Jørgen Koch

Access 2007 for alle

Normalisering m.v.

Access 2007 for alle – normalisering m.v.
1. udgave, 1. oplag 2007

Forfatter: Jørgen Koch

Indledning

Dette tillæg omhandler normalisering m.v., som der ikke blev plads til i hæftet *Access 2007 for alle*, der er udgivet på forlaget Libris.

Selv om skærmbillederne i dette hæfte stammer fra tidligere versioner, gælder disse principper også for Access 2007.

God fornøjelse!

Jørgen Koch

Normalisering

Når man designer databaser gælder det om at *normalisere* sine tabeller, hvilket vil sige, at man sørger for at mindske mængden af overflødige data mest muligt. Ideen i normalisering er – uden at vi skal blive alt for teoretiske her – er, at man ikke skal gemme sine data i en eneste stor tabel, da det giver alt for meget redundans.

Man skal i stedet splitte sine tabeller op i flere mindre tabeller, som man så knytter sammen ved hjælp af *relationer*. Ved hjælp af disse relationer er det muligt at arbejde med oplysningerne, som om de var placeret i en og samme tabel – men uden de ulemper, som vi har omtalt indtil nu.

Man har gennem tiden udviklet flere normalformer, hvor de mest normale er (startende med den mest simple, og sluttende med den mest effektive – man skal opfylde en lavere normalform for at kunne gå videre til næste normalform):

- Første normalform
- Anden normalform
- Tredie normalform
- Boyce-Codd normalform
- Fjerde normalform
- Femte normalform

I dette tillæg vil vi nøjes med at omtale de første tre normalformer, da de øvrige normalformer medfører en meget tungere administration og derved en database, der er vanskeligere at vedligeholde.

Første normalform

For at være på *første normalform* må en database kun indeholde atomistiske værdier. Det vil sige, at der for hver række og kolonne kun eksisterer en eneste værdi.

Ordrer - dårligt tabeldesign - tabel

	Ordrenr	Kundenr	Produkter
	1	4	5 Røgede sild, 3 Spegesild, 6 Chokolade
	2	23	1 Røget sild
	3	15	2 Geitost, 2 Filo Mix
	4	2	15 Gorgonzola Telino
	5	23	1 Spegesild
▶	6	2	5 Mozzarella di Giovanni
*	0	0	

Post: 6 af 6

Denne dårligt designede ordretabel bryder første normalform, idet oplysningerne i kolonnen Produkter ikke er atomistiske.

Fordelen ved denne regel skulle gerne være indlysende, da man ellers ikke har nogen mulighed for at manipulere disse værdier. Det ville fx være meget svært at arbejde med denne tabel – hvordan vil du oprette en rapport, der opsummerer det samlede køb pr. produkt?

Første normalform forbyder også repeterede felter (kolonner), idet man jo ellers kunne forbedre tabellen ved at erstatte kolonnen Produkter med seks kolonner som vist i den efterfølgende figur.

Ordrer - også et dårligt tabeldesign - tabel

Ordrenr	Kundenr	Antal1	Produkt1	Antal2	Produkt2	Antal3	Produkt3
1	4	5	Røgede sild	3	Spegesild	6	Chokolade
2	23	1	Røget sild				
3	15	2	Geitost	2	Filo Mix		
4	2	15	Gorgonzola Telino				
5	23	1	Spegesild				
6	2	5	Mozzarella di Giovanni				
✳	0	0	0	0		0	

Post: 1 af 6

Et bedre, men stadig ikke godt nok tabeldesign – de gentagne grupper sidst i tabellen bryder stadig første normalform.

Selv om dette design opdeler tabellen i flere felter, er der stadig et problem i designet. Hvis man skal oprette en forespørgsel, der leder efter det samlede antal solgte spegesild, skal forespørgslen lede i alle tre produktkolonner – og hvad nu, hvis en kunde ønsker at bestille mere end tre forskellige varer. Du kan selvfølgelig tage højde for dette ved at indføre tilstrækkeligt mange felter, men hvor vil du stoppe?

Lad os for eksemplets skyld sige, at du beslutter dig for 25 varer, og designer tabellen efter dette. Det betyder, at du vil have 50 kolonner i hver post, der kan indeholde oplysninger om antal og varer – også selv om kunden kun køber en eller to forskellige varer. Du kan sikkert godt selv se, at det er spild af plads, og du kan være helt sikker på, at der en eller anden dag dukker en kunde op, der vil have mere end 25 forskellige varer – og så er du lige vidt.

Tabeller i første normalform har ikke dette problem, og som du kan se i efterfølgende illustration, har vi løst problemet ved at indføre kolonnen *Ordrelinienr*. Den primære nøgle i denne tabel er en sammensat nøgle af ordrenummeret og ordrelinienummeret.

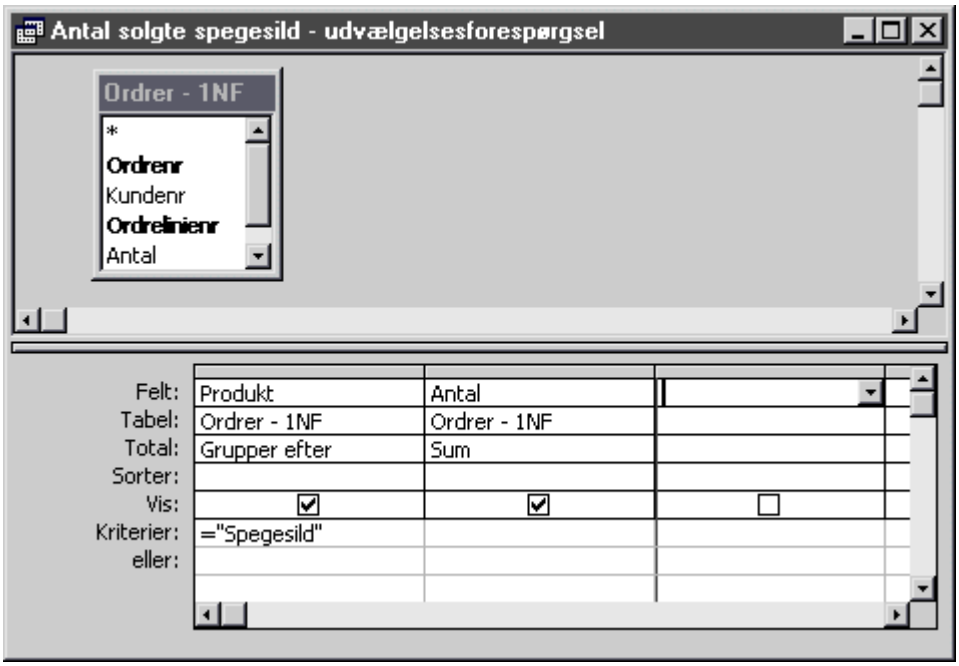
Ordrer - 1NF - tabel

Ordrenr	Kundenr	Ordrelinienr	Antal	Produkt
1	4	1	5	Røgede sild
1	4	2	3	Spegesild
1	4	3	6	Chokolade
2	23	1	1	Røget sild
3	15	1	2	Geitost
3	15	2	2	Filo Mix
4	2	1	15	Gorgonzola Telino
5	23	1	1	Spegesild
6	2	1	5	Mozzarella di Giovanni
✳	0	0	0	

Post: 1 af 9

Denne tabel er i første normalform.

Det er nu ganske let at oprette en forespørgsel, der tæller antallet af solgte spegesild, og tabellen er derfor meget lettere at arbejde med.



Da ordretabellen nu er i første normalform, er det ganske let at oprette en forespørgsel, der opsummerer antallet af solgte spegesild.

Anden normalform

En tabel siges at være på *anden normalform*, når den er på første normalform, og hver kolonne (nøglekolonner undtaget) i tabellen er afhængig af (hele) den primære nøgle. Med andre ord, så skal tabellen kun indeholde oplysninger om en enkelt entitet (begrebet *entitet* bruges til at beskrive de "ting", som tabellen beskriver – fx en ordre, en kunde, en patient osv.), og denne entitet bør være beskrevet af dens primære nøgle.

Tabellen i den efterfølgende illustration er en modificeret udgave af den tabel, som vi normaliserede til første normalform i forrige afsnit. Tabellen er stadig på første normalform, idet hver kolonne er atomistisk, og der ikke optræder gentagede felter i tabellen.

Både kundenummeret og ordredatoen gentaget for hver ordrelinie, men du kan sagtens identificere begge disse uden at kende ordrelinienummeret, hvorfor disse to kolonner ikke er afhængige af *hele* den primære nøgle. Tabellen er derfor ikke på anden normalform, men det klarer vi ganske let.

Ordrer - 1NF II - tabel

	Ordrenr	Kundenr	Ordredato	Ordrelinienr	Antal	Produkt
▶	1	4	01-05-2000	1	5	Røgede sild
	1	4	01-05-2000	2	3	Spegesild
	1	4	01-05-2000	3	6	Chokolade
	2	23	09-05-2000	1	1	Røget sild
	3	15	04-07-2000	1	2	Geitost
	3	15	04-07-2000	2	2	Filo Mix
	4	2	01-08-2000	1	15	Gorgonzola Telino
	5	23	02-08-2000	1	1	Spegesild
	6	2	02-08-2000	1	5	Mozzarella di Giovanni
*	0	0		0	0	

Post: 1 af 9

Eksempel på en tabel, der med fordel kan normaliseres til anden normalform.

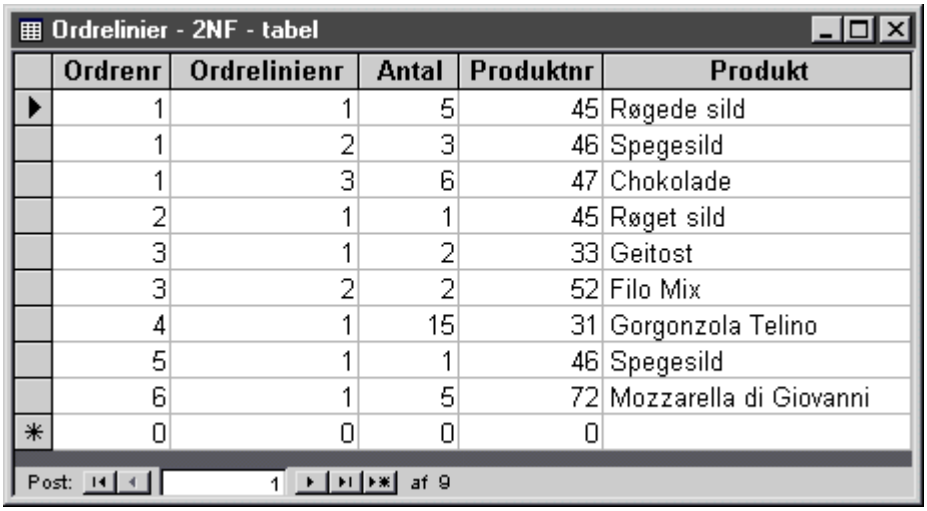
Du kan normalisere tabellen til anden normalform ved at opdele den i to tabeller. Placer alle oplysninger om den enkelte *ordre* i en tabel, og alle oplysninger om hver *ordrelinie* i en anden tabel.

Ordrer - 2NF - tabel

	Ordrenr	Kundenr	Ordredato
▶	1	4	01-05-2000
	2	23	09-05-2000
	3	15	04-07-2000
	4	2	01-08-2000
	5	23	02-08-2000
	6	2	02-08-2000
*	0	0	

Post: 1 af 6

Ordretabellen.



Ordrelinier - 2NF - tabel

Ordrenr	Ordrelinienr	Antal	Produktnr	Produkt
1	1	5	45	Røgede sild
1	2	3	46	Spegesild
1	3	6	47	Chokolade
2	1	1	45	Røget sild
3	1	2	33	Geitost
3	2	2	52	Filo Mix
4	1	15	31	Gorgonzola Telino
5	1	1	46	Spegesild
6	1	5	72	Mozzarella di Giovanni
0	0	0	0	

Post: 1 af 9

Ordrelinietabellen.

De to tabeller er nu på anden normalform. Kolonnen *Ordrenummer* i ordrelinietabellen er en fremmednøgle, som gør det muligt at samle de to tabeller igen i en forespørgsel.

Tredie normalform

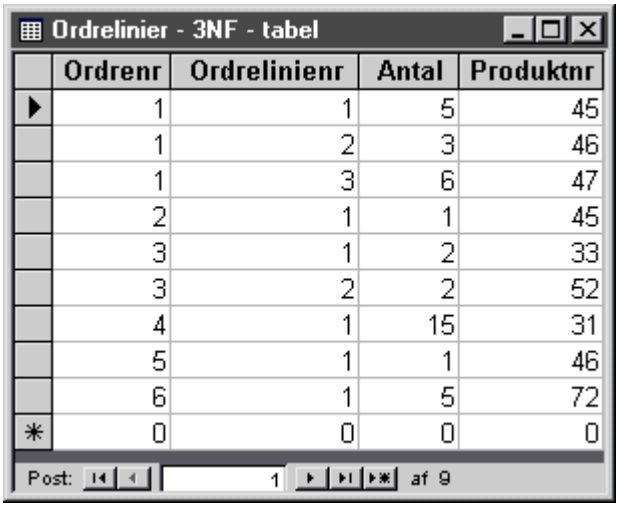
En tabel er på *tredie normalform*, når den er på anden normalform og når alle kolonner (nøglekolonner undtaget) er uafhængige af hinanden. Et indlysende eksempel på afhængighed er en beregnet kolonne. Hvis en tabel fx indeholder kolonnerne *Antal* og *PrisPerEnhed*, kunne man vælge at indsætte og beregne en kolonne med de totale omkostninger (det er jo bare at gange Antal med PrisPerEnhed), men en sådan tabel vil ikke være på tredie normalform. Det er bedre at udelade kolonnen fra tabellen, og så beregne omkostningerne direkte i forespørgslen, formularen eller rapporten – herved sparer du både plads i databasen og er fri for at skulle opdatere de totale omkostninger, hver gang enten antallet eller stykprisen ændrer sig.

Der kan også optræde andre afhængigheder i en tabel, som du kan se det i forrige illustration, hvor der er medtaget både et produktnummer og et produktnavn.

Denne afhængighed kan medføre problemer, når du tilføjer, opdatererer eller sletter poster i din tabel. Forestil dig fx, at du skal tilføje 100 ordrelinier, som alle involverer købet af røgede sild. Dette betyder, at du både skal indtaste produktnummeret ***45*** og produktbeskrivelsen ***Røgede sild*** (eller er det ***Røget sild***, såfremt der kun købes en?) for hver af disse poster, hvilket tydeligvis bryder med alle regler om redundans. Det samme problem vil opstå, hvis man ønsker at ændre betegnelsen til fx *Røgede Bornholmersild*, da du i så fald skal opdatere alle 100 poster i tabellen.

Eller hvad nu, hvis du ved udgangen af året ønsker at slette alle købsregistreringer. Når først alle posterne er slettet, vil du ikke længere vide, hvad produktnummer 45 er, da du både har slettet de historiske data og det faktum, at produktnummeret svarer til røgede sild.

Du kan løse dette ved at normalisere din database til tredie normalform, og opdele ordrelinietabellen i to tabeller – en tabel til de egentlige ordrelinier, og en opslagstabel indeholdende de forskellige produktnavne. Produktnummeret bliver den *primære nøgle* i den nye tabel og *fremmednøgle* i ordrelinietabellen, hvorved du kan forbinde de to tabeller i en forespørgsel.



Ordrelinier - 3NF - tabel

	Ordrenr	Ordrelinienr	Antal	Produktnr
▶	1	1	5	45
	1	2	3	46
	1	3	6	47
	2	1	1	45
	3	1	2	33
	3	2	2	52
	4	1	15	31
	5	1	1	46
	6	1	5	72
*	0	0	0	0

Post: 1 af 9

Her har vi opdelt ordrelinietabellen, så den kun består af produktnummeret, som så henviser til produktopslagstabellen. Tabellen er nu på tredie normalform, og opfylder derfor de mest normale krav til en velorganiseret database.

Som tidligere nævnt findes der en række yderligere normalformer, herunder Boyce-Codd, fjerde normalform og femte normalform, hvilket vi dog ikke vil omtale nærmere i dette kursusmateriale.

Der er dog et par punkter, som det er værd at bemærke sig:

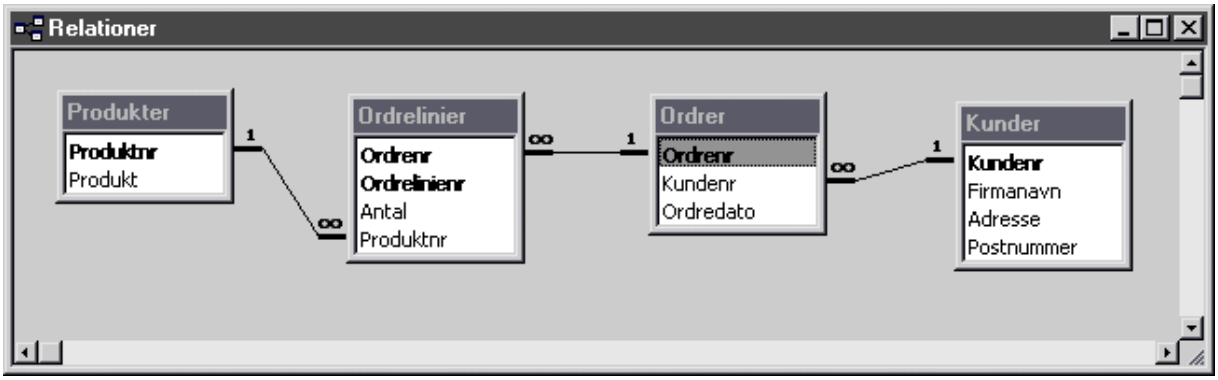
- Enhver højere normalform består automatisk af alle lavere former – hvis dit databasedesign er på tredie normalform (3NF), opfylder den pr. definition også første normalform (1NF) og anden normalform (2NF).
- Hvis du har normaliseret din database til 3NF, opfylder den sandsynligvis også Boyce-Codd (og måske endda 4NF og 5NF).
- Principperne i databasedesign kan være meget teoretiske, men er *egentlig ikke mere end formaliseret fornuftig sans*, for nu at citere en anden forfatter om emnet.

Vi har flere gange nævnt, at vi vil forbinde de opdelte tabeller i en forespørgsel, hvilket gøres ved at oprette en *relation* mellem tabellerne.

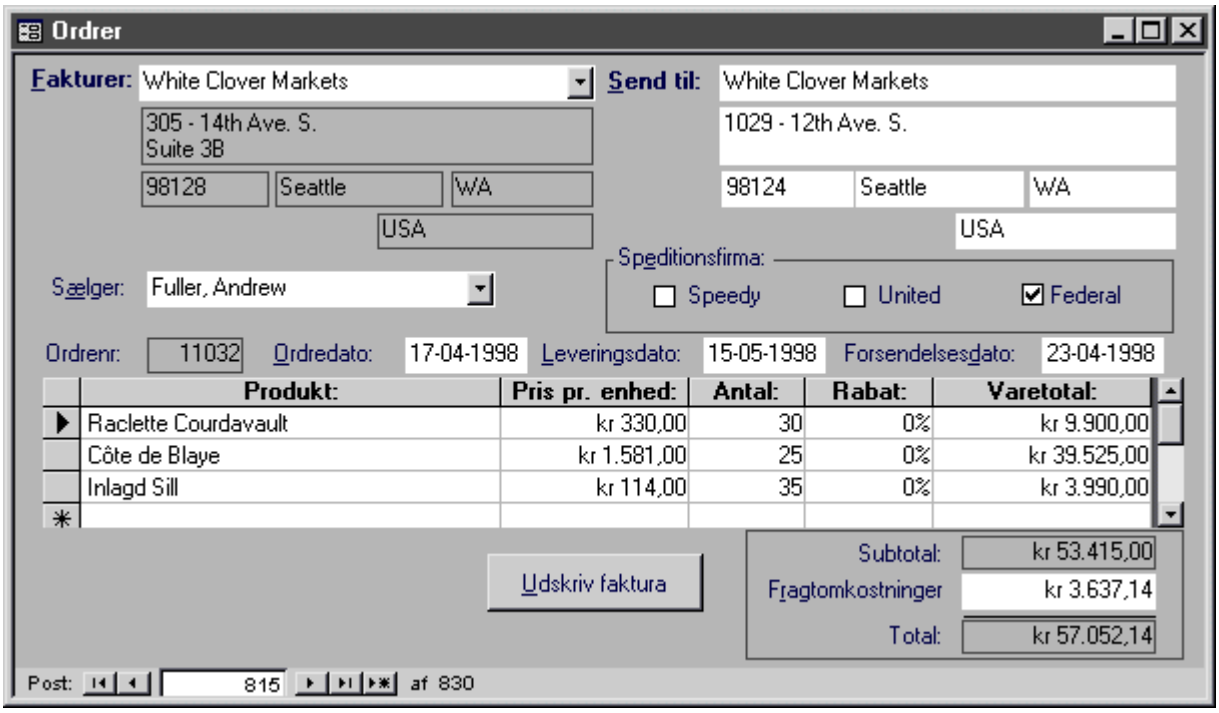
Relationer

De foregående eksempler viste alle, hvordan vi fjerner redundans i en database ved at opdele tabellen i flere tabeller. Vi skal nu se på, hvordan vi samler de opdelte oplysninger igen ved hjælp af relationer.

Forestil dig fx, at du vil ringe til en af dine kolleger med et spørgsmål omkring et af de salg, som pågældende har stået for. Kollegaens telefonnummer findes i medarbejdertabellen, mens salgene er registreret i ordretabellen. Når du fortæller Access, hvilket salg du er interesseret i, kan Access slå telefonnummeret op på baggrund af den relation, der er mellem de to tabeller.



Eksempel på opbygning af relationer.



Eksempel på en formular, der benytter sig af flere forskellige tabeller – oplysningerne øverst i formularen kommer fra kundetabellen, sælgerne kommer fra sælgertabellen, produktkolonnen kommer fra produkttabellen, mens pris, antal og rabat kommer fra ordrelinietabellen.

At det i det hele taget kan lade sig gøre skyldes, at feltet *Medarbejdernr*, som er primær nøgle i medarbejdertabellen, også findes i ordretabellen – feltet *Medarbejdernr* i ordretabellen kaldes for en fremmednøgle, idet feltet er en primær nøgle fra en anden tabel.

Når du skal opbygge relationer mellem to tabeller, skal du tilføje den ene tabels primære nøgle til den anden tabel, så den optræder i begge tabeller. Spørgsmålet er så bare, hvilken af tabellernes primære nøgle, man skal anvende? For at kunne svare på dette, skal du først afgøre, hvilken relation der er tale om.

Der findes tre forskellige relationstyper:

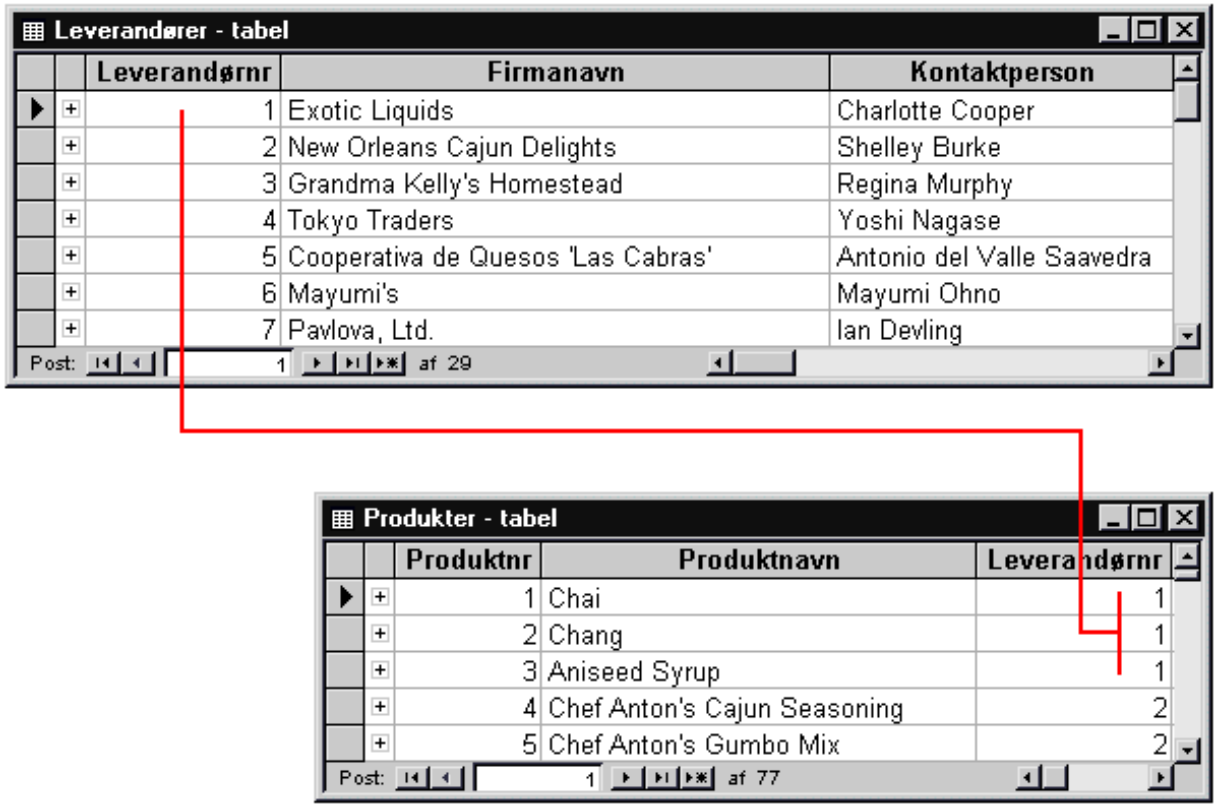
- En-til-mange-relationer

- Mange-til-mange-relationer
- En-til-en-relationer

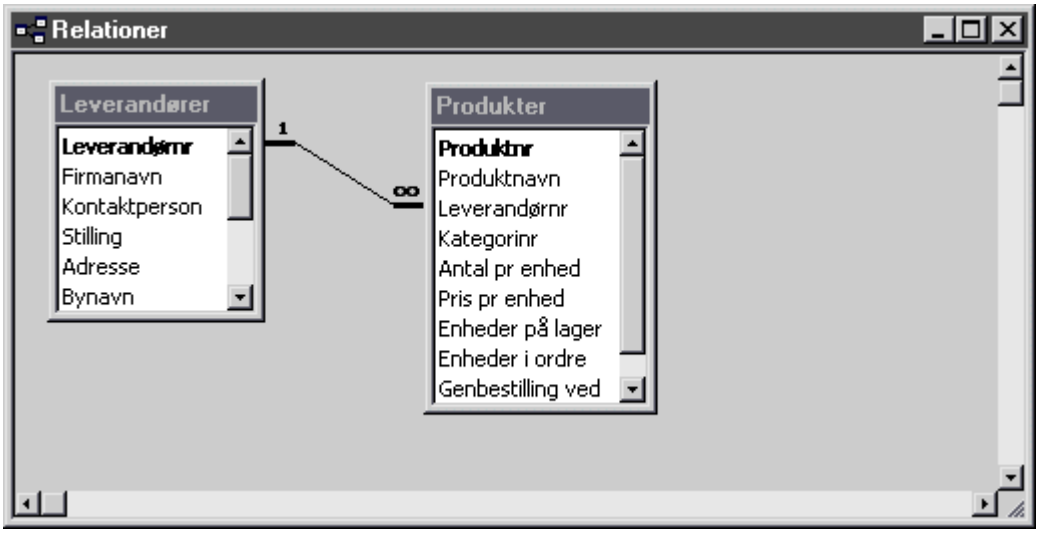
Vi vil i det følgende kort introducere dig for de forskellige typer, og forklare, hvordan du skal designe dine tabeller på den rigtige måde.

En-til-mange-relationer

En *en-til-mange-relation* er den mest normale relationstype i en relationsdatabase. I en en-til-mange-relation kan en post i tabel A have mere end en matchende post i tabel B, mens en post i tabel B højest har *en* matchende post i tabel A.



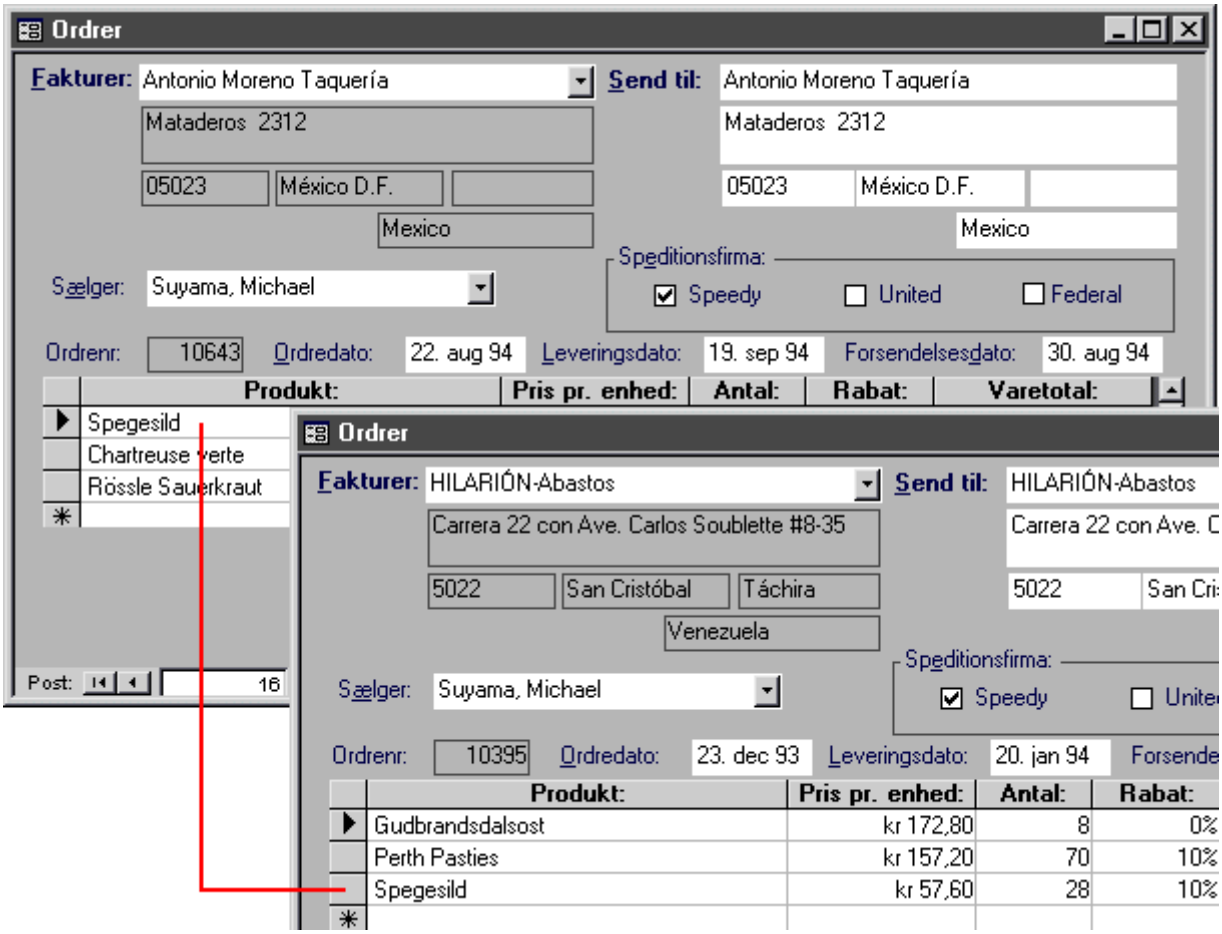
Når du skal oprette denne form for relation, skal du tilføje det eller de felter, der udgør den primære nøgle på *en*-siden, til tabellen på *mange*-siden. I det viste eksempel har vi tilføjet leverandørnummeret fra leverandørtabellen til produkttabellen, idet *en* leverandør kan levere *mange* produkter – ved hjælp af leverandørnummeret kan vi nu finde den korrekte leverandør af hvert produkt.



Eksempel på en en-til-mange-relation.

Mange-til-mange-relationer

I en *mange-til-mange-relation* kan en post i tabel A have mere end en matchende post i tabel B, men en post i tabel B kan også have mere end en matchende post i tabel A.



For at bestemme mange-til-mange-relationer i din database, er det vigtigt at du betragter begge sider af relationen. Se fx på relationen mellem ordrer og produkter i eksempeldatabasen. En ordre kan indeholde mere end et produkt (eller fik vi jo aldrig gang i biksen, vel?), hvorfor der for hver post i ordretabellen kan findes *mange* poster i produkttabellen. Men prøv at se på det fra produkttabelsiden, da hvert produkt jo kan optræde på mange forskellige ordrer. For hver post i produkttabellen kan der findes *mange* poster i ordretabellen.

Indholdet i de to tabeller – ordrer og produkter – anvender en mange-til-mange-relation, hvilket giver et lille problem i designet af databasen. For at forstå dette problem, skal du prøve at forestille dig, hvad der ville ske, hvis du tilføjede produktnummeret til ordretabellen. For at have mere end et produkt pr. ordre, ville du behøve en post pr. produkt pr. ordre – du ville med andre ord gentage ordreoplysninger igen og igen i hver post, der henviste til samme ordre, hvilket jo ikke er den mest hensigtsmæssige måde at designe databaser på.

Du løber ind i samme problem, hvis du placerer ordrenummeret i produkttabellen – du vil ende med at have flere poster i produkttabellen for hvert produkt. Hvordan pokker løser man dette problem?

Løsningen på dette er at oprette en tredie tabel, der nedbryder mange-til-mange-relationen i to en-til-mange-relationer ved at placere den primære nøgle fra de to tabeller i den tredie tabel.

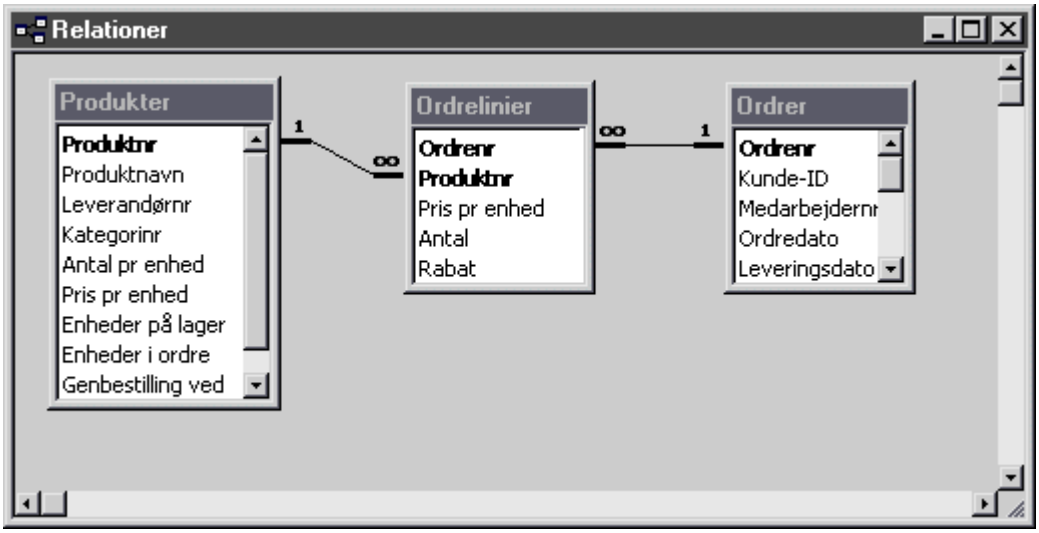
Ordrelinier - tabel

Ordrenr	Produktnr	Pris pr. enhed	Antal	Rabat
10248	11	kr 84,00	12	0%
10248	42	kr 58,80	10	0%
10248	72	kr 208,80	5	0%
10249	14	kr 111,60	9	0%
10249	51	kr 254,40	40	0%
10250	41	kr 46,20	10	0%
10250	51	kr 254,40	35	15%
10250	65	kr 100,80	15	15%
10251	22	kr 100,80	6	5%
10251	57	kr 93,60	15	5%
10251	65	kr 100,80	20	0%

Post: 1 af 2228

I denne tabel kan du sikkert genkende den første kolonne som den primære nøgle fra ordretabellen og den anden kolonne som den primære nøgle fra produkttabellen. De sidste tre koloner indeholder oplysninger som refererer til både ordre- og produktnummeret – hvilket jo svarer til den enkelte ordrelinie.

Hver post i ordrelinietabellen repræsenterer en ordrelinie i en ordre. Ordrelinietabellens primære nøgle består af to felter – fremmednøglerne fra ordre- og produkttabellerne. Ordrenummeret alene kan ikke bruges som primær nøgle i denne tabel, da en ordre kan have flere ordrelinier, og ordrenummeret derfor ville blive gentaget for hver ordrelinie i tabellen – og derved ikke opfylde kravet om unikke værdier til nøglefelter. Produktnummeret alene kan heller ikke bruges som primær nøgle, da et produkt jo også kan optræde på flere ordrer, men ved at sammensætte ordre- og produktnummeret får man en unik værdi for hver post.



Eksempel på en mange-til-mange-relation.

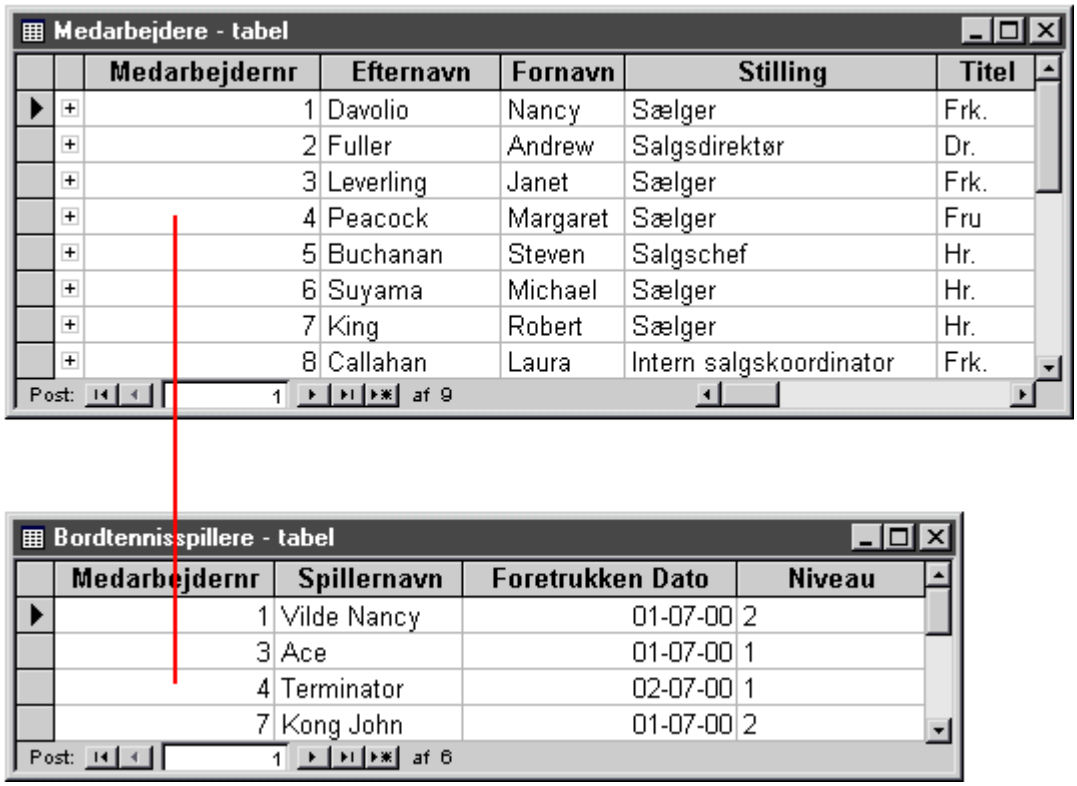
I vores eksempeldatabase er ordre- og produkttabellerne ikke direkte relateret, men i stedet relateret indirekte ved hjælp af ordrelinietabellen. Mange-til-mange-relationen mellem ordrer og produkter er repræsenteret ved at oprette to en-til-mange-relationer:

- Ordre- og ordrelinietabellen har en en-til-mange-relation, idet hver ordre kan have mange ordrelinier, men hver ordrelinie kun kan tilhøre en ordre.
- Produkt- og ordrelinietabellen har en en-til-mange-relation, idet hvert produkt kan optræde på mange ordrelinier, men hver ordrelinie kun kan indeholde et produkt.

En-til-en-relationer

I en en-til-en-relation kan en post i tabel A højest have *en* matchende post i tabel B, og en post i tabel B også højest have *en* matchende post i tabel A. Denne form for relation er lidt usædvanlig, og kan som mange-til-mange-relationen medføre, at du skal ændre dit tabeldesign.

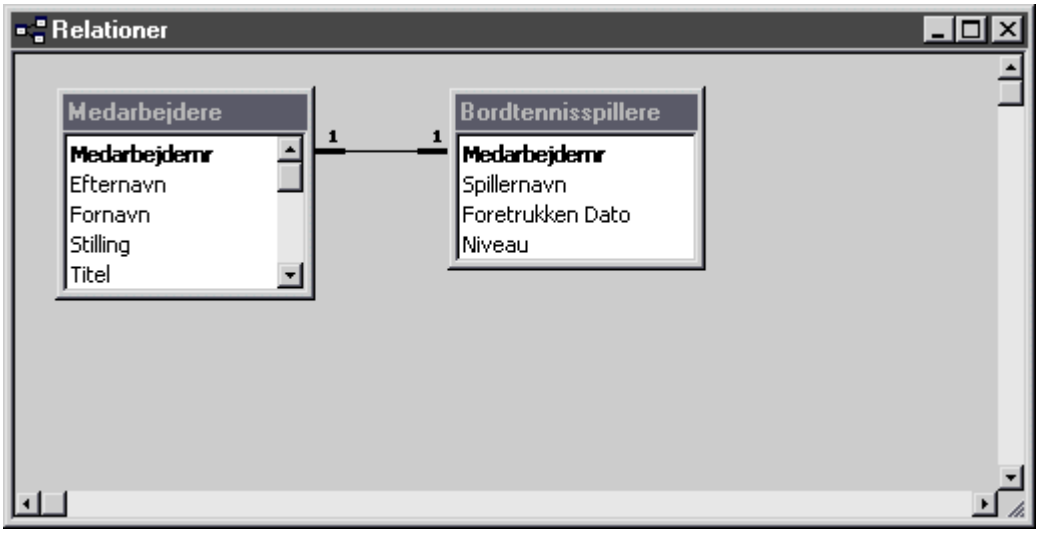
En-til-en-relationer er udsædvanlige, idet man i mange tilfælde blot kunne kombinere oplysningerne i de to tabeller i en og samme tabel. Lad os fx antage, at du opretter en tabel over bordtennisspillende medarbejdere i forbindelse med det sportsstævne, som ledelsen har besluttet at afholde i forbindelse med årtusindskiftet. Da alle bordtennisspillerne jo også er ansatte i virksomheden, er der tale om en en-til-en-relation til din medarbejdertabel.



Hver bordtennisspiller har en matchende post i medarbejdertabellen.

Man kunne selvfølgelig agitere for, at man da bare kunne tilføje disse felter til medarbejdertabellen, men da sportsstævnet kun afholdes denne ene gang (der er jo lidt længe til det næste årtusindskifte), og du derfor ikke efterfølgende har brug for oplysninger, er det smartere at oprette en ekstra tabel til netop dette formål.

Derudover er det jo ikke alle medarbejderne, der spiller bordtennis, hvorfor disse felter ville være tomme for de fleste af medarbejdernes vedkommende, hvilket man skal sørge for at undgå.



Eksempel på en en-til-en-relation.

Integritet

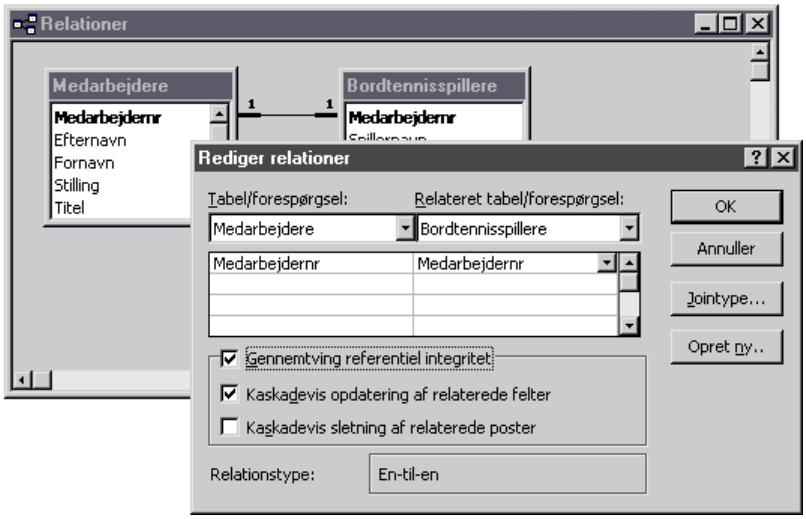
Den relationelle databasemodel definerer flere integritetsregler, som omfatter entitetsintegritet og referentiel integritet.

Entitetsintegritetsreglen (sikken et ord) er meget simpel, idet den går ud på, at primære nøgler ikke kan indeholde nul-værdier (såkaldet Null-værdier), hvilket Microsoft Access tager hånd om helt automatisk.

Den referentielle integritet går ud på, at databasen ikke må indeholde umatchede fremmednøgler, hvilket kan forklares således:

- Det er ikke muligt at tilføje en post til en tabel med en fremmednøgle, medmindre værdien i fremmednøglekolonnen eksisterer i den relaterede tabel.
- Hvis den primære nøgles værdi ændres (eller slettes) i den relaterede tabel, skal posterne i tabellen med fremmednøglen ændres således, at de så refererer til den ændrede nøgle (eller slettes).

Microsoft Access understøtter to generelle løsninger på dette problem, idet du kan vælge enten *ikke* at tillade ændringer eller gennemføre kaskadevis opdatering i forbindelse med opdatering eller sletning af poster i den primære tabel.



I ovenstående figur kan du se et eksempel på, hvordan man kan etablere referentiel integritet. Vi har i eksemplet både valgt referentiel integritet og kaskadevis opdatering af relaterede felter (vær varsom med kaskadevis sletning af relaterede poster, da du i så fald kan komme til at slette en masse poster, som du ikke havde regnet med).