Tutoriat #8

Evaluarea interogărilor. Optimizarea cererilor

Recapitulare – Tutoriat 6 (Algebra Relațională)

- O **expresie** în **algebra relațională** se obține prin aplicarea unor **operatori** (*PROJECT*, *SELECT*, *UNION*, *DIVISION*, etc.) asupra unor **relații**.
- Rezultatul evaluării unei expresii este o relație, derivată din relațiile menționate ca operanzi în cadrul expresiei.

Ce presupune optimizarea?

Atunci când scrieți o interogare SQL, aceasta este **evaluată** în mai multe etape:

1. se verifică **sintactic** și **semantic**
2. cererea se **descompune în operații elementare**, asupra cărora se stabilește o **ordine de execuție** (în urma acestui pas, se obține **planul de execuție** al cererii)
3. **execuția** efectivă: se efectuează operațiile elementare furnizate de planul de execuție pentru a obține rezultatul final

Am văzut că, pentru o cerere, se pot scrie mai mulți arbori algebrici, care conduc la același rezultat. Deci, nici **planul de execuție nu este unic**, deoarece există mai multe modalități de a ordona operațiile elementare implicate în scrierea unei cereri mai complexe. Înainte de a fi executate, cererile trec printr-o **etapă de optimizare**, al cărei scop este obținerea unui **plan de execuție echivalent, de cost minim**.

O modalitate de a determina ordinea de execuție care oferă costul minim este bazată pe **proprietățile algebrei relaționale**: se aplică **transformări algebrice**, care conduc la **expresii echivalente**, dar care se vor executa mai eficient.

Reguli de optimizare

Motivația principală a regulilor de optimizare este că se dorește o minimizare a costului operațiilor (CPU cost), ceea ce presupune că vrem să eliminăm cât mai multe date care nu ne sunt necesare, cât mai devreme posibil (în planul de execuție). De exemplu, operația de join este costisitoare (dpdv al puterii de procesare), deci am dori să nu includem în tabelele pe care aplicăm această operație coloane de care nu vom mai avea nevoie ulterior.

Câteva reguli de optimizare frecvent folosite sunt:

1. **Selecțiile** (operatorul *SELECT*) **se execută cât mai devreme posibil**, deoarece reduc substanțial dimensiunea relațiilor. Concret, nu dorim să efectuăm mai întâi un *JOIN*, și apoi un *SELECT*, ci invers. De asemenea, între *PROJECT* și *SELECT* aplicate succesiv, vom alege să executăm mai întâi *SELECT* (din aceleași considerente).

Transformările algebrice care se pot aplica pentru această optimizare:

$$\sigma_{cond1}(\sigma_{cond2}(R)) = \sigma_{cond1 \wedge cond2}(R) = \sigma_{cond2}(\sigma_{cond1}(R))$$

(compunerea selecțiilor)

$$\Pi_{A1,\dots,Am}(\sigma_{cond}(R)) = \sigma_{cond}(\Pi_{A1,\dots,Am}(R))$$

(comutarea selecției cu proiecția)

2. **Produsele carteziene se înlocuiesc cu join-uri**, de câte ori este posibil, deoarece un produs cartezian este mult mai costisitor și generează un rezultat foarte mare (de care nu avem nevoie).
3. Dacă sunt mai multe JOIN-uri, atunci **primul care se execută este cel mai restrictiv**, adică produce o relație cât mai mică.

Transformările algebrice care se pot aplica pentru această optimizare:

$$\text{JOIN}(\text{JOIN}(R_1, R_2), R_3) = \text{JOIN}(R_1, \text{JOIN}(R_2, R_3))$$

(asociativitatea join-urilor)

Exemplu: Fie relațiile: *STUDENT*(id_student#, nume, prenume), *CURS*(id_curs#, nume_curs, număr_credite, metodă_evaluare), *PARTICIPARE_CURS*(id_student#, id_curs#). Să se afle numele și prenumele studenților care participă la cursul “Tehnici Web”.

Soluție: Pentru a rezolva această interogare, vom avea nevoie de două JOIN-uri, iar deoarece acestea sunt asociative, putem alege pe care din ele să îl executăm mai întâi (și, indiferent de alegerea noastră, vom obține același rezultat).

Prima metodă: executăm mai întâi *JOIN*-ul dintre *STUDENT* și *PARTICIPARE_CURS*

```
Rezultat = Π_{nume,prenume}(JOIN(Π_{nume,prenume,id_curs}(JOIN(STUDENT,
PARTICIPARE_CURS)),CURS, nume_curs = ”Tehnici Web”))
```

- Rezultatul primului join efectuat (dintre *STUDENT* și *PARTICIPARE_CURS*) va avea un număr de înregistrări egal cu numărul de înregistrări din *PARTICIPARE_CURS*

A doua metodă: executăm mai întâi *JOIN*-ul dintre *CURS* și *PARTICIPARE_CURS*

```
Rezultat = Π_{nume,prenume}(JOIN(Π_{id_student}(JOIN(CURS, PARTICIPARE_CURS, nume_curs
= ”Tehnici Web”)), STUDENT))
```

- Deoarece se aplică o filtrare după o condiție, rezultatul primului join efectuat (dintre *CURS* și *PARTICIPARE_CURS*) va avea un număr de înregistrări substanțial mai mic (dintre toate înregistrările care apar în *PARTICIPARE_CURS*, se vor selecta doar id-urile acelor studenți înscriși la cursul de Tehnici Web).

4. **Proiecțiile se execută la început pentru a îndepărta atributele nefolositoare.** Dacă un atribut al unei relații nu va mai fi folosit ulterior, atunci acesta trebuie eliminat.

Transformările algebrice care se pot aplica pentru această optimizare: **comutarea proiecției cu join-ul:**

Dacă $A_1,...,A_m$ este o listă de atribute ce apar în schemele relaţionale R_1 şi R_2 şi dacă lista este formată din atribute aparţinând lui R_1 (notate prin $B_1,...,B_n$) şi din atribute aparţinând lui R_2 (notate prin $C_1,...,C_k$) atunci:

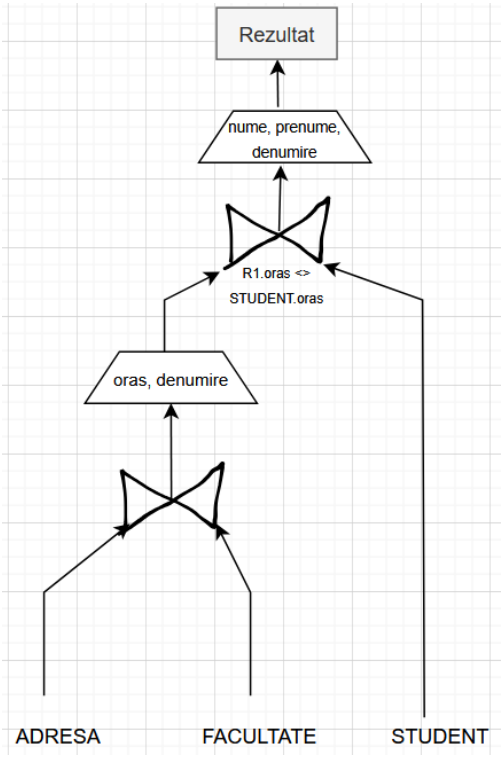
$$\Pi_{A1,...,Am}(\text{JOIN}(R_1,R_2,D)) = \Pi_{A1,...,Am}(\text{JOIN}(\Pi_{D,B1,...,Bn}(R_1), \Pi_{D,C1,...,Ck}(R_2),D)$$

Exemplu de optimizare

În tutoriatul anterior, am avut următorul exemplu:

Afișați numele, prenumele și facultatea pentru acei studenți care învață la o facultate aflată într-un alt oraș (diferit de domiciliul lor).

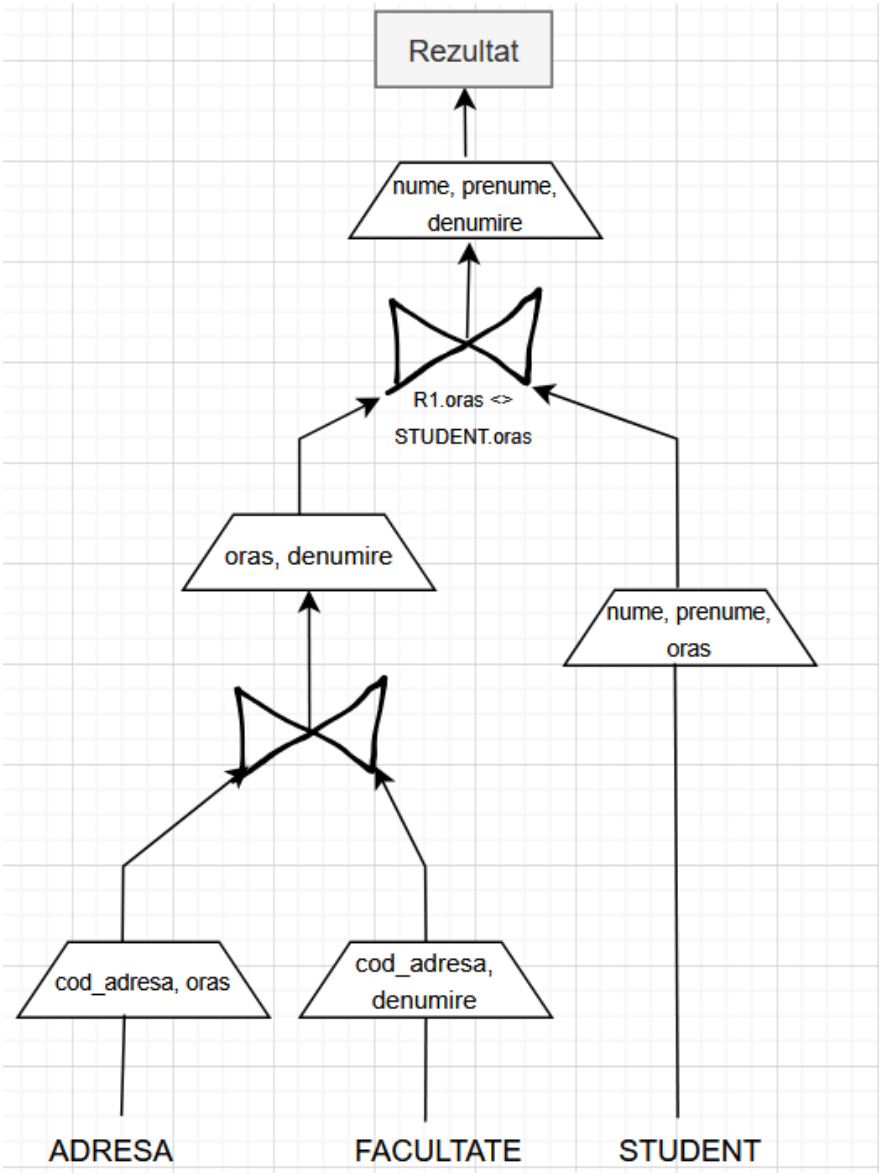
FACULTATE(cod_facultate#, denumire, telefon, email, cod_adresă)
ADRESĂ(cod_adresă#, stradă, număr, **oraș**, județ)
STUDENT(cod_student#, nume, prenume, **oraș**, cod_cămin, nr_cameră, dată_cazare)



În dreapta, avem arborele **neoptimizat**. Observăm că efectuăm o proiecție abia la final, deci în ultimul JOIN, vor intra toate atributele din tabelul STUDENT, deși nu avem nevoie decât de nume, prenume (pentru afișarea rezultatului) și oraș (pentru condiția de JOIN) !!! De asemenea, în JOIN-ul dintre ADRESĂ și FACULTATE, intră toate atributele celor două tabele, deși nu sunt necesare decât adresă.oraș, adresă.cod_adresă, facultate.cod_adresă, facultate.denumire !!!

Numărul mare de coloane ale tabelelor pe care se aplică JOIN-uri va conduce la un timp de execuție sporit. Pentru a remedia această situație, putem aplica regulie de optimizare derivate din proprietățile algebrei relaționale.

Vom aplica **regula 4 de optimizare**, astfel: înainte de join-ul dintre ADRESĂ și FACULTATE, vom efectua proiecții, pentru a păstra doar atributele pe le mai folosim ulterior în expresie. Similar, vom păstra din STUDENT doar atributele care ne interesează, aplicând o proiecție înainte de join. **Arborele optimizat** va arăta astfel (observați modificările):



În SQL, cererea asociată acestui arbore optimizat se poate scrie astfel:

```
SELECT st.nume, st.prenume, adr_fac.denumire
FROM
```

```
(SELECT a1.oras, f1.denumire FROM
   (SELECT cod_adresa, oras, FROM adresa) a1 JOIN
   (SELECT cod_adresa, denumire FROM facultate) f1 ON
a1.cod_adresa = f1.cod_adresa
)adr_fac,
(SELECT nume, prenume, oras FROM student) st
WHERE adr_fac.oras <> st.oras;
```