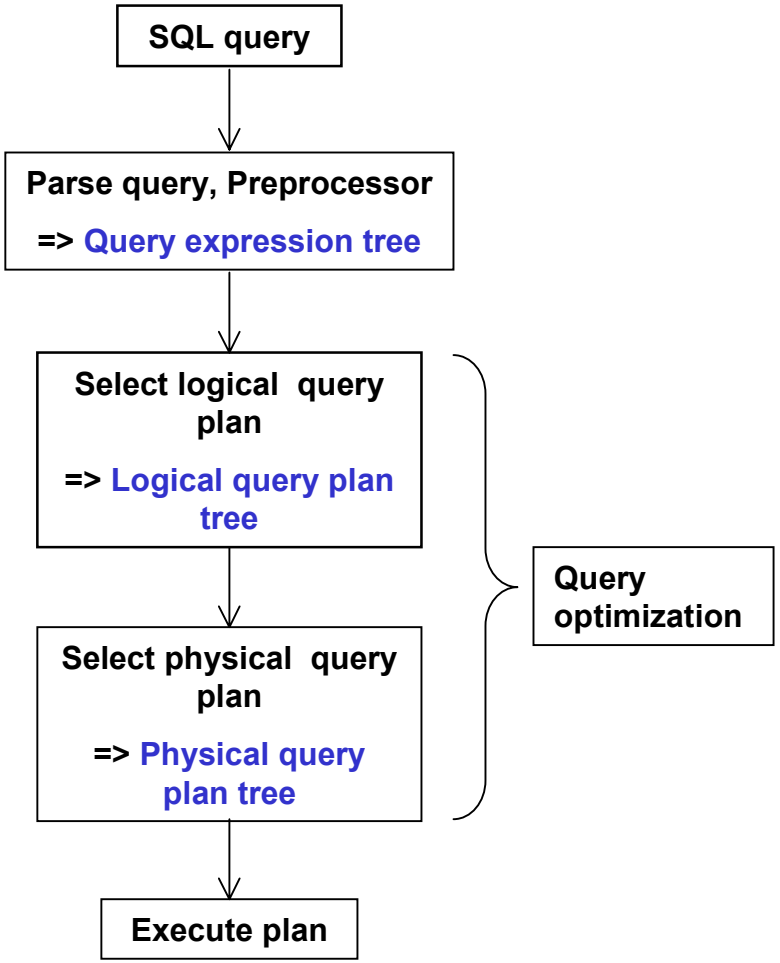
Užklausų kompiliavimas

(Query Compiler)

paruošė Margarita Kazakevičiūtė



1.Užklausų kompiliavimas (Query compiliation)

Sudaro 3 pagrindiniai žingsniai:

1.1 ***Parsing'as*** – iš SQL sakinio sudaromas parse medis (**parse tree**), atspindintis užklausos strukūrą.

1.2 ***Loginis užklausos planas (logical query plan)*** – algebrinė užklausos išraiška. Sudaromas loginio plano išraiškų medis (**logical plan tree**), kur parse medžio SQL opearacijos yra pakeistos loginėmis/algebrinėmis operacijomis. Viena užklausa gali turėti kelis loginius planus.

1.3 ***Fizinis užklausos planas (physical query plan)*** – kiekviena loginio plano algebrinė operacija yra pakeičiama atitinkamu operacijos algoritmu. Sudaromas fizinio plano medis (**physical plan tree**).

2-ras ir 3-čias žingsniai apima ir užklausos optimizavimo veiksmus (***query optimization***):

- Pradinis loginis užklausos planas yra keičiamas optimalesniu ekvivalentišku algebriniu planu.

b) Kiekvienai loginio plano operacijai parenkamas optimalus ją vykdantis algoritmas, parenkama gera operatorių vykdymo tvarka.

c) Nagrinėjama, kokiu būdu duomenys turi būti perduodami iš vienos operacijos į kitą, t.y. per pagrindinę atmintį, kietąjį diską, ar kitu būdu.

2. Parse medžiai

Iš SQL sakinio sudaromas parse medis (**parse tree**), atspindintis užklausos strukūrą.

Parse medžio **viršūnėse** yra:

- SQL raktiniai žodžiai (keywords: SELECT ir pan.),
- laukų, lentelių pavadinimai,
- konstantos,
- operatoriai (+, <, >, AND, OR ir t.t.),
- kategorijos, atstovaujančios užklausos dalį (<Condition>, <Query>, ir pan.).

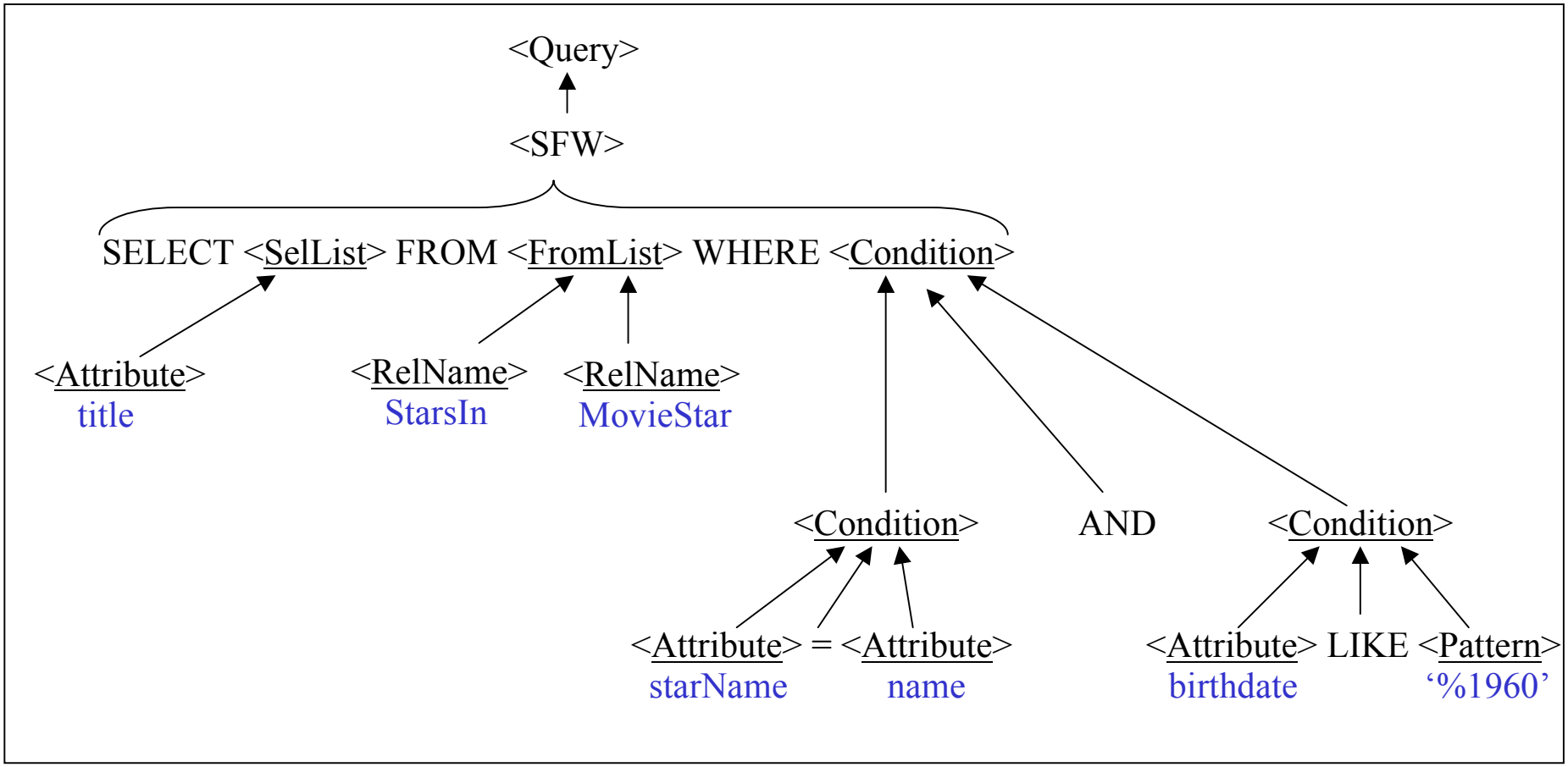
Pažymėjimai:

- **<SFW> = SELECT <SelList> FROM <FromList> WHERE <Condition>**
- **<Tuple>** - įrašas
- **<Attribute>** - laukas
- **<SelList>** - laukų sąrašas
- **<Relation>** - lentelė
- **<FromList>** - lentelių sąrašas
- **<Condition>** - sąlyga (WHERE <Condition>)
- **<Pattern>** - šablonas (LIKE <Pattern>)

Pvz.: **MovieStar** (name, addr, gender, birthdate), **StarsIn** (title, year, strName)

Rasti filmus, kur aktoriai gimę 1960.

SELECT title **FROM** StarsIn, MovieStar **WHERE** starName = name **AND** birthdate **LIKE** '%1960'



Pav.: Parse medis

2.1 Preprocesorius (Preprocessor)

Preprocesorius atlieka funkcijas:

- Patikrina, ar egzistuoja FROM- sąraše išvardintas lentelės ir užklausos;
- Patikrina, ar nurodytose lentelėse egzistuoja SELECT- ir WHERE- sąrašuose išvardinti stulpeliai;
- Patikrina tipų suderinamumą išraiškose, ryšiuose, Where sąlygoje.

Jei **parse medis** netenkina bent vienos iš šių sąlygu, grąžinama klaida.

Tolimesnis užklausos vykdymas nutraukiamas.

3. Loginio užklausos plano algebriniai operatoriai

3.1 Union, Intersection, Difference operatoriai

Atliekant algebrines operacijas reikia skirti **aibes (sets)** ir **bag'us**:

- ***Aibėse (set)*** kiekvienas elementas (įrašas) kartojasi po vieną kartą;
- ***Bag'uose*** elementas (įrašas) gali kartotis kelis kartus.

Duomenų bazių lentelės yra bag'ai.

Aibių (set) atveju yra standartinės algebrinės operacijos:

sąjunga $\cup_S$, sankirta $\cap_S$, skirtumas $-_S$

SQL komandos **UNION, INTERSECT, EXCEPT** eliminuoja pasikartojancius irašus, t.y. jos veikia kaip aibiu operacijos.

Bag'ų atveju:

R $\cup_B$ P – įrašas **t** įena: k=(#R + #P) kartų;

R $\cap_B$ P – įrašas **t** įena: k=min(#R , #P) kartų;

R $-_B$ P – įrašas **t** įena: k=(#R - #P) kartų, jei k>0, 0-kartų, jei k<=0;

Bag'us atitinka SQL komandos: **UNION ALL, INTERSECT ALL, EXCEPT ALL**.

3.2 Selection operacija

$\sigma_C(R)$ – atitinka SQL sakini: **SELECT * FROM R WHERE <C>** , kur **C** - <Condition>.

Rezultatas yra bag'as.

3.3 Projection operacija

$\pi_L(R)$ – atitinka SQL sakini: **SELECT <L> FROM R** , kur **L** – atrenkamu lauku sarašas.

Rezultatas yra bag'as.

3.4 Product operacija

$R \times S$ – atitinka SQL sakini: **SELECT * FROM R, S**

Jei m = #R, n = #S, tai #(R x S) = mn.

Rezultatas yra bag'as.

3.5.1 Natural-Join operacijos

R ⇔ S – atitinka SQL sakini: **SELECT R.a, R.b, S.c FROM R(a,b), S(b,c) WHERE R.b=S.b**

Arba: $R \Leftrightarrow S = \pi_L(\sigma_C(R \times S))$, kur **C= {R.b=S.b}, L={a,b,c}.**

Rezultatas yra bag'as.

3.5.2 Theta-Join operacija

$R \Leftrightarrow_C S$ – atitinka SQL sakini: **SELECT * FROM R(a,b), S(b,c) WHERE R.b=S.b**

Arba: $R \Leftrightarrow_C S = \sigma_C(R \times S)$, kur **C= {R.b=S.b},** lauku sarašas bus **L={a,b,b,c}.**

Rezultatas yra bag'as.

3.6 Duplicate Elimination operacija

δ (R) – atitinka SQL sakini: **SELECT DISTINCT * FROM R**

Iš bag'o padaro aibę, t.y. rezultatas yra aibė (set).

3.7 Grouping, Aggregation operacijos

Grupavimo ir agregavimo operacijos visada eina kartu.

$\gamma_L(R)$ – atitinka SQL sakini:

SELECT <L>, Aggreg(<L>) FROM R GROUP BY <L> HAVING <Condition>

kur **L** – bendras sąrašas laukų, pagal kuriuos grupuojama, (grouping attributes) ir kuriems taikomos agregavimo operacijos (aggregated attributes),

Aggreg - agregavimo operatoriai: **AVG, SUM, COUNT, MIN, MAX,**

Grupavimo operacija eliminuoja pasikartojančius įrašus, t.y. rezultatas yra aibė (set).

3.8 Rūšiavimo operatorius

$\tau_L(R)$ – atitinka SQL sakini: **SELECT * FROM R ORDER BY <L>**

kur L – sarašas lauku, pagal kuruos atliekamas lenteles rušiavimas.

Rezultatas nera aibe ar bag'as, o surušiuotas irašu sarašas, tuo tarpu, kai kitos operacijos gražina aibes arba bagus. Todel rušiavimo operacijos algebrinese išraiškose yra atliekamos paskutineje eileje.

4. Algebrinės Išraiškos medis (expression tree)

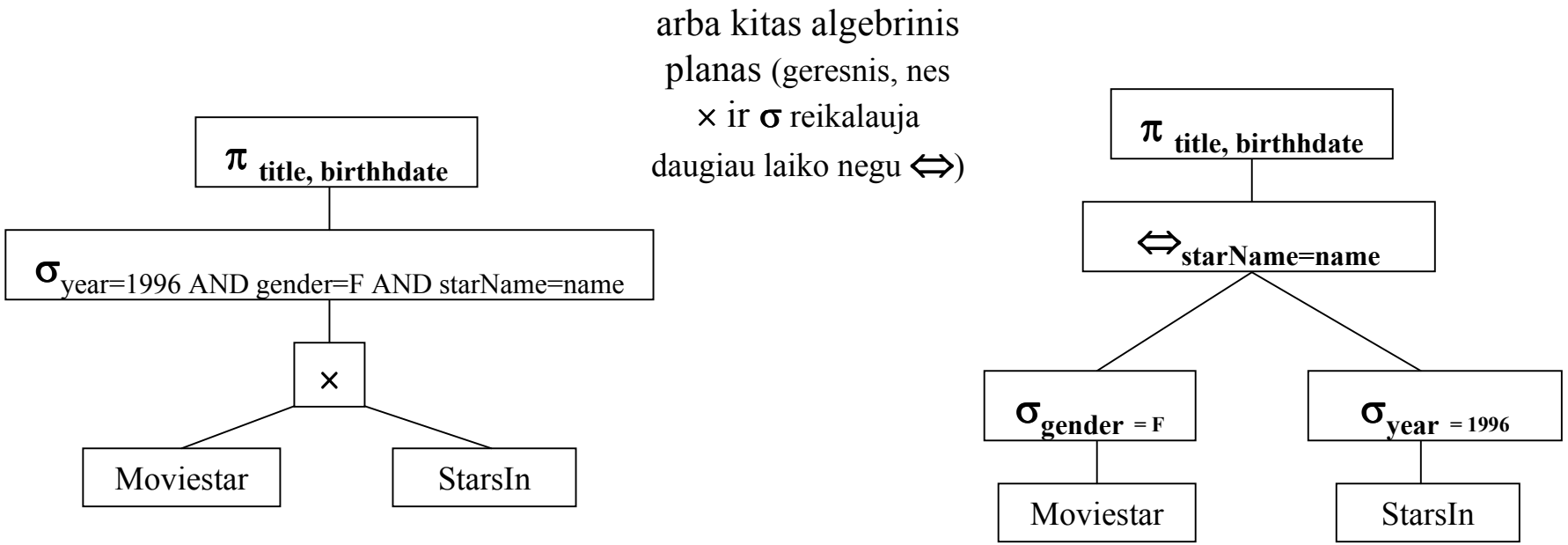
SQL sakinį galima užrašyti kaip algebrinę išraišką, kombinuojant algebrinius operatorius, kai vienas operatorius pritaikomas kito rezultatui (parse medyje SQL operatoriu pakeitus algebriniais operatoriais).

Algebrinę išraišką galima pavaizduoti kaip **išraiškos medį (expression tree).**

Medžio **lapai** yra lentelių pavadinimai, **vidiniai mazgai** yra algebriniai operatoriai.

Pvz: MovieStar (name, addr, gender, birthdate), StarsIn (title, year, strName)

SELECT title, birthdate FROM MovieStar, StarsIn WHERE year=1996 AND gender='F' AND starName = name

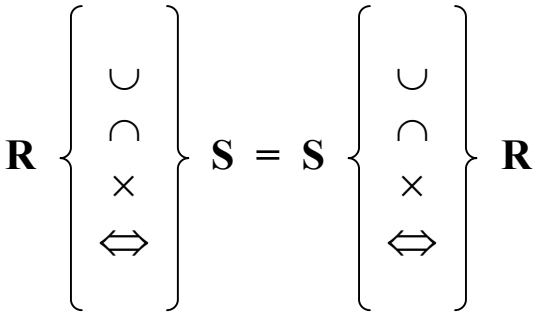


5. Algebrinių operacijų dėsniai

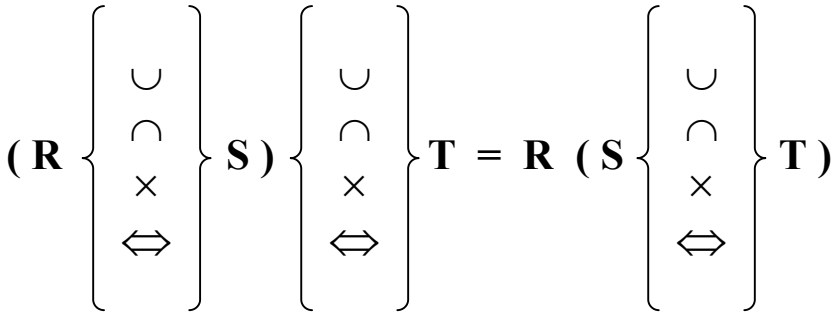
5.1 Komutatyvumas (x+y=y+x), Asociatyvumas ((x+y)+z = x+(y+z))

Kai algebrinis operatorius yra komutatyvus ir asociatyvus, visus šio operatoriaus operandus grupuojant ir keičiant jų tvarką, bendras operatoriaus rezultatas nesikeis.

Tai galima pritaikyti optimizuojant užklausos loginius planus.



Komutatyvios operacijos



Asociatyvios operacijos

Reik atkreipti dėmesį į skirtumus tarp aibių ir bag'ų. Aibėms turime:

$\mathbf{A \cap_S (B \cup_S C) = (A \cap_S B) \cup_S (A \cap_S C)}$

Tačiau neteisinga bag'ams:

Pvz.: kai A=B=C={x}: $\mathbf{A \cap_B (B \cup_B C) = \{x\}}$, $\mathbf{(A \cap_B B) \cup_B (A \cap_B C) = \{x, x\}}$

5.2. Selection σ dėsniai

Pagrindinė taisyklė:

Perkelti Selection veiksmą σ (WHERE) algebriniame medyje kaip galima į žemesnį lygmenį (push-down).

Tai daugeliu atvejų smarkiai optimizuoja užklausą, kuo anskčiau sumažinamas įrašų, su kuriais dirbama, skaičius. Yra išimtiniai atvejai, kai šis veiksmas nepasiteisina.

Veiksmo σ perkelimo žemyn (push-down) algebriniame medyje desniai:

- **<Condition>** sąlygą išskaidyti į sudedamąsias dalis.

Sąlygos dalis, kuriose dalyvayja tik vienos lentelės laukai, nuleisti į žemesnį medžio lygį:

- $(\sigma_{C1}$ **AND** $\sigma_{C2})\ (R) = \sigma_{C1}\ (\sigma_{C2}\ (R))$
- $(\sigma_{C1}$ **OR** $\sigma_{C2})\ (R) = \sigma_{C1}\ (R) \cup_S \sigma_{C2}\ (R)$,

(galioja tik aibems, negalioja bag'ams, nes $\cup_S$ eliminuoja pasikartojancius irašus)

Veiksmo σ perkelimo žemyn (push-down) algebriniame medyje desniai:

2. Sąlygos, kaip selection veiksmą leisti į žemesnį medžio lygį per **binarines operacijas**:

- **Sajunga (union) – σ** butina leisti i abudu sajungos operatoriaus argumentus (abi sajungos operatoriaus šakas):

  $\sigma_C (R \cup S) = \sigma_C (R) \cup \sigma_C (S)$

- **Skirtumas (difference) – σ** butina leisti i pirmaji skirumo operatoriaus argumenta. Galima pasirinktinai ir i antraji, tai rezultatui dideles itakos neturi:

  $\sigma_C (R - S) = \sigma_C (R) - S$ **, arba** $\sigma_C (R - S) = \sigma_C (R) - \sigma_C (S)$

- **Sankirta (Intersection), Sandauga (Product), Saryšis (Join)** - **σ** butina leisti bent i viena operatoriaus argumenta. Galima pasirinktinai ir i antraji, tai rezultatui dideles itakos neturi:

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

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

  $\sigma_C (R \Leftrightarrow S) = \sigma_C (R) \Leftrightarrow S$;

  $\sigma_C (R \Leftrightarrow_C S) = \sigma_C (R) \Leftrightarrow_C S$;

  Jei salyga C neišsiskaido i nepriklausomas dalis: $\sigma_C (R \Leftrightarrow S) = \sigma_C (R) \Leftrightarrow \sigma_C (S)$

Veiksmo σ perkelimo žemyn (push-down) algebriniame medyje desniai:

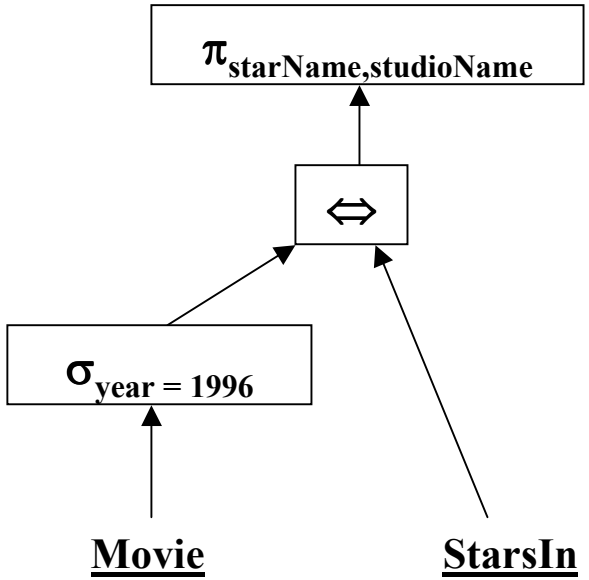
ATVIRKŠTNIS ATVEJIS:

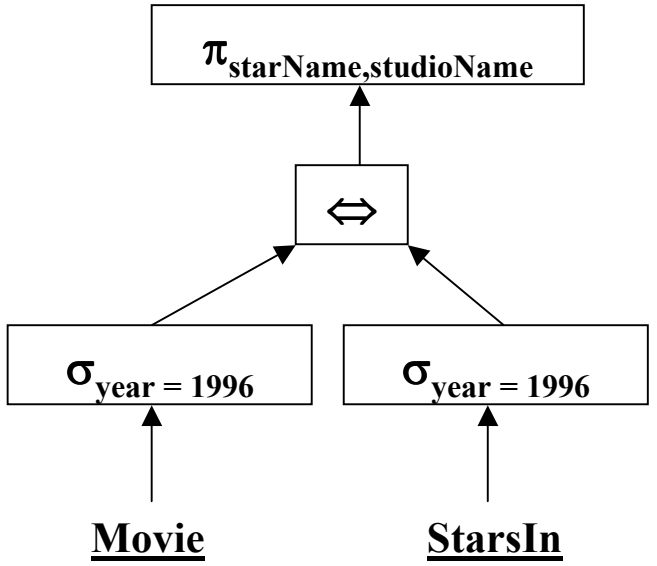
- Galimas atvejis, kai optimaliau selection veiksmą pakelti aukštyn, ir tuomet leisti visomis galimomis šakomis į žemesnį medžio lygį:

Pvz.: StarsIn(title, year, starName), Movie (title, year, length, studioName)

CREATE VIEW MoviesOf1996 As SELECT * FROM Movie WHERE year = 1996;

SELECT starName, studioName FROM MoviesOf1996 NATURAL JOIN StarsIn





5.3. Projection π dėsniai

Veiksmas π (SELECT a1, …, an) - dažnai projekcija išlieka pradinėje pozicijos, ir papildomai nustūmiama į žemesnį medžio ligį (push-down).

Projekcija neduoda tokio didelio optimzavimo, kaip Selection operatoriaus nustūmimas, nes įrašų skaičius nesumažinamas, gali būti sumažinamas tik laukų skaičius.

Projekcijos π perkelimo žemyn (push-down) medyje desniai:

- **Projekciją galima taikyti betkuriame algebrinio medžio mazge**, kur aukštesnuose lygiuose arba galutiniame rezultate nėra naudojami eliminuojami laukai.
- $\pi_L (R \Leftrightarrow S) = \pi_L (\pi_M (R) \Leftrightarrow \pi_N (S))$, $\pi_L (R \Leftrightarrow_C S) = \pi_L (\pi_M (R) \Leftrightarrow_C \pi_N (S))$,

  kur M – visi ryšyje (salygoje C) naudojami ir saraše L esantys R lenteles laukai, N – visi ryšyje naudojami ir saraše L esantys S lenteles laukai.

3. $\pi_L (R \times S) = \pi_L (\pi_M (R) \times \pi_N (S))$, kur M – visi saraše L esantys R lenteles laukai, N – visi saraše L esantys S lenteles laukai.
4. $\pi_L (\sigma_C (R)) = \pi_L (\sigma_C (\pi_M (R)))$, kur M –salygoje C naudojami ir saraše L esantys R laukai.
5. $\pi_L (R \cup_B S) = \pi_L (R) \cup_B \pi_L (S)$.
6. **Projekcijos negalima nustumti žemiau sajungos aibiu atveju, ir žemiau sankirtos ir skirtumo aibiu ir bag'u atveju.** Pvz.: R(a,b)={(1,2)}, S(a,b)={(1,3)}. Tada: $\pi_L (R \cap S) = 0$, $\pi_L (R) \cap \pi_L (S) = \{(1)\}$

5.4. Join ir Product dėsniai

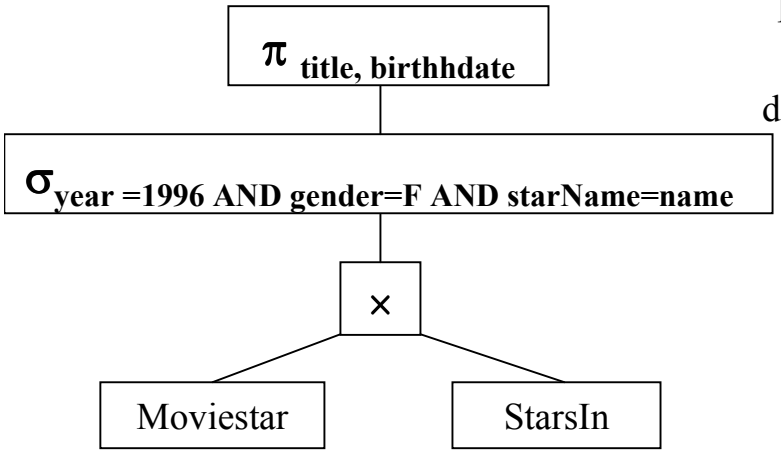
Sąryšio (join) ir sandaugos (produckt) dėsniai (=> iš apibrėžimo):

- **$R \Leftrightarrow S = \pi_L (\sigma_C (R \times S))$,**
- **$R \Leftrightarrow_C S = \sigma_C (R \times S)$**

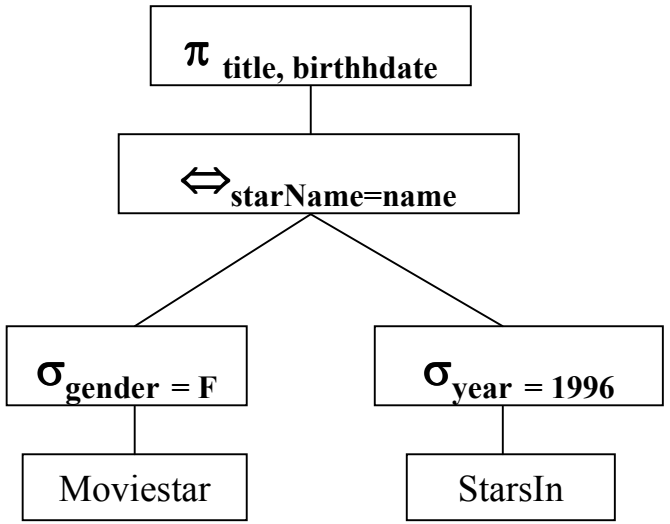
Optimizuojant apsimoka Sandaugos (Product) ir Select operaciju kombinacija keisti saryšiu (Join), nes esant dideliems duomenu kiekiams saryšio operatoriaus algoritmai veikia greicau negu Product ir Select operaciju kombinacijos algortmai.

Pvz: MovieStar (name, addr, gender, birthdate), StarsIn (title, year, strName)

SELECT title, birthdate FROM MovieStar, StarsIn WHERE year=1996 AND gender='F' AND starName = name



arba kitas algebrinis planas (geresnis, nes × ir σ reikalauja daugiau laiko negu ⇔)



5.5. Duplicate Elimination dėsniai

Duplicate Elimination δ dėsniai:

1. δ (R) = R, jei R nera pasikartojanciu irašu.

Lenetelems su pirminiu raktu (primary key) ir po γ (GROUP BY) operacijos gautiems irašamas, nes grupavimo operacija eliminuoja pasikartojancius irašus.

2. $\delta (R \times S) = \delta (R) \times \delta (S)$

3. $\delta (R \Leftrightarrow S) = \delta (R) \Leftrightarrow \delta (S)$, $\delta (R \Leftrightarrow_C S) = \delta (R) \Leftrightarrow_C \delta (S)$

4. $\delta (\sigma_C (R)) = \sigma_C(\delta(R))$

5. $\delta (R \cap_B S) = \delta(R) \cap_B S = R \cap_B \delta(S) = \delta(R) \cap_B \delta(S)$

6. $\delta (R \cup_S S) = R \cup_S S$

5.6. Grupavimo ir agregavimo operacijų γ dėsniai

Grupavimo ir agregavimo operacijos γ dėsniai:

- Po γ operacijos gautiems irašamas nereikia taikyti duplicate elimination δ, nes grupavimo operacija eliminuoja pasikartojancius irašus:

$$\delta (\gamma_L (R)) = \gamma_L (R) \qquad \text{<- cia } \delta \text{ - nereikalingas}$$

2. Prieš γ operacija irašamas taikyti duplicate elimination δ galima tik dviem atvejais:

- Kai agreguojancios funkcijos yra **MIN** arba **MAX,** duplicate elimination δ operacija galutiniam rezultatui itakos neturi:

$$\gamma_L (R) = \gamma_L (\delta (R))$$

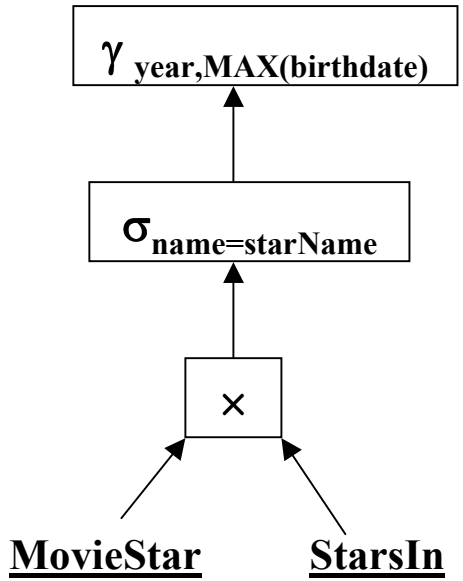
Tokiu atveju γ operacija vadinama **duplicate-invariantiška** (duplicate-impervious).

b) Kai agreguojancios funkcijos yra **SUM, COUNT** arba **AVG,** duplicate elimination δ operacija gali pakeisti galutini rezultata, todel δ veiksmo šiuo atveju negalima atlikti prieš grupavima.

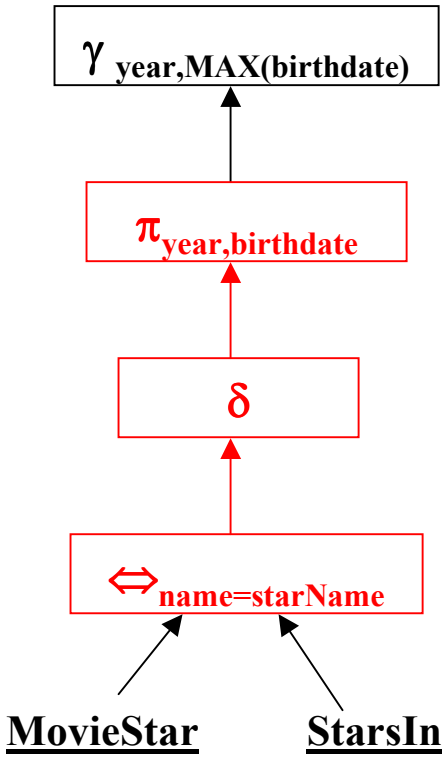
Grupavimo ir agregavimo operacijos γ desniai:

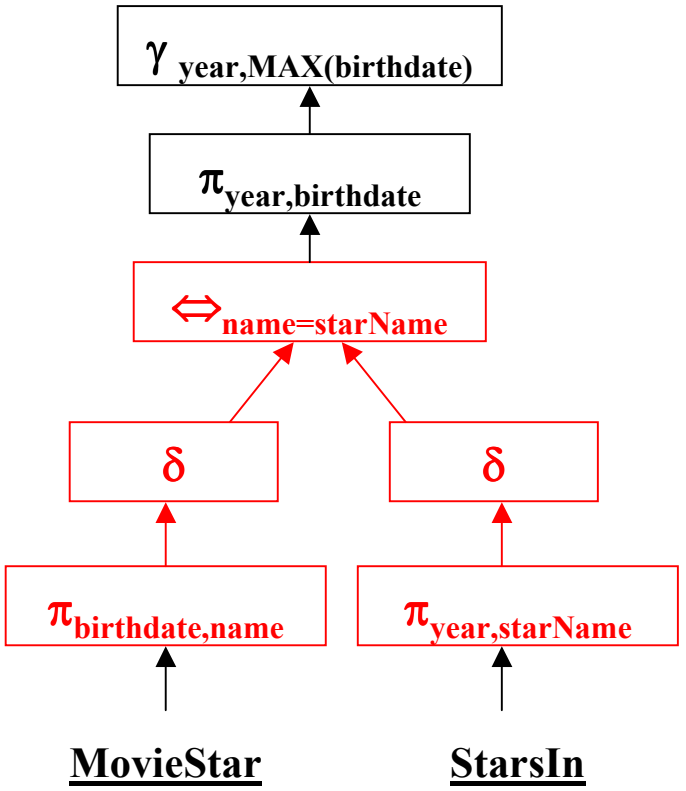
Pvz.: MovieStar (name, addr, gender, birthdate), StarsIn (title, year, starName)

SELECT year**, MAX(**birthdate**) FROM** MovieStar**,** StarsIn **WHERE** name=starName **GROUP BY** year ;



Pradinis užklausos planas





6. Parse medžių konvertavimas į Loginius užklausos medžius

Du žingsniai:

- Parse medžio viršūnėse esančius SQL operatorius pakeisti algebriniais operatoriais;
- Gautą algebrinį medį optmizuoti, pasinaudojant algebrinių operacijų dėsniais.

6.1 Parse medžio konvertavimas į algebrinį medį, kai <Condition> nėra subužklausa

Pagrindinė taisyklė:

Turime **<SFW>** užklausą: **<SFW> = SELECT <SelList> FROM <FromList> WHERE <Condition>**

kur <Condition> nėra subužklausa, o paprastas reiškinys. Tada visa <SFW> užklausą iš karto galime pakeisti algebrine išraiška, kuri konstruojama iš apačios į viršų:

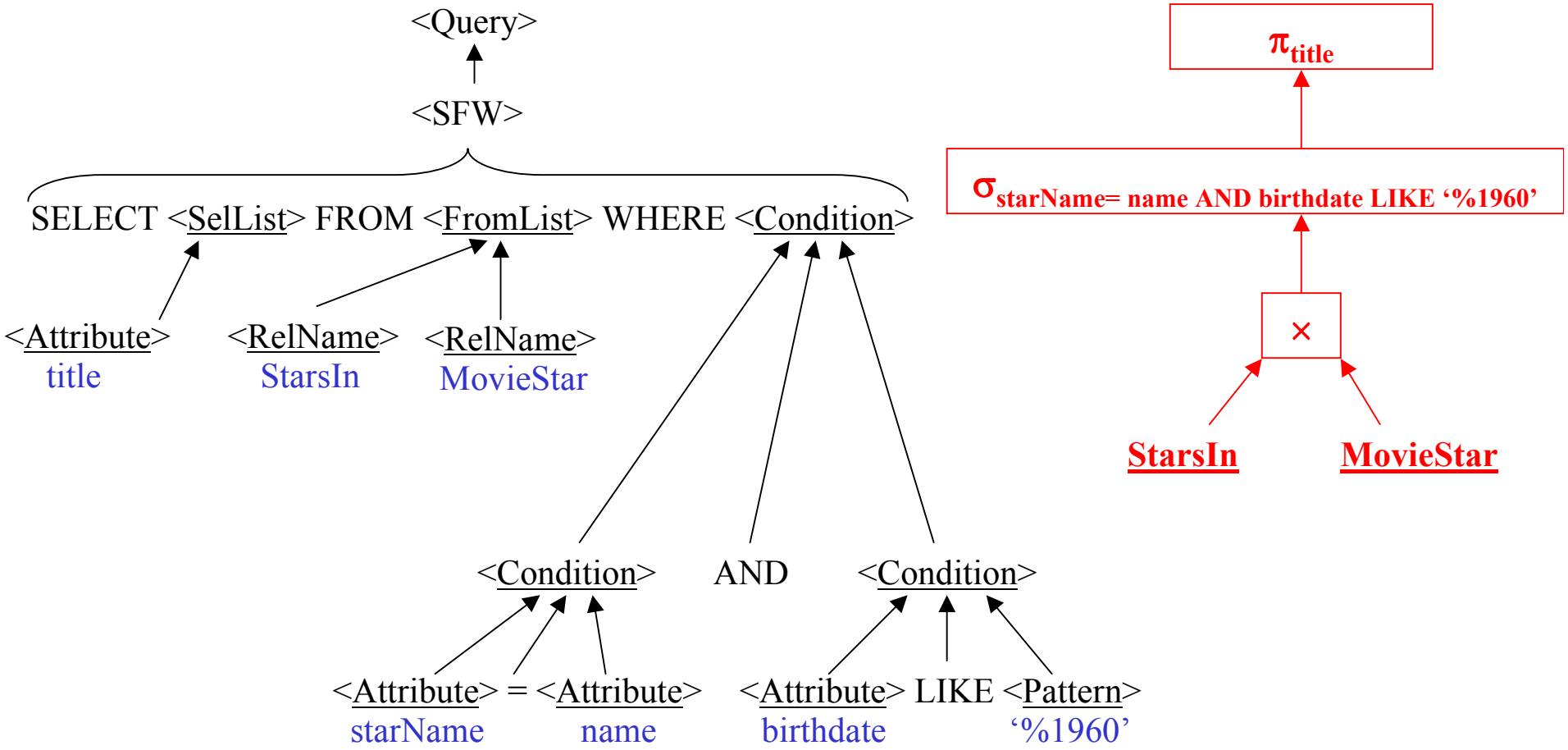
- Imama sandauga × (product) visų <FromList> sąraše išvardintų lentelių. Rezultatas **R1** yra 2-ros operacijos argumentas:
- Taikomas selection operatorius σ_C **(R1)**, kur C=<Condition>. Rezultatas **R2** yra 3-čios operacijos argumentas:
- Taikoma projekcija π_L **(R2)**, kur L=<SelList>.

=> Algebrine **<SFW>** išraiška: $\pi_{<SelList>} (\sigma_{<Condition>} (\times (<FromList>)))$

Pvz.: **MovieStar** (name, addr, gender, birthdate), **StarsIn** (title, year, starName)

Rasti filmus, kur aktoriai gimę 1960.

SELECT title **FROM** StarsIn, MovieStar **WHERE** starName = name **AND** birthdate **LIKE** '%1960'



6.2 Parse medžio konvertavimas į algebrinį medį, kai <Condition> yra subužklausa

Pagrindinė taisyklė:

Turime **<SFW>** užklausą: **<SFW> = SELECT <SelList> FROM <FromList> WHERE <Condition>**

kur <Condition> yra subužklausa.

- Pažymėkime σ - selection operatoriu, kuris gali tureti du argumentus (two argument selection). Tokio σ mazgo kairysis vaikas bus lentele R iš <FROM> sarašo, dešinysis vaikas bus <Condition> išraiška atspindintis pomedis.
- Operatorių σ pakeisime standartiniu vieno argumento select operatoriumi $\sigma_C()$, arba kitais algebriniais operatoriais.

2.a) **Kai subužklausa nėra susijusi su išorine užklausa (uncorrelated)**, ją galima paskaičiuoti vieną kartą.

Tegu σ turi du argumentus: **R – lentele, <Condition>= t IN S**, kur **S** yra nesusieta subužklausa, **t** – yra lenteles R stulpeliu sarašas. Tada medi transformuosim pagal taisykles:

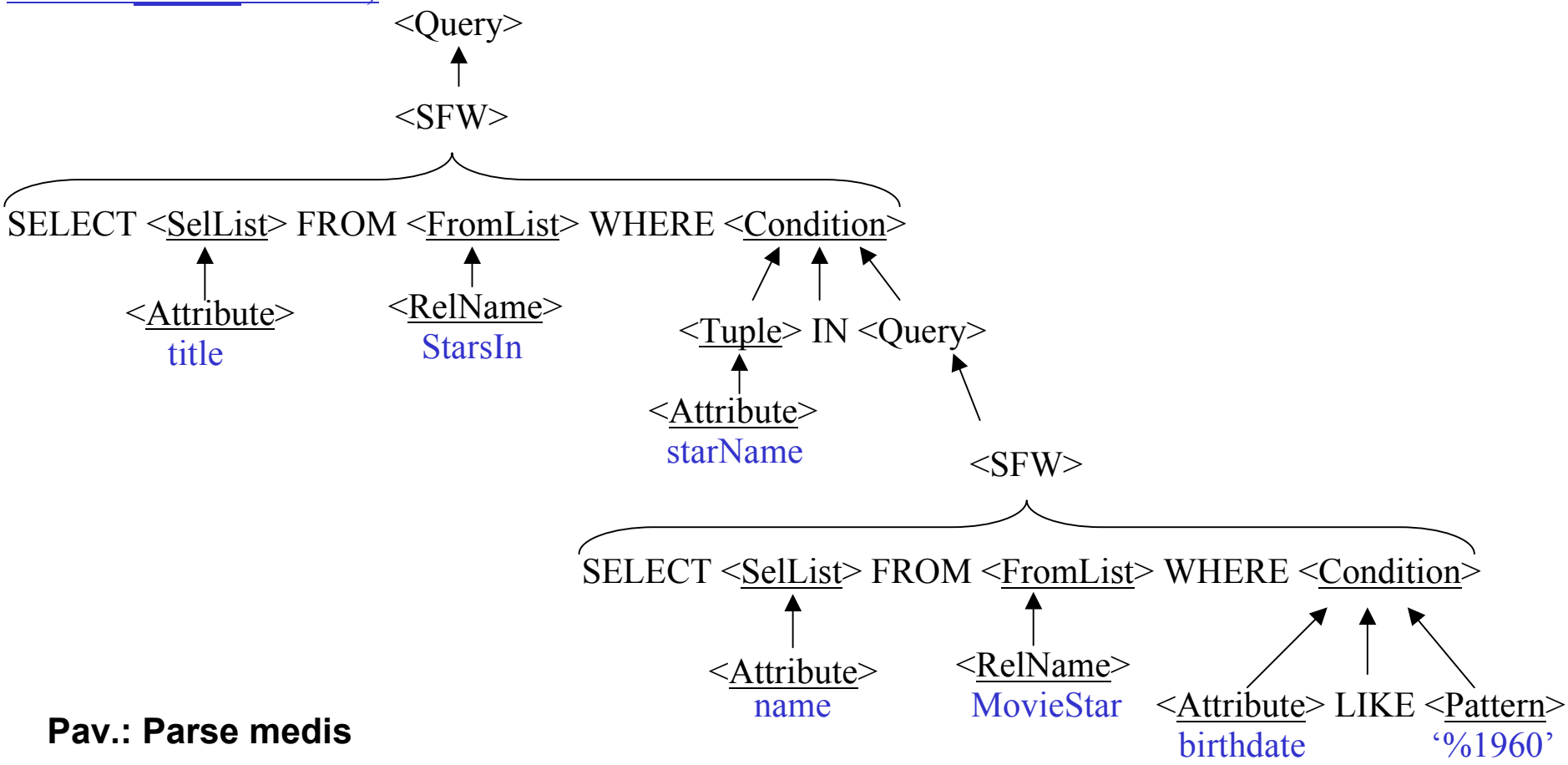
- Mazgas <Condition> pakeiciamas subužklausos **S** medžiu. Jei **S** gali tureti pasikartojanciu reikšmiu, **S**-medžio šaknyje pridedamas operatorius δ.
- Dvieju argumentu select operatorius σ keiciamas vieno argumentu select operatoriumi $\sigma_C(\mathbf{P})$,

  kur : **C** - yra salyga prilyginanti kiekviena stulpeli iš **t** atitinkamam **S** stulpeliui, **P = R × S**

Pvz.: **MovieStar** (name, addr, gender, birthdate), **StarsIn** (title, year, starName)

Rasti filmus, kur aktoriai gimę 1960.

SELECT title **FROM** StarsIn **WHERE** starName IN (SELECT name **FROM** MovieStar **WHERE** birthdate **LIKE** '%1960')

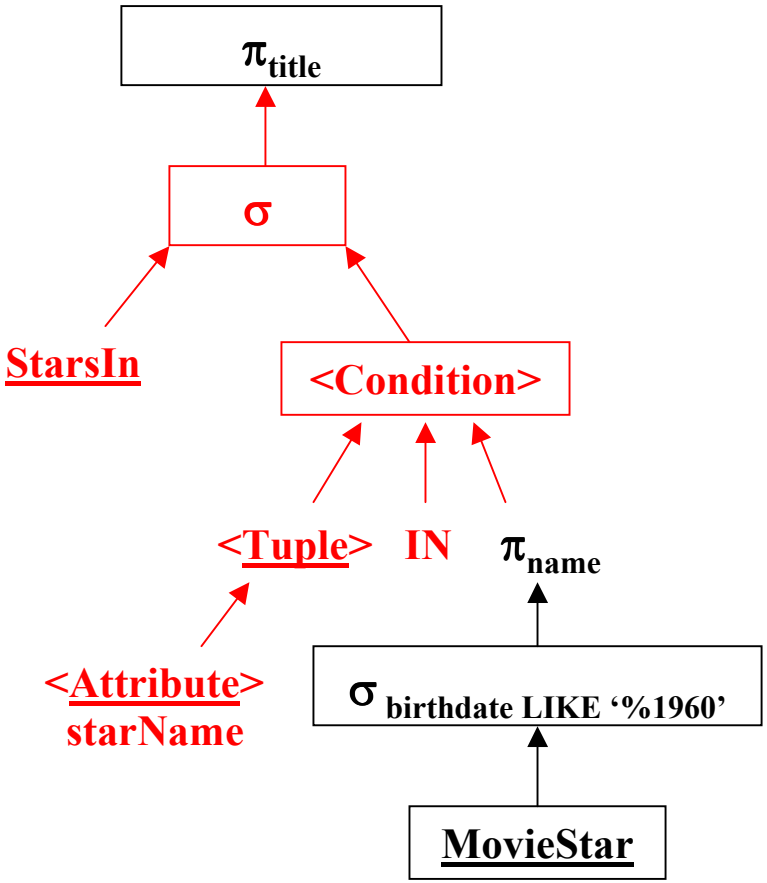


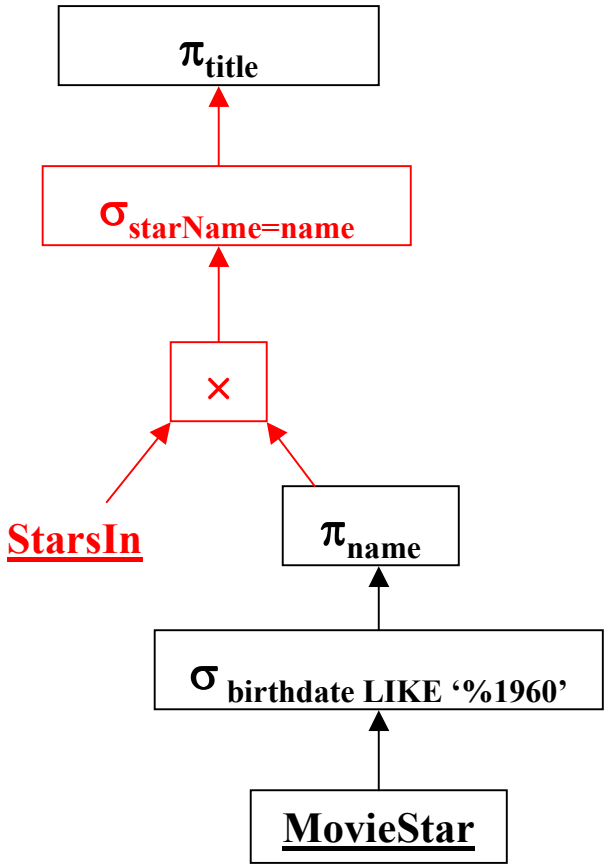
Pav.: Parse medis

Pvz.: **MovieStar** (name, addr, gender, birthdate), **StarsIn** (title, year, starName)

Rasti filmus, kur aktoriai gimę 1960.

SELECT title **FROM** StarsIn **WHERE** starName IN (SELECT name **FROM** MovieStar **WHERE** birthdate **LIKE** '%1960')





6.3 Parse medžio konvertavimas į algebrinį medį, kai <Condition> yra Subužklausa

2.b) Kai <Condition> subužklausa yra susijusi su išorine užklausa (correlated):

- Subužklausa turi būti sudaryta taip, kad grąžintų vidinius laukus, kurie aukštesniame medžio lygyje bus naudojami sudarant ryšį (relation) su išorinės užklausos laukais.
- Išorinėje užklausoje nenaudojamus laukus galima eliminuoti projekcijos pagalba.
- Gavus rezultatą, patikrinti, ar eliminuoti pasikartojantys įrašai.

Pvz.: **MovieStar** (name, addr, gender, birthdate), **StarsIn** (title, year, starName)

Rasti filmus, kur aktorių vidutinis amžius filmo statymo metu buvo <= 40 metų.

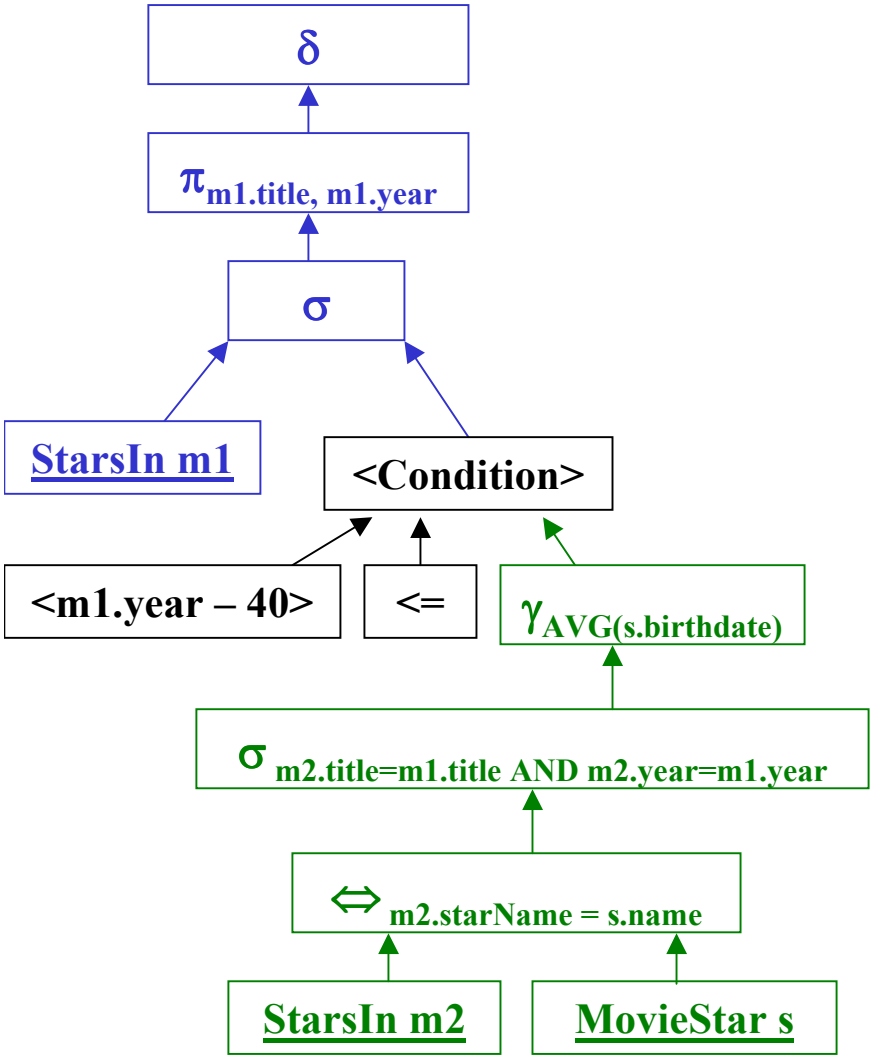
```
SELECT DISTINCT m1.title, m1.year FROM StarsIn m1 WHERE m1.year – 40 <=
(SELECT AVG(birthdate) FROM StarsIn m2, MovieStar s
WHERE m2.starName = s.name AND m1.title = m2.title AND m1.year = m2.year)
```

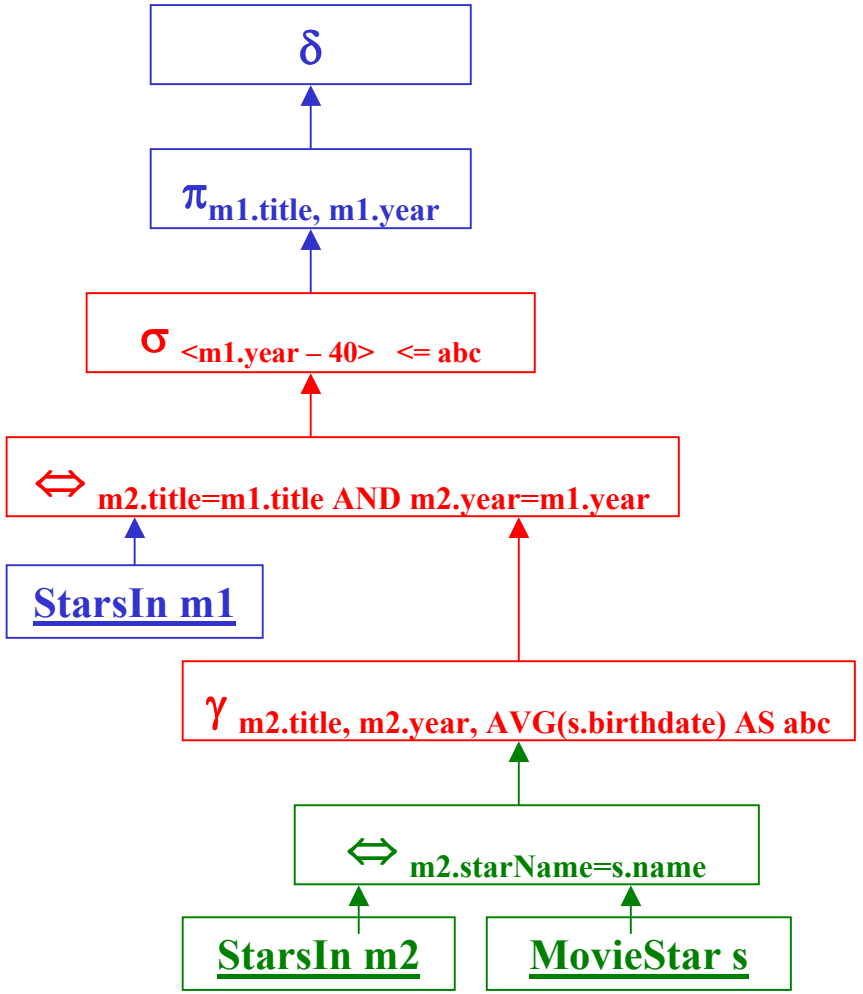
Pvz.: Kai <Condition> subužklausa yra susijusi su išorine užklausa (correlated):

SELECT DISTINCT m1.title, m1.year FROM StarsIn m1 WHERE m1.year – 40 <=

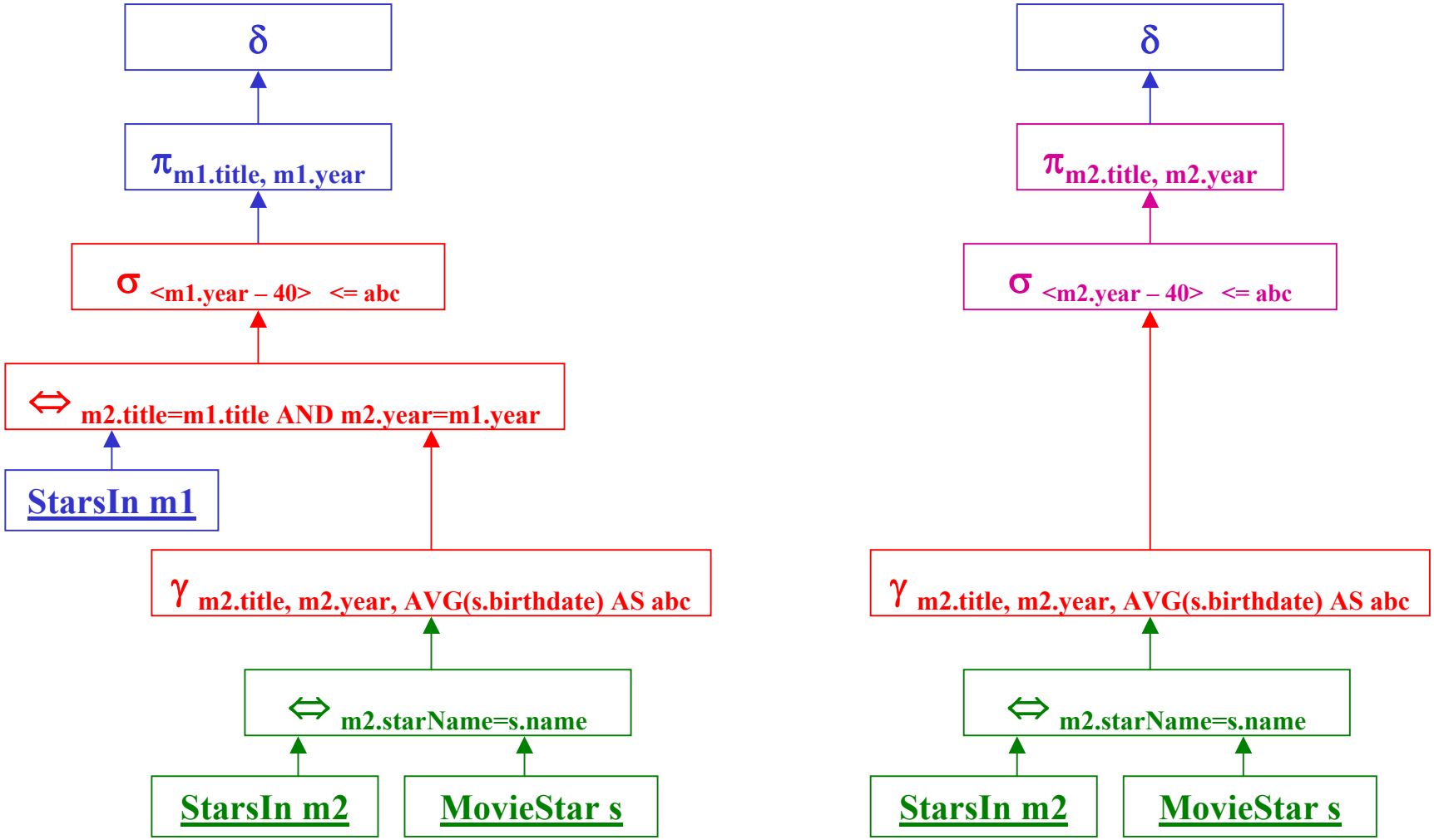
(SELECT AVG(birthdate) FROM StarsIn m2, MovieStar s

WHERE m2.starName = s.name AND m1.title = m2.title AND m1.year = m2.year)





Pvz.: Kai <Condition> subužklausa yra susijusi su išorine užklausa (correlated):



6.4 Loginio medžio optimizavimas (improving)

Pagrindinės algebrinio medžio optimizavimo taisyklės:

- Selection operatorius σ nuleidžiamas į kiek galima žemesnį medžio lygį;
- Projekcijos operatorius π nuleidžiamas į kiek galima žemesnį medžio lygį. Pridedamos papildomos projekcijos;
- Duplicate elimination δ operatorius kai kur galima eliminuoti, arba perstumti į kitą poziciją;
- Selection ir sandaugos operatorių kombinacijas $\sigma(R \times S)$ pakeisti ryšio operatoriumi $\Leftrightarrow$ (join).
- Pritaikyti operaciju komutatyvumo ir asociatyvumo savybes:

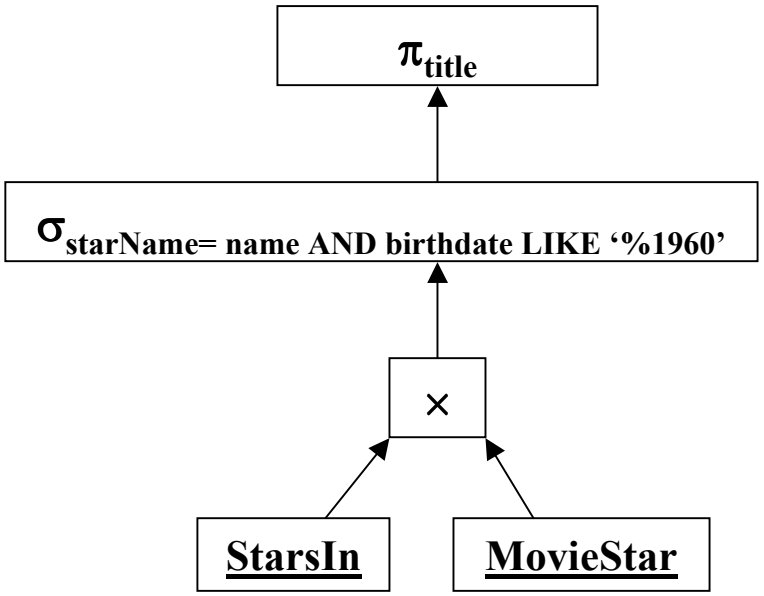
Mazgus, kurie turi tą patį komutatyvų-asociatyvų operatorių, galima apjungti į vieną mazgą su daug vaikų.

Dažniausiai pasitaikančios komutatyvios-asociatyvios operacijos yra paprastas ryšys (natural join) sąjunga (union) ir sankirta (intersection).

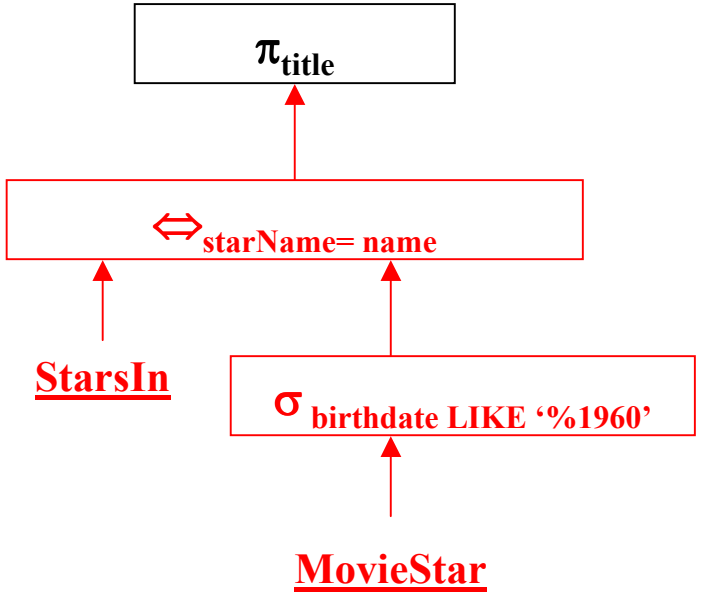
Pvz.: **MovieStar** (name, addr, gender, birthdate), **StarsIn** (title, year, starName)

Rasti filmus, kur aktoriai gimę 1960.

SELECT title **FROM** StarsIn, MovieStar **WHERE** starName = name **AND** birthdate **LIKE** '%1960'



Pradinis planas



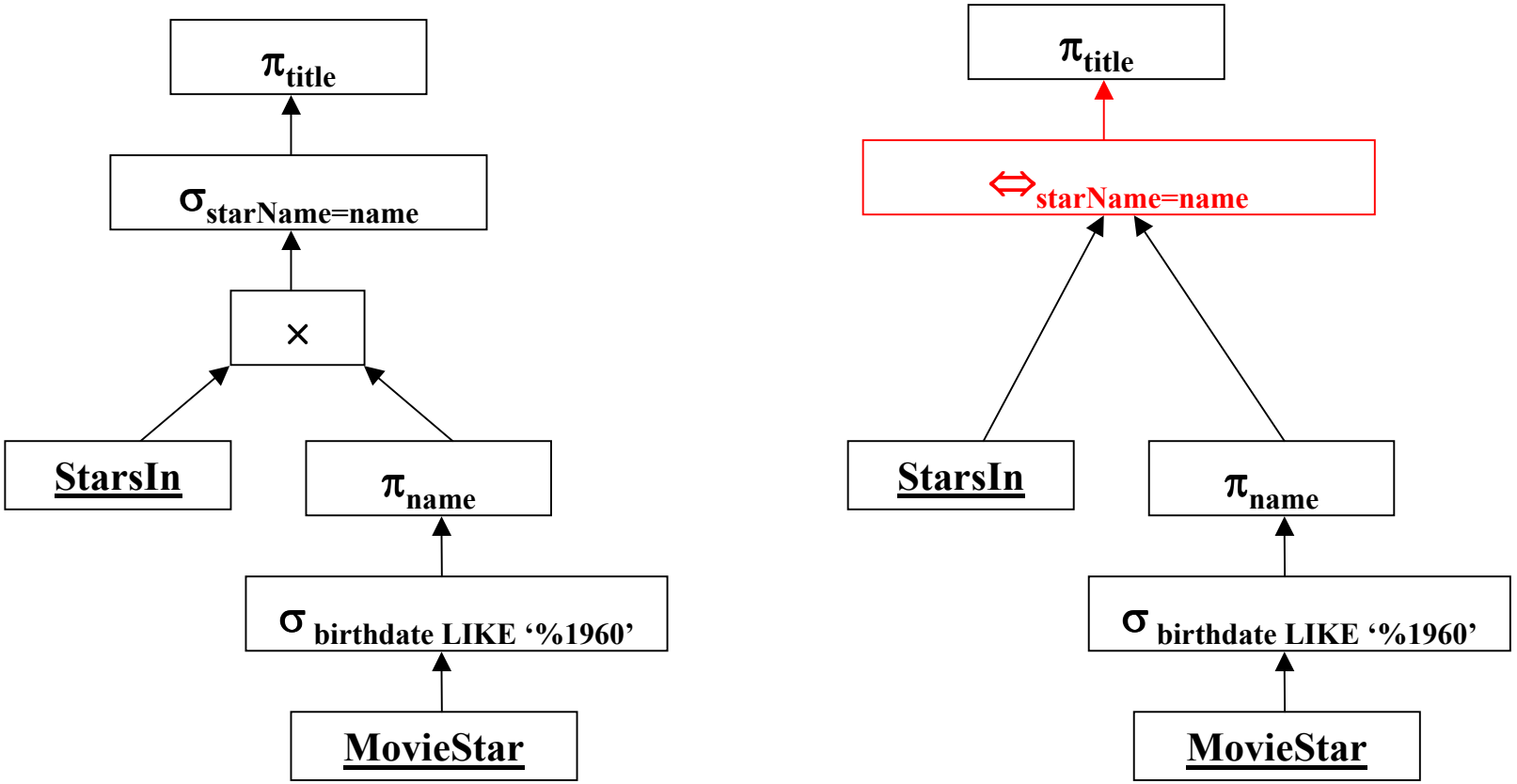
Optimizuotas planas

Pvz.: MovieStar (name, addr, gender, birthdate), StarsIn (title, year, starName)

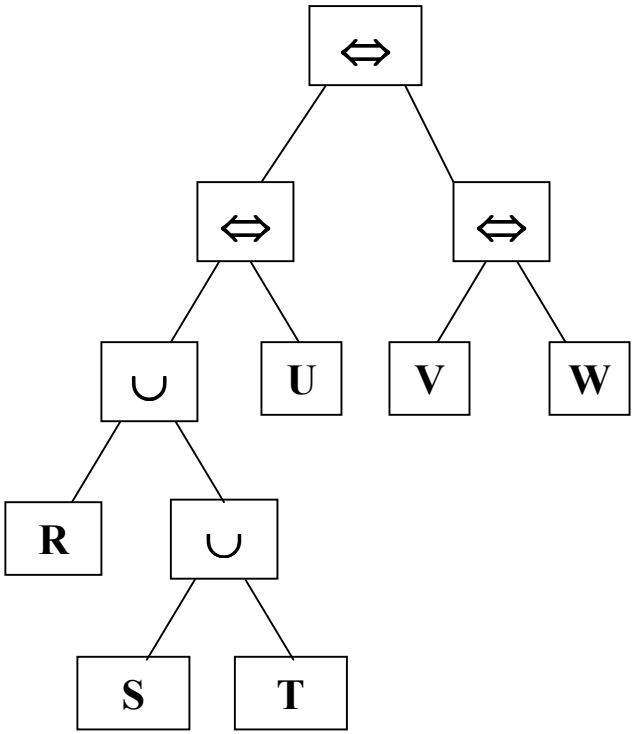
Rasti filmus, kur aktoriai gimę 1960.

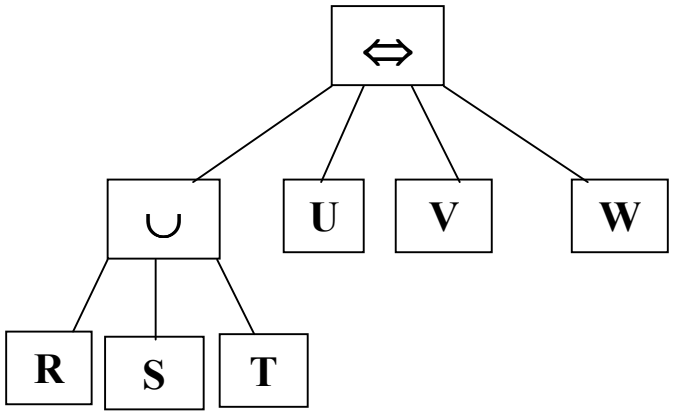
SELECT title FROM StarsIn WHERE starName IN

(SELECT name FROM MovieStar WHERE birthdate LIKE '%1960')



Pvz.: **Komutatyvių/asociatyvių operatorių grupavimas**





FIZINIS UŽKLAUSOS PLANAS

7. Fizinio užklausos plano operatoriai

Fizinis operatorius (physical operator) – yra loginio plano algebrinį operatorių realizuojantis algoritmas. Taip pat fizinis operatorius gali būti su algebriniu operatoriumi nesusijęs veiksmas, kaip pvz. lentelės skanavimas iš antrinės atminties (secondary memory), ir jos įrašymas į pagrindinę atmintį (main memory).

7.1 Lentelės skanavimo (Table Scanning) fiziniai operatoriai

Lentelės skanavimas – yra jos turinio nuskaitymas iš antrinės atminties ir įrašymas į pagrindinę atmintį. Reikalingas veiksmas atliekant užklausas, pvz. sudarant dviejų lentelių sąjungą (union).

Yra du pagrindiniai įrašų radimo (locating) būdai:

7.1.1 Lentelės skanavimas (table-scan) – lentelės įrašai saugomi antrinėje atmintyje blokuose, duomenys į pagrindinę atmintį yra nuskaitomi po vieną bloką. Sistema turi turėti info apie visus blokus.

7.1.2 Indekso skanavimas (index-scan) – kai lentelės duomenys yra pasiekiami skanuojant lentelės ideksą, pvz.: reto indekso (sparse index) pagalba galima lengvai rasti ir nuskaityti lentelės blokus su įrašais. Šis metodas ypač efektyvus, kai reik nuskanuoti įrašus, atitinkančius pateiktas sąlygas, nors ir tiksliai nežinome kuriuose būtent blokuose yra reikalingi duomenys.

7.1.3 Rūšiavimas skanavimo metu (sort-scan)

Nemažai algebrinių operacijų yra reikalingi surūšiuoti argumentai.

Fizinis operatorius **sort-scan** kaip parametrus gauna lentelę R ir laukus, pagal kuriuos turi būti rūšiuojama, o grąžina surūšiuotą R.

Yra keli **sort-scan** įdiegimo/veikimo būdai:

- Jei lentelės ***R*** laukui ***a*** yra sudarytas indekso B-medis, arba ***R*** yra saugoma kaip index-sequential failas, išrūšiuotas pagal lauką ***a,*** tuomet skanuojant indeksą (index-scan) galima iš karto gauti surūšiuotus įrašus.
- Jei lentelė ***R*** yra maža ir telpa pagrindinėje atmintyje (main memory), visi įrašai table-scan arba index-scan pagalba įrašomi pagrindinėje atmintyje, yra nemažai efektyvių rūšiavimo pagrindinėje atmintyje algoritmų.
- Jei lentelė ***R*** netelpa pagrindinėje atmintyje, galima pasinaudoti multiway-merging'u (2.3.3 skyrius). Tačiau surūšiuota ***R*** nerašoma atgal į diską, o sudaromas vienas surūšiuotas **R** blokas, kai tik prireikia ***R*** įrašų.

8. Fizinių operatorių veikimo įvertinimas

Fizinio operatoriaus parametrai (lentelės) yra saugomi diske, rezultatas saugomas pagrindinėje atmintyje, ir išvedamas į išvedimo įrenginį (output).

Fizinio operatoriaus "kainą" vertinsime **disko I/O operacijų skaičiumi**, nes pasiekti įrašams diske reikia žymiai daugiau laiko, negu darbui su duomenimis, jau esančiais pagrindinėje atmintyje.

Fizinių operatorių įvertinimo **parametrai**:

- **M** – pagrindinėje atmintyje telpančių buferių skaičius, kurį gali naudoti operatorius. Šie buferiai yra naudojami laikyti operatoriaus įeinančius parametrus ir tarpinius rezultatus. M gali būti visa atmintis, arba tik dalis, jei tuo pat metu dalį pagrindinės atminties naudoja kiti procesai.
- **B(R)** – blokų skaičius diske, reikalingas saugoti lentelę ***R***. Lentelės duomenys paprastai yra nuskaitomi po vieną bloką. Tada galima įvertinti, kad **I/O operacijų skaičius bus B**.
- **T(R)** – lentelės ***R*** įrašų skaičius. Jei lentelės įrašai yra paskirstyti tarp kitos lentelės įrašų, lentelės duomenys bus nuskaitomi ne po bloką, o po vieną įrašą. Tada galima įvertinti, kad **I/O operacijų skaičius bus T**.
- **V(R,[a1,...an])** – skirtingų įrašų pagal laukus a1,...,an skaičius, t.y. $\delta(\pi_{a1,..,an}(R))$.

8.1 Scan-operatoriu ivertinimas

I/O-operaciju skaicius	**R klasterizuota** (saugoma blokais)	**R neklasterizuota** (irašai pasiskirste tarp kitos lenteles irašu)
table-scan	**B**	**T**
sort-scan Jei lentele ***R* tepla pagrindineje atmintyje**, nuskaityti visa R i pagrindine atminti, ir pagrindineje atmintyje atlikti rušiavima.	**B**	**T**
sort-scan Jei lentele ***R* netepla pagrindineje atmintyje**, naudojamas multiway-merging'as (atliekamas 2 - faziu skanavimas).	**3B** (B - operaciju nuskaityti R i pirm. atminti, B - irašyti surušiuotus blokus i sublist'us diske, B – nuskaityti blokus iš sublist'u suliejimui)	**T+2B** (T - operaciju nuskaityti R i pirm. atminti, B - irašyti surušiuotus blokus i sublist'us i diska, B - nuskaityti sublist'us, jei sublist'ai saugomi blokais)
index-scan apima indekso skanavima ir R skanavima. taciau ***R*** skanavimas užima žymiai daugiau negu indekso skanavimas, todel iverti lemia ***R*** skanavimas.	**B**	**T**

9. Iteratoriai (iterators)

Daugelis fiziniu operatoriu gali buti aprašyti kaip iteratoriai.

Iteratorius – triju funkciju grupe, kuri per iteracija gražina po viena iraša:

Open – inicializuoja nuorodas i duomenu strukturas (lenteles), su kuriomis bus dirbama, pradeda irašo gavimo pracesa, bet irašo negražina.

GetNext – gražina sekanti iraša, arba pranešima, kad daugiau irašu nera.

Close – baigia iteracini irašu skaitymo procesa, t.y. Close iškvieciamas kiekvienam operatoriaus argumentui (skanuojamai lentelei).

***Pvz.:* Table-scan operatorius**. Tegu R – klasterizuota lentele (saugoma blokuose).

Open (R) { **b**: = pirmas R blokas ; **t** := pirmas R bloko įrašas ; **Found** := TRUE ; } *// b ir t – nuorodos*	***GetNext(R)*** { IF (**t** yra už paskutinio bloko **b** įrašo) { **b** := next block ; IF (nėra kito bloko){ **Found** := FALSE ; RETURN ; } ELSE **t** := pirmas bloko **b** įrašas ; **oldt** := **t** ; **t** := next record of **b** ; RETURN **oldt** ; } } *// grąžina vieną įrašą*	***Close(R)***{}

9. Iteratoriai (iterators) - tesinys

Pvz.: **Sort-scan operatorius**. Tegu R – klasterizuota lentele, netelpanti pagrindineje atmintyje.

Šiai lentelei reikia taikyti dvieju skanavimo faziu **multiway merge-sort** algoritma.

Rezultatas gaunamas tik nuskanavus visus R irašus.

Šiuo atveju:

Open() turi atlikti tokius veiksmus:

- Nuskaityti R irašus i grupes, kuriu dydis lygus pagrindines atminties dydžiui, jas surušiuoti, ir irašyti i diska.
- Inicializuoti duomenu struktura antrajai suliejimo (merge) fazei. Iš kiekvienos grupes užkrauti po pirmaji bloka i pirmine atminti.

GetNex() atlieka suliejima (merge). Kai baigiasi vienas iš bloku, vietoj jo i pirmine atminti užkraunamas sekantis blokas.

10. Algoritmu klasifikacija

Transformuojant logini užklausos plana i fizini, svarbus žingsnis yra kiekvenam operatoriui parinkti optimalu algoritma.

Algoritmai skirstomi i 3 tipus:

- **Sorting-based** metodai
- **Hash-based** metodai
- **Index-based** metodai

Šie algoritmai skirtstomi i **3 sudetingumo klases**:

- **One-pass** – duomenys iš disko nuskaitomi viena karta. Dažniausiai naudojami, kai bent vienas iš operatoriaus argumentu (lenteliu) pilnai telpa pirmineje atmintyje;
- **Two-pass** - duomenys iš disko nuskaitomi du kartus. Dažniausiai naudojami, kai operatoriaus argumentai (lenteles) netelpa pirmineje atmintyje. Pvz multiway merge-sort algoritmas.
- **Three, more-passes** – kai dirbama su labai dideliais duomenu kiekiais. Tai dažniausiai yra rekursyvus two-pass algoritmo panaudojimas.

11 One-pass algoritmai

Operacijos, pagal tai, kokie one-pass algoritmai jas realizuoja, skirstomos i 3 grupes:

- **Tuple-at-a-time, unarines operacijos** . Šioms operacijoms (selection σ ir projection π) nereikia visus duomenis užkrauti i pirmine atminti iškarto. Duomenys yra nuskaitomi po viena bloka.
- **Full-relation, unarines operacijos** . Šioms operacijoms (grouping γ ir distinct δ) reikia visus duomenis ar didele ju dali užkrauti i pirmine atminti iškarto. Todel šie algoritmai dirba su dydžio <= M lentelemis.
- **Full-relation, binarines operacijos** . Šioms operacijoms (union $\cup$, intersection $\cap$, difference -, join $\Leftrightarrow$, product $\times$) reikia, kad bent vieno iš argumentu dydis butu <= M.

11.1 One-pass algoritmai Tuple-at-a-time unarinems operacijoms

Tuple-at-a-time operacijoms duomenys i pirmine atminti nuskaitomi po viena bloka, atliekama reikalinga unarine operacija (selection **σ(R)** ar projection **π(R)**), ir rezultatas paduodamas i išvedimo buferi (output buffer).

Reikalavimas: **M >= 1** ,

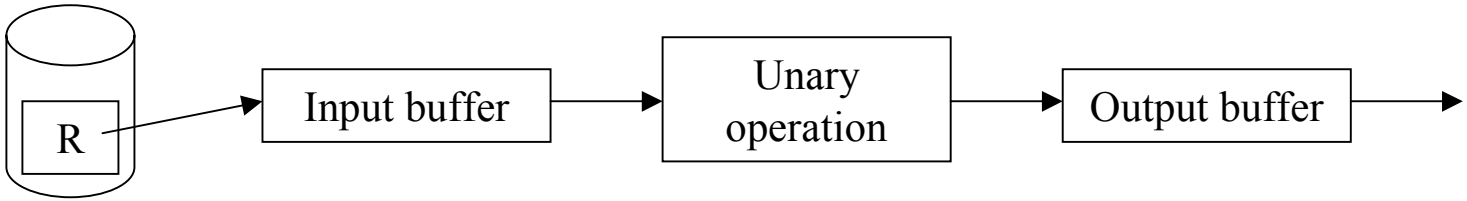
t.y. kad i atminti tilptu bent vienas blokas, nepriklausomai nuo bendro bloku skaciaus B.

I/O operaciju skaicius naudojant table-scan arba index-scan:

Jei R klasterizuota: **B**

Jei R neklasterizuota: **T**

Jei selection **σ(R)** operacija atliekama pagal lauka, kuris turi indeksa, naudojant index-scan algoritma galima žymiai pagreitinti rezultato radima.



11.2 One-pass algoritmai *Full-relation Unarinems operacijoms*

Full-relation unarinems operacijoms reikia visus duomenis ar didele ju dali užkrauti i pirmine atminti iškarto. Todel šie algoritmai dirba su dydžio <= M lentelemis.

Duplicate elimination δ

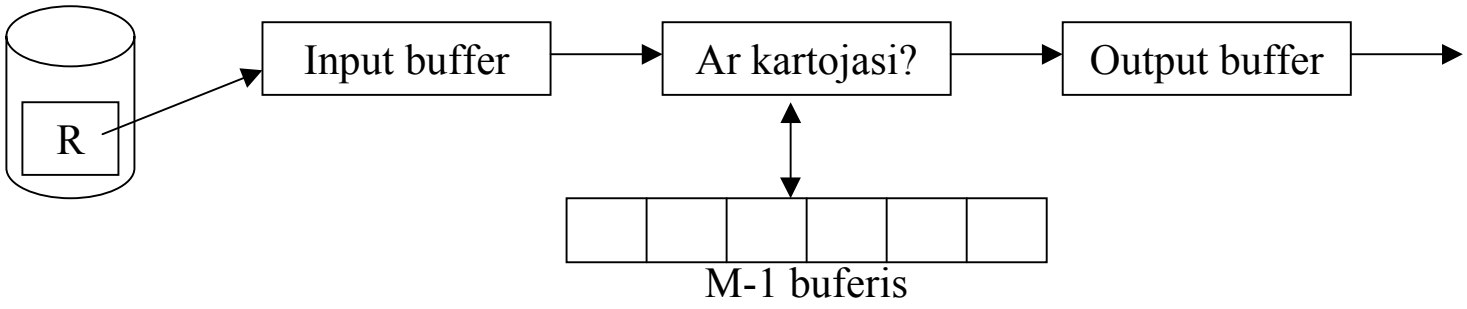
Duomenys nuskaitomi po bloka. Kiekvienam irašui atliekamas patikrinimas:

1. Jei irašas nesutampa su iki šiol matytais, jis kopijuojamas i output'a ir i buferi atmintyje, kur laikoma po viena kiekvieno jau matyto irašo kopija.

2. Jei irašas sutampa su iki šiol matytu, jis atmetamas.

Nauja iraša reik palyginti su kiekvienu jau matytu irašu buferyje. Siekiant pagreitinti šia operacija galima jau matytus irašus laikyti greitai paieškai ir iterpimui tinkama struktura, kaip hash lenteles arba subalansuoti binariniai paieškos medžiai. **I/O operaciju skaicius: B.**

Taigi 1 pirmines atminties blokas naudojamas laikyti nuskaityta ***R*** bloka, like (M-1) blokai naudojami laikyti po viena ***R*** irašu kopija: **B(δ(R)) <= M-1.**



11.2 One-pass algoritmai Full-relation Unarinems operacijoms - tesinys

Grupavimas $\gamma(R)$

Duomenys nuskaitomi po bloka. Kiekvienai grupei pirmineje atmintyje sukuriamas atitinkamas irašas, sudarytas iš lauku, pagal kuriuos grupuojama, ir lauku, kuriem skaiciuojamos agregacines išraiškos. Kiekvienam nuskaitomam irašui iš naujo apskaiciuojama agregacine išraiška:

- **MIN()** ir **MAX()** atveju yra atliekamas palyginimas tarp esancios toje grupeje reikšmes ir naujai nuskaityto lauko reikšmes.
- **COUNT()** atveju atitinkamoje grupeje reikšme padidinama 1-tu.
- **SUM()** atveju atitinkamoje grupeje prie esancios reikšmes pridedama naujai nuskaityto lauko reikšme.
- **AVG()** atveju reikia atmintyje laikyti COUNT ir SUM agregacines reikšmes. Nuskaicius visus ***R*** irašus, jas padalinti.

I/O operaciju skaicius: B.

Tarp M ir B nera akivaizdaus saryšio, bet dažniausiai buna M<B.

11.3 One-pass algoritmai Full-relation Binarinems operacijoms

Bag Union $R \cup_B S$

Randama paprastai: pradžioj nuskaitomi ir nukopijuojami i output'a visi ***R*** irašai, po to ***S*** irašai.

I/O operaciju skaicius: B(R)+B(S).

M ir B pakanka saryšio: M=1 nepriklausomai nuo B(R) ir B(S) dydžiu.

Kitos binarines operacijos (union $\cup$, intersection $\cap$, difference -, join $\Leftrightarrow$, product $\times$)

Šioms operacijoms reikia, kad bent vieno iš argumentu dydis butu <= M: **min(B(R),B(S))<=M.**

Vienas pirmines atminties buferis naudojamas naujai nuskaitytiems irašu blokams laikyti, like M-1 buferiai naudojami laikyti mažesne lentele.

Nukopijavus mažesni argumenta (pvz lentele ***S***) i pirmine atminti, reikia atlikti ivairius palyginimus su naujai nuskaitomais ***R*** irašais. Todel lentele ***S*** patogu laikyt greitai paieškai ir iterpimui patogia struktura, kaip hash lenteles arba subalansuoti binariniai paieškos medžiai.

I/O operaciju skaicius: B(R)+B(S).

One-pass algoritmu apibendrinimas

Operatorius	Reikalingas M	Diskinių I/O operacijų skaičius
σ, π	1<=M	B
δ, γ	B(R) <= M	B
$\cup$, $\cap$, –, $\Leftrightarrow$, $\times$	min(B(R), B(S)) <= M	B(R) + B(S)

12 Two-pass rušiavimo (sorting) algoritmai

Two-pass rušiavimo principas:

3 pagrindiniai žingsniai:

- Iš disko i pirmine atminti nuskaitoma lenteles ***R*** M bloku. Duomenys pirmineje atmintyje surušiuojami;
- Surušiuoti duomenys irašomi i M bloku diske. Šie duomenys sudaro lenteles ***R*** **surušiuotus sublist'us (sorted sublists)**.
- Sorted sublist'ai nuskaitomi iš disko, atliekant reikalingas operacijas ir suliejima (merge). Rezultatas kopijuojamas i output'a.

Naudojami, kai operatoriaus argumentai (lenteles) netelpa pirmineje atmintyje.

12.1 Dupliacate elimination δ(R) naudojant two-pass rušiavima

- Iš lenteles sudarome sorted sublist'us diske:

Pvz.: ***R*={2,5,2,1,2,2,4,5,4,3,4,2,1,5,2,1,3},** bloka sudaro 2 irašai, M=3.

***R*={{1,2,2,2,2,5},{2,3,4,4,4,5},{1,1,2,3,5}}**

- Iš kiekvieno sorted list'o nukopijuojame po pirmaji bloka i pirmine atminti. Iš kiekvieno bloko pirmineje atmintyje imame po pirma iraša surušiuota tvarka, t.y. 1-tas iš ***R1***.
- Pirmaji iraša, t.y. 1-ta kopijuojame i outputa, ir ištriname iš visu atmintyje esanciu bloku.
- Dabar pirmasis iraša yra 2-tas, ji kopijuojame i outputa, ir ištriname iš visu atmintyje esanciu bloku.

Sublist	Pirminėje atmintyje	Diske
R1	12	22, 25
R2	23	44, 45
R3	11	23, 5

Sublist	Pirminėje atmintyje	Diske
R1	2	22, 25
R2	23	44, 45
R3	23	5

Sublist	Pirminėje atmintyje	Diske
R1	5	
R2	3	44, 45
R3	3	5

Šie žingsniai kartojami, kol perskaitomi visi sublistai.

I/O operaciju skaicius: 3B

(B - operaciju nuskaityti ***R***, B - irašyti sublist'us i diska, B - nuskaityti sublist'us)

B ir M saryšis: $B<=M^2$

(t.y negali buti daugiau negu M sublistu, bei tarus, kad kiekvienas sublist'as turi po M bloku).

12.1 Grupavimas γ(R) naudojant two-pass rušiavima

- Iš lenteles ***R*** sudaromi sorted sublist'ai diske;
- Iš kiekvieno sorted list'o nukopijuojame po pirmaji bloka i pirmine atminti. Iš kiekvino bloko pirmineje atmintyje imame po pirma iraša surušiuota tvarka (mažiausia iraša) ***t***, kuris atstovaus grupe.
- Pagal ši iraša ***t*** randame visuose blokuose atitinkamus irašus ir suskaiciuojame agregacines išraiškas. Ištriname iš visu bloku iraša ***t*** atitinkancius irašus.
- Jei kuris buferis pasidaro tuščias, vietoj jo iš disko užkraunamas kitas irašu blokas iš atitinkamo sublist'o.
- Šie žingsniai kartojami, kol perskaitomi visi sublistai.

I/O operaciju skaicius: 3B

(B - operaciju nuskaityti ***R***, B - irašyti sublist'us i diska, B - nuskaityti sublist'us)

B ir M saryšis: $B<=M^2$

(t.y negali buti daugiau negu M sublistu, bei tarus, kad kiekvienas sublist'as turi po M bloku).

12.1 Union $R \cup S$ naudojant two-pass rušiavima

Bag $\cup_B$ sajungai rasti geriausia naudoti one-pass algoritma, kuris lenteles irašus nuskaito po viena karta, nepriklausomai nuo lenteles dydžio.

Set $\cup_S$ sajungai rasti one-pass algoritmas tinka tik kai bent vienas iš argumentu (lenteliu) pilnai telpa pirmineje atmintyje. Jei taip nera, naudojamas **two-pass algoritmas**:

- Lentelems ***R*** ir ***S*** atskirai sudarome sorted sublist'us diske;
- Iš kiekvieno ***R*** ir ***S*** sorted sublist'o nukopijuojame po pirmaji bloka i pirmine atminti. Iš kiekvieno bloko pirmineje atmintyje imame po pirma iraša surušiuota tvarka (mažiausia iraša) ***t***.
- Pirmaji iraša ***t*** kopijuojame i outputa, ir ištriname iš visu ***R*** ir ***S*** bloku.
- Jei kuris buferis pasidaro tuščias, vietoj jo iš disko užkraunamas kitas irašu blokas iš atitinkamo sublist'o.
- Šie žingsniai kartojami, kol perskaitomi visi sublistai.

I/O operaciju skaicius: 3 (B(R) + B(S))

(B - operaciju nuskaityti lentele, B - irašyti sublist'us i diska, B - nuskaityti sublist'us)

B ir M saryšis: $B(R)+B(S)<=M^2$

(t.y negali buti daugiau negu M sublistu, bei tarus, kad kiekvienas sublist'as turi po M bloku).

12.1 Sankirta $R \cap S$ ir skirtumas $R - S$, naudojant two-pass rušiavima

Bag ir **Set atvejais two-pass algoritmas yra toks pat:**

- Lentelems R ir S atskirai sudarome sorted sublist'us diske;
- Iš kiekvieno R ir S sorted list'o nukopijuojame po pirmaji bloka i pirmine atminti. Iš kiekvieno bloko pirmineje atmintyje imame po pirma iraša surušiuota tvarka (mažiausia iraša) ***t***.
- **$R \cap_S S$** (set)***:*** Pirmaji iraša ***t*** kopijuojame i outputa, jei jis yra ir R ir S blokuose, ir ištriname iš visu atmintyje esanciu R ir S bloku.
- **$R \cap_B S$** (bag)***:*** Pirmaji iraša ***t*** kopijuojame i outputa min(#R blokai, #S blokai) kartu, ir ištriname iš visu atmintyje esanciu R ir S bloku.
- **$R -_S S$** (set)***:*** Pirmaji iraša ***t*** kopijuojame i outputa, jei jis yra R blokuose, bet nera S blokuose, ir ištriname iš visu atmintyje esanciu R ir S bloku.
- **$R -_B S$** (bag)***:*** Pirmaji iraša ***t*** kopijuojame i outputa ((#R blokai) – (#S blokai)) kartu, ir ištriname iš visu atmintyje esanciu R ir S bloku.
- Jei kuris buferis pasidaro tuščias, vietoj jo iš disko užkraunamas kitas irašu blokas iš atitinkamo sublist'o.
- Šie žingsniai kartojami, kol perskaitomi visi sublistai.

I/O operaciju skaicius: 3B(R) + 3B(S)

(B - operaciju nuskaityti lentele, B - irašyti sublist'us i diska, B - nuskaityti sublist'us)

B ir M saryšis: $B(R)+B(S)<=M^2$

(t.y negali buti daugiau negu M sublistu, bei tarus, kad kiekvienas sublist'as turi po M bloku).

12.1 Sankirta $R \cap S$ ir skirtumas $R - S$, naudojant two-pass rušiavima

Pvz.: ***R*={2,5,2,1,2,2,4,5,4,3,4,2} *S*={1,5,2,1,3},** bloka sudaro 2 irašai, M=3.

Rasime $R -_B S$:

- Iš lenteles sudarome sorted sublist'us diske:

  ***R*={{1,2,2,2,2,5},{2,3,4,4,4,5}},S={1,1,2,3,5}**

- Iš kiekvieno sorted list'o nukopijuojame po pirmaji bloka i pirmine atminti. Iš kiekvieno bloko pirmineje atmintyje imame po pirma iraša surušiuota tvarka, t.y. 1-tas iš ***R1***.
- Kadangi 1-tas ***R*** blokuose pasikartoja 1 karta, o ***S*** blokuose 2 kartus (#***R*** - #***S***<0), šis irašas nekopijuojamas i outputa , ir ištriname iš visu atmintyje esanciu bloku.
- Dabar pirmasis iraša yra 2-tas. Ji kopijuojame i outputa (#***R*** - #***S*** = 5-1=4) kartus, ir ištriname iš visu bloku.
- Ir t.t.
- Rezultatas bus: 2, 2, 2, 2, 4, 4, 4, 5.

Sublist	Pirminėje atmintyje	Diske
R1	12	22, 25
R2	23	44, 45
S1	11	23, 5

Sublist	Pirminėje atmintyje	Diske
R1	2	22, 25
R2	23	44, 45
R3	23	5

Sublist	Pirminėje atmintyje	Diske
R1	5	
R2	3	44, 45
R3	3	5

12.1 Join R ⇔S naudojant two-pass rušiavima

R*(X,Y), *S*(X,Y)** – lenteles, jungiamos pagal lauka ***Y.

R ⇔S naudojamas **two-pass algoritmas**:

- Lentele ***R*** surušiuojama pagal lauka ***Y*** multiway merge sort algoritmu ir irašoma i diska;
- Lentele ***S*** surušiuojama pagal lauka ***Y*** multiway merge sort algoritmu ir irašoma i diska;
- Surušiuotu ***R*** ir ***S*** apjungimas:
- Iš surušiuoto R i pirmine atminti nuskaitomas pirmas blokas.
- Iš surušiuoto S i pirmine atminti nuskaitomas pirmas blokas.
- Iš šiu bloku pirmineje atmintyje randamas irašas su mažiausia lauko ***Y*** reikšme ***y***.
- Jei kitoje lenteleje pirmojo irašo laukas ***Y*** nelygus ***y***, visi lenteliu irašai, kur ***Y=y,*** išmetami iš atitinkamu bloku.
- Jei kitoje lenteleje pirmojo irašo laukas ***Y*** lygus ***y***, iš disko nuskaitomi visi ***R*** ir ***S*** irašai, kur ***Y=y*** , ir i output'a paduodami visi irašai sudaryti sujungus kiekviena ***R*** iraša, kur ***Y=y,*** su kiekvienu ***S*** irašu, kur ***Y=y***.
- **Salyga: atmintyje turi tilpti visi *R* ir *S* irašai, kur *Y=y*.**
- Jei kuris buferis pasidaro tuščias, vietoj jo iš disko užkraunamas kitas irašu blokas iš atitinkamos surušiuotos lenteles.
- Šie žingsniai kartojami, kol perskaitomos visos surušiuotos lenteles.

12.1 Join R ⇔S naudojant two-pass rušiavima - tesinys

I/O operaciju skaicius: 5 (B(R) + B(S))

(multiway merge sort algoritmui reik 4B: B - operaciju nuskaityti lenteles blokus i pirmine atminti rušiavimui, B - irašyti surušiuotus blokus i diska, B - nuskaityti surušiuotus blokus iš disko ir sujungti (merge), B – irašyti sujungtus surušiuotus blokus i diska,

5-tas B – surušiuotu lenteliu blokai nuskaitomi iš disko, kad daryti Join sujungima).

B ir M saryšis: B(R) <= M^2 , B(S) <= M^2 - multiway merge sort algoritmui

Atliekant Join sujungima dydis M nera viršijamas, todel bendram iverciui itakos neturi.

Jei egzistuoja ***Y=y*** reikšme, kad visi ***R*** ir ***S*** irašai, kur ***Y=y*** kartu netelpa pirmineje atmintyje,

galimi du atvejai:

- Jei vienos lenteles visi ***Y=y*** irašai telpa atmintyje, pav lenteles ***R***, juos surašome i M-1 pirmines atminties bloka. I likusi bloka užkraunam po viena ***S*** lenteles bloka. Tada irašams ***Y=y*** atliekamas one-pass join'as (nagrinetas 11.3).
- Jei ne vienos lenteles ***Y=y*** irašai netelpa pirmineje atmintyje, taikomi kiti algoritmai, kurie bus nagrinejami veliau.

12.2 Efektyvesnis Join R ⇔S radimo SORT-JOIN algoritmas

***R*(X,Y), *S*(X,Y)** – lenteles, jungiamos pagal lauka **Y**.

R ⇔S naudojamas **Sort-Join algoritmas**:

- Lentelems ***R*** ir ***S*** atskirai sudaromi sorted sublist'ai diske.
- **Pagrindine salyga, kad bendras sublist'u skaicius neviršytu M.**
- Iš kiekvieno surušiuoto sublist'o (***R*** ir ***S*** lenteliu) i pirmine atminti nuskaitoma po pirmaji bloka.
- Iš šiu bloku pirmineje atmintyje randamas irašas su mažiausia lauko ***Y*** reikšme ***y***.
- Abieju lenteliu blokuose randami ***Y=y*** irašai, i output'a paduodami surišti visi ***Y=y*** irašai iš ***R*** su irašais iš ***S***.
- Jei kuris buferis pasidaro tuščias, vietoj jo iš disko užkraunamas kitas irašu blokas iš atitinkamos surušiuotos lenteles.
- Šie žingsniai kartojami, kol perskaitomos visos surušiuotos lenteles.

I/O operaciju skaicius: 3 (B(R) + B(S))

(B - operaciju nuskaityti lenteles blokus i pirmine atminti rušiavimui, B - irašyti surušiuotus blokus sublist'us diske, B - nuskaityti surušiuotus blokus iš disko ir sujungti (join) paduoti i output'as).

B ir M saryšis: B(R) + B(S) <= M^2 - multiway merge sort algoritmui

13. Sort-based algoritmų apibendrinimas:

Operatorius	Reikalingas M	Diskinių I/O operacijų skaičius
δ, γ	$B <= M^2$	3B
$\cup$, $\cap$, –	$B(R)+B(S) <= M^2$	3 (B(R) + B(S))
$\Leftrightarrow$	$B(R) <= M^2$, $B(S) <= M^2$	5 (B(R) + B(S))
$\Leftrightarrow$	$B(R)+B(S) <= M^2$	3 (B(R) + B(S))

14 Two-pass hash algoritmai

Algoritmai naudojami, kai operatoriaus argumentai (lenteles) netelpa pirmineje atmintyje.

Irašai hash funkcijos pagalba suskirstomi i hash grupes:

Parenkamas hash parametras taip, kad irašams pritaikius hash funkcija visi ta pati hash parametra atitinkantys irašai bus vienoje grupeje.

Toliau dirbama ne su irašais, o su vienu arba dviem bucket'ais. Taigi operandu dydis sumažinamas nuo irašu skaiciaus (lenteleje) iki bucket'u skaiciaus.

Lenteles skaidymo hash pagalba rušiavimo principas:

Tegu ***h*** – hash funkcija, kurios argumentas yra lenteles irašas.

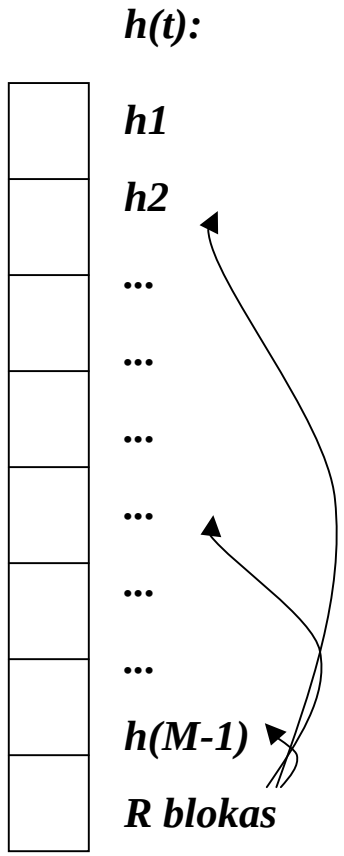
Lentele ***R*** suskaidysime i M-1 panašaus dydžio bucket'a.

Kiekviena iš M-1 buferio pirmineje atmintyje susiesim su bucket'u. Paskutiniame M-ajame buferyje bus saugomas nuskaitytas lenteles R blokas.

- Kiekvienam R irašui t paskaiciuojama hash funkcija ***h(t).*** Irašas t nukopijuojamas i ***h(t)*** reikšme atitinkanti buferi pirmineje atmintyje (salyga, kad irašas telpa buferyje).
- Jei kuris buferis užsipildo, jo turinys nukopijuojamas i diska, buferis ištuštinamas ir pradedamas pildyti iš naujo.
- Nuskaicius visus ***R*** irašus, pirmineje atmintyje esantys netušti buferiai su paskutiniais irašais kopijuojami i diska.

Lenteles skaldymo i buket'us algoritmas:

```
Initialize M-1 buckets using M-1 emty buffers;
FOR each block b of R DO {
        read block b into M-th buffer
        FOR each tuple t in b DO {
                if buffer for bucket h(t) has no room for t THEN {
                copy buffer to disk;
                initialize emty block in that buffer;
                }
        copy t to buffer for bucket h(t);
        }
}
FOR each bucked h DO {
        IF buffer for bucket h is not empty THEN
                write the buffer to disk;
}
```



14.1 Hash algoritmas Dupliacate elimination δ(R) operacijai

- Lentele ***R*** suskaidoma i M-1 bucket'a. Akivaizdu, kad dvi to paties irašo kopijos priklausys tam paciam bucket'ui;
- Kiekvienam bucket'ui R_i atskirai taikoma operacija δ. Šiai operacijai atlikti galima pasinaudoti one-pass algoritmu. Algoritmui reikia, kad kiekvienas bucketas R_i tilptu pirmineje atmintyje.
- Imama gautu rezultatu $\delta(R_i)$ sajunga;

Kiekviename bucket'e R_i yra apie B(R)/(M-1) bloku.

One-pass algoritmas veiks, jei: B(R)/(M-1) <=M

Kadangi M ~ M-1, **turime B ir M iverti: $B(R) <= M^2$**

Diskiniu I/O-operaciju skaicius: 3B(R)

(B - operaciju nuskaityti lenteles blokus i pirmine atminti hash suskaidymui, B - irašyti kiekvieno bucket'o blokus i diska, B - nuskaityti kiekvieno bucketo blokus iš disko one-pass δ algoritmui, eliminuojant pasikartojancius irašus).

14.2 Hash algoritmas Grupavimo γ(R) operacijai

- Lentele ***R*** suskaidoma i M-1 bucket'a. Kad vienos grupes irašai patektu i ta pati bucket'a, hash funkcija parenkama priklausanti nuo lauku, pagal kuriuos vyksta grupavimas;
- Kiekvienam bucket'ui R_i atskirai taikoma operacija **γ**. Šiai operacijai atlikti galima pasinaudoti one-pass algoritmu. Algoritmui nebutina, kad kiekvienas bucketas R_i tilptu pirmineje atmintyje, pakanka, kad atmintyje tilptu jau sugrupuoti bucket'o irašai (t.y. kiekviena grupe atstovauja po viena iraša).
- Imama gautu rezultatu $\gamma(R_i)$ sajunga;

Kiekviena bucket'a galima apdoroti atmintyje, jei: $B(R_i) <= M^2$

Kiekviename bucket'e R_i sugrupavus jo irašus yra apie $B(R_i) = B(R)/N$ bloku, kur **N vidutinis irašu vienoje grupeje skaicius atlikus grupavima**.

One-pass algoritmas veiks, jei: $B(R)/N <= M^2$

Turime B ir M iverti: $B(R) <= M^2 N$

Diskiniu I/O-operaciju skaicius: 3B(R)

(B - operaciju nuskaityti lenteles blokus i pirmine atminti hash suskaidymui, B - irašyti kiekvieno bucket'o blokus i diska, B - nuskaityti kiekvieno bucketo blokus iš disko one-pass grupavimo **γ** algoritmui).

14.3 Hash algoritmas Sajungai $R \cup S$, Sankirtai $R \cap S$ ir Skirtumui $R - S$

Abiem lentelem turi buti taikoma ta pati hash funkcija

- Lenteles R ir S suskaidomos i M-1 bucket'a : $R_1 \ldots R_{M-1}$ ir $S_1 \ldots S_{M-1}$;
- Kiekvienami porai R_i ir S_i taikoma atitinkama operacija sajunga $R_i \cup S_i$, sankirta $R_i \cap S_i$ arba skirtumas $R_i - S_i$. Šiai operacijai atlikti galima pasinaudoti one-pass algoritmu. Algoritmui reikia, kad kiekvienas bucketas R_i tilptu pirmineje atmintyje.
- Imama gautu rezultatu sajunga;

Kiekviename bucket'e R_i yra apie $\mathbf{B(R_i) = B(R)/(M-1)}$ bloku, S_i yra apie $\mathbf{B(S_i) = B(S)/(M-1)}$ bloku.

One-pass algoritmas veiks, jei**: min (B(R)/(M-1), B(R)/(M-1)) <=M**

Kadangi M ~ M-1, **turime B ir M iverti: min(B(R), B(S)) <= M^2**

Diskiniu **I/O-operaciju skaicius: 3 (B(R) + B(S))**

(B - operaciju nuskaityti vienos lenteles blokus i pirmine atminti hash suskaidymui, B - irašyti kiekvieno bucket'o blokus i diska, B - nuskaityti kiekvieno bucketo blokus iš disko one-pass algoritmui).

14.4 Hash algoritmas Join $R \Leftrightarrow S$ operacijai

R*(X,Y), *S*(X,Y)** – lenteles, jungiamos pagal lauka(-us) ***Y.

- Lenteles ***R*** ir ***S*** suskaidomos i M-1 bucket'a : $R_1 \ldots R_{M-1}$ ir $S_1 \ldots S_{M-1}$;
- Hash funkcija parenkama priklausanti nuo lauku ***Y***, pagal kuriuos vyksta ryšio sudarymas, kad ryšio sudaryme dalyvaujantys irašai patektu i atitinkamai R_i ir S_i bucket'us;
- Kiekvienai porai R_i ir S_i taikoma join operacija $\Leftrightarrow$. Šiai operacijai atlikti galima pasinaudoti one-pass algoritmu. Algoritmui reikia, kad kiekvienas bucketas R_i tilptu pirmineje atmintyje.
- Imama gautu rezultatu sajunga.

Kiekviename bucket'e R_i yra apie $\mathbf{B(R_i)} = B(R)/(M-1)$ bloku, S_i yra apie $\mathbf{B(S_i)} = B(S)/(M-1)$ bloku.

One-pass algoritmas veiks, jei: $\mathbf{\min (B(R)/(M-1), B(R)/(M-1)) <= M}$

Kadangi M ~ M-1, **turime B ir M iverti: $\min(B(R), B(S)) <= M^2$**

Diskiniu **I/O-operaciju skaicius: 3 (B(R) + B(S))**

(B - operaciju nuskaityti vienos lenteles blokus i pirmine atminti hash suskaidymui, B - irašyti kiekvieno bucket'o blokus i diska, B - nuskaityti kiekvieno bucketo blokus iš disko one-pass algoritmui).

13. Hash algoritmų apibendrinimas:

Operatorius	**Reikalingas M**	**Diskinių I/O operacijų skaičius**
δ, γ	**$B <= M^2$**	**3B**
$\cup$, $\cap$, **–**	**$\min(B(R), B(S)) <= M^2$**	**3 (B(R) + B(S))**
$\Leftrightarrow$	**$\min(B(R), B(S)) <= M^2$**	**3 (B(R) + B(S))**

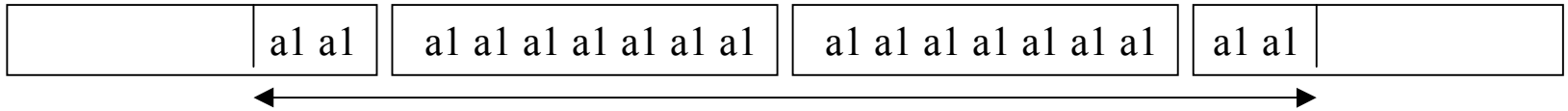
15 Index-based algoritmai

Šie algoritmai ypatingai naudingi select operacijoms.

Klasteriniai ir neklasteriniai indeksai:

Lentele yra klasterizuota, jei jos irašai yra išsideste blokais.

Indeksas vadinamas klasteriniu, jei jis yra tokio lauko indexas, kad irašai su fiksuotomis to lauko reikšmemis yra išsideste kaip galima mažesniame bloku skaiciuje, t.y. vienos reikšmes irašai išsideste tame paciame arba gretimuose blokuose.



15.1 Index-based algoritmas selection σ(R) operacijai

Kai lenteleje nera indekso one-pass algoritme selection **σ(R)** operacijai duomenys i pirmine atminti nuskaitomi po viena bloka, iš bloko atrenkami salyga tenkinantys irašai, ir rezultatas paduodamas i išvedimo buferi (output buffer). Reikalavimas: **M >= 1** , t.y. kad i atminti tilptu bent vienas blokas. I/O operaciju skaicius: jei R klasterizuota **B** , jei R neklasterizuota **T.**

Kai lenteleje laukas ***a*** turi indeksa, ir selection **σ(R)** operacija atliekama su salyga ***a=v,*** galima paieška taikyti ideksui (index-scan). Rezultate bus greitai gautos visos nuorodos i reikiamus irašus.

Jei lauko *a* indeksas klasterinis, tai I/O operaciju skaicius: B(R)/V(R,*a*)

Jei lauko *a* indeksas neklasterinis irašai gali buti skirtinguose blokuose, tai I/O operaciju skaicius: T(R)/V(R,*a*)

15.2 Index-based algoritmas join R⇔S operacijai

R(X,Y), S(X,Y) – lenteles, jungiamos pagal lauka ***Y***. Lentele ***S*** turi indeksa laukui ***Y***.

- Nuskaitomas kiekvienas ***R*** blokas.
- Kiekvienam ***R*** irašui reik rasti visus ryšyje dalyvaujancius irašus iš ***S***, pasinaudojant ***S*** lenteles indeksu laukui ***Y***.

Jei B klasterizuota, jos nuskaitymui reikia N1 = B(R) I/O-operaciju.

Jei B neklasterizuota, jos nuskaitymui reikia N1 = T(R) I/O-operaciju.

Tada kiekvienam R irašui ***t*** reikia nuskaityti vidutiniškai:

N2 = R(S)/V(S,Y) irašu iš S, jei indexas klasterinis;

N2 = T(S)/V(S,Y) irašu iš S, jei indexas neklasterinis;

Bendras **I/O-operaciju skaicius: N1 * N2.**

15.3 Sorted-Index algoritmas *join R⇔S* operacijai

R(X,Y), S(X,Y) – lenteles, jungiamos pagal lauka ***Y***. Lenteles ***R*** arba/ir ***S*** turi sorted-indeksa laukui ***Y***. Pvz, kai indekas yra B-medis.

Galima gauti iškarto surušiuotus duomenis:

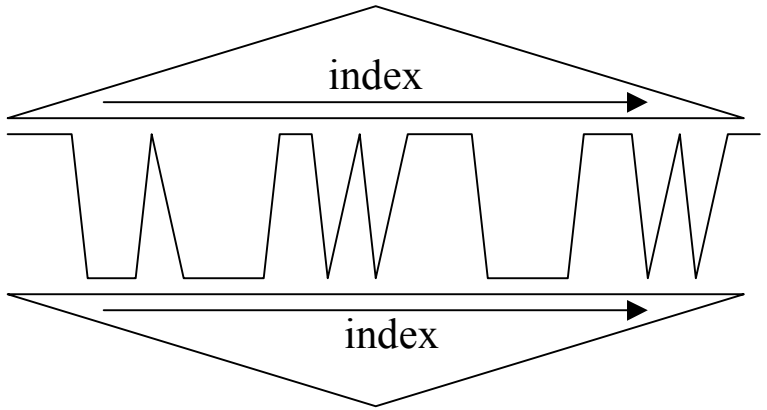
1. Jei lenteles ***R*** turi sorted-indeksa laukui ***Y***, lentele ***S*** neturi:

R⇔S surasti, galima atlikti standartini sort-join algoritma. Taciau remiantis sorted-indeksu nebereikia tarpiniuose žingsniuose atlikti duomenu rušiavimo pagal ***Y*** lauka.

2. Jei lenteles ***R*** ir ***S*** turi sorted-indeksa laukui ***Y***:

R⇔S surasti, uždavinys dar supaprasteja, galima atlikti tik paskutini sort-join algoritmo žingsni.

Šis metodas vadinamas **zig-zag-join**, kadangi tereikia šokineti tarp indexu, ieškant Y reikšmiu, kurias turi abi lenteles. Jei indeksas yra B-medis, reikia skanuoti B-medžio lapus iš kaires i dešine.



Be to lenteles ***R*** irašai su tomis ***Y*** reikšmemis, kuriu nera lenteleje ***S***, niekada nera nuskaitomi. Ir atvirkščiai. Del to sumažeja I/O operaciju skaicius

16 Multi-pass algoritmai

Kai duomenu yra daug ir netelpa pirmineje atmintyje, pagrindine ideja rekursyviai panaudoti two-pass algoritmus.

16.1 *Multipass* Sort-based algoritmai:

R- lentele. M – pirmines atminties buferiu skaicius.

Jei R netelpa i M bloku, suskirstyti lenteles R blokus i M grupiu: R_1 , ... , R_M .

Rekursyviai išrušiuoti kiekviena iš grupiu R_i .

1. Jei reikia atlikti rušiavimo operacija, sulieti gautus M surušiuotus sublistus.

2. Jei reik atlikti distinct operacija, i output'a iš sublistu nukopijuoti po viena kiekvieno irašo kopija

3. Jei reik atlikti grupavima, grupuoti irašus pagal nurodytus laukus.

Atliekant binarines operacijas (sajunga, sankirta, skirtumas) veiksmai analogiški, tik reikia kiekvienos iš lenteliu ***R*** ir ***S*** blokus suskirstyti i M surušiuotu sublistu.

FIZINIS UŽKLAUSOS PLANAS

17. Fizinio užklausos plano sudarymas

Turime optimalų loginį planą. Jį reik paversti fiziniu planu, kuris bus pateiktas užklausų vykdymo mechanizmui.

Tuo tikslu reikia mokėti įvertinti ir parinkti optimalius fizinius metodus kiekvienai operacijai.

17.1 Selection $\sigma_C(R)$ metodo parinkimas

Tegu lentelėje ***R*** laukai a ir b turi indeksus.

Turime <Condition> pavidalo: **<a = 10> AND < b < 20 >**

1. **Table-scan** algoritmo įvertinimas (nuskanuojama lentelė, atrenkami sąlygą tenkinantys įrašai):

(a) B(R) jei R – klasterizuota, (b) T(R) jei R - neklasterizuota

- Algoritmo **index-scan, kuris parenka įrašus su sąlyga a=10** ir patikrina, ar tenkinama likusi sąlygos dalis, kai laukas ***a*** turi indeksą, įvertinimas:

(a) B(R)/V(R,a) jei indexsas – klasterinis, (b) T(R)/V(R,a) jei indexas – neklasterinis

3. Algoritmo **index-scan, kuris parenka įrašus su sąlyga b<20** ir patikrina, ar tenkinama likusi sąlygos dalis, kai laukas ***b*** turi indeksą, įvertinimas:

- **B(R)/3 jei indexsas – klasterinis, (b) T(R)/3 jei indexas – neklasterinis**

čia daroma statistinė prielaida, kad paprastai nelygybė grąžina 1/3 įrašų.

17.2 Join R ⇔ S *metodo parinkimas*

Nagrinėtų Join algoritmų įvertinimas rėmėsi tuo, kad žinome kiek pirminės atminties blokų galima naudoti operacijai, ir koks V(R,a) skaičius. Tačiau realiai iš anksto šie parametrai nėra žinomi.

Algoritmo pasirinkimo kriterijai (analogiškai parenkama ir binarinėms operacijoms kaip sąjunga, unarinėms kaip grupavimas ir duplicate elimination operacijoms):

- **One-pass Join** – galima naudoti tik tikintis, kad pirminėje atmintyje yra pakankamai laisvų buferių.
- **Sort-Join** – naudoti, kai:
- Viena ar abi lentelės yra surūšiuots pagal lauką, pagal kurį daromas surišimas.
- Kai pagal tą patį lauką daromi keli surišimai: **(R(*a*,*b*) ⇔ S (*a*,*c*)) ⇔ T (a,d)**

kur surušiavus R ir S, rezultatas R ⇔ S irgi bus surušiuotas pagal ***a***, kuo galima pasinaudoti atliekant antraji sort-join surišima.

- **Index-join** - naudoti, kai sąryšis **R(*a*,*b*) ⇔ S (*a*,*c*)** darmas pagal lauka ***a***, kuris turi indexa.
- **Hash-join** – patartina naudoti, kai nera galimybes nauoti surušiuotas lenteles, ir nera indeksu, nes irašu nuskaitymai priklauso nuo mažesnio argumenti (lenteles), o ne nuo abieju argumentu.

18. Pipelining principas

Vykdant užklausos plana, operatoriai vykdomi iš eiles einancia tvarka. Ituityviai vieno operatoriaus rezultatas turetu buti irašomas i diska, ir saugomas, kol jo prireiks kitam operatoriui.

Efektyvesnis budas yra skaiciuojant užklausos plana, kelias operacijas vykdyti vienu metu. Vienos operacijos rezultate gauti irašai iškarto paduodami kitai vykdomai operacijai, šiuos tarpinius irašus neirašant i diska. Toks budas vadinamas Pipelining procesu.

Teigiama: Taip smarkiai sumažinamas I/O operaciju skaicius.

Neigiama: Tuo pat metu pirmine atmintimi naudojasi kelio operacijos.