Database
design

Udarbejdet af Benny Dyhr Thomsen

INDHOLD

Database

Introduktion

Et af de mest forvirrende omstændigheder ved at tilegne sig teori om database design er efter min mening ordene. Niels Bohr sagde engang det eneste vi mangler for at forstå, er ordene. Det er delvist fordi der mangler ord i danske oversættelser og det faktum, at der er så mange ord, der samme betydning. Eksempelvis bruges ordene post, record, række, row, tuble om det samme.

Introduktion af nøglespillerne

- Entiteter
- Attributter
- Relationer

Hvorfor lave forståelsesmodeller (eng. Conceptual Model)

Hvorfor bruger vi forståelsesmodeller til design af databaser? Hvorfor bruge tid på at lave datamodeller når det er tabeller vi behøver? Hvorfor besværet med at forstå foretningsgang, interviewe personer og deltage i møder når det vi behøver er programmer. Du vil lære hvorfor tiden er givet godt ud til modelering. Du vil lære metoder, begreber og vil lære at skabe/læse modeller og ud af dem danne tabeller og nøgledefinitioner fra dem.

Modeller laver vi fordi:

- Det beskriver nøjagtigt de informationer der kræves i forretningen.
- Det tjerner til diskussion.
- Det forebygger misforståelser og fejl.
- Det former dokumentation på ”ideal systemer”.
- Det former en sund basis for fysisk databasedesign.
- Det er en god praksis, specielt når mange deltager.

Forestil dig nogen der ønsker at bygge et hus. I starten eksisterer huset kun i hovedet på de fremtidige ejer som ideer og brudstykker af ønsker. Undertiden ved de fremtidige ejere ikke hvad de ønsker, eller ved om de er gennemførlige. Ønsker kan være helt eller delvist modsigende eller umulige. Det er ikke noget problem i drømmeverdenen, men i den fysiske verden vil alle modsigelser og forhindringer skulle løses før nogen kan bygge huset. Et byggeselskab skal have en solid plan, et sæt dokumenter med tegninger, beskrivelser af materialer, dimensioner på blælker, kloaksystem, el-system m.m. Men

hvordan bliver en ide om et hjem til en plan for et byggeselskab? Der er her arkitekten bliver indvolveret.

Arkitekten er mellemled mellem kunden og byggeselskabet. Arkitekter er trænede i at omsætte ideer til modeller. Arkitekten lytter til kundens beskrivelse af ideer og stiller masser af spørgsmål for at forstå hvad kunden ønsker. Arkitektens har færdigheder i at forstå ideer og putte ideer i et format, som tillader diskussion, analyse, rådgivning, beskrive kritiske detaljer og dokumentere dem, og aftale med kunden hvad byggeselskabet skal bygge.

Målsætning for entitet-relationel-modelering

- Fang *alle* krævede informationer.

  Alle stykker af informationer som kræves for at køre forretningen ordentlig er genkendt.

- Informationer optræder kun *en gang*

  Dette er et vigtigt mål. Så snart et system gemmer information to gange, løber du ind i muligheden for at informationerne ikke er ens begge steder. Hvis du er bruger af et sådant system, hvilken information vil du da stole på?
  Denne målsætning indbefatter et ideelt system der ikke har afledte informationer.

- Modeler *ingen* informationer der er afledt fra andre informationer der allerede er i modellen.

- Informationer er i en forudsigelig, logisk placering.

  Relaterede informationer er holdt sammen.

Normalisering

Normalisering er en metode til at nedbryde data i tabeller der gør dem egnede til lagring. Normalformerne defineret i relationel database teori repræsenterer forskrifter for database design. Forskrifterne strækker sig fra 1. til 5. normalform og leder til en optimal opsplitning af tabeller. Normaliseringsregler er designet til at modvirke opdaterings-anormaliteter og –inkonsistens. Med hensyn til ydelses tab er forskrifterne baseret på formodningen, at alle ikke-nøgle-felter bliver opdateret jævnligt. De tenderer til at forklejne forespørgsler, data kan læses fra måske en enkelt record i en unormaliseret form, men skal nødvendigvis læses fra flere tabeller i normaliseret form. Der er ikke nogen krav til fuldt ud at normalisere tabeller når ydelse er foretrukket.

Første Normalform (1NF)

Første normalform vedrører ”formen” af en record. En relation er i første normalform (1NF) hvis feltværdier er udelelige (eng. atomic) og alle records er unikke. Attributter som deles i underattributter og gentagne grupper er ikke tilladt. Records hvor værdier for alle records er nøjagtig magen til en anden record er ikke tilladt. Under den første normalform skal alle tilfælde af en recordtype indeholde det samme antal felter. Første normalform udelukker variabelt gentagne felter og grupper.

Der er to almindelige typer af ikke-atomic felter. Den første ikke-atomic type af felter er en liste af gentagne grupper. For eksempel, en person kan have flere kompetencer. Hvis disse er listed i det samme felt så er relationen ikke i første normalform.

Den anden type er et struktureret felt f.eks. en adresse. Hvis hele adressen puttes i et felt er relationen ikke i første normalform fordi en adresse består af gadeadressen, postnummer og bynavn. Om et felt opfattes som struktureret eller ikke beror på applikationen. Hvis du aldrig vil have tilgang til data gennem bynavn eller gennem postnummer så kan det godt være passende at have hele adressen i et felt, men hvis du, måske senere, ønsker at sortere på postnummer, for eksempel, så er det ikke passende at putte hele adressen ind i et felt.

Anvendeligheden af første normalform er rimelig klar. Gentagne grupper ødelægger den naturlige rektangulære struktur af en relation. Det er ekstremt vanskeligt at referere til et bestemt element i en repeterende gruppe, fordi en eller anden position inden i attributtens værdi skal specificeres. Ydermere, de forskellige dele af en opdelt attribut, kan opføre sig forskelligt udfra vinklingen af afhængighed. Reglen for første normalform udtrykker fornuftige krav, at hver attribut har hver deres navn. Relationen i Figur 1 er ikke i 1NF for den indeholder lister med gentagne grupper.

Ordrer

Kundenr	**Ordrenr**	**Varer** (lister)
97123456	101	2 hvedemel, 10 æble, 2 smør
96123355	102	10 letmælk, 5 appelsin
75112387	103	1 ost, 2 sukker, 1 smør
97123456	104	1 sukker, 1 stødt kanel, 2 smør

- Figur 1 Unormaliseret relation med lister.

Hvis vi tager relationen i Figur 1 og deler listen op i separate attributter får vi en relation i Figur 2, den relation er ikke i 1NF fordi den indeholder strukturerede felter med gentagne grupper.

Ordrer

Kundenr	**Ordrenr**	**Antal1**	**Vare1**	**Antal2**	**Vare2**	**Antal3**	**Vare3**
97123456	101	2	hvedemel	10	æble	2	smør
96123355	102	10	letmælk	5	appelsin		
75112387	103	1	ost	2	sukker	1	smør
97123456	104	1	sukker	1	stødt kanel	2	smør

- Figur 2 Unormaliseret relation med strukturerede felter.

Hvis vi opdeler lister eller strukturerede felter op i records får vi tabellen i Figur 3.

Ordrer

Kundenr	**Ordrenr**	**Antal**	**Vare**
97123456	101	2	hvedemel
97123456	101	10	æble
97123456	101	2	smør

96123355	102	10	letmælk
96123355	102	5	appelsin
75112387	103	1	ost
75112387	103	2	sukker
75112387	103	1	smør
97123456	104	1	sukker
97123456	104	1	stødt kanel
97123456	104	2	smør

- Figur 3 Relation i 1NF.

Vi får et problem når vi skal vælge hvilke felter, der skal indgå i primær-nøglen, hvis vi skal have en unik reference til records. I princippet kan vi godt finde felter til primærnøglen som kunde bestå af Ordrenr og Vare for at danne en unik nøgle, men nøglen bliver lidt lang, i stedet kan vi indføre en Ordrelinie med et nummer og bruge Ordrenr + Ordrelinie, vi kan så få en god relation som vist i Figur 4.

Ordrer

Ordrenr	**Ordrelinie**	**Kundenr**	**Antal**	**Vare**
101	1	97123456	2	hvedemel
101	2	97123456	10	æble
101	3	97123456	2	smør
102	1	96123355	10	letmælk
102	2	96123355	5	appelsin
103	1	75112387	1	ost
103	2	75112387	2	sukker
103	3	75112387	1	smør
104	1	97123456	1	sukker
104	2	97123456	1	stødt kanel
104	3	97123456	2	smør

- Figur 4 Relation i 1NF med ordrelinie.

Informationer der kun er i 1NF er redundante, det betyder der er overflødige data. For at reducere redundans vil vi konvertere til anden normalform (2NF). At rette relationer der ikke er i første normalform er forholdsvist enkelt. Hver attribut er tildelt sit eget navn og repeterende gruppeelementer bliver til individuelle records.

Normalformer højere en første normalform er motiveret af opdagelsen af anormaliteter, operationer på en relation som resulterer i en inkonsistens eller uønsket tab af data. Fjernelsen af anormaliteter involverer fremskridt gennem flere niveauer af normalformer. Disse fremskridt leder til idealet:

Enhver information associeret mellem dataværdier optræder en og kun en gang og afhænger ikke af tilstedeværelsen af andre associationer.

Anden normalform (2NF)

En tabel er i anden normalform (2NF), hvis den er i 1NF og alle ikke-nøglefelter er afhængig af hele primærnøglen (d.v.s . ingen af felterne er associeret til dele af nøglen). Teknikken for at omstrukturere er også nogenlunde simpel: konstruer en separat relation til at indeholde den delte afhængighed og fjern de afhængige attributter fra den originale relation.

For at gøre et eksempel mere tydelig med overflødige data indføres Ordredato se Figur 5, den er stadig i 1NF idet alle data er atomiske. Det ses at hverken Kundenr eller Ordredato ikke relaterer til hele primærnøglen, de relaterer kun delvist idet de relaterer til Ordrenr.

Ordrer

Ordrenr	Ordrelinie	Ordredato	Kundenr	Antal	Vare
101	1	2003.02.19	97123456	2	hvedemel
101	2	2003.02.19	97123456	10	æble
101	3	2003.02.19	97123456	2	smør
102	1	2003.02.20	96123355	10	letmælk
102	2	2003.02.20	96123355	5	appelsin
103	1	2003.02.21	75112387	1	ost
103	2	2003.02.21	75112387	2	sukker
103	3	2003.02.21	75112387	1	smør
104	1	2003.02.22	97123456	1	sukker
104	2	2003.02.22	97123456	1	stødt kanel
104	3	2003.02.22	97123456	2	smør

- Figur 5 Tabel i 1NF med Ordredato

Vi bringer den til 2NF ved at flytte Ordredato og Kundenr til en separat tabel data se Figur 6 og fjerne redundante data, vi behøver Ordrenr som primærnøgle eller fremmednøgle.

Ordrer

Ordrenr	Ordredato	Kundenr
101	2003.02.19	97123456
102	2003.02.20	96123355
103	2003.02.21	75112387
104	2003.02.22	97123456

- Figur 6 Ordre

Vi får en dannet en ny tabel som vi kalder Ordrelinier.

Ordrerlinier

Ordrenr	Ordrelinie	Antal	Vare
101	1	2	hvedemel

101	2	10	æble
101	3	2	smør
102	1	10	letmælk
102	2	5	appelsin
103	1	1	ost
103	2	2	sukker
103	3	1	smør
104	1	1	sukker
104	2	1	stødt kanel
104	3	2	smør

- Figur 7 Ny tabel Ordrelinier uden Kundenr og Ordredato

Tredje normalform (3NF)

En relation er i Tredje normalform (3NF) hvis den er i 2NF og der ikke er indirekte afhængigheder (dvs. ingen af ikke-nøglefelter er afhængig af et andet ikke-nøglefelt som til gengæld er afhængig af et nøglefelt).

Tredje normalform er ikke overholdt hvis et ikke-nøglefelt indeholder fakta om et andet ikke-nøgle-felt.

Ordrerlinier

Ordrenr	**Ordrelinienr**	**Antal**	**Varenr**	**Vare**	**Pris**
101	1	2	506	hvedemel	15,75
101	2	10	420	æble	2,10
101	3	2	205	smør	7,85
102	1	10	208	letmælk	7,25
102	2	5	424	appelsin	2,35
103	1	1	211	ost	24,55
103	2	2	540	sukker	18,60
103	3	1	205	smør	7,85
104	1	1	540	sukker	18,85
104	2	1	568	stødt kanel	5,70
104	3	2	205	smør	8,15

- Figur 8 2NF tabel med Varenr og Pris

Vi indfører to nye kolonner for at skabe eksemplet til belysning af 3NF se Figur 8. Hvis vi kigger på tabellen kan man sige at **Vare** indeholder fakta om **Varenr** så dette er oplagt at det kræver en opspilting. Vi kan lave er tabel Varer med **Varenr**, **Vare** og **Pris** se Figur 9.

Men hvad med **Pris**? **Pris** er tilsyneladende også fakta om **Varenr**, men nej, den er ikke fakta om et andet felt. Hvorfor ikke er indlysende, for hvis du tænker på formålet med Ordrelinier: en kunde bestiller varer på baggrund af en aftalt pris og der registreres i ordredata. Hvad nu hvis vi foretager en opjustering af priser i tabel Varer, hvad sker der så med ordredata – de er jo købt til en pris og vi kan ikke opjustere priser i ordredata. Derfor når vi sætter tabel Ordrelinier i 3NF skal **Pris** stadig være der men **Vare** forsvinder se Figur 10.

Varer		
Varenr	**Vare**	**Pris**
205	smør	7,85
208	letmælk	7,25
211	ost	24,55
420	æble	2,10
424	appelsin	2,35
506	hvedemel	15,75
540	sukker	18,60
568	stødt kanel	5,70

- Figur 9 Varer, 3NF tabel

Ordrerlinier				
Ordrenr	**Ordrelinienr**	**Antal**	**Varenr**	**Pris**
101	1	2	506	15,75
101	2	10	420	2,10
101	3	2	205	7,85
102	1	10	208	7,25
102	2	5	424	2,35
103	1	1	211	24,55
103	2	2	540	18,60
103	3	1	205	7,85
104	1	1	540	18,85
104	2	1	568	5,70
104	3	2	205	8,15

- Figur 10 Ordrelinier konverteret til 3NF.

Case 1 Cirkus

Unormaliseret

Forestil dig at vi har en database som indeholder information om alle vores dyr i et cirkus. Data for et sådant cirkus i en fuldstændig unormaliseret form ser sådan ud:

Dyr

dyre_nr	dyre_navn	telt_nr	telt_navn	telt_placering	trik_nr1	trik_navn1	trik_lært_ved1	trik_fag_niveau1	trik_nr2	trik_navn2	trik_lært_ved2	trik_fag_niveau2		trik_fag_niveau16
1	Hest	2	Rød	Nord	1	pioette	gulerod	2	2	galop	gulerod	1		
2	Abe	1	Blå	Syd	3	klappe	banan	2						

Hvis vi koncentrerer os om attributterne (kolonnenavne) får vi listen:

Attributter i Dyr tabel
dyre_nr
dyre_navn
telt_nr
telt_navn
telt_placering
trik_nr1
trik_navn1
trik_lært_ved1
trik_fag_niveau1
trik_nr2
trik_navn2
trik_lært_ved2
trik_fag_niveau2
trik_nr16
trik_navn16
trik_lært_ved16
trik_fag_niveau16

Dette er en unormaliseret struktur. Men hvorfor gøre besværet med at normalisere den? Fordi, ellers vil der være at antal logiske uregelmæssige situationer som du skal håndtere. Der vil/bør være logiske fejl hver gang du forsøger at indsætte, slette eller opdatere data i denne tabel. Hver gang du normaliserer til et højere niveau løses disse.

Indsætte Problem: Antag at en chimpanse er købt og skal tilføjes til tabellen. Det vil være umuligt at tilføje en ny chimpanse uden også at tilføje triks omend afsatte felter til trik er tomme. Omvendt vil det også være umuligt at lægge et nyt trik ind uden også at tildele et cirkusdyr der kan udføre det. Der kan ikke læres flere trik end 16 uden at udvide

triks for alle dyrene. Du kan heller være sikker på at samme trik er stavet ens, hvilket vil gøre det mere besværligt at søge triks op end det er i forvejen.

Slette Problem: Hvad sker der hvis vi vil slette et trik, desværre, vigtige data på et cirkusdyr vil blive slettet også. Omvendt vil triks også forsvinde når et dyr slettes, et trik som måske stadig vil anvendes af cirkuset. Det kan være du sidder og tænker "man kan da bare lave blanke felter", meget vel men det er besværligt at finde de triks der skal blankes når et trik ikke skal være der mere.

Opdaterings Problem: Opdatering på en unormaliseret tabel vil kunne skabe problemer på flere poster. Stor omhu må tillægges for at finde de rigtige poster når ændringer skal udføres. Hvis der skal opgraderes på trik_fag_niveau skal alle dyr med det pågældende trik opdateres.

Første Normal Form (1NF)

For at sætte tabellen i 1. Normal Form (1NF) må vi eliminere gentagne grupper. Det betyder at alle undergrupper af data som optræder inde i posten skal flyttes til en separat tabel. Derved bliver vores unormaliserede tabel til 2 tabeller.

Første Normal Form (1NF)

Cirkus_dyr tabel
dyre_nr
dyre_navn
telt_nr
telt_navn
telt_placering

Triks tabel
dyre_nr
trik_nr
trik_navn
trik_lært_ved
trik_fag_niveau

Nu da vores tabeller er i 1NF, har vi en klar fordel i pladsbesparelse. Nu, vil et dyr der kun kender få triks ikke optage unødvendig plads tildelt til optil 16 triks, som var tilfældet i den unormaliserede form. Og som sidegevinst kan et dyr udføre flere en 16 triks eller måske slet ingen hvis den ikke har lært nogen endnu.

Anden Normal Form (2NF)

Lad os tage tabeller til 2NF ved at eliminere redundante (overflødige) data. Et ikke-nøgle felt må ikke have en værdi der kan udledes af en del af nøglen. For eksempel, et triknavn optræder i hver eneste post for et dyr, men trik_nr er nok da vi allerede har trik_navn i TRIKS tabellen. Så denne 1NF tabel:

Triks tabel
dyre_nr
trik_nr
trik_navn
trik_lært_ved
trik_fag_niveau

Bliver til disse 2NF tabeller:

Triks tabel
trik_nr
trik_navn

Dyre_Triks tabel
dyre_nr
trik_nr
trik_lært_ved
trik_fag_niveau

Cirkus_dyr tabel (Uændret)
dyre_nr
dyre_navn
telt_nr
telt_navn
telt_placering

2NF, såvel som 1NF, har sine egne logiske fejl når data skal manipuleres.

Indsætte Problem: Antag at vi ønsker at kreere en ny post som repræsenterer en tiger tildelt til "De store kattes" telt. Det er umuligt at oprette en ekstra tigers post for Cirkus_dyr tabellen uden først at kreere en ny afdeling. Dette skyldes at telt_nr attributten er bundet til dyre_nr attributten

Slette Problem: Sletning af et bestemt dyr kan resultere i tabet af et helt telt, selvom om du ønsker at beholde informationer om teltet.

Opdatering Problem: Telt informationer er redundante i Cirkus_dyr tabellen, enhver ændring i telt information kræver en søgning på hele tabellen for at lokalisere alle poster som behøver opdateringer.

Endnu et trin i normalisering er nødvendig for at eliminere disse logiske fejl.

Tredje Normal Form (3NF)

For at tage denne samling af tabeller til 3NF, behøver vi nu eliminere attributter der ikke er afhængig af primærnøglen. Husk at ordet 'eliminere' ikke betyder at slette. I stedet betyder det at opsplitte i separerede tabeller.

Tabeller i 3NF:

Triks tabel (Uændret allerede 3NF)
trik_nr
trik_navn

Dyre_Triks tabel (Uændret allerede 3NF)
dyre_nr
trik_nr
trik_lært_ved
trik_fag_niveau

Cirkus_dyr tabel
dyre_nr
dyre_navn
telt_nr

Telte tabel
telt_nr
telt_navn
telt_placering

Ideen med 3NF er, at spørge 'attributterne i denne tabel relaterer de til hinanden?'. Når du når til 2NF, vil attributterne formodentlig alle afhænge af primærnøglen for deres identifikation. Dog vil attributterne have en mangel på tilhørsforhold til hinanden således at de bør udskilles til en separat tabel.

Opsummering

For at opsummere: Normalisering af data betyder eliminering af redundante information fra en tabel og genkende data så fremtidige ændringer i tabellen er lettere. En tabel er i sin første normal form når tabellen indeholder den mindste meningsfulde datamængde og tabellen indeholder ingen gentagne data. Anden normal form refererer kun til tabeller med flere primære nøgler. Tredje normal form refererer kun til tabeller med en enkelt nøgle og kræver at hvert ikke nøgle at være en direkte beskrivelse at primærnøglen.

Øvelse

Følgende ses data for en filmforening, hvert medlem kan reservere optil 10 billetter til forestillinger. Normaliser gennem 1., 2. og 3. normal form.

Unormaliseret
medlems_nr
medlems_navn
medlems_adresse
medlems_postby
forestilling_nr1
forestilling_navn1
forestilling_ugenr1
billet_plads1
billet_pris1
reservation_antal1
forestilling_nr10
forestilling_navn10
forestilling_ugenr10
billet_plads10
billet_pris10
reservation_antal10

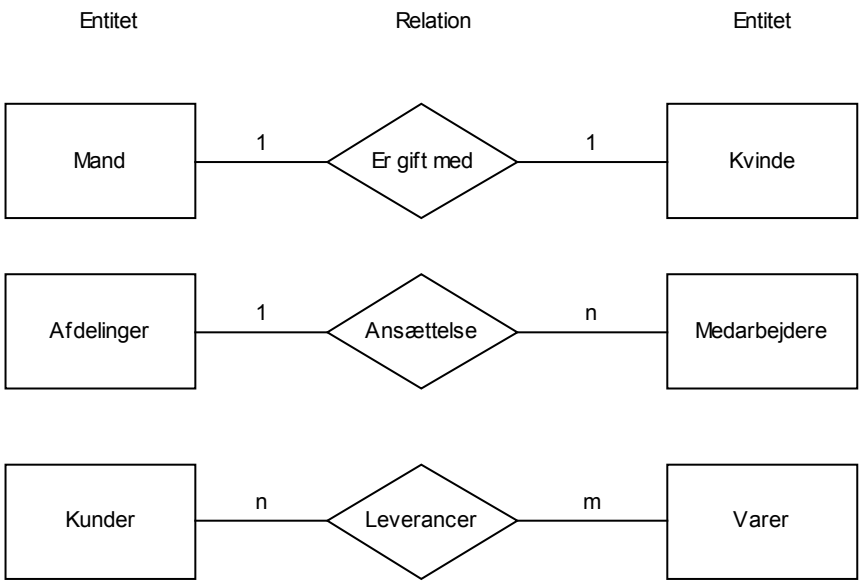
Normalisering udover 3NF.

Den gode nyhed er her at ingen bruger disse normal former, i hvert fald i en normal database applikation. Fjerde normal form (4NF) kræver at en tabel ikke indeholder uafhængige en til mange relationer. Uafhængige en til mange relationer, behøver at opsplittes til separate tabeller. Femte normal form (5NF) refererer til en tilstand af hele databasen, kræver at hver tabel er nedbrudt til mindst mulige for at eliminere redundante ikke-nøgler. Forbindelses tabeller giver dig en link til informationer mellem tabellerne i den femte normal form.

Constraints

Følgende viser et E-R diagram (Entity-Relation), det fremstiller forskellige relationer. Med *Mapping Cardinality* udtrykkes hvormed det antal entiteter som en anden entitet kan associeres. Ordet Constraint som kan betyde begrænsning, tvang og indskrænkning passer

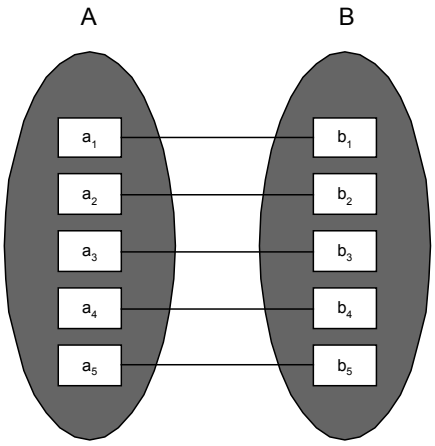
ikke helt i oversættelsen fra ordbøger, men jeg mener *præcisering* er bedre. Cardinality betyder et antal elementer i et matematisk talmængde.



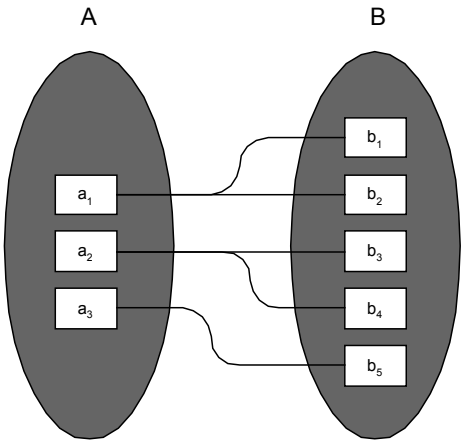
- Figur 11 E-R diagram viser Mapping Cardinality

På dansk kaldes disse:

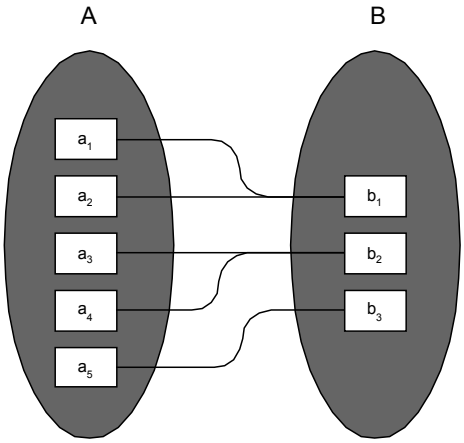
- 1:1 En til en relation.
- 1:n En til mange relation.
- n:1 Mange til en relation.
- n:m Mange til mange relation.



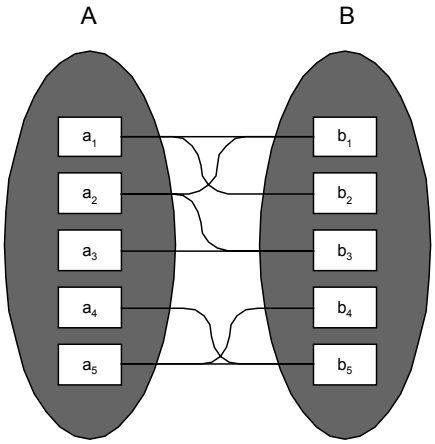
- Figur 12 En til en relation



- Figur 13 En til mange relation



- Figur 14 Mange til en relation



- Figur 15 Mange til mange relation

Relationel algebra

Relationel algebra er et *produral query language*. Det indeholder et udvalg af operationer som tager en eller to relationer som input og giver en ny relation som resultat.

Fundamentale operationer

De fundamentale operationer er *select*, *project*, *union*, *set difference*, *Cartesian product* og *rename*. Select, project og rename er kaldet *unary* operationer, fordi de kun bruges på en relation. De andre tre union, set difference og Cartesian product er *binary* operationer fordi de bruges på relationer parvis.

Select operation

Select operationen udvælger rækker som opfylder et givent kriterium. Vi bruger det lille græske bogstav sigma (σ) til at notere en select. Selvom det hedder en select-operation

svarer det til WHERE i en SQL-kommando. Tabellen eller tabeller sættes ind i parantesen og svarer til FROM i en SQL-kommando.

$\sigma_{Ugenr=52}(\text{Forestilling})$

Forestilling

FNr	Titel	Ugenr
52-1	Nøddebro præstegård	52
52-2	Julebal i nisseland	52

$\sigma_{Pris>25}(\text{Biletter})$

Billetter

FNr	Plads	Pris
50-1	Balkon	40,00
51-1	Gulv	30,00
51-1	Balkon	45,00
52-1	Gulv	30,00
52-1	Balkon	45,00
52-2	Balkon	35,00

Vi kan bruge sammenlignings operatorer $=,\neq,<,\leq,>,\geq$ i select. Udover kan vi kombinere kriterier til større kriterier med logiske operatorer *and* ($\wedge$), *or* ($\vee$) og *not* ($\neg$). For at finde billetter på balkon og som koster mere end 12 kr. skriver vi:

$\sigma_{Plads="Balkon"\wedge Pris>30}(\text{Biletter})$

Billetter

FNr	Plads	Pris
50-1	Balkon	40,00
51-1	Balkon	45,00
52-1	Balkon	45,00
52-2	Balkon	35,00

Project operation

Hvis vi kun ønsker en del af kolonnerne vist bruger vi project med det store græske bogstav pi ($\prod$). Vi beskriver de attributter vi ønsker til en selection. Derved svarer Project-operationen til SELECT i en SQL-kommando.

$\Pi_{navn,adresse}(\text{Medlemmer})$

Medlemmer	
Navn	**Adresse**
Anders Nielsen	Kirkegade 14
Bodil Pedersen	Søndergade 19
Anne Overgård	Vestergade 45

Sammensæt relationelle operationer

$$\Pi_{\text{Titel}}(\sigma_{\text{Ugenr=52}}(\text{Forestilling}))$$

Forestilling
Titel
Nøddebro præstegård
Julebal i nisseland

svarer til SQL-kommandoen

SELECT Titel FROM Forestilling WHERE Ugenr=52

Union operation

Forestil dig at du skal finde navne på personer der er debitorer der eller kreditorer eller begge dele. Vi ved hvordan vi finder debitorer:

$$\Pi_{\text{Navn}}(\text{debitorer})$$

debitorer	
Navn	**Adresse**
Anders Nielsen	Kirkegade 14
Bodil Pedersen	Søndergade 19
Anne Overgård	Vestergade 45

og vi ved hvordan vi skal finde kreditorer

$$\Pi_{\text{Navn}}(\text{kreditorer})$$

kreditorer	
Navn	**Adresse**
Peter Hansen	Alpegade 33
Bodil Pedersen	Søndergade 19

Anne Overgård	Vestergade 45

Vi bruger ∪ som kendes fra alm. mængdelære som *foreningsmængden*.

$$\Pi_{\text{Navn}}(\text{debitorer}) \cup \Pi_{\text{Navn}}(\text{kreditorer})$$

Navn
Anders Nielsen
Peter Hansen
Bodil Pedersen
Anne Overgård

To krav skal opfyldes for at union giver mening, for en union r ∪ s skal være valid kræves:

1. At relationen r og s skal have det samme antal attributter.
2. At attributter er defineret i samme orden.

Set difference operation

Set-difference operationer, noteret med -, giver os mulighed for at finde rækker der findes i en relation med ikke i en anden. Udtrykket r – s finder rækker som findes i r men ikke i s. Vi kan finde debitorer som ikke også er kreditorer:

$$\Pi_{\text{Navn}}(\text{debitorer}) - \Pi_{\text{Navn}}(\text{kreditorer})$$

Navn
Anders Nielsen

Som ved union må vi stille samme krav om kompabilitet.

Cartesian product

Cartesian-product, noteret med (X), tillader os at kombinere informationer fra to relationer. For to relationer r1 og r2 med n1 og n2 rækker gælder at vi får en relation med n1*n2 rækker.

Rename operation

Modsat relationer i database, resultatet af relationel-algebra udtryk har ikke nogen navne som vi kan bruge til at referere til dem. Det er belejligt at kunne give dem navne. **Rename** operatoren, noteret med det lille græske bogstav rho (ρ) lader os gøre dette.

Givet et relationel-algebra udtryk *E*, vil udtrykket

$\rho_x(E)$

returnere resultatet af udtrykket E under navnet x. Hvis vi vil finde den debitor der skylder flest penge, kunne vores strategi være, (1) først danne en temporær relation for alle balancer der ikke er den største og (2) med set-difference på debitorer og den temporære relation finde den række der ikke findes i den temporære.

Find alle balancer der ikke er den største:

$$\Pi_{\text{debitorer.balance}}(\sigma_{\text{debitorer.balance}<\text{d.balance}}(\text{debitorer} \times \rho_{\text{d}}(\text{debitorer})))$$

Find den største balance:

$$\Pi_{\text{balance}}(\text{debitorer}) - \Pi_{\text{debitorer.balance}}(\sigma_{\text{debitorer.balance}<\text{d.balance}}(\text{debitorer} \times \rho_{\text{d}}(\text{debitorer})))$$

Aggregate functions

Tager et udvalg af værdier og returnerer en enkelt værdi som resultat. F.eks. aggragat funktionen **sum** tager et udvalg af værdier og returnerer summen af værdierne. Således at funktionen sum brugt på følgende værdier {1,2,3} returnerer værdien 6.

Symbolet ς som ligner et G, jeg kan ikke finde den i det græske alfabet, anvendes til aggragat notationen.

Forudsæt at vi ønsker at finde hvor meget debitorer skylder i alt. Den relationelle algebraiske udtryk for dette er:

debitorer

Navn	Saldo
Anders Nielsen	2600
Bodil Pedersen	3500
Anne Overgård	5300

$$\varsigma_{\text{sum(Saldo)}}(\text{debitorer})$$

Hvis vi vil bruge en group som vi kender fra SQL kommandoer på nedenstående tabel kunne vi finde hvor summen af lån optaget i forskellige afdelinger af en bank.

sager

Navn	Afd	lånebeløb
Hansen	Herning	40000
Nielsen	Herning	30000
Pedersen	Herning	25000
Andersen	Ikast	15000
Mikkelsen	Ikast	45000

Det skrives som:

$$_{\text{Afd}}\varsigma_{\text{sum(lånebeløb)}}(\text{sager})$$

Der er også tilladt at skrive:

$$_{\text{Afd}}\varsigma_{\text{sum(lånebeløb) as sum-beløb, max(lånebeløb) as max-beløb}}(\text{sager})$$

med den formulering skulle vi få summen og max beløbet ud per. afdeling, vi får endda mulighed for at vise at vi ønsker at døbe navnet for resultaterne sum-beløb og max-beløb.

Diverse

funktionel afhængighed

En **funktionel afhængighed** er en type af constraint som er en generalisering på fastlæggelsen af en nøgle.

Ordliste

atom	Mindste enhed der ikke kan deles.
atomistisk, atomisk	Udelelig.
attribut	Attribut, kolonne.
cardinality	Kardinalitet. Et matematisk begreb for tal sæt i f.eks. talfølger, primtal, kubiktal, naturlige tal m.m.
composite	Sammensat. om attributter der er sammensat af flere værdier. En sammensat attribut kan være fornavn+mellemnavn+efternavn i en attribut.
constraint	Begrænsning, indskrænkning, tvinge. Præcisering. Det at lægge rammer eller bånd på noget.
DBA	Data Base Administrator.
DDL	Data Defination Language
derived depencies	afledte afhængigheder. Beløb er en sådan hvis den er afledt af Pris*Antal
DML	Data-Manipulation Language
domain	Værdisæt. For hver attribut er der et tilladt sæt af værdier kaldet domain eller værdisæt.
entity	entitet, individ. Beskriver et objekt i den virkelige verden der adskiller sig fra andre. Svarer til en post eller tuble.
entity set	tabel. Beskriver en samling af entiteter der er af samme type.
felt	En post består af felter. Bruges om et felt i en post. Derved adskiller den sig fra attribut og kolonne. Hvis alle felter i en kolonne har forskellig værdi kan man sige at det er et nøglefelt fordi den entydigt kan identificere posten.
functional dependency	Funktionel afhængighed. En **funktionel afhængighed** er en type af constraint som er en generalisering på fastlæggelsen af en nøgle.
kolonne	Attribut.
primærnøgle	Et eller flere felter der tilsammen danner unik reference til en række.
relation	Tabeller der sammenknyttes ved hjælp af *relationer* dannes ved primærnøgle og fremmednøgle. Bruges også om kolonner der danner en relation til hinanden.
relation schema	
relationship	Forhold. Et stadie af relation der binder deltagere i et sammenhold (relationship).
relationship set	
row	Række, record, post, tubel.
række	Row, Record, post, tubel
selection	
superkey	
tabel	Består af kolonner og rækker. Entity set.
tubel	Record, post, row, række.

Skabelon

•

Parameter	Beskrivelse

program

1.	
2.	

Personer

Navn	Adresse	Postby
Anders Nielsen	Kirkegade 14	7430 Ikast
Bodil Pedersen	Søndergade 19	7430 Ikast

Noter

3NF formulering

En anden fremstilling af 3NF er at det resultat i relationer som repræsenterer entiteter og relationer som repræsenterer forhold mellem entiteter. En acceptabel måde at konvertere en tabel til 3NF er at placere de felter, som ikke er direkte afhængige af en nøgle, i en separat tabel. En opsplitning vist indtil videre har to vigtige egenskaber:

1. Hver afhængighed er omsluttet i nogle relation (opsplitningen bevarer afhængigheder)

2. Hvis en udvidelse af en original relation er opsplittet, kan den rekonstrueres via en JOIN med dens komponent (opsplitningen har tabsfri join).