UNIVERSITETET
I OSLO

Relasjonsdatabasedesign

Funksjonelle avhengigheter

Oppdateringsanomalier

Dekomposering

Definisjon av nøkler (rep.)

Gitt et relasjonsskjema $R(A_1,A_2,\ldots,A_n)$ med tilhørende integritetsregler

La X være en delmengde av {A1,A2,…,An}.
Hvis t er et tuppel i en instans av R, betegner t[X] verdiene i t's X-attributter.

- **Supernøkkel**: En delmengde X av $\{A_1,A_2,\ldots,A_n\}$ som er slik at hvis t og u er to tupler hvor t≠u, så er t[X] ≠u[X].
  Dvs. t og u skal alltid ha forskjellig verdi i minst ett av attributtene i X
- **Kandidatnøkkel**: En minimal supernøkkel.
  Dvs: Fjerning av et hvilket som helst attributt fører til at de gjenværende attributtene ikke lenger utgjør en supernøkkel
- **Primærnøkkel**: En utvalgt blant kandidatnøklene.
  Alle relasjonsskjemaer skal ha nøyaktig én primærnøkkel
- **Nøkkelattributt** (*prime attribute*): Attributt som er med i (minst) en kandidatnøkkel.

Supernøkler benyttes til å uttrykke integritetsregler

Formell definisjon av funksjonell avhengighet

- Gitt et relasjonsskjema $R(A_1,A_2,\ldots,A_n)$ og la X, Y være delmengder av $\{A_1,A_2,\ldots,A_n\}$
- Y er **funksjonelt avhengig av** X hvis vi for enhver lovlig instans av R har at hvis instansen inneholder to tupler t_1 og t_2 hvor $t_1[X] = t_2[X]$, så må $t_1[Y] = t_2[Y]$
- I så fall skriver vi $X \rightarrow Y$

Funksjonelle avhengigheter (forts.)

- Ofte snakker vi for korthets skyld om «FDen X $\rightarrow$ Y»
  (der **FD** står for Functional Dependency)
- Vi sier at «Y følger av X»,
  eller at «X bestemmer Y»
- Merk at hvis X er en supernøkkel,
  så holder X $\rightarrow$ Y for enhver Y
- Omvendt: Hvis X $\rightarrow$ Y for enhver Y,
  så er X en supernøkkel
- FDer er integritetsregler

Ekvivalente mengder av FDer

La S, T være to mengder av FDer

- Definisjon:
  Vi sier at **S følger av T** hvis det er slik at enhver instans som oppfyller alle FDer i T, også oppfyller alle FDer i S
- Definisjon:
  Vi sier at **S og T er ekvivalente**
  hvis S følger av T og T følger av S

Armstrongs slutningsregler

1. Refleksivitet:
   Hvis Y er en delmengde av X, så $X \rightarrow Y$
2. Utvidelse: Hvis $X \rightarrow Y$, så $XZ \rightarrow YZ$
3. Transitivitet: Hvis $X \rightarrow Y$ og $Y \rightarrow Z$, så $X \rightarrow Z$

Regelsettet er

- Sunt: Vi kan ikke utlede selvmotsigelser
- Komplett: Alle FDer som kan vises å følge fra en mengde FDer ved bruk av definisjonen av FD, kan også vises ved å bare bruke slutningsreglene

Trivielle FDer

- En FD som følger av refleksivitetsregelen

  «Hvis Y er en delmengde av X, så $X \rightarrow Y$»

  kalles *triviell* fordi den er automatisk oppfylt

- En FD $X \rightarrow Y$ hvor $Y - X \neq \emptyset$, kalles *ikketriviell*

Bevis for utvidelsesregelen:
Hvis $X \rightarrow Y$, så $XZ \rightarrow YZ$

- Anta at $X \rightarrow Y$, og at det finnes to tupler t og u slik at t[XZ] = u[XZ]
- Da har vi spesielt at t[X] = u[X] og at t[Z] = u[Z]
- Siden $X \rightarrow Y$, og t[X] = u[X], er t[Y] = u[Y]
- Da har vi både at t[Y] = u[Y], og at t[Z] = u[Z]
- Sammen gir det at t[YZ] = u[YZ]
- Dermed har vi bevist at $XZ \rightarrow YZ$ holder

Bevis for den transitive regelen: Hvis $X \rightarrow Y$ og $Y \rightarrow Z$, så $X \rightarrow Z$

- Anta at $X \rightarrow Y$, at $Y \rightarrow Z$, og at det finnes to tupler t og u slik at t[X] = u[X]
- Siden $X \rightarrow Y$, og t[X] = u[X], er t[Y] = u[Y]
- Siden $Y \rightarrow Z$, og t[Y] = u[Y], er t[Z] = u[Z]
- Dermed har vi bevist at $X \rightarrow Z$ holder

Tillukningen av X mhp F

- Definisjon av **tillukningen av X mhp F**:
  La X være en mengde attributter og F en mengde FDer
  Da er tillukningen av X med hensyn på F lik mengden av attributter som er bestemt av X
- Notasjon: X^+ (mhp F)
- Egenskaper:
  - $X \rightarrow X^+$
  - Hvis A ikke er med i X^+, så har vi **ikke** at $X \rightarrow A$
- Kjennetegnet på supernøkler (og derfor også kandidatnøkler) er at dens tillukning inneholder **alle** attributtene i relasjonen

Tillukningsalgoritmen

Algoritme som beregner X^+ mhp. F:

1. T=X
2. Sålenge T forandres:
   1. Søk etter $Y \rightarrow Z$ i F hvor Y er en delmengde av T
   2. Hvis slik Y finnes: Legg Z inn i T
      (Dvs. Sett $T = T \cup Z$)
3. $X^+=T$

Tillukningsalgoritmen: Eksempel

- Gitt R(A,B,C,D,E,F) med FDene AB$\rightarrow$C, BC$\rightarrow$AD, D$\rightarrow$E og CF$\rightarrow$B
- Hva er $\{A,B\}^+$?

Test som avgjør om en FD $X \rightarrow Y$ følger fra en mengde FDer F

1. Beregn X^+ mhp. F

2. Hvis alle attributtene i Y er med i X^+, følger $X \rightarrow Y$ av F

   Hvis ikke, følger ikke $X \rightarrow Y$ av F

Hvordan finne alle kandidatnøkler (minimale supernøkler)

Gitt et relasjonsskjema $R(A_1, A_2, \ldots, A_n)$
og en mengde FDer F

1. La S = $\{A_1A_2 \ldots A_n\}$ ($A_1A_2 \ldots A_n$ er en supernøkkel)
La K = { }

2. For hver supernøkkel X i S:
   - For hvert attributt A i X: Sett Y = X-{A} og beregn Y^+ mhp. F
     Hvis $Y^+ = A_1A_2 \ldots A_n$, er X ikke en minimal supernøkkel;
     Y er også en supernøkkel, så legg Y i S
   - Hvis $Y^+ \neq A_1A_2 \ldots A_n$ for alle slike Y, er X en minimal supernøkkel
     I så fall, legg X i K
   - Fjern X fra S

3. Nå er K mengden av alle kandidatnøkler mhp. F

Algoritmen er ikke særlig effektiv, men den virker

Hvordan effektivt finne alle kandidatnøkler

Eksempel:

Gitt F = {AB→DE, C→A, BD→E, AE→B} på R(A,B,C,D,E)

- Siden C ikke er med i noen høyreside i F, må C være med i alle supernøkler (spesielt i alle kandidatnøkler)
- C^+=AC, så C er ikke en supernøkkel
- BC^+=ABCDE, så BC er en kandidatnøkkel
- CD^+=ACD, så CD er ikke en supernøkkel
- CE^+=ABCDE, så CE er en kandidatnøkkel
- For å utvide CD til en nøkkel, har vi to muligheter
  - BCD som inneholder kandidatnøkkelen BC
  - CDE som inneholder kandidatnøkkelen CE
- Altså er BC og CE de eneste kandidatnøklene i R

Hva kjennetegner god relasjonsdatabasedesign?

- Beslektet informasjon legges i samme relasjon
  - Ubeslektet informasjon legges *ikke* i samme relasjon
  - Brudd på dette vil ofte medføre dobbeltlagring av data og dermed forårsake ***oppdateringsanomalier***
- Så lite dobbeltlagring som mulig
  - Plassbehovet minimaliseres
  - Oppdatering forenkles
- Så få «glisne» relasjoner som mulig
  - Plassbehovet minimaliseres
  - Unngår problemer med håndtering av nil-verdier
- Korrekt totalinformasjon kan gjenskapes nøyaktig
  - Ingen falske data genereres

Eksempel: Grossistdatabase versjon 1

Produkt(Kode, Produktnavn, Produsent, #Enheter)

Bestilling(Kode, Kundenr, Navn, Adresse, #Bestilt)

Skranker:

1) Alle verdier i Kode i Produkt skal være unike
2) For hvert par av Kundenr, Kode i Bestilling skal det være bare en mulig verdi av #Bestilt
3) Verdien i Kundenr bestemmer verdiene i Navn og Adresse
4) Kode i Bestilling er fremmednøkkel til (refererer til) verdier i Kode i Produkt

Grossistdatabase versjon 1 med integritetsregler

Produkt(Kode, Produktnavn, Produsent, #Enheter)

Bestilling(Kode, Kundenr, Navn, Adresse, #Bestilt)

Integritetsregler:

1) Kode→Produktnavn Produsent #Enheter (i Produkt)
2) Kundenr Kode→#Bestilt (i Bestilling)
3) Kundenr→Navn Adresse (i Bestilling)
4) Kode i Bestilling er fremmednøkkel til Produkt

   Merk at {Kode} er en kandidatnøkkel i Produkt
   Den er den eneste slike i Produkt og blir derfor primærnøkkel

   Merk at 2 og 3 gir at {Kode, Kundenr} er supernøkkel i Bestilling
   Dette er dessuten en kandidatnøkkel, og den eneste slike i Bestilling, og den blir derfor primærnøkkel

Eksempelekstensjon Bestilling

Bestilling

Kode	Kundenr	Navn	Adresse	#Bestilt
1 2 1	1 1 2	A A B	a a b	3 8 2

Sekundær informasjon

«Navn» og «Adresse» er eksempler på ***sekundær informasjon***

Dette er nyttig informasjon om kundene, men den er unødvendig i en tabell over bestillingene

Oppdateringsanomalier

- **Innsettingsanomalier**
  - Opprettholde konsistente verdier
  - Håndtere sekundær informasjon
  - Håndtere nil i kandidat- og fremmednøkler
- **Slettingsanomalier**
  - Unngå tap av sekundær informasjon
- **Modifiseringsanomalier**
  - Opprettholde konsistente verdier
  - Oppdatere sekundær informasjon

Hvordan unngå oppdateringsanomalier

- Unngå/fjern oppdateringsanomalier og dobbeltlagring ved å splitte (dekomponere) relasjonene slik at dobbeltlagring blir borte!

Dekomposisjon av et relasjonsskjema

- Gitt et relasjonsskjema $R(A_1,A_2,\ldots,A_n)$

  En ***dekomposisjon*** $D = \{R_1,R_2,\ldots,R_m\}$ av R er en samling med relasjonsskjemaer $R_1,R_2,\ldots,R_m$ som er slik at følgende to krav holder:

  – alle attributtene i hver R_k er også attributt i R

  – samtlige attributter $A_1,A_2,\ldots,A_n$ kan gjenfinnes i minst ett av relasjonsskjemaene $R_1,R_2,\ldots,R_m$

Dekomposisjon: Ønskede egenskaper

- Eliminasjon av oppdateringsanomalier
- Rekonstruksjon av den opprinnelige eksensjonen
  - Dekomposisjon er **tapsfri**, dvs. at naturlig join aldri vil gi falske tupler
- Bevaring av funksjonelle avhengigheter

Eksempel: Grossistdatabase versjon 2

Produkt(Kode, Produktnavn, Produsent, #Enheter)

Kunde(Kundenr, Navn, Adresse)

Ordre(Kode, Kundenr, #Bestilt)

Integritetsregler:

- Kode i Ordre er fremmednøkkel til Produkt
- Kundenr i Ordre er fremmednøkkel til Kunde

Eksempelekstensjoner Kunde, Bestilling

Ny modell

Kunde

Kundenr	Navn	Adresse
1	A	a
2	B	b

Ordre

Kode	Kundenr	#Bestilt
1	1	3
2	1	8
1	2	2

Gammel modell

Bestilling

Kode	Kundenr	Navn	Adresse	#Bestilt
1	1	A	a	3
2	1	A	a	8
1	2	B	b	2

Naturlig join

- Vi ønsker å kunne rekonstruere den opprinnelige ekstensjonen
- ***Naturlig join*** er en operasjon der to relasjoner slås sammen til en ved at to og to tupler pares hvis og bare hvis de har like verdier i attributter med likt navn

Kunde ⋈ Ordre

Kunde

Kundenr	Navn	Adresse
1	A	a
2	B	b

Ordre

Kode	Kundenr	#Bestilt
1	1	3
2	1	8
1	2	2

Kunde ⋈ Ordre

Kundenr	Navn	Adresse	Kode	#Bestilt
1	A	a	1	3
1	A	a	2	8
2	B	b	1	2

Eksempel: Grossistdatabase, versjon 3

Produkt(Kode, Produktnavn, Produsent, #Enheter)

Kunde(Kundenr, Navn, Adresse)

Koderegister(Kode, Kundenr)

Antall(Kundenr, #Bestilt)

Integritetsregler:

- Kode i Koderegister er fremmednøkkel til Produkt
- Kundenr i Koderegister er fremmednøkkel til Kunde
- Kundenr i Antall er fremmednøkkel til Kunde

Eksempelekstensjoner Kundereg, Bestilling

Koderegister

Kode	Kundenr
1	1
2	1
1	2

Antall

Kundenr	#Bestilt
1	3
1	8
2	2

Koderegister ⋈ Antall

Kode	Kundenr	#Bestilt
1	1	3
~~1~~	~~1~~	~~8~~
2	1	8
~~2~~	~~1~~	~~3~~
1	2	2

Naturlig join på de to tabellene gir flere tupler enn i den opprinnelige tabellen!
= **Falske tupler**

Retningslinjer for dekomposisjon av relasjoner

- **Motivasjon**: Fjerne oppdateringsanomalier
- **Teknikk**: Dekomponere problemrelasjoner
- **Betingelse**: Opprinnelig ekstensjon skal alltid kunne rekonstrueres nøyaktig ved naturlig join (dekomposisjonen er tapsfri)
- **Problemer**:
  - Hvordan vurdere objektivt om en samling relasjoner er god/dårlig?
  - Hvordan sikre at en dekomposisjon aldri gir falske tupler (er tapsfri)?

Chasealgoritmen

Gitt en dekomposisjon av R(A,B,...) til relasjonsskjemaene S_1, ... , S_k og et sett F med FDer for R. Er dekomposisjonen tapsfri?

1.Lag en tabell med en kolonne for hvert attributt i R og en rad for hver S_i

2.I kolonnen for attributt A, for hver rad i:

- Skriv a hvis A er et attributt i S_i
- Skriv a_i hvis A ikke er et attributt i S_i

3.Så lenge det skjer forandringer i tabellen og det ikke fins en rad uten subskript-verdier:

For hver FD $X \rightarrow Y$, for alle rader i tabellen med lik X-verdi, gjør Y-verdiene like

- Hvis en av Y-ene er en verdi uten subskript, skal denne velges

4.Hvis en rad er uten subskript-verdier, er dekomposisjonen tapsfri, ellers ikke.

Chasealgoritmen: Eksempel 1

- R(A,B,C,D,E)
- FDer: A$\rightarrow$C, B$\rightarrow$C, C$\rightarrow$D, DE$\rightarrow$C, CE$\rightarrow$A
- Dekomposisjon: AD, AB, BE, CDE, AE

Chasealgoritmen, eksempel 2

- Versjon 1:
  Bestilling(Kode, Kundenr, Navn, Adresse, #Bestilt)
  FDer: Kode Kundenr $\rightarrow$ #Bestilt
  Kundenr $\rightarrow$ Navn Adresse
  (Primærnøkkelen følger av disse)
- Versjon 2:
  Kunde(Kundenr, Navn, Adresse)
  Ordre(Kode, Kundenr, #Bestilt)
  FDer: Som for versjon 1
- Versjon 3:
  Kunde(Kundenr, Navn, Adresse)
  Koderegister(Kode,Kundenr)
  Antall(Kundenr, #Bestilt)
  FDer: Som for versjon 1

Normalformer

- Normalformer er et uttrykk for hvor godt vi har lykkes i en dekomposisjon
- Jo høyere normalform, jo færre oppdateringsanomalier
- Det fins algoritmer for å omforme fra lavere til høyere normalformer