trees

language=sql, basicstyle=, numbers=left, numberstyle=, stepnumber=1, numbersep=5pt, backgroundcolor=

, tabsize=2, captionpos=b, breaklines=true, breakatwhitespace=false, title=, keywordstyle=, commentstyle=, stringstyle=, escapeinside=%**), morekeywords=*,...

Notater: INF3100

Veronika Heimsbakk
veronahe@student.matnat.uio.no

20. august 2013

Innhold

1 Intro

Disse notatene er basert på egne notater og forelesningsfoilene til Ellen Munthe-Kaas fra *INF3100 - Databasesystemer* våren 2013, notatene er i meget liten grad basert på pensumboka. Notatene inneholder sikkert veldig mange skrivefeil, uten at dette skal gå ut over innholdet. Merk også at det finnes noen mangler, dette inkluderer bl.a. bagoperasjoner, theta-join, ekvijoin og divisjon.

2 Relasjonsdatabasedesign

2.1 Funksjonelle avhengigheter

$X \to Y$, Y følger av X.

- Refleksiv: hvis $Y \subseteq X$, så $X \to Y$
- Utvidelse: hvis $X \to Y$, så $XZ \to YZ$
- Transitiv: hvis $X \to Y$ og $Y \to Z$, så $X \to Z$

En refleksiv FD kales triviell fordi den automatisk er oppfylt.

Ekvivalente mengder av FDer Hvis X er en supernøkkel, så holder $X \to Y$ for alle Y. Y er like sterk, eller sterkere enn X. Ekvivalente. X følger av Y og Y følger av X.

Hvis det fins et attributt som ikke fins i høyresiden til noen FD, så må den være med i *alle* kandidatnøklene.

Eksempel Gitt $F = \{AB \to DE, C \to A, BD \to E, AE \to B\}$ på relasjonen $R(A, B, C, D, E)$. Her må C være med i alle kandidatnøklene.

$C^+ = AC$, så C er ingen supernøkkel. $BC^+ = ABCDE$, så dette er en kandidatnøkkel. Det samme er CE^+. BC og CE er de eneste kandidatnøklene i R.

2.1.1 Trivielle FDer

En FD som følger av refleksivitetsregelen: «Hvis Y er en delmengde av X, så X → Y» kalles *triviell* fordi den er automatisk oppfylt. En FD X → Y hvor Y − X ≠ ∅, kalles *ikke-triviell*.

2.1.2 Elementære FDer

En FD $X \to A$ kalles elementær dersom

- A er et attributt.
- $X \to A$ er ikke-triviell.
- X er minimal (dvs. at hvis $X \subseteq Y$ og $Y \to A$, så $Y = X$.

En kandidatnøkkel K er elementær hvis det finnes en elementær FD $K \to B$ i R. Iallefall ett attributt B som avhenger av hele K.

2.2 Hvordan finne kandidatnøkler

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

Ettersom C ikke er med i noen høyreside av FDene, må C være med i alle kandidatnøkler.

$C^+ = AC$ C alene er ingen supernøkkel.

$BC^+ = ABCDE$ er en kandidatnøkkel, siden vi får tak i alle attributter ved å gå fra BC.

$CE^+ = ABCDE$ er også en kandidatnøkkel.

BC og CE er de eneste kandidatnøklene til R.

2.3 Dekomposisjon

Gitt en relasjon $R(A_1, A_2, \ldots, A_n)$. En dekomposisjon $D = \{R_1, \ldots, R_m\}$ av R.

1. Alle attributter i hver R_k er også attributt i R.
2. Samtlige attributter $A_1, \ldots, A_n$ kan gjenfinnes i minst en av relasjonene $R_1, \ldots, R_m$.

De samme egenskapene skal bli med i dekomposisjonen. Dekomposisjonen er *tapsfri*, dvs. natural join aldri vil gi falske tupler. Problemer ved dekomposisjon kan være hvordan vurdere objektivt om en samling relasjoner er god/dårlig? Og hvordan sikre at den aldri gir falske tupler?

2.4 Natural join

- Ønsker å kunne rekonstruere den opprinnelige ekstensjonen.
- Slår sammen tupler hvis og bare hvis de har like verdier i attributter med likt kolonnenavn.

Eksempel Gitt tabellene: Kunde(knr, navn, adr) og Ordre(kode, knr, #bestilt).

Kunde

knr	navn	adr
1	A	a
2	B	b

Ordre

kode	knr	#bestilt
1	1	3
2	1	8
1	2	2

Kunde ⋈ Ordre

knr	navn	adr	kode	#bestilt
1	A	a	1	3
1	A	a	2	8
2	B	b	1	2

Figur 1: Eksempel på natural join.

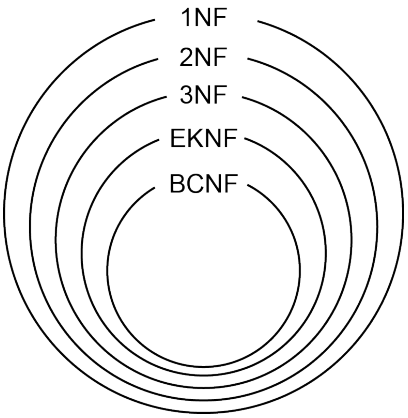
3 Normalformer

Normalformer (NF) brukes til å vurdere når man skal dekomponere.

Felles for alle normalformene i Fig. 2 er at alle integritetsregler er i form av FDer. Når en normalform brytes skal man dekomponere ved å bruke fagins teorem.

Første normalform - 1NF alle domenene består av atomære verdier. Alle relasjoner er automatisk 1NF.

Andre normalform - 2NF En relasjon R er på 2NF hvis *alle* ikke-trivielle FDer i R på formen $X \rightarrow A$, der X er mengden attributter og A er et attributtt i R, som tilfredsstiller *minst ett* av følgende krav:



Figur 2: Normalformene fra 1NF til BCNF. EKNF er kursorisk pensum for INF3100.

1. X er en kandidatnøkkel i R eller inneholder en.
2. A er med i en kandidatnøkkel.
3. Det er ingen kandidatnøkkel som inneholder X.

Tredje normalform - 3NF R er på 3NF hvis *alle* ikke-triville FDer i R på formen $X \to A$ tilfredsstiller *minst ett* av følgende:

1. X er en kandidatnøkkel i R eller inneholder en.
2. A er med i en kandidatnøkkel.

3NF $\subseteq$ 2NF, men skjerping av krav i 3NF. 3NF er også lett å oppnå.

Elementary Key Normal Form - EKNF R er på EKNF hvis *alle* ikke-trivielle FDer i R på formen $X \to A$ oppfyller *minst ett* av følgende:

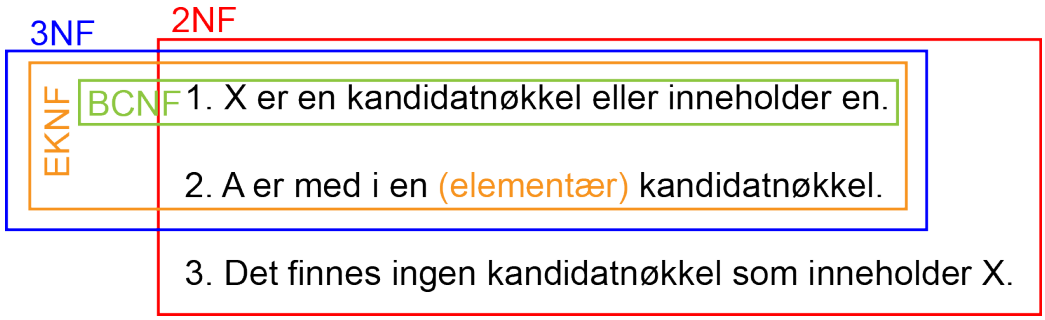
1. X er en supernøkkel i R.
2. A er et attributt i en elementær kandidatnøkkel i R.

EKNF $\subseteq$ 3NF, med skjerping av krav fra 3NF.

Boyce-Codd Normal Form - BCNF R er BCNF hvis *alle* ikke-trivielle FDer i R på formen $X \to A$ oppfyller:

1. X er en supernøkkel i R.

Man kan alltid dekomponere til BCNF, men det er ikke alltid FD bevarende.



Figur 3: Regler for de forskjellige normalformene.

3.1 Høyere Normalformer

3.1.1 4 NF

Alle integritetsregler i form av FDer og MVDer. En relasjon R er på 4NF hvis alle ikke-trivielle MVDer $X \twoheadrightarrow Y$ tilfredsstiller:

1. X er en supernøkkel i R.

4 Flerverdiavhengigheter (MVD)

Flerverdiavhengigheter brukes til å uttrykke integritetsregler utover FDer.

MVDer oppstår når man plasserer to $n : n$-forhold i samme relasjon. Og for å uttrykke integritetsregler.

Eksempel Gitt tabellen Salg(pub, øltype, bryggeri). Regel: dersom en pub selger en øltype fra et bryggeri må den selge denne øltypen fra alle bryggeriene den forhandler med.

Hvis da en pub selger pils fra Mack og fatøl fra Rignes. Må de også selge pils fra Rignes og fatøl fra Mack.

pub	øltype	bryggeri
P1	pils	Mack
P1	fatøl	Rignes
P1	pils	Rignes
P1	fatøl	Mack

Tabellen påtvinges da flere tupler.

Y er fleravhengig av X, $Y \twoheadrightarrow X$. Hvis vi for enhver instans av R har at hvis instansen inneholder to tupler t_1 og t_2 hvor $t_1[x] = t_2[x]$, så finnes også u_1 og u_2.

Hvis $Y \subseteq X$, så $X \twoheadrightarrow Y$, fordi:

X		Z
W	Y	
w_1	y_1	z_1
w_1	y_1	z_2

Alltid to tuple som er like i $x(w, y)$.

Trivielle MVDer X $\twoheadrightarrow$ Y kalles triviell hvis og bare hvis vi enten har at Y $\subseteq$ X eller at XY er samtlige attributter i R.

Ekte MVDer En ikke-triviell X $\twoheadrightarrow$ Y, hvor X $\rightarrow$ Y *ikke* holder, kalles en ekte MVD.

5 Relasjonsalgebra

Definerer en mengde av operasjoner på relasjoner. Gir oss et språk til å beskrive spørsmål om innholdet i relasjonene. Relasjonsalgebra er et **prosedyralt** spørrespråk – sier *hvordan* svaret skal beregnes.

5.1 Algebra

Domene samling av verdier.

Atomære operander konstanter (representerer konkrete verdier i domenet) og variable (representerer vilkårlige verdier fra domenet).

Operatorer tar som argument operander og leverer som resultat en operand.

Uttrykk bygges av atomære operander med operatorer og parenteser.

5.2 Mengdeoperatorer

Union : R $\cup$ S

Snitt : R $\cap$ S

Differanse : R $-$ S

R og S må ha identiske attributtmengder og identiske domener. Før operasjonen utføres, må S ordnes slik at attributtene kommer i samme rekkefølge som i R.

Eksempel på union : {a,b,c} $\cup$ {a,c,d} = {a,b,c,d}

Eksempel på snitt : {a,b,c} $\cap$ {a,c,d} = {a,c}

Eksempel på differanse : {a,b,c} $-$ {a,c,d} = {b}

5.3 Seleksjon

$\sigma_C(R)$ er relasjonen som fås fra R ved å velge ut de tuplene i R som tilfredsstiller betingelsen C. C er et vilkårlig boolsk uttrykk bygget opp fra atomer på formen $op_1 \phi op_2$, der:

- Operandene op_1 og op_2 er enten to attributter i R med samme domene, eller ett attributt i R og en konstant fra dette attributtets domene.
- Operatoren $\phi \in \{=, \neq, <, >, \leq, \geq, \texttt{LIKE}\}$.

5.4 Projeksjon

$\pi_L(R)$ hvor R er en relasjon og L er en liste av attributter i R, er relasjonen som fås fra R ved å velge ut kolonnene til attributtene i L.

- Relasjonen har et skjema med attributtene i L.
- Ingen tupler skal forekomme flere ganger i $\pi_L(R)$.

5.5 Renavning

$\rho_{S(A_1,A_2,\ldots,A_n)}(R)$ renavner R til en relasjon med navn S og attributter $A_1, A_2, \ldots, A_n$. $\rho_S(R)$ renavner R til en relasjon med navn S. Attributtnavnene fra R beholdes.

5.6 Kartetisk produkt

$R \times S$ er relasjonen som fås fra R og S ved å danne alle mulige sammensetninger av ett tuppel fra R og ett tuppel fra S. Vi sier ofte at et tuppel t fra R og et tuppel u fra S blir **konkatenert** til et tuppel $v = tu$ i $R \times S$.

5.7 Naturlig join

$R \bowtie S$ er relasjonen som fås fra R og S ved å danne alle mulige sammensmeltinger av ett tuppel fra R med ett fra S der tuplene skal stemme overens i samtlige attributter med sammenfallende navn.

- Fellesattributtene forekommer bare en gang i de sammensmeltede attributtene.
- Resultatskjemaet har attributtene i R etterfulgt av de attributtene i S som ikke også forekommer i R.

5.7.1 Hengetuppel

Et hengetuppel er et tuppel i en av relasjonene som ikke har noe matchende tuppel i den andre relasjonen. Dette får ingen representant i resultatrelasjonen etter en join.

6 Indeksering

En **indeks** på et attributt A er en datastruktur som gjør det lett å finne de elementene som har en bestemt verdi for A (**søkenøkkelen**). Indeksen er sortert på søkenøkkelen.

6.1 Tette og tynne indekser

En **tett** indeks har ett oppslag for hver verdi av søkenøkkelen. En **tynn** indeks har ett oppslag for hver datablokk.

	Tett	**Tynn**
plass	ett indeksfelt pr. post	ett indeksfelt pr. datablokk
blokkaksesser	mange	få
postaksesser	direkte aksess	må lete innenfor blokken
exist-spørringer	bruker indeksen alene	må alltid aksessere blokken
bruk	overalt	ikke på uordnede elementer
endringer	må alltid oppdateres hvis postrekkefølgen endres	oppdateres bare hvis den første posten i en blokk endres

6.2 Duplikate søkenøkler

Ofte kalt clusterindekser. En klusterindeks kan brukes hvis filen er sortert selv om søkenøkkelen ikke er unik. Eksempel for tett indeks: ett indeksfelt pr. post.

6.3 Sekundærindekser

Hvis filen er usortert, eller sortert på et annet attributt, kan man bruke en **sekundærindeks**. Denne er sortert på søkenøkkelen – rakst søk. Første nivå er alltid tett – høyere nivåer er tynne. Duplikater tillates.

6.4 B^+-trær

Hver node har n søkenøkler og $n + 1$ pekere. Indre noder: alle pekere er til subnoder. Løvnoder: n datapekere og 1 nestepeker. Ingen noder kan være tomme.

Positive ting med B^+-trær er at antall nivåer er vanligvis veldig lavt – typisk 3. Intervallsøking går veldig raskt. Ved stor n er det sjelden nødvendig å splitte/slå sammen noder. Disk I/O kan reduseres ved å holde indeksblokkene i minnet. Et minus med slike trær kan være at et søk må alltid gå fra roten til en løvnode. Dvs. antall blokkaksesser er lik høyden på treet pluss aksessering av selve postene.

6.5 Hashtabeller

Bruker en hashfunksjon fra søkenøkkelen til en arrayindeks med peker videre til hvilken bøtte (bucket) som eventuelt inneholder den aktuelle posten.

Positive sider ved dette er at man får signifikant færre diskoperasjoner enn med vanlige indekser og B-trær. Også raskt søk etter spesifikk søkenøkkel. Negative sider med hashtabeller kan være at flere poster kan føre til flere blokker per bucket. Og dårlig den er dårlig på intervallsøk.

6.6 Indekser i SQL

6.6.1 Syntaks

CREATE INDEX name ON relation$_name(attribute)CREATEUNIQUEINDEXnameONrelat$

Linje nr. 2 definerer en kandidatnøkkel. Man kan ikke angi type indeks, f.eks. B-tre, hash osv. Og man kan ikke angi parametere som loadfaktor, hashstørrelse osv.

6.6.2 Spørringer med flere betingelser

SELECT ... FROM R WHERE a = 30 AND b ¡ 5

Strategi: bruk en indeks, f.eks på a. Finn og *hent* alle postene med a = 30. Søk gjennom disse postene for å finne dem med b < 5.

7 Systemfeil og logging

Vi ønsker at data alltid skal være riktige og nøyaktige. Eksempel på tvilsomme data kan være: (Hansen er 51, Lie er 48, Olsen er 61).

Ansatt

Navn	Alder
Hansen	56
Lie	2489
Olsen	1

7.1 Integritetsregler og konsistens

Integritetsreglerer predikater som data må tilfredsstille. Eksempel: $X \rightarrow Y$ holder i R.

Def. av konsistens Konsistent tilstand: tilfredsstiller alle (statiske) integritetsregler. Konsistent database: databasen er i en konsistent tilstand.

7.2 Transaksjoner

En **transaksjon** er en samling aksjoner som bevarer konsistens.

En viktig antagelse Hvis T starter i en konsistent tilstand, og T eksekverer alene, så vil T avslutte med å etterlate databasen i en kosistent tilstand.

7.3 Klassifikasjon av hendelser

level 1=[level distance=2.5cm, sibling distance=2.5cm] level 2=[level distance=2.5cm, sibling distance=1cm]

bag = [text width=4em, text centered] end = [circle, minimum width=3pt,fill, inner sep=0pt]

[grow=right, sloped] [bag] Hendelser child node Ønskede child node[bag] Uønskede child node[end, label=right: Forventede] edge from parent node[above] node[below] child node[end, label=right: Uventede] edge from parent node[above] node[below] ;

7.4 Transaksjonsoperasjoner

input(x) diskblokk med x $\rightarrow$ primærminnet.

read(x,v) verdien av x i blokken $\rightarrow$ v.

write(x,v) v $\rightarrow$ verdien av x i blokken.

output(x) minneblokk med x $\rightarrow$ disk.

En viktig antakelse I beskrivelsen av operasjonene har vi implisitt antatt at hvert dataelement ligger i en diskblokk/minneblokk. Dette er opplagt riktig hvis dataelementet er en blokk.

7.5 Ufullførte transaksjoner

Dette er hovedproblemet i transaksjonshåndtering. Eksempel med integritetsregel: A = B.

$$\begin{array}{ll} \text{T1:} & A \leftarrow A \times 2 \\ & B \leftarrow B \times 2 \end{array}$$

$$\begin{array}{lll} \text{T1:} & \text{read(A,t);} & t \leftarrow t \times 2; \\ & \text{write(A,t);} & \\ & \text{read(B,t);} & t \leftarrow t \times 2; \\ & \text{write(B,t);} & \\ & \text{output(A);} & \\ & & \textbf{Her skjer feilen.} \\ & \text{output(B);} & \end{array}$$

I primærminnet vil både A og B være 16 etter transaksjonen, men på disk er A satt til 16 og B til 8. Transaksjoner må være **atomære**. Dette gir oss to muligheter:

1. Å utføre alle operasjonene i transaksjoner, eller
2. å ikke utføre noen av operasjonene i transaksjonen.

7.6 Logging

8 Transaksjonshåndtering

En **transaksjon** er en *sekvens* av operasjoner som *bevarer konsistens* i databasen.

ACID-egenskapene

Atomicity enten blir hele transaksjonen utført eller så blir ikke noe av den utført.

Consistency transaksjoner skal bevare konsistens.

Isolation transaksjoner skal ikke merke at andre transaksjoner utføres samtidig med dem selv.

Durability når tansaksjoner er avsluttet, skal effekten av dem være varig og ikke kunne påvirkes av systemfeil.

8.1 Eksekveringsplaner

En **eksekveringsplan** S for en mengde transaksjoner $\{T_1, \ldots, T_n\}$ er en *fletting* av operasjonene i $\{T_1, \ldots, T_n\}$.

Egenskaper hos S:

- Hvert element i S er en operasjon i nøyaktig en av transaksjonene.
- Hver operasjon i en transaksjon er element i S nøyaktig en gang.
- S bevarer rekkefølgen på operasjonene fra hver enkelt transaksjon.

Transaksjonseksempel Integritetsregel: A = B

T1:	read(A);	T2:	read(A);
	A ← A+100;		A ← A ×2;
	write(A);		write(A);
	read(B);		read(B);
	B ← B+100;		B ← B ×2;
	write(B);		write(B);

Eksikveringsplan

T1	T2	A	B
		25	25
read(A); A ← A+100;			
write(A);		125	
read(B); B ← B+100;			
write(B);			125
	read(A); A ← A ×2;		
	write(A);	250	
	read(B); B ← B ×2;		
	write(B);		250
		250	250

Dette er en *serielle* eksikveringsplan, tar en transaksjon om gangen.

Man vil ha eksikveringsplaner som er gode. Begrepet god skal være uavhengig av initialtilstanden og transaksjonssemantikken. Begrepet skal også bare være avhening av lese- og skriveoperasjonene og deres innbyrdes rekkefølge.

8.1.1 Noen begreper

Transaksjon En sekvens av leseoperasjoner $r_i(A)$ og skriveoperasjoner $w_i(B)$.

Eksekveringsplan For transaksjonene $\{T_1, \ldots, T_n\}$: en fletting av $T_1, \ldots, T_n$.

Seriell eksikveringsplan Plan hvor alle operasjonene i en transaksjon fullføres før neste transaksjon startes.

Konflikt i en eksikveringsplan

1. **Lese-skrive-konflikt:** et par av operasjonene av formen $\ldots r_i(A) \ldots w_k(A) \ldots$ eller $\ldots w_i(A) \ldots r_k(A) \ldots$, hvor i $\neq$ k.
2. **Skrive-skrive-konflikt:** et par av operasjoner av formen $\ldots w_i(A) \ldots w_k(A) \ldots$, hvor i $\neq$ k.
3. **Intratransaksjonskonflikt:** Et par av operasjoner av formen $\ldots o_i(A) \ldots o_i(B) \ldots$, hvor o_i er w_i eller r_i.

8.2 Konfliktserialiserbarhet

To eksikveringsplaner S_1 og S_2 kalles **konfliktekvivalente** hvis S_1 kan omformes til S_2 ved en serie ombyttinger av nabooperasjoner som ikke er i konflikt med hverandre. En eksikveringsplan er **konfliktserialiserbar** hvis den er konfliktekvivalent med en seriell eksekveringsplan.

8.2.1 En dårlig eksikveringsplan

$S_D = r_1(A); w_1(A); r_2(A); w_2(A); r_2(B); w_2(B); r_1(B); w_1(B);$

Her har vi en konflikt mellom $w_2(B)$ og $r_1(B)$. Disse kan ikke bytte plass, så S_D kan ikke være konfliktekvivalent med den serielle planen $T_1; T_2$. Vi har også en konflikt mellom $w_1(A)$ og $r_2(A)$ som følgelig heller ikke kan bytte plass. Dermed kan S_D heller ikek være konfliktekvivalent med den serielle planen $T_2; T_1$. S_D er altså ikke konfliktserialiserbar.