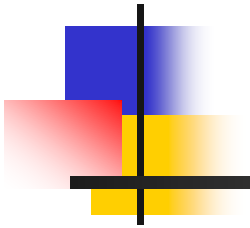
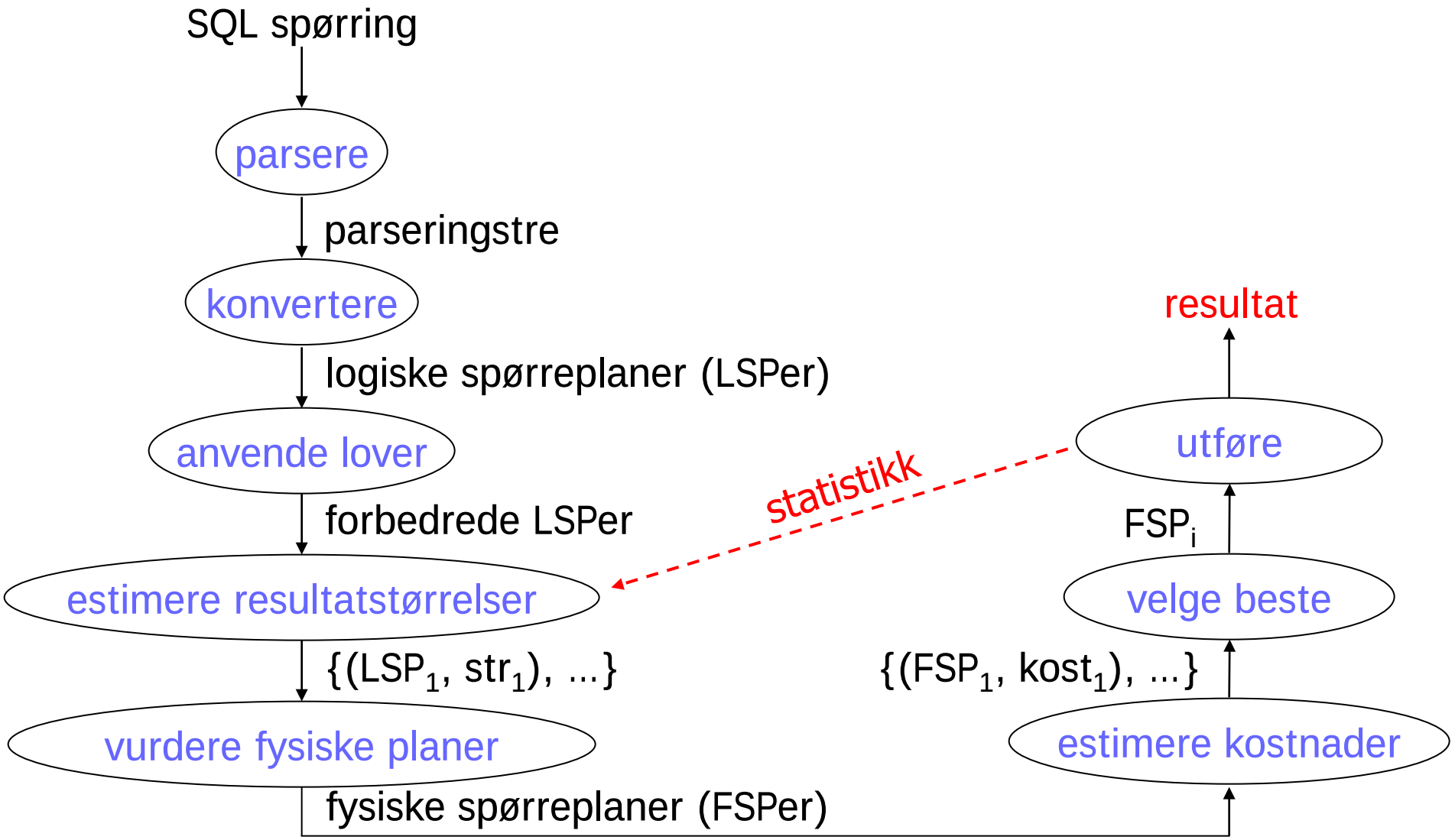


Spørsmålskompilering

Basert på foiler av
Hector Garcia-Molina

Oversikt: Fra spørring til resultat



Eksempel

```
SELECT B,C,Y
FROM R,S
WHERE W = X AND A = 3 AND Z = “a”
```

Relasjon R

A	B	C	...	W
1	z	1	...	4
2	c	6	...	2
3	r	8	...	7
4	n	9	...	4
2	j	0	...	3
3	t	5	...	9
7	e	3	...	3
8	f	5	...	8
1	h	7	...	5

Relasjon S

X	Y	Z
1	a	a
2	f	c
3	t	b
4	b	b
7	k	a
6	e	a
7	g	c
8	i	b
9	e	c

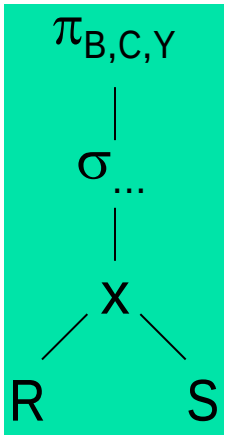
Eksempel (forts)

```
SELECT B,C,Y
FROM R,S
WHERE W=X AND A=3 AND Z="a"
```

ide 1 – ta kartesisk produkt, velg tupler, projiser attributter

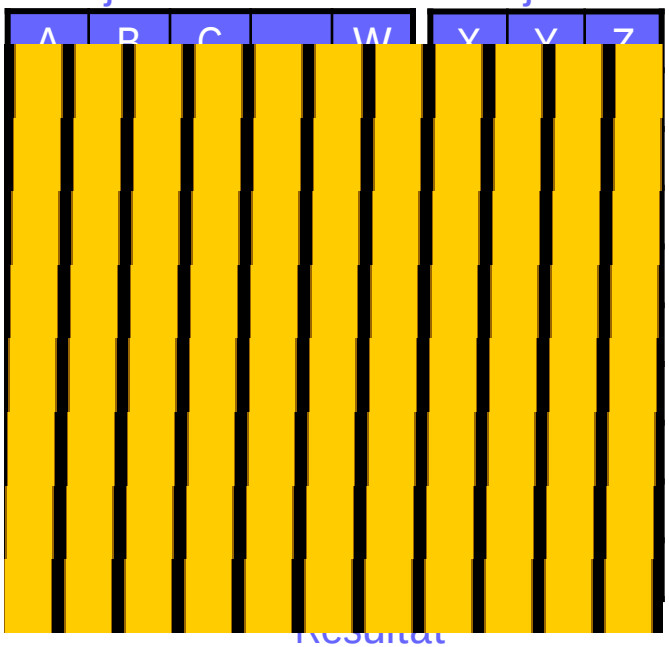
⇨ $\pi_{B,C,Y}\ (\sigma_{W=X\ \wedge\ A=3\ \wedge\ Z=\text{“a”}}\ (R \times S))$

Merk:
#attributter = #R-attributter + #S-attributter
#tupler = #R-tupler * #S-tupler



Relasjon R

Relasjon S



A	B	C	...	W	X	Y	Z
1	z	1	...	4	1	a	a
...	...	...	...	...	2	f	c
...	...	...	...	...	...	...	...
2	c	6	...	2	1	a	a
...	...	...	...	...	2	f	c
...	...	...	...	...	...	...	...
3	r	8	...	7	1	a	a
...	...	...	...	...	2	f	c
...	...	...	...	...	...	...	...
3	r	8	...	7	7	k	a
4	n	9	...	4	1	a	a
...	...	...	...	...	2	f	c
...	...	...	...	...	...	...	v
2	j	0	...	3	1	a	a
...	...	...	...	...	2	c	c
...	...	...	...	...	...	...	...
3	t	5	...	9	1	a	a
...	...	...	...	...	2	f	c
...	...	...	...	...	...	...	...
7	e	3	...	3	1	a	a
...	...	...	...	...	2	f	c
...	...	...	...	...	...	...	v
...	...	...	...	...	...	...	...

Resultat

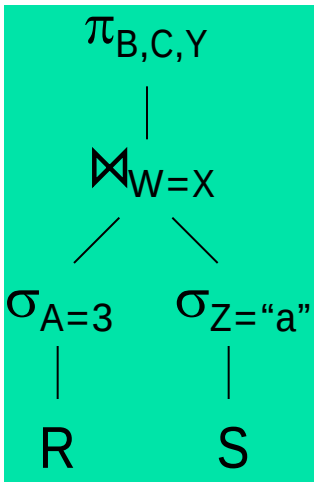
B	C	Y
r	8	k

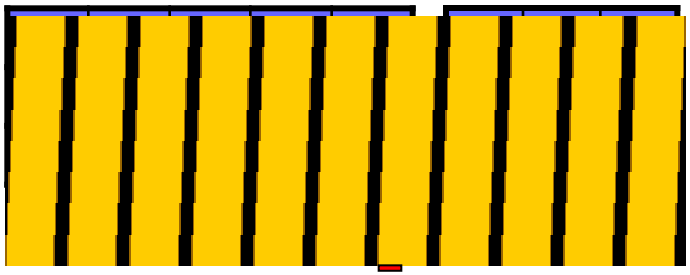
Eksempel (forts)

```
SELECT B,C,Y
FROM R,S
WHERE W=X AND A=3 AND Z="a"
```

ide 2 – velg tupler, gjør eqvijoin, projiser attributter

⇨ $\pi_{B,C,Y}((\sigma_{A=3}(R)) \bowtie_{W=X} (\sigma_{Z="a"}(S)))$





Relasjon R

A	B	C	...	W
1	z	1	...	4
2	c	6	...	2
3	r	8	...	7
4	n	9	...	4
2	j	0	...	3
3	t	5	...	9
7	e	3	...	3
8	f	5	...	8
1	h	7	...	5

A	B	C	...	W	X	Y	Z
3	r	8	...	7	7	k	a

B	C	Y
r	8	k

Relasjon S

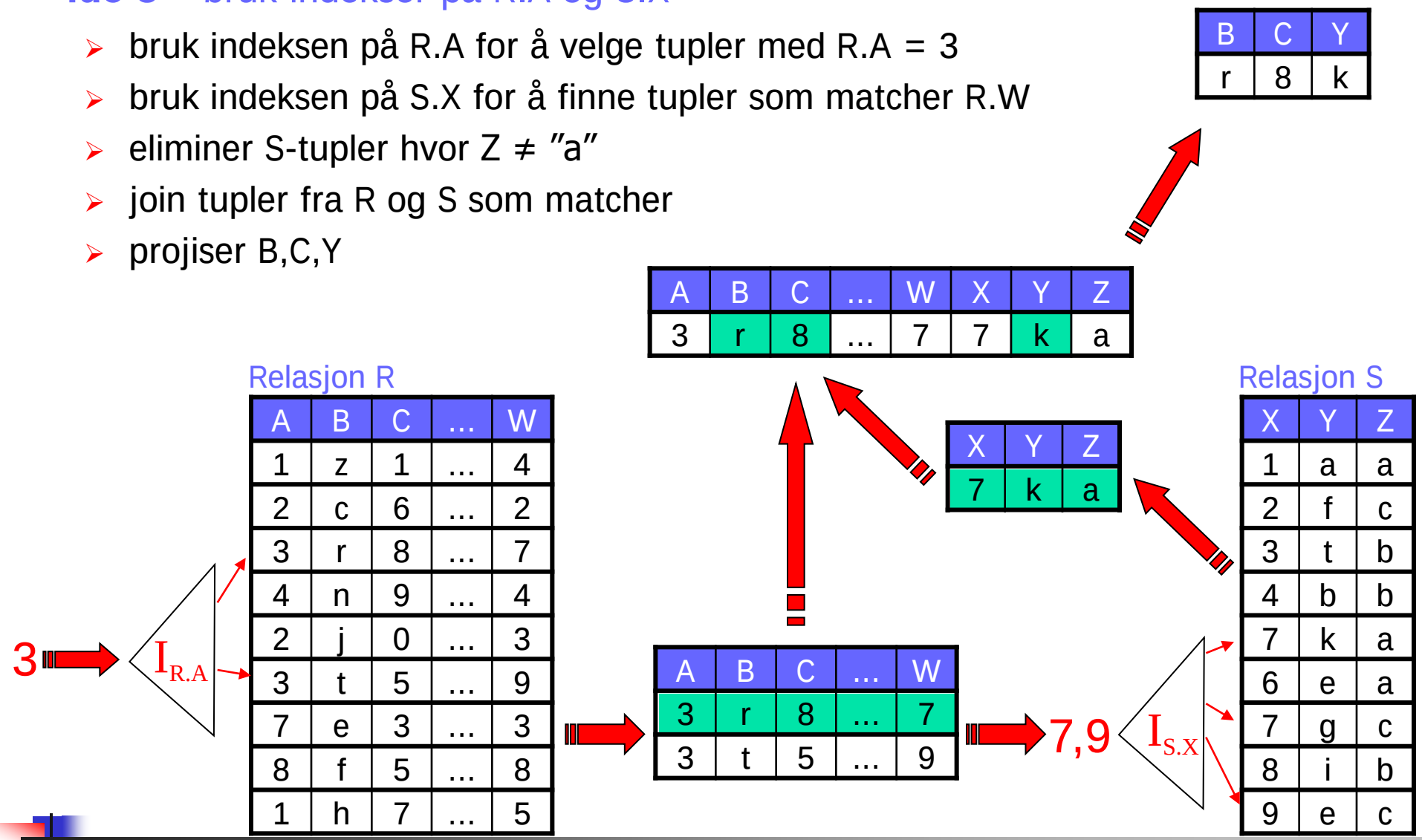
X	Y	Z
1	a	a
2	f	c
3	t	b
4	b	b
7	k	a
6	e	a
7	g	c
8	i	b
9	e	c

Eksempel (forts)

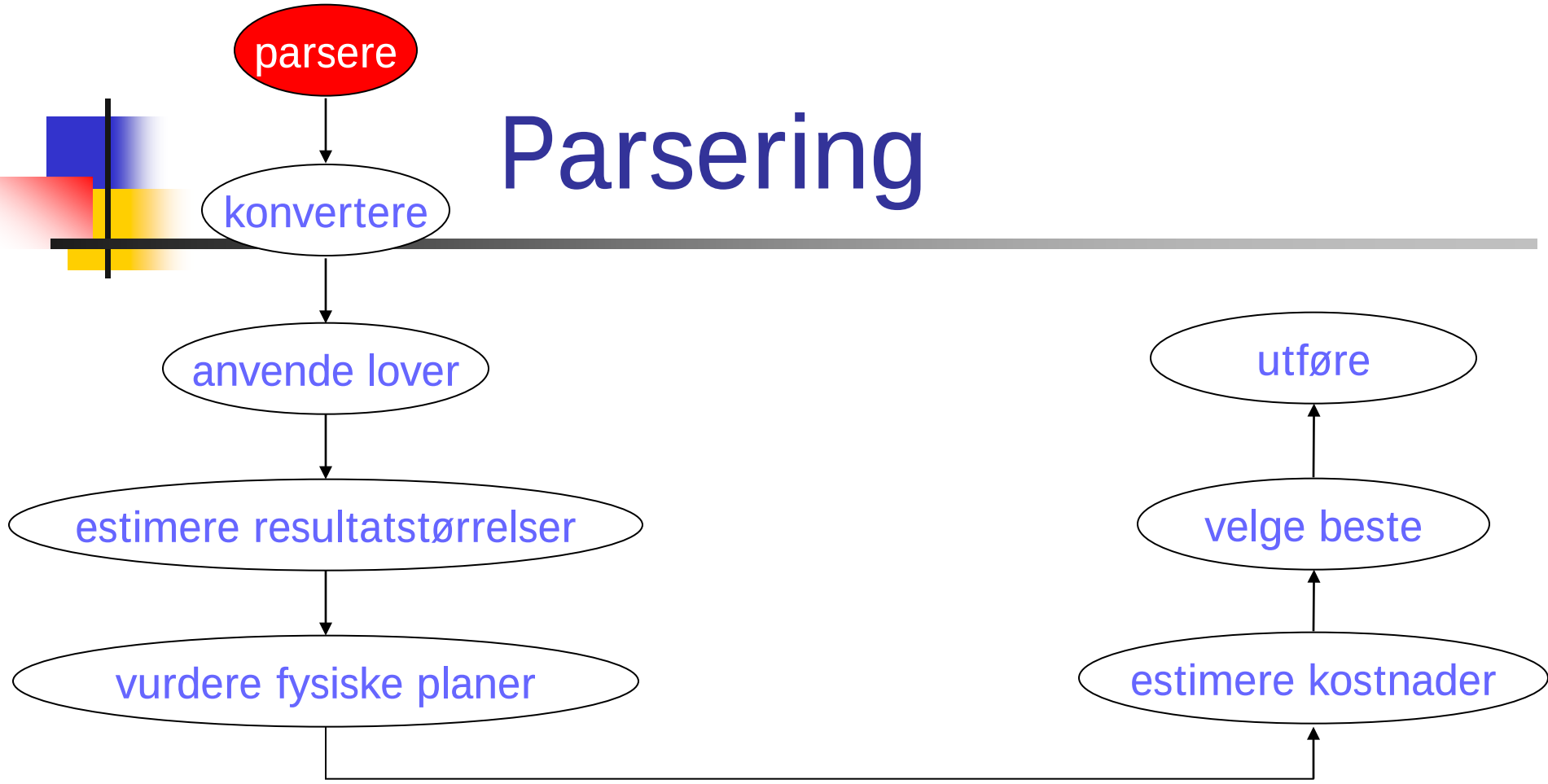
```
SELECT B,C,Y
FROM R,S
WHERE W=X AND A=3 AND Z="a"
```

ide 3 – bruk indekser på R.A og S.X

- bruk indeksen på R.A for å velge tupler med R.A = 3
- bruk indeksen på S.X for å finne tupler som matcher R.W
- eliminer S-tupler hvor Z ≠ "a"
- join tupler fra R og S som matcher
- projiser B,C,Y

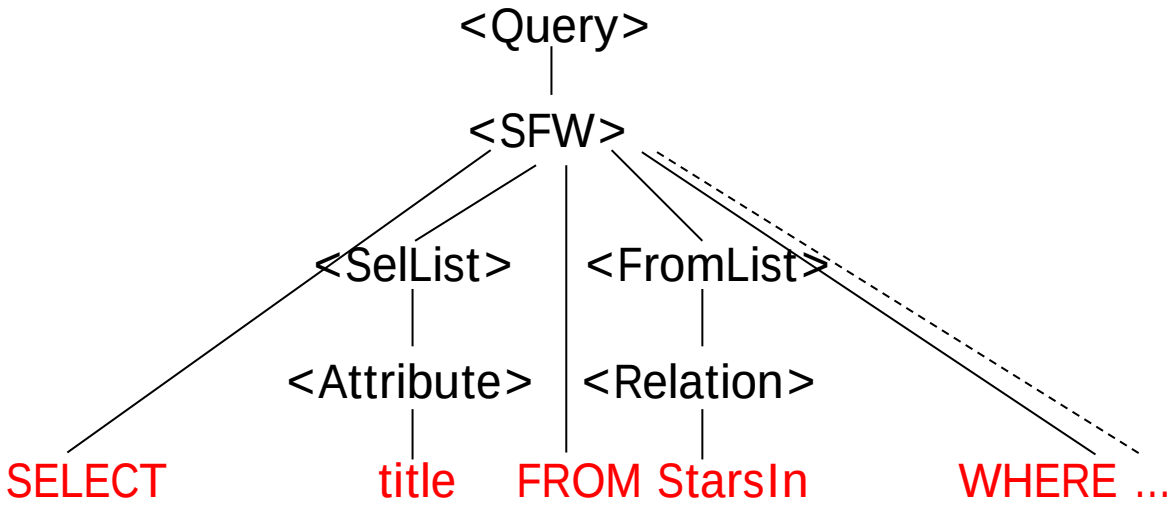


Parsering


parsere
konvertere
anvende lover
estimere resultatstørrelser
vurdere fysiske planer
utføre
velge beste
estimere kostnader

Parsering

- Mål: konvertere en SQL spørring til et parseringstre.



- Hver node i et parseringstre er enten:
  - *atomer* – leksikalske elementer som nøkkelord, navn, konstanter, parenteser og operatorer. (Løvnoder.)
  - *syntaktiske* kategorier – deler av spørringer. (Indre noder.)

Enkel grammatikk

- Query:
  - <Query> ::= <SFW>
  - <Query> ::= **(** <Query> **)**
  - en komplett grammatikk vil også inneholde operatorer som UNION, JOIN, ...
  - regel 2 brukes typisk i sub-spørringer
- Select-From-Where:
  - <SFW> ::= SELECT <SelList> FROM <FromList> WHERE <Condition>
  - en komplett grammatikk må inkludere GROUP BY, HAVING, ORDER BY, ...
- Select list:
  - <SelList> ::= <Attribute>
  - <SelList> ::= <Attribute>, <SelList>
  - en komplett grammatikk må inkludere uttrykk og aggregatfunksjoner
- From list:
  - <FromList> ::= <Relation>
  - <FromList> ::= <Relation>, <FromList>
  - en komplett grammatikk må inkludere aliasing og uttrykk som R JOIN S

Enkel grammatikk (forts)

- Condition:
  - <Condition> ::= <Condition> AND <Condition>
  - <Condition> ::= <Tuple> IN <Query>
  - <Condition> ::= <Attribute> = <Attribute>
  - <Condition> ::= <Attribute> LIKE <Pattern>
  - en komplett grammatikk må inkludere operatorer som OR, NOT, osv. og alle andre sammenligningsoperatorer
- Tuple:
  - <Tuple> ::= <Attribute>
  - en komplett grammatikk må inkludere tupler med flere attributter, ...
- Grunnleggende syntaktiske kategorier som <Relation>, <Attribute>, <Pattern>, osv. har ikke egne regler, men erstattes av et navn eller en tekststreng

Enkel grammatikk: eksempel

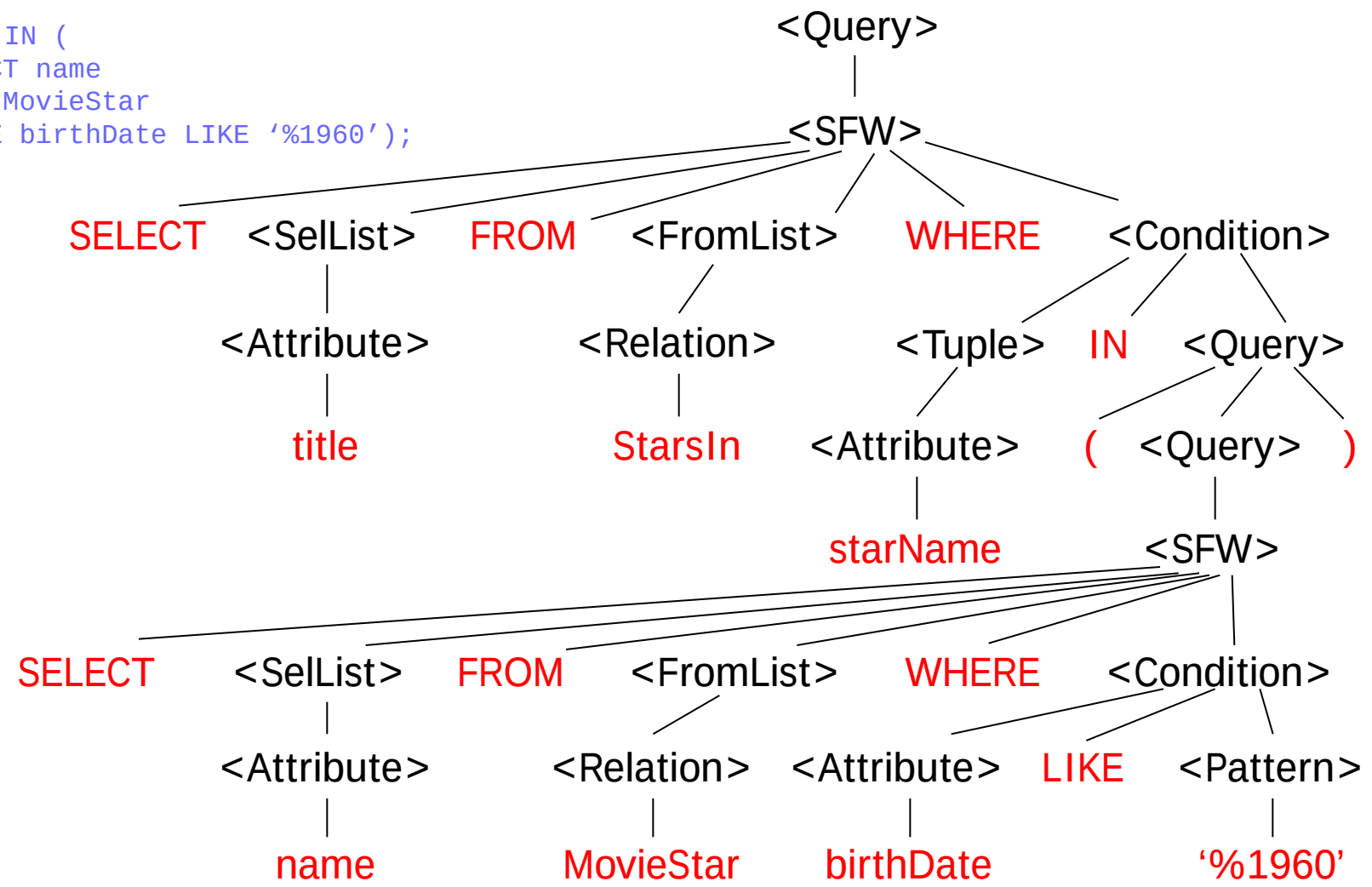
Finn filmer med skuespillere født i 1960:

```
SELECT title
FROM StarsIn
WHERE starName IN (
      SELECT name
      FROM MovieStar
      WHERE birthDate LIKE '%1960');
```

Enkel grammatikk: eksempel

Finn filmer med skuespillere født i 1960:

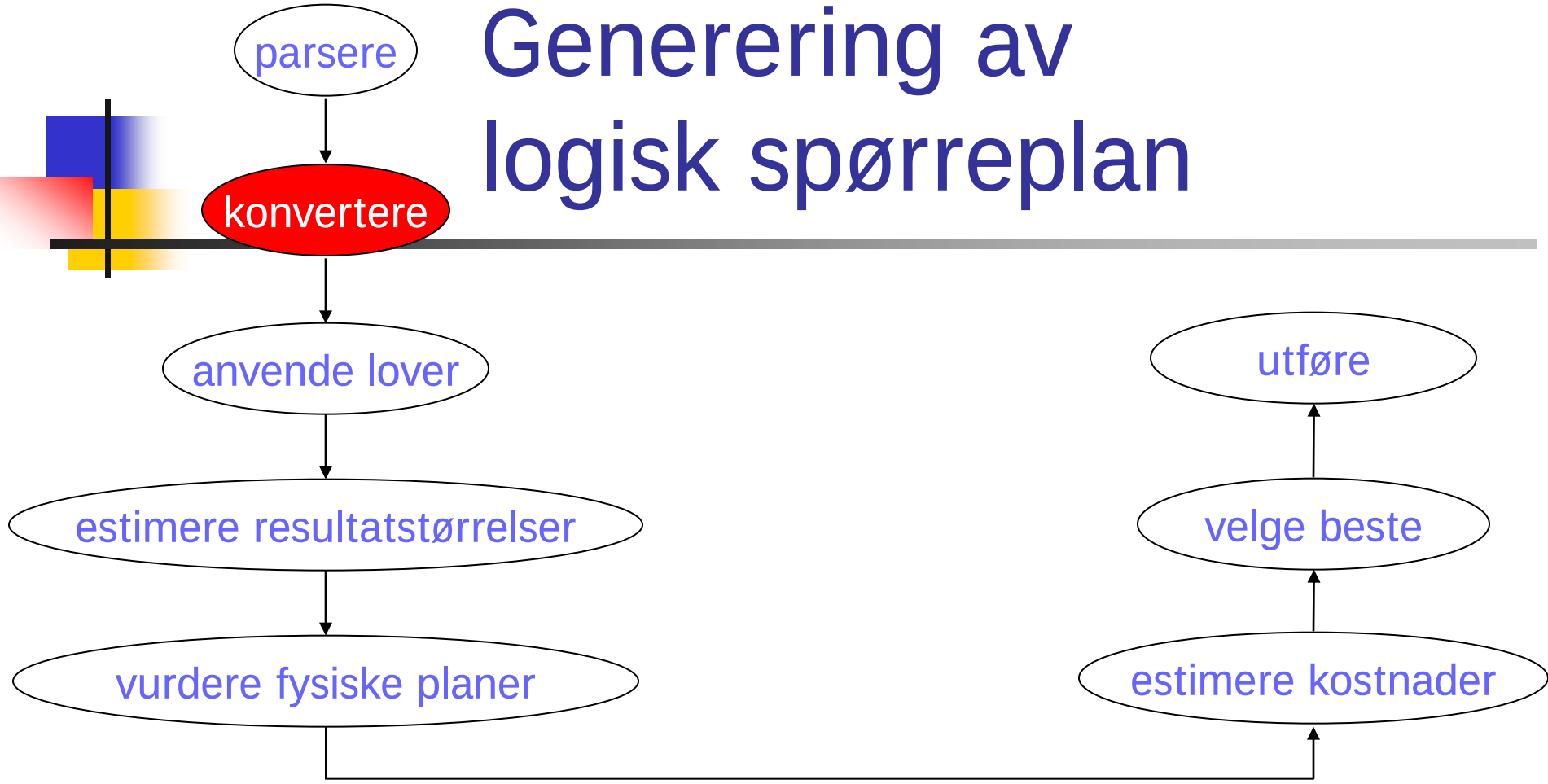
```
SELECT title
FROM StarsIn
WHERE starName IN (
          SELECT name
          FROM MovieStar
          WHERE birthDate LIKE '%1960');
```



Preprosessor

- Sjekker at spørringen er semantisk korrekt:
  - *relasjoner* – hver relasjon i FROM må være en relasjon eller et view i skjemaet hvor spørringen utføres.
    Hvis det er et view, må det erstattes av et nytt (sub-) parseringstre.
  - *attributter* – hvert attributt må finnes i en av relasjonene i det aktuelle skopet i spørringen.
  - *typer* – alle bruk av attributter må være i henhold til typene deres.

Generering av logisk spørreplan



Konvertering av SFW

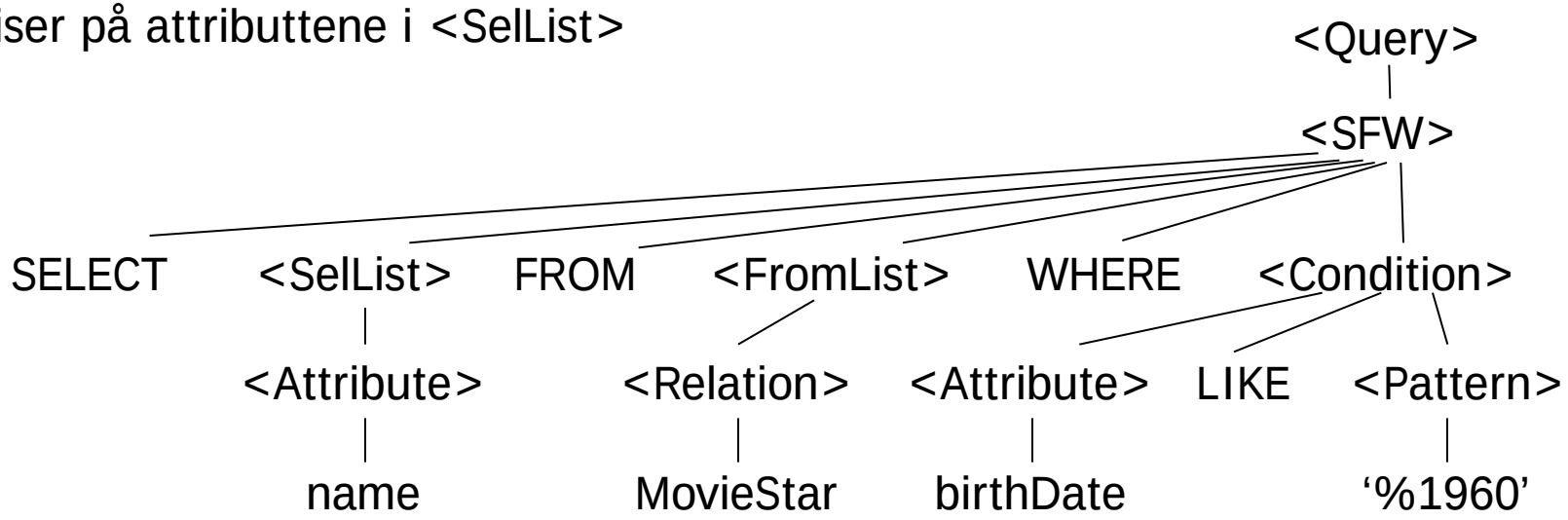
SELECT <SelList> FROM <Fromlist> WHERE <Condition>

- erstatt relasjonene i <FromList> av produktet, x, av alle relasjonene
- dette produktet er argumentet til en seleksjon, σ_C, hvor C tilsvarer <Condition>
- denne seleksjonen er argumentet til en projeksjon, π_L, hvor L er listen av attributter i <SelList>

Konvertering av SFW – eksempel

```
SELECT name FROM MovieStar WHERE birthDate LIKE '%1960'
```

- produktet av relasjonene i <FromList>
- gjør seleksjon basert på <Condition>
- projiser på attributtene i <SelList>



π_{name}

|

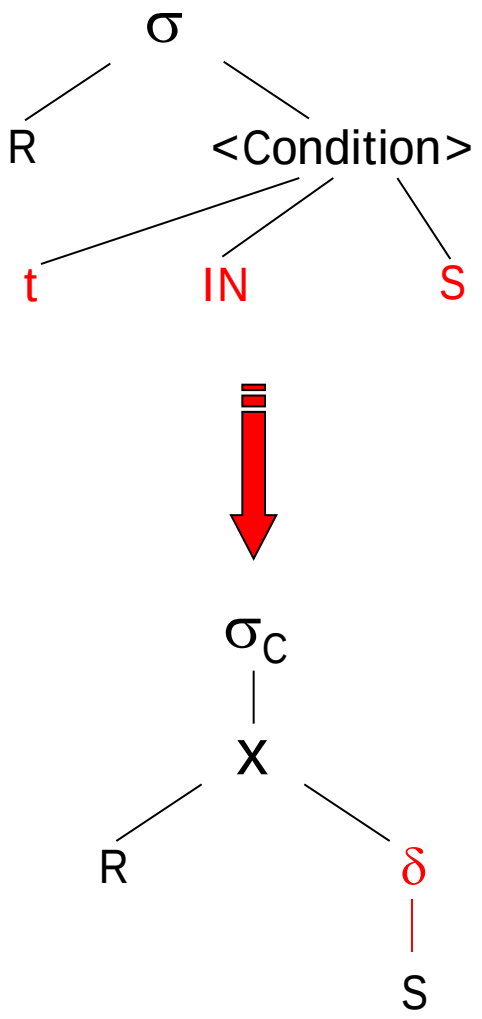
$\sigma_{birthDate\ LIKE\ '\%1960'}$

|

MovieStar

Konvertering av sub-spørringer

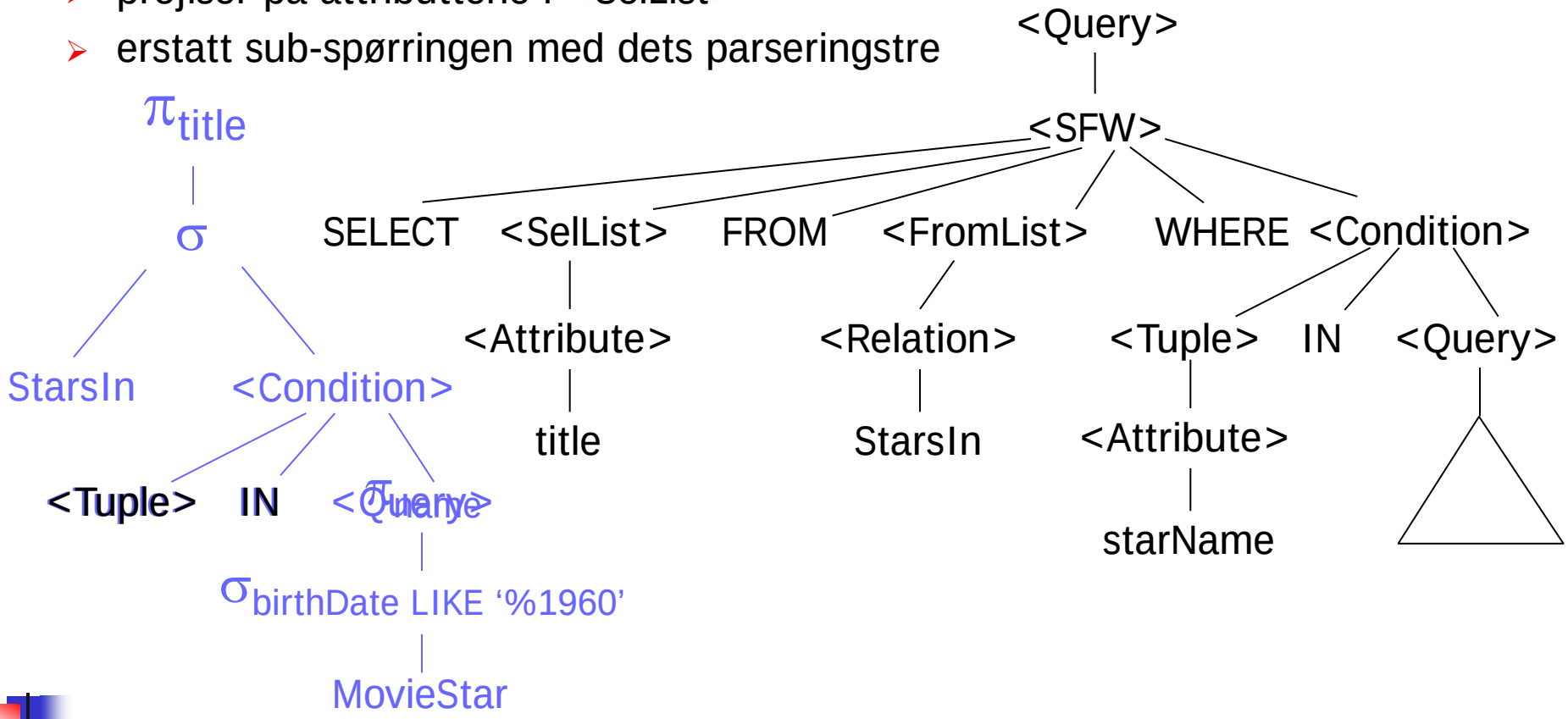
- For sub-spørringer bruker vi en foreløpig operator – to-arguments seleksjon σ.
- Videre behandling avhenger av typen <Condition>. Vi skal se på **`t IN S`** som et eksempel:
  - erstatt <Condition> med treet for S. Hvis S kan inneholde duplikater må vi legge til en δ–operator på toppen.
  - ertstatt to-arguments seleksjon med ett-arguments seleksjon σ_C, hvor C sammenligner hver komponent i t med det tilsvarende attributtet i S.
  - la σ_C ha produktet av R og S som argument.



Konvertering av sub-spørringer - eksempel

```
SELECT title FROM StarsIn WHERE starName IN
(SELECT name FROM MovieStar WHERE birtDate LIKE '%1960')
```

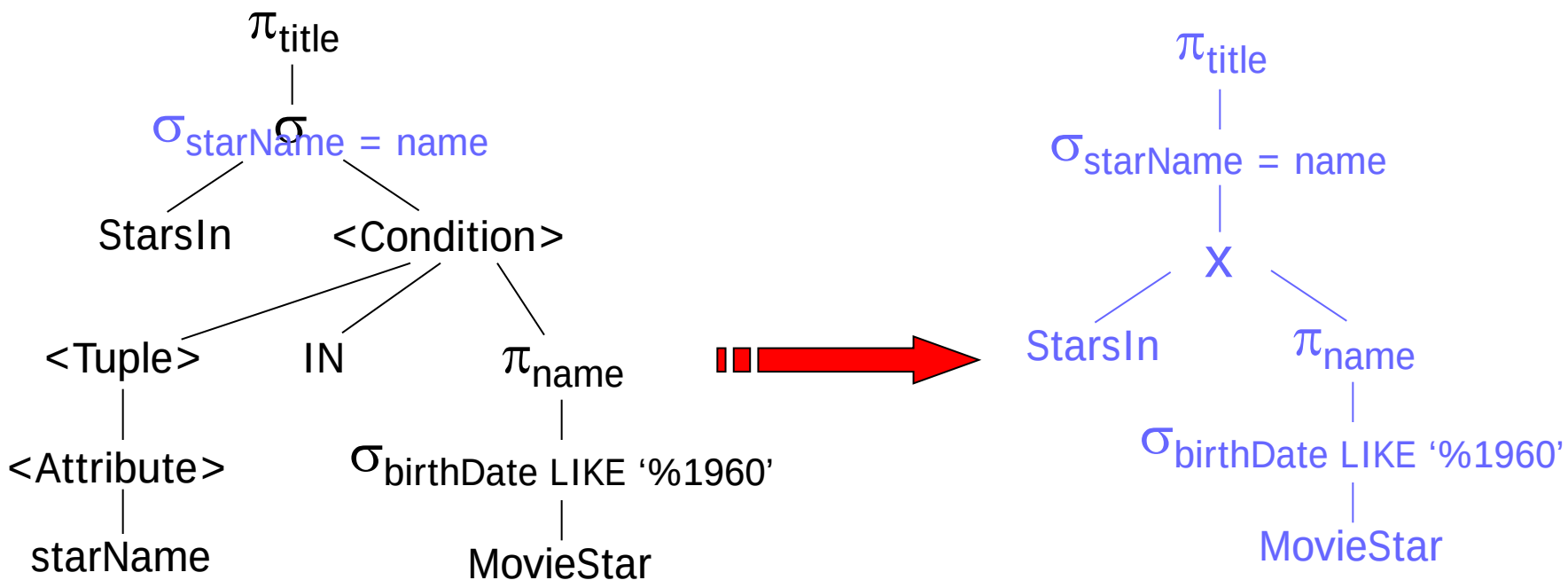
- produktet av relasjonene i <FromList>
- gjør seleksjon basert på <Condition>, representert ved to-arguments seleksjon
- projiser på attributtene i <SelList>
- erstatt sub-spørringen med dets parseringstre



Eksempel (forts)

```
SELECT title FROM StarsIn WHERE starName IN (...)
```

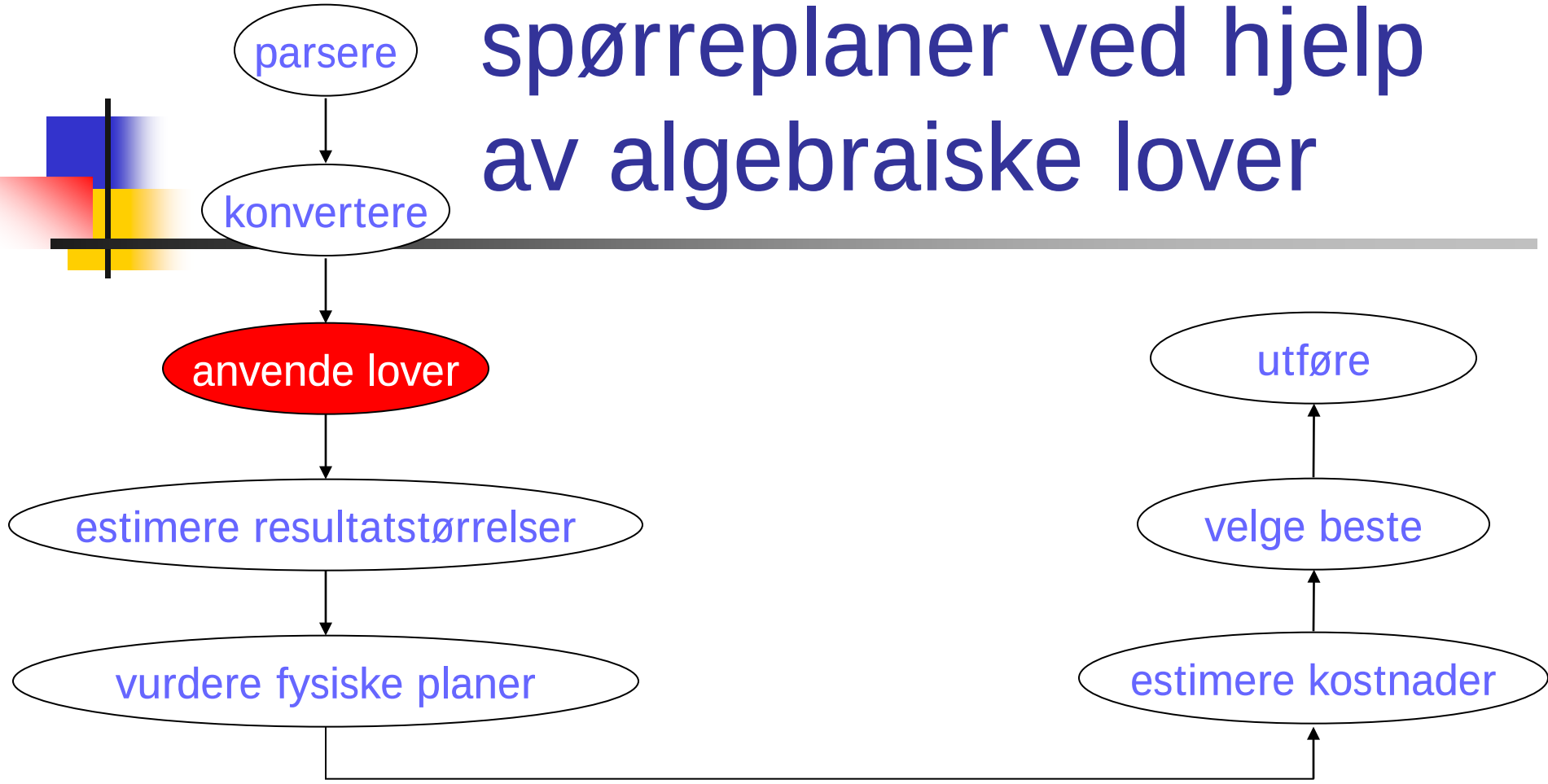
- erstatt <Condition> med treet for sub-spørringen
- erstatt to-arguments seleksjon med ett-arguments seleksjon σ_C, hvor C er `starName = name`
- la σ_C ha produktet av `StarsIn` og `MovieStar` som argument



Mer om konvertering

- Konvertering av sub-spørringer blir mer komplisert hvis sub-spørringen er relatert til verdier definert utenfor skopet dens
  - vi må da lage en relasjon med ekstra-attributter for sammenligning med de eksterne attributtene
  - de ekstra attributtene fjernes senere ved hjelp av projeksjoner
  - i tillegg må alle duplikate tupler fjernes

Forbedring av logiske spørreplaner ved hjelp av algebraiske lover



Kommutativitet og assosiativitet

- En operator ω er kommutativ hvis rekkefølgen på argumentene ikke har noen betydning:

  $$x \; \omega \; y = y \; \omega \; x$$

- En operator er assosiativ hvis grupperingen av argumentene ikke har noen betydning:

  $$x \; \omega \; (y \; \omega \; z) = (x \; \omega \; y) \; \omega \; z$$

Algebraiske lover – Produkt og join

- Natulig join og produkt er kommutative og assosiative:

  $R \bowtie S = S \bowtie R; \quad R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$

  $R \times S = S \times R; \quad R \times (S \times T) = (R \times S) \times T$

- Hva med theta-join?
  - Kommutativ:

    $R \bowtie_c S = S \bowtie_c R$
  - Ikke alltid assosiativ, feks:

    R(a,b) S(b,c) T(c,d)

    $(R \bowtie_{R.b < S.b} S) \bowtie_{a < d} T \neq R \bowtie_{R.b < S.b} (S \bowtie_{a < d} T)$

Rekkefølgen på produkt og join

- Har rekkefølgen og grupperingen av en produkt eller join noe å si i forhold til effektivitet?
  - hvis bare en av relasjonene får plass i minnet bør denne være første argument – ett-pass operasjon som reduserer antall disk I/O.
  - hvis produkt eller join av to av relasjonene gir en temporær relasjon som får plass i minnet bør disse tas først for å spare både minneplass og disk I/O.
  - temporære relasjoner bør være så små som mulig for å spare minneplass.
  - hvis vi kan estimere (vha statistikk) antall tupler som skal joines, kan vi spare mange operasjoner ved å joine de to relasjonene som gir færrest tupler først.

Algebraiske lover – union og snitt

- Union og snitt er kommutative og assosiative:

$$R \cup S = S \cup R; \qquad R \cup (S \cup T) = (R \cup S) \cup T$$
$$R \cap S = S \cap R; \qquad R \cap (S \cap T) = (R \cap S) \cap T$$

- Merk: Andre lover kan være ulike for **sett** og **bag,** feks distribusjon av snitt over union:

$$R \cap_S (S \cup_S T) = (R \cap_S S) \cup_S (R \cap_S T)$$
$$R \cap_B (S \cup_B T) \neq (R \cap_B S) \cup_B (R \cap_B T)$$

Algebraiske lover – seleksjon

- ✓ Seleksjon er en veldig viktig operator for optimalisering
  - reduserer antall tupler (størrelsen på relasjonen)
  - generel regel: *dytt seleksjoner så langt ned i treet som mulig*

- ✓ Betingelser med AND og OR kan splittes:
  - $\sigma_{a \text{ AND } b}(R) = \sigma_a(\sigma_b(R))$
  - $\sigma_{a \text{ OR } b}(R) = (\sigma_a(R)) \cup_s (\sigma_b(R))$

    (fungerer bare når R er et sett, bag-versjonen vil inkludere et tuppel to ganger i det siste uttrykket hvis begge betingelsene er oppfylt)

- ✓ Rekkefølgen av påfølgende seleksjoner har ingen betydning:
  - $\sigma_a(\sigma_b(R)) = \sigma_b(\sigma_a(R))$

Dytting av seleksjon

- Hvis seleksjon dyttes nedover i treet...
  - ... må den dyttes til begge argumentene for
    - union: $\sigma_a(R \cup S) = \sigma_a(R) \cup \sigma_a(S)$
    - kartesisk produkt: $\sigma_a(R \times S) = \sigma_a(R) \times \sigma_a(S)$
  - ... må den dyttes til første argument, valgfritt til andre argument for
    - differanse: $\sigma_a(R - S) = \sigma_a(R) - S = \sigma_a(R) - \sigma_a(S)$
  - ... kan den dyttes til en eller begge argumentene for
    - snitt: $\sigma_a(R \cap S) = \sigma_a(R) \cap \sigma_a(S) = R \cap \sigma_a(S) = \sigma_a(R) \cap S$
    - join: $\sigma_a(R \bowtie S) = \sigma_a(R) \bowtie \sigma_a(S) = R \bowtie \sigma_a(S) = \sigma_a(R) \bowtie S$
    - theta-join: $\sigma_a(R \bowtie_b S) = \sigma_a(R) \bowtie_b \sigma_a(S) = R \bowtie_b \sigma_a(S) = \sigma_a(R) \bowtie_b S$

Dytting av seleksjon – eksempel

- Eksempel: hvert attributt er 1 byte
  - $\sigma_{A=2}(R \bowtie S)$
    - join:
      kombiner 4 * 4 elementer = 16 operasjoner
      lagre relasjon $R \bowtie S$ = 52 bytes
    - seleksjon:
      sjekk hvert enkelt tuppel: 2 operasjoner
  - $\sigma_{A=2}(R) \bowtie S$
    - seleksjon:
      sjekk hvert enkelt tuppel: 4 operasjoner
      lagre releasjon $\sigma_{A=2}(R)$ = 24 bytes
    - join:
      kombiner 1 * 4 elementer = 4 operasjoner
  - $R \bowtie \sigma_{A=2}(S)$
    gir ikke mening siden a ikke er et attributt i S

Relasjon R

A	B	C	...	X
1	z	1	...	4
2	c	6	...	2
3	r	8	...	7
4	n	9	...	4

Relasjon S

X	Y	Z
2	f	c
3	t	b
7	g	c
9	e	c

Relasjon $R \bowtie S$

A	B	C	...	X	Y	Z
2	c	6	...	2	f	c
3	r	8	...	7	g	c

Relasjon $\sigma_{A=2}(R)$

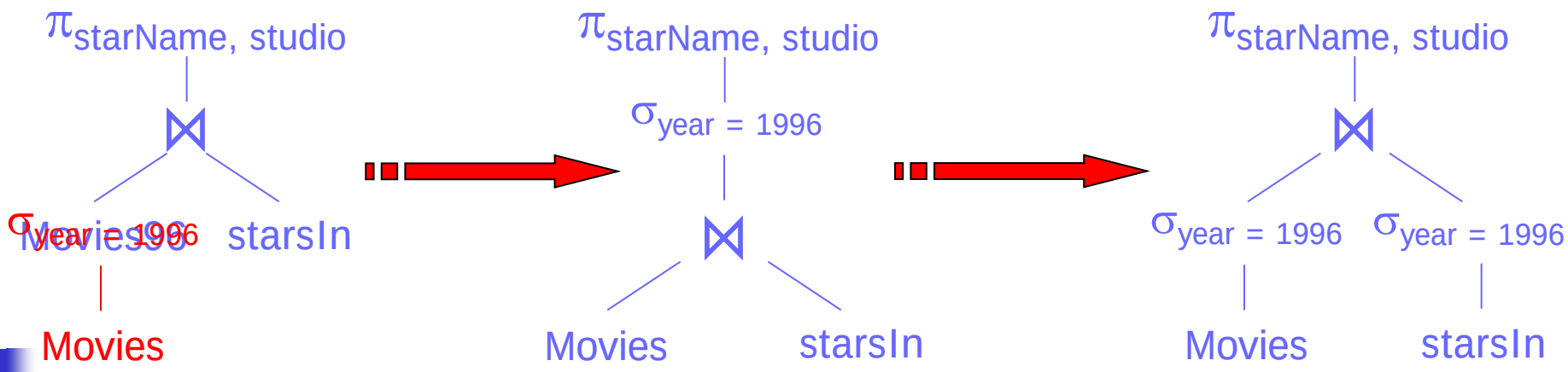
A	B	C	...	X
2	c	6	...	2

Dytting av seleksjon oppover i treet

- Noen ganger er det nyttig å dytte seleksjon den andre veien, dvs oppover i treet, ved å bruke loven $\sigma_a(R \bowtie S) = R \bowtie \sigma_a(S)$ "bakvendt".
- Eksempel:

```
StarsIn(title, year, starName);   Movies(title, year, studio …)
```

  - ```
    CREATE VIEW Movies96 AS
                SELECT *
                FROM Movies
                WHERE year = 1996;
    ```
  - ```
    SELECT starName, studio FROM Movies96 NATURAL JOIN StarsIn;
    ```



Algebraiske lover – projeksjon

- Projeksjoner kan dyttes nedover gjennom mange operatorer eller nye introduseres hvor som helst så lenge vi ikke fjerner attributter som brukes lenger oppe i treet.
  - $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))$, hvis
    - M = join attributt eller del av L i R
    - N = join attributt eller del av L i S
  - $\pi_L(R \bowtie_C S) = \pi_L(\pi_M(R) \bowtie_C \pi_N(S))$, hvis
    - M = join attributt (del av C) eller del av L i R
    - N = join attributt (del av C) eller del av L i S
  - $\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))$, hvis
    - M = del av L i R
    - N = del av L i S
  - $\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$ hvis
    - M = er alle attributtene i L eller del av C
  - $\pi_L(R \cup_B S) = \pi_L(R) \cup_B \pi_L(S)$

Merk: Projeksjoner kan *ikke* dyttes gjennom sett-union, snitt eller differanse

Algebraiske lover – join, produkt, seleksjon, projeksjon

- To viktige lover som følger fra definisjonen av join:
  - $\sigma_C(R \times S) = R \bowtie_C S$
  - $\pi_L(\sigma_C(R \times S)) = R \bowtie S$, hvis
    - C sammenligner hvert par av tupler fra R og S med samme navn
    - L = alle attributtene fra R og S (uten duplikater)

Eksempel

- ✓ $\pi_L(\sigma_{R.a = S.a}(R \times S))$ vs. $R \bowtie S$
  - R(a,b,c,d,e,..., k), T(R) = 10.000, S(a,l,m,n,o,...,z) , T(S) = 100
  - hvert attributt er 1 byte, a er nøkkel i både R og S
  - antar at alle tupler i S finner en match i R, dvs 100 tupler i resultatet
  - $\pi_L(\sigma_C(R \times S))$:
    - produkt:
      kombiner 10.000 * 100 elementer = 1.000.000 operasjoner
      lagre relasjon R **x** S = 1.000.000 * (11 + 16) = 27.000.000 bytes
    - seleksjon:
      sjekk hvert enkelt tuppel: 1.000.000 operasjoner
      lagre relasjon $\sigma_{R.a = S.a}(R \times S)$ = 100 * 27 = 2700 bytes
    - projeksjon:
      sjekk hvert enkelt tuppel: 100 operasjoner
  - $R \bowtie S$:
    - join:
      sjekk 10.000 * 100 elementer = 1.000.000 operasjoner

Algebraiske lover - duplikateliminasjon

- ✓ Duplikateliminasjon kan redusere størrelsen på temporæare relasjoner ved å dyttes gjennom
  - kartesisk produkt: $\delta(R \times S) = \delta(R) \times \delta(S)$
  - join: $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
  - theta-join: $\delta(R \bowtie_C S) = \delta(R) \bowtie_C \delta(S)$
  - seleksjon: $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$
  - **bag**-snitt: $\delta(R \cap_B S) = \delta(R) \cap_B \delta(S) = \delta(R) \cap_B S = R \cap_B \delta(S)$

- ✓ Men, duplikateliminasjon kan ikke dyttes gjennom
  - **sett**-operasjoner (gir ikke mening)
  - **bag**-union og differanse
  - projeksjoner

Forbedring av logiske spørreplaner

- De vanligste optimaliseringene er:
  - dytt seleksjoner så langt ned som mulig.
    Hvis betingelsen består av flere deler, splitt i flere seleksjoner og dytt hver enkelt så langt ned i treet som mulig.
  - dytt projeksjoner så langt ned som mulig.
    Projeksjoner kan legges til hvor som helst så lenge attributter over i treet er inkludert.
  - duplikateliminasjoner kan noen ganger fjernes (feks på nøkkel)
  - hvis mulig, kombiner seleksjon og kartesisk produkt til en join

Estimere resultatstørrelser

Estimere størrelser

- ✓ Ideelt ønsker vi regler som er:
  - nøyaktige – en liten feil kan resultere i valg av lite hensiktsmessig algoritme i den fysiske spørreplanen
  - enkle å beregne – minimal ekstrakostnad for å gjøre selve valget
  - logisk konsistent – ikke avhengig av konkret algoritme for operatoren
- ⇨ MEN, ingen universell algoritme finnes for dette
- ✓ Heldigvis hjelper også omtrentlige estimater til å velge en god fysisk spørreplan
- ✓ Husk:
  - B(R) angir antall blokker som er nødvendig for R
  - T(R) angir antall tupler i R
  - V(R, a) angir antall ulike verdier for attributt a i R (gjennomsnittlig antall tupler med like a-verdier er da T(R)/V(R,a))
- ✓ I tillegg lar vi S(R) angi størrelsen av et tuppel i R

Størrelsen av en projeksjon

- Størrelsen av en projeksjon (π) kan beregnes nøyaktig.
  - ett resultattuppel for hvert argumenttuppel
  - endrer bare størrelsen av hvert tuppel
  - $sizeof[\pi_{A, B, ...}(R)] = T(R) * [sizeof(R.A) + sizeof(R.B) + ...]$
- Eksempel:

Relasjon R

A	B	C	D
1	cat	1999	a
2	cat	2002	b
3	dog	2002	c
4	cat	1998	a
5	dog	2000	c

A: 4 byte integer
B: 20 byte text string
C: 4 byte date (year)
D: 30 byte text string

T(R) = 5

S(R) = 58

V(R,A) = 5
V(R,B) = 2
V(R,C) = 4
V(R,D) = 3

sizeof(R)

$sizeof[\pi_{A}(R)]$

$sizeof[\pi_{A, B, C, D, (A+10)\rightarrow E}(R)]$

Størrelsen av en seleksjon

- En seleksjon (σ) reduserer antall tupler, men størrelsen av hvert tuppel er uendret
  - sizeof[$\sigma_X(R)$] = T($\sigma_X(R)$) * S(R), hvor X er seleksjonsbetingelsen
  - estimering av antall tupler avhenger av
    - fordelinger av verdier for de aktuelle attributtene – vi antar en uniform fordeling hvor vi bruker V(R,A) for å estimere antall tupler i resultatet
    - seleksjonsbetingelsen

Størrelsen av en seleksjon (forts)

- Likhet, $\sigma_{A = c}(R)$, for attributt A og konstant c:
  - $T(\sigma_{A=c}(R)) = T(R) / V(R, A)$

    $T(\sigma_{A = 3}(R))$

    $T(\sigma_{B = \text{'cat'}}(R))$
- Forskjellig, $\sigma_{A < c}(R)$:
  - $T(\sigma_{A<c}(R)) = T(R) / 3$

    $T(\sigma_{A > 2}(R))$
- Ulikhet, $\sigma_{A \neq c}(R)$:
  - $T(\sigma_{A \neq c}(R)) = T(R) * [(V(R,A) - 1) / V(R,A)]$

    $T(\sigma_{B \neq \text{'cat'}}(R))$
- Eksempel:

A	B	C	D
1	cat	1999	a
2	cat	2002	b
3	dog	2002	c
4	cat	1998	a
5	dog	2000	c

A: 4 byte integer
B: 20 byte text string
C: 4 byte date (year)
D: 30 byte text string

T(R) = 5

S(R) = 58

V(R,A) = 5
V(R,B) = 2
V(R,C) = 4
V(R,D) = 3

Størrelsen av en seleksjon med AND eller NOT

- Seleksjon med flere betingelser med **AND**, $\sigma_{A \textbf{ AND } B \textbf{ AND} \ldots}(R)$
  - estimer størrelsen ved hjelp av en selektivitetsfaktor for hver betingelse, 1/3 for $< > \leq \geq$, 1 for $\neq$, 1 / V(R,A) for = på attributt A
  - $T(\sigma_{A \textbf{ AND } B \textbf{ AND} \ldots}(R)) = T(R) * faktor_A * faktor_B * \ldots$

    $T(\sigma_{C = 1999 \textbf{ AND } A < 4}(R))$
- Seleksjon med **NOT**, $\sigma_{\textbf{NOT } A}(R)$
  - $T(\sigma_{\textbf{NOT } A}(R)) = T(R) - T(\sigma_A(R))$

    $T(\sigma_{\text{NOT } A = 3}(R))$
- Eksempel:

A	B	C	D
1	cat	1999	a
2	cat	2002	b
3	dog	2002	c
4	cat	1998	a
5	dog	2000	c

A: 4 byte integer
B: 20 byte text string
C: 4 byte date (year)
D: 30 byte text string

T(R) = 5

S(R) = 58

V(R,A) = 5
V(R,B) = 2
V(R,C) = 4
V(R,D) = 3

Størrelsen av en seleksjon med OR

- ✓ Seleksjon med flere betingelser med **OR**, $\sigma_{A \textbf{ OR } B \textbf{ OR} \ldots}(R)$
  - Alternativ 1: $T(\sigma_{A \textbf{ OR } B \textbf{ OR} \ldots}(R)) = T(\sigma_A(R)) + T(\sigma_B(R)) + \ldots$
  - Alternativ 2: $T(\sigma_{A \textbf{ OR } B \textbf{ OR} \ldots}(R)) = \min(\ T(R),\ \ (T(\sigma_A(R)) + T(\sigma_B(R)) + \ldots)\ )$
  - Alternativ 3:
    - anta at m_1 tupler tilfredsstiller den første betingelsen, m_2 tilfredsstiller den andre betingelsen, ...
    - $1 - m_x/T(R)$ er da andelen tupler som ikke tilfredsstiller den x'te betingelsen
    - $T(\sigma_{A \textbf{ OR } B \textbf{ OR} \ldots}(R)) = T(R) * \left[1 - (1 - m_1/T(R)) * (1 - m_2/T(R))\right]$

Størrelsen av et produkt

- Størrelsen av et kartesisk produkt (x) kan beregnes nøyaktig:
  - får ett tuppel for hver mulige kombinasjon av tuplene i relasjonene R og S:
    T(RxS) = T(R) * T(S)
  - størrelsen av hvert nye tuppel er summen av størrelsen til hvert av de opprinnelige tuplene:
    S(RxS) = S(R) + S(S)
  - sizeof(R x S) = T(RxS) * S(RxS) = T(R) * T(S) * (S(R) + S(S))

Størrelsen av (naturlig) join

- Utfordring: Vet ikke hvordan verdiene til join-attributtet Y fordeler seg mellom relasjonene.
  - disjunkte sett med y-verdier – tomt resultat:
    $T(R \bowtie S) = 0$
  - y er en fremmednøkkel til S i R – hvert tuppel i R matcher ett tuppel i S:
    $T(R \bowtie S) = T(R)$
  - nesten alle tuplene i R og S har samme y-verdi A – kombiner alle tuplene i hver relasjon:
    $T(R \bowtie S) = T(R) * T(S)$
- Antagelser:
  - inkludering av verdi-sett:
    hvis $V(R, y) \leq V(S, y)$, vil hver y-verdi i R ha en match i S
  - bevaring av verdi-sett:
    verdi-settet til ikke-join attributter er det samme før og etter join, dvs
    $V(R \bowtie S, x) = V(R, x)$

Størrelsen av (naturlig) join (forts)

- Antall tupler i $R(x, y) \bowtie S(y, z)$ kan nå estimeres som følger:
  - hvis $V(R, y) \leq V(S, y)$, vil hvert tuppel i R matche anslagsvis T(S)/V(S,y) tupler i S:
    $T(R \bowtie S) = T(R) * T(S) / V(S, y)$
  - tilsvarende, hvis $V(S, y) \leq V(R, y)$:
    $T(R \bowtie S) = T(S) * T(R) / V(R, y)$
  - generelt:
    $T(R \bowtie S) = T(S) * T(R) / \max[V(R, y), V(S, y)]$
- Eksempel: $T(A \bowtie B \bowtie C)$

A(a, b)	B(b, c)	C(c, d)
T(A) = 10.000	T(B) = 2.000	T(C) = 5.000
V(A, a) = 5.000	V(B, b) = 100	V(C, c) = 100
V(A, b) = 1.000	V(B, c) = 1.000	V(C, d) = 100

Størrelsen av (naturlig) join (forts)

- Hvis det er *mer enn ett join-attributt,* $R(x, y_1, y_2,...) \bowtie S(y_1, y_2,..., z)$, får vi:

$$T(R \bowtie S) = \frac{T(S) * T(R)}{\max[V(R, y_1), V(S, y_1)] * \max[V(R, y_2), V(S, y_2)] * ...}$$

for hvert y_x attributt som er felles for R og S

- For naturlig join mellom flere relasjoner $R_1 \bowtie R_2 \bowtie R_3 \bowtie ... \bowtie R_n$
  - start med maksimalt antall tupler $T(R_1) * T(R_2) * T(R_3) * ... * T(R_n)$
  - for hvert attributt A som forekommer i mer enn en relasjon, divider med alle unntatt den minste V(R,A)

Størrelsen av en union

- Avhenger av om vi bruker **sett**- eller **bag**-versjonen:
  - bag:
    resultatet er nøyaktig lik summen av tupler i argumentene:
    $T(R \cup_b S) = T(R) + T(S)$
  - sett:
    - som for bag hvis relasjonene er disjunkte
    - antall tupler i den største relasjonen hvis den minste er et subsett av denne
    - vanligvis et sted mellom disse, kan for eksempel bruke:
      $T(R \cup_s S) = T(R) + T(S)/2$
      hvor S er den minste relasjonen

Størrelsen av snitt og differanse

- Antall tupler i et snitt ($\cap$) kan være
  - 0 hvis relasjonene er disjunkte
  - min(T(R), T(S)) hvis den ene relasjonen er et subsett av den andre
  - vanligvis et sted i mellom, kan for eksempel bruke gjennomsnittet:
    min(T(R), T(S)) / 2
- Antall tupler i en differanse (–), R – S, er
  - T(R) hvis relasjonene er disjunkte
  - T(R) – T(S) hvis alle tuplene i S finnes i R
  - vanligvis et sted i mellom, kan for eksempel bruke:
    T(R) – T(S)/2

Størrelsen av en duplikateleminasjon

- Antall distinkte tupler som resultat av en duplikateliminasjon (δ) er
  - 1 hvis alle tuplene er like
  - T(R) hvis alle tuplene er forskjellige
  - en tilnærming:
    - gitt V(R, a_i) for alle *n* attributter, vil maksimalt antall ulike tupler være V(R, a_1) * V(R, a_2) * ... * V(R, a_n)
    - la estimert antall tupler være det minste av dette tallet og antall tupler i relasjonen
- Tilsvarende for gruppering

Vurdere fysiske planer

Sammenligning av logiske spørreplaner

- Vi sammenligner ulike spørreplaner for en gitt spørring ved hjelp av størrelsen på temporære relasjoner
  - estimer resultatet av hver operator i spørreplanen
  - legg til kostnaden i treet
  - kostnaden til planen er lik summen av alle kostnadene i treet, bortsett fra for:
    - roten – sluttresultatet
    - løvnodene – data lagret på disk

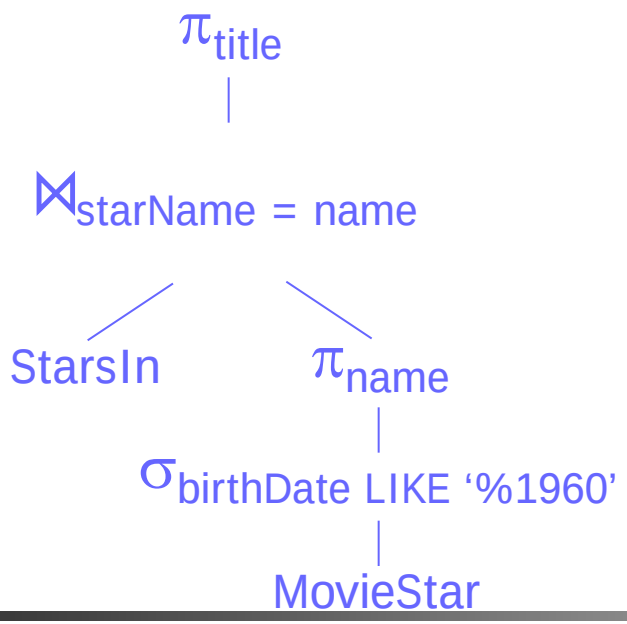
Sammenligning av logiske spørreplaner - eksempel

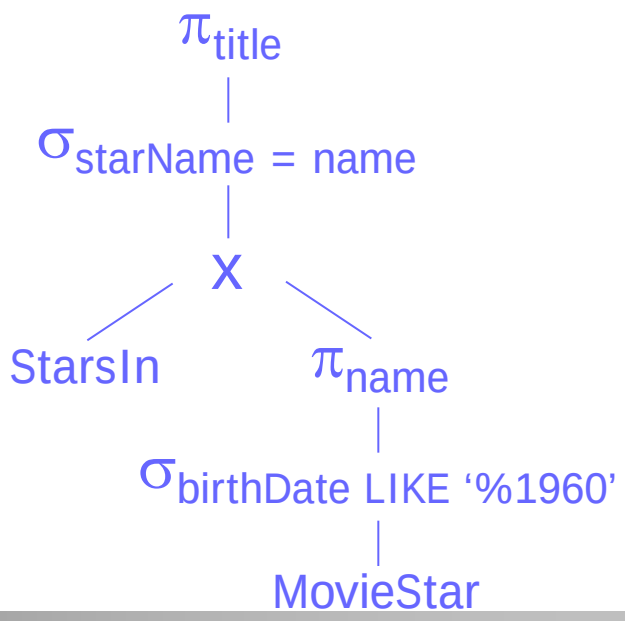
```
StarsIn(title, year, starName)
MovieStar(name, address, gender, birthDate)

SELECT title
FROM StarsIn
WHERE starName IN (
            SELECT name
            FROM MovieStar
            WHERE birthDate LIKE '%1960');
```

Statistikk:

T(StarsIn) = 10.000
V(StarsIn, starName) = 500
S(StarsIn) = 60

T(MovieStar) = 1.000
V(MovieStar, name) = 1.000
V(MovieStar, birthDate) = 50
S(MovieStar) = 100





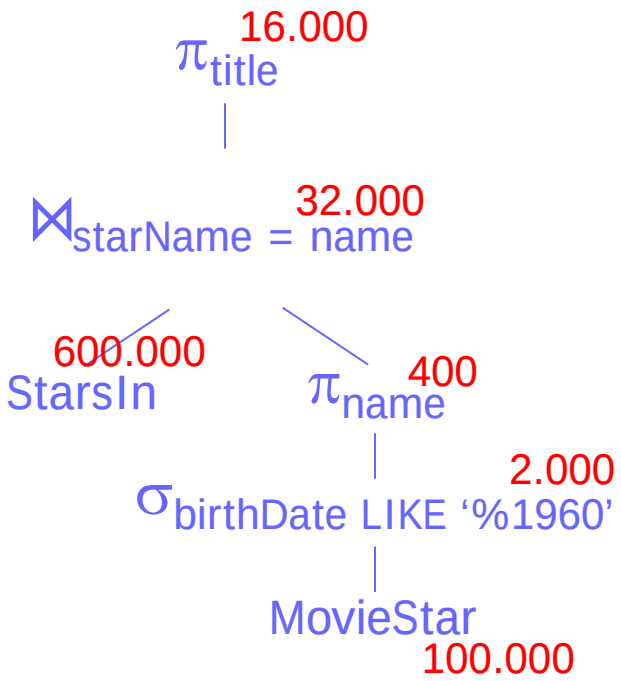
Eksempel (forts)

- $A_1 = \sigma_{\text{birthDate LIKE '\%1960'}}(MS)$:
  - $T(\sigma(MS)) = T(MS) / V(MS, \text{birthDate}) = 1000 / 50 = 20$
  - $\text{sizeof}(A_1) = 20 * 100 = 2000$
- $A_2 = \pi_{\text{name}}(A_1)$:
  - $T(\pi(A_1)) = T(A_1) = 20$
  - anta at attributt name tar 20 byte
  - $\text{sizeof}(A_2) = 20 * 20 = 400$
- $A_3 = SI \bowtie A_2$:
  - $T(SI \bowtie A_2) = T(SI)*T(A_2) / \max[V(SI, \text{starName}), V(A_2, \text{name})] = 10000 * 20 / \max(500, 20) = 400$
  - $S(A_2) = 20$
  - $\text{sizeof}(A_3) = 400 * (60 + 20) = 32000$
- $A_4 = \pi_{\text{title}}(A_3)$:
  - $T(\pi(A_3)) = T(A_3) = 400$
  - anta at title tar 40 byte
  - $\text{sizeof}(A_4) = 400 * 40 = 16000$

Statistikk:

T(SI) = 10.000
V(SI, starName) = 500
S(SI) = 60

T(MS) = 1.000
V(MS, name) = 1.000
V(MS, birthDate) = 50
S(MS) = 100



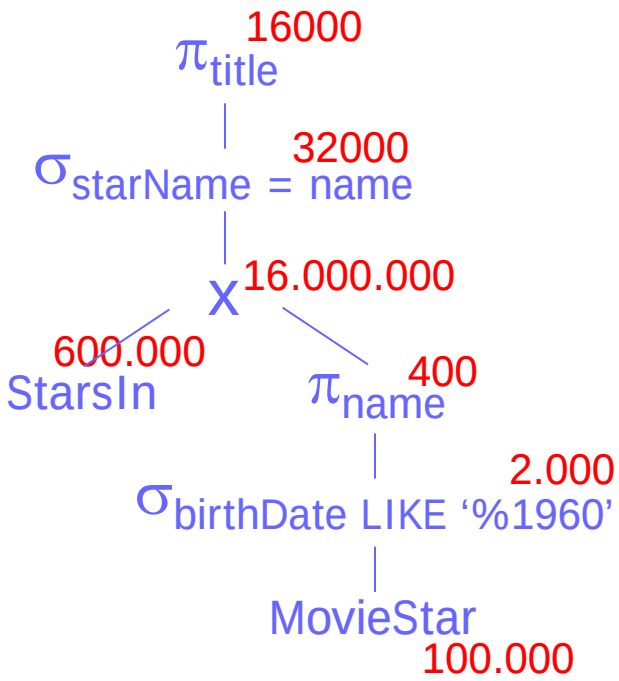
Eksempel (forts)

- $A_1 = \sigma_{\text{birthDate LIKE '\%1960'}}(MS)$ → som før: 2000, $T(\sigma(MS))=20$
- $A_2 = \pi_{\text{name}}(A_1)$ → som før: 400, $T(A_2) = T(A_1) = 20$
- $A_3 = SI \times A_2$:
  - $T(SI \times A_2) =$
    $T(SI) * T(A_2) = 10000 * 20 = 200.000$
  - $S(A_2) = 20$
  - $sizeof(A_3) = 200.000 * (60 + 20) = 16.000.000$
- $A_4 = \sigma_{\text{starName = name}}(A_3)$:
  - $T(\sigma(A_3)) =$
    $T(A_3) / \max(V(A_3, \text{name}), V(SI, \text{starName}))$
    $= 200.000 / \max(20, 500) = 400$
  - $S(A_4) = S(SI) + S(A_3) = 60 + 20 = 80$
  - $sizeof(A_4) = 400 * 80 = 32000$
- $A_5 = \pi_{\text{title}}(A_4)$ → som før: 400 * 40 = 16000

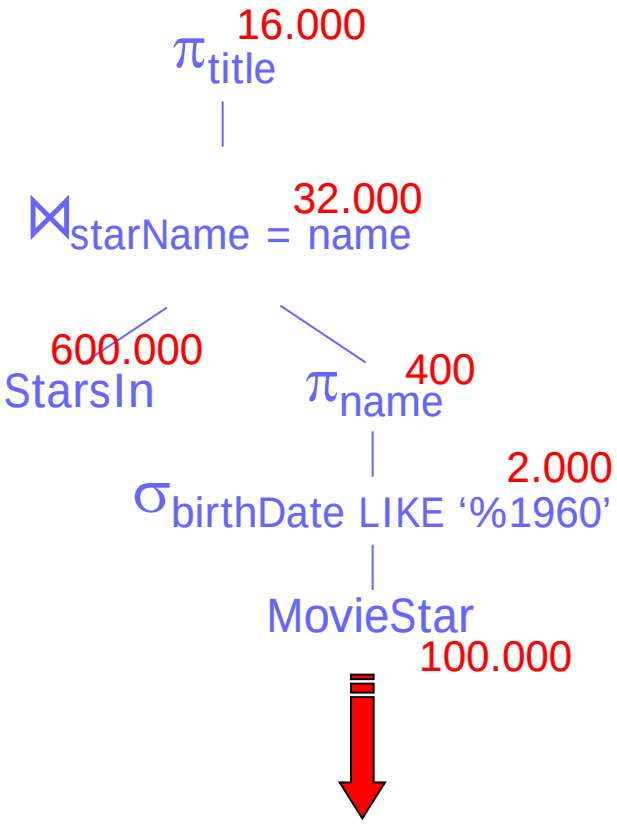
Statistikk:

T(SI) = 10.000
V(SI, starName) = 500
S(SI) = 60

T(MS) = 1.000
V(MS, name) = 1.000
V(MS, birthDate) = 50
S(MS) = 100

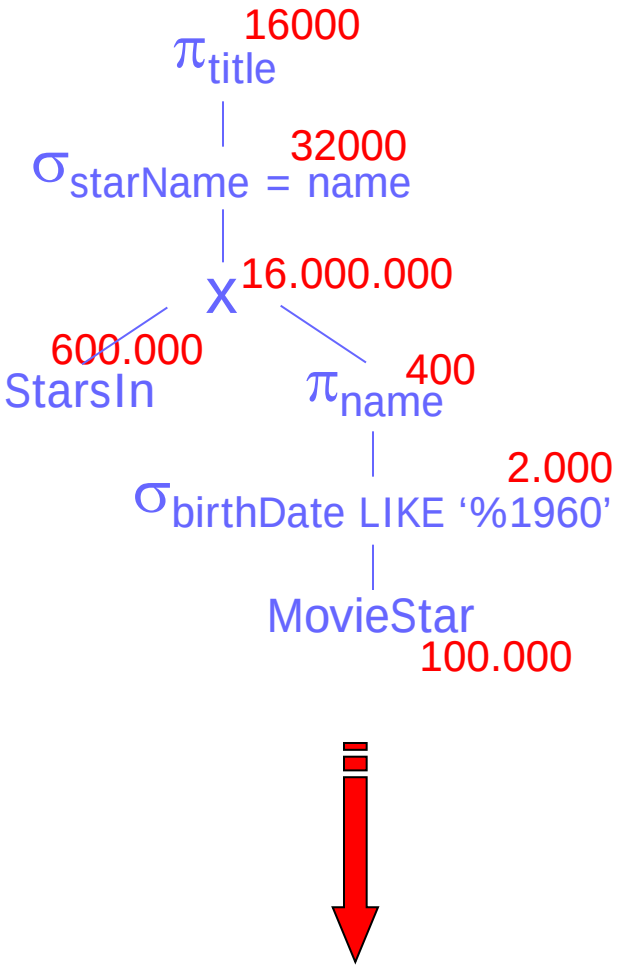


Eksempel (forts)



Totalt:

2000 + 400 + 32000 = 34400



Totalt:

2000 + 400 + 16000000 + 32000= 16034400

Valg av fysisk spørreplan

- Mulige alternativer for å velge “billigst” fysisk spørreplan:
  - exhaustive
  - heuristic
  - branch-and-bound
  - hill climbing
  - dynamic programming
  - Selinger-style optimizations

Valg av fysisk spørreplan (forts)

- Exhaustive:
  - se på alle mulige kombinasjoner av valg i planen
  - estimer kostnaden til hver plan
  - mange planer, kostbart

- Heuristic
  - velg en plan i henhold til heuristiske regler, dvs basert på tidligere erfaringer som feks:
    - bruk indeks på operasjoner som $\sigma_{A = 10}(R)$
    - bruk den minste relasjonen først ved join av mange relasjoner
    - hvis argumentene er sortert, bruk en sorterings-basert operator
    - ...
  - rask, men bare basert på generelle regler

Valg av fysisk spørreplan (forts)

- Branch-and-Bound:
  - finn en plan ved hjelp av heuristiske regler
  - se på mindre deler av planen for å finne optimaliseringer

- Sellinger-style optimization:
  - for alle sub-uttrykk, ta vare på kostnaden og forventet type resultat
  - en operator kan ha høyere individuell kostnad, men hvis resultatet feks er sortert, kan senere operatorer bruke dette
    - ingen effekt ved vurdering av størrelsen på mellomresultatet
    - ved vurdering av antall disk I/O kan første del av en sorterings-basert operasjon spares

Valg av seleksjons-algoritme

- Eksempel:
  - R(x, y, z), T(R) = 5000, B(R) = 200, V(R, x) = 100, V(R, y) = 500
  - indekser på alle attributtene, R er sortert på z
  - $\sigma_{x=1 \text{ AND } y=2 \text{ AND } z<5}(R)$
  - table-scan – les blokk for blokk:
    - kostnad: B(R) = 200 disk I/O siden R ligger samlet
  - index-scan på x – finn tupler med x=1 ved bruk av indeksen, sjekk så y og z:
    - i verste fall ligger alle tuplene på ulike blokker
    - kostnad: T(R) / V(R, x) = 5000 / 100 = 50 disk I/O
  - index-scan på y – finn tupler med y=2 ved bruk av indeksen, sjekk så x og z:
    - i verste fall ligger alle tuplene på ulike blokker
    - kostnad: T(R) / V(R, y) = 5000 / 500 = 10 disk I/O
  - index-scan på z – finn tupler med z<5 ved bruk av indeksen, sjekk så x og y:
    - vi har estimert slike seleksjoner til 1/3 av tuplene, R er sortert på z
    - kostnad: B(R) / 3 = 67 disk I/O

Valg av join-algoritme

- Hvis vi ikke vet hvor mye ressurser som er tilgjengelige:
  - velg ett-pass og håp at det er tilstrekkelig med minneplass
  - velg sort-join hvis...
    - ... begge argumentene allerede er sorterte
    - ... det joines tre eller flere relasjoner på samme attributt
  - velg index-join hvis den ene relasjonen er liten og det finnes en passende indeks på den andre
  - velg ellers hash-join siden det krever minst minne

Pipelining versus materialisering

- Det siste viktige spørsmålet er hvordan overføre mellomresultater mellom operatorer:
  - pipelining –
    send resultatet direkte videre til den nye operatoren, dvs data forblir i minnet og operasjoner kan arbeide samtidig
    - kan være mer effektivt
    - krever mer minne – risikerer flere disk-aksesser
  - materialisering –
    lagre alle mellomresultater på disk inntil de trengs av en annen operator
    - mer disk I/O
    - kan tillate enklere algoritmer siden en operator kan bruke mer minne

Pipelining versus materialisering (forts)

- ✓ *Unære operasjoner*, seleksjon og projeksjon, bør pipelines siden operasjonene utføres på ett tuppel om gangen.

- ✓ *Binære operasjoner* kan pipelines, men
  - ➢ antall buffere som trengs for beregningene varierer
  - ➢ resultatstørrelsene varierer
  - ⇨ valget om pipelining avhenger av minneplassen

Oppsummering

- ✓ Parsering
- ✓ Logiske spørreplaner uttrykt i relasjonsalgebra
- ✓ Optimalisering ved hjelp av algebraiske lover
- ✓ Estimere størrelsen på mellomresultater
- ✓ Vurdere fysiske spørreplaner