Normalisering

Igennem en årrække er grundprincipperne bag relationelle databaser blevet præsenteret på en ret teoretisk måde – som regel med begreber fra mængdelæren og diverse matematisk inspirerede regelsætninger. I praksis er de forskellige begreber inden for relationel databaseteori imidlertid ikke specielt vanskelige at forstå. I mange tilfælde er det almindelig sund fornuft, der blot præsenteres på en teoretisk måde.

Dette gør sig i allerhøjeste grad gældende for begrebet normalisering, der i mange fremstillinger næsten ikke er til at forstå. Som vi skal se, er det primært et spørgsmål om en grundig analyse af sammenhænge mellem de data, man har brug for at registrere. Resten er enten almindelig sund fornuft eller stringent logik.

Det er en fordel at forstå, hvorfor en database bør normaliseres, og at vide lidt om, hvordan man kan gribe processen an. Hovedprincippet bag normaliseringen er ganske simpelt: Det drejer sig så vidt muligt om at fjerne alle former for redundans med det formål at gøre databasen effektiv, konsistent og let at vedligeholde.

Normalisering består i at tage en kompleks datastruktur, der typisk tidligere ville være blevet lagret i en post i én tabel, og bryde den ned i dens grundelementer, således at vi ender med mange tabeller med simple datarelationer.

I teorien bør man altid normalisere sine data så langt som muligt. I praksis er man naturligvis nødt til at overveje, hvordan normalisering påvirker databasens hastighed og den kompleksitet, eksempelvis hyppige forespørgsler ender med at få. Det tager generelt længere tid at forbinde data fra forskellige tabeller end at hente dem fra en. Så hvis man finder ud af, at man hyppigt har brug for en bestemt, meget kompleks SQL-forespørgsel, så kan man af praktiske hensyn vælge at give køb på normalisering til fordel for hastighed.

Den helt centrale overvejelse bag normaleringsprocessen er imidlertid, at man altid skal kunne *rekonstruere* den komplekse datastruktur, der er udgangspunktet for det hele.

I forbindelse med normalisering bruger man begrebet *normalformer*, og man skelner for det meste mellem 5 normalformer, hvoraf vi her kun skal se nærmere på de tre første.

Første normalform

Det er lidt forskelligt, hvad der bliver fremhævet som kendetegnet på første normalform, men typisk skal følgende tre ting være opfyldt, for at en database siges at være i første normalform:

- Der skal være en nøgle, der entydigt identificerer den enkelte række i tabellen.
- De enkelte felter må kun indeholde én værdi.
- Der må ikke være kolonner, der gentages.

Hvad betyder det så i praksis? Hvis vi vender tilbage til vores simple system til styring af bogudlån, bliver de praktiske konsekvenser af principperne ganske enkle at gennemskue.

Vores udgangspunkt er, at vi gerne vil kunne registrere følgende data om et givet udlån:

Udlånsdata			
Låner	**Udlån 1**	**Udlån 2**	**Udlån ...**
Peter Nielsen	Karen Blixen	Johannes V. Jensen	Jostein Gaarder
Ellevang 3	Vintereventyr	Den lange rejse	Sofies verden
Holstebro	Gyldendal	Gyldendal	Høst og Søn
7500	1964	1977	1981
	22-11-2000	22-11-2000	22-11-2000

Vi har set på problemet med den manglende nøgle, der entydigt kan identificere den enkelte bog. Det er oplagt et problem i et system, der typisk køber flere eksemplarer af hver bog. Hvis ikke vi har en nøgle, ved hjælp af hvilken vi kan identificere den enkelte bog, har vi et problem, ikke kun når vi skal kassere en bog, men også hvis en låner låner to eksemplarer af samme bog. Når det ene eksemplar afleveres, bliver begge udlånsregistreringer slettet/nulstillet.

Den entydige nøgle er altså generelt en nødvendighed.

Det andet kriterium siger, at et felt kun må indeholde én værdi. Et eksempel, hvor man lægger flere værdier i et felt, ville være en låner-registrering, hvor man lagrede vejnavn, husnummer, by og postnummer i et samlet adressefelt. Det er et eksempel på, at man blander værdier sammen, der egentlig tilhører forskellige *domæner*. Vejnavne og bynavne er to forskellige egenskaber eller attributter, og de bør derfor lægges i separate kolonner.

Der er også oplagte grunde til, at det er bedre at lagre disse egenskaber i forskellige kolonner: Søgning og sortering på eksempelvis bynavn vil være noget sværere at lave, hvis navnet er lagret som del af et stort tekstfelt.

Det sidste problem er kolonnerne, som vi er nødt til at gentage for hver bog, der lånes, idet vi jo får en tabel med følgende format, hvis vi tillader at kolonner gentages:

Udlånsregistrering			
Kolonner til alle lånerdata	**Kolonner til alle data for bog 1**	**Kolonner til alle data for bog 2**	**Udlånsdato**
Peter Jensen Ringgaden 10 7500 Holstebro	Karen Blixen Vintereventyr Gyldendal 1964	Johannes V. Jensen Den lange rejse Gyldendal 1977	24-12-2000

Man skal her forstille sig, at der er fire kolonner til lånerdata, fire kolonner til data for bog 1, fire til data for bog 2 osv.

Bare det, at vi ikke ved, hvor mange kolonner vi har brug for, når vi laver databasen, er et problem. Men også her har vi et problem med søgning og sortering. Vi ved jo ikke, i hvilken kolonne bogtitlen *Vintereventyr* står. Så når vi skal finde ud af, hvem denne bog er udlånt til, skal vi søge i alle de forskellige bogtitel-kolonner. Vi kan heller ikke så let lave sorteringer, når vi skal se nærmere på data.

Nogle af problemerne kan vi løse med ganske enkle midler, nemlig ved at splitte data op to tabeller: en lånertabel og en udlånstabel.

Lånertabel			
LånerID	**Navn**	**Adresse**	**...**
1001	Peter Jensen	Ringgaden 10	...

Udlånstabel					
LånerID	**BogID**	**Forfatternavn**	**Titel**	**Forlag**	**Dato**
1001	1	Johannes V. Jensen	Den lange rejse	Gyldendal	22-11-2000
1001	4	Karen Blixen	Vintereventyr	Gyldendal	22-11-2000

Vi kan nu få vist alle de bøger, en given låner har lånt, hvis blot vi har vedkommendes *LånerID*. Vi kan søge og sortere på de enkelte titler, på udlånsdato etc. Så alt i alt er det et stort fremskridt.

Med denne lille ændring har vi også opfyldt alle kravene til en database i første normalform: Vi har en entydig identifikation af de enkelte rækker, både i lånertabellen og i udlånstabellen; vores felter indeholder kun en værdi, og vi har ikke længere brug for at gentage kolonner, der indeholder data, der egentlig hører til samme domæne. Kolonnerne er blevet konverteret til separate rækker i vores tabel.

Vores nøgle består i dette tilfælde af de to ID-felter: *LånerID* og *BogID*. Det er de to felter i kombination, der giver en entydig identifikation af udlånet.

Men kunne man indvende: Helt entydig er nøglen dog ikke. Hvad nu hvis samme låner låner samme bog to gange og lige netop får fat i samme eksemplar begge gange? Så er de to rækker identiske på nær datofeltet. For at håndtere en sådan situation skal *datofeltet* egentlig med i nøglen for at gøre den entydig. Og selv da er den ikke helt sikker: Hvad nu hvis låneren låner, afleverer og låner bogen igen samme dag? Så er vores nøgle med datofeltet ikke engang entydig.

Her kan man så indvende, at det er ret teoretisk og tænkt – at det måske slet ikke kan lade sig gøre i praksis. Men ved et ordrestyringssystem kunne samme kunde godt bestille samme vare to gange på samme dag. Det er faktisk slet ikke så usandsynligt, at det sker. Potentielt er der altså nogle problemer i den datastruktur, vi er nået frem til på nuværende tidspunkt. For at komme ud over dette problem burde vi indføre en nøgle, der entydigt identificerer det enkelte udlån.

Men lad os lige lade det ligge et øjeblik, fordi det i denne sammenhæng som sagt er et ret teoretisk problem. I den datastruktur, der er vist ovenfor, har vi opfyldt alle kravene til en database i første normalform. Lad os så se på, hvad anden normalform kræver af vores datastruktur.

Anden normalform

En database er i anden normalform, hvis den opfylder følgende krav:

- Den opfylder alle kravene til første normalform.
- Ingen attributter/egenskaber, der ikke selv tilhører nøglen, må afhænge af en del af nøglen.

Man kan også formulere denne regel lidt mere konkret: Alle kolonner i en tabel skal indeholde data om én og kun én entitet. Der opstår typisk problemer med anden normalform i de situationer, hvor vi har en primærnøgle, der er sammensat af to eller flere felter. I sådanne tilfælde må der ikke være felter i tabellen, der kun står i relation til en del af nøglen.

Som nævnt ovenfor består vores nøgle af to felter: `LånerID` og `BogID`. Det er disse to felter i kombination, der udgør den entydige

identifikation af den enkelte række. Hvis vi ser nærmere på vores udlånstabel, kan vi se, at alle vores data om de enkelte bøger kun afhænger af den del af nøglen, der hedder *BogID*. Der er ingen direkte sammenhæng eller afhængighed mellem *LånerID* og de forskellige boginformationer. Så egentlig indeholder vores kolonner data om forskellige entiteter, idet vi blander data om udlånet sammen med data om den enkelte bog.

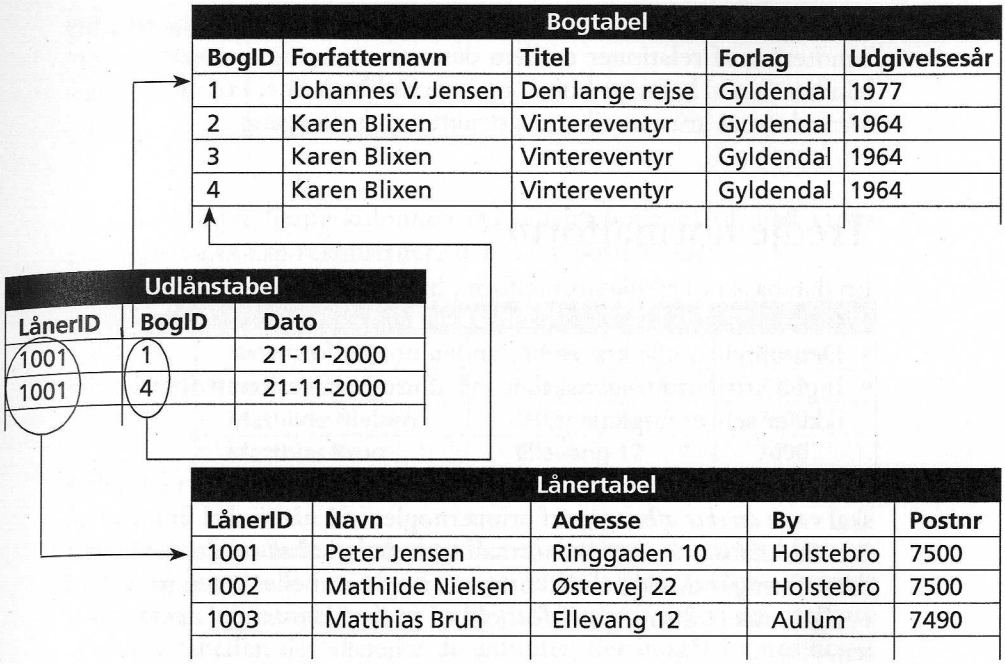
Udlånstabel					
LånerID	**BogID**	**Forfatternavn**	**Titel**	**Forlag**	**Dato**
1001	1	Johannes V. Jensen	Den lange rejse	Gyldendal	22-11-2000
1001	4	Karen Blixen	Vintereventyr	Gyldendal	22-11-2000

Vores database strider altså mod den ene af reglerne for anden normalform, idet 'ingen attributter/egenskaber, der ikke selv tilhører nøglen, må afhænge af en del af nøglen.' Og det er det, der er tilfældet her: Egenskaberne `Forfatternavn`, `Titel` og `Forlag` afhænger kun af `BogID`. Vi kan uden problemer udskifte `LånerID` uden at det påvirker de resterende data i rækken. Vi kan derimod ikke udskifte `BogID`, uden at vi også skal udskifte alle bogdata samtidig. Heraf kan vi se, at der er en afhængighed mellem netop disse kolonners data.

For at komme ud over dette problem er vi nødt til at oprette en ny tabel, hvis eneste funktion er at håndtere selve udlånet:

Udlånstabel		
LånerID	**BogID**	**Dato**
1001	1	21-11-2000
1001	4	21-11-2000

Nu er vi ved at være tilbage ved den model, vi brugte i forbindelse med introduktionen til grundbegreberne, hvor vi ved hjælp af de forskellige nøgler skaber relationer mellem tabeller, der indeholder data om separate ting, *entiteter*:



Bogtabel

BogID	Forfatternavn	Titel	Forlag	Udgivelsesår
1	Johannes V. Jensen	Den lange rejse	Gyldendal	1977
2	Karen Blixen	Vintereventyr	Gyldendal	1964
3	Karen Blixen	Vintereventyr	Gyldendal	1964
4	Karen Blixen	Vintereventyr	Gyldendal	1964

Udlånstabel

LånerID	BogID	Dato
1001	1	21-11-2000
1001	4	21-11-2000

Lånertabel

LånerID	Navn	Adresse	By	Postnr
1001	Peter Jensen	Ringgaden 10	Holstebro	7500
1002	Mathilde Nielsen	Østervej 22	Holstebro	7500
1003	Matthias Brun	Ellevang 12	Aulum	7490

Med denne struktur har vi opfyldt alle kravene til anden normalform, dvs.:

- Vi har for hver tabel en nøgle, der entydigt identificerer den enkelte række i tabellen.
- De enkelte felter indeholder kun en værdi.
- Vi har ingen kolonner, der gentages.
- Ingen af vores attributter eller egenskaber, der ikke selv tilhører nøglen, afhænger af en del af nøglen.

Det eneste problem, vi har, er det lidt teoretiske problem, der blev nævnt tidligere, hvor samme låner låner samme bog flere gange på samme dag. Det problem kan vi dog let komme ud over ved at indføre et unikt *UdlånsID* for hvert bogudlån. Så bliver dette ID den primære nøgle, der identificerer det enkelte udlån.

Udlånstabel

UdlånsID	LånerID	BogID	Dato
1	1001	1	21-11-2000
2	1001	4	21-11-2000

Som man kan se, bliver reglerne for databasens opbygning og dens håndtering af relationer mellem data strengere og strengere, efterhånden som vi kommer højere op i normalformerne. I tredje normalform skærpes kravene til vores struktur endnu engang.

Tredje normalform

En database er i tredje normalform, hvis den opfylder følgende krav:

- Den opfylder alle kravene til anden normalform.
- Ingen attributter/egenskaber må afhænge af andre attributter, der ikke er selv er nøgler.

Det centrale bag denne formulering er, at alle egenskaber i en række skal være *direkte afhængige* af primærnøglen. Så når vi skal finde ud af, om en database er i tredje normalform, skal vi altså se efter data, der ikke afhænger direkte af den primære nøgle i tabellen, men som står i et eller andet afhængighedsforhold til andre 'ikke-nøgle' data i tabellen.

Lad os se lidt nærmere på vores tabel med lånerdata:

Lånertabel				
LånerID	**Navn**	**Adresse**	**By**	**Postnr**
1001	Peter Jensen	Ringgaden 10	Holstebro	7500
1002	Mathilde Nielsen	Østervej 22	Holstebro	7500
1003	Matthias Brun	Ellevang 12	Aulum	7490

To af felterne står i et særligt forhold til hinanden: Her kan vi ikke bare skifte informationerne i det ene felt ud med tilfældige andre informationer. Det drejer sig om by og postnummer. Vi kan ikke her sætte et andet bynavn ind, uden at vores data bliver fejlagtige. Vi er nødt til at skifte indholdet i begge felter, fordi bynavnet afhænger af postnummeret.

Her har vi et eksempel på en afhængighed mellem én egenskab, nemlig bynavnet, og en anden egenskab, nemlig postnummeret, der ikke i denne sammenhæng er en nøgle. Hvis vi vil opfylde kravene til tredje normalform, skal vi lave en separat tabel til at håndtere relationen mellem postnumre og bynavne.

Postnummertabel	
Postnummer	**Bynavn**
7500	Holstebro
8520	Lystrup

Vi kan herefter fjerne kolonnen bynavn fra vores tabel, idet vi via postnummeret kan rekonstruere den fulde postadresse:

Lånertabel			
LånerID	**Navn**	**Adresse**	**Postnr**
1001	Peter Jensen	Ringgaden 10	7500
1002	Mathilde Nielsen	Østervej 22	7500
1003	Matthias Brun	Ellevang 12	7490

Med denne lille ændring har vi nu fået en database, der er i tredje normalform. I stedet for en stor og noget uhåndterlig tabel med mange kolonner – og mange små og store problemer – har vi en database med fire tabeller, der afspejler de entiteter, der indgår i vores noget forsimplede system til styring af et biblioteks bogudlån.

Yderligere normalformer

I de fleste fremstillinger af relationel databaseteori og normalisering opstilles der yderligere tre normalformer:

- Boyce-Codd normalformen (BCNF)
- fjerde normalform
- femte normalform.

Disse tre normalformer er indført for at kunne håndtere mere specielle problemer, som man i de fleste tilfælde kan komme uden om ved en konsekvent analyse og normalisering ud fra de tre første normalformer.