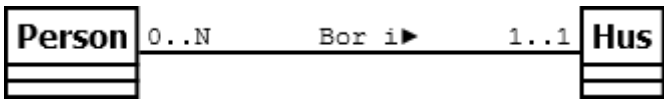
Konceptuell datamodellering

I lektion 2 tog vi en närmare titt på relationsmodellen. Denna datamodell beskriver data i databasen som en uppsättning relationer. Sambanden mellan raderna i relationerna baseras på referensintegritet (främmande nycklar). Så länge som vi skapar små databaser med några få relationer i, behöver vi inte några andra modelleringstekniker för att utveckla en databas. I större projekt, som omfattar ett stort antal relationer, kommer det dock att vara svårt att skapa en fullständig databas bara med hjälp av tillvägagångssättet relationsmodellen ger oss. Därför finns det så kallade konceptuella datamodeller vi kan använda oss av i stället. Det är mycket lättare att använda dessa konceptuella datamodeller för att representera mycket komplex verkligheter vi önskar att lagra data om.

De mest populära konceptuella modelleringstekniken är baserad på Entity-Relationship-modellen (ER-modellen). I figur 1 nedan kan du se några av de symboler som används för att visa relationer och de samband som finns mellan dem. Detta är en konceptuell hög-nivå-datamodelleringsteknik avsedd för att designa databaser. Den har fått en uppsättning begrepp och objekt som beskriver struktur och data i den databas som ska skapas. Det används också för att göra det lättare för användaren att tolka data som ska lagras.

Figur 1. Exempel på ett Entity Relational-diagram ritad med Chen-notationen.

Alla de konceptuella datamodeller har någon form av schematiskt sätt att visa förhållandet mellan så kallade entitetstyper (”saker” som ska lagras i databasen), attribut och eventuella samband mellan entitetstyperna. I ER-modellen kallas ofta dessa figurer för ER-diagram och består av ett antal specifika symboler. Det finns flera, mer eller mindre lika, notationer för hur figurerna i diagrammet ska ritas. Den alternativa läroboken använder UML som notation, se exempel i figur 2, medan kursboken använder Chen-notationen. Vi kommer i lektionsmaterialet att fortsättningsvis använda UML som notation.



Figur 2. Samma exempel som i figur 1, men ritad med UML-notationen.

Det är viktigt att känna till att konceptuella datamodeller är helt oberoende av vilken DBMS vi väljer att implementerar databasen i och vilken plattform samt hårdvara som väljs för att köra DBMS på. Det spelar ingen roll om databasen kommer att implementeras på en PC, arbetsstation eller UNIX-dator. Teoretiskt är detta mycket viktigt, eftersom vi väljer arkitekturen, RDBMS och OS baserat på storleken på databasen, användning och komplexitet. Men tyvärr verkar det som många företag i praktiken väljer först den fysiska miljön som plattform och verktyg, innan de lär sig att använda den och för vilket ändamål. Detta skulle kunna ge mycket olyckliga konsekvenser. Vi skulle kunna implementera en överdimensionerad databashanterare i förhållande till de behov som behövs, vilket är dåligt i ekonomiska hänseenden, eller att vi väljer att använda ett för enkelt DBMS-system som MS-Access när Oracle bör väljas.

Datamodellering ger självklart inte ett fullständigt svar på alla sådana faktorer, men det visar åtminstone komplexiteten i databasstrukturen. En datamodell kan också säga något om storleken av databasen: hur många entitetstyper som måste användas och hur många entiteter av varje typ ska lagras?

Inte allt avspeglas i en ER-modell. Saker som säkerhetskrav och responstid är exempel på detta.

ER-modellen är nära knuten till relationsmodellen, eftersom de båda modellerna använder vissa begrepp som är lika. En ER-modell skulle dock även kunna användas för att konstruera andra typer av databaser, till exempel en objektorienterad databas.

Varför ER-diagram är viktiga

Många har ofta svårt med att förstå varför det är så viktigt att konstruera ER-diagram. Vissa föredrar i stället att utforma tabellerna direkt i den databashanterare som ska användas. Anledningen till detta kan vara bristen på förståelse av följande två punkter:

1. Verkligheten som ska avbildas saknar komplexitet.
2. Många tror att ER-modellering är alltför abstrakt. De kan inte föreställa sig verkligheten i abstrakta termer. De föredrar tydliga uppgifter där lösningen redan är synliga.

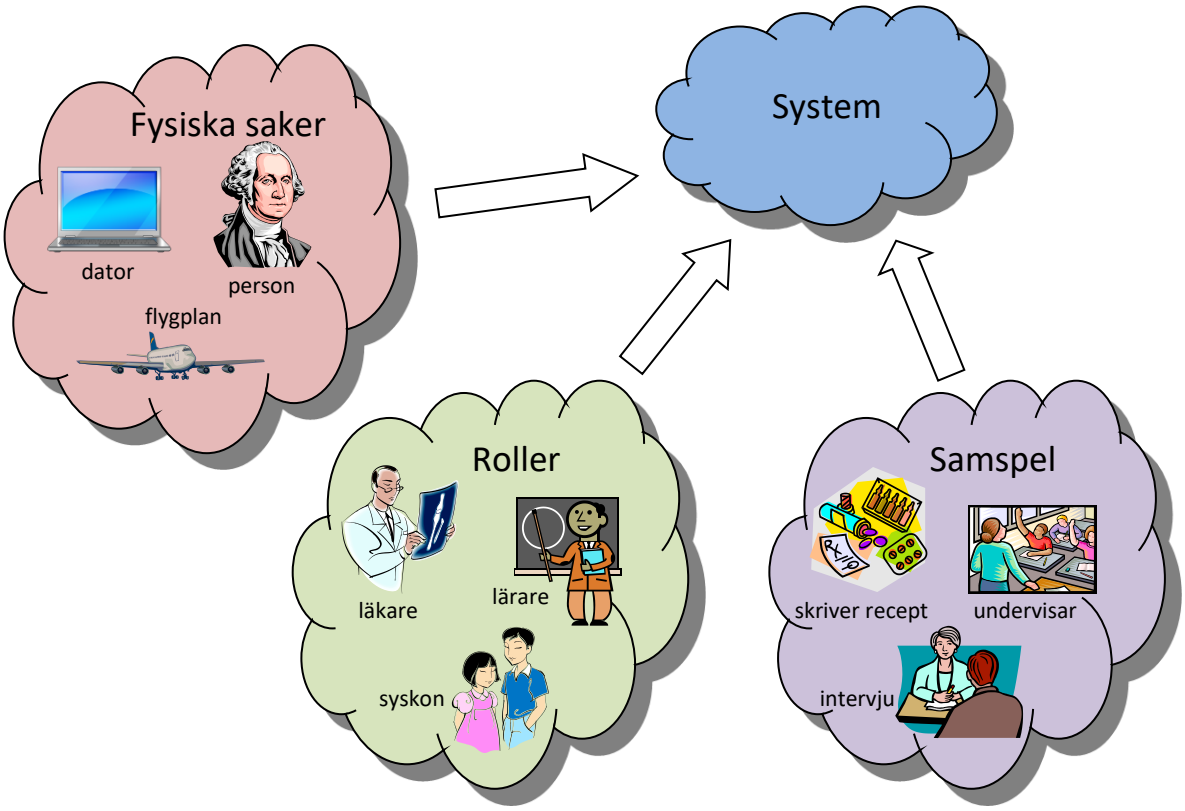
En förutsättning för att skapa bra ER-diagram är att utförligt känna till den verklighet som ska modelleras. Då kommer skapandet av ER-diagrammet att kännas mindre abstrakt och förvirrande.

Den kanske största fördelen med ett ER-diagram får vi om verkligheten som modelleras är så komplex att den är svår att förklara för någon utomstående hur den fungerar. Då kan ett ER-diagram användas vid kommunikationen mellan uppdragsgivaren och systemutvecklarna. Därför är det en fördel att känna till datamodellering även om man du aldrig kommer att utveckla ett databassystem själv.

Poängen med att skapa ER-diagram är inte att göra utvecklingen av databaser mer komplex. Snarare tvärtom. Genom att lära dig ER-modellering kommer du att få det lättare att utveckla en bra databaser med databasstrukturen i åtanke.

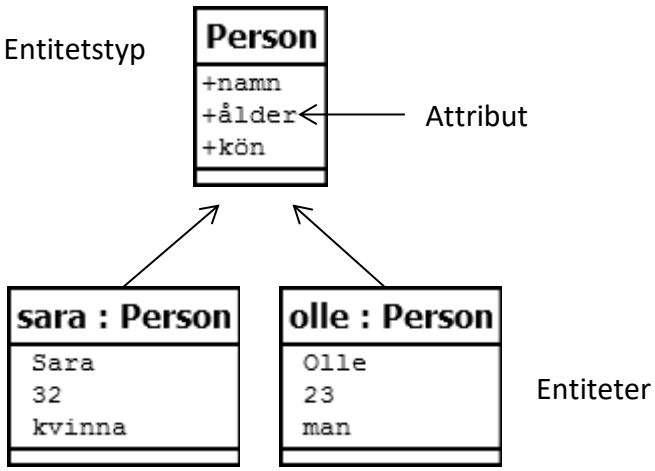
Entitetstyper och entiteter

En entitetstyp är en abstraktion, ett objekt, ett koncept eller en sak. Den har en klar avgränsning och måste finnas i problemdomänen. Entitetstypen kommer av användaren att uppfattas att ha en självständig existens. Ofta representerar entitetstypen ett substantiv i en textuell beskrivning av den verklighet som ska modelleras. En entitetstyp samspelar även med andra entitetstyper. Entitetstypen kan ha många olika former så som visas i figur 3.



Figur 3. En entitetstyp kan anta många olika former.

Entitetstypen representerar en uppsättning entiteter eller föremål från den verkliga världen där alla dessa har samma attribut (egenskaper). De enskilda entiteterna definieras genom sina entitetstyper. De känner till sin entitetstyp. Figur 4 visar dessa begrepp.



Figur 4. Samband mellan entitetstyp och entiteter.

I en ER-modell är det de entitetstyper som beskrivs, inte entiteter. En entitet är en förekomst som kan identifieras unikt bland alla andra entiteter. Det är en instans av en entitetstyp. Vi kan tänka oss att alla entiteter som har attribut som är lika, är av samma entitetstyp. Till exempel kan alla anställda i ett företag vara av samma entitetstyp.

Det är möjligt att gruppera entiteter i en klass. Olika personer har olika roller i verkligheten och tillhör därför olika klasser: anställd, sjuksköterska, mekaniker. Dessa är subklasser och superklassen Person. Detta är en utökning av ER-modellen som är väldigt lik arv i objektorienterade modeller. Vi återkommer till detta i nästa lektion.

När vi ritar ER-diagram är det entitetstyperna vi ritar och inte de olika entiteterna. Symbolen för en entitet är en rektangel med namnet på entitetstypen.

Svaga och starka entitetstyper

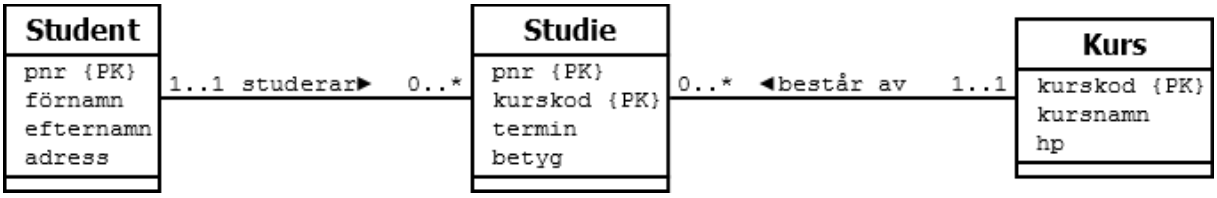
En stark entitetstyp har en självständig existens och är inte beroende av några andra entitetstyper. Den har en egen unik primärnyckel, som kallas PK. Entitetstypen bok i figur 5 är en stark entitetstyp.

Figur 5. Entitetstypen Bok.

Denna entitetstyp översätt till relationsmodellen i ett relationsschema så här (primärnyckeln stryks under):

```
Bok (isbn, titel, förlag, pris)
```

En svag entitetstyp är beroende på andra entitetstyper för dess identitet. Den är vad vi kallar identitetsberoende av andra entitetstyper. I figur 6 har vi fått entitetstypen Studie, vilken är en svag entitetstyp.



Figur 6. ER-diagram över studenter och de kurser som läses.

Detta ER-diagram översätts till relationsmodellen på följande sätt (främmandenycklar markeras med *):

```
Student (pnr, förnamn, efternamn, adress)
Studie (pnr*, kurskod*, termin, betyg)
Kurs (kurskod, kursnamn, hp)
```

Entitetstypen Studie är svag eftersom den sammansatta primärnyckeln pnr och kurskod båda är primärnycklar i andra entitetstyper. Studie har inte någon egen unik primärnyckel. Båda attributen i primärnyckeln är även främmandenycklar. Varken UML-notationen eller

Chen-notationen har stöd för främmandenycklar. Därför markerar vi i ER-diagramet inte dessa attribut med någon * utan det görs enbart i relationsschemat.

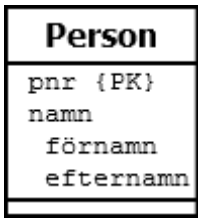
Attribut

En entitetstyp kännetecknas av att den består av ett antal attribut. Attributen säger något om entitetstypens egenskaper i verkligheten. Det är vanligt att vi tar med alla attribut i ER-diagramet. Ett attribut har en viss domän (vilken typ och uppsättning av värden) som kan lagras i attributet. Detta är dock inget vi normalt tar ställning till i ER-modellen utan görs i ett senare skede.

Attributen i entitetstypen Bok (i figur 5) är så kallade enkla attribut. Detta är attribut som håller ett atomiskt värde som inte kan delas upp ytterligare i andra värden.

Sammansatta attribut

Ett sammansatt attribut är ett attribut som består av flera komponenter, var och en med en självständig existens. Attributet namn i entitetstypen Person (figur 7) är ett sammansatt attribut som består av de enkla attributen förnamn och efternamn.



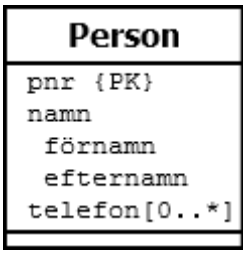
Figur 7. Sammansatt attribut.

När vi översätter ett sammansatt attribut till relationsmodellen tar vi endast med de enkla attributen som det sammansatta attributet består av:

```
Person (pnr, förnamn, efternamn)
```

Flervärda attribut

Ett flervärt attribut är ett attribut som innehåller flera värden för en enda entitet. En person kan till exempel ha mer än ett telefonnummer. Följaktligen kan det finnas flera värden som hör till attributet telefon för varje person. I figur 8 har vi använt de lämpliga symbolerna för att beskriva det flervärda attributet telefon. Inom hakparenteserna anger vi först det minst antalet följt av det högsta antalet (* innebär obegränsat många).



Figur 8. Flervärt attribut.

Flervärda attribut ger oss problem när vi översätter ER-diagramet till relationsmodellen. Vi kan inte ge attributet telefon fler värden för en rad i tabellen, eftersom en cell i tabellen bara kan ha atomära värden. Om vi gör en ny rad i tabellen för samma entitet och med samma unika värde för primärnyckel, kommer vi att bryta mot regeln om entitetsintegritet. Ett möjligt sätt att hantera detta problem är att i stället införa en ny kolumn för att lagra ytterligare ett telefonnummer:

```
Person (pnr, förnamn, efternamn, telefon1, telefon2)
```

Detta är som du säkert listat ut ingen optimal lösning. För det första är det en väldigt statisk lösning. Vad händer om en person helt plötsligt har tre telefonnummer? Då måste vi in i databasens schema och lägga till ytterligare en kolumn för ett telefonnummer. Vad händer då med personer som inte har eller bara har ett telefonnummer? För dessa personer måste vi registrera null-värden i de celler som saknar telefonnummer. Null-värden är något vi ska försöka undvika att få i en databas när vi designar den.

Som tur är finns det en bättre lösning på detta problem:

```
Person (pnr, förnamn, efternamn)
Telefon (telnr, pnr*)
```

Vi skapar en ny tabell för det flervärda attributet, där primärnyckeln i tabellen Person blir en främmande nyckel i den nya tabellen. På så sätt har vi nu knutit samman de båda tabellerna och vi får en mycket dynamisk lösning. Nu kommer bara personer med telefoner att registreras i tabellen Telefon. Det spelar ingen roll hur få eller många telefonernummer de har. Ingen person utan telefon skulle lagras och på så sätt undviker vi helt NULL-värden i databasen.

Naturligtvis är detta sätt att lösa problemet med flervärda attribut under förutsättning att endast ett telefonnummer kan tillhöra en person. Om fler än en person kan ha samma telefonnummer måste även pnr vara en del av primärnyckeln. Det vill säga vi behör då i Telefon en sammansatt nyckel som består av attributen telnr och pnr.

Härledda attribut

Ett härlett attribut är ett attribut som representerar ett värde som kan härledas från värdet av ett eller flera andra attribut. Dessa attribut behöver nödvändigtvis inte återfinnas i samma entitetstyp. Ett härlett attribut lagras inte fysiskt i databasen utan fås vanligtvis fram genom en SQL-fråga.

Figur 9. Härlett attribut.

I figur 11 ser vi det härledda attributet yta. Ytan kan fås fram genom attributen bredd och längd. När vi översätter ER-diagrammet till relationsmodellen får vi:

```
Rum (rumsnr, längd, bredd)
```

Det härledda attributet tas inte med i relationsmodellen. Med följande SQL-fråga kan vi dock få fram värdet när vi så behöver det:

```
SELECT rumsnr, längd, bredd, längd * bedd AS yta
FROM rum;
```

En kolumn för det härledda attributet upprättas enbart i resultatet av frågan varje gång frågan körs.

Nycklar i ER-modellen

I ER-modellen används tre nyckelbegrepp: kandidatnyckel, primärnyckel och sammansatt nyckel. En kandidatnyckel är ett attribut eller en uppsättning attribut, som unikt identifierar en entitet, som tillhör en entitetstyp. En entitetstyp kan ha fler än en möjlig kandidatnyckel. Av kandidatnycklarna väljs en till att vara primärnyckel. En sammansatt nyckel är en kandidatnyckel som består av två eller fler attribut. De bildar tillsammans en unik kombination av värden som möjligt identifiering av enskilda entiteter. Som vi har nämnt tidigare existerar inte begreppet främmande nyckel i ER-modellen.

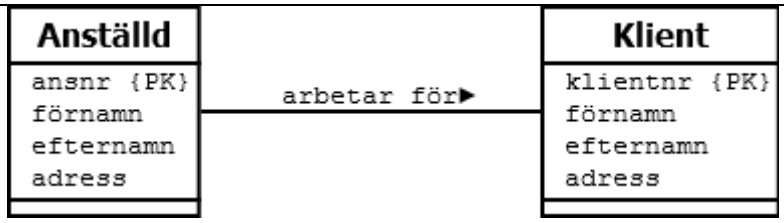
I relationsmodellen är främmande nycklar däremot nödvändiga för de upprätthåller referensintegriteten i databasen. De ”limmar” samma de olika tabellerna med varandra.

Samband mellan entitetstyper

I ER-modellen finns det mekanismer som beskriver sambandstyper mellan olika entitetstyper. En sambandstyp beskriver sambandet mellan entiteter från olika entitetstyper.

Sambandstyper

I ER-modellen är en sambandstyp ett meningsfullt samband mellan två eller flera entitetstyper. Varje sambandstyp har ett namn som beskriver sambandet. Sambandstyper har vi redan stött på när vi i lektion 2 pratade om relationsmodellen. Skillnaden mellan dessa två modeller är att det i ett ER-diagram räcker med att rita ut att det finns ett samband mellan entiteter som till exempel tillhör två entitetstryper utan behöva tänka på främmande nycklar, vilket vi måste göra när vi arbetar med relationsmodellen. Ett exempel visas i figur 10.

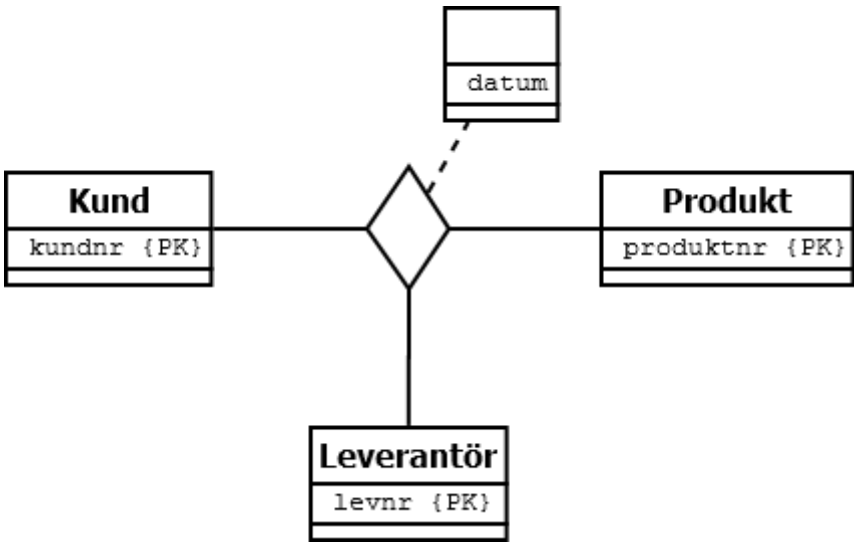


Figur 10. Exempel på en sambandstyp.

Symbolen för en sambandstyp är en linje mellan de entitetstyper som ingår i sambandet. Sambandstypens namn ska beskriva sambandet på ett tydligt sätt, gärna med ett verb: arbeta för, delta i, ingå i, ha, är etc. Vi avläser sambandet gärna från vänster till höger så som i figur 10, men tvärtom går också lika bra.

Flervägssamband

Det finns olika typer av sambandstyper. De vanligast är tvåvägssamband, trevägssamband och rekursiva samband. Tvåvägssambanden är absolut vanligast att använda och är vad vi hittills sett exempel på i de olika figurerna. Denna sambandstyp involverar två olika entitetstyper. Ett trevägssamband är ett samband mellan tre olika entitetstyper. Exempel på detta visas i figur 11.



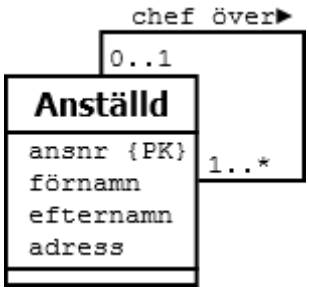
Figur 11. Exempel på ett trevägssamband.

I ER-modellen används en ”diamant” som symbol för att rita ett trevägssamband. Behöver trevägssamband själv ha några attribut kopplar vi på det till diamanten som en ny entitetstyp utan namn. Översatt till relationsmodellen får vi:

```
Kund (kundnr)
Produkt (produktnr)
Leverantör (levnr)
Leverans (kundnr*, produktnr*, levnr*, datum)
```

Alla ingående entitetstypers primärnycklar blir en sammansatt nyckel i en ny tabell. Varje attribut i den sammansatta nyckeln blir även främmande nycklar till sina respektive tabeller.

I ett rekursivt samband ingår samma entitetstyp mer än en gång med olika roller. Låt oss som exempel utgå från entitetstypen Anställd. Några anställda är chefer medan andra anställda är anställda av en chef. Detta visas i figur 12.



Figur 12. Exempel på ett rekursivt samband.

Figuren ovan säger oss att en anställd som är chef är chef över en eller flera andra anställda. En anställd kan ha ingen eller en chef (högsta chefen har ingen chef över sig). När vi översätter ER-diagrammet i figur 12 till relationsmodellen får vi följande tabeller:

```
Anställd (ansnr, förnamn, efternamn, adress)
Chef_över (anställd_ansnr*, chef_ansnr*)
```

I den nya tabellen Chef_över kommer primärnyckeln i Anställd bli två främmande nycklar. Den ena kommer referera till chefen och den andra till den anställde som chefen chefar över. Båda attributen ges olika namn eftersom två kolumner i en tabell inte kan ha samma namn. Att attributen ges nya namn har ingen egentlig betydelse så längs som de båda refererar till ansnr i Anställd. Hur detta implementeras i SQL visas nedan. Primärnyckel i den nya tabellen blir anställd_ansnr (eftersom en anställd max har en chef kommer vi inte få upprepning i detta attribut).

```
CREATE TABLE anställd (
  ansnr CHAR(4),
  förnamn VARCHAR(40),
  efternamn VARCHAR(35),
  adress VARCHAR(30),
CONSTRAINT anställd_pk PRIMARY KEY(ansnr));

CREATE TABLE chef_över (
  anställd_ansnr CHAR(4),
  chef_ansnr CHAR(4),
CONSTRAINT chef_över_pk PRIMARY KEY(anställd_ansnr),
CONSTRAINT chef_över_fk1 FOREIGN KEY(chef_ansnr) REFERENCES anställd (ansnr),
CONSTRAINT chef_över_fk2 FOREIGN KEY(anställd_ansnr) REFERENCES anställd(ansnr));
```

Kardinalitetsförhållande

Kardinalitetsförhållandet bestämmer antalet möjliga samband mellan entiteter i ett tvåvägssamband. Ett samband har alltid en kardinalitet. Det finns tre typer av kardinalitet (vanligtvis säger vi att det finns tre typer av samband mellan entitetstyper):

- En-till-en (1:1)

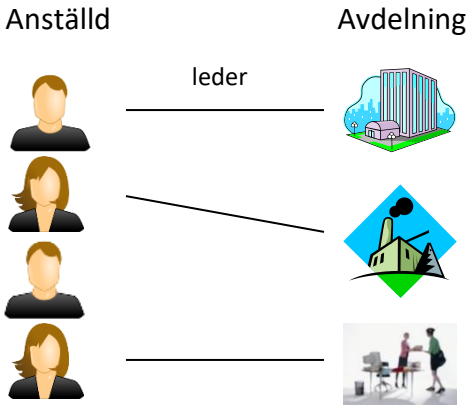
- En-till-många (1:M)
- Många-till-många (N:M)

Bokstäverna N och M innebär ett godtyckligt antal möjliga samband en entitet är inblandad i. Notera att den högra delen alltid beskriver det maximala antalet möjliga samband en entitet kan ha samband med i en annan entitetstyp.

Den vänstra delen berättar huruvida en entitet måste delta i ett samband eller inte. Om vänstra delen innehåller siffran 1 är deltagandet fullständigt. Det innebär att för varje entitet i en entitetstyp måste det finnas en annan entitet i den andra entitetstypen. Om den vänstra siffran är 0 har vi ett partiellt deltagande. Det innebär at en entitet i en entitetstyp inte nödvändigtvis måste ha ett samband till en entitet i den andra entitetstypen.

Ett-till-ett-samband

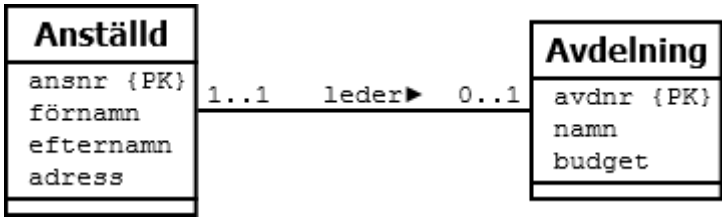
Figur 13 visar hur sambanden mellan entiteter i en sambandstyp av typen ett-till-ett kan se ut.



Figur 13. Ett-till-ett-samband.

Figuren ovan visar sambandstypen *leder* som beskriver sambandet mellan entiteterna i entitetstyperna Anställd och Avdelning. Den visar vilka anställda som är ledare för vilka avdelningar. Följande verklighet gäller: En person kan vara ledare för en avdelning, medan en avdelning måste ha en och endast en ledare. Det kan finnas anställda som inte leder någon avdelning.

ER-diagrammet i figur 14 visar hur översättningen av verkligheten, som beskrivs av figur 13, ser ut.



Figur 14. Anställd leder avdelning.

Vi har ett ett-till-ett-samband som säger oss att en anställd leder inget eller maximalt en avdelning. Här har vi ett partiellt deltagande. Vidare leds en avdelning av en och endast en anställd, vilket är ett fullständigt deltagande. När vi översätter ER-diagrammet i figur 14 till relationsmodellen får vi följande tabeller:

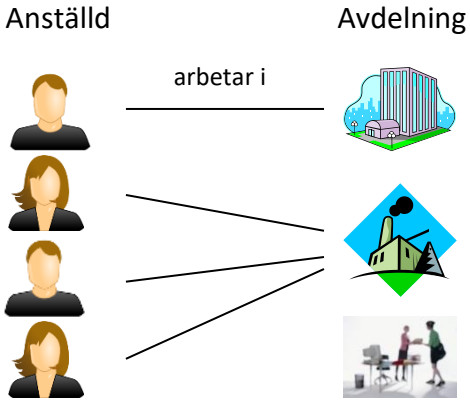
```
Anställd (ansnr, förnamn, efternamn, adress)
Avdelning (avdnr, namn, budget, ansnr*)
```

Vi tar primärnyckeln från Anställd och sätter som främmande nyckel i Avdelning. Vid ett-till-ett-samband kan vi även sätta främmande nyckeln tvärtom, men om ena sidan har ett fullständigt deltagande är det bättre att sätta främmande nyckeln där. Eftersom en avdelning alltid har en anställd som leder avdelningen (fullständigt deltagande) sätter vi främmande nyckeln i tabell Avdelning. Gör vi tvärtom, sätter avdnr som främmande nyckel i Anställd, riskerar vi att få null-värden i främmande nyckeln för de anställda som inte leder någon avdelning.

Sambandstypen ett-till-ett är inte så vanlig. Helst ska vi undvika att få fullständigt deltagande på båda sidorna (1..1 vid båda entitetstyperna). I dessa fall bör vi i stället skapa en ny uppsättning attribut i en av de två ingående entitetstyperna som innehåller samma data. Överfört till vårt exempel: om alla anställda leder en avdelning och alla avdelningar har en ledare, då bör attributen i Avdelning sättas som attribut i Anställd (eller tvärtom). Varför vi använder två entitetstyper i vårt exempel är som sagt för att undvika NULL-värden då ena sidan har ett partiellt deltagande. Det kan finnas vissa anställda som inte är leder någon avdelning.

Ett-till-många-samband

Figur 15 visar hur sambanden mellan entiteter i en sambandstyp av typen ett-till-många kan se ut.

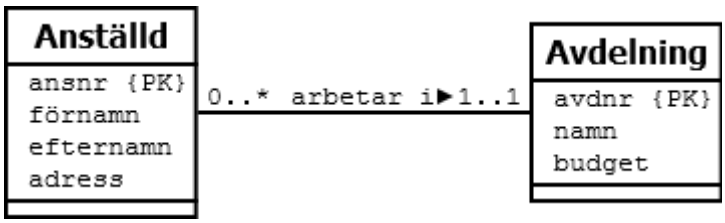


Figur 15. Ett-till-många-samband.

Figuren ovan visar sambandstypen *arbetar i* som beskriver sambandet mellan entiteterna i entitetstyperna Anställd och Avdelning. Den visar vilka anställda som är arbetar i vilka

avdelningar. Närmare bestämt kan en anställd arbeta i maximalt en avdelning och en avdelning kan maximalt ha många anställda som jobbar där. Vidare måste en anställd jobba i en avdelning, men det kan finnas avdelningar utan anställda.

I figur 16 har vi översatt denna verklighet till ett ER-diagram.



Figur 16. Anställd arbetar i en avdelning.

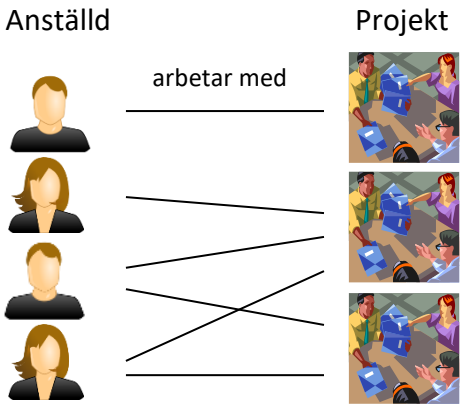
Detta är den vanligaste sambandstypen i ett ER-diagram. Det är också det enklaste att översätta till relationsmodellen. Den entitetstyp med * (eller en siffra större än 1) får alltid främmande nyckeln. Primärnyckeln i Avdelning kommer med andra ord att bli en främmande nyckel i Anställd. Varför inte tvärtom? Prova det själv med några testdata. Om du tar ansnr som främmande nyckel i Avdelning, kommer du att komma i konflikt med regeln för entitetstintegritet. För varje avdelning måste du då ange många anställda, och det kommer inte att fungera eftersom det inte går att ange mer än ett värde i ett attribut.

När vi översätter ER-diagrammet i figur 16 till relationsmodellen får vi följande tabeller:

```
Anställd (ansnr, förnamn, efternamn, adress, avdnr*)
Avdelning (avdnr, namn, budget)
```

Många-till-många-samband

Figur 17 illustrerar ett samband av typen många-till-många.

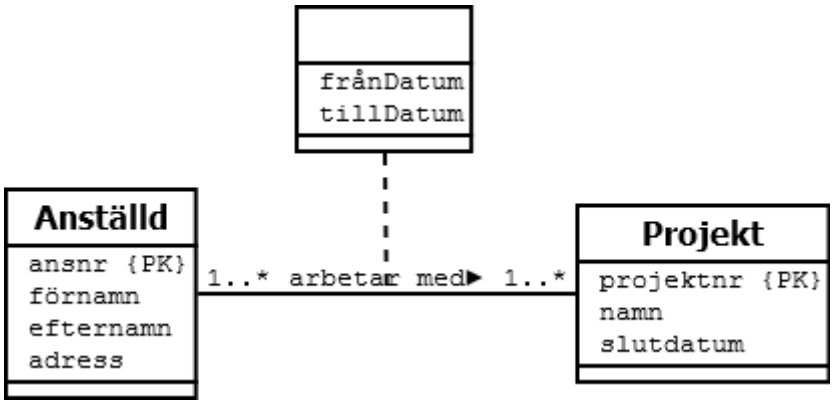


Figur 17. Många-till-många-samband.

I figur 17 ser vi sambandstypen *arbetar med* som beskriver sambandet mellan entiteterna i entitetstyperna Anställd och Projekt. Dessa samband visar vilka anställda som arbetar med vilka projekt. Alltså: En anställd kan jobba i en eller flera projekt. Ett projekt involverar en

eller flera anställda. Varje anställd måste jobba i ett projekt och varje projekt har minst en anställd som deltagare.

När vi översätter detta till ER-modellen får vi diagrammet som visas i figur 18.



Figur 18. Anställd arbetar med projekt.

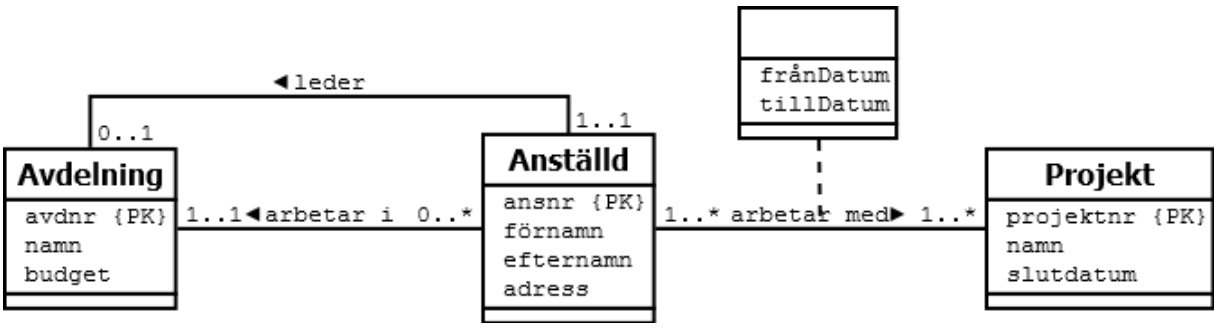
Lägg märke till att en sambandstyp i ER-modellen kan ha egna attribut. I det här fallet har sambandstypen 'arbetar med' två attribut: frånDatum och tillDatum. Ett många-till-många-samband låter sig inte direkt översättas till relationsmodellen. Vi får nämligen problem med att avgöra vilken av primärnycklarna som blir främmande nyckel i vilken entitetstyp.

Vad vi alltid gör vid många-till-många-samband är att lägga till en ny tabell i relationsmodellen. Den nya tabellen får som primärnyckel en sammansattnyckel bestående av de båda primärnycklarna som ingick i sambandet. Dessa blir även främmande nycklar till respektive ingående tabell:

```
Anställd (ansnr, förnamn, efternamn, adress)
Projekt (projektnr, namn, slutdatum)
Anställd_Projekt (ansnr*, projektnr*, frånDatum, tillDatum)
```

Komplett ER-diagram

Avslutningsvis tittar vi på ett relativt enkelt ER-diagram som använder sig av de olika symboler vi har tittat på i denna lektion. Fler symboler kommer att introduceras i nästa lektion.



Figur 19. ER-diagram över ett företag.

Översatt till relationsmodellen får i följande tabeller:

```
Anställd (ansnr, förnamn, efternamn, adress, avdnr*)
Projekt (projektnr, namn, slutdatum)
Anställd_Projekt (ansnr*, projektnr*, frånDatum, tillDatum)
Avdelning (avdnr, namn, budget, ansnr*)
```