MODELLER OCH SPRÅK FÖR RELATIONSDATABASER:

Relationsalgebra, Relationskalkyl (Tuple calculus) & SQL

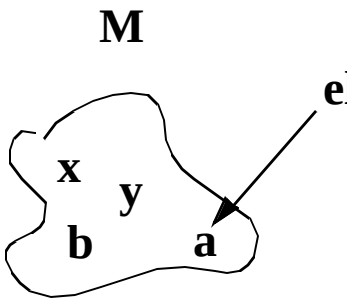
Ted Codd 1970 - klassisk artikel: "The relational model of data"

- DATASTRUKTUR
- OPERATIONER
- "INTEGRITY CONSTRAINTS" (ENTITY&REFERENTIAL)

DATASTRUKTUR

- Relationer (mängder)

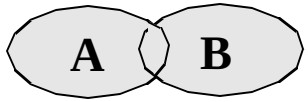
Baserat på mängdteori och första ordningens predikatlogik



M = {a, b, x, y }

A = {a, b, c } **B =** {a, d } **C =** {a, b }

Union A U B = {a, b, c, d}



Snitt A ∩ B = {a}

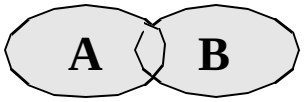
Delmängd C ⊆ A

$

DATASTRUKTUR forts.

Mängdlära: **A =** {a, b, c } **B =** {a, d }

Union A U B = {a, b, c, d}



"Det som finns i A plus det som finns i B"

Snitt A ∩ B = {a}

"Bara det som finns i *både* A och B"

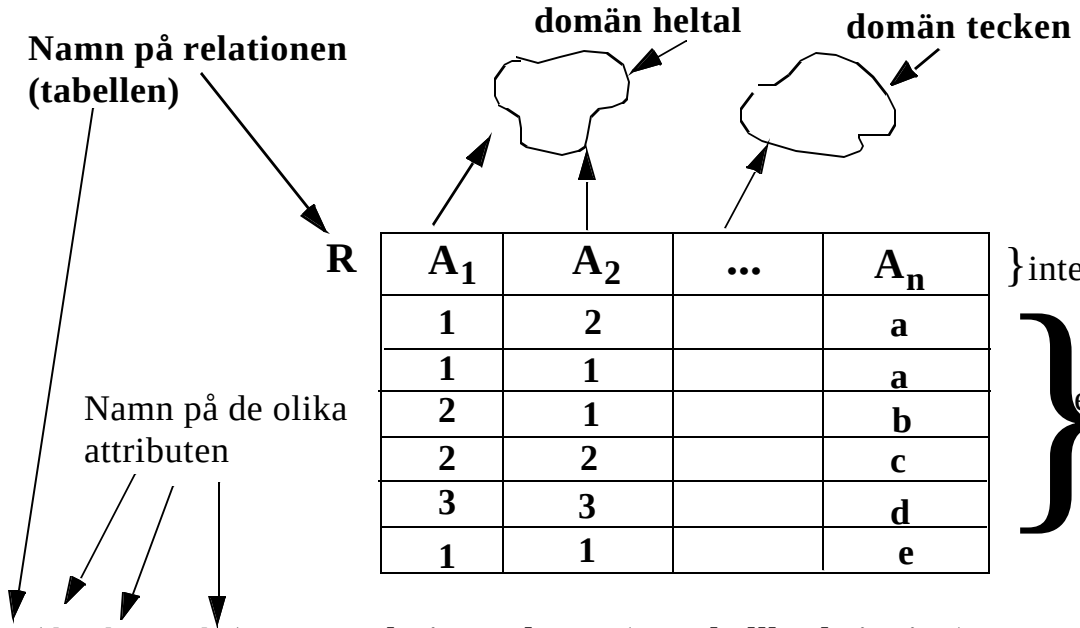
Differens A - B = {c} "Det som finns i A men inte i B"

Mängder är oordnade: {a, b} = {b, a}

Mängder saknar dubletter: ~~{a, b, a}~~ dvs, *alla* element i en mängd är olika!

Övning: Är A U B = B U A ?, Är A ∩ B = B ∩ A ?, Är A - B = B - A ?

Datastruktur



Relation = tabell
Relationsschema = tabellbeskrivning
Rad = tuple = tuppel = post
Attribut = kolumn

Vad är då en relation? Jo, en relation, vi kan kalla den R, är en MÄNGD av rader (tupler, poster) som instansierar relationsschemat. T ex är raden (1, 2, ..., a) en delmängd i R. Man säger att relationen R har *graden* n, är n-ställig, har n st attribut, kolumner. Vi kommer att referera till detta som t_s(R) (av *Tuppel-Storlek*). Antalet rader (tupler) kallas R:s *kardinalitet* (brukar anges som absolutbeloppet av R, |R|).

Operationer

- Relationsalgebra (procedurell, “hur”, vilken operationsföljd)
- Relationskalkyl (deklarativ, “vad”)
  - tuppelkalkyl
  - domänkalkyl

Frågespråk, t ex SQL baserar sig på algebra eller kalkyl, basformalismer

Relationsalgebra ≡ Tuppelkalkyl

↑

Samma uttryckskraft

Algebrauttryck ↔ Kalkyluttryck

↑

Transformationsregler

Relationsalgebra

Primtiva operatorer

- projektion π
- selektion σ
- union $\cup$
- differens -
- kryssprodukt x

Med hjälp av dessa operatorer kan andra (icke-primitiva) operatorer definieras:

Icke-primitiva operatorer

- theta-join θ
- ekvi-join
- naturlig join $|x|$
- snitt $\cap$
- division $\div$

Tilldelning :=

Relationsalgebran är *sluten*:

R_1 op $R_2 \Rightarrow R_3$

relation relation relation

$((R_1 \text{ op } R_{2)} \text{ op } R_3) \text{ op } R_4$

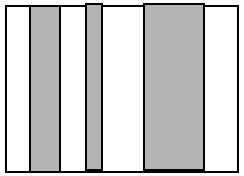
Tar en eller två tabeller som argument (indata).

Producerar en ny tabell som resultat!

Projektion π

- Unär operator $\pi_{A1, A2,...An}$(Relationsnamn) "Vertikal delmängd av attribut"

attribut



ANSTÄLLD

Namn	Lön	Chef	Avd
Per Kvist	15000	Eva Berg	Parfym
Bo Gren	20000	Eva Berg	Parfym
Sten Rot	22000	Nils Hed	Skor
Nils Hed	30000	Eva Berg	Skor
Eva Berg	35000	Eva Berg	Parfym

$t_s(\pi_{A1,A2,...An}(R))$ = antalet attribut, dvs just här = n,

$|\pi_{A1,A2,...An}(R)| \leq |R|$, vanligen är $|\pi_{A1,A2,...An}(R)| = |R|$

Projektion innebär att välja ut ett antal attribut ur en relation.

$p_{Namn,Lön}$**ANSTÄLLD**

Namn	Lön
Per Kvist	15000
Bo Gren	20000
Sten Rot	22000
Nils Hed	30000
Eva Berg	35000

Omordning av attribut möjligt via π

$\pi_{Lön, Namn}$ **(ANSTÄLLD)**

Lön	Namn
15000	Per Kvist
20000	Bo Gren
22000	Sten Rot
30000	Nils Hed
35000	Eva Berg

Selektion σ

- Unär operator

"Horisontelll delmängd av rader"

$\sigma_{villkor}(R)$

Enkelt villkor: attribut θ attribut
attribut θ värde, där
θ ∈ **{=,** ≠, **<, >,** ≤, ≥**}**

Sammansatt villkor: villkor operator villkor
där operator ∈ **{NOT, AND, OR}**

ANSTÄLLD

Namn	Lön	Chef	Avd
Per Kvist	15000	Eva Berg	Parfym
Bo Gren	20000	Eva Berg	Parfym
Sten Rot	22000	Nils Hed	Skor
Nils Hed	30000	Eva Berg	Skor
Eva Berg	35000	Eva Berg	Parfym

Selektion innebär att man väljer ut ett antal tupler ur en relation baserat på något villkor.

$S_{Avd=Parfym}$**ANSTÄLLD**

Namn	Lön	Chef	Avd
Per Kvist	15000	Eva Berg	Parfym
Bo Gren	20000	Eva Berg	Parfym
Eva Berg	35000	Eva Berg	Parfym

$t_s(\sigma_{villkor}(R)) = t_s(R)$

$|\sigma_{villkor}(R)| \leq |R|$

Kartesisk product

A × B (eller ibland A * B) ska tolkas som "alla rader i A kombinerade med alla rader i B"

A
a
b
c

B
x
y

A × B
a x
a y
b x
b y
c x
c y

$t_s(A \times B) = t_s(A) + t_s(B)$

$|A \times B| = |A| \cdot |B|$

Kartesisk produkt, ett exempel till

ANSTÄLLD

Namn	Lön	Chef	Avd
Per Kvist	15000	Eva Berg	Parfym
Bo Gren	20000	Eva Berg	Parfym
Sten Rot	22000	Nils Hed	Skor
Nils Hed	30000	Eva Berg	Skor
Eva Berg	35000	Eva Berg	Parfym

AVDELNING

Anamn	Våning
Leksaker	2
Livsmedel	3
Parfym	3
Skor	2
Trädgård	1

ANSTÄLLD × AVDELNING

Namn	Lön	Chef	Avd	Anamn	Våning
Per Kvist	15000	Eva Berg	Parfym	Leksaker	2
Per Kvist	15000	Eva Berg	Parfym	Livsmedel	3
Per Kvist	15000	Eva Berg	Parfym	Parfym	3
Per Kvist	15000	Eva Berg	Parfym	Skor	2
Per Kvist	15000	Eva Berg	Parfym	Trädgård	1
Och så vidare på samma sätt...					
Eva Berg	20000	Eva Berg	Parfym	Leksaker	2
Eva Berg	20000	Eva Berg	Parfym	Livsmedel	3
Eva Berg	20000	Eva Berg	Parfym	Parfym	3
Eva Berg	20000	Eva Berg	Parfym	Skor	2
Eva Berg	20000	Eva Berg	Parfym	Trädgård	1

Totalt har ANSTÄLLD × AVDELNING 25 st rader! (ANSTÄLLD har 5 rader och AVDELNING har 5 rader = totalt 5*5 st rader).

θ–JOIN (“theta-join”)

där $\theta \in$ **{=,** $\neq$, **<, >,** $\leq$, $\geq$**}**

ANSTÄLLD

Namn	Avd
Pia	5
Mia	3
Ken	3

AVDELNING

Avd	ANamn
5	Bröd
1	Ost
3	Vin

ANSTÄLLD θ **AVDELNING**
Anställd.Avd = Avdelning.Avd

Namn	Anst. Avd	Avd. Avd	Anamn
Pia	5	5	Bröd
Mia	3	3	Vin
Ken	3	3	Vin

Här bildas den nya tabellen genom att matcha kolumnvärden (här kolumen Avd) från två tabeller.
Om jämförelsevillkoret är “=” talar man om en “equi-JOIN”.

Observation: $\sigma_{\theta\text{-villkor}}(A \times B) \Leftrightarrow A\ \theta_{\theta\text{-villkor}}\ B$

$t_s(A \theta B) = t_s(A) + t_s(B)$

$|A \theta B| \leq |A| \cdot |B|$

NATURAL JOIN

ANSTÄLLD

Namn	Avd
Pia	5
Mia	3
Ken	3

AVDELNING

Avd	ANamn
5	Bröd
1	Ost
3	Vin

ANSTÄLLI |X| AVDELNING

Namn	Avd	Anamn
Pia	5	Bröd
Mia	3	Vin
Ken	3	Vin

En NATURAL JOIN är en EQUI-JOIN där man projicerat bort ett av de ingående JOIN-attributen (här tar vi bort dupliceringen av JOIN-attributet "Avd"). En NATURAL JOIN förutsätter att JOIN-attributet (attributen) heter likadant i det två tabeller som ska joinas (i annat fall blir NATURAL JOIN samma sak som Cartesisk produkt).

$t_s(A |X| B) \leq t_s(A) + t_s(B)$

$|A |X| B| \leq |A| \cdot |B|$

Observation: $\pi(\sigma_{=\text{-villkor}}(A \times B)) \Leftrightarrow A |X| B$

Övning i relationsalgebra

ANSTÄLLD(Namn, Lön, Chef, Avd)

Vad innebär följande uttryck i naturligt språk?

$\pi_{\text{Namn}}(\sigma_{\text{Lön} > 20000}(\text{ANSTÄLLD}))$

Skriv ett relationsalgebraiskt uttryck som ger namnen på cheferna för de anställda som tjänar mer än 25000 på skoavdelningen.

ANSTÄLLD

Namn	Lön	Chef	Avd
Per Kvist	15000	Eva Berg	Parfym
Bo Gren	20000	Eva Berg	Parfym
Sten Rot	22000	Nils Hed	Skor
Nils Hed	30000	Eva Berg	Skor
Eva Berg	35000	Eva Berg	Parfym

Relationsalgebra övning

: Ta fram namnen på alla chefer över anställda på skoavdelningen som tjänar mer än 20000:

$$\pi_{\text{Chef}}(\sigma_{\text{Lön>20000 AND Avd=”Sko”}}(\text{ANSTÄLLD}))$$

alternativt:

$$\pi_{\text{Namn}}(\text{ANSTÄLLD}) \mid \text{X} \mid \pi_{\text{Chef}}(\sigma_{\text{Lön>20000 AND Avd=”Sko”}}(\text{ANSTÄLLD}))$$

Övning i relationsalgebra

ANSTÄLLD(Namn, Lön, Chef, Avd)
AVDELNING(Anamn, Våning)

Skriv ett relationsalgebraiskt uttryck som ger namn och lön för de anställda som arbetar på andra våningen.

$$\pi_{\text{Namn, Lön}}(\text{ANSTÄLLD} \;\theta_{\text{ANSTÄLLD.Avd=AVDELNING.Anamn}}\; \sigma_{\text{Våning=2}}(\text{AVDELNING}))$$

Union compatibility

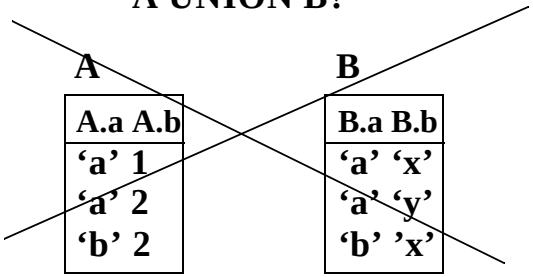
Somliga av de relationsalgebraiska operatorerna kräver att de tabeller som de opererar på ska vara "unionskompatibla". Detta betyder att de två tabellerna måste ha:

- samma grad (lika många attribut/kolumner)
- attributen måste heta likadant och komma i samma ordning
- attribut som motsvarar varandra måste ha samma domän

UNION, SNITT och DIFFERENS är unionskompatibla operatorer.

Ska man ta unionen av två tabeller måste tabellerna ha lika många kolumner och de par av kolumner som svarar mot varandra måste vara av samma typ (ha samma domän).

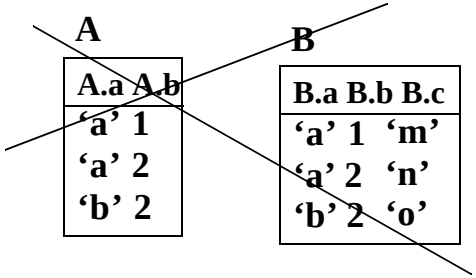
A UNION B?



A	
A.a	A.b
'a'	1
'a'	2
'b'	2

B	
B.a	B.b
'a'	'x'
'a'	'y'
'b'	'x'

A UNION B?



A	
A.a	A.b
'a'	1
'a'	2
'b'	2

B		
B.a	B.b	B.c
'a'	1	'm'
'a'	2	'n'
'b'	2	'o'

A UNION B!

A	
A.a	A.b
'a'	'x'
'a'	'y'
'b'	'x'

B	
B.a	B.b
'b'	'x'
'a'	'm'
'b'	'x'

UNION $\cup$

A

Namn	Ras
Fido	Tax
Karo	Tax
Mindy	Pudel
Ossi	Dvärgtax

B

Namn	Ras
Fido	Tax
Emir	Tax
Lady	Schäfer
Morris	Welsh Corgie

A $\cup$ B

Namn	Ras
Fido	Tax
Karo	Tax
Mindy	Pudel
Ossi	Dvärgtax
Emir	Tax
Lady	Schäfer
Morris	Welsh Corgie

"Alla rader som finns i A plus alla rader som finns i B"

$t_s(A \cup B) = t_s(A) = t_s(B)$

$|A \cup B| \leq |A| + |B| \geq \max(|A|, |B|)$

Observation: $A \cup B = B \cup A$

SNITT (INTERSECT) ∩

A

Namn	Ras
Fido	Tax
Karo	Tax
Mindy	Pudel
Ossi	Dvärgtax

B

Namn	Ras
Fido	Tax
Emir	Tax
Lady	Schäfer
Morris	Welsh Corgie

A ∩ B

Namn	Ras
Fido	Tax

"Alla rader som finns i både A och B"

Observation I : $A \cap B = B \cap A$

$t_s(A \cap B) = t_s(A) = t_s(B)$

$|A \cap B| \leq \min(|A|, |B|)$

DIFFERENCE (MINUS)

A

Namn	Ras
Fido	Tax
Karo	Tax
Mindy	Pudel
Ossi	Dvärgtax

B

Namn	Ras
Fido	Tax
Emir	Tax
Lady	Schäfer
Morris	Welsh Corgie

A MINUS B

Namn	Ras
Karo	Tax
Mindy	Pudel
Ossi	Dvärgtax

"Alla tupler som finns i A men inte i B"

$t_s(A - B) = t_s(A) = t_s(B)$

$|A - B| \leq |A|$

Observation: A - B ≠ B - A

Differens fortsättning...

Differens är användbart för frågor av typ: Ta fram alla som *inte* gjort ngt/*helt saknar* en viss egenskap etc. T ex "Ta fram alla personer som aldrig ätit glass"

PERSON

Namn
Maria
Stina
Pelle

GLASS

Glass
Päronsplit
Storstrut
Vaniljpuck

ÄTANDE

Namn	Glass
Maria	Päronsplitt
Maria	Vaniljpuck
Maria	Storstrut
Pelle	Vaniljpuck
Pelle	Storstrut
Stina	Vaniljpuck
Stina	Storstrut
Stina	Päronsplit

1. Skapa en relation som som innehåller alla personer som har ätit glass:
GLASSÄTARE := π_{Namn}(ÄTANDE)

2. **Dra sen bort denna mängd tupler från relationen PERSON (dvs mängden av alla personer):**

ALDRIG_ÄTIT_GLASS := PERSON - GLASSÄTARE

Observera att man måste projicera ut "Namn" från ÄTANDE eftersom differens kräver att de ingående relationerna ska vara unionskompatibla!

Kvot ÷

R

A	B
x	m
x	n
x	o
y	m
y	n
z	m
z	n
z	o

S

B
m
n
o

R ÷ S

A
x
z

Vi har två relationer R och S. För att förstå vad R KVOT S är kan man tänka på ungefär följande sätt: Vilka kolumner kommer resultatet att innehålla: Jo R - S dvs de kolumner som förekommer i R men inte i S. I vårt fall blir det kolumnen A i R. Vilka kolumn-*värden* kommer med i kolumnen A? Jo alla de A-värden i R som är relaterade till ALLA B-värden i S. I vårt fall är detta sant för A-värden x och z. Dvs x har ett B-värde för både m,n och o. Det samma gäller z. A-värdet y kommer inte med eftersom det saknar ett B-värde.

Kvot, forts.

R

A_1	A_2	A_3
1	a	1
1	a	2
1	a	3
2	a	1
2	a	2

S

A_2	A_3
a	1
a	2
a	3

$R(A_1,..., A_n, A_{n+1}, ..., A_{n+m})$
$S(A_{n+1}, ..., A_{n+m})$
$R \div S = Resultat(A_1, ..., A_n)$

Varje A_1, ...A_n - del av en R-tuppel ("huvudet") har A_{n+1},...A_{n+m} - del ("svansen") som är lika med *var och en* av S-tupplerna, till vänster är n=1 och m=2.

Resultat

A_1
1

$t_s(A \div B) = t_s(A) - t_s(B)$

$|A \div B| \leq |A|$

För att kunna ta kvoten mellan två relationer måste nämnarens kolumner uppfylla kraven på unions-kompatibilitet med en delmängd av täljarens kolumner (mao nämnaren måste vara lika med "svansen" i täljaren)

Kvot fortsättning...

Kvot är användbart för frågor av typ: Ta fram alla som gjort *allt*/har *alla* egenskaperna etc. T ex "Ta fram alla som ätit av alla glass-sorterna"

ÄTANDE

Namn	Glass
Maria	Päronsplitt
Maria	Vaniljpuck
Maria	Storstrut
Pelle	Vaniljpuck
Pelle	Storstrut
Stina	Vaniljpuck
Stina	Storstrut
Stina	Päronsplit

GLASS

Glass
Päronsplit
Storstrut
Vaniljpuck

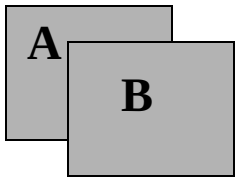
ÄTANDE ÷ GLASS

Namn
Maria
Stina

KVOT kallas även DIVISION

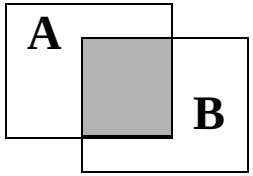
Sammanfattning av några relationsalgebraiska operatorer

UNION INTERSECT(= SNITT) DIFFERENCE



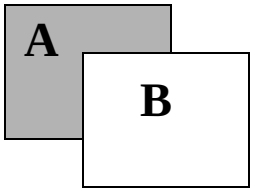
A UNION B

"Alla rader som förekommer i A *eller* B"



A INTERSECT B

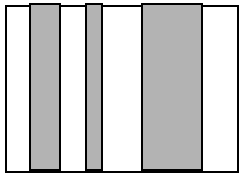
"Alla rader som förekommer i *både* A och B"



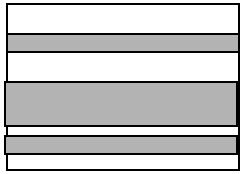
A DIFFERENCE B (A MINUS B)

De rader som förekommer i A men inte i B

PROJECTION



SELECTION



SQL - Structured Query Language

SQL har funktioner för att hantera:

databeskrivning, SQL-DDL (Data Definition Language)

databearbetning, SQL-DML (Data Manipulation Language)

behörighet, SQL-DCL (Data Control Language)

1986 antogs en internationell standard för data sub-language för relations-DBMS (SQL). 1992 publicerade ISO en standard som kallas SQL2 (eller SQL92). Denna standard överensstämde med aktuella SQL-dialekter för de vanligaste DBMS. En ny standard SQL3 är på väg.

SQL - Structured Query Language

SELECT Namn, Lön p

FROM ANSTÄLLD X

WHERE Lön > 17000 s

SQL kan uttrycka allt som går att uttrycka i relationsalgebran - språket är "relationally complete"

Nästlade frågor

ANSTÄLLD(Namn, Lön, Chef, Anamn)
AVDELNING(Anamn, Våning)

Ta fram namn och lön på alla anställda som har en chef som tjänar mer än 30000 kr:

```
SELECT Namn, Lön
FROM ANSTÄLLD
WHERE Chef IN
      (SELECT Namn
      FROM ANSTÄLLD
      WHERE Lön > 30000)
```

Här använde vi nästling som ett alternativ till att joina en tabell med sig själv! Hur skulle join:en se ut?

Nästlade frågor motsvarande "aldrig, inga etc."

Ta fram namn på alla personer som aldrig ätit glassen 'Päronsplitt'

```
SELECT Namn
FROM ÄTANDE
WHERE Namn NOT IN
      (SELECT Namn
      FROM ÄTANDE
      WHERE Glass='Päronsplitt')
```

```
SELECT Namn
FROM ÄTANDE
WHERE Glass ≠ 'Päronsplitt'
```

```
SELECTNamn
FROM ÄTANDE
EXCEPT
SELECT Namn
FROM ÄTANDE
WHERE Glass = 'Päronsplitt'
```

Alternativ

ÄTANDE

Namn	Glass
Maria	Päronsplitt
Maria	Vaniljpuck
Maria	Storstrut
Pelle	Vaniljpuck
Pelle	Storstrut
Stina	Vaniljpuck
Stina	Storstrut
Stina	Päronsplit

Nästlade frågor med NOT EXISTS

Ta fram namn på alla som ätit av alla glasstyperna:

```
SELECT Ä.Namn
FROM ÄTANDE Ä
WHERE NOT EXISTS
      (SELECT Glass
      FROM GLASS
      WHERE Glass NOT IN
            (SELECT Glass
            FORM ÄTANDE
            WHERE Namn = Ä.Namn))
```

ÄTANDE

Namn	Glass
Maria	Päronsplitt
Maria	Vaniljpuck
Maria	Storstrut
Pelle	Vaniljpuck
Pelle	Storstrut
Stina	Vaniljpuck
Stina	Storstrut
Stina	Päronsplit

GLASS

Glass
Päronsplit
Storstrut
Vaniljpuck

Resultat:

Namn
Maria
Stina

"Det får *inte* existera någon glass i GLASS som *inte* ätits av Ä.Namn"

Nästlade frågor forts.

Ta fram namn på alla som ätit samma glassar som Pelle:

```
SELECT Ä.Namn
FROM ÄTANDE Ä
WHERE NOT EXISTS
      (SELECT Glass
      FROM ÄTANDE
      WHERE Namn = Pelle
      AND Glass NOT IN
            (SELECT Glass
            FROM ÄTANDE
            WHERE Namn = Ä.Namn))
```

ÄTANDE

Namn	Glass
Maria	Päronsplitt
Maria	Vaniljpuck
Maria	Storstrut
Pelle	Vaniljpuck
Pelle	Storstrut
Stina	Vaniljpuck
Stina	Storstrut
Stina	Päronsplit

GLASS

Glass
Päronsplit
Storstrut
Vaniljpuck

"Det får *inte* existera någon glass som Pelle ätit som *inte* den vi söker ätit". Hmm... Räcker detta?

Nästlade frågor forts.

```
SELECT Ä.Namn
FROM ÄTANDE Ä
WHERE NOT EXISTS
      (SELECT Glass
      FROM ÄTANDE
      WHERE Namn = Pelle
      AND Glass NOT IN
            (SELECT Glass
            FROM ÄTANDE
            WHERE Namn = Ä.Namn))
AND NOT EXISTS
      (SELECT Glass
      FROM ÄTANDE
      WHERE Namn = Ä.Namn
      AND Glass NOT IN
            (SELECT Glass
            FROM ÄTANDE
            WHERE Namn = Pelle))
```

Ta fram namnen på de som ätit samma glassar som Pelle ätit:

"Det får inte existerar någon glass som Pelle ätit som inte de vi söker ätit. Det får heller inte existera någon glass som de vi söker ätit som inte Pelle ätit".

Resultat:

Namn
Pelle

Ingen hade ätit precis de glassar Pelle ätit (utom Pelle).

DISTINCT

SQL rensar *inte* duplikat automatiskt (vilket ju operatorerna i relations-algebra gjorde). För att eliminera dubletter anges DISTINCT.

Ta fram alla som ätit minst en glass som även Pelle ätit:

```
SELECT DISTINCT Namn
From ÄTANDE
WHERE Glass IN Pelles_glassar
```

Resultat:

Namn
Maria
Stina

```
SELECT Namn
From ÄTANDE
WHERE Glass IN Pelles_glassar
```

Resultat:

Namn
Maria
Maria
Maria
Stina
Stina
Stina

Inbyggda funktioner, aggregatfunktioner

COUNT(*), räknar antalet rader i en tabell
COUNT(kolumnnamn), räknar antalet värden i en kolumn
SUM(kolumnnamn), summerar värdena i en kolumn
AVG(kolumnnamn), tar genomsnittet av värdena i en kolumn
MAX(kolumnnamn), ger största värdet i en kolumn
MIN(kolumnnamn, ger minsta värdet i en kolumn

Exempel på aggregat-funktioner

```
SELECT COUNT(*)
FROM ÄTANDE
```

Ger antalet rader i tabellen ÄTANDE

Resultat:

8

```
SELECT COUNT(DISTINCT Namn)
FROM ÄTANDE
```

Ger antalet (unika) personer i tabellen ÄTANDE

Resultat:

3

Mer om nästlade frågor (subselect)

Vissa frågor kräver att man beräknar ett värde i databasen för att använda det i en WHERE-klausul. Detta kan endast lösas med en subselect (nåja en vy skulle också fungera).

Ta fram alla personer som ätit fler glassar än Pelle ätit

```
SELECT Namn, count(Glass)
FROM ÄTANDE
GROUP BY Namn
HAVING COUNT(Glass) > (SELECT count(Glass)
                       FROM ÄTANDE
                       WHERE NAMN = Pelle)
```

Resultat:

Namn	
Maria	3
Stina	3

UNION

Ta fram alla avdelningar som ligger på plan 1 eller avdelningar där personer är antällda som tjänar mer än 20000 eller båda delar:

Alternativ

```
SELECT Anamn
FROM AVDELNING
WHERE Våning = 1
UNION
SELECT Anamn
FROM ANSTÄLLD
WHERE Lön > 20000
```

```
SELECT AVDELNING.Anamn
FROM AVDELNING, ANSTÄLLD
WHERE AVDELNING.Anamn =
                ANSTÄLLD.Anamn
AND Våning = 1
OR Lön > 20000
```

ANSTÄLLD(Namn, Lön, Chef, Anamn)
AVDELNING(Anamn, Våning)

INTERSECT(SNITT)

Ta fram alla avdelningar som ligger på plan 1 OCH där alla personer som arbetar där tjänar mer än 20000:

ANSTÄLLD(Namn, Lön, Chef, Anamn)
AVDELNING(Anamn, Våning)

Alternativ

```
SELECT Anamn
FROM AVDELNING
WHERE Våning = 1
INTERSECT
SELECT Anamn
FROM ANSTÄLLD
WHERE Lön > 20000
```

```
SELECT AVDELNING.Anamn
FROM AVDELNING, ANSTÄLLD
WHERE AVDELNING.Anamn =
                ANSTÄLLD.Anamn
AND Våning = 1
AND Lön > 20000
```

A

EXCEPT(MINUS)

Ta fram alla avdelningar som inte har några personer som tjänar mer än 30000:

ANSTÄLLD(Namn, Lön, Chef, Anamn)
AVDELNING(Anamn, Våning)

Alternativ

```
SELECT Anamn
FROM AVDELNING
EXCEPT
SELECT Anamn
FROM ANSTÄLLD
WHERE Lön > 30000
```

```
SELECT Anamn
FROM AVDELNING
WHERE Anamn NOT IN
      (SELECT Anamn
       FROM ANSTÄLLD
       WHERE Lön > 30000)
```

Textsträngsmatchning

Ta fram alla anställda vars namn börjar på bokstaven n:

```
SELECT Namn
FROM ANSTÄLLD
WHERE Namn LIKE 'n%'
```

Kolumnnamn LIKE textkonstant, där textkonstant kan utgöras av
% (svarar mot 0 eller flera tecken) och/eller
_ (svarar mot ett godtyckligt tecken) och/eller
övriga tecken (som motsvarar sig själva).
'Mari- K%sson' svarar mot en textsträng som börjar på 'Mari'
och följs av minst ett godtyckligt tecken, ett blanktecken, ett K,
samt ett godtyckligt antal valfria tecken avslutat med bokstäverna
sson. Exempel: 'Maria Karlsson', 'Marie Klasson' men också 'Mari Ksson'.

I t ex Access svarar % mot * och _mot ?

Relationskalkyl (tuppelkalkyl)

- Relationsalgebra (procedurell, “hur”, “vilken sekvens av operationer”)
- Relationskalkyl (deklarativ, “vad”)
  - tuppelkalkyl
  - domänkalkyl

Frågespråk, t ex SQL baserar sig på algebra eller kalkyl, basformalismer

Relationsalgebra ≡ Tuppelkalkyl

↑

Samma uttryckskraft

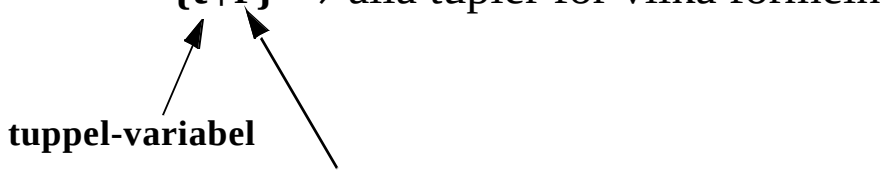
Algebrauttryck ↔ Kalkyluttryck

↑

Transformationsregler

Frågeuttryck

$\{t \mid F\} \Rightarrow$ alla tupler för vilka formeln F evaluerar till sant



F sant ≡ enligt databasens innehåll vid frågetillfället

$\{t \mid R(t)\}$ **"Alla tuper t som tillhör relationen R" ≡ Alla tupler t sådana att R(t) sant" ≡**
"Alla tupler t sådana att $t \in R$**"**

$t \in R$ **mängduttrykc**
R(t) predikatuttryck

$\{t.A_1, t.A_2, t.A_3 \mid R(t)\}$ **"attributen** A_1**,** A_2 **och** A_3 **i alla tupler av R"**
Relationsoperatorer $\{=, \neq, <, >, \leq, \geq\}$
Logiska operatorer $\{\wedge, \vee, \neg, \rightarrow\}$
Exempel: $\{t.A_1 \mid R(t) \wedge (t.A_1 > t.A_3 \vee t.A_2 < t.A_3)\}$

Evaluering av formler med logiska operatorer

t	s	¬t	t∧s	t∨s	t→s
1	1	0	1	1	1
1	0	0	0	1	0
0	1	1	0	1	1
0	0	1	0	0	1

Priorietsordning (om inte paranteser anger annat)

1. ¬
2. ∧
3. ∨
4. →

Kvantifierare

∀ **allkvantifierare (“alla”)**

∃ **allkvantifierare (“minst en”)**

Exempel: rödhårig(X) SANT eller FALSKT?

⇓

Vilken domän?

⇓

klassrummet!

⇓

Resultat: {Olle, Stina}

Detta är ÖPPEN formel (“variabeln X är inte i någon kvantifierares “scope”)

Kvantifierare

SLUTNA formler:

$(\forall x)$**(rödhårig (x))** **"För alla indivier i domänen gäller att att de är rödhåriga"**

$(\exists x)$**(rödhårig (x))** **"Det finns minst en individ i domänen som är rödhårig"**

Om vi nu vet att just Olle och Stina råkar vara rödhåriga så är den första formeln FALSK och den andra SANN. Väljer vi andra domäner kan sanningsvärdena får formlerna förstås ändras.

Kalkylens motsvarighet till algebraiska operationer:

R

A_1	A_2
1	a
1	b
2	a

S

A_2
a
b

T

A_1	A_2
1	a
1	b

$\pi_{A1}(\mathbf{R}) \Rightarrow \mathbf{\{t.A1 \mid R(t)\}}$

$\sigma_{A1=2}(\mathbf{R}) \Rightarrow \mathbf{\{t \mid R(t) \wedge t.A1=2\}}$

$\mathbf{R \cup T \Rightarrow \{t \mid R(t) \vee T(t)\}}$

$\mathbf{R - T \Rightarrow \{t \mid R(t) \wedge \neg T(t)\}}$

$\mathbf{R \times T \Rightarrow \{t \mid R(t) \wedge T(t)\}}$

$\mathbf{R \div S \Rightarrow \{t.A1 \mid R(t) \wedge (\forall s)(S(s) \rightarrow (\exists r)(R(r) \wedge (s.A2 = r.A2) \wedge (r.A1=t.A1))\}}$

ÄTANDE

Namn	Glass
Maria	Päronsplitt
Maria	Vaniljpuck
Maria	Storstrut
Pelle	Vaniljpuck
Pelle	Storstrut
Stina	Vaniljpuck
Stina	Storstrut
Stina	Päronsplit

"Ta fram de som ätit glassen 'Päronsplit'"

{t.Namn| ÄTANDE(t)∧t.Glass='Päronsplit'}

"Ta fram de som aldrig ätit glassen 'Päronsplit'"

{t.Namn| ÄTANDE(t)∧¬ t.Glass='Päronsplit'}

Blev det rätt? Nix, nu tog vi bara fram de som förekommer i relationen ÄTANDE på *någon* rad i kombination med något annat än just päronsplitt! En kvantifierare behövs!

{t.Namn| ÄTANDE(t)∧(¬∃t) (t.Glass='Päronsplit')}

ÄTANDE

Namn	Glass
Maria	Päronsplitt
Maria	Vaniljpuck
Maria	Storstrut
Pelle	Vaniljpuck
Pelle	Storstrut
Stina	Vaniljpuck
Stina	Storstrut
Stina	Päronsplit

GLASS

Glass
Päronsplit
Storstrut
Vaniljpuck

"Ta fram de som *enbart* ätit glassen 'Päronsplit'"

$$\{t.Namn \mid ÄTANDE(t) \wedge (\neg \exists t)\ (t.Glass \neq \text{'Päronsplit'})\}$$

"Ta fram de som ätit *alla* glass-sorter"

$$\{t.Namn \mid ÄTANDE(t) \wedge (\forall g)(Glass(g) \rightarrow (\exists ä)(ÄTANDE(ä) \wedge (g.Glass = ä.Glass) \wedge (ä.Namn = t.namn))\}$$

Omformningsregler

$$\neg(\forall x) \Leftrightarrow (\exists x)\neg$$
$$\neg(\exists x) \Leftrightarrow (\forall x)\neg$$

"Ta fram de som ätit *alla* glass-sorter" (en gång till...)

$\{t.Namn \mid ÄTANDE(t) \wedge (\forall g)(Glass(g) \rightarrow (\exists ä)(ÄTANDE(ä) \wedge$
$(g.Glass = ä.Glass) \wedge (ä.Namn = t.namn))\}$

"För alla glassar gäller att det ska finnas ett ätande som är kopplat till mitt ätande!

$\{t.Namn \mid ÄTANDE(t) \wedge \neg(\exists g)(Glass(g) \wedge \neg(\exists ä)(ÄTANDE(ä) \wedge$
$(g.Glass = ä.Glass) \wedge (ä.Namn = t.namn))\}$

"Det får inte finnas en enda glass för vilken det saknas ett ätande där jag är inblandad"

Bevis:

$\{t.Namn \mid ÄTANDE(t) \wedge (\forall g)(\neg Glass(g) \vee (\exists ä)(ÄTANDE(ä) \wedge (g.Glass = ä.Glass) \wedge (ä.Namn = t.namn))\}$

$\{t.Namn \mid ÄTANDE(t) \wedge (\forall g)\neg(Glass(g) \wedge \neg(\exists ä)(ÄTANDE(ä) \wedge (g.Glass = ä.Glass) \wedge (ä.Namn = t.namn))\}$

$\{t.Namn \mid ÄTANDE(t) \wedge \neg(\exists g)(Glass(g) \wedge \neg(\exists ä)(ÄTANDE(ä) \wedge (g.Glass = ä.Glass) \wedge (ä.Namn = t.namn))\}$

ANSTÄLLD(Namn, Lön, Chef, Avd)
AVDELNING(Anamn, Våning)

Uttryck "Ta fram alla namn och lön för alla anställda som arbetar på andra våningen" i Relationsalgebra, SQL och tuppelkalkyl:

$$\pi_{\text{Namn, Lön}}(\text{ANSTÄLLD} \underset{\text{ANSTÄLLD.Avd=AVDELNING.Anamn}}{\theta} \sigma_{\text{Våning=2}}(\text{AVDELNING}))$$

```
SELECT Namn, Lön
FROM ANSTÄLLD, AVDELNING
WHERE ANSTÄLLD.Avd = AVDELNING.Anmn
AND Våning=2
```

$$\{t.Namn, t.Lön \mid ANSTÄLLD(t) \wedge (\exists av)(AVDELNING(av) \wedge t.Avd = av.Anamn \wedge Våning = 2)\}$$

ÄTANDE

Namn	Glass
Maria	Päronsplitt
Maria	Vaniljpuck
Maria	Storstrut
Pelle	Vaniljpuck
Pelle	Storstrut
Stina	Vaniljpuck
Stina	Storstrut
Stina	Päronsplit

Uttryck "Ta fram de som ätit åtminstone samma glassar som Pelle (= alla Pelles glassar) " i relationsalgebra och tuppelkalkyl:

Alla_pelles_glassar := $\pi_{\text{Glass}}(\sigma_{\text{Namn='Pelle'}}(\text{ÄTANDE}))$

Ätit_alla_pelles_glassar := $\pi_{\text{Namn}}(\text{ÄTANDE} \div \text{Alla\_pelles\_glassar})$

$$\{ä.\text{Namn} \mid \text{ÄTANDE}(ä) \wedge (\forall pä)((\text{ätande}(ä) \wedge ä.\text{Namn}=\text{'Pelle'}) \rightarrow (\exists aä)(\text{ätande}(aä) \wedge (aä.\text{glass}=pä.\text{glass}) \wedge (aä.\text{Namn}=ä.\text{Namn})))\}$$