Relasjonsalgebra.

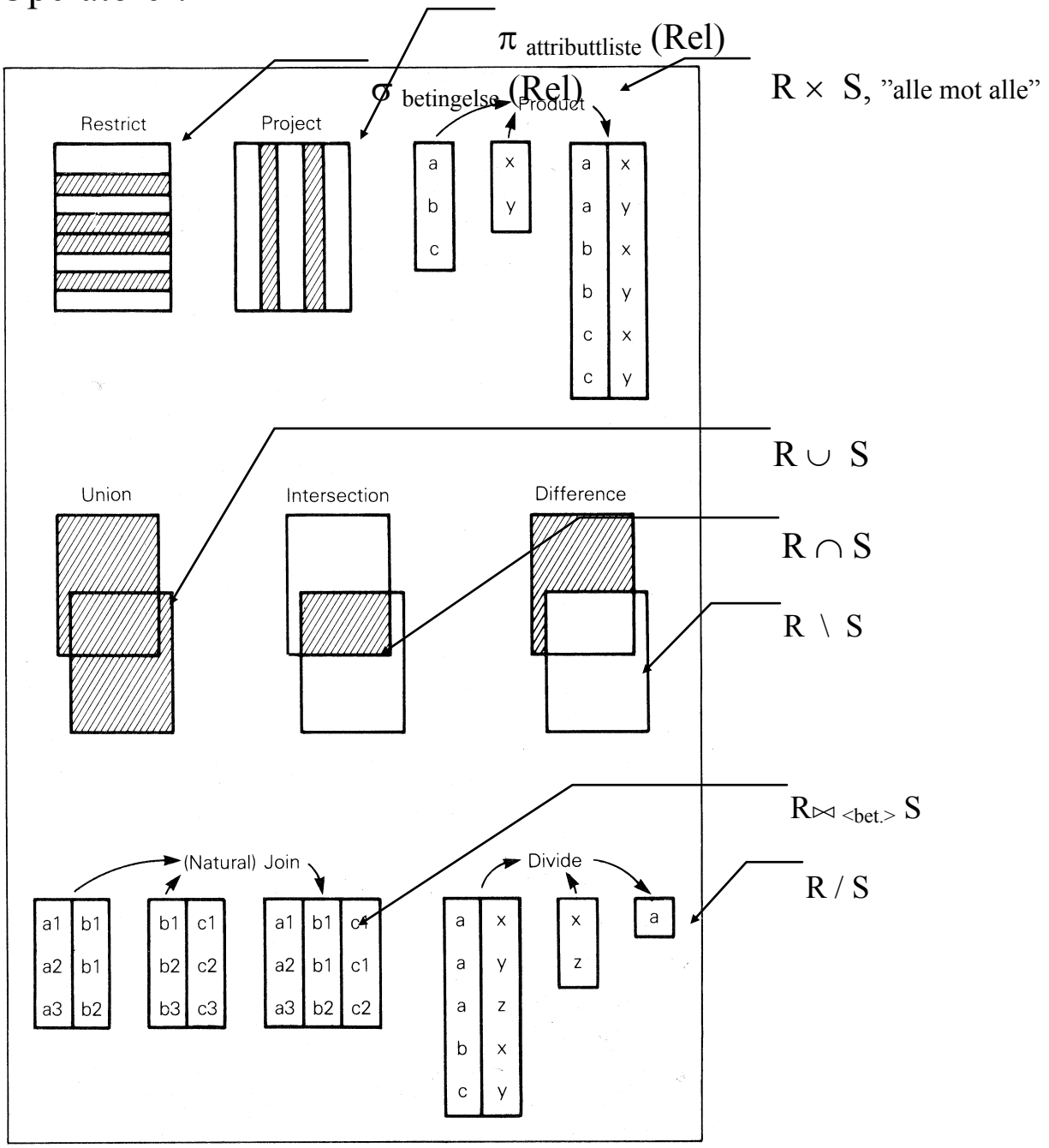
Hva?

Relasjonsalgebra består av et sett med høynivås operatorer som kan brukes til å manipulere med relasjoner (slå sammen to tabeller, selektere data etc.). Tankegangen er viktig å kjenne godt til dersom man skal arbeide seriøst med relasjonsdatabaser.

Brukes til bl.a.

- å forstå grunnlaget for relasjonsdatabaser og -spørrespråk
- å forstå hvorledes SQL-setninger kan utføres (og effektiviseres/optimaliseres)
- forstå og arbeide med datavarehus
- forstå og arbeide med distribuerte databaser

Operatorer:



(fra Date: Introduction to Database Systems, utgave 6)

Relasjonsalgebra - vanlige operasjoner.

Mengdeoperasjoner:	*Notasjon, variant 1*	*Notasjon, variant -2*
Union	$R \cup S$	R union S
Snitt	$R \cap S$	R intersect S
Mengdedifferanse	R - S R \ S	R difference S R minus S
Mengdeprodukt, kartesisk produkt ("alle mot alle")	$R \times S$	R product S R times S
Spesielt for relasjoner:		
Horisontalt utvalg (sigma)	$\sigma_{<beting.>}(R)$	R where <bet.> R where <bet.>
Vertikalt utvalg (pi)	$\pi_{<feltliste>}(R)$	R[<feltliste>]
Mengdedivisjon. (Gitt R[c,d] og S[d]. c er med i mengden R dividert med S hvis c i R forekommer sammen med alle d-er som finnes i S.)	$R \div S$ R / S	R divideby S
Spesialiteter av produkt:		
θ-join (produkt med en eller annen betingelse på kompatible attributter, f.eks. >, <, og komb.)	$R \bowtie_{<bet.>} S$	R join$_{<betingelse>}$ S (R join S) where <bet.>
Equi-join (θ-opersjonen er =)	" "	" "
Natural join (Equi-join hvor felles attributt kommer bare en gang) ** den mest vanlige jointypen **	" "	" "
Varianter for produkt:		
Outer join, normalt venstre.. (alle i R, samt alle fra S som oppfyller koblingsbetingelsen)	$R \overset{\bowtie}{\ }$ S	R left join$_{<bet.>}$ S
Full join (alle i R, alle i S, samt alle som oppfyller koblingsbet.)	$R \overset{\bowtie}{\ }$ S	R full join$_{<bet.>}$ S
Semijoin (de i R som tilfredsstiller R join$_{<betingelse>}$ S)	$R \triangleright_{<bet.>} S$	R semijoin$_{<bet.>}$ S

Legg merke til at operasjonene her er på mengder, slik at evt. dublikater tas bort – tilsvarende select distinct i SQL.

Dersom betingelsen er på primær/fremmednøkkelkomb., droppes ofte <bet>.

Eksempler: - med bruk av σ π × ⋈

Gitt tabellene:
Avdeling (Avdnr, Avdnavn, Etasjenr)
Ansatt (Ansnr, Etternavn, Fornavn, Avdnr, Postnr)
Post (Postnr, Poststed)

- Skriv ut alt om ansatte med ansattnr > 100

  $\sigma_{\text{ansnr} > 100}$ **(Ansatt)**

- Skriv ut ansattnr, etternavn og fornavn:

  $\pi_{\text{Ansnr,Etternavn, Fornavn}}$ **(Ansatt)**

- Skriv ut ansattnr og etternavn og fornavn for ansnr > 100:

  $\pi_{\text{Ansnr,Etternavn, Fornavn}}$ **(**$\sigma_{\text{ansnr} > 100}$ **(Ansatt))**

  evt. delt opp:

  R1 := $\sigma_{\text{ansnr} > 100}$ **(Ansatt)**

  R2 := $\pi_{\text{Ansnr,Etternavn, Fornavn}}$ **(R1)**

- Skriv ut avdnr, avdnavn og ansnr for alle med postnr < 2000:
  f.eks.:

  $\pi_{\text{avdnr, avdnavn, ansnr}}$ ($\sigma_{\text{postnr} < 2000}$(Ansatt ⋈ Avdeling))
  NB! Kan gjøres på flere andre måter.

- Skriv ut avdnr, avdnavn for avdelinger som holder til i 3. etasje og har minst en ansatt som bor i 'Bø'.
  f.eks.:

  R1 := $\sigma_{\text{poststed= 'Bø'}}$**(Post)**
  R2 := (R1 ⋈ Ansatt)
  R3 := π_{avdnr}**(R2)**
  R4 := R3 ⋈ Avdeling
  R5 := $\sigma_{\text{etasjenr} = 3}$**(R4)**
  R6 := $\pi_{\text{avdnr, avdnavn}}$ **(R5)**

  NB! Siden det er mengde-operasjoner, tas dublikater automatisk bort

  - Kunne vært samlet i en. Prøv!
  - Kunne vært gjort på mange måter

- Finn avdelingsnr for avdelinger uten ansatte

  π_{Avdnr} **(Avdeling) ** π_{Avdnr} **(Ansatt)**

- Finn avdelinger uten ansatte – ta med både avdelingsnr, -navn og etasje

  (π_{Avdnr} **(Avdeling) ** π_{Avdnr} **(Ansatt))** ⋈ **Avdeling**

Eksempler, forts: $\sigma\pi\times$ $\bowtie$

Gitt tabellene:
Avdeling (Avdnr, Avdnavn, Etasjenr)
Ansatt (Ansnr, Etternavn, Fornavn, Avdnr, Postnr) - fast ansatte
Post (Postnr, Poststed)
MAnsatt (Ansnr, Etternavn, Fornavn, Avdnr, Postnr) - midlertidige.

- Lag en liste over evt. som **både** er midlertidige og faste ansatte
  Mansatt $\cap$ Ansatt - NB! Forutsetter at like Ansnr == > likt på de andre
- Lag en liste over både midlertidige og faste ansatte
  Mansatt $\cup$ Ansatt - NB! Forutsetter at like Ansnr == > likt på de andre.
  Evt. dublikater forsvinner.

- Hvis noen er både midlertidige og faste, skal de slettes fra midlertidige
  Mansatt := Mansatt \ Ansatt $\cap$ Mansatt - NB! Forutsetter … som over.

- Finn alle postnr, og liste over de som har dette postnr

  $\pi_{\textbf{Postnr}}$ **(Post)** $\bowtie$ $\pi_{\textbf{Postnr, Ansnr, ansnavn}}$ **(Ansatt)**

- Skriv ut alle ansatte i avdelingen for Gressklippere
  enklest:
  Ansatt $\triangleright$ $\sigma_{\text{avdnavn = 'Gressklippere'}}$(Avdeling)

- Kombiner alle etternavn med alle fornavn
  π_{fornavn}**(Ansatt)** $\times$ $\pi_{\text{etternavn}}$**(Ansatt)**

- Finn postnr som er slik at minst en fra hver avdeling har dette postnr

  $\pi_{\text{avdnr, postnr}}$**(Ansatt)** / π_{avdnr}**(Avdeling)**

- Finn avdelingsnr som er slik at de har ansatte fra alle aktuelle postnr

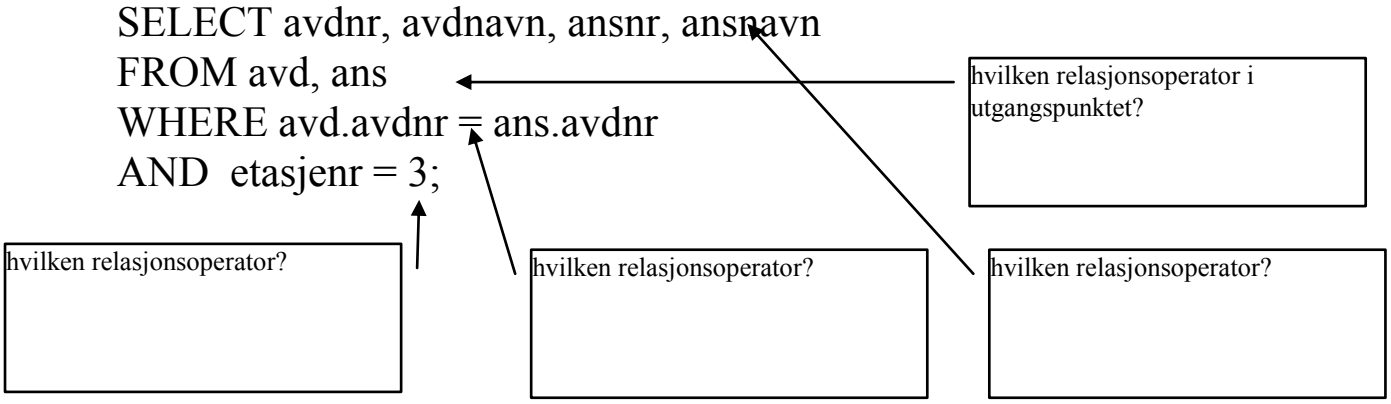
  $\pi_{\text{avdnr, postnr}}$**(Ansatt)** / π_{postnr}**(Post)**

Relasjonsalgebra vs. SQL

Det kan vises at alle utsagn som skrives i SQL kan skrives i relasjonsalgebra og motsatt – dvs. språkene er like kraftfulle.

Vi finner igjen relasjonsoperatorene i vanlige SQL-setninger:

Eks. 1:



Eks. 2:

```
SELECT avdnr
FROM ans
WHERE NOT EXISTS
  (SELECT postnr
  FROM post
  WHERE NOT EXISTS
    (SELECT postnr
     FROM ans ans2
    WHERE ans2.ansnr = ans.ansnr and ans.postnr = post.postnr)
```

dvs. "Selekter ut avdnr som er slik at det ikke eksisterer postnr som er slik at det ikke finnes korresponderende avdnr/postnr"

skriv en setning i relasjonsalgebra som gir samme resultat:

Noen regler/egenskaper i rel. algebra.

(temmelig teoretisk …..)

- **Lukkethet:**
  Relasjonsalgebra er lukket under relasjonsoperatorene.

- **Assoiativitet:**
  Union og snitt er assosiative:

$(R \cup S) \cup T \equiv R \cup (S \cup T)$ og

$(R \cap S) \cap T \equiv R \cap (S \cap T)$.

Med vår def. av produkt er også denne assosiativ.

- **Kommutativitet:**
  Union og snitt er kommutative:
  $R \cup S \equiv S \cup R$ og $R \cap S \equiv S \cap R$

  Det samme gjelder produkt og inner join.
  Derimot er ikke f.eks. mengdedifferanse og outer join.

- **Distributivitet**:
  I matematikk f.eks. $a \cdot (b+c) = a \cdot b + a \cdot c$,
  $\sqrt{x \circ y} = \sqrt{x} \circ \sqrt{y}$ er distributiv når operatoren $\circ$ er $\cdot$ eller /, men ikke dersom den er + eller -.

  - Restrict er distributiv mht. join i enkle operasjoner, f.eks.

    $\sigma_{<R\_betingelse> \text{ and } <S\_betingelse>}(R \bowtie S) \equiv \sigma_{<R\_betingelse>}(R) \bowtie \sigma_{<S\_betingelse>}(S)$

  - Projeksjon er distributiv mht. join (bortsett fra attributter som deltar i koblingen),

    $\pi_{<R\_attrliste> \cup <S\_attrliste>}(R \bowtie S) \equiv \pi_{<R\_attrliste>}(R) \bowtie \pi_{<S\_attrliste>}(S)$

disse brukes bl.a. i forbindelse med optimaliseringsstrategier

Utvidelser av relasjonsalgebra, renames.

Renames

- Vi har tidligere definert nye relasjoner ved tilordning, f.eks. Minavdeling := $\sigma_{avdelingsnr = 3}$ (Avdeling).

- Vi kan også lage en kopi av en relasjon, f.eks. S2 := S1, ligner alias, men lager kopi i stedet for å glemme det gamle navnet.

- Vi kan bruke operatoren ρ (rho, gresk r) for å gi et nytt navn på en beregning eller relasjon.
  - $\rho_{Minavdeling}(\sigma_{avdelingsnr = 3}$ (Avdeling)), er det samme som tilordningen over.
  - $\rho_{NyAnsatt}$(Ansatt) gjør at vi får en ny relasjon NyAnsatt.
  - Ønsker vi andre attributtnavn, er det også mulig, f.eks. $\rho_{Employee}$(Empno,LastName,FirstName) ($\pi_{Ansattnr, Etternavn, Fornavn}$(Ansatt)).

Beregninger via renames

Dermed kan man også bruke renames for å gjøre beregninger. Vi tenker oss at vi har en varerelasjon med bl.a. varenr og lengde i cm på varen. Vi vi ha en relasjon som passer for England, og vi skal ha med varenr og varenr og lengde både i cm og i tommer.

- $\rho_{EnglishItem}$(Itemno,CmLength,InchLength) ($\pi_{Varenr, Lengde, Lengde*2.54}$(Vare)).

Utvidelser av relasjonsalgebra, gruppering.

Summering / gruppering

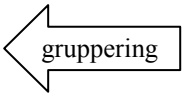
Tips: gjør en rask repetisjon på hvordan dette gjøres i SQL.

Siden vi her har bruk for et attributt for å «ta vare på» resultatet f.eks. av en summering i en ny tabell, trenger vi en ny relasjon og nytt/nye attributter. Eksempel:

Tell opp antall ansatte i hver avdeling. Legg den i en ny relasjon med NyAns, med attributtene Avdnr og AntallAnsatte.

NyAns

Avdnr	AntallAnsatte
17	2
34	3



Ansatt

Ansnr	Etternavn	Fornavn	…..	Avdnr
10001	….	…..	….	34
10011	…	…	….	17
19212	…..	…..	….	34
10004	…..	….	….	34
11221	….	….	….	17

$\rho_{NyAns(Avdnr, AntallAnsatte)}$ Avdnr , COUNT Ansnr (Ansatt)

Lag en ny relasjon — grupper på Avdnr — Tell opp ant.ansatte

Skal vi gjøre flere grupperingsoperasjoner, er det også mulig. Anta at vi har et attributt Lønn for hver ansatt. Vi kan da skrive

$\rho_{NynyAns(Avdnr, AntallAnsatte, SumLønn)}$ Avdnr, COUNT Ansnr SUM Lønn (Ansatt)

Tilsvarende hvis man skal gruppere på flere attributter, f.eks. primært på fylke, og så på kommune innenfor hvert fylke. Kommaen blir da skilletegn mellom attributtet det grupperes på og operatorene.

Hvilke grupperingsoperasjoner? De vanlige:

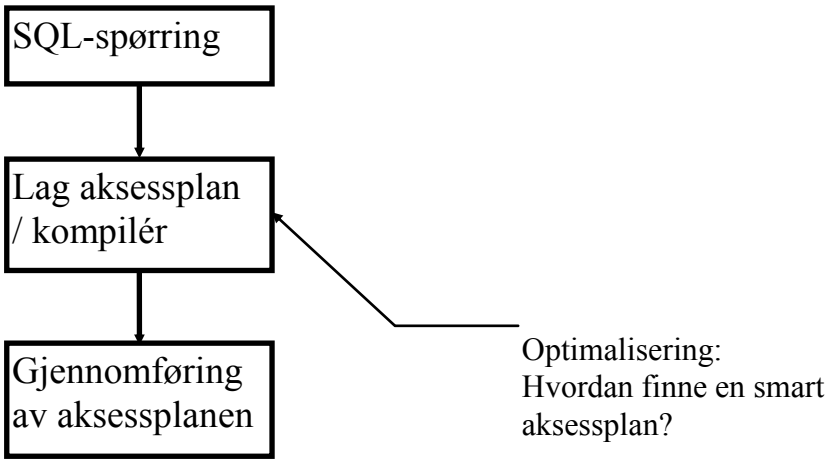
- COUNT
- SUM
- AVG
- MIN
- MAX

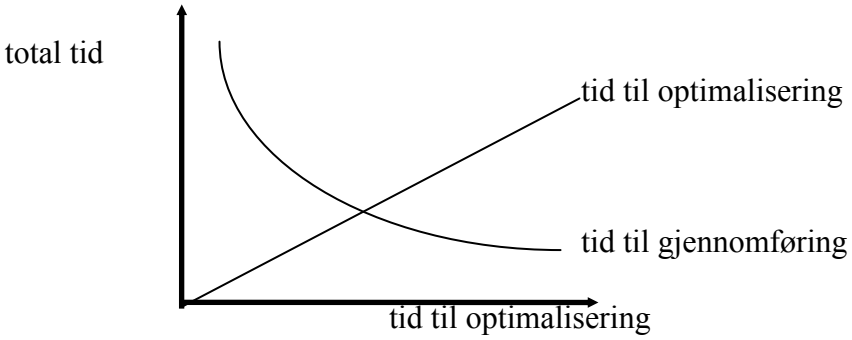
Bruksområder for relasjonsalgebra

- allmenn begrepsdannelse
- forståelse av hva som foregår "bak kulissene"
- mulighet for å sjekke kompletthet av et språk
- forståelse av optimaliseringsprosesser
- forståelse av distribuerte databaser (bl.a. hvordan distribuere data i et nettverk)
- kunne vært brukt som spørrespråk, inkl. endring av databasen, men neppe særlig praktisk.
- m.m.

Optimalisering

Hvorledes gjennomføres en spørring?





===> Kan ikke analysere alt for mye for å finne den optimale aksessplanen !!

Spørringer kan være
- adhoc, en kompilering + direkte gjennomføring
- lagrede, en kompilering + $\overset{m}{\underset{0}{}}${gjennomføring}

Tidkrevende / "tunge" operasjoner:

- sortering
- fjerning av dublikater
- gruppering
  - oftest lønnsomt å sortere først
- kobling (join)
  - "rå kraft" (alle mot alle, $O(n^2)$)
  - sorter + sammenlign (evt. via fletting)
  - sammenlign via indekser (full søk på en relasjon, via indeks på den andre)
  - hash-metodikk

Optimalisering - hvorledes?

Skritt i optimaliseringsprosessen:

- **nedbrytning** til enkeltoperasjoner og kanonisk form.
- **analyse** med omformulering i hht. lovlige regler, bl.a. ut fra
    - betingelser i where-setningen (= vs. > etc.)
    - hvilke attributter som skal med i resultatet
    - krav til sortering og gruppering
    - antall tupler pr. relasjon
    - antall bytes i attributter som skal kobles
    - indekser som finnes (og om clustede, unike, ikke-unike)
    - ideelt sett hvor dataene finnes (lokalt, på nettet etc.)
    - ideelt sett også variasjonsbredden i dataene.
- **sammenstilling** til en samlet aksessplan

Enkle optimaliseringskriterier:

- **Overordnet: begrens data mest mulig før tunge operasjoner utføres.**
- Projeksjoner gjøres først, men reelt bare hvis relasjoner trengs til mellomregninger, ellers bare ved å vite hvilke attributter som skal med
- Restrict / select-setninger med = verdi/verdisett gjøres først, og de som begrenser mest gjøres først.

    Eks: $\sigma_{R.a = <v>}(R \bowtie S) \Longrightarrow \quad \sigma_{R.a = <v>}(R) \bowtie S$

- Operasjoner som bør benytte seg av indekser gjøres først
- Operasjoner som gjelder primærnøkkelen gjøres først
- Sjekk om data i en subselect er stabile eller om de må beregnes for hver gang utenforliggende løkke beregnes.
- Boolske shortcut og forenklinger, f.eks.: P and P ≡ P; P and false ≡ false; P or false ≡ P; P and not P ≡ false; P and (P or Q) ≡ P

NB! Antall kombinasjoner øker svært fort, så effektive avskjæringsmekanismer er nødvendig.

Optimalisering - eksempel:

Gitt en Avdelings- og ansattabell.
Avdeling (avdnr, avdnavn, ……) 5 attributter, 100 tupler
Ansatt (ansnr, ansnavn, avdnr, adresse, postnr) 15 attributter, 5000 tupler

Finn ansnr & -navn for de(n) i Oste-avd. som har bor på postnr 1855.

```
SELECT   ansnr, ansnavn
FROM     Avdeling, Ansatt
WHERE    Avdeling.avdnr = Ansatt.avdnr
     AND postnr = '1850'
     AND avdnavn = 'Ost'
```

Alternativ 1:

$S1 := \sigma_{Avdeling.Avdnr = Ansatt.Avdnr} (Avdeling \bowtie Ansatt)$

-- 19 attributter, 500.000 poster kobles, resultat ≤ 5000 poster

$S2 := \sigma_{Postnr = '1850'} (S1)$

$S3 := \sigma_{Avdnavn = 'Ost'} (S2)$

$S4 := \pi_{ansnr, ansnavn}(S3)$ -- fra 19 til 2 attributter

Alternativ 2:

Skritt 1: a) Sett betingelser som gjelder bare en tabell sammen med denne tabellen (kalles detachment = frakobling)
b) Fjern overflødige attributter
Skritt 2: Gjør nødvendige koblinger
Skritt 3: Fjern evt. overflødige attributter

$S1 := \sigma_{Avdnavn = 'Ost'} (Avdeling)$ -- antagelig bare 1 post

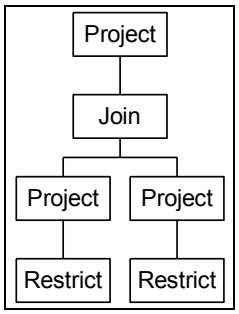
$S2 := \pi_{avdnr}(S1)$ -- og en kolonne / attributt

$S3 := \sigma_{Postnr = '1850'} (Ansatt)$ -- en/noen få rader / tupler

$S4 := \pi_{avdnr, ansnr, ansnavn} (S3)$ -- og 3 kolonner / atrributter

$S5 := S2 \bowtie S4$ -- 1 x 1 / svært få å koble

$S6 := \pi_{ansnr, ansnavn} (S5)$ -- kobl.attributtet skal ikke med i resultatet



I en jafs:

$\pi_{ansnr,ansnavn}$ (
π_{avdnr} ($\sigma_{Avdnavn = 'Ost'}$ (Avdeling))
$\bowtie$
$\pi_{avdnr,ansnr,ansnavn}$ ($\sigma_{Postnr = '1850'}$ (Ansatt)))

Optimalisering i databasesystemer:

I teorien:

- Alle spørringer som er ekvivalente burde gi opphav til samme aksessplan.

I praksis:

- Stor forskjell på databasesystemer m.h.t. optimalisering.
- I mange systemer kan man titte på, evt. også endre aksessplanen.
- Med mangeprosessormaskiner eller maskiner koblet i klynge blir parallelliserbarhet et viktig kriterium for
    - hele aksessplanen
    - enkeltalgoritmer f.eks. for sortering og kobling

Relasjonskalkyle I.

I stedet for å basere et språk på mengder, kan det baseres på vanlig predikatlogikk. For én variabel:

{ x | P(x) }, P er et utsagn som er sant eller usant avhengig av x.
Mer kompliserte uttrykk kan settes sammen av:

	symbol	alternativt symbol
og	$\wedge$	AND
eller	$\vee$	OR
negasjon / ikke	$\neg$ eller ~	NOT
for alle	$\forall$	FORALL
det eksisterer minst en	$\exists$	EXISTS

Eksempler:

Gitt: Range of x_ans is Ansatt; Range of x_avd is Avdeling;

1. Finn ansattnr med postnr = 1500 og avdnr = 17:

{x_ans.ansattnr | x_ans.postnr = 1500 $\wedge$ x_ans.avdnr = 17}

2. Finn alle avdelinger som har minst en ansatt:

{x_avd | $\exists$ (x_ans | x_avd.avdnr = x_ans.avdnr)}

3. Finn alle avdelinger uten ansette

{x_avd | $\neg\exists$ (x_ans | x_avd.avdnr = x_ans.avdnr)}

4. Finn om det er én avdeling som har alle ansatte:

{x_avd | $\forall$ (x_ans | x_avd.avdnr = x_ans.avdnr)}

5. Skriv ut avdelinger som har samme navn.
Vi trenger to avdelingsvariable, slik at vi kan sammenligne alle mot alle.
Range of y_avd is avdeling;

{x_avd | $\exists$ (y_avd | (x_avd.avdnavn = y_avd.avdnavn) $\wedge$
(x_avd.avdnr <> y_avd.avdnr)))}

Relasjonskalkyle II.

WFF, frie og bundne variable.

Et syntaktisk og semantisk korrekt utsagn kalles en wff, well-formed formula.

Dersom F er en WFF, så gjelder

$\{\forall x : F(x)\} \equiv \{ \neg\exists x \mid \neg F(x) \}$ - dvs.: $\forall$ er egentlig ikke nødvendig. SQL har ikke noen FORALL, derfor må denne omformuleringen brukes.

Tilsvarende :
$\{ \exists x \mid F(x) \} \equiv \{\neg(\forall x : \neg F(x))\}$
$\{ \forall x : F(x) \wedge G(x) \} \equiv \{\neg\exists x (\neg F(x) \vee \neg G(x))\}$ og

Variablene er av to typer:

Frie variable	"løper over alle verdier"
Bundne variable	frie variable som knytter seg til formler som har $\forall$ eller $\exists$

NB! Sammenlign med frie og bundne kontroller i 4.gen.syst.

Relasjonskalkyle - konkluderende punkter:

- like kraftfullt som relasjonsalgebra
- men helt non-prosydurelt
- SQL bygger mye på tenkningen i relasjonskalkyle.