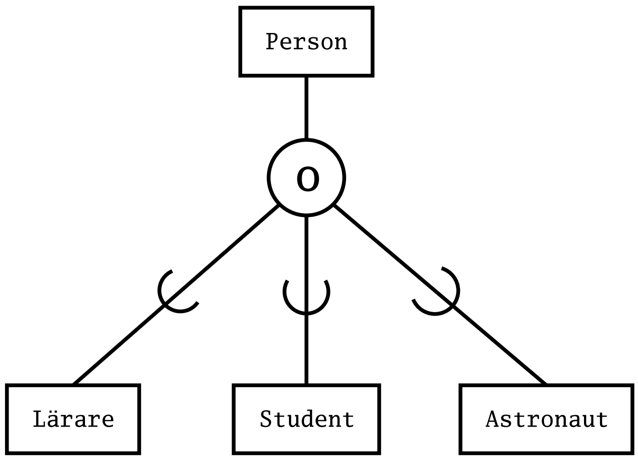
Den första frågan handlar om *fullständig* eller *partiell specialisering*, och den andra frågan handlar om ifall subklasserna är *disjunkta* eller *överlappande*.

Det här går att rita upp i EER-diagrammet. Här har vi lagt till en del information:



d:et står för *disjunkt*, och betyder att subklasserna är disjunkta: en person kan vara *antingen* student, lärare eller astronaut, men hon kan inte vara flera saker på en gång. Dubbelstrecket mellan entitetstypen **Person** och cirkeln med **d**:et betyder fullständig specialisering: varje person måste vara student, lärare eller astronaut, och det får inte finnas personer som inte är någondera. Vi känner igen notationen med dubbelstreck från hur man ritar fullständigt deltagande mellan entitetstyper.

Motsatsen till **d**:et (för disjunkta subklasser) är **o**, som står för *overlapping*, vilket är engelska för *överlappande*. Här är en annan variant på samma EER-diagram:



Det här diagrammet visar, precis som förut, att personer kan vara studenter, lärare och astronauter. Men det visar också att det kan finnas personer som inte tillhör någon av de tre subklasserna, utan som bara är personer. Det visar dessutom att det kan finnas personer som tillhör mer än en subklass, och till exempel är både lärare och studenter samtidigt.

2.19 Översättning till relationsmodellen

Om man ritat ett ER-diagram, och vill lagra sin databas i en databashanterare, måste man normalt översätta ER-diagrammet till en datamodell som databashanteraren förstår. Det kan vara till exempel en objektorienterad databashanterare, men det allra vanligaste är att man vill arbeta med en relationsdatabashanterare. ER-diagrammet ska alltså översättas till tabeller. Läs mer i kapitel 6, *Översättning från ER-modellen till relationsmodellen*.

2.20 Varför inte ER-diagram hela vägen?

I avsnitt 2.2 skrev vi att ER-diagram ger en fördel över relationsmodellens tabeller genom att vara både enklare och överskådligare.

Därför är det för det mesta bättre att börja med att rita ett ER-diagram, som man sen översätter till tabeller, än att konstruera tabeller direkt. Men om nu ER-diagram har en fördel över tabeller, varför då alls blanda in relationsdatabaser? Det finns inget som säger att en databashanterare måste använda sig av relationsmodellen, så varför inte använda ER-diagram även i databashanteraren?

Det finns objektorienterade och objektrelationella databaser som använder en datamodell som är ganska lik ER-modellen. Det finns också åtminstone någon enstaka databashanterare där man kan mata in ER-diagram direkt. Men de allra flesta databashanterare kräver att man översätter sitt ER-diagram till något annat format. För det mesta använder man en relationsdatabashanterare, vilket betyder att data lagras i form av tabeller.

Orsakerna till det är dels tradition, och dels att man har lång erfarenhet av hur man bygger relationsdatabashanterare så att de blir snabba, kraftfulla och lättanvända.

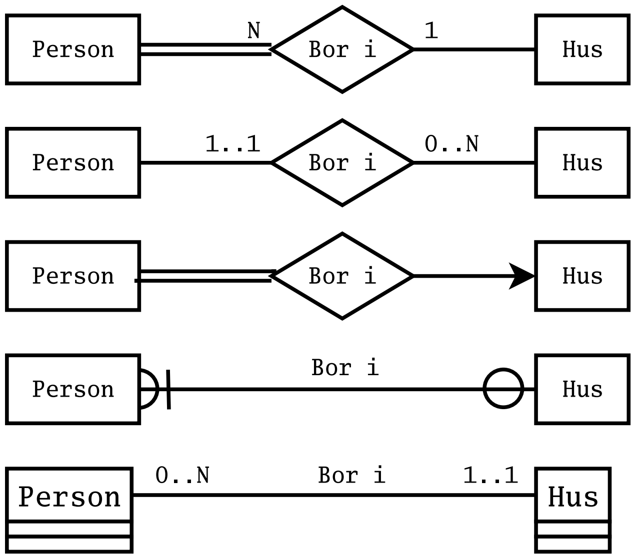
Det finns också en del val som man kan göra när man översätter från ER-diagrammet till hur saker ska lagras internt i databashanteraren, som det kanske är bättre att låta en människa göra, än att databashanteraren väljer en standardlösning. En standardlösning skulle kunna leda till att en del databaser blir onödigt stora, eller att en del sökningar går onödigt långsamt.

Ytterligare ett skäl är att det finns bra och etablerade frågespråk (främst SQL) för relationsdatabaser, och man har ännu inte kommit lika långt med hur man gör sökningar i grafiska beskrivningar som ER-diagram.

2.21 Alternativa notationer

Som vi nämnde ovan kan ER-diagram ritas på flera olika sätt, och det gäller särskilt hur man anger kardinalitetstförhållanden (1:N, N:M osv).

Här är några olika sätt att rita samma 1:N-samband mellan personer och hus:



Det första alternativet är vårt vanliga skrivsätt. I det andra anger man hur många sambandsinstanser varje entitetsinstans kan delta i: en person kan delta i exakt ett **Bor i**-samband, och ett hus kan delta i noll till hur många som helst. (Det blir alltså lite bakvänt jämfört med det första.) Pilen i det tredje alternativet liksom pekar ut det enda hus som en person bor i. Det fjärde alternativet är vanligt i Sverige och kallas för en "infologisk modell". "Gaffeln" ska illustrera att ett hus hör ihop med flera personer.

Det sista exemplet är ritat med UML, som egentligen inte är ett sätt att rita ER-diagram, utan ett objektorienterat modelleringsspråk. UML är ganska likt ER-modellen, men i UML kan man också modellera beteende, inte bara data.

2.22 De viktigaste begreppen

Som en repetition och sammanfattning går vi här igenom de viktigaste begreppen från det här avsnittet.

Entity-Relationship-modellen, **ER-modellen**, **entitets-sambands-modellen** (engelska: **the Entity-Relationship model**, **the ER model**, **the E/R model**). En konceptuell datamodell där man beskriver verkligheten genom att ange de typer av entiteter (engelska: **entities**), alltså ungefär "saker", som finns i den verkligheten, och vilka typer av samband (engelska: **relationships**) som finns mellan dessa. Beskrivningen ritas oftast upp grafiskt i form av ett ER-diagram.

ER-schema (engelska: **ER schema**). En beskrivning av verkligheten gjord enligt ER-modellen. Som vanlig utgör schemat en beskrivning av vilka data som kan finnas i databasen, oberoende av vilka data som råkar finnas där just nu.

ER-diagram (engelska: **ER diagram**). Ett ER-schema uppritat som ett diagram.

Entitetstyp (engelska: **entity type**). I ER-modellen beskriver man världen med hjälp av två grundläggande byggblock: typer av saker, eller entiteter, och typer av samband mellan dessa saker.

Sambandstyp (engelska: **relationship type**). I ER-modellen beskriver man världen med hjälp av två grundläggande byggblock: typer av saker, eller entiteter, och typer av samband mellan dessa saker. Undvik att översätta "relationship" med "relation" eller "relationstyp", för en relation är en tabell i relationsmodellen.

Instans eller **förekomst** (engelska: **instance**). En viss sak som tillhör en entitetstyp eller sambandstyp. Till exempel kan personen Kalle vara en instans av entitetstypen Person.

Kardinalitetsförhållande (engelska: **cardinality ratio**). Hur många saker som kan delta i en och samma instans av en sambandstyp: ett-till-ett-samband, ett-till-många-samband och många-till-många-samband.

Fullständigt deltagande (engelska: **total participation**). Kravet att varje förekomst av en entitetstyp måste delta i en viss sambandstyp.

Attribut (engelska: **attribute**). Betyder "egenskap", och används om egenskaperna hos entiteterna i ER-modellen.

Svag entitetstyp (engelska: **weak entity type**). En entitetstyp i ER-modellen där instanserna av den entitetstypen inte har en egen nyckel, och därför inte går att unikt identifiera utan hjälp av en annan entitetstyp.

Den utvidgade ER-modellen, **EER-modellen** (engelska: **the Enhanced Entity-Relationship Model**, **the EER model**). En variant av ER-modellen där man lagt till arv, på samma sätt som i objektorienterad datamodellering. Beskrivningen ritas oftast upp som ett **EER-diagram**.

Fullständig specialisering (engelska: **total specialization**). Kravet att varje instans av basklassen måste tillhöra (minst) en av subklasserna.

Partiell specialisering (engelska: **partial specialization**). Egenskapen att varje instans av basklassen inte nödvändigtvis måste tillhöra någon av subklasserna.

Disjunkta subklasser (engelska: **disjoint subclasses**). Egenskapen att varje förekomst av en entitetstyp bara kan tillhöra (högst) en av den entitetstypens subklasser.

Överlappande subklasser (engelska: **overlapping subclasses**). Egenskapen att varje förekomst av en entitetstyp kan tillhöra mer än en av den entitetstypens subklasser.

UML. Av **Unified Modeling Language**. Ett grafiskt modelleringsspråk som påminner om ER-modellen, men som är mer omfatttande och mer standardiserat. Läs mer på **http://www.uml.org/**.

2.23 Övningar

Det finns svar till övningarna, och en del diskussioner kring svaren, på bokens webbplats, men försök lösa uppgifterna själv innan du tittar i facit.

Övning 1: Jultomtens databas

Varje jul ska jultomten distribuera flera miljarder julklappar till flera hundra miljoner barn. För att allt ska bli rätt måste jultomten och hans nissar hålla reda på stora mängder information. De måste till exempel hålla reda på vilka barn finns och var de bor, om de varit snälla eller elaka, vilka presenter de önskat sig, osv. Tidigare har det gått bra med papper och pärmar, men förra året blev mängden papper så stor att hela arkivavdelningen sjönk genom isen vid Nordpolen, där tomten ju bor. Därför har jultomten bestämt sig för att datorisera julklappsadministrationen med hjälp av en databas.

Jultomten berättar att han vill ha med följande saker i databasen:

- Alla barn i hela världen, med namn och adress.

- Länder, som barnen bor i. Länder har unika namn.
- Presenter, alltså de leksaker som barnen önskar sig och som de får om de varit snälla.
- Önskelistor, som anger vilka presenter som varje barn önskar sig. Varje barn kan önska sig hur många presenter som helst, och man måste kunna hantera att det finns en inbördes ordning mellan de önskningarna.
- Vilka presenter på önskelistan som varje barn sen faktiskt får.

Rita ett ER-diagram för databasen. (Använd grunderna. Det behövs varken arv eller svaga entitetstyper.)

Övning 2: I jultomtens hemliga tjänst

Jultomten är nöjd med det ER-diagram du ritade i övning 1, och nu har han kommit på ytterligare några saker som behöver vara med i databasen. Rita därför ett nytt diagram genom att utöka det gamla ER-diagrammet. Den här gången behöver du förmodligen använda arv, och alltså ska du göra ett EER-diagram. Följande ska vara med:

- *Tomtenissar*. Jultomten sköter ju inte hela verksamheten själv, utan har hjälp av sina nissar.
- Alla tomtenissar är inte lika, utan det finns tre olika sorter. *Tillverkningstomtar* arbetar i leksaksfabrikerna och tillverkar leksaker. *Underrättelsetomtar* arbetar i TUT, Tomtens underrättelsetjänst, och samlar information om barnen så jultomten vet om de varit snälla. Varje underrättelsetomte har en säkerhetsklass. *Chefstomtar* är chefer över de andra nissarna. Varje tomtenisse, inklusive chefstomtarna, har en chef, så det bildas en hierarki med flera nivåer. Bara chefstomtar kan vara chefer.
- *Rapporter*. Underrättelsetomtarna skriver rapporter om barnen. Varje rapport handlar om ett visst barn, den är skriven av en viss underrättelsetomte, och förutom själva rapporttexten innehåller den ett datum och ett positivt eller negativt tal som anger hur snällt barnet varit. Varje rapport ska också granskas av en underrättelsetomte, som inte får vara samma som skrev den rapporten. Granskaren ska också ha högre säkerhetsklass än författaren.
- *Ansvar*. Varje present har en ansvarig tillverkningstomte. Varje barn har en ansvarig underrättelsetomte. Det finns också

en ansvarig underrättelsetomte för varje land, och den tomten måste ha högre säkerhetsklass än alla de underrättelsetomtar som ansvarar för barnen i det landet.

Det är inte säkert att allt som tomten vill ha med går att representera i EER-diagrammet. Notera i så fall dessa separat, så de kraven inte glöms bort.

2.24 Litteratur

De angivna kapitlen i böckerna motsvarar det här kapitlet.

- Connolly et al, fjärde upplagan: kapitel 11-12. Till skillnad från de andra vanliga databasböckerna har den här boken gått över till UML-notation för att rita ER-diagram.
- Date, åttonde upplagan: kapitel 14
- Elmasri/Navathe, fjärde upplagan: kapitel 3, 4 och 7
- Silberschatz et al, fjärde upplagan: kapitel 2