Databaser

- Normalisering -

Innholdsfortegnelse

1. Normalisering 2
   1.1. Redundans 2
   1.2. Anomalier (uregelmessigheter etter oppdateringer i databasen) 2
      1.2.1. Innsettingsanomalier (Insertion anomalies) 3
      1.2.2. Sletteanomalier (Delete anomalies) 3
      1.2.3. Oppdateringsanomalier (Update anomalies) 3
   1.3. Normalisering av tabeller - Normalformer 3
   1.4. 1NF (Første normalform) 4
   1.5. Supernøkkel (Super Key) 4
   1.6. Kandidatnøkkel (Candidate Key) 4
   1.7. Funksjonelle avhengigheter 5
   1.8. 2NF (Annen normalform) 5
   1.9. 3NF (Tredje normalform) 6
   1.10. BCNF (Boyce Codd normalform) 7
   1.11. Ulemper med normalisering: 7

1. Normalisering

Normalisering er en metode knyttet til dekomponering av tabeller (splitting av tabeller/relasjoner i mindre enheter), for å sikre en god tabellstruktur. I dette kapittelet ses det kort på noen av de viktigste prinsippene knyttet til dette, så man har noen «verktøy» til å sjekke tabellstrukturen med, før det lages en database. Målet er å unngå redundans og sikre dataintegritet.

Det å lage et godt E/R-diagram er gjerne det første bidraget til en god tabellstruktur. Etter dette bør normalisering utføres. To viktige ting normalisering bidrar til er å unngå:

- redundans (dvs. lagring av overflødig informasjon)
- anomalier (dvs. uregelmessigheter i databasen som konsekvens av oppdateringer i en dårlig tabellstruktur)

1.1. Redundans

Redundans kan betraktes som overflødige data lagret i databasen. Et klassisk eksempel er lagring av postnummer og poststed når det lagres persondata, som vist et eksempel på i Tabell 1.

Tabell 1: Persontabell

1	Per	Hansen	3918	Porsgrunn
2	Lise	Olsen	3918	Porsgrunn
3	Frank	Olsen	3720	Skien
4	Hanne	Fjell	3720	Skien

I dette eksempelet ses det at det at for hver nye person som registreres med postnummeret 3918, gjentas Porsgrunn. På tilsvarende vis gjentas Skien for hver forekomst av personer med postnummer 3720. Dette er et typiske eksempler på redundante data. Lagring på denne måten har flere utfordringer:

- Det kan inntreffe anomalier (uregelmessigheter/feil), ved at ikke alle data oppdateres korrekt Dette ses det nærmere på i kapittel 1.2.
- Unødvendig dobbeltlagring av data kan inntreffe når ett datasett kan avledes av et annet. Er det for eksempel nødvendig å lagre pris både *uten* og *med* mva.?
- Unødvendig dobbeltlagring krever ekstra lagringsplass.

I enkelte tilfeller kan det være ønskelig eller påkrevd med redundante data, eksempelvis for å sikre raske nok oppslag i databasen. Håndteringen må da gjøres på en måte som sikrer at feil ikke kan inntreffe, det vil si kontrollert redundans.

I Tabell 1 kan redundansen unngås ved å skille ut postnummer og poststed i en egen tabell. Dette kalles en normalisering, noe vi senere skal se nærmere på reglene for.

1.2. Anomalier (uregelmessigheter etter oppdateringer i databasen)

Anomalier er uregelmessigheter, avvik eller feil som kan inntreffe som en konsekvens av en tabell-struktur som ikke er normalisert.

Det er vanlig å inndele anomalier i tre typer:

1. Innsettingsanomalier (Insertion anomalies)
2. Sletteanomalier (Delete anomalies)
3. Oppdateringsanomalier (Update anomalies)

I det følgende forklares disse anomalitypene med henvisning til dataene i Tabell 2.

Tabell 2: Tabell for registrering av personer som melder seg på ulike kurs

PersonId	Fornavn	Etternavn	Postnr	Poststed	KursId	Kursnavn	Kursdato
1	Per	Hansen	3918	Porsgrunn	1	Databaser	3. mai
1	Per	Hansen	3918	Porsgrunn	7	C#	30. mai
3	Frank	Olsen	3720	Skien	7	C#	30. mai
4	Hanne	Fjell	3720	Skien	1	Databaser	3. mai
3	Frank	Olsen	3720	Skien	1	Databaser	3. mai
1	Per	Hansen	3918	Porsgrunn	12	Java	5. juli
5	Lise	Karlsen	3740	Skien	19	Operativsystemer	2. februar

1.2.1. Innsettingsanomalier (Insertion anomalies)

Dersom innlegging av én type data i en tabell, gjør at man samtidig er nødt til å registrere en annen type ikke-samhørende data i den samme tabellen, foreligger det en innsettingsanomali.

Av Tabell 2 ses det at radene ikke kan identifiseres unikt kun med attributtene KursId, Kursnavn og Kursdato. Dette betyr at dersom det skal registreres et nytt kurs, må det samtidig registreres en person. I annet fall brytes det med entitetsintegritetsreglene, da det vil bli stående nullmerker i attributter som kreves for å identifisere en rad. Dette er ett eksempel på en innsettingsanomali.

Skal det påmeldes en ny person på et kurs, må samtidig all informasjon om kurset (KursId, Kursnavn og Kursdato) registreres i tillegg til persondataene. Dette må gjøres for hver eneste person som skal påmeldes et kurs. Dette gir mulighet for feilregistrering av data i noen av radene, sånn at det kan bli liggende ulik informasjon registrert om ett og samme kurs. Dette er et eksempel på en innsettingsanomali.

1.2.2. Sletteanomalier (Delete anomalies)

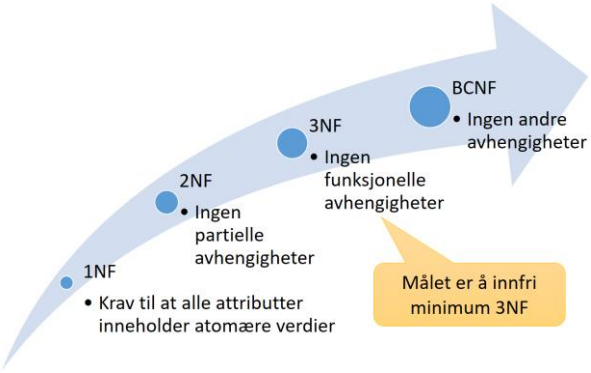
Ønskes det å slette kurset «Operativsystemer», f.eks. fordi det kun er én påmeldt, kan dette ikke gjøres uten samtidig å slette all informasjon om personen påmeldt dette kurset. Dersom denne personen kun er påmeldt dette kurset, har man mistet all personinformasjonen. Dette kalles en sletteanomali.

1.2.3. Oppdateringsanomalier (Update anomalies)

La oss si at kursnavnet «Databaser» ønskes endret til «Relasjonsdatabaser». Da må alle forekomster der noen er påmeldt dette kurset endres. Dette gir mulighet for feil, dersom ikke alle radene oppdateres, og er et eksempel på en oppdateringsanomali.

1.3. Normalisering av tabeller - Normalformer

For å løse problemet med redundans og anomalier normaliseres tabellene. Det finnes ulike normalformer, første, andre, tredje, Boyce Codd, fjerde og femte normalform, gjerne forkortet til 1NF, 2NF, 3NF, BCNF, 4NF og 5NF. Vi ønsker i våre eksempler å normalisere tabellene så de innfrir kravene til 3NF. Gjør de det, innfrir de også kravene til 1NF og 2NF. Normaliseringstrinnene er illustrert i Figur 1-1, der man først sjekker mot 1NF, så 2NF osv.



Figur 1-1: Normaliseringstrinnene.

1.4. 1NF (Første normalform)

For at en tabell skal innfri 1NF-kravet, kreves det at alle attributter inneholder atomære verdier, hvilket vil si at dataene må representere entydige verdier. I Tabell 3 er det vist et eksempel på en tabell der dette kravet brytes. For kurset med KursId 1 (med kursnavn «Databaser»), er det registrert tre ulike kursdatoer, hvilket ikke er en atomær verdi.

Tabell 3: Tabell som bryter med kravet til kun atomære verdier, altså brudd på 1NF

KursId	Kursnavn	Kursdato
1	Databaser	3. mai 4. mai 6. mai

For å løse dette problemet, kan det opprettes en ny tabell kalt KURSAVVIKLING. I denne blir kombinasjonen av KursId og Kursdato en kombinert primærnøkkel og KursId en fremmednøkkel mot den opprinnelige tabellen. Da kan det for hver dato registreres flere kurs og hvert kurs kan oppføres på flere datoer. Den nye tabellen er vist i Tabell 4, med en del eksempeldatoer innlagt. Tabellen KURSAVVIKLING blir da en koblingstabell, som representerer en kobling mellom datoer og kurs.

Tabell 4: Tabell som innfrir 1NF-kravet om atomære verdier.

KURSAVVIKLING	
Kursdato	**KursId**
3. mai	1
4. mai	1
4. mai	10
5. mai	17
6. mai	1

Før det gås videre for å finne ut om det foreligger noen brudd på 2NF, defineres begrepene Supernøkkel, Kandidatnøkkel og funksjonell avhengighet

1.5. Supernøkkel (Super Key)

En supernøkkel er en kombinasjon av attributter i en tabell, som kan identifisere samtlige attributter i tabellen. Dette betyr at kombinasjonen av ***alle*** attributtene i tabellen, alltid vil være en supernøkkel. I tillegg vil alle andre kombinasjoner av attributter, med tilsvarende egenskaper, også være supernøkler. En tabell vil derfor normalt ha mange supernøkler.

1.6. Kandidatnøkkel (Candidate Key)

En kandidatnøkkel er en minimal supernøkkel, hvilket gjør den til en **kandidat** til å velges til primærnøkkel. At supernøkkelen er minimal, betyr at det ikke kan fjernes attributter fra supernøkkelen ***uten*** at den ***mister*** sin egenskap som supernøkkel. Ofte har en tabell bare ***én*** kandidatnøkkel, men den ***kan*** ha flere.

Dersom en tabell for studenter har både attributtene Studentnummer og Personnummer, vil begge være supernøkler. Begge vil unikt kunne identifisere alle de øvrige attributtene. Samtidig kan det ikke fjernes noe fra dem uten at de mister sin egenskap som supernøkler. I dette tilfellet består begge nøklene av kun ***ett*** attributt, og da er det innlysende at ikke noe kan fjernes fra dem. Begge er dermed kandidatnøkler, det vil si mulige primærnøkler, i en slik tabell.

En tabell må alltid ha nøyaktig ***én*** primærnøkkel, aldri flere, men primærnøkkelen kan bestå av ***flere*** attributter, hvilket i så fall gjør den til en kombinasjonsnøkkel (Combined Primary Key).

Primærnøkkelen velges blant kandidatnøklene. Er det bare én kandidatnøkkel, er valget enkelt, for da ***må*** denne brukes som primærnøkkel. Er det flere kandidatnøkler, må en av dem velges som primærnøkkel. I Studenttabelleksempelet vil Studentnummer trolig være et naturlig valg som primærnøkkel, fordi det er knyttet en god del restriksjoner fra Datatilsynet når det gjelder bruk av en persons personnummer.

1.7. Funksjonelle avhengigheter

For å finne kandidatnøkler må man først definere funksjonelle avhengigheter. For å kunne gjøre dette, kan man ikke bare se på ***noen*** av radene med data, men man må ***vite*** noe om hvordan attributtene henger sammen. Dette henger sammen med spesifikasjonen av systemet, etter oppdragsgivers behov.

Dersom det er sånn at A, her definert som ett attributt eller en kombinasjon av flere attributter, alltid gir B, definert som et annet attributt eller kombinasjon av attributter, sier vi at det en funksjonell avhengighet fra A til B, hvilket skrives A → B (A gir B). Den funksjonelle avhengigheten må i så fall gjelde for enhver rad i tabellen. I en slik avhengighet er A en **determinant**, som gir B.

I Tabell 3 kan man tenke seg flere funksjonelle avhengigheter. For eksempel vil det være naturlig å anta at én PersonId alltid vil gi det samme fornavnet og etternavnet: PersonId → Fornavn, Etternavn

Videre vil det være sånn at ett spesifikt postnummer alltid vil gi samme poststed. Dette er altså også en funksjonell avhengighet: Postnummer → Poststed.

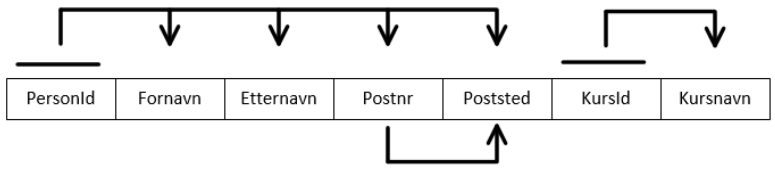
Én KursId vil høyst sannsynlig alltid gi samme kursnavn: KursId → Kursnavn. Dersom det også er sånn at ett konkret kurs alltid settes opp på én konkret dato, vil også følgende avhengighet gjelde: KursId -> Kursnavn, KursDato, men det er ikke sikkert at det foreligger en slik avhengighet. Det må derfor utføres en analyse m.h.p. før det kan konkluderes med hvilke avhengigheter som foreligger.

For å gjøre normaliseringen mer oversiktlig, erstattes ofte attributtnavnene med A, B, C osv. Dette fordi at attributtnavnene ikke er av betydning for normaliseringen etter at de funksjonelle avhengighetene og primærnøkkelen er påført. Deretter følger normaliseringen et fast sett med regler.

1.8. 2NF (Annen normalform)

For at en tabell skal innfri kravet til 2NF, må det ikke foreligge noen partielle avhengigheter. Partielle avhengigheter vil si at det finnes funksjonelle avhengigheter med attributter ***partielt*** avhengig av primærnøkkelen, eller sagt på en annen måte, avhengig av ***en del*** av primærnøkkelen.

For å avgjøre om en tabell er på 2NF (andre normalform), må det avklares hva som skal være tabellens primærnøkkel, fordi denne inngår i definisjonskravet for 2NF. Etter en analyse er de funksjonelle avhengighetene, illustreres avhengighetene med piler, som vist i Figur 1-2.



Figur 1-2: Tabellen påført funksjonelle avhengigheter og primærnøkkel.

Kandidatnøkkel/kandidatnøkler:

Det første som må gjøres er å finne tabellens kandidatnøkkel/-nøkler. ***PersonId og KursId*** i kombinasjon er en supernøkkel. Det kan ikke fjernes noe fra denne nøkkelen, uten at den mister sin egenskap som supernøkkel. Dermed er dette en kandidatnøkkel.

Primærnøkkel:

Det er ingen andre kandidatnøkler enn kombinasjonen ***PersonId*** og ***KursId*** i denne tabellen, og dermed må denne kombinasjonen velges som primærnøkkel. Primærnøkkelen er markert i Figur 1-2 ved å påføre en strek over attributtene som inngår i denne.

Primærnøkkel: PersonId, KursId → Fornavn, Etternavn, Postnr, Poststed, Kursnavn.
(PersonId, KursId gir underforstått også seg selv, men dette tas normalt ikke med i oppstillingen).

Andre funksjonelle avhengigheter:

PersonId → Fornavn, Etternavn, Postnr, Poststed
Postnr → Poststed
KursId → Kursnavn

Er tabellen på 2NF?

Det sjekkes så om noen av de funksjonelle avhengighetene utenom primærnøkkelen bryter med kravet til at det ikke skal foreligge noen partielle avhengigheter.

Det ses da at Fornavn, Etternavn, Postnr, Poststed er funksjonelt avhengig av PersonId, som er en del av primærnøkkelen. Dette er dermed et brudd på kravet til 2 NF.

I tillegg er Kursnavn funksjonelt avhengig av KursId, som også er en del av primærnøkkelen.

Den siste avhengigheten, Postnr → Poststed bryter ***ikke*** med 2NF, da Postnr ikke er en del av primærnøkkelen.

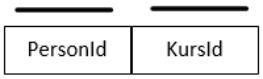
Siden det foreligger to avhengigheter som bryter med 2NF, betyr dette at tabellen er på 1 NF.

Hvordan løse brudd på 2NF:

For å løse problemet med brudd på 2NF, normaliseres tabellen. Dette gjøres ved å skille ut attributtene som er funksjonelt avhengige av en del av primærnøkkelen i egen/egne tabeller. De nye tabellene er vist i Figur 1-3.

Figur 1-3: De funksjonelle avhengighetene som bryter med 2NF er skilt ut i to nye tabeller.

PersonId og KursId må samtidig bevares i den opprinnelige tabellen, så man ikke mister den logiske sammenhengen mellom de opprinnelige attributtene. For å bevare denne sammenhengen blir disse attributtene fremmednøkler som refererer/peker til de nye tabellene. Den resterende delen av den opprinnelige tabellen blir etter dette en tabell kun med attributtene PersonId, KursId, som vist i Figur 1-4.



Figur 1-4: Resterende del av den opprinnelige tabellen.

Den opprinnelige tabellen er nå normalisert til tre tabeller. Alle innfrir kravene til 2NF, da det ikke lenger er noen attributter som er partielt avhengig av primærnøkkelen.

Tabellene kan skrives med følgende notasjon, der primærnøkler angis med understrekning og fremmednøkler er angitt med en stjerne bak (*). Dersom primærnøkler og/eller fremmednøkler består av flere attributter, settes en parentes rundt attributtene som inngår i nøkkelen. Gjelder dette en fremmednøkkel, settes stjernen i så fall bak parentesen.

De normaliserte tabellene kan da stilles opp sånn, der passende tabellnavn er valgt:

PERSON (PersonId, Fornavn, Etternavn, Postnr, Poststed)

KURS (KursId, Kursnavn)

PÅMELDING (PersonId*, KursId*)

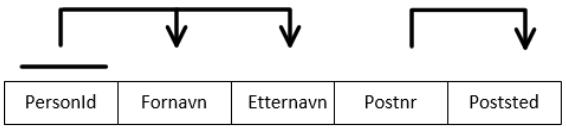
1.9. 3NF (Tredje normalform)

I 3NF er kravet at det ikke foreligger noen transitive avhengigheter. Dette betyr at ingen attributter kan være indirekte avhengig av primærnøkkelen. I praksis vil dette si at det ikke må foreligge noen funksjonelle avhengigheter mellom ikke-nøkkelattributter.

I den nye tabellen PERSON, som ble opprettet gjennom normaliseringen til 2NF, er det en funksjonell avhengighet mellom Postnr og Poststed (Postnr → Poststed). Dette er vist med pil i Figur 1-5. Ingen av disse to attributtene inngår som en del av primærnøkkelen. Dette er dermed en avhengighet mellom ikke-nøkkelattributter, hvilket innebærer et brudd på 3NF.

Den funksjonelle avhengigheten her, kalles transitiv avhengighet. Dette fordi Postnr er avhengig av PersonId og Poststed er avhengig av Postnr. Dette kan stilles opp sånn: PersonId → PostNr og PostNr → Poststed. Implisitt gjelder da at PersonId → Poststed, hvilket er den transitive avhengigheten.

Tabellen er dermed på 2NF, fordi den bryter med 3NF.



Figur 1-5: Avhengigheten Postnr → Poststed bryter med kravet til 3NF. Tabellen er dermed på 2NF.

1.10. BCNF (Boyce Codd normalform)

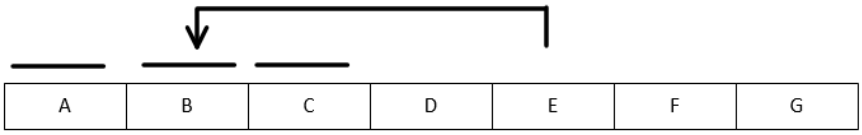
Dersom en tabell er normalisert til 3NF, vil den i de fleste tilfeller også innfri kravene til BCNF. Brudd på BCNF er sjeldne og litt vanskelige å håndtere på en god måte, så derfor stopper man ofte normaliseringen når tabellen er på 3NF.

Kravet til BCNF er at enhver determinant (attributt/attributter i en funksjonell avhengighet som bestemmer det/de andre attributtet/attributtene) skal være en kandidatnøkkel.

En konsekvens av et brudd på BCNF er at det vil være en determinant som peker tilbake til en del av primærnøkkelen (da alle transitive avhengigheter jo ble fjernet med 3NF).

I tabellene vi har operert med i eksemplene i dette kapittelet har det ikke vært noen brudd på BCNF.

Et BCNF-brudd kan illustreres som vist i Figur 1-6. Her vises det en funksjonell avhengighet (E → B), med en determinant (E) som ikke er en kandidatnøkkel (da ikke alle attributtene i tabellen er funksjonelt avhengig av denne). Dette bryter med BCNF-kravet, og tabellen er dermed på 3NF.



Figur 1-6: En illustrasjon av et brudd på BCNF. Tabellen er på 3NF.

1.11. Ulemper med normalisering:

Konsekvensen av normalisering er at en tabell splittes opp i mindre tabeller. Dette betyr at det ved spørringer hyppigere vil forekomme «joining» av tabeller, hvilket kan gjøre spørringene tregere, samt mer kompliserte å lage.

Å lage VIEWs kan kompensere noe for problemet med hyppig «joining». Likevel vil det av og til være hensiktsmessig ikke å normalisere fullt ut av slike hensyn. Som en konsekvens vil det da lagres redundante data i databasen, som må håndteres kontrollert.

Tilsvarende gjelder når data lagret i én kolonne, kan avledes/beregnes ut fra data i en annen/andre kolonne/kolonner. Dersom det kreves omfattende beregninger med data som må hentes fra en annen/andre kolonner, vil dette kunne bli svært tidkrevende. I noen tilfeller vil det da velges bevisst å dobbeltlagre data i slike sammenhenger.

Et eksempel kan være bankkontoer der det stadig skal hentes ut saldo. Det må der hver gang det skal hentes ut en saldo, utføres beregninger basert på en rekke posteringer, dersom ikke saldo fortløpende lagres som tilleggsdata i en egen kolonne. Da kan et valg om å innføre en redundant saldo-kolonne være et alternativ.

Ved kontrollert redundans blir det naturlig nok svært viktig at det programmeringsmessig sikres at det ikke kan inntreffe feil/uregelmessigheter som konsekvens av den redundante datastrukturen.