DATABASE DESIGN

En note om database design, normalisering og database generalisering

Resume:
Følgende note, er en indførsel i problemstillingerne for at gå fra 'virkelighedens' problemstilling der skal designes til et databaselayout som en meget kort introduktion. Begreber fra den objekt-orienterede analyse introduceres.

Ligeledes en introduktion til 3 typer af normalisering: Felt normalisering, Tabel normalisering, og Database normalisering.
Oftest kun Tabel normalisering der undervises i på diverse undervisningsinstitutioner (og ofte kun til 3. normalform).

Der beskrives med eksempler på tabel normalisering af 1-5 normal form inkl. Boyce-Codd normalform.

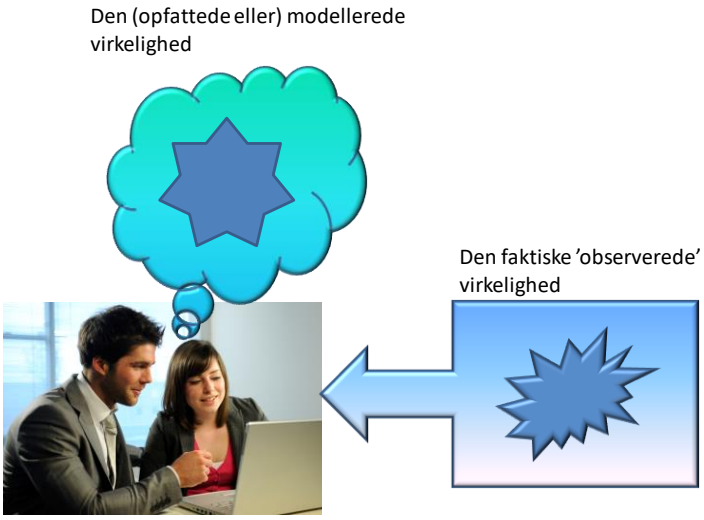
I appendix diskuteres problemstillingen med definition af 1. normalform.

Database design af virkeligheden

Model af virkeligheden

Når man skal skabe en database for at lagre data for en given problemstilling er database ofte en simplificering af det problemområde som eksisterer i virkeligheden. Det som man ønsker at håndtere er ofte en række datastrukturer der er indbyrdes relatede. Det er som oftest et ønske om kun at registere en mindre delmængde af de mest relevante informationer fra den ønskede virkelighed.

Man oplever dog ofte, at der opstår en kombination af bevidst valgte afgrænsninger men også ikke-opfattede væsentlige informationer når der skabes en given model jf. Nedenstående figur.



Virkeligheden som de fleste der skal skabe en model opfatter består af objekter. Hvert objekt har nogle unikke informationer der identificerer det givne objekt, og det er disse data som man ønsker at lagre i en database. Dette kan være informationer om personer, virksomheder, Produkter, begivenheder etc.

For at kunne gemme informationerne fra specifikke indentificerede objekter skal man have beskrevet det som er fælles for alle objekter. Altså en klassificering af ensartede objekter - (en generel beskrivelse af ensartede objekter kan beskrives som en klasse). Herom i det næste afsnit.

Der findes ingen formel for hvorledes man bliver en "god konstruktør af virkeligheden". Det er oftest et spørgmål om masser af erfaring, men også i høj viden om værktøjerne som vil blive beskrevet i det følgende, samt en lille smule flair.

Det objekt-orienterede begrebsapparat

I det følgende bliver en lille delmængde af den objekt-orienterede teori introduceret, da ideer fra dette område kan benyttes når man skal modellere en given virkelighed og have en database beskrevet.

Klasse og objekt definitioner

I det forrige afsnit blev begreberne objekt og klasse introduceret.

Klasse I objekt-orienteret termonologi er en klasse en beskrivelse der definerer ensartede objekter. Altså en skabelon for hvordan objekter af den givne klasse ser ud og agerer.

Objekt I objekt-orienteret termonologi er et objekt en instans af en givet klasse. Med instans menes et faktisk skabt og fungerende objekt baseret på klasse skabelonen.

Attribut I objekt-orienteret termonologi er et attribut en specifik karakteristik i en given klasse.

En attribut kan eksempelvis være navn, farve, størrelse etc. Beskrivelsen der definerer en klasse er altså bl.a. netop en mængde attributter der beskriver klassens indhold. En attribut har en given datatype og en given indholdsmængde også kaldet domæne (definition af domæne senere). En attribut svarer til et felt i en tabel i en database.

Eksempelvis kan man sige, at en arkitekttegning af et hus er en klasse af det givne hus. Det faktisk byggede hus er en instans af den givne klasse og dermed et objekt af denne arkitekttegning. Der kan oftest bygges mange forholdsvis ensartede objekter af den samme klasse, men objektets attributter såsom valg af byggestenstype, tag, farver etc. kan være forskellige fra objekt til objekt, men de tilhører stadig den samme klasse, og klassen har samme mængde attributter.

Fra objekter over klasse til relations tabeller

Når man i sin model af virkeligheden har identificeret et eller flere objekter som man ønsker at gemme informationer om er der som oftest tale om objektets klasse attributter.

Det gælder derfor om at få analyseret objekterne og identificeret hvad der er fælles attributter som beskriver den generaliserede klasses attributter. Klassen i form af dens attributter kan mappes direkte til en tabel (eller mere korrekt benævnt: en relation – Se definition nedenfor).

Det gælder altså om, at tænke i objekternes abstrakte klasse attributter for at få defineret hvilke karisteristika (klassens attributter) man ønsker at lagre på database niveau.

Man skal ikke tænke i specifikke objekter, men i klasser og disse klassers attributter. En Tabel definition kan sammenlignes med beskrivelsen af en klasses attributter (attribut=felt), mens de enkelte rækker i tabellen kan sammenlignes med forskellige objekter med egentlige specifikke værdier i hver af attributterne.

Tabeller og normaliseringstankegange

Nogle database tekniske definitioner

Relation Database teoretisk er en relation en uordnet mængde af relaterede attributter og en mulig mængde af uordnede tupler.

En relation består derfor af en uordnet men relateret mængde af attributter og en mulig mængde af uordnede tupler (tupler: svarende til rækker i en tabel og rekords i en database). En tuple kan derfor opfattes som værende en tabels 'objekt', mens tabeldefinitionen kan opfattes som værende klassebeskrivelsen som definerer hvilke objekter der kan tilhøre klassen.

Med relaterede attributter menes, at hver attribut der indgår i relationen har minimum en relation til en anden attribut i relationen.

Kortfattet sammenhæng mellem objekt-orienteret teori og database definition.

- Klasse => Tabeldefintion (relation) (mængde af relaterede attributter/felter)
- Objekt => Rekords i tabellen (tupler)

Domæne Database teoretisk er et domæne udtryk for det udfald af værdier en given attribut kan indeholde i en relation.

En attribut der hedder farve kan altså indeholde navne på farver eller farvekoder eller en anden identifikation af hver farve, og domæne begrebet er altså dels en beskrivelse af typen der identificerer farven og den mulige udfaldsmængde af værdier til attributten.

En tabeldefinition i en database er derfor en implementeret beskrivelse af en relation som beskriver mængden af attributter (felter) og deres typer. De kommercielt største tilgængelige relationsdatabaser implementerer dog ikke tabel definitions begrænsninger via tabeldefinitionen på attributtens domæne da dette ikke indgår i de officielt standardiserede sprog til database manipulation (SQL).

Ved at få beskrevet 'virkeligheden' som man ønsker at modellere i den afgrænsede models klasser og de attributter som man ønsker at gemme informationer om, har man altså de nødvendige informationer til at få defineret de tabeller i databasen som skal benyttes i modellen.

Integritets begrebet

Man arbejder med 3 hoved integritetsområder inden for database teori. Entitets integritet (se senere under beskrivelsen af 1. normalform), og dels referentiel integritet og dels semantisk integritet.

Referentiel integritet

Ved referentiel integritet forstås, at sammenhørende data mellem 2 tabeller altid sikres.

Eksempel: En kundetabel og en ordretabel. Kundetabellen består af alle oprettede definerede kunder. Hver kunde har et unikt kundenummer som primærnøgle. Ordretabellen består bl.a. af et kundenummer og et ordrenummer.

Hvis der er mulighed for ændre kundenummeret i kundetabellen, skal man med databasemæssig referentiel integritet sikre sig, at alle rekords i ordretabellen også bliver ændret til det nye ændrede kundenummer for at kunne sige, at der er 'referencemæssig integritet' mellem de 2 tabeller. Sikres dette ikke har vi 'forældreløse' (Engelsk: Orphans) kundenumre i ordretabellen hvor man ikke længere kan finde den korrekte kundebeskrivelse (forælder) i kundetabellen.

Det samme problem opstår hvis man sletter en kunde i kundetabellen.

På engelsk kalder man også denne sammenhæng mellem de 2 tabeller som et Master/Slave relationship. Kundetabellen er Master (kunde som primærnøgle), og Ordretabellen er Slave (med kundenummer som fremmednøgle).

I SQL sproget (ANSI standard) er en mulig sikring af sådan referentiel integritet implementeret med at man kan definere ON DELETE CASCADE og ON UPDATE CASCADE på tabel niveau som kan sikre, at når man enten sletter en rekord eller opdaterer primærnøglen i en hovedtabel så gennemfører databasen en tilsvarende opdatering i underliggende tabeller (der har en fremmednøgle linket op på hovedtabellen).

Dermed er der referencemæssig integritet i databasen, men samtidig skal man jo også gøre sig klart, at eksempelvis når man sletter en kunde i kundedatabasen i ovenstående eksempel bliver alle ordrer (og andre transaktioner der tilsvarende ville være hæftet op på kundetabellen) slettet. Man sikrer integriteten, men mister historikken.

Hele problemstillingen med historik håndtering og eksempelvis dataware house opbygning vil være for omfattende at komme nærmere ind på i dette skrift. Se nærmere i skrift om Business Intelligence og "slowly changing dimensions".

Semantisk integritet

Med semantisk integritet (også af andre kaldet domæne integritet) forstås sikring af at værdier i et givet felt kun indeholder tilladte værdier af det givne domæne. Se definitionen tidligere.

Man må ikke forveksle semantisk/domæne integritet med datatypen. Datatypen kan siges at være laveste niveau på attributniveau for semantisk integritet. Datatypen definerer hvilken type af værdier en given attribut kan indeholde, men den definerer ikke domænet (alle valide udfald af værdier). Domænet er som sagt den udfaldsmængde af den givne definerede datatype som attributten/feltet i relationen/tabellen må indeholde.

Der er en tydelig problemstilling med hvorledes man sikrer sig semantisk integritet, i at det er endog meget problematisk til alle tider, at have fuld viden om alle mulige udfald af et givent felt. Eksempelvis et felt der må indeholde valide telefonnumre. Hvordan sikrer man sig, at der kun indtastes valide eksisterende telefonnumre i et sådant felt.

Teoretisk er det enkelt, at forlange semantisk integritet, men i praksis er der for masser af attributters vedkommende store problemer med at sikre dette.

I en række tilfælde hvor man på forhånd kun har en vis mængde udfald, som man deslige ønsker bliver registreret ensartet fra transaktion til transaktion, implementerer man ofte dette ved også at benytte referentiel integritet. Udfaldsmængden bliver defineret endeligt i en Master tabel (se ovenfor under referentiel integritet), og der laves så en fremmednøgle fra feltet i den tabel hvor transaktionerne skal lagres til denne mastertabel. Derved sikrer databasen også, at kun værdier der er defineret i mastertabellen (domænet der fuldt defineret) kan indtastes.

Dette er god praksis i alle de tilfælde hvor man har muligheden og hvor det er væsentligt at de værdier man registrerer er de samme fra gang til gang (ingen tastefejl, ingen forskel på små og store bogstaver, ingen ugyldige koder etc, men det betyder også samtidigt, at en given "virkelighed" der skal beskrives hurtigt får rigtig mange 'hjælpetabeller' med definition af domæner. Alle disse domænedefinitionstabeller skal jo også vedligeholdes og opdateres.

Hvis man gik hele vejen, og havde et domænefelt man havde defineret som alle valide danske telefonnumre, og havde mulighed for at få disse indlæst i en tabel (inklusive hemmelige numre m.m.) ville denne tabel jo nærmest skulle opdateres online real-time for at man kunne være sikker på, at ville være valid.

Som det fremgår af ovenstående skal man altså i designet af databasen sikre sig den nødvendige semantiske integritet på de områder hvor det er væsentligt for den model der forsøges at blive implementeret.

Rigtig mange applikationer (inklusive mange ERP-systemer) er set med store databaser hvor der er væsentlige huller i både den referentielle integritet og i den semantiske integritet. Når man efterfølgende skal lave rapportering på en sådan database og pludselig finder ud af, at der er 'ting der ikke hænger sammen' står man med store problemer.

Som minimum skal man sikre sig den nødvendige referentielle integritet, og må sørge for at der er tilstrækkelig semantisk integritet i forhold til både den daglige brug af modellen i forhold til registreringerne og i forhold til en efterfølgende nødvendig rapportering og statistik på de registrerede informationer.

Som et underpunkt til den semantiske integritet findes også begrebet overgangs- eller faseskift integritet. Hermed menes, at der kan være en vis udfaldsmængde i domænet (eksempelvis status angivelse / status koder) som skal have en vis rækkefølge. Altså, at man kun kan gå fra en status til en efterfølgende (eller tidligere) alt efter typen. Det kan være en ordre som har forskellige stati i form af: oprettet, under plukning, plukket, pakket, afsendt, leveret, faktureret, betalt e.l.).
I disse tilfælde er det ikke nok, at oprette en ekstra tabel med domænets mulige udfald, men man bliver også nødt til at implementere logik (enten direkte i databasen via database programmets muligheder) eller via applikationskode logik der sikrer, at kun valide faseskift sker som defineret.

Introduktion til normalisering

Normalisering som begreb

Normalisering som begreb kan have flere betydninger afhængig af hvilken 'videnskab' man arbejder indenfor. Inden for matematikken kan det være en teknik for bevisførelse eller generel transformation fra en tilstand til en anden.

Som generelt begreb kan normalisering siges at være, en hvilken som helst proces eller funktion der transformerer noget til en mere 'normal' eller logisk form.

Database teoretisk kan man faktisk sige, at der består 3 typer af normalisering, hvoraf de fleste kun er blevet – eller ofte bliver - introduceret til den mellemste af nedenstående 3 former:

- **Felt normalisering (semantisk analyse der fjerner domæne redundans på attribut niveau)**
- **Tabel normalisering (fjernelse af redundante data på tabel niveau)**
- **Database normalisering (generalisering af tabeller, der fjerner redundante felter ml. tabeller)**

Felt normalisering

Med felt normalisering skal forstås at man semantisk (betydningsmæssigt) forsøger at gennemføre en analyse over en given attributs domæne for at sikre sig, at feltet ikke indeholder flere repeterende informationer der burde splittes op i hver sin attribut i forhold til entitetens definition.

Eksempelvis hvis man har et TYPE felt der beskriver eller anfører en eller anden form for kategorisering af den enkelte rekord. Lad os sige, at domænet har følgende udfald:

Rød og Lille
Rød og Mellem
Rød og Stor
Gul og Lille
Gul og Mellem
Gul og Stor

Det er klart, at for hver gang man introducerer en ny farve (og dermed udvider domænet) kommer der 3 nye værdier i domænet, og hvis man også introducerer en ny størrelse (f.eks. mellemstor) ja så bliver antallet i domænet ganget op med antallet af definerede farver.

I ovenstående situation har vi en multiværdi kombination mellem 2 typer af information på felt niveau. Hver information om farve kan ganges med hver information om størrelse for at give domænets udfald, og dermed har man redundante data i domænet.

Dette løses ved at splitte feltet/attributten op i 2 felter. Et felt der indeholder farven, og et felt der indeholder størrelsen. Dette vil de fleste sikkert gøre naturligt allerede ved definitionen af

klassens/tabellens attributter, men det er ikke et klart defineret krav i normal tabel normalisering at man foretager denne skelnen på feltniveau.
Dermed bliver en given farve kun medtaget en gang i farveattributtens domæne, og størrelsen kun en gang i størrelsesfeltets domæne. Har man kun 3 kategorier af størrelse som angivet i ovenstående eksempel vil størrelsesdomænet altså kun bestå af {Lille, Mellem, Stor}.

Et felt kan altså siges at være normaliseret når det (efter bedste evne – se appendix) ikke indeholder repeterende multiværdi information i domænet.

Semantisk feltanalyse skal ikke forveksles med semantisk integritet. Med semantisk feltanalyse undersøges, at domænet ikke har repeterende multiværdi information. Med semantisk integritet forsøger man at sikre sig, at et rekordfelt ikke indeholder en værdi som ikke er defineret i domænet.

Tabel normalisering

Normalisering af en tabel

At normalisere en tabel kan lidt populært siges at sikre at data ikke gentages flere steder når man indtaster flere rækker. Data der står flere steder i samme database kaldes *redundante* data. Ifølge normaliseringsteori gælder det om at reducere redundante data mest muligt.

Første sikring mod redundans er altså på feltniveau at sikre, at der ikke er multi værdi information med repeterende værdier i domænet. Dernæst skal man på tabelniveau sikre, at værdier i det givne felt i mindst muligt omfang repeteres mellem rækkerne.

Der findes mange grader af normalisering, hvor man nedbryder en eksisterende tabel til stadig flere tabeller for at sikre ovenstående.
Man taler om forskellige normaliseringsformer. Disse går fra tabeller fra 1. normalform til tabeller af 5. normalform (eller 6. normalform). I mange tilfælde vil tabeller der er normaliseret til 3. normalform også være normaliseret 'helt i bund', således at de også er på 4. og 5. normalform. De sidste 2-3 normalformer kan oftest opfattes som specialtilfælde for visse tabeller. Ofte med primærnøgler sammensat af flere felter.

Der findes helt formelle definitioner for hvornår en tabel er på en given normalform. I det følgende forsøges disse at blive'oversat' og forklaret fra den mere formelle definition.

Bemærk dog, at tabel normaliseringsteori har en teoretisk tilgangsvinkel med mange teoretikere, og mange forskellige definitioner hvor der er forfattere der faktisk ikke engang er enige om definitionen af 1. normalform. To af de mest kendte forfattere (Edgar F. Codd og Chris Date) har derfor forskellige definitioner af 1. normalform. Der henvises til disse forfattere for nærmere beskrivelser af forskelle eller til søgning på internettet af de 2 definitioner. Se dog også Appendix for en teoretisk diskussion af problemstillingerne ved definitionen af 1. normalform.

En u-normaliseret tabel.

Vi starter med en unormaliseret tabel med 5 oplysninger: Leverandørnummer, Postnummer (på leverandøren), by, produktnr, samt antal. I den unormaliserede tabel er rekord 3 længere (har 2 felter mere hvor produktnr og antal felterne gentages) end de andre rekords.

Lev-nr	Postnr	By	Produktnr	Antal	Produktnr	Antal
1	4000	Roskilde	1, 2, 3	100, 150, 125		
2	8800	Viborg	1, 2, 3	100, 100, 250		
3	4000	Roskilde	1	50	2	100
4	9000	Ålborg	1, 2, 3	100, 125, 175		

1NF *Tabellen skal have fast rekordlængde og alle rekords skal være ens (en og kun en rekordtype pr. tabel). Hver rekord skal kunne identificeres entydigt (der skal kunne dannes en primærnøgle - dermed er en NULL værdi i nøglen ikke tilladt). Tabellen skal indeholde atomariske værdier. Sammendrag: Hver rekord skal kunne identificeres entydigt, og skal indeholde atomariske værdier.*

Med angivelsen af, at en rekord skal kunne identificeres entydigt er dette også beskrivelsen af hvad man kalder entitetsintegritet.

I ovenstående definition benyttes begrebet 'atomariske værdier'. Dette begreb har været til diskussion, da et tekstfelt jo kan indeholde flere ord, og i teorien vil et sådant felt ikke være atomarisk i sin mest stringente fortolkning. Uden at skulle gå ind i en alt for dyb teoretisk diskussion her, kan man tillægge en praktisk fortolkning af atomarisk således, at et givet felt ikke må indeholde flere værdier af samme type (domæne værdier - eksempelvis et telefonnummerfelt der indeholder flere telefonnumre), eller et felt der indeholder flere værdier af forskellig type (Eksempelvis et felt der både beskriver størrelse og farve: Stor og Rød; Mellem og Grøn; Lille og Rød etc, som beskrevet tidligere under felt normalisering). I disse tilfælde vil tabellen ikke være på 1NF under denne definition. Se nærmere diskussion i Appendix omkring problemerne med definitionen af en normaliseret virkelighed på 1. normal form.

Spørgsmål: Er ovenstående tabel på 1NF? - svar: NEJ

Første produktnr felt indeholder repeterende værdier (eks: 1, 2 og 3). Det gør første antalfelt også. Ligeledes er der ikke fast rekordlængde, da rekord 3 istedet for at have repeterende værdi i produktnr og antal felterne har gentaget værdierne i 2 ekstra felter, hvorved rekord 3 er længere end de andre.
I ovenstående tabel kan man se at en attribut som produktnr indeholder 3 numre (repeterende gruppe), og antal indeholder ligeledes 3 værdier svarende til de 3 produktnumre. En sådan tabel er ikke normaliseret, da hver kolonne skal være *atomarisk* (der må være en, og kun en værdi i en hver rekords enkelte felter).
En anden måde at angive repeterende gruppe på, et at gentage feltet flere gange (i ovenstående tabel er produktnr og antal felterne angivet 2 gange, og dermed udtryk for repeterende værdier i rekord 3). Med atomarisk menes derfor også, at et felt ikke dubleres.

Med domæne menes alle de værdier som et givet felt kan/må bestå af.

Med atomarisk menes altså i praksis: Et felt indeholder en og kun en værdi af det givne domæne, og et givent felt er angivet en og kun en gang i hver rekord/tabel. Et givet felt må kun indeholde en betydning (domæneværdierne har kun et betydningsindhold for brugen af den givne rekord).

For at få ovenstående tabel på første normalform, bliver man derfor nødt til at gentage rækkerne, således at kolonnerne kun indeholder 1 værdi, og man fjerner (felt-dubletterne) – produkt-nr. og antal kolonnerne.

Første normal form.

Nu kommer tabellen derfor til at se således ud:

Lev-nr	Postnr	By	Produktnr	Antal
1	4000	Roskilde	1	100
1	4000	Roskilde	2	150
1	4000	Roskilde	3	125
2	8800	Viborg	1	100
2	8800	Viborg	2	100
2	8800	Viborg	3	250
3	4000	Roskilde	1	50
3	4000	Roskilde	2	100
4	9000	Ålborg	1	100
4	9000	Ålborg	2	125
4	9000	Ålborg	3	175

Primærnøgle: Lev-nr+Produktnr

2NF *Tabellen skal være på 1NF. Alle felter der ikke indgår i primærnøglen skal være fuldt funktionelt afhængige af primærnøglen.*

Spørgsmål: Er ovenstående tabel på 2NF? - svar: NEJ

Med "fuldt funktionelt afhængig" menes, at et felt der ikke indgår i primærnøglen skal være afhængig af ***hele*** nøglen og ikke blot en delmængde af nøglen. I ovenstående tabel består primærnøglen af 2 felter (lev-nr og produktnr), men postnr og by felterne er kun afhængige af lev-nr feltet (de er ikke afhængige af produktnr-feltet), og disse 2 felter indgår ikke i primærnøglen. Disse 2 felter er altså ikke fuldt funktionelt afhængige - de er kun delvist afhængige!

Vi bliver derfor nødt til at splitte ovenstående tabel op i 2 tabeller. En tabel der indeholder oplysninger om leverandøren, og de felter der knytter sig til lev-nr (postnr og by), og en anden tabel der indeholder oplysninger om produkterne. Tabellerne skal kunne knyttes sammen (i en 1-til-mange relation), så vi må også angive lev-nr feltet i produkttabellen.

Bem: En tabel der er på første normalform, og hvor primærnøglen kun består af et felt, vil derfor også altid være på anden normalform!

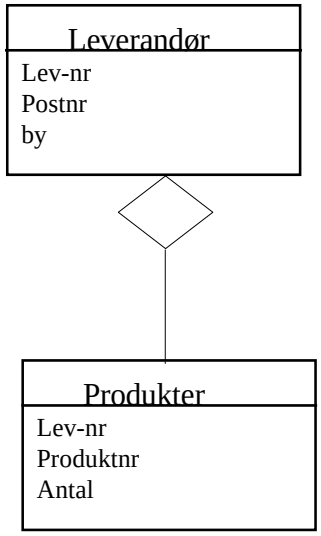
Anden normalform.

Leverandørtabel

Lev-nr	Postnr	By
1	4000	Roskilde
2	8800	Viborg
3	4000	Roskilde
4	9000	Ålborg

Produkttabel

Lev-nr	Produktnr	Antal
1	1	100
1	2	150
1	3	125
2	1	100
2	2	100
2	3	250
3	1	50
3	2	100
4	1	100
4	2	125
4	3	175



Primærnøgle Leverandørtabel: Lev-nr
Primærnøgle Produkttabel: Lev-nr+Produktnr

For at knytte ovenstående tabeller sammen skal der laves en fremmednøgle på produkttabellens lev-nr, der knytter sig til leverandørtabellen (pilen anigvet mellem tabellerne).

I ovenstående tabeller er felter der ikke indgår i primærnøglerne fuldt funktionelt afhængige af primærnøglerne.

Bemærk, at tabellen på første normalform godt kunne opfattes som en klasse, men at vi via normaliseringsteorien får fundet frem til at det faktisk er 2 logiske klasser.

3NF *Tabellen skal være på 2NF. Alle felter der ikke indgår i primærnøglen må ikke være transitivt afhængige af primærnøglen.*

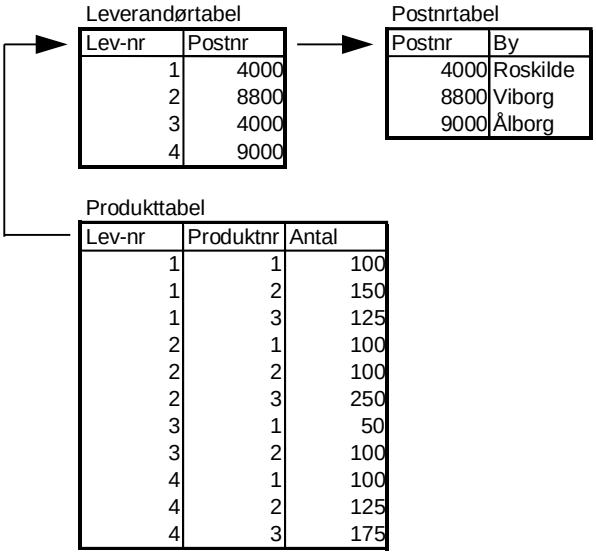
Spørgsmål: Er ovenstående tabel på 3NF? - svar: NEJ

Med 'ikke være transitivt afhængige' menes, at felter der ikke indgår i primærnøglen ikke må have nogen form for indbyrdes afhængighed eller relationer. Eksempelvis indeholder leverandørtabellen 2 felter der ikke indgår i primærnøglen (postnr og by). Både By og postnrfelterne er fuldt afhængige/tilknyttede til lev-nr feltet, men der er samtidig en sammenhæng mellem bynavn og postnr. Der er altså en form for tilknytning mellem 2 felter der ikke indgår i primærnøglen! Vi kan altså gå fra leverandørfeltet til by og derudfra udlede postnr.

Vi bliver derfor nødt til at splitte leverandør tabellen op i 2 tabeller, således at postnr og by felterne ikke begge knytter sig til lev-nr feltet.

Tredje normalform / Boyce-Codd Normalform

3. Normalform



Leverandørtabel

Lev-nr	Postnr
1	4000
2	8800
3	4000
4	9000

Postnrtabel

Postnr	By
4000	Roskilde
8800	Viborg
9000	Ålborg

Produkttabel

Lev-nr	Produktnr	Antal
1	1	100
1	2	150
1	3	125
2	1	100
2	2	100
2	3	250
3	1	50
3	2	100
4	1	100
4	2	125
4	3	175

Primærnøgle Leverandørtabel: Lev-nr.
Primærnøgle Postnrtabel: Postnr
Primærnøgle Produkttabel: Lev-nr+Produktnr

For at knytte de 3 tabeller sammen skal man som på 2NF have en fremmednøgle i produkttabellens lev-nr der refererer til leverandørtabellen. Ligeledes skal man have en fremmednøgle på leverandørtabellens postnr felt der refererer til postnrtabellen (pile med tabeller).

Som man kan se af ovenstående tabeller er der nu ikke længere felter der ikke indgår i primærnøglerne som er indbyrdes afhængige.

I ovenstående tabeller er der nu kun et felt i hver tabel der ikke indgår i primærnøglen, og derfor kan der heller ikke være et afhængighedsforhold mellem felter der ikke indgår i primærnøglen:

Tabel	Primærnøgle	Felter ikke i primærnøgle
Leverandørtabel	Lev-nr	Postnr
Postnrtabel	Postnr	By
Produkttabel	Levnr+Produktnr	Antal

Bemærk at tabellerne på 2. normalform kunne opfattes som 2 klasser, men at vi via normaliserings-teorien får analyseret os frem til at der faktisk er 3 logiske klasser!

Boyce-Codd Normalform (BCNF)

BCNF omhandler tabeller, hvor man kan forestille sig attributter en eller flere andre attributter er fuldt afhængige af (determinant), hvor denne attribut (eller sammensatte attribut) samtidig skal kunne være primærnøgle (alle determinanter skal være kandidatnøgler). En sådan tabel er på 3 normal form og også på BCNF. Er tabellen på 3. normal form, men der kan findes en eller flere af ovenstående attributter (også kaldet determinanter) som ikke samtidig kan være primærnøgle for hele tabellen er tabellen ikke på Boyce/Codd normal form, men kan opdeles i 2 tabeller. Hvis en determinant er en del af en primærnøgle, og denne determinant ikke kan være en primærnøgle i sig selv, er tabellen ikke på BCNF. Man skal altså teste alle felter i tabellen for om de er determinanter, og om alle determinanter kan være primærnøgler.
BCNF er en skærpelse af 3. normalform.

BCNF En derterminant er et felt (eller et sammensat felt af flere felter) som en eller flere andre felter er fuldt afhængige af. En tabel er på BCNF hvis alle determinanter er kandidatnøgler (kan være primærnøgle) for tabellen.

Hvis vi ser på produkttabellens felter for at lede efter determinanter kan det ses at på enkeltfelt niveau er der ingen determinanter. Der er ingen afhængighed mellem produktnummer og leverandørnummer, og der er ingen afhængighed mellem disse felter individuelt og antal. Til gengæld er der en afhængighed mellem antal og den sammensatte determinant bestående af leverandørnummer og produktnummer, men der er ikke andre determinanter. Da dette er tilfældet, er tabellen også på Boyce-Codd's normalform.

Generelt om Fjerde og Femte og øvrige normalformer

Dette er ofte spekulative og mere teoretiske normaliseringsformer som tit er teoretisk konstruerede end egentlige tabeller man ofte støder på i virkeligheden. Ofte er dette tabeller hvor eksemplet angives til at have sammensatte primærnøgler bestående af 2-3 felter med repeterende værdier "inde i nøglen" hvorfor disse til tider kan normaliseres yderligere for at fjerne redundansen. Hvis en tabel derfor er normaliseret til et niveau hvor nøglen kun består af 1 (et) felt, og der ikke er andre kandidatnøgler (andre mulige nøglefelter) vil tabellen ikke kunne normaliseres yderligere.

Er en tabel bestående af en sammensat primærnøgle skal man altså undersøge om der er 'repeterende værdier' i primærnøglen hvilket ofte er tilfældet da man netop er nødt til at sammensætte 2 (eller flere) felter for unikt at kunne identificere en given rekord, og derved er minimum 1 af felterne i primærnøglen ikke unik men har flere værdier der går igen. Hvis det er tilfældet kan tabellen til tider splittes op i 2 eller flere nye tabeller indtil man har normaliseret ned til et niveau hvor tabellen kun har et felt der udgør primærnøglen, og hvor der ikke er andre kandidatnøgler som primærnøgle. Alternativt ned til et niveau hvor tabellens felter tilsammen udgør primærnøglen og hvor disse felter ikke kan opdeles i flere tabeller der tilsammen kan skabe informationen fra denne tabel.

Fjerde normalform (4NF)

4NF er en nedbrydning af en tabel, når der eksisterer 2 felter som er uafhængige af hinanden, men begge er afhængige af et andet felt og kan kombineres (multipliceres af deres respektive værdier). Dette kaldes også på dansk for multiværdi afhængighed. Kravet er altså, at der ikke må eksistere felter hvor værdierne i disse domæneer/felter skal multipliceres for at give alle udfald: eksempelvis en tabel med 3 felter:

Bilmodel, Motortype (benzin, diesel), farve. Hvis der er 2 bilmodeller (sedan og Stationcar) 2 motortyper (Benzin og Diesel) og 3 farver og alle modeller kan leveres i alle farver er der altså 12 kombinationer og dermed 12 rekords under de givne domæne definitioner.Vi har derved en multiværdi afhængighed hvor en given kombination af motortype og farve vil kunne byttes om mellem rekords og give samme udfald for alle kombinationer. Det er også klart, at denne tabel indeholder redundant information. Det er også klart, at der ikke eksisterer ”fuld afhængighed” mellem nogle af disse felter. Farven er ikke bestemt af hverken bilmodel eller motortype, motortype er ikke bestemt af hverken bilmodel eller farve, og bilmodellen er ikke bestemt af hverken motortype eller farve. Altså er der ingen kombination af 2 felter der er determinanter under BCNF. Omvendt er der et klart defineret værdisæt af motortyper og farver der kan vælges imellem til hver bilmodel.

Bilmodel	Motortype	Farve
Sedan	Benzin	Rød
Sedan	Benzin	Gul
Sedan	Benzin	Grøn
Sedan	Diesel	Rød
Sedan	Diesel	Gul
Sedan	Diesel	Grøn
Stationcar	Benzin	Rød
Stationcar	Benzin	Gul
Stationcar	Benzin	Grøn
Stationcar	Diesel	Rød
Stationcar	Diesel	Gul
Stationcar	Diesel	Grøn

I en sådan tabel ville alle 3 felter skulle sammensættes for at udgøre primærnøglen. Da der ikke eksistererer andre determinanter end primærnøglen vil tabellen derfor være på BCNF form. Ligeledes vil den kunne splittes op i 2 tabeller (Bilmodel, Farve og Bilmodel, Motortype), hvor vi har fjernet redundant information, således at i tabellen bil/motortype er der 4 rekords, og i tabellen bil/farve er der 6 rekords. I hver tabel er begge felter stadig en del af primærnøglen, og der er stadig repeterende værdier i primærnøglen, men tabellerne kan ikke nedbrydes yderligere uden tab af information.

Bilmodel	Farve
Sedan	Rød
Sedan	Gul
Sedan	Grøn
Stationcar	Rød
Stationcar	Gul
Stationcar	Grøn

Bilmodel	Motortype
Sedan	Benzin
Sedan	Diesel
Stationcar	Benzin
Stationcar	Diesel

4NF En tabel er på 4NF hvis - når der eksisterer en flerværdi afhængighed fra eksempelvis felt A - alle felter i en given rekord være funktionelt afhængige af A.

Hermed menes, at består en tabel eksempelvis som benævnt ovenfor af 3 felter a,b, og c, hvor b og c hver har en givet sæt af kombinærbare værdier i forhold til a, skal enhver kobination af b,c unikt kunne identificeres af a. Dette er ikke tilfældet i det nævnte eksempel da b og c kan kombineres på forskellig vis til samme a værdi. Splitter man derimod tabllen op i 2 tabller a,b og a,c som vist ovenfor af mulige kombinationer, vil der kun være en farve til hver biltype, og en motortype til hver biltype. Der vil da ikke eksistere en multiværdi afhængighed, og da der ikke er nogle funktionelle determinanter, er disse 2 tabller altså både på BCNF og 4NF.

Femte normalform (5NF)

Hvis vi tager foregående eksempel under 4. normal formog udskifter farve med sælger, kan man forestille sig følgende tabel:

Bilmodel	Motortype	Sælger
Sedan	Benzin	Hansen
Sedan	Diesel	Hansen
Sedan	Diesel	Jensen
Sedan	Diesel	Olsen
Stationcar	Benzin	Hansen
Stationcar	Diesel	Hansen
Stationcar	Diesel	Jensen

En given sælger kan sælge en given bilmodel med en given motortype under nogle givne forudsætninger. I ovenstående er der den forudsætning, at hvis en given sælger kan sælge en motortype gælder det for alle bilmodeller.
Det er ikke sikkert, at en sælger er 'tilladt' at sælge alle kombinationer, men omvendt kan en sælger måske godt have 'tilladelse' til at sælge alt. Herved er der ikke en afledt afhængighed mellem motortype og bilmodel. Ej heller afhængighed mellem hverken sælger og motortype eller sælger og bilmodel. Alene alle 3 felter udgør primærnøglen for tabellen, og der er ikke andre determinanter, så tabellen er på BCNF. Ligeledes kan motortype og sælger ikke per automatik kombineres, så der eksisterer ikke en flerværdi afhængighed, så tabellen kan også siges at være på 4. normal form.

5NF En tabel er på 5. normalform hvis alle join-afhængigheder i tabellen er en konsekvens af tabellens kandidatnøgler.

I en tabel hvor felterne x1,x2,x3,...,xN er en delmængde af tabellens felter siges join-afhængigheden at eksistere for denne delmængde af felter såfremt enhver forekomst af rækker i tabellen kan dannes ud fra projektionerne af tabellen på delmængderne af x1,x2,x3,...,xN.

Ovenstående tabel er derfor ikke på 5. normalform. Tabellen kan opsplittes i 3 tabeller således at en projektion af disse 3 tabeller kan skabe ovenstående informationer. Se tabellerne nedenfor.

Bilmodel	Sælger
Sedan	Hansen
Sedan	Jensen
Sedan	Olsen
Stationcar	Hansen
Stationcar	Jensen

Motortype	Sælger
Benzin	Hansen
Diesel	Hansen
Diesel	Jensen
Diesel	Olsen

Bilmodel	Motortype
Sedan	Benzin
Sedan	Diesel
Stationcar	Benzin
Stationcar	Diesel

I ovenstående tabeller har man angivet hvilken sælger der kan sælge hvilke bilmodeller. Ligeledes hvilken sælger der kan sælge hvilke motortyper, og sidst hvilke bilmodeller der har hvilke motortyper i udgangstabellen.
Eksempelvis kan man ved at sammenkøre (joine) informationeerne fra disse 3 tabeller kunne komme tilbage til udgangspunktet i den første tabel. Det fremgår eksempelvis af ovenstående, at Sælger Olsen alene kan sælge bilmodellen Sedan med en Dieselmotor. Ligeledes fremgår det, at Sælger Hansen kan sælge både Sedan og Stationcar og begge både med Benzin og Dieselmotor (da det fremgår af sidste tabel at begge bilmodeller sælges med begge motortyper. Ligeledes kan det ses af ovenstående, at kun sælger Hansen kan sælge bilmodeller med benzin motorer.

Informationerne i ovenstående tabeller kan ikke opsplittes i yderligere tabeller uden tab af information. Hver af ovenstående 3 tabeller har begge felter som primærnøgle, og i disse er der repeterende værdier, men i dette tilfælde vil man ikke kunne opsplitte informationen yderligere.

Bemærk ligeledes, at hvis man ændrer præmisserne for den virkelighed der skal beskrives således at man kunne forestille sig, at en sælger eksempelvis godt kunne sælge en given bilmodel med en given motortype, men ikke en anden bilmodel med samme motortype ville man ikke kunne foretage denne projektion uden tab af information. Lad os antage, at sælger Jensen i den oprindelige tabel dermed stod til også at kunne sælge bilmodel Sedan med Benzin motor, men ikke var 'tilladt' at kunne sælge Stationcar med benzinmotor ville projektionerne i de 3 tabeller ikke længere være gældende. Vi ville ikke kunnne repræsentere den givne viden i de 3 underliggende tabeller. Sælger Jensen ville stå til at kunne sælge både Benzin og Diesel i tabellen (motortype, sælger), og ligeledes vil han stå til at kunne sælge både Sedan og Stationcar i tabellen (bilmodel, sælger), men informationen om at han kun må sælge benzinmotor udgaven af Sedan modellen og ikke i stationcar udgaven vil dermed være tabt. Dermed vil man ikke kunne gendanne den nødvendige information via projektionerne fra de underliggende tabeller.

Under 4. normal forms eksempel opsplittede man en tabel med 3 felter (a,b,c) i 2 tabeller med to felter på formen a,b og a,c der tilsammen indeholdt samme information som den tabel hvorfra disse blev udledt. Dette skyldes multiværdi afhængighed.
Under 5. normal forms eksempel opsplittede man en tabel med 3 felter (a,b,c) i 3 tabeller med to felter på formen a,b og a,c og b,c (alle kombinationer af attributter nødvendige) for at kunne genskabe informationen fra den tabel hvor informationen kom fra.

Andre normalformer

I litteraturen (Se bl.a. på engelsk beskrivelse om 'Database Normalization' på www.wikipedia.org) kan der bl.a. læses om Domain/Key normal form og Sjette normalform.

Domain/Key er ikke at betragte som en 6. normal form i sin puristiske definition, men er mere en teoretisk anderledes begrebsanalyse til at betragte tabeller på ud fra en normaliserings synspunkt. Der er dog ikke angivet en klar og entydig metodik til at benytte DK/NF til at få alle tabeller normaliseret i bund.

Domain/Key eksemplet på ovenstående link kan dog diskuteres om overhovedet er på første normalform, da eksemplet indeholder fler-betydnings information i samme attribut med repeterende værdier i domænet, og dermed ikke er blevet normaliseret på felt niveau, men der henvises herved til appendix for en nærmere diskussion af definitions problematikken med 1. normalform.

Fordele og ulemper ved tabel normalisering.

Fordelen ved tabel normalisering ligger på flere områder:

- Vedligeholdelse lettes (i form af indsættelse af nye data, opdatering og sletning) (i de fleste tilfælde)
- Fleksibilitet og overblik (kan også give mindre overblik)
- Pladsbesparelse (fjernelse af redundante data).

Vedligeholdelse, fleksibilitet og overblik.

Har man kun data liggende et sted, er det betydeligt lettere at vedligeholde (og det sparer oftest plads).

Eksempelvis hvis vi ved at ovenstående 3 tabeller i beskrivelsen af 3. normalform der er på 3. normalform er forbundet med fremmednøgler ville man blot kunne ændre eksempelvis leverandørnummer 1 til nummer 10, og være sikre (under forudsætning af at man arbejder med fuld integritet - altså med CASCADE UPDATES og CASCADE DELETES) på at alle data er opdaterede i alle tabeller.

Da data ligeledes er placeret i logisk sammenhængende tabeller (logiske klasser), har man også større overblik over hvor de enkelte oplysninger findes, og evt. ny funktionalitetsbehov for programmet er ikke så afhængig af den fysiske lagringsstruktur.

Pladsbesparelse.

Den anden fordel ved normalisering er pladsbesparelse. Når man undgår repeterende værdier fylder data ikke så meget, og derved vil man spare plads i databasen og harddisken (som udgangspunkt).

Umiddelbart kan man ikke se dette ud fra ovenstående eksempler, når tabellen indeholder så få rekords, men i visse transaktionssystemer tales der sommetider om millioner af transaktioner per dag eller per time, og da er en enkelt byte af betydning for databasens størrelse.

Ulemper ved normalisering.

Der er også ulemper ved normalisering. Hvor godt det end lyder med lettere vedligeholdelse større fleksibilitet, større overblik og pladsbesparelse, kan uigennemtænkt normalisering faktisk risikere at være det modsatte!

Dette skyldes, at hvis man blot normaliserer uden at gennemtænke hvorfor man normaliserer, kan man komme til at få flere ulemper end fordele. Mulige ulemper kan eksempelvis være:

- Mange tabeller betyder lavere performance (access af flere tabeller).
    - Den faktiske brug af systemet kan altså være af betydning for normaliseringsgraden

- Mange relationer med fremmednøgler, betyder mange indeks, hvilket igen betyder (se afsnit 2.3):
    - Forøget plads til indeks
    - Forøget procestid til opdatering af indeks

- Mange tabeller kan betyde mindre overblik:
    - Hvor er et givet felt placeret?
    - Hvorledes hænger data egentlig logisk sammen?

Man skal derfor ikke ukritisk normalisere.

Afvejede behov.

Desværre kan man ofte læse diverse kommentarer, at ikke fuldt tabel normaliserede databaser er udtryk for 'dårligt database design'. Dette er ikke nødvendigvis altid tilfældet. Afgørende er hvad databasen skal bruges til, og hvilke slutbrugerbehov der skal løses.

Man må altså afveje en lang række krav og behov til det edb-system man vil udvikle og hvor databasen skal benyttes og afveje fordelene ved tabel normalisering op mod ulemperne.

Database normalisering

Generalisering af tabeller

Et punkt som ikke får mange bemærkninger eller fokus i undervisning omkring database modellering i forhold til eksempelvis tabel normalisering er begrebet generalisering (og specialisering). Disse begreber kendes primært fra den objekt-orienterede teori, men gælder i høj grad også for databaser. Emnet har været beskrevet siden 1970'erne, men som sagt fylder dette emne langt mindre i database teoribøger end tabel normaliserings teorien gør.

Mange databaser består af ikke-generaliserede tabeller.

Tabel normalisering foregår altid på enkelt tabel niveau. En tabel kan blive til flere tabeller ved at redundante data i rekords bliver fjernet ved at tabellen bliver splittet op således at data kun indgår en gang.

Ved generalisering ser man ikke på den enkelte tabel, men ser på om felter (domæne data) er redundante på tværs af tabeller. Altså om de samme felter (samme logiske indhold) indgår i flere tabeller. Hvis dette er tilfældet er det muligt at overveje en generalisering. Her kommer et eksempel med en kunde og en leverandør tabel:

KUNDE TABEL

Kundenr	Kundenavn	Adresse1	Adresse2	Postnr	Land	Tlf.nr	kreditmax	ABC-kode	…	Rabat%
1	Eriksen A/S	Husvej 2	Lyssinge	4123	Danmark	12345678	45000,00	B		10
2	Hansen Imp.	Bådvej 23		4567	Danmark	34567890	25000,00	C		3
3	Trafico	Pigvej 56		7643	Danmark	56124598	100000,00	A		15

LEVERANDØR TABEL

Lev.nr	Lev.navn	Adresse1	Adresse2	Postnr	Land	Tlf.nr	Kretitmax	Min.køb	…	Lev.Tid
1	Larsen vin	Rossevej 11		8800	Danmark	5327890	10000,00	300,00		5

Ser man på de 2 tabeller indeholder disse en række felter som er helt identiske. Disse 2 tabeller kan faktisk generaliseres således at de felter som er ens fjernes fra de 2 tabeller og lægges over i en ny tabel. Der skal dog så samtidig oprettes en hjælpetabel som beskriver om det er en kunde eller en leverandør der er i den generaliserede tabel:

TYPE TABEL

Type	Typetekst
K	Kunde
L	Leverandør

ADRESSE TABEL

Type	Nummer	Navn	Adresse1	Adresse2	Postnr	Land	Tlf.nr	Kretitmax
K	1	Eriksen A/S	Husvej 2	Lyssinge	4123	Danmark	12345678	45000,00
K	2	Hansen Imp.	Bådvej 23		4567	Danmark	34567890	25000,00
K	3	Trafico	Pigvej 56		7643	Danmark	56124598	100000,00
L	4	Larsen vin	Rossevej 11		8800	Danmark	5327890	10000,00

LEVERANDØR TABEL

Nummer	Min.køb	…	Lev.Tid
4	300,00		5

KUNDE TABEL

Nummer	ABC-kode	…	Rabat%
1	B		10
2	C		3
3	A		15

I objekt-orienteret tankegang er det udtryk for at adresse tabellen er en generaliseret klasse som leverandør- og kundetabellerne ”arver” fra, og disse indeholder så kun de specialiserede felter som er unikke for deres ’klasse’.

Ovenstående eksempel er faktisk en ret simplificeret generalisering, da der kan generaliseres så klart, at navn og type IKKE indgår i adresse-tabellen, men alene et adresse-nummer, og så selve adresseinformationerne. Derved vil man kunne benytte tabellen således at hvis en kunde samtidig er leverandør og eksisterer både i kundetabellen og leverandør-tabellen henvises alene til et adresse nummer, som i dette tilfælde er det samme.

Dette kan ses i nedenstående eksempel:

Kunder

Kundenr	Kundenavn	AdresseNr	Kreditkode	ABC kode	Rabat%
1	Eriksen A/S	100	1000	B	10
2	Hanen Imp	101	1001	C	3
3	Trafico	102	1002	A	15

Leverandør

Lev. Nr	Lev Navn	AdresseNr	Kreditkode	Tlf nr	Min køb	Lev tid
1	Larsen vin	103	1003	5327890	300	5

Adresser

Adressenr	Vej	Nr	Adresse2	Postnr
100	Husvej	2	Lyssinge	4123
101	Bådvej	23		4567
102	Pigvej	56		7643
103	Rossevej	11		8800

Postnr

Postnr	By	Land
4123	Rosenby	Danmark
4567	RosOverby	Danmark
7643	NedreUnder	Danmark
8800	Aarhus	Danmark

Kredit info

Kreditkode	Beløb
1000	45000
1001	25000
1002	1000000
1003	10000

I dette nye eksempel er Adresser og Kredit information generaliseret således at data i de to tabeller for kunder og leverandører alene refererer til en adresse kode og det samme gælder for kredit information. I dette tilfælde vil en kunde der også er leverandør blot kunne have den samme

adressekode, og en kunde og/eller leverandør med samme kreditmaksimum vil kunne have samme kreditkode.

Dermed skal man se generalisering i sin yderste konsekvens således at data alene skal opdeles i sine logisk sammenhængende attributter. Vi burde altså også undlade Telefonnummer, da disse i visse tilfælde også kan deles på tværs af tabeller. For alle attributter i databasen hvor der er mulig deling mellem tabeller er der altså også en mulig generalisering.

Fordele og ulemper ved database generalisering

På samme måde som ved normalisering er der fordele og ulemper ved denne metode.

Fordelen er klart, at ovenstående kan benyttes på alle tabeller der indeholder adresser (kunne være medarbejder tabel, kontakt tabel, etc) og man kan blot tilføje adresser som man har behov for. Altså sikring af, at ensartede informationer på tværs af tabeller ikke skal vedligeholdes i flere tabeller samtidigt, og at den samme type af information kun placeres et sted i databasen, og ikke repeteres igen og igen i forskellige tabeller.

Ulempen er helt som ved normalisering, at der skal laves flere fremmednøgler, og attributter/felter skal køres sammen fra flere tabeller når som i eksemplet en kunde eller leverandør skal vises med alle sine attributter.

Igen må database normalisering i form af tabel generalisering afvejes mellem fordelen ved at gennemføre en sådan normalisering kontra de ulemper der er ved det.

Appendix: Definitionen af 1. normalform

C.J. Date skriver i sin bog: "An introduction to database systems – Third edition", 1981 i definitionen på første normalform på side 243: "A relation R is in *first normal form* (1NF) if and only if all underlying domains contain atomic values only.". I denne definition bygger Date sin opfattelse på arbejde udført af E.F. Codd, og Date citerer bl.a. til dette i nedenstående nævnte skrift til Codd's definition omkring atomariske værdier (Codd, The Relational Model for Database management Version 2, Addison-Wesley, 1990)

I et noget senere værk beskriver C.J. Date ikke længere 1NF som atomarisk, men alene at en given attribut skal have en og kun en værdi af et givent domæne. Se C.J. Date's definition på 1. normal form nedenfor, hvor det nærmeste han kommer på problemstillingen er punkt 4.

> 1. There's no top-to-bottom ordering to the rows.
> 2. There's no left-to-right ordering to the columns.
> 3. There are no duplicate rows.
> 4. Every row-and-column intersection contains exactly one value from the applicable domain (and nothing else).
> 5. All columns are regular [i.e. rows have no hidden components such as row IDs, object IDs, or hidden timestamps].
> —Chris Date, "What First Normal Form Really Means", pp. 127-8

Med denne definition i punkt 4 må man eksempelvis godt have værdier med forskellige menings typer i samme felt (eksempelvis Stor og Rød, Mellem og Grøn, Lille og Rød) så længe disse kombinationer er defineret som værende mulige værdier i det givne felt (mulige valg fra det givne domæne). Det er klart, at dette medfører, repeterende værdier inde i det enkelte felt, som beskrevet under felt normalisering. I normaliseringsdefinitionerne fra 1-6 normalform taler man alene om redundante data mellem rekords (repeterende data i rekords i tabeller), men ikke om redundante data i felter.

Det 'filosofiske' dilemma for begge definitioner

Det næsten filosofiske problem med Codd's atomariske definition er tydelig. Hvorledes ved man hvornår en værdi er endegyldig og ikke kan nedbrydes yderligere?

Et givet tekstfelt kan bestå af flere bogstaver (sammensat til meningsfyldte ord eller symboler) som kan have forskellig betydning afhængig af begrebsramme og kontekst. Et datofelt kan typisk splittes op i dag-måned-år, og et felt der indeholder et tidspunkt kan have time, minut, sekund og et utal af decimaler der angiver en nærmere præcision afhængigt afdatabasens feltmuligheder.

Hvad der på et tidspunkt i forhold til en applikation er atomarisk kan for en anden applikation være en sammensat information, som i en anden kontekst enten kan opfattes som værende repeterende i denne anden konteksts domæne, eller kan opfattes som værende information der kan splittes i 2 domæner (farve og størrelse) eller tidspunkt opdelt i time og minut i hvert sit felt etc.

Som Date også argumenterer for i tidligere nævnte skrift, er der klare problemer med Codd's definition.

Her kommer Date's definition punkt 4. ovenfor desværre også i problemer og faktisk på samme måde. Hvad forstår man ved 'applicable domain', og hvorledes ved man hvad domænets værdier er på forhånd? Hvis man ikke har en fuldgyldig definition af alle feltets/domænets mulige udfald, hvordan kan man så sørge for at feltet kun indeholder 'exactly one value'.

Er alle mulige værdier for et givet felt defineret entydigt (i så fald vil man kunne diskutere om ikke tabellen er på DK/NF)?

Hvordan sikrer man sig som angivet ovenfor, at når man designer en tabel efter Date's definition, at alle værdier der indtastes, kun medtager en og kun en værdi fra det mulige domæne?

Tanken bag Date's defintion er vel som ved Codd's også rimelig klar. Hvis man har en attribut der hedder bynavn, eller et telefonnummerfelt må førstnævnte altså kun indeholde et (og kun et navn per rekord) navne på definerede byer, og et telefonnummer må indeholde et nummer - på en given telefon. I sig selv logisk, men selv her kan man komme ud i diskussioner om hvad domænet indeholder. Er et telefonnummer med landekode, og i givet fald er det da med for Danmark 0045 foran et 8 cifret tal eller er det +45. Man kan sikkert definere feltet som det fulde nummer der skal tastes alene med cifrene 0-9 uanset fra hvor i verden man befinder sig for at få forbindelse til det givne nummer. Problemet opstår dog, hvis man har mindre klart definerbare attributter. Hvad hvis et felt er et 100 ciffer kommentar eller notat tekstfelt. Hvordan sikrer man sig 'en og kun en værdi' i et sådant domæne. Man kan argumentere for, at feltet er defineret som en 'samling af ord af op til totalt 100 karakteres længde', og alle tekststrenge der overholder dette krav dermed også udgør en en og kun en værdi af det mulige udfald. Men dette giver dog ingen mening, da dette blot er en definition på attributtens felttype og ikke dets domæne. At noget er et tekstfelt af en given længde, er ikke det samme som at definere hvad dette tekstfelts domæne er.
Hvordan sikrer man sig, at det der kaldes semantisk integritet, er til stede på alle felter på alle tabeller?

Date's definition holder kun så længe, at man på forhånd kan beskrive en fuldstændig og entydig definition på hver attributs mulige domæne, og dette vil sikkert være muligt for nogle tabeller, men i en praktisk verden også umulig for andre tabeller, men som de fleste alligevel vil opfatte som værende på 1. normalform selvom man ikke på forhånd kan garantere, at de overholder Date's definition punkt 4.

Altså er dette i praksis en umulighed i sin teoretiske form, da vi ender i samme diskussion omkring kontekst og applikationsbegrebsramme som ovenfor beskrevet, med mindre man som sagt definerer klart og entydigt alle domæneværdier for en given tabels felter, eller indlægger regler der tjekker for attributtens indhold for alle definerede udtømmelige muligheder i databasen.

For begge definitioner har vi altså problemet med på forhånd altid at kunne definere hvad en atomarisk værdi eller 'en og kun en værdi af domænet' er.

Den praktiske håndtering

Det virker som om, at det både Codd (atomic) og Date (exactly one value) forsøger at få beskrevet er:

"I selve designet af tabellen, skal den normaliserede tabel (1NF) have en opdeling og repræsentation af data således, at det 'efter bedste evne og viden på designtidspunktet' er udtryk for, at hvert felt/attribut i hver af tabllens rekords/tupler skal have en enkelt (en og kun en) og entydig værdi i forhold til den attribut fra den 'virkelige verden som tabellen/relationen skal repræsentere'".

Altså som en sammenkogning fra begge teoretikere, at sige at der skal foretages feltnormalisering (Codd's atomariske tilgang, således at der ikke kommer repeterende værdier i domænet), og derefter må hvert felt i alle rekords kun indeholde en enkelt værdi af domænet.

Det som ovenstående forsøger, at beskrive er ligeledes, at man ikke kan designe en tabel bedre end den viden som den modelansvarlige har om nutiden og kendte forventninger om fremtiden af den givne virkelighed som tabellen skal repræsentere på designtidspunktet. Eller sagt igen på en lidt anden måde: Tabeller skal designes således at felter gøres så 'atomariske som muligt' inden for den kendte virkelighed (kendskab til nuværende og forventede fremtidige domæneindhold).

Problemet er som sagt universelt da ingen på designtidspunktet kan forudsige om den givne information er unik (atomarisk eller 'indeholder en og kun en værdi') i al evighed for alle tænkelige attributter. I nogle tilfælde er domænet kendt og afgrænset, og da er det simpelt at sikre at værdierne er på laveste 'atom' niveau og, at der kun er 'en' værdi, men i andre tilfælde er domænet umuligt at kende (alle mulige udfald), og da kan man ikke være sikker på hverken 'atomarisk', 'en og kun en værdi' eller at der ikke er repeterende værdier i domænet (semantisk felt analyse).
Data og information er blot symboler som kan tillægges forskelligt indhold over tid, og som til tider kan opdeles/splittes op i mere information som tidligere beskrevet.

Der vil være felter/attributter hvor man ***ikke*** har en fuldstændig viden om domænets totale mængde, men hvor man alligevel i praksis vil sige, at tabellen er på første normal form. Tiden kan evt. medføre, at en given tabel kan normaliseres yderligere når flere informationer kommer til, og man får større viden om domænets mulige værdier.