Normalisering

Normalisering är en formell metod för att designa relationsdatabaser. Grovt sett går normalisering ut på att dela upp databasen i många små tabeller för att undvika redundans. Det är en process som går ut på att avslöja inre samband mellan kolumner i en tabell och mellan olika tabeller som gör det nödvändigt att dela upp tabellen till fler tabeller. Vidare försöker vi hitta om data i en tabell upprepas eller bildar en grupp i en tabell. Om så är fallet måste vi återigen dela upp tabellen i nya tabeller.

Att normalisera en relationsdatabas innebär att vi delar upp befintliga tabeller till nya tabeller genom att följa en viss uppsättning regler. Dessa regler kan klassificeras i så kallade normalformer. Varje normalform sätter ett antal strikta krav på hur tabellerna ska utformas. Ju högre den normala formen är, desto strängare blir kraven. Vanligtvis behöver vi inte normalisera längre än till den tredje normalformen. Det finns dock en utökad version av den tredje normalformen som kallas Boyce-Codd normalform och vanligtvis är det denna normalform vi strävar att uppnå när vi normaliserar en tabell. Utöver denna normalform finns det även en fjärde och en femte normalform. Dessa två normalformer är inget vi går in på i denna kurs.

För att kunna normalisera en relationsdatabas måste alla tabeller har en primärnyckel och vi måste ha kännedom från verkliga livet om hur kolumnerna och tabellerna är relaterade till varandra. Bara att ha en lista över alla attribut som grund är normalt inte tillräckligt.

Som nämnts tidigare normaliserar vi för att vi önskar att förhindra lagring av onödig information och för att förhindra att vi för data som inte stämmer, så kallad inkonsistent data. Normaliserade tabeller underlättar också underhåll av databasen, samtidigt som databasen tar mindre plats på lagringsenheten. Det senare är mest relevanta om man talar om mycket stora databaser.

Nu är det emellertid så att genom att rita bra ER-diagram kommer vi ofta få en normaliserad lösning utan att egentligen behöva tänka på normaliseringsprocessen. I praktiken är det därför nästan viktigare att tänka på att skapa förnuftiga tabeller än vilka metoder vi använder för att komma fram till resultatet.

Normalisering är normalt den sista aktiviteten i processen att designa en databas. Ofta betraktas normalisering som en kontrollmekanism för tabellerna som följer av ER-diagramet. Ett slags kvalitetssäkring. Det lönar sig att göra denna kontroll innan databasen genomförs och används. Det är även lättare att upprätthålla en relationsdatabas som består av normaliserade tabeller.

Det är viktigt att påpeka att en normaliserad databas inte nödvändigtvis kommer att leda till en snabbare databas. I slutändan beror det på hur databasen ska användas. Om databasen ska uppdateras ofta skulle en normaliserad databas i alla fall vara tryggare, det vill säga vi undgår inkonsistent data i databasen.

En annan viktig aspekt som bör beaktas när det gäller databasens prestanda är vilka frågor (SQL) som kommer att användas mest. Det är så att en SQL-fråga som hämtar data bara från en tabell är mycket snabbare än en fråga som hämtar data från två eller flera tabeller. I stora databaser bör vi kanske därför undvika att använda allt för många kopplade frågor, vilket är lättare att göra om vi har större tabeller (tabeller som inte är uppdelade).

Å andra sidan finns det fördelar med att till exempel ha en separat tabell över postorter i Sverige, om det är så att vi ofta kommer att söka efter just postort. (Så här kan vi fortsätta att diskutera fram och tillbaka utan att egentligen komma fram till ett generellt svar på när och om vi ska fortsätta att normalisera en tabell).

Funktionella beroenden

För att förstå hur normaliseringsreglerna fungerar måste vi först förstå vad begreppet funktionella beroenden innebär. Funktionella beroenden har att göra med kolumnerna i en tabell. När vi börjar analysera en tabell i detalj kommer vi snart att upptäcka att några av kolumnerna i tabellen kan vara beroende på vissa andra kolumner i samma tabell när det kommer till de värden som ska infogas.

Helst bör varje kolumn i tabellen vara beroende av endast primärnyckeln. Eller sagt något annorlunda: primärnyckeln ska bestämma de övriga kolumnerna. Kolumnen valt att vara primärnyckel, som bestämmer värdet på en annan kolumn eller grupp av kolumner, kallas för en determinant. En determinant bestämmer värdet för, determinerar, de andra kolumnerna.

Tabell 1. Postort

postnummer	ort
83146	Östersund
83432	Brunflo
83988	Östersund

I tabell 1 har vi tabellen Postort med de två kolumnerna postnummer och ort där postnummer är primärnyckel. Här är postnummer en determinant. Om vi vet värdet av postnummer kommer det bara att finnas en stad med detta postnummer. Inte tvärtom. Om vi tar en viss stad kan den ha många olika postnummer. Postnummer är därmed en determinant. Detta funktionella beroende skriver vi så här:

```
postnummer --> ort
```

Även en sammansatt nyckel kan fungera som en determinant.

Tabell 2. Timdebitering

klientnr	ansnr	timmar
1	2	10
1	1	15
2	3	3
3	2	10

I tabell 2 visas tabellen Timdebitering med en sammansattnyckel bestående av klientnr och ansnr. Det är kombinationen av vilken anställd som har aktuell klient som determinerar hur många timmar den anställde ska debitera klienten. Detta funktionella beroende skriver vi så här:

```
{klientnr, ansnr} --> ort
```

Formellt sett kan vi uttrycka ett funktionellt beroende (fb) som: Om A och B är attribut i en relation R så är B fb av A (skrivs A --> B) om det för varje värde på A finns ett och endast ett tillhörande värde på B. A och B kan var för sig bestå utav flera attribut.

Normalformer

Som nämnts tidigare innebär normalisering att vi delar upp befintliga tabeller till nya tabeller genom att följa en viss uppsättning regler. Vilka dessa regler är definieras i olika normalformer som vi nu ska titta närmare på.

Onormalierad form (UNF)

En onormaliserad tabell innehåller det vi kallar för repeterande grupper. Det betyder att en tabell som är onormaliserad kan innehålla en lista av värden i en kolumn. Tabellen Studerat i tabell 3 nedan innehåller repeterande grupper i kolumnen kurskod.

Tabell 3. Studerat

portalid	förnamn	efternamn	postnr	ort	kurskod
nian0003	Nils	Andersson	83146	Östersund	DT022G, DT013G
jaby1200	Jan	By	83432	Brunflo	DT022G, DT031G
anla1401	Anna	Larsson	83988	Östersund	DT062G
sawe1312	Sara	Wedin	83146	Östersund	DT013G

Vi ser att för studenterna med portalid nian003 och jaby1200 finns det en lista av värden i kolumnen kurskod. Uppgiften nu är att få tabellen till första normalformen.

Första normalformen (1NF)

1NF innehåller den enklaste regeln av alla normalformer. Den säger att: I alla kolumner i tabellen ska det alltid bara vara ett värde, ett så kallat atomärt värde. En tabell som uppfyller 1NF får med andra ord inte innehålla några repeterande grupper, vi godtar alltså inte kolumner som innehåller en lista med värden.

För att normalisera en tabell från UNF till 1NF skriver vi varje värde i listan på en egen rad. Övriga kolumners värden kopierar vi från den ursprungliga raden. Som ny primärnyckel i tabellen används en sammansatt nyckel bestående av den ursprungliga primärnyckeln och den eller de kolumner som innehöll repeterande grupper. Tabellen Studerat kommer att se ut så som tabell 4 visar efter att vi har normaliserat den till 1NF.

Tabell 4. Studerat

portalid	förnamn	efternamn	postnr	ort	kurskod	termin
nian0003	Nils	Andersson	83146	Östersund	DT022G	HT01
nian0003	Nils	Andersson	83146	Östersund	DT013G	VT02
jaby1200	Jan	By	83432	Brunflo	DT022G	VT12
jaby1200	Jan	By	83432	Brunflo	DT031G	VT12
anla1401	Anna	Larsson	83988	Östersund	DT062G	HT14
sawe1312	Sara	Wedin	83146	Östersund	DT013G	HT14

Märk väl att det som vi betraktar som ett atomärt värde är en fråga om tolkning. I studenttabellen kurskoderna DT022G och DT013G är atomära värden. Det är logiskt att tolkningen är sådan eftersom det gör det lättare att senare hitta alla personer som studerat en viss kurs.

Att ta en tabell från UNF till 1NF är bara ett litet steg på vägen till en helt normaliserad tabell. Att få bort de repeterande grupperna är inte tillräckligt för att råda bot på alla eventuella problem som kan bryta mot någon av normaliseringsreglerna. Vid närmare analys av en sådan tabell (som gått från UNF till 1NF) kommer vi upptäcka att en sådan tabell innehåller det vi kallar för partiella beroenden. Detta uppstår när delar av en primärnyckel, eller andra kandidatnycklar, determinerar en kolumn som inte är en eller ingår i en kandidatnyckel.

Tittar vi i tabell 4 på tabellen Studerat består den nu av en sammansatt primärnyckel bestående av kolumnerna portalid och kurskod. Vi har lagt till kolumnen termin som anger vilken termin studenten läste en viss kurs. Helst ska hela primärnyckeln (alla kandidat-nycklar) nu determinera resterande kolumner i tabellen. I denna tabell är dock så inte fallet. Vi ser att på de rader där det i kolumnen portalid står värdet nian0003 alltid står t.ex. Nils och Andersson i kolumnerna förnamn och efternamn. Kolumnerna förnamn, efternamn, postnr och ort determineras alltså av portalid, som är en del av en kandidatnyckel.

De funktionella beroenden som vi identifierat så långt i tabellen är följande:

```
{portalid, kurskod} --> förnamn, efternamn, postnr, ort, termin
portalid --> förnamn, efternamn, postnr, ort
```

Tabellen uppfyller kraven för 1NF, men inte någon högre normalform. Hur löser vi då problemet med partiella beroenden? Jo, vi delar upp tabellen genom att flytta alla kolumner som ingår i det funktionella beroendet till en ny tabell där determinanten blir primärnyckel. Kvar i den gamla tabellen lämnas determinanten som i relationsmodellen blir en främmande nyckel till den nya tabellens primärnyckel.

Tabell 5. Studerat

portalid	kurskod	termin
nian0003	DT022G	HT01
nian0003	DT013G	VT02
jaby1200	DT022G	VT12
jaby1200	DT031G	VT12
anla1401	DT062G	HT14
sawe1312	DT013G	HT14

Tabell 6. Student

portalid	förnamn	efternamn	postnr	ort
nian0003	Nils	Andersson	83146	Östersund
jaby1200	Jan	By	83432	Brunflo
anla1401	Anna	Larsson	83988	Östersund
sawe1312	Sara	Wedin	83146	Östersund

Översatt till relationsmodellen får vi följande tabeller:

```
Student (portalid, förnamn, efternamn, postnr, ort)
Studerat (portalid*, kurskod, termin)
```

Både tabellen Studerat och Student uppfyller nu kraven i andra normalformen eftersom vi har åtgärdet det partiella beroendet som fanns. Vi har nu följande funktionella beroenden:

```
Studerat
{portalid, kurskod} --> termin

Student
portalid --> förnamn, efternamn, postnr, ort
```

Andra normalformen (2NF)

En tabell uppfyller 2NF om alla kolumner som inte ingår i någon kandidatnyckel är funktionellt beroende på hela primärnyckeln (och andra kandidatnycklar). Med andra ord måste alla kolumner, som inte ingår i någon nyckel, determineras av alla och hela kandidatnycklarna och inte bara en del av den (gäller alltså sammansatta kandidatnycklar).

Tabeller som uppfyller 2NF har dock fortfarande en svaghet. De kan innehålla det vi kallar för transitiva beroenden som gör att tabellen inte uppfyller tredje normalformen. Detta innebär att en kolumn som inte ingår i någon kandidatnyckel determinerar en annan kolumn i tabellen.

Tar vi en närmare titt på tabellen Student (i tabell 6) ser vi att det finns ett samband mellan postnr och ort. På varje rad där det står 83146 som postnr står det alltid Östersund som postort (tvärtom gäller inte eftersom orten Östersund har olika postnummer). Vi har ett funktionellt beroende mellan postnummer och ort enligt följande:

```
portalid --> förnamn, efternamn, postnr, ort
postnr --> ort
```

Kolumnen postnr är ingen kandidatnyckel eftersom det kan finnas flera studenter som bor i samma postnummerområde. Inte heller ingår postnr i någon annan kandidatnyckel. Vi har därför ett transitivt beroende mellan postnr och ort.

För att få tabellen till 3NF måste vi återigen dela upp tabellen enligt samma sätt som vi gjorde för att få tabellen till 2NF. Vi flyttar kolumnerna som ingår i det funktionella beroendet (postnr och ort) till en ny tabell. I den nya tabellen sätter vi determinanten postnr

som primärnyckel. Kvar i den gamla tabellen lämnar vi determinanten som en främmande nyckel till den nya tabellens primärnyckel.

Tabell 7. Student

portalid	förnamn	efternamn	postnr
nian0003	Nils	Andersson	83146
jaby1200	Jan	By	83432
anla1401	Anna	Larsson	83988
sawe1312	Sara	Wedin	83146

Tabell 8. Postort

postnr	ort
83146	Östersund
83432	Brunflo
83988	Östersund

Översatt till relationsmodellen får vi följande tabeller:

```
Student (portalid, förnamn, efternamn, postnr*)
Postort (postnr, ort)
```

Både tabellen Student och Postort uppfyller kraven i tredje normalformen eftersom vi har åtgärdet det transitiva beroendet som fanns. Vi har nu följande funktionella beroenden:

```
Student
portalid --> förnamn, efternamn, postnr

Postort
postnr --> ort
```

Tredje normalformen (3NF)

En tabell uppfyller kraven för 3NF om 2NF är uppfyllt och det inte finns några funktionella beroenden mellan kolumner som inte ingår i någon kandidatnyckel. Det vill säga en kolumn som inte ingår i en kandidatnyckel kan alltså inte vara beroende av några andra kolumner än de som ingår i en kandidatnyckel.

Boyce-Codds normalform (BCNF)

Om vi har en tabell i som uppfyller 3NF och den tabellen bara har en kandidatnyckel, då uppfyller tabellen automatiskt även BCNF. Följaktligen motsvarar 3NF BCNF om det bara finns en kandidatnyckel i tabellen. Om vi har mer än en kandidatnyckel eller om tabellen har en sammansatt primärnyckel, innebär 3NF inte automatiskt BCNF. Därför är BCNF bara av intresse om det handlar om en tabell som består av sammansatta nycklar. Vi sammanfattar detta som: en tabell är i BCNF om och endast om varje determinant är en kandidatnyckel.

För att förstå BCNF bättre tittar vi på ett nytt exempel. I tabell 9 finns en tabell som heter Undervisning. Den innehåller information om elever, vilka kurser de studerar och deras lärare på kursen.

Tabell 9. Undervisning

student	kurs	lärare
Olsson	Databaser	Pettersson
Nilsson	Databaser	Hansson
Nilsson	Programmering	Karlsson
Nilsson	Linux	Oskarsson
Elofsson	Programmering	Andersson
Elofsson	Databaser	Pettersson
Berg	Databaser	Larsson
Albertsson	Databaser	Hansson

Denna tabell har problem med både insättningar, borttagningar och uppdateringar. Om vi vill ersätta läraren Hansson måste vi göra en ändring på två ställen eftersom han är lärare för två studenter. Om vi måste ta bort studenten Elofsson kommer både läraren Andersson och Pettersson att tas bort. Detta är mycket olyckligt.

De funktionella beroenden som existerar i tabellen är följande:

```
{student, kurs} --> lärare
lärare --> kurs
```

Som vi ser har vi en sammansatt primärnyckel (kandidatnyckel) bestående av kolumnerna student och kurs. Den sammansatta nyckeln determinerar lärare. Tabellen uppfyller 2NF eftersom det inte finns några partiella beroenden. Det finns inte heller några transitiva beroenden i tabellen och därmed uppfyller tabellen även 3NF.

Vi ser dock att lärare, som inte är en kandidatnyckel eller ingår i någon, determinerar kurs, som är en del av en sammansatt kandidatnyckel. Vi har en determinant, lärare, som inte är en kandidatnyckel. Det betyder att tabellen inte uppfyller kraven för BCNF.

För att normalisera denna tabell till BCNF gör vi precis som tidigare. Vi flyttar kolumnerna som ingår i det funktionella beroendet (lärare och kurs) till en ny tabell. I den nya tabellen sätter vi determinanten (lärare) som primärnyckel. Kvar i den gamla tabellen lämnar vi determinanten som en främmande nyckel. Notera att vi i tabellen Undervissning måste använda en sammansatt primärnyckel av student och lärare för att vi ska få en unik nyckel.

Tabell 10. Undervisning

student	lärare
Olsson	Pettersson
Nilsson	Hansson
Nilsson	Karlsson
Nilsson	Oskarsson
Elofsson	Andersson
Elofsson	Pettersson
Berg	Larsson
Albertsson	Hansson

Tabell 11. Lärare

lärare	**kurs**
Pettersson	Databaser
Karlsson	Programmering
Oskarsson	Linux
Andersson	Programmering
Larsson	Databaser
Hansson	Databaser

Översatt till relationsmodellen får vi följande tabeller:

```
Undervisning (student, lärare*)
Lärare (lärare, kurs)
```