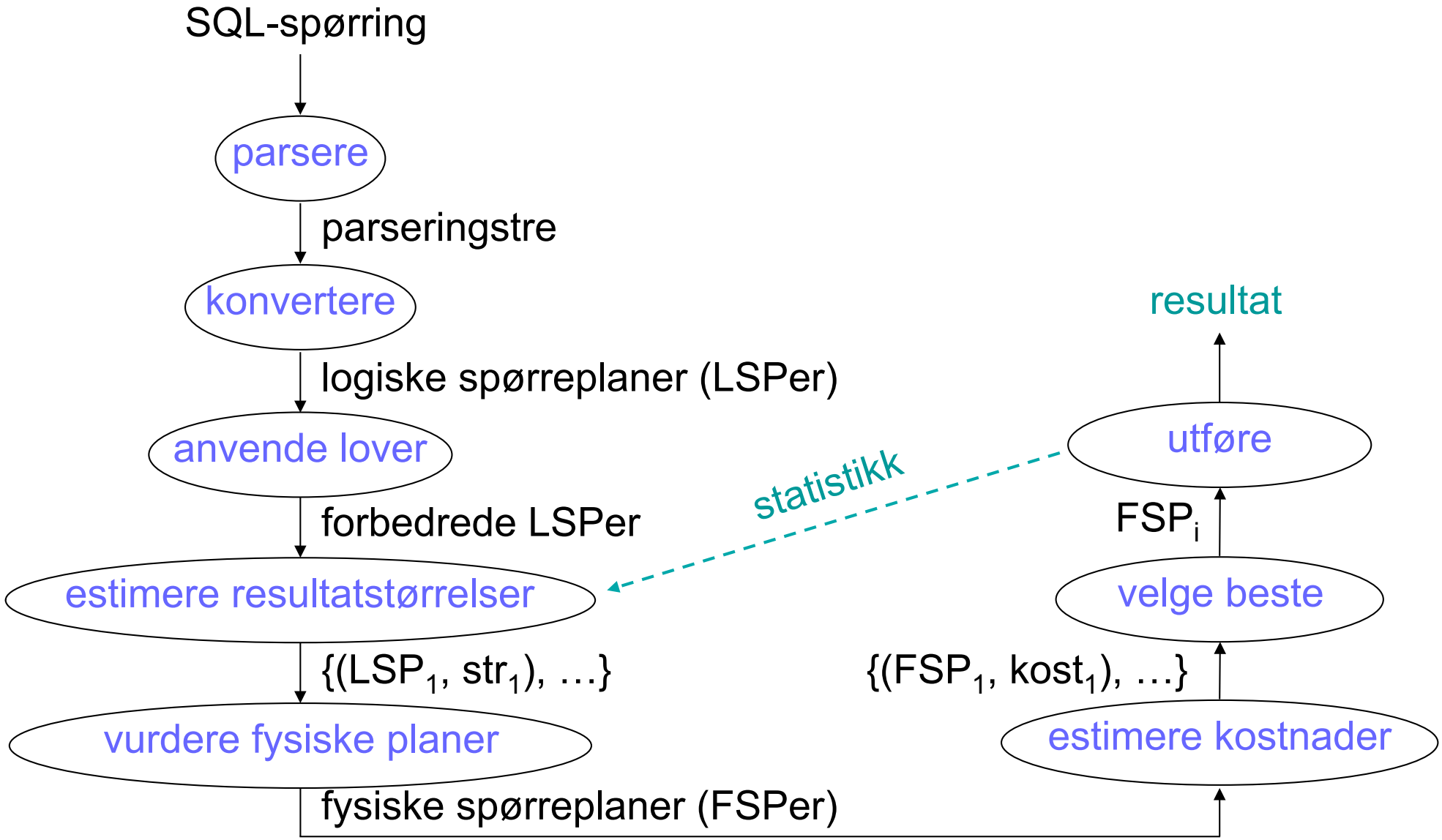
Spørsmålskompilering del 1

- Parsering
- Logiske spørreplaner uttrykt i relasjonsalgebra
- Optimalisering ved hjelp av algebraiske lover

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'
```

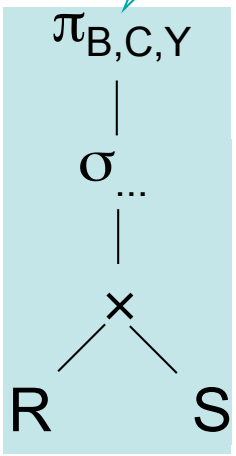
idé 1 – ta kartesisk produkt, velg tupler, projiser attributter

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

Merk:

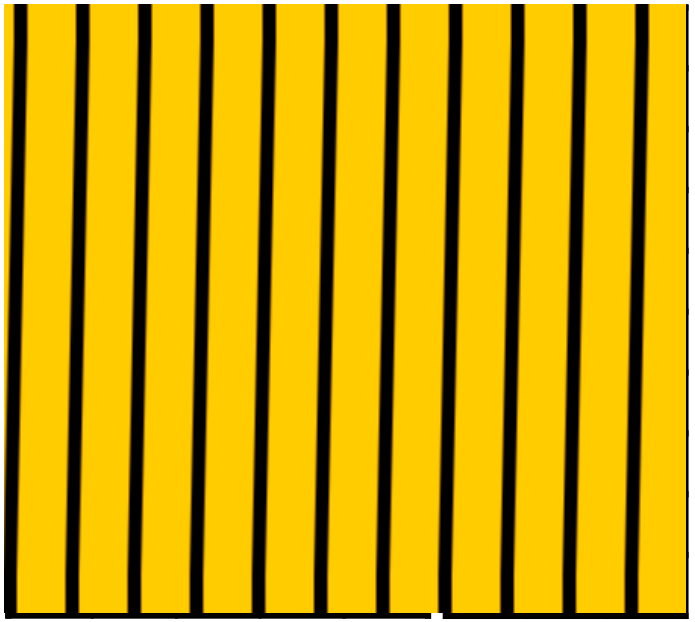
#attributter = #R-attributter + #S-attributter

#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

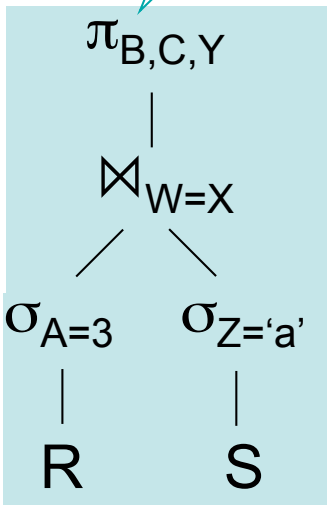
B	C	Y
r	8	k

Eksempel (forts.)

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

idé 2 – velg tupler, gjør equijoin, 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

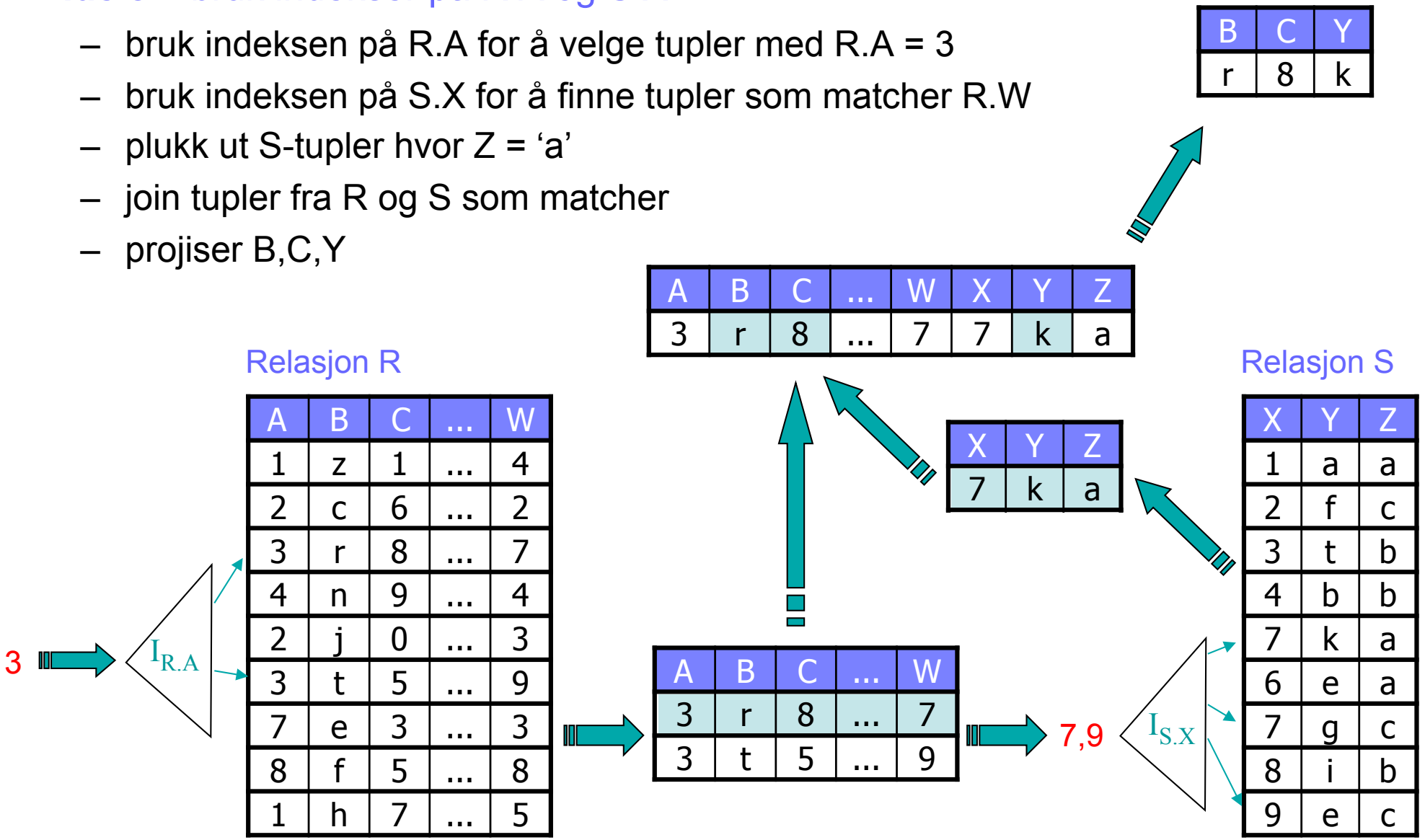
X	Y	Z
1	a	a
2	f	c
3	t	b
4	b	b
7	k	a
6	e	a
7	g	c
8	i	b
9	e	c

Eksempel (forts.)

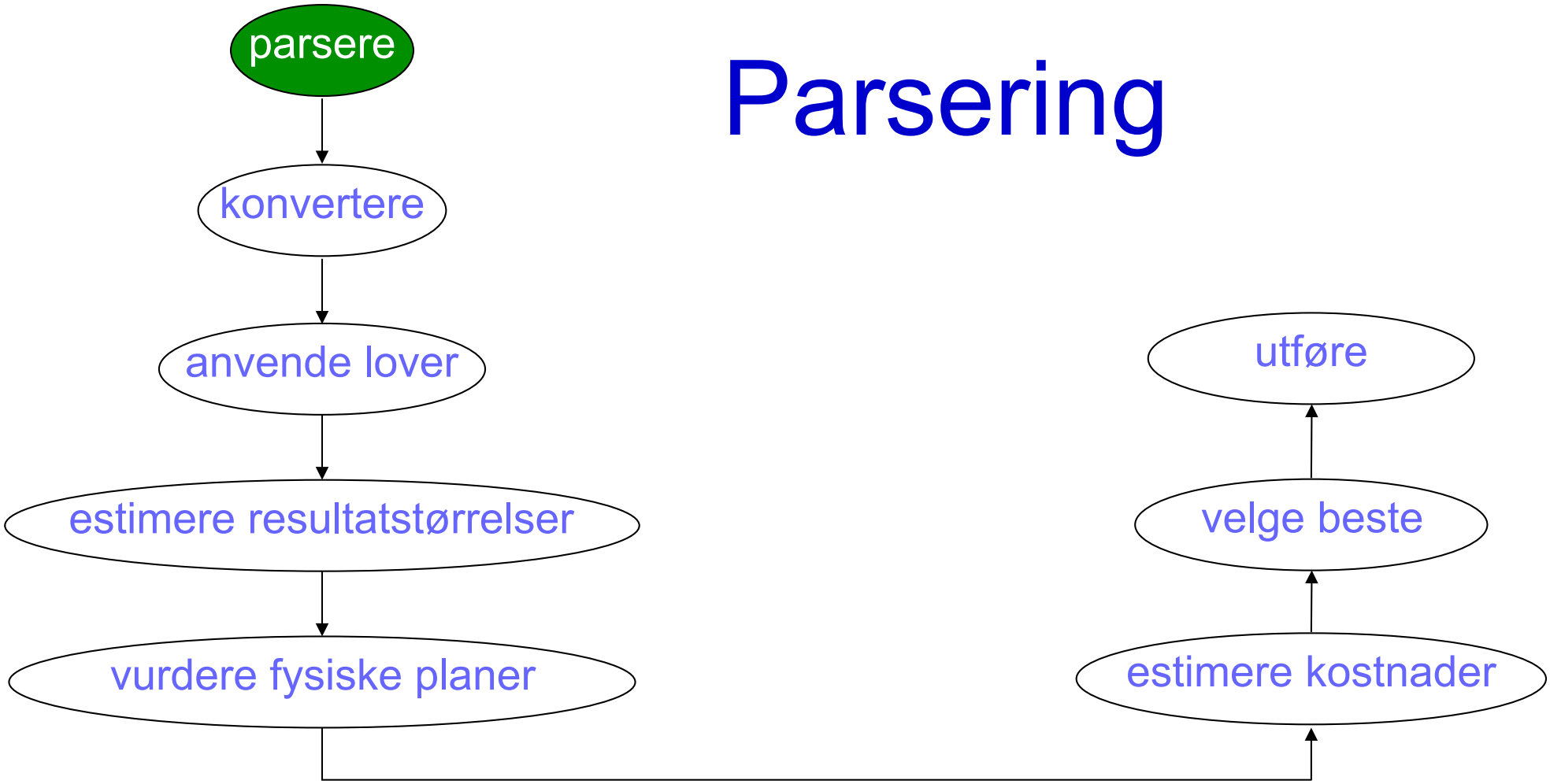
```
SELECT B,C,Y
FROM R,S
WHERE W=X AND A=3 AND Z='a'
```

idé 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
- plukk ut S-tupler hvor Z = 'a'
- join tupler fra R og S som matcher
- projiser B,C,Y

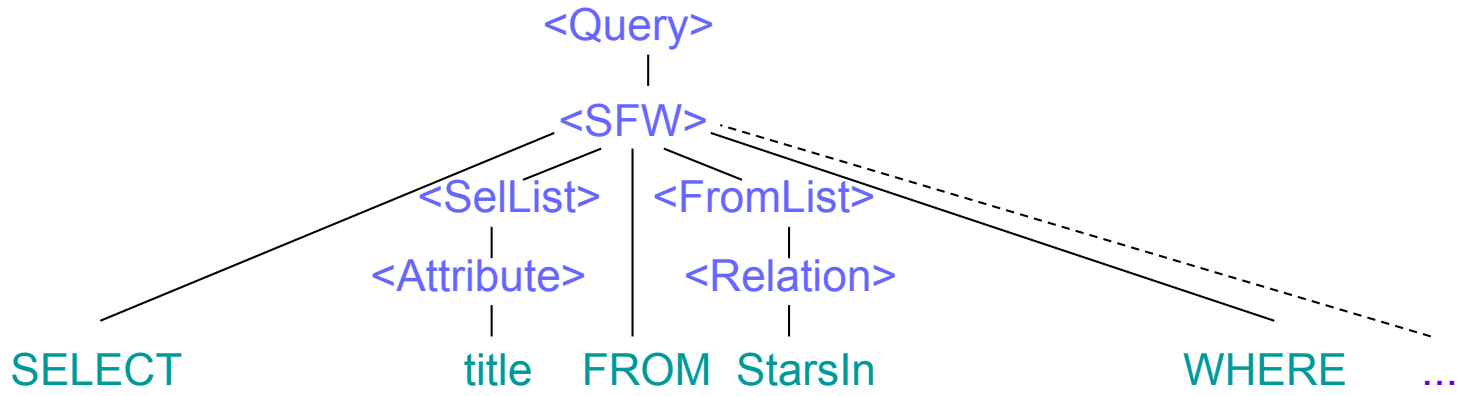


Parsering


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:
  - *et atom* – leksikalsk element som nøkkelord, navn, konstanter, parenteser og operatorer (løvnode)
  - *en syntaktisk* kategori – del av en spørring (indre node)

Enkel grammatikk

- Query:
  - <Query> ::= <SFW>
  - <Query> ::= **(** <Query> **)**
  - en komplett grammatikk vil også inneholde operatorer som UNION, JOIN, …
  - regel 2 brukes typisk i subspørringer
- 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

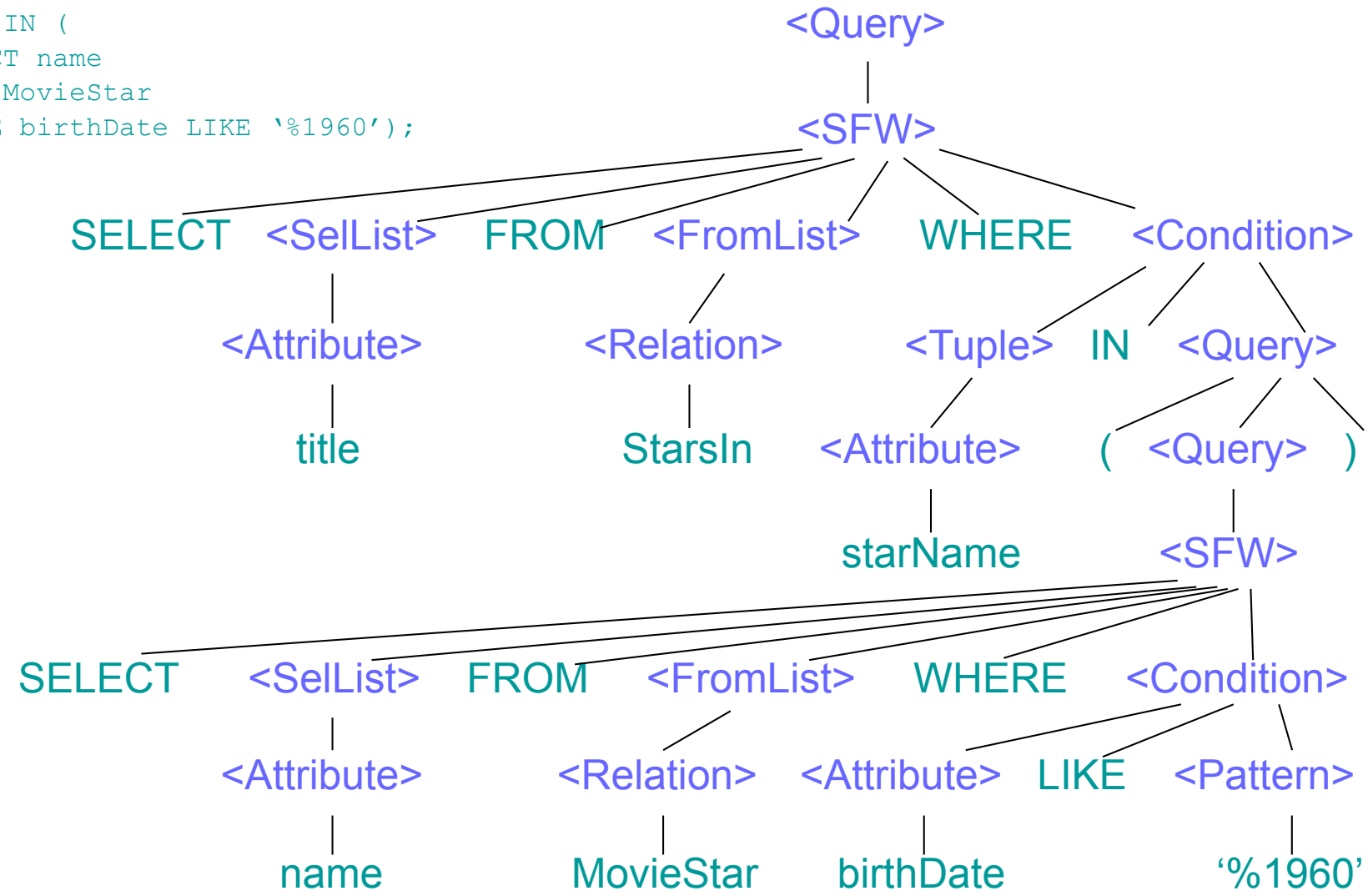
Finn filmer med skuespillere født i 1960:

```
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:

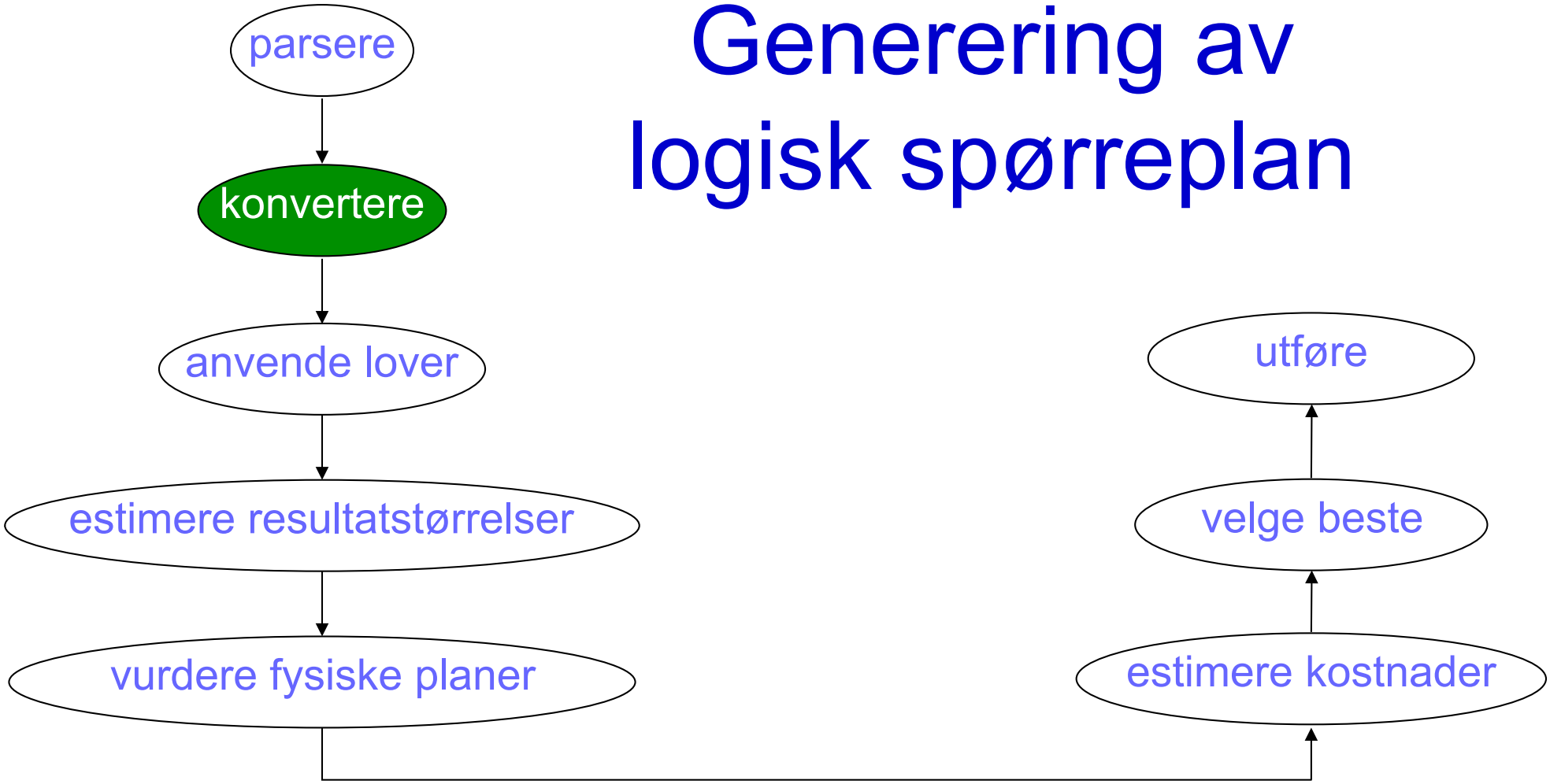
```
SELECT title
FROM StarsIn
WHERE starName IN (
          SELECT name
          FROM MovieStar
          WHERE birthDate LIKE '%1960');
```



Preprosessor

- Sjekker at spørringen er semantisk korrekt:
  - *relasjoner* – hver relasjon i FROM må være en relasjon eller et view i skjemaet hvor spørringen utføres.
    Hvert view må erstattes av et parseringstre.
  - *attributter* – hvert attributt må finnes i en av relasjonene i det aktuelle skopet i spørringen.
  - *typer* – all bruk av attributter må være i henhold til angitte typer.

Generering av logisk spørreplan



Konvertering av SFW

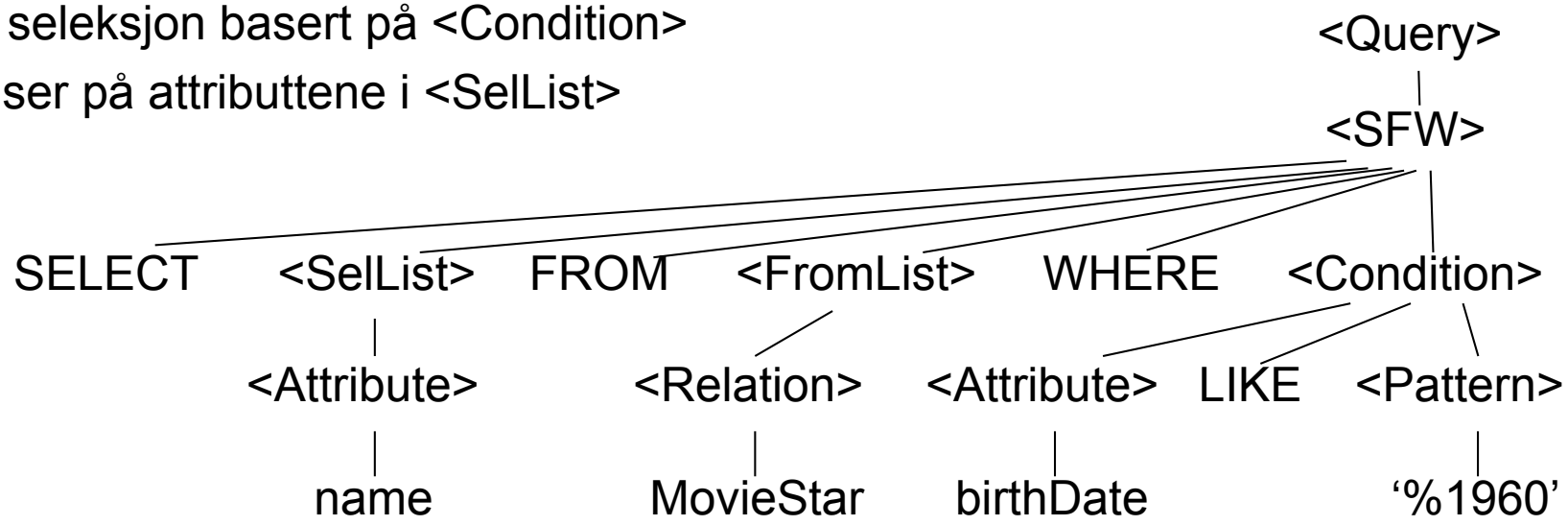
SELECT <SelList>
FROM <Fromlist>
WHERE <Condition>

- erstatt relasjonene i <FromList> med produktet $(\times)$ 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}

|

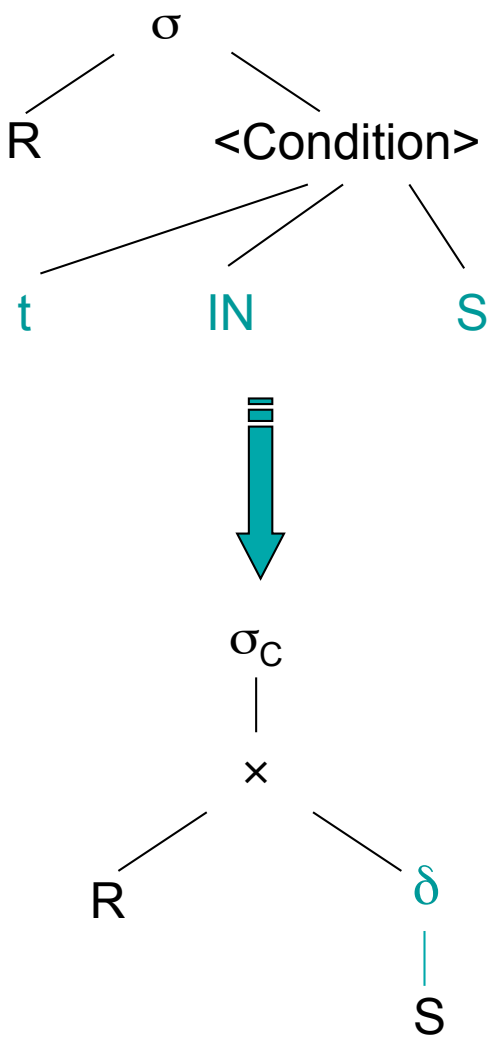
$\sigma_{birthDate\ LIKE\ '\%1960'}$

|

MovieStar

Konvertering av subspørringer

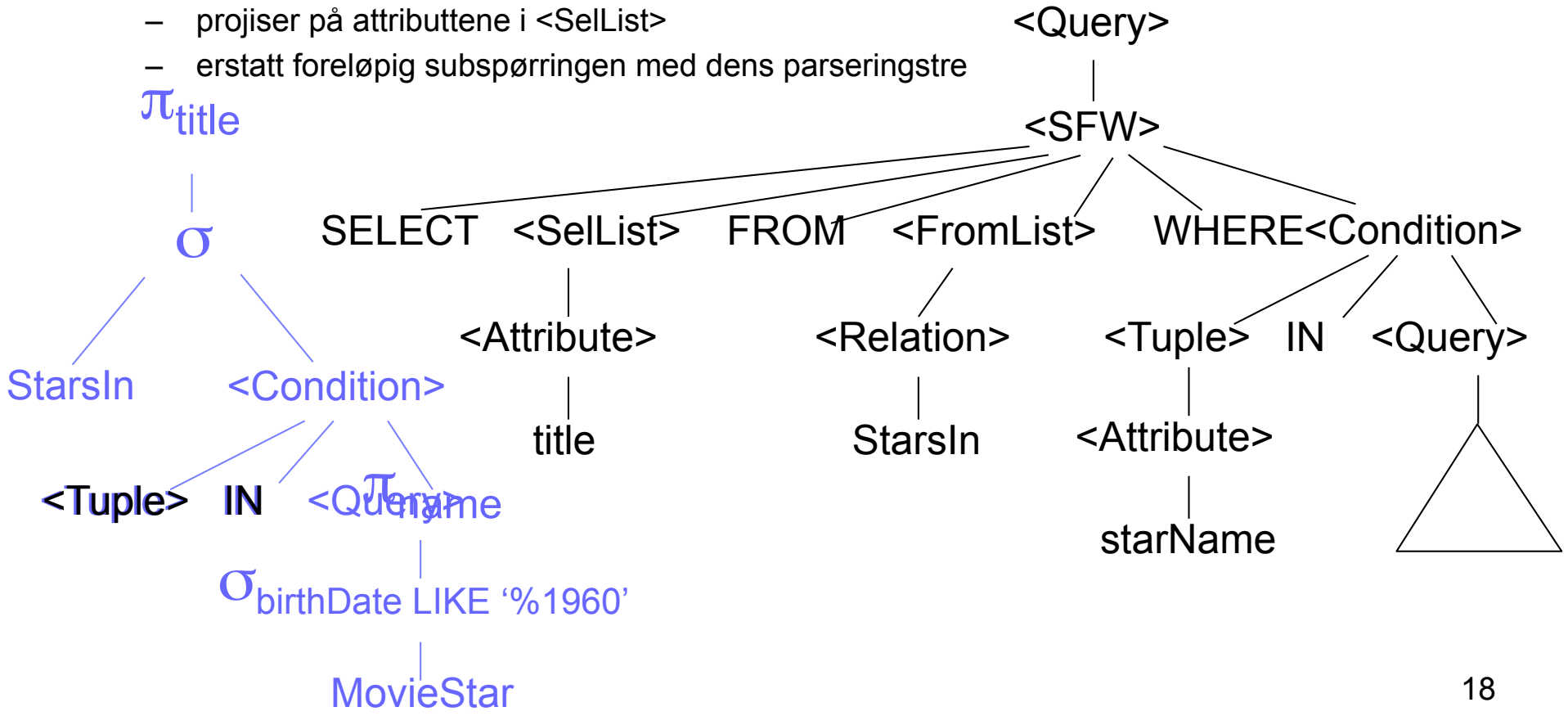
- For subspørringer bruker vi en hjelpe-operator – to-arguments seleksjon $\sigma(R,T)$ der T representerer subspørringen (dvs. det som svarer til <Condition>).
- Videre behandling avhenger av type subspørring. Vi skal se på `t IN S` som et eksempel:
  1. Erstatt <Condition> med treet for S. Hvis S kan inneholde duplikater, må vi legge til en δ–operator på toppen.
  2. Erstatt to-arguments seleksjon med ett-arguments seleksjon σ_C, hvor C sammenligner hver komponent i t med det tilsvarende attributtet i S.
  3. La σ_C ha produktet av R og S som argument.



Konvertering av subspørringer - eksempel

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

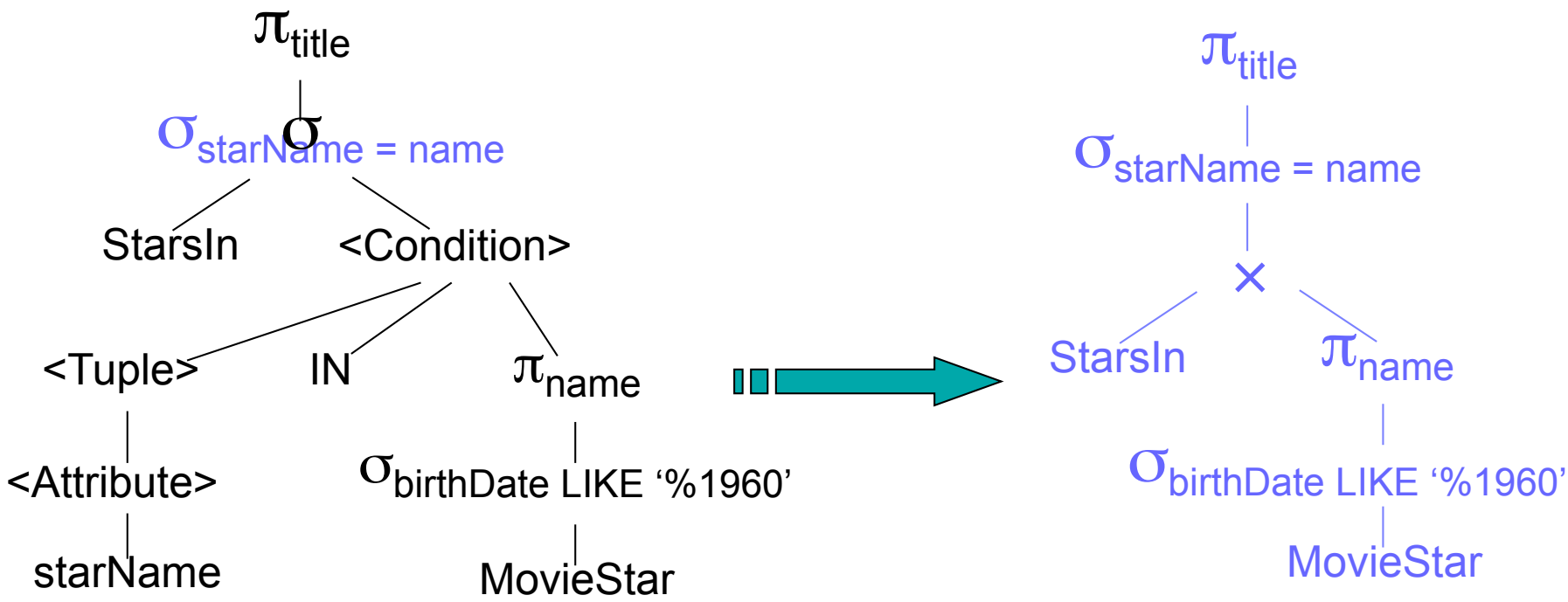
- produktet av relasjonene i <FromList>
- gjør seleksjon basert på <Condition>, representert ved to-arguments seleksjon
- projiser på attributtene i <SelList>
- erstatt foreløpig subspørringen med dens parseringstre



Eksempel (forts.)

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

- erstatt <Condition> med treet for subspørringen
- 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 subspørringer blir mer komplisert hvis subspørringen er relatert til verdier definert utenfor skopet til subspørringen
  - vi må da lage en relasjon med ekstraattributter for sammenligning med de eksterne attributtene
  - de ekstra attributtene fjernes senere ved hjelp av projeksjoner
  - i tillegg må alle duplikattupler fjernes

Forbedring av logiske spørreplaner ved hjelp av algebraiske lover

Kommutativitet og assosiativitet

- En operator ω er kommutativ hvis rekkefølgen til 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

- Naturlig join og produkt er kommutative og assosiative:

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

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

- Hva med thetajoin?
  - Kommutativ:
    $R \bowtie_c S = S \bowtie_c R$
  - Ikke alltid assosiativ.
    Eksempel: Gitt relasjonene R(a,b), S(b,c), T(c,d). Da er

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

Rekkefølgen på produkt og join

- Har rekkefølgen og grupperingen av et produkt eller en join noe å si i forhold til effektivitet?
  - Hvis bare en av relasjonene får plass i minnet, bør denne være første argument – en-passoperasjon som reduserer antall disk-IO
  - Hvis produkt eller join av to av relasjonene gir en temporær relasjon som får plass i minnet, bør disse joinene tas først for å spare både minneplass og disk-IO
  - Temporære relasjoner (mellomresultater) bør være så små som mulig for å spare minneplass
  - Hvis vi kan estimere (ved hjelp av statistikk) antall tupler som skal joines, kan vi spare mange operasjoner ved å joine de relasjonene som gir færrest tupler først (gjelder ikke for produkt)

Algebraiske lover – union og snitt

- Union og snitt er kommutative og assosiative:

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

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

- Merk: Andre lover kan være ulike for **set** og **bag,** f.eks. 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 svært viktig operator for optimalisering
  - reduserer antall tupler (størrelsen på relasjonen)
  - generell regel: *dytt seleksjoner så langt ned i treet som mulig*
- Betingelser med AND og OR kan splittes:
  - $\sigma_{a \text{ AND } b}(R) = \sigma_a(\sigma_b(R))$
  - $\sigma_{a \text{ OR } b}(R) = (\sigma_a(R)) \cup_s (\sigma_b(R))$ der R er en mengde

    (Splitting av OR fungerer bare når R er en mengde: Hvis R er en bag og begge betingelsene er oppfylt, vil bagunion inkludere et tuppel to ganger - en gang i hver seleksjon.)
- Rekkefølgen av påfølgende seleksjoner har ingen betydning for resultatmengden:
  - $\sigma_a(\sigma_b(R)) = \sigma_b(\sigma_a(R))$

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 et 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: $\sigma_a(R \bowtie S) = \sigma_a(R) \bowtie \sigma_a(S) = R \bowtie \sigma_a(S) = \sigma_a(R) \bowtie S$
    - thetajoin: $\sigma_a(R \bowtie_b S) = \sigma_a(R) \bowtie_b \sigma_a(S) = R \bowtie_b \sigma_a(S) = \sigma_a(R) \bowtie_b S$

Dytting av seleksjon – eksempel

- Eksempel: hvert attributt er 1 byte
  - $\sigma_{A=2}(R \bowtie S)$
    - join:
      sammenlign 4 * 4 elementer = 16 operasjoner
      mellomlagre relasjon: R ⋈ S gir 52 bytes
    - seleksjon:
      sjekk hvert enkelt tuppel: 2 operasjoner
  - $\sigma_{A=2}(R) \bowtie S$
    - seleksjon:
      sjekk hvert enkelt tuppel i R: 4 operasjoner
      mellomlagre relasjon: $\sigma_{A=2}(R)$ gir 24 bytes
    - join:
      sammenlign 1 * 4 elementer = 4 operasjoner

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 ⋈ S

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

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

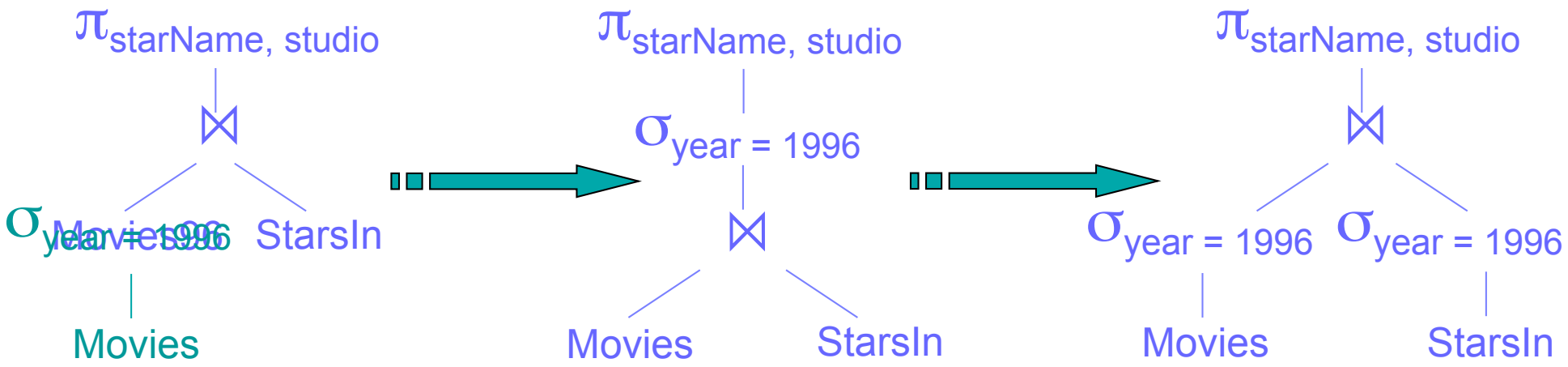
A	B	C	...	X
2	c	6	...	2

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 I

Projeksjon kan dyttes gjennom join og produkt (dvs. nye projeksjoner kan introduseres) 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 inneholder joinattributtene og det av L som er i R
  - N inneholder joinattributtene og det av L som er i S
- $\pi_L(R \bowtie_C S) = \pi_L(\pi_M(R) \bowtie_C \pi_N(S))$ hvis
  - M inneholder de av attributtene i C og L som er i R
  - N inneholder de av attributtene i C og L som er i S
- $\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))$ hvis
  - M inneholder de av attributtene i L som er i R
  - N inneholder de av attributtene i L som er i S

Algebraiske lover – projeksjon II

Projeksjon kan dyttes gjennom bagunion:

- $\pi_L(R \cup_B S) = \pi_L(R) \cup_B \pi_L(S)$

- Merk: Tilsvarende regel gjelder *ikke* for mengdeunion, mengdesnitt, bagsnitt, mengdedifferanse eller bag-differanse fordi projeksjon kan endre multiplisiteten til tuplene:
  - Hvis R er en mengde, er ikke nødvendigvis $\pi_L(R)$ en mengde
  - Hvis R er en bag og et tuppel t forekommer k ganger i R, så kan projeksjonen av t på L forekomme mer enn k ganger i $\pi_L(R)$.

Algebraiske lover – projeksjon III

Projeksjon kan dyttes gjennom seleksjon (nye projeksjoner introduseres) så lenge vi ikke fjerner attributter som brukes lenger oppe i treet:

- $\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$ hvis
  - M inneholder attributtene i C og L

Merk at hvis R har en indeks på noen av attributtene i seleksjonsbetingelsen C, så vil ikke denne indeksen kunne benyttes under seleksjonen hvis vi først gjør en projeksjon på M.

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 kandidatnøkkel 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
      mellomlagre relasjon R × S = 1.000.000 * (11 + 16) = 27.000.000 bytes
    - seleksjon:
      sjekk hvert enkelt tuppel: 1.000.000 operasjoner

      mellomlagre 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 temporære relasjoner ved å dyttes gjennom
  - kartesisk produkt: $\delta(R \times S) = \delta(R) \times \delta(S)$
  - join: $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
  - thetajoin: $\delta(R \bowtie_C S) = \delta(R) \bowtie_C \delta(S)$
  - seleksjon: $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$
  - **bag**-snitt: $\delta(R \cap_B S) = \delta(R) \cap_B \delta(S) = \delta(R) \cap_B S = R \cap_B \delta(S)$

- Merk: duplikateliminasjon kan *ikke* dyttes gjennom
  - **set**-operasjoner
  - **bag**-union og -differanse
  - projeksjoner

Algebraiske lover - grupperingsoperatoren

- Kan ikke gi generelle lover

Forbedring av logiske spørreplaner

- De vanligste optimaliseringene er:
  - Dytt seleksjoner så langt ned som mulig.
    Hvis seleksjonsbetingelsen 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.
  - Kombiner seleksjon og kartesisk produkt til en join av passende type
  - Duplikateliminasjoner kan noen ganger fjernes
- Men ikke ødelegg utnyttelse av indekser:
  - Dytting av projeksjon forbi en seleksjon kan ødelegge for bruk av indekser ved seleksjonen