Databaser E2001: Kort om mængdelære, algebra og en lille smule logik

Henning Christiansen

4. september 2001

1 Indledning

Dette notat opsummere elementære, matematiske begreber og notation som ligger til grund for relationelle databaser.

Traditionelle lærebøger om databaser forudsætter som regel en matematisk baggrund hos den studerende, som ikke vi ikke kan forudsætte på Datalogi på RUC. Derfor kan en kort introduktion være nyttig, så man f.eks. bliver mindet om, at en relation er et "klassisk" og elementært mængdeteoretisk begreb, som man så indenfor databaser af pragmatiske grunde har tilpasset med særlig notation.

Notatet tjener også til at gøre den videnskabelige litteratur på databaseområdet lidt mere tilgængelig for den studerende, som trænger til at få (gen-)opfrisket sine matematiske forudsætninger.

Den algebraiske måde, som man indenfor databaser behandler mængdelærens begreber på, opleves ofte som usædvanlig selv for studerende, som har et grundigt kendskab til matematik (specielt hvis matematikken er blevet motiveret og motiveret ved problemstillinger indenfor fysik).

I afsnit 2.1 gives en oversigt over elementære mængdeteoretiske begreber: mængder, mængdeprodukter og tupler, relationer og funktioner. Afsnit 3 udpensler algebraiske begreber helt generelt og fremhæver også, hvordan brugen af den sædvanlige mængdelære kan forstås algebraisk. I afsnit 4 giver vi så en kort introduktion til relationel algebra, som er en tilpasning af mængdelærens begreber og notationer, så det egner sig bedre til databasebrug. Hvordan den relationelle algebra så bruges i forhold til databaser og de særlige problemer, det indebærer, kan man læse om i en lærebog om databaser. Et planlagt afsnit 5 vil kort introducere til matematisk logik, som også bruges til at beskrive og ræsonnere om databaser, bl.a. når man betragter integritetsbegrænsninger; afsnitte vil blive tilføjet senere, hvis vi får brug for det på kurset.

2 Mængdelære

2.1 Hvad er en mængde?

En given mængde er en samling af objekter, f.eks. abstrakte, matematiske objekter så som tal, konstantsymboler, tekststrenge eller andre mængder. For givet objekt a og mængde M gælder, at enten er a med i mængden, noteret $a \in M$, eller også er a ikke med, $a \notin M$; vi siger, at a er *medlem* af M såfremt $a \in M$. En mængde M er entydigt karakteriseret ved sine medlemmer, dvs.:

For to mængder M_1, M_2 gælder: $M_1 = M_2$ hvis og kun hvis, for et hvert element $a \in M_1$, at $a \in M_2$, og omvendt.

En mængde bestående af endeligt mange elementer kan noteres ved at angive elementerne adskilt af komma og omsluttet af krøllede paranteser. F.eks. mængden af hele tal fra og med 1 til og med 5 kan skrives $\{1, 2, 3, 4, 5\}$ eller $\{5, 5, 4, 5, 3, 2, 1\}$. Rækkefølge eller gentagelse af elementer er ikke egenskaber ved mængder. For uendelige mængder kan vi en gang imellem anvende notationen suppleret af "...", f.eks. som i $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ der angiver mængden af alle hele tal.

En mængde kan også angives implicit ved at beskrive egenskaber ved dens elementer:

$$\{n \mid n \text{ er et heltal mellem ét og fem, begge incl.}\},$$

og man kan inkludere reference til en overmængde, f.eks. hvis $\mathcal{N}$ refererer til de naturlige tal kan man skrive

$$\{n \in \mathcal{N} \mid 1 \leq n \leq 5\}.$$

Udtrykket efter den lodrette streg kan være formuleret i et mere eller mindre formelt sprog.

Af særlige interesse er den tomme mængde, noteret $\emptyset$ eller sommetider også $\{\}$. Den er karakteriseret ved, at $a \notin \emptyset$ for ethvert a.

Der er forskel på mængden bestående af et element a, dvs. $\{a\}$, og så elementet a selv. Vi har altså følgende egenskaber:

$$a \in \{a\},\ a \neq \{a\},\ b \notin \{a\} \textit{ for ethvert } b \neq a.$$

Vi kan også have mængder af mængder, f.eks. $\{\{1,2\},\{3,4\},\{1,4\},\emptyset\}$. For en mængde M defineres *potensmængden* $\wp M$ som mængdne af alle mulige delmængder af M. F.eks.

$$\wp\{1,2,3\} = \{\emptyset,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$$

Der er nogle mængder, som bruges ofte, og derfor reserverer vi særlige symboler til dem:

$\mathcal{N} = \{0,1,2,\ldots\}$

$\mathcal{Z} = \{\ldots,-2,-1,0,1,2,\ldots\}$

$\mathcal{B} = \{\text{sand},\text{falsk}\}$

$\mathcal{T}$: mængden af teksstrenge.

$\mathcal{U}$: mængden bestående af alting.

2.2 Operationer på mængder

Vi har allerede set operationerne "$\in$" og "$\notin$" som kan anvendes mellem (prospektive) elementer og mængder, og hvor resultatet enten er "sand" eller "falsk". Tilsvarende "$=$" og "$\neq$", som kan anvendes mellem to vilkårlige størrelser, herunder mængder. Til at kombinere mængder for at få nye mængder har vi følgende operationer i den almindelige mængdelære:

	Dansk	Engelsk	Definition
$A \cap B$	fælles-mængde	intersection	mgd. af x hvor både $x \in A$ og $x \in B$
$A \cup B$	forenings-mængde	union	mgd. af x hvor $x \in A$ eller $x \in B$ (eller begge dele)
$A \setminus B$ ell. $A - B$	differens-mængde	difference	mgd. af $x \in A$ som ikke er med i B

Til sammenligning af mængder har vi, udover "$=$" og "$\neq$", følgende, som resulterer i "sand" eller "falsk"

	Dansk	Engelsk	Definition
$A \subseteq B$	delmængde	subset	for alle $x \in A$ gælder $x \in B$
$A \subset B$	ægte delmængde	proper subset	$A \subseteq B$ men ikke $A = B$

2.3 Produktmængder og tupler

Ved en *n-tupel* forstås et struktur af formen $\langle a_1, \ldots, a_n \rangle$, hvor a'erne er vilkårlige størrelser, og hvor rækkefølgen har betydning, dvs. $\langle 1, 2\rangle \neq \langle 2, 1\rangle$ og $\langle 1, 1\rangle \neq \langle 1\rangle$. Bemærk, at $\langle 1\rangle$ er en tupel med et element, og $\langle 1\rangle \neq 1$. En gang i mellem støder man også på 0-tupler, dvs. en tupel, som indeholder 0 elementer og kan noteres $\langle\rangle$. En to-tupel kaldes også et *ordnet par* eller slet og ret et *par*.

Givet en n-tupel t kan vi referere til dens i'te element ved notationen $t\downarrow_i$, dvs. for vilkårlig tupel t har vi:

$$t = \langle t\downarrow_1, t\downarrow_2, \ldots\rangle$$

For en endelig række af mængder $M_1, \ldots, M_n$ defineres deres *produkt*, som noteres $M_1 \times \ldots \times M_n$, som mængden af alle mulige n-tupler $\langle a_1, \ldots, a_n\rangle$, hvor $a_1 \in M_1, \ldots, a_n \in M_n$. Vi har for eksempel:

$$\{\clubsuit, \spadesuit\} \times \{\heartsuit, \diamondsuit\} = \{\langle\clubsuit, \heartsuit\rangle, \langle\clubsuit, \diamondsuit\rangle, \langle\spadesuit, \heartsuit\rangle, \langle\spadesuit, \diamondsuit\rangle\}$$

Hvis vi for eksempel tager produktet af 10 mængder, som hver indeholder 10 elementer, får vi en mængde bestående af 10^{10} tupler. Danner vi produktet af n udgaver af den samme mængde M skriver vi produktet som M^n.

Som notation for tupler benyttes i litteraturen både hak-paranteser som her eller almindelige runde paranteser.

2.4 Relationer

En *relation* over $M_1, \ldots, M_n$, $n \geq 1$, er en delmængde af $M_1 \times \ldots \times M_n$. Hvis nu symbolet R refererer til en relation, kan vi benytte skrivemåden Rt for $t \in R$, hvor man dog som regel erstatter hak-paranteserne for tuplen t med runde og skriver f.eks. $R(\clubsuit, \spadesuit)$.

Ved *ariteten* af R en relation over $M_1, \ldots, M_n$ forstås tallet n, og udtrykket $M_1, \ldots, M_n$ omtales som dens *rang*; ved et *skema* for relationen forstås udtrykket $R(M_1, \ldots, M_n)$, hvor R, $M_1, \ldots, M_n$ skal forstås som udtryk, der refererer til de matematiske objekter, vi tænker på.

For monadiske ($n = 1$) og binære relationer ($n = 2$) benyttes ofte prefiks- og infiksnotation uden paranteser.

Vi kender mange eksempler på binære relationer. Det lighedstegn, vi har benyttet mange gange, refererer til den binære relation over mængden af alting, som består af par hvis første og anden komponent er identiske. Dvs. når vi skriver $a = b$ kunne dette også skrives $=(a, b)$ eller $\langle a, b\rangle \in =$. Når vi skriver udtrykket $a = b$ er det ikke en påstand om at de to bogstaver a og b

er det samme, men at de i den givne kontekst refererer til det samme objekt. Operationerne for medlemsskab og ikke-medlemsskab af mængder kan også forstås som relationer:

$$\in \;=\{\langle x,\{\ldots,x,\ldots\}\rangle \mid \text{for hvilket som helst } x\}$$
$$\notin \;=\{\langle x,M\rangle \mid x\neq y \text{ for ethvert } y\in M\}$$

Der findes mange forskellige relationer over en række mængder $M_1, \ldots, M_n$; $\emptyset$ er den mindste og $M_1 \times \ldots \times M_n$ den største. For 10 mængder, hver med 10 elementer findes der ialt $2^{(10^{10})}$ relationer.

Opgave: Giv argumentet for denne påstand og opskriv tallet $2^{(10^{10})}$ i decimalform.

Endelige relationer kan selvfølgelig opskrives med de krøllede mængdeparanteser fyldt med tupler, men vi kan også benytte en tabelform, hvor hver række angiver en tupel og hver søjle rummer værdier fra kun en af de indgående mængder. Følgende tabel angiver en relation over mængderne $\{1,\ldots,13\}$ og $\{\clubsuit, \spadesuit\, \heartsuit, \diamondsuit\}$.

1	♣
1	♠
1	♡
13	♠
13	♢

Opgave: Giv dit bud på en intuitiv tolkning af relationen $\{1,\ldots,13\} \times \{\clubsuit, \spadesuit\, \heartsuit, \diamondsuit\}$ og relationen i tabellen ovenfor. Skriv begge relationer op vha. sædvanlig mængdenotation.

Her følger en relation over $\mathcal{N}^3$, som man så kan undre sig over hvad skal forestille:

1	1	2
1	2	3
2	3	5
3	5	8
5	8	13

Følgende relation over $\mathcal{T} \times \mathcal{T} \times \mathcal{T} \times \mathcal{N} \times \mathcal{N} \times \mathcal{T}$ giver mere associationer til en databaseanvendelse end to foregående:

Peter	Jensen	Skolegade	7	8000	Århus
Hans	Jensen	Jomfru Ane Gade	3	9000	Ålborg
Børge	Børgesen	Markvej	1	5772	Kværndrup
Peter	Hansen	Snevej	312	3900	Nuuk
Petrine	Hansen	Snevej	312	3900	Nuuk
Peter	Hansen	Morbærvej	8	4200	Slagelse

Opgave: Giv dit bud på en intuitiv tolkning af relationen i tabellen ovenfor og giv i forhold hertil (igen intuitive) krav til nye rækker, som måtte blive tilføjet i tabellen.

2.5 Funktioner

Dataloger forledes ofte til at tro, at en funktion er en proces, som tager noget input og producerer noget entydigt output, men grundlæggende er en funktion blot et specialtilfælde af en relation. Vi definerer her en *funktion* $F\colon M_1, \ldots, M_n \to M_{n+1}, \ldots, M_{n+k}$ som en relation over $M_1 \times \ldots \times M_{n+k}$, med den egenskab, at der for vilkårlig $x_1 \in M_1, \ldots, x_n \in M_n$ findes højst én tupel af formen $\langle x_1, \ldots, x_{n+k}, \ldots \rangle \in F$; i såfald benyttes notationen $F(x_1, \ldots, x_n)$ for at angive tuplen $\langle x_{n+1}, \ldots, x_{n+k} \rangle$; vi omtaler $x_1, \ldots, x_n$ som funktion(sanvendels)ens argumenter og $x_{n+1}, \ldots, x_{n+k}$ som dens resultat.

Funktionens *aritet* er tallet n og dens *rang* er udtrykket $M_1, \ldots, M_n \to M_{n+1}, \ldots, M_{n+k}$

Funktionen F kaldes *total* såfremt der for samtlige kombinationer af $x_1 \in M_1, \ldots, x_n \in M_n$ findes $x_{n+1} \in M_{n+1}, \ldots x_{n+k} \in M_{n+k}$ med $\langle x_1, \ldots, x_{n+k} \rangle \in F$. En funktion, der ikke er total, kaldes *partiel*.

Det ville ikke gøre den store forskel, hvis vi definerede funktioner generelt som binære relationer over $(M_1 \times \ldots \times M_n) \times (M_{n+1} \times \ldots \times M_{n+k})$, men vi har valgt vor formulering for at foregribe funktionel afhængighed i databaser.

Endelige funktioner kan opskrives som tabeller, hvor vi benytte en dobbeltstreg til at skelne argumenter fra resultatet. Følgende tabel angiver en funktion fra $\{1, \ldots, 5\}$ til $\mathcal{N}^2$, som opløfter sit argument til anden og tredje potens.

1	1	1
2	4	8
3	9	27
4	16	64
5	25	125

Opgave: Opskriv en tabel for funktionen fra $\{1,\ldots,5\}$, $\{1,\ldots,5\}$ til $\mathcal{N}$, som adderer sine to argumenter.

Opgave: Hvordan kan man afgøre om en tabel med en dobbeltstreg faktisk repræsenterer en funktion?

Opgave: Betragt følgende tabel, som, når vi tager definitionen bogstaveligt, også beskriver funktioner. Forklar funktionen intuitivt og giv eksempler på funktionsnvendelse og hvilke værdier, de repræsenterer.

1	abe
4	gris
2	ko
7	hest

Tupler som funktioner

Når vi kommer til at beskrive relationel algebra i afsnit 4 nedenfor vil vi ændre notationen for tupler, således, at vi kan benytte sigende navne for de enkelte komponenter af en tupel. Sådan som vi har beskrevet n-tupler indtil nu, har mængden $\{1,\ldots,n\}$ fungeret som navngivere for komponenterne. Det at danne tupler af elementer og det at udvælge komponenter af tupel kan forstås som funktioner:

$$\langle -,\ldots,- \rangle \colon M_1 \times \ldots \times M_n \to M_1 \times \ldots \times M_n$$
$$\downarrow \colon M_1 \times \ldots \times M_n \times \{1,\ldots,n\} \to M_1 \cup \ldots \cup M_n$$

For en given tupel $t \in M_1 \times \ldots \times M_n$ kan udvælgelse af dens komponenter forstås som en funktion $t\!\downarrow \colon \{1,\ldots,n\} \to M_1 \cup \ldots \cup M_n$. Betragt som eksempel tuplen $t = \langle \text{Peter}, \text{Jensen}, \text{Skolegade}, 7, 8000, \text{Århus} \rangle \in \mathcal{T}^3 \times \mathcal{N}^2 \times \mathcal{T}$. Her kan $t\!\downarrow$ opskrives som følger.

1	Peter
2	Jensen
3	Skolegade
4	7
5	8000
6	Århus

(Hvis vi vender denne tabel på passende vis, kan vi se, at tallene 1,...,7 kommer til at fungere som en slags kolonnenavne.) Tuplingsfunktionen $\langle -,\ldots,- \rangle$ vil i princippet også kunne skrives som tabel, men vil være meget, meget stor (med mindre de indgående mængder er meget, meget små).

Nøgler i relationer

Lad os betragte en relation R over $M_1, \ldots, M_n$. Vi siger, at R er *funktionel* i $\{1,\ldots,i\}$ såfremt der findes et i og en funktion $R_{\{1,\ldots,i\}}\colon M_1,\ldots,M_i \to M_{i+1},\ldots,M_n$, hvor $R_{\{1,\ldots,i\}}$ betragtet som relation er sammenfaldende med R. Vi siger også, at $\{1,\ldots,i\}$ er en *nøgle* til R.

Nøgler eller funktionalitet i en relation kan angives i dens tabelform ved at tegne én af de lodrette streger op som en dobbelt streg. Vi så i afsnit 2.4 en relation med noget som mindede om navne og adresser. Vi kan ikke bruge $\{1\}$ som nøgle, da der er flere rækker som starter med "Peter", og heller ikke $\{1,2\}$ da der er to rækker med "Peter Hansen", men $\{1,2,3\}$ kan bruges:

Peter	Jensen	Skolegade	7	8000	Århus
Hans	Jensen	Jomfru Ane Gade	3	9000	Ålborg
Børge	Børgesen	Markvej	1	5772	Kværndrup
Peter	Hansen	Snevej	312	3900	Nuuk
Petrine	Hansen	Snevej	312	3900	Nuuk
Peter	Hansen	Morbærvej	8	4200	Slagelse

Det er vigtigt at holde sig for øje, at nøglen $\{1,2,3\}$ her er baseret på ren og skær observation af de faktiske data i relationen og ikke afspejler en designbeslutning eller en egenskab ved verden. Vi observeret også at $\{1,2,3,4\}$ kan bruges som nøgle i den faktiske tabel og også $\{1,2,3,4,5\}$, $\{1,2,3,4,5,6\}$ er en nøgle.

De "funktionelle afhængigheder" og nøgler vi kan indfange på denne måde er for begrænsede til det, vi er interesseret i i forbindelse med databaser, da det bl.a. hænger på, hvilken rækkefølge komponenterne står i. Når vi senere i den relationelle algebra indfører navne til de enkelte felter bliver det langt mere fleksibelt at karakterisere sådanne afhængigheder.

Opgave: Vi forestiller os, at tabellen ovenfor er en del af en database, som fremover vil blive udvidet med nye tupler baseret på data, som er korrekte i forhold til virkeligheden. Vil der være nogen fornuftig nøgle, som vil gælde for alle kommende udgaver af relationen? Kan man omordne rækkerne og få nøgler ud af det? Prøv at smide forskellige søjler væk og byt rundt på de tilbageværende og se, om der opstår meningsfyldte nøgler.

Opgave: Der gælder generelt, at hvis $\{1,\ldots,i\}$ er nøgle for en relation R med aritet n, da er $\{1,\ldots,j\}$ også nøgle for R, $i < j \leq n$. Giv et argument for, hvorfor det forholder sig således. Argumentér også for, at $\{1,\ldots,n\}$ altid er en nøgle for R.

Et eksempel på en “naturlig” nøgle vil optræde i en tabel hvis første søjle repræsenterer CPR-numre, og hvis øvrige søjler er personlige oplysninger så som navn, adresse osv. Til givet CPR-nummer kan der højst være én tupel i funktionen, nemlig “kartotekskortet” svarende til én bestemt person.

Karakteristiske funktioner

En given mængde M er entydigt identificeret ved sin *karakteristiske funktion* $I_M : \mathcal{U} \to \mathcal{B}$ defineret ved:

$$I_M(x) = \begin{cases} \text{sand} & \text{hvis } x \in M \\ \text{falsk} & \text{ellers} \end{cases}$$

Som specialtilfælde heraf, kan en relation R over $M_1, \ldots, M_n$ forstås som en funktion $I_R : M_1, \ldots, M_n \to \mathcal{B}$ defineret ved:

$$I_R(x_1, \ldots, x_n) = \begin{cases} \text{sand} & \text{hvis } R(x_1, \ldots, x_n) \\ \text{falsk} & \text{ellers} \end{cases}$$

Karakteristiske funktioner er dog mest af ren matematisk interesse, da de ikke egner sig som til at opskrive i tabelform. Hvis vi nu begrænser den karakteristiske funktion for en mængde M til ikke at tage udgangspunkt i hele $\mathcal{U}$ men blot en eller anden endelig grundmængde M^+, vil den karakteristiske funktion i tabelform svare til en bitvektor.

Skal vi repræsentere en relation over $\{1, \ldots, 13\} \times \{\clubsuit, \spadesuit\, \heartsuit, \diamondsuit\}$ vil tabellen få 52 rækker, og hver sådan tabel svarer til en vektor på 52 bits. Hvis vi nu vores computer understøtter aritmetik på 64 bits tal og har bitvise logiske operationer, kan mængdeoperationerne beskrevet i afsnit 2.2 implementeres meget effektivt.[1]

3 Algebra

Ordet “algebra” er her ikke et mængdeord, som relaterer til en bestemt matematisk disciplin, men et tingsord, som man kan sætte “en” eller “flere” foran. Vi har allesammen erfaring med den algebra, som består af hele tal udstyret med de fire regningsarter.

I denne algebra opererer vi med nogle konstantsymboler, f.eks. “2”, og nogle operatorsymboler f.eks. “+”, og de kan sættes sammen efter kendte regler til at danne udtryk; da den tekstlige præsentation ofte skjuler hvordan

[1] Programmeringssproget Pascals “set types”, som man også genfinder i nyere programmeringssprog, er implementeret på denne måde.

udtrykkene er bygget op, hjælper vi til med paranteser, men grundlæggende er de ikke en del af algebraen (vi kan undvære dem helt, jvf. omvendt polsk notation, eller udtryk repræsenteret som syntakstræer).

Hvert konstantsymbol i algebraen svarer til en matematisk størrelse, og hver operation svarer til en matematisk funktion. Symbolet "2" svarer almindeligvis til det matematiske to-tal $\in \mathcal{Z}$, og "+" svarer til den funktion $\mathcal{Z} \times \mathcal{Z} \to \mathcal{Z}$ som til to tal tilordner deres sum. Vi kan udvide algebraen med yderligere operatorsymboler så som "=" og ">", som svarer til de forventede funktioner $\mathcal{Z} \times \mathcal{Z} \to \mathcal{B}$, og vi kan tilføje de kendte boolske operatorer "$\wedge$", "$\vee$", osv., med deres forventede betydninger. Betragt som eksempel udtrykket "2+2=4"; vi konstaterer det er sat korrekt sammen og at det repræsenterer værdien "sand", hvorimod "(1=2)+1" ikke har noget med den velkendte algebra at gøre.

Dette er princippet i algebraer generelt: Vi har nogle mængder og et udvalg af konstant- og operatorsymboler, som refererer til værdier og funktioner over disse mængder.

For at forstå baggrunden for relationel algebra, som vi kommer til senere, kan det være nyttigt at se på algebraer generelt — også således at læseren kan være forberedt på terminologien, som anvendes i den videnskabelige litteratur.

Vi starter med at definere en *signatur* som består af:

- et antal *sorter* eller *typer*, som kan være *simple* hvilket vil sige, at de blot er symbolske navne, eller *strukturerede typer* som er opbygget som selvstændige udryk.

- et antal *konstantsymboler* hver knyttet til en type, og

- en antal *operatorsymboler* med given *rang*
  $\tau_1, \ldots, \tau_n \to \tau,$
  hvor τ'erne er typer.

En signatur svarer så nogenlunde til det, som i relationel algebra kaldes et skema og indenfor programmeringssprog en grammatik. I dette afsnit nøjes vi med at se på simple typer og vender tilbage til strukturerede typer i afsnittet om relationel algebra.

Eksempel: Vi definerer en signatur Σ_1 svarende til regning med tal, jvf. indledningen til dette afsnit. Σ_1 indeholder typerne $\mathcal{Z}$, $\mathcal{B}$, konstantsymbolerne $\ldots$, -1, 0, 1, $\ldots$ af type $\mathcal{Z}$ og "sand", "falsk" af type $\mathcal{B}$. Nogle af operatorsymbolerne i Σ_1 er $+ : \mathcal{Z}, \mathcal{Z} \to \mathcal{Z}$ og $>: \mathcal{Z}, \mathcal{Z} \to \mathcal{B}$.

En *term* over given signatur Σ er et udtryk bygget ud fra konstanterne og operatorsymbolerne, således at typerne stemmer, og vi taler også om *typen af* en given term, hvilket er defineret på sædvanlig måde. Vi kan definere dette præcist på følgende måde:

- Et konstantsymbol c af type τ er en term af type τ.
- Hvis $t_1, \ldots, t_n$ er termer af type $\tau_1, \ldots, \tau_n$ og op er et operatorsymbol af rang $\tau_1, \ldots, \tau_n \rightarrow \tau$, da er $op(t_1, \ldots, t_n)$ er term af type τ.
- Intet andet er termer.

I praksis skriver vi termer med en varieret notation, f.eks. infiks, så vi skriver "2+2=4" i stedet for "=(+(2,2),4)".

Eksempel: Under antagelse af signaturen Σ_1 er "2+2" en term af type $\mathcal{Z}$, og "2+2=4" og "2+2=5" er termer af type $\mathcal{B}$.

En Σ-*algebra* $\mathcal{A}$ for en signatur Σ består af

- for hver type τ i Σ en mængde $\tau^{\mathcal{A}}$,
- for hvert konstantsymbol c af type τ, et element $c^{\mathcal{A}} \in \tau^{\mathcal{A}}$, og
- for hvert operatorsymbol $f\colon type_1, \ldots, type_n \rightarrow type_{n+1}, \ldots, type_{n+1}$ en funktion $f^{\mathcal{A}}\colon type_1^{\mathcal{A}}, \ldots, type_n^{\mathcal{A}} \rightarrow type_{n+1}^{\mathcal{A}}, \ldots, type_{n+k}^{\mathcal{A}}$

En Σ-*algebra* $\mathcal{A}$ for en signatur Σ udvides på naturlig måde til en funktion, som til en vilkårlig Σ-term t tilknytter en værdi, som noterer $t^{\mathcal{A}}$. For nogle signaturer, f.eks. Σ_1, udvælger vi en bestemt algebra som den *kanoniske algebra*, og som definerer "standardbetydningen".

Eksempel: Den kanoniske algebra N for Σ_1 defineres, så typen $\mathcal{Z}$ knyttes til mængden af alle hele tal, $\mathcal{B}$ til en mængde med to sandhedsværdier, hver talkonstant til det naturlige tal, vi tænker på, og hver operator til den funktion, vi tænker på. Vi forventer, at værdien af termen "$2 + 2$" i N, dvs. $(2 + 2)^N$, er den samme som værdien af termen "4", og at værdien af "$2 + 2 = 4$" i N svarer til "sand". Bemærk, at funktionen repræsenteret ved t^N ikke er fuldstændig, jvf. "1/0".

Vi kan have flere forskellige algebraer for den samme signatur Σ, bl.a. *termalgebraen* $T\Sigma$, hvor hver type τ relateres til en mængde af termer af type τ, konstantsymbolerne til sig selv, og operatorsymbolerne til funktioner, som bygger termer ud fra deltermer.

Eksempel: I $T\Sigma_1$ repræsenterer "2+2" og "4" to forskellige værdier (nemlig de to respektive termer), og termen "2+2=4" har ikke tilknyttet nogen værdi svarende til "sand" eller "falsk", men repræsenterer slet og ret sig selv, dvs. den meningsløse term "2+2=4".

En algebra kan også definere en *abstrakt fortolkning*, som så at sige samler værdierne i den kanoniske algebra i grupper.

Eksempel: For Σ_1 kunne vi f.eks. associere $\mathcal{Z}$ med en mængde $\{\text{lige}, \text{ulige}\}$, $\ldots, -4, -2, 0, 2, 4, \ldots$ med "lige", $-3, -1, 1, 3, 5, \ldots$ med "ulige" og til "+" en funktion som afbilder "lige, lige" og "ulige, ulige" over i "lige" og de sidste to kombinationer over i "ulige". Vi kan opskrive denne funktion for plus på følgende måde:

+:

lige	lige	lige
lige	ulige	ulige
ulige	lige	ulige
ulige	ulige	lige

For "$*$" (multiplikation) synes det fornuftigt at afbilde "ulige, ulige" over i "ulige" og alle andre kombinationer over i "lige". Vi kan ikke her knytte en interessant funktion til "$>$" eller "/".

Eksempel: Lad Σ_2 være en udvidelse af Σ_1 med to symboler, begge skrevet som "?", ét for hver at typerne $\mathcal{Z}$ og $\mathcal{B}$. Dette kan være relevant, når vi overvejer optimeringer i en compiler for et programmeringssprog, således at så mange udtryk som muligt udregnes på oversættelsestidspunktet, idet "?" svarer til et ukendt argument. En passende algebra associerer $\mathcal{Z}$ med mængden af naturlige tal udvidet med "ukendt" og analogt for $\mathcal{B}$. Talsymbolerne afbildes over i deres kanoniske værdier og de til operatorsymbolerne associerede funktioner udvider de kanoniske, således at når et af argumenterne er "ukendt" bliver resultat "ukendt". I denne algebra kan vi regne os frem til at "$1 + 2 >?$" har værdien "ukendt". Til det formål, vi har i tankerne med denne algebra, kan vi forbedre betydningen af gange-operationen ved at vedtage, at "ukendt" ganget med nul giver nul (og ikke "ukendt"), således at "$(1 + 2*?) * 0 > 1$" evalueres til en værdi svarende til "falsk".

Mængder som en algebra

I afsnit 2.1 og 2.2 beskrev vi, hvad vi forstod ved mængder og de operationer, vi kan udføre på dem. Vi kan præcisere dette som en algebra. Signaturen Σ_3 indeholder typerne *Mængde*, *Element* og *Bool*. Symbolerne $\mathcal{N}$, $\mathcal{Z}$, $\mathcal{B}$,

$\mathcal{T}$, $\mathcal{U}$ og $\emptyset$ er nu konstantsymboler af type *Mængde*, og *Element* har konstantsymbolerne svarende til de sædvanlige talkonstanter, teksstrenge osv. Bemærk, at *Bool* er en særskilt type, som i denne sammenhæng er forskellig fra konstanten $\mathcal{B}$ af type *Mængde*. Her følger nogle af operatorsymbolerne i signaturen:

$\in$: *Element*, *Mængde* $\rightarrow$ *Bool*

$\notin$: *Element*, *Mængde* $\rightarrow$ *Bool*

$\cup$: *Mængde*, *Mængde* $\rightarrow$ *Mængde*

$\subseteq$: *Mængde*, *Mængde* $\rightarrow$ *Bool*

$\{\ldots\}$: *Element*, $\ldots$, *Element* $\rightarrow$ *Mængde*

$\cdots \times \cdots \times \cdots$: *Mængde*, $\ldots$, *Mængde* $\rightarrow$ *Mængde*

$\langle \cdots, \cdots, \cdots \rangle$: *Element*, $\ldots$, *Element* $\rightarrow$ *Element*

– en usynlig operator med aritet *Mængde* $\rightarrow$ *Element* (så vi kan have mængder af mængder)

Den kanoniske algebra for Σ_3 afbilder konstantsymbolerne over i deres respektive elementer eller mængder og operatorsymbolerne over i de funktioner, vi forventer.

Opgave: Udvid Σ_3 med konstanter "?" i hver af de tre typer, og overvej en fornuftig tilpasning af de til operatorsymbolerne tilknyttede funktioner i algebaren.

I denne version af algebraiseret mængdelære benyttede vi én type til alle elementer og én type til samtlige mængder. På denne måde udnytter vi ikke potentialet i mængdebegrebet fuldt ud. Eksempelvis vil vi acceptere termen $\langle 1, 2\rangle \in \mathcal{N} \times \mathcal{N} \times \mathcal{B}$ med type *Bool*. Intuitivt er der noget galt idet ingen to-tupel kan være med i et mængdeprodukt af tre mængder — altså (endnu intuitivt) en form for typefejl. I den relationelle algebra introduceret i det følgende går vi mere detaljeret til værks, idet vi bruger strukturerede typer, som rummer information om, hvordan de indgående mængder er konstrueret.

Det vi har ønsket at illustere her er at operationer på mængder og operationer på tal grundlæggende handler om det samme: Vi har nogle værdier, dette være sig tal eller mængder eller noget helt tredje, og nogle operationer, vi kan udføre på dem, f.eks. "+" eller "$\cup$", og får nye værdier ud af det. Lige såvel som udtryk kan evaluere til talværdier, kan andre udtryk evaluere til mængder.

4 Relationel algebra: Mængdelære med typer og navngivning

Vi introducerer her grundprincipperne i den relationelle algebra og nogle få elementære operationer. Kig i din databasebog for et større udvalg af relationelle operationer, og hvordan de kan bruges i forbindelse med databaser.

Vi så tidligere, hvordan mængdelæren kan opfattes som en algebra, og denne mængdelære omfatter også operationer på relationer (som jo blot er en slags mængder). Relationel algebra tilføjer ikke noget grundlæggende ny, men giver nogle begrænsninger og en notation, som'er mere velegnet i forhold til at beskrive databaser.

Tupler og mængdeprodukt med navngivne komponenter

I forhold til at anvende mængdelærens begreber i forhold til databaser er det praktisk at benytte navne til at referere til tupler. Vi indfører straks dette inden vi begynder at formalisere en algebra. Disse navne kaldes *attributter* og vi antager en eller anden mængde $\mathcal{A}$ af attributter, som er symbolske konstanter.

For en endelig delmængde $A \in \mathcal{A}$ af attributer definerer vi en A-*tupel* som en total funktion fra A over i $\mathcal{U}$. Vi benytter en notation, som minder om den gamle notation for tupler. Antag som eksempel $A = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ og følgende er da et eksempel på en A-tupel:

$$\langle \mathrm{a}{:}\,10, \mathrm{b}{:}\,20, \mathrm{c}{:}\,30 \rangle$$

I den gamle notation benyttede vi positionen i tuplen som identifikation af den enkelte komponent, men her bruger vi altså attributter. Da tupler (de "nye") er defineret som funktioner fra attributter, er rækkefølgen i notationen i følge sagens natur underordnet, så følgende skrivemåder er alle ækvivalende med ovenstående.

$$\begin{array}{c}
\langle \mathrm{a}{:}\,10, \mathrm{c}{:}\,30, \mathrm{b}{:}\,20 \rangle \\
\langle \mathrm{b}{:}\,20, \mathrm{a}{:}\,10, \mathrm{c}{:}\,30 \rangle \\
\langle \mathrm{b}{:}\,20, \mathrm{c}{:}\,30, \mathrm{a}{:}\,10 \rangle \\
\langle \mathrm{c}{:}\,30, \mathrm{a}{:}\,10, \mathrm{b}{:}\,20 \rangle \\
\langle \mathrm{c}{:}\,30, \mathrm{b}{:}\,20, \mathrm{a}{:}\,10 \rangle
\end{array}$$

Til at udvælge de enkelte komponenter i en tupel benytter vi en priknotation, som vi kender det fra programmeringssprog. Hvis t er tuplen, vi har vist seks gange ovenfor, har vi $t.\mathrm{a} = 10$, $t.\mathrm{b} = 20$ og $t.\mathrm{c} = 30$.

Mængdeprodukter skal nu defineres som mængder af ensartede tupler: Lad $M_1, \ldots, M_n$ være mængder og $A = \{a_1, \ldots, a_n\}$ være en mængde af attributter. *Mængdeproduktet*

$$a_1 \colon M_1, \ldots, a_n \colon M_n$$

består af samtlige A-tupler t, hvor $t.a_i \in M_i$ for alle $i = 1, \ldots, n$. Det er klart at rækkefølgen af de indgående par af attribut og mængde er ligegyldig.

Relationer defineres som tidligere som delmængder af et mængdeprodukt. Indenfor relationelle databaser taler man om *skemaet* for en relation, som angiver, hvilket mængdeprodukt, man arbejder indenfor. Her er vi nødt til at være en smule stringente og tænke lidt algebraisk idet skemaer gerne skulle kunne skrives i en kort form. Vi benytter derfor navne på de indgående mængder, da et skema ellers måtte indeholde mængderne på en eller anden mystisk vis. Så for hver mængde M_i som kan indgå i mængdeprodukter antage vi en type (dvs. et symbolsk navn) for den, som vi her skriver som $\mathcal{M}_i$.

Vi definerer således: En *relation* med skema $a_1 \colon \mathcal{M}_1, \ldots, a_n \colon \mathcal{M}_n$ er en endelig delmængde af mængdeproduktet $a_1 \colon M_1, \ldots, a_n \colon M_n$.

Forudsætningen om at relationer her skal være endelige skyldes naturligvis at de skal kunne repræsenteres i noget der minder om tabelform på en hard-disk. Relationer kan altså altid skrives som tabeller, og i relationelalgebra-forstand er "=" altså ikke en relation (af den slags som man kan udsætte for de relationelle operationer, som indføres nedenfor).

Relationer opskrives (som tidligere) i tabelform, hvor vi nu benytter attributterne som kolonnenavne og hver tupel optræder som en række. Følgende tabel viser en relation med skema a: $\mathcal{N}$, c: $\mathcal{N}$, b: $\mathcal{N}$

a	b	c
10	20	30
17	18	19
34	43	56
100	200	5000

Igen er det klart, at vi kan bytte om på rækkerne lige som vi lyster og det samme med rækkerne for den sags skyld uden at vi ændrer på, hvilket relation, som er angivet, f.eks.:

c	b	a
5000	200	100
30	20	10
56	43	35
19	18	17

4.1 En algebra om relationer

Algebraen skal forestille at modellere de operationer, vi kan foretage på en given database i et tidsrum, hvor dens indhold ikke ændrer sig. Termer i algebraen er relationelle udtryk, som svarer til de forespørgsler, man kan stille, og svarene, man får tilbage er relationer — dvs. mængder af tupler — som er den værdi, som algebraen tilordner udtrykket.

Vi antager, der foreligger et *databaseskema*, af formen:

$R_1(S_1)$

$\ldots$

$R_n(S_n)$

hvor R'erne er forskellige *relationsnavne* og S'erne er *relationsskemaer* af den form, vi så ovenfor. Der kan referes til at antal grundmængder — i database-terminologi *domæner* — i relationsskemaerne. Vi antager følgende navne er tilgængelige: $\mathcal{N}$, $\mathcal{Z}$, $\mathcal{B}$, $\mathcal{T}$ med de betydninger vi gav dem tidligere, samt endelige mængder af symboler, som skrives med sædvanlig mængdenotation, f.eks. $\{\text{abe}, \text{ko}, \text{gris}\}$ som svarer til en mængde af tre forskellige symboler. Vi tillader os at genbruge domænet $\mathcal{A}$ af attributter som udgangspunkt for disse eksplicitte mængder.

For at undgå tekniske problemer antager vi, at de attributter, som optræder i databaseskemaet benyttes konsekvent for de samme mængder i de forskellige relationsskemaer.[2]

Eksempel: Følgende er et eksempel på et databaseskema.

Kasse(højde: $\mathcal{N}$, bredde: $\mathcal{N}$, dybde: $\mathcal{N}$, farve: {rød,grøn,blå})

Dåse(højde: $\mathcal{N}$, diameter: $\mathcal{N}$, farve: {rød,grøn,blå})

Cylinder(højde: $\mathcal{N}$, diameter: $\mathcal{N}$, farve: {rød,grøn,blå})

Det ville ikke være tilladt, hvis Dåse-skemaet havde indeholdt "farve: {rød,grøn,blå,gul}" eller "højde $\mathcal{Z}$". Bemærk, at de to relationsnavne Dåse og Cylinder har identiske skemaer.

Da dette nu drejer sig om databaser, må vi forestille os, at der er noget indhold i den, og vi antager derfor, at der eksisterer én bestemt relation IR_i for hver R_i i databaseskemaet, som naturligvis lever op til skemaet S_i.

[2] Man kan ikke forvente således i andre formuleringer af relationel algebra, som du f.eks. kan finde i din databaselærebog.

Det er normalt, at man omtaler IR_i som en *instans* af R_i, og når vi omtaler "relationen R_i" hentyder det til den aktuelle instans af R_i, altså IR_i. Samlingen af disse IR_i omtaler vi som en *instans* af databasen eller databasens *tilstand* (eller slet og ret "databasen"). Man skal være forberedt på at databaselitteraturen ofte er en smule sløset i sin terminologi, eksempelvis benyttes sprogbrugen "relationen R_i" også i betydningen af en fysisk tabel, som man lægge ting ned og tilføje og fjerne over tid.[3]

Eksempel: Følgende tabeller beskriver en database(tilstand) for databaseskemaet i det foregående eksempel.

Kasse:

højde	bredde	dybde	farve
110	25	35	rød
10	10	10	blå
50	40	17	blå

Dåse:

højde	diameter	farve
28	7	grøn
10	50	rød
1	30	rød
5	5	rød

Cylinder:

højde	diameter	farve
28	7	blå

Relationerne omtalt som $R_1, \ldots, R_n$ fungerer som de basale byggesten i den relationelle algebra, dvs. som dens konstantsymboler; i det konkrete eksempel er Kasse, Dåse og Cylinder algebraens konstanter, og deres respektive værdier er angivet ved de tre tabeller ovenfor.

Algebraens typer er alle mulige relationsskemaer, dvs. vi har en i princippet uendelig stor mængde af strukturerede typer; hvordan det kommer til at spænde af kan vi se i følgende definition af signaturen for den relationelle algebra. Desuden antager vi typerne $\mathcal{N}$, $\mathcal{Z}$, $\mathcal{B}$, $\mathcal{T}$ og $\mathcal{A}$ således at vi kan benytte deres normale konstantsymboler i betingelser.[4]

[3] Det er helt analogt til, når man i forhold til et Java-program omtaler en heltalsvariabel som et heltal.

[4] Bemærk, at vi benytter symbolet $\mathcal{N}$ både som noget, der kan indgå i et skema og som en selvstændig type; ditto for $\mathcal{Z}$, $\mathcal{B}$, $\mathcal{T}$ og $\mathcal{A}$. Dette må endelig ikke misforstås på den måde, at f.eks. $\mathcal{N}$ fungerer som en konstant der repræsenterer en monadisk relation, hvilket ikke er muligt, da mængden som $\mathcal{N}$ henviser til er uendelig, og relationer skal pr. definition være endelige.

Vi kan altså præcisere, i det vi antager et databaseskema $R_1(S_1), \ldots, R_n(S_n)$, at den relationelle algebra indeholder følgende konstanter.

- R_i er et konstantsymbol med type S_i, $i = 1, \ldots, n$; algebraen associerer disse konstantsymboler med relationerne IR_i, $i = 1, \ldots, n$.
- For vilkårligt skema S findes en konstant $\emptyset$ af type S; algebraen associerer disse konstantsymboler med den tomme mængde.
- De konstanter, vi plejer at henføre til typerne $\mathcal{N}$, $\mathcal{Z}$, $\mathcal{B}$, $\mathcal{T}$ og $\mathcal{A}$; algebraen associerer disse konstantsymboler (f.eks. "1") med deres sædvanlige værdier (f.eks. den matematiske èt-tal).

Vi antager uden yderligere armbevægelser, at algebraen forsyner $\mathcal{N}$, $\mathcal{Z}$, $\mathcal{B}$, $\mathcal{T}$ og $\mathcal{A}$ med et tilstrækkeligt udvalg operationer.

Signaturen indeholder de sædvanlige mængdeoperationer, som kan anvendes på relationer. Da resultatet af en sådan operation også skal være en relation, må vi kræve, at de indgående relationerne har samme type:

- For samtlige skemaer S er $\cup : S, S \to S$ en operation. Den tilhørende funktion i algebraen tager to relationer A og B med fælles skema S og producerer en ny relation svarende til den klassiske forening $A \cup B$.
- Tilsvarende for $\cap$ og $\setminus$.
- For samtlige skemaer S er $\subseteq: S, S \to \mathcal{B}$. Den tilhørende funktion i algebraen tager to relationer A og B med fælles skema S og tester klassisk mængde-indeholdt $A \subseteq B$.
- Tilsvarende for $\subset$ og $=$ til test af identitet for relationer med samme skema.[5]

Opgave: Hvilke af følgende tegnkombinationer er korrekte udtryk i den relationelle algebra? Og hvad er i givet fald deres type (dvs. skema) og værdi:

- Kasse $\cup$ Dåse $\cup$ Cylinder
- Dåse $\cup$ Cylinder
- Dåse $\cap$ Cylinder

[5] Helt analogt til programmeringssprog: "1=2" er typisk acceptabel mens "1=true" typisk er en syntaksfejl.

- Dåse $\subseteq$ Cylinder
- Dåse $\subseteq$ (Cylinder $\cup$ Dåse)

I et relationelt udtryk af formen "$R = \emptyset$" er det klart at typen af $\emptyset$ er skemaet for R, men når vi skriver "$\emptyset = \emptyset$" er det uklart, hvad den fælles type for de to $\emptyset$'er er. Men udtrykket er så alligevel så uinteressant, at vi ikke gider bruge tid på det problem.

Det er nyttigt med andre operationer på relationer, hvor man kan henvise til attributterne. Følgende operation tynder ud i en relation, så kun de tupler, som overholder en angivet betingelse er tilbage:

- **Selektion:** For givet skema S, og hvor B er et udtryk af type $\mathcal{B}$ som kan indeholde attributter i S (med den type de så at sige arver fra S) er $\sigma_B\colon S \to S$ en operation. Værdien af $\sigma_B R$ for en eller anden relation R findes som mængden af de tupler i R, for hvilke B evaluerer til "sand".

Selektion og de tilpassede mængdeoperationer producerer nye relationer med samme skema, som de relationer de blev anvendt på. Der er andre operationer som producerer nye relationer med helt nye skemaer, som er "beregnet" ud fra de indgående relationers skemaer. Her er et eksempel på en sådan:

- **Projektion:** For givet skema S, hvor A er en delmængde at attributterne for S, er $\pi_A\colon S \to S'$ en operation, S' er det skema som fremkommer ved fra S at fjerne komponenter, hvis attribut ikke optræder i A. Værdien af $\pi_A R$ for en eller anden relation R findes ved, for hver tupel i R, at slette de komponenter, hvis attribut ikke er A.

Vi har en operation, der fungerer som en pendant i algebraen til mængdeprodukt:

- **Produkt** For givne skemaer S_1 og S_2 uden sammenfaldende attributter, er $\times\colon S_1, S_2 \to S_1S_2$ en operation, hvor S_1S_2 er det skema som fremkommer ved at tage samtlige attributter fra S_1 og S_2. Værdien af $R_1 \times R_2$ er relationen, som fås ved at danne samtlige mulige tupler $\langle a_1\colon v_1, \ldots, a_{n+k}\colon v_{n+k}\rangle$ hvor $\langle a_1\colon v_1, \ldots, a_n\colon v_n\rangle \in R_1$ og $\langle a_{n+1}\colon v_{n+1}, \ldots, a_{n+k}\colon v_{n+k}\rangle \in R_2$.

NB: I denne formulering er der ikke taget hensyn til det tilfælde, at der optræder attributter med samme navn i de relationer som indgår i et produkt. Relationel algebra indeholder operatorer til at ombenævne attributter, så man kan komme ud over dette problem; der henvises til en databaselærebog. En anden vigtig operation er naturlig join, hvor vi også henviser til databaselærebogen.

Opgave: Skriv nogle relationelle udtryk, som benytter de relationelle operatorer på de "konstante" relationer Kasse, Dåse og Cylinder, og udregn deres værdier.

Opgave: Giv eksempler på sammensatte relationelle udtryk, som indeholder flere relationelle operatorer, og udregn deres værdier.

5 Logik

Tiføjes måske senere.