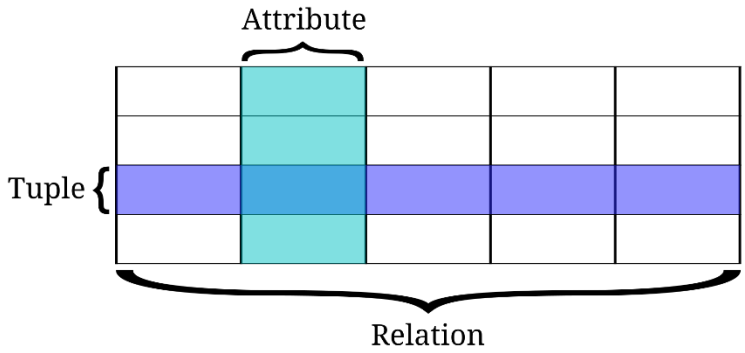
Tutoriat #6

Algebra relațională: expresii și arbori

Recap din tutoriatele trecute

O relație *R(Atribut₁, Atribut₂, ..., Atributₙ)* este o submulțime a produsului cartezian: $R \subset D_1 \times D_2 \times ... \times D_n$, unde D_i reprezintă mulțimea de valori a atributului i (adică a tuturor valorilor posibile ale acelui atribut – de exemplu, pentru atributul ”căsătorit”, avem mulțimea de valori posibile {„da”, „nu”}). Un element al relației este un tuplu (⇔ o înregistrare/ o linie din tabel). Atributele unei relații sunt coloanele din tabel.



De ce algebra relațională?

- Limbajele de interogare a bazelor de date relaționale (SQL) au nevoie de un fundament teoretic
- Din operațiile matematice care stau la baza algebrei relaționale, se pot deduce o serie de reguli de optimizare, aplicate de către executorul cererii SQL.

Operatorii algebrei relaționale sunt:

- **Operatori tradiționali pe mulțimi** (deoarece, așa cum am menționat mai sus, o relație poate fi privită ca o mulțime) – **UNION**, **INTERSECT**, **PRODUCT**, **DIFFERENCE**
- **Operatori relaționali speciali** – **PROJECT**, **SELECT**, **JOIN**, **DIVISION**

Ce este o expresie algebrică?

- un operator algebric aplicat unei relații
- atunci când aplicăm un operator unei relații, se obține tot o relație, deci putem compune (imbrica) expresiile în algebra relațională
  (în mod formal, spunem că relațiile sunt închise față de algebra relațională).

PROJECT

Notații: **PROJECT($R, A_1, A_2, ..., A_m$), $\Pi_{A1, ..., Am}(R)$**

- Submulțime pe ”verticală” (selectez anumite coloane/atribute)

SELECT

Notații: **SELECT(R, *condiție*), $\sigma_{condiție}(R)$**

- Submulțime pe ”orizontală” (selectez anumite tupluri/linii care satisfac **o** condiție)
- Operatorul SELECT va întoarce toate atributele acelei relații! Dacă mă interesează doar anumite coloane, va trebui să aplic un PROJECT peste acel SELECT:

`SELECT [coloana_1, coloana_2] FROM [nume_relatie] WHERE [conditie]` (în SQL) ⇔

$PROJECT(SELECT(nume\_relatie, conditie), coloana_1, coloana_2)$ (în algebra relațională).

UNION

- Reuniunea de la mulțimi (relația rezultat va conține toate tuplurile care se află fie în prima relație, fie în a doua)

Notații: **UNION(R, S), R ∪ S**

Ex: Sa se afiseze ID-urile studentilor care au fost cazati (la un moment dat) in camera 203 sau care sunt din orasul Botosani. (Exemplele pornesc de la diagrama bazei de date a căminelor, din tutoriatul anterior).

ISTORIC(cod_student#, dată_cazare#, dată_plecare, cod_cămin, nr_cameră)

STUDENT(cod_student#, nume, prenume, oraș, cod_cămin, nr_cameră, dată_cazare)

Vom prezenta trei modalități diferite de a scrie acest query:

1. Folosind limbajul **SQL**:

```
SELECT DISTINCT cod_student
FROM istoric
WHERE nr_camera = 203

UNION

SELECT cod_student
```

```
FROM student
WHERE oras like 'Botosani';
```

2. Sub forma unei expresii algebrice (folosind notațiile pentru operatori)

Obs: ne vom folosi de faptul că rezultatul unei expresii algebrice este o relație, deci putem scrie etapizat această cerere: mai întâi, vom selecta din tabelul ISTORIC acele linii care corespund unor studenți ce au locuit în camera 203 (vom aplica SELECT), iar apoi din acea mulțime de linii rezultată, vom selecta doar atributul cod_student, eliminând duplicatele (aplicăm PROJECT).

R1 = SELECT(ISTORIC, nr_camera = 203)

R2 = PROJECT(R1, cod_student)

R3 = SELECT (STUDENT, oraș = 'Botosani')

R4 = PROJECT(R3, cod_student)

Rezultat = UNION(R2, R4)

- Observați cum aplicăm un operator asupra unei relații obținute anterior!
- Bineînțeles, aceeași expresie poate fi scrisă și într-un singur rând, compunând relația finală din relațiile intermediare:

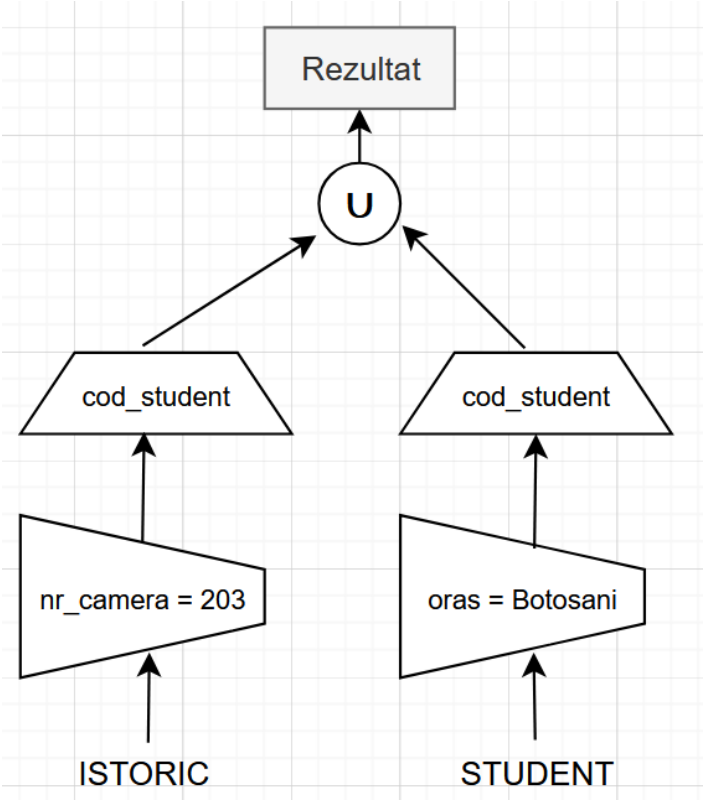
Rezultat = UNION(
PROJECT(SELECT(ISTORIC, nr_cameră = 203), cod_student),
PROJECT(SELECT(STUDENT, oras = 'Botosani'), cod_student)
)

- Sau, putem folosi pentru operatori, în loc de cuvinte, **notația cu simboluri**:

Rezultat = $\Pi_{cod\_student}\ (\sigma_{nr\_cameră\ =\ 203}\ (ISTORIC))\ \cup$
$\Pi_{cod\_student}\ (\sigma_{oras\ =\ 'Botosani'}\ (STUDENT))$

- **Observație**: Pentru început, se recomandă scrierea etapizată (ca să vă asigurați că nu faceți vreo greșeală). Dacă ulterior doriți să o scrieți ca și compunere de relații, se recomandă notația cu simboluri, pentru simplitate.

3. Sub forma unui arbore algebric (vezi notațiile pentru operatori!)
- Nodurile arborelui reprezintă operatorii
- Frunzele arbelui reprezintă tabelele incluse în interogare
- Construcția arborelui începe din stânga jos, rezultatul va fi regăsit în rădăcină.



DIFFERENCE

- Relația rezultat va conține toate tuplurile care se află în prima relație și nu se află în a doua relație

Notații: **DIFFERENCE(*R, S*), R - S**

Ex1: Aflați toate camerele din căminul P16 care nu au o capacitate mai mare de 3 persoane.

CAMERĂ(cod_cămin#, nr_cameră#, capacitate, cod_facultate, cod_tip)

1. Expresia algebrică: Vom determina mulțimea tutoror camerelor din căminul P16, din care vom ”scădea” camerele care au o capacitate mai mare de 3 persoane.

```
R1 = SELECT(CAMERA, cod_camin = ’P16’)
R2 = PROJECT(R1, nr_camera)
R3 = SELECT (CAMERA, capacitate > 3)
```

R4 = PROJECT(R3, nr_camera)

Rezultat = DIFFERENCE (R2, R4)

2. Implementări în SQL:

a) Operatorul MINUS:

```
SELECT nr_camera
FROM camera
WHERE cod_camin = 'P16'

MINUS

SELECT DISTINCT nr_camera
FROM camera
WHERE capacitate > 3;
```

b) Folosind NOT IN + subcerere necorelată (Atenție la posibile valori null în subcerere!!! Dacă lista de valori din subcerere conține cel puțin un **`null`**, atunci operatorul NOT IN se va evalua la **`false`**, iar rezultatul final va afișa **`no rows selected`**!! De aceea, folosim funcția NVL – dacă **`nr_camera`** este null, atunci NVL întoarce 0):

```
SELECT nr_camera
FROM camera
WHERE cod_camin = 'P16'
AND nr_camera NOT IN (SELECT NVL(nr_camera, 0)
                      FROM camera
                      WHERE capacitate > 3);
```

c) Folosind NOT EXISTS + subcerere corelată:

```
SELECT r1.nr_camera
FROM camera r1
WHERE cod_camin = 'P16'
AND NOT EXISTS (SELECT r2.nr_camera
    FROM camera r2
    WHERE capacitate > 3 AND r1.cod_camin = r2.cod_camin
AND r1.nr_camera = r2.nr_camera);
```

3. **Arborele algebric – try it yourselves! (notațiile pentru operatori)**

INTERSECT

Notații: **INTERSECT(*R, S*), R ∩ S**

Ex: Să se afișeze codul căminului și numărul camerei pentru acele camere în care stă în prezent o persoană numită Vicențiu și care nu sunt ale facultății de Geologie (cod_facultate = 'geol').

STUDENT(cod_student#, nume, prenume, oraș, cod_cămin, nr_cameră, dată_cazare)

CAMERĂ(cod_cămin#, nr_cameră#, capacitate, cod_facultate, cod_tip)

1. Expresia algebrică

Rezultat = $\Pi_{cod\_cămin,nr\_cameră}\ (\sigma_{prenume='Vicențiu'}\ (STUDENT)) \cap$
$\Pi_{cod\_cămin,nr\_cameră}\ (\sigma_{cod\_facultate<>'geol'}\ (CAMERĂ))$

2. Implementări SQL:

a) Folosind operatorul 'INTERSECT'

```
Select DISTINCT cod_camin, nr_camera
FROM student
WHERE prenume = 'Vicentiu'

INTERSECT

Select cod_camin, nr_camera
FROM camera
WHERE cod_facultate <> 'geol';
```

b) Folosind EXISTS + subcerere corelată

```
SELECT s.cod_camin, s.nr_camera
FROM student s
WHERE s.prenume = 'Vicentiu'
AND EXISTS (SELECT c.cod_camin, c.nr_camera
              FROM camera c
              WHERE c.cod_facultate <> 'geol'
              AND s.cod_camin = c.cod_camin
              AND s.nr_camera = c.nr_camera);
```

c) Folosind IN + subcerere necorelată (verific că o valoare din cererea externă se află printre cele întoarse de subcerere)

```
SELECT s.cod_camin, s.nr_camera
FROM student s
WHERE s.prenume = 'Vicentiu'
AND (s.cod_camin, s.nr_camera) IN
          (SELECT c.cod_camin, c.nr_camera
              FROM camera c
              WHERE c.cod_facultate <> 'geol');
```

3. **Arborele algebric – try it yourselves!**

PRODUCT

Notații: **PRODUCT(*R, S*), R x S**

- Echivalent cu CROSS JOIN in sintaxa SQL3
- Efectuează produsul cartezian al relațiilor (mulțimilor) R și S. Dacă R are m atribute, iar S are n atribute, atunci relația $\boldsymbol{R} \times \boldsymbol{S}$ va avea $m + n$ atribute.

R(A_1, A_2, ..., A_m), S(B_1, B_2, ..., B_n) ⇒ $R \times S$ (A_1, A_2, ..., A_m, B_1, B_2, ..., B_n)

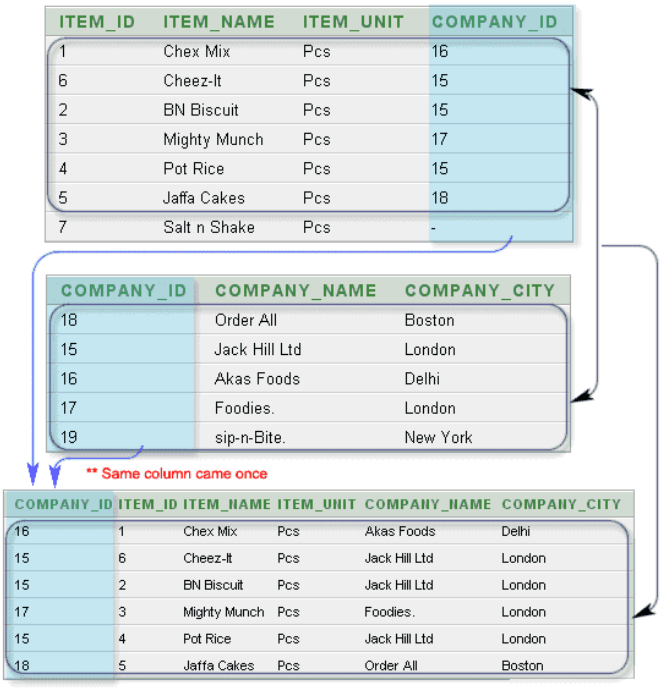
OPERATORUL JOIN

1. Natural Join

Notație: **JOIN(*R, S*)**

- Combin tupluri din relații diferite pe baza valorilor identice din coloane care au același nume în ambele relații.
- *A natural join is a type of equi-join where the **join** predicate arises implicitly by comparing all columns in both tables that have the same column-names in the joined tables. The resulting joined table contains only one column for each pair of equally named columns. In the case that no columns with the same names are found, the result is a cross join.*
- În Oracle SQL:

```
SELECT *
FROM tabela_1 t1
JOIN tabela_2 t2
USING(coloana_comuna);
```



ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

** Same column came once

COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
15	6	Cheez-It	Pcs	Jack Hill Ltd	London
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
17	3	Mighty Munch	Pcs	Foodies.	London
15	4	Pot Rice	Pcs	Jack Hill Ltd	London
18	5	Jaffa Cakes	Pcs	Order All	Boston

2. Θ-JOIN

Notație: **JOIN(*R, S, condiție*)**

- Combin tupluri din relații diferite, nu neapărat corelate, dacă valorile atributelor satisfac o anumită condiție enunțată explicit
- Echivalent cu o selecție peste produsul cartezian
- Caz particular de Θ-JOIN: **equi-join** (inner join în Oracle SQL)

Exp: 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)

i. Expresia algebrică:

R1 = JOIN(ADRESA, FACULTATE)

R2 = PROJECT(R1, oras, denumire)

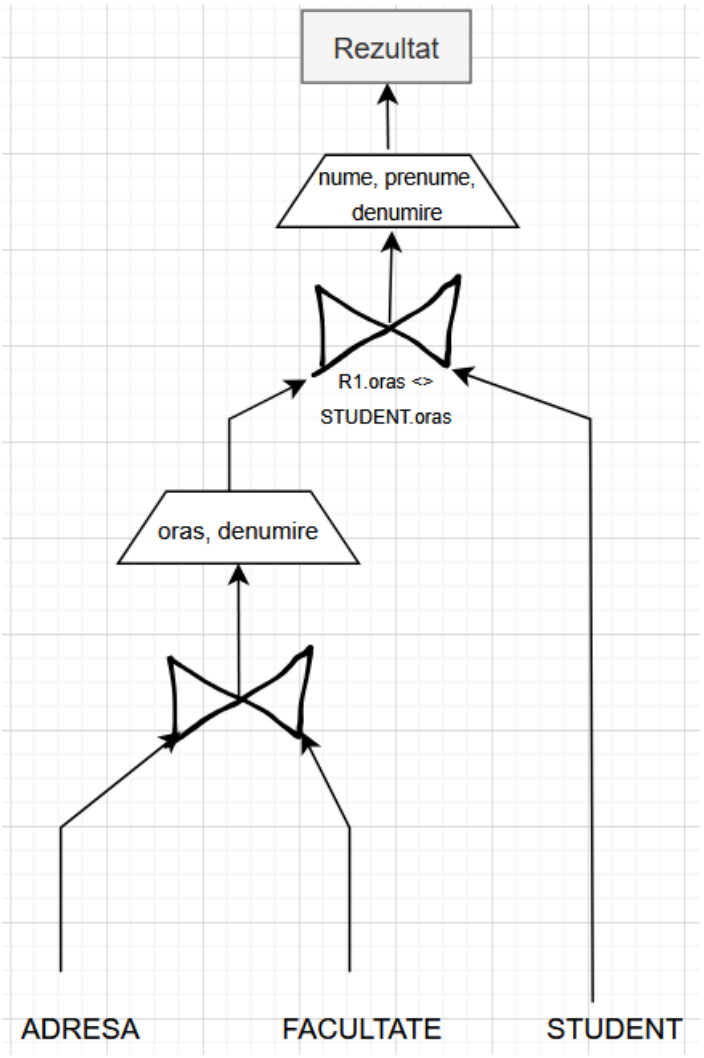
R3 = JOIN(R2, STUDENT, R1.oras <> STUDENT.oras)

Rezultat = PROJECT(R3, nume, prenume, denumire)

ii. SQL:

```
SELECT s.nume, s.prenume, f.denumire
FROM student s,
(SELECT a.oras, f.denumire FROM adresa a, facultate f
USING(cod_adresa))AS t1
WHERE s.oras <> t1.oras;
```

iii. Arborele algebric:



3. SEMI-JOIN

Notații: **SEMIJOIN(*R*, *S*), SEMIJOIN(*R*, *S*, condiţie)**

- Iau atributele unei singure relații care participă la join
- Combinație de project + natural join sau project + θ-join

Exp: Afișați informații despre data cazării și data eliberării camerei, împreună cu codul studentului care a stat în acea cameră, pentru cazările care au avut loc în anul 2021 în camerele Facultății de Drept (cod_facultate = ‘drept’).

ISTORIC(cod_student#, dată_cazare#, dată_eliberare, cod_cămin, nr_cameră)
CAMERĂ(cod_cămin#, nr_cameră#, capacitate, cod_facultate, cod_tip)

i. Expresie algebrică:

R1 = SELECT(CAMERA, cod_facultate = ’drept’)

R2 = SEMIJOIN(ISTORIC, R1)

R3 = SELECT(R2, to_char(data_cazare, ’yyyy’) = ’2021’)

Rezultat = PROJECT(R3, cod_student, data_cazare, data_eliberare)

ii. SQL:

```
SELECT cod_student, data_cazare, data_eliberare
FROM (SELECT i.* FROM istoric i JOIN
           (SELECT * FROM camera WHERE cod_facultate =
                ’drept’)) camere_drept USING(cod_camin,
nr_camera)) rez
WHERE(TO_CHAR(rez.data_cazare, ’yyyy’) = ’2021’);
```

iii. **Arborele algebric – try it yourselves!**

4. OUTER-JOIN
   - Compun două relații R și S (printr-un equi-join), adăugând și acele tupluri din R și din S care nu sunt conținute în compunere.
   - Adică, voi include tuplurile pentru care nu există aceleași valori în coloana/coloanele comune dintre cele două relații
   - Este de 3 tipuri:
     - **LEFT OUTER JOIN(R, S)**
     - **RIGHT OUTER JOIN(R, S)**
     - **FULL OUTER JOIN(R, S**)

OPERATORUL DIVISION

Notații: DIVISION(R, S), *R ÷ S*

- Selectez din R acele valori(linii) pentru care am un corespondent cu **fiecare tuplu din relația S**

Exp: Avem următoarele relații: *PARTICIPĂ(nume_student, nume_curs)*, *CURS_OBLIGATORIU(nume_curs)*

Nume_student	Nume_curs
Mihnea	Baze de Date
Mihnea	Programarea Algoritmilor
Simona	Arhitectura Calculatoarelor
Simona	Baze de Date
Simona	Machine Learning
Marius	Programarea Algoritmilor
Marius	Statistică

Nume_curs
Baze de Date
Programarea Algoritmilor

Vrem să selectăm numele acelor studenți care au participat la toate cursurile obligatorii.

REZULTAT = DIVISION(PARTICIPĂ, CURS_OBLIGATORIU)

Nume_student
Mihnea

În SQL, nu avem un operator special pentru division, ar trebui să îl simulăm într-un fel. Ne folosim de faptul că operația de ”division” este echivalentă din punct de vedere logic cu cuantificatorul universal (∀), și îl vom simula folosind următoarea echivalență:

$$\forall x P(x) \leftrightarrow \neg \exists x \neg P(x)$$

- Selectez numele studenților care au participat la toate ($\forall x$) cursurile obligatorii, adică selectez studenții pentru care nu există vreun curs obligatoriu la care să nu fi participat.

```
SELECT DISTINCT p.nume_student
FROM participa p
WHERE NOT EXISTS
    (SELECT 1 FROM curs_obligatoriu co
     WHERE NOT EXISTS
         (SELECT 1 FROM participa pp
          WHERE pp.nume_student = p.student AND co.nume_curs
= pp.nume_curs));

-- NOT EXISTS intoarce TRUE daca nu gaseste nicio linie care sa
satisfaca conditiile
```

```
--intorc numele studentilor, pentru care nu exista curs_obligatoriu la care sa nu existe participare pentru acel student
```

Anexă: notațiile pentru arbori