Algebra relationala

Algebra relationala a fost introdusa de *Edgar Frank Codd* si are la baza o multime de operatii care se aplica asupra unor relatii, rezultatul fiind tot o relatie.

Este un limbaj abstract cu ajutorul caruia se pot scrie cererile (se poate face o paralela cu pseudocodul care se transforma in cod interpretat de masina).

Pentru a scrie cereri folosind *algebra relationala* avem nevoie de simboluri specifice, operatii si implicit operatori.

Cu alte cuvinte, in algebra relationala folosim anumiti operatori specifici, reprezentati cu ajutorul unor simboluri si notatii, pe care ii aplicam unor relatii, rezultatul final fiind tot o relatie.

De exemplu: Fie *R* si *S* doua relatii asupra carora se aplica operatorul UNION (reuniune). Rezultatul reuniunii celor doua relatii este tot o relatie si se noteaza:

Rezultat = UNION(R,S)

Prin algebra relationala se pot construi cererile pas cu pas pana la obtinerea unui rezultat final, prin scrierea *expresiilor relationale.*

Operatorii algebrei relationale:

- operatori clasici pe multimi (**UNION**, **INTERSECT**, **PRODUCT**, **DIFFERENCE**);
- operatori relationali speciali (**PROJECT**, **SELECT**, **JOIN**, **DIVISION**).

Operatorul **PROJECT** – se utilizeaza acest operator atunci cand se doreste preluarea anumitor valori din baza de date (atributele/coloanele A1, A2, … , An), fara a fi necesara o conditie.

Notatie: ***PROJECT(R, A1, A2, … , An)***

Exemplu : Sa se obtina o lista care cuprinde numele si job-ul angajatilor.

Cerere SQL:

```
SELECT last_name, job_id
FROM employees;
```

Algebra relationala (expresia algebrica):

Rezultat = PROJECT(EMPLOYEES, nume, job);

Operatorul **SELECT** – se utilizeaza atunci cand preluam date pe baza unei conditii.

Notatie: ***SELECT(R, conditie)***

Exemplu : Sa se obtina toate informatiile despre angajatii care au jobul ‘IT_PROG’.

Cerere SQL:

```
SELECT *
FROM employees
WHERE job_id = 'IT_PROG';
```

Algebra relationala (expresia algebrica):

Rezultat = SELECT(EMPLOYEES, job_id = ‘IT_PROG’)

Operatorul **UNION** – reuniunea a doua relatii R si S (elementele comune si necomune afisate o singura data).

Notatie: ***UNION(R, S)***

Exemplu: Se cer codurile departamentelor al caror nume contine sirul “re” sau in care lucreaza angajati avand codul job-ului “SA_REP”.

Cerere SQL:

SELECT department_id
FROM departments
WHERE lower(department_name) like '%re%'

UNION

SELECT department_id
FROM employees
WHERE upper(job_id) like '%SA_REP%';

Algebra relationala (expresia algebrica):

R1 = SELECT(DEPARTMENTS, department_id LIKE '%re%')
R2 = PROJECT(R1, department_id)
R3 = SELECT(EMPLOYEES, job_id LIKE '%SA_REP%')
R4 = PROJECT(R3, department_id)
***Rezultat** = UNION(R2, R4)*

Operatorul **DIFFERENCE** – diferenta a doua relatii R si S (elementele care sunt in R si nu sunt in S).

Notatie: ***DIFFERENCE(R, S)***

Exemplu: Sa se obtina codurile departamentelor in care nu lucreaza nimeni.

Cerere SQL:

SELECT department_id
FROM departments -- department_id este cheie primara, deci avem o lista unica de departamente

MINUS -- din lista tuturor departamentelor dorim sa eliminam departamentele in care lucreaza angajati

SELECT department_id
FROM employees; -- department_id este cheie externa, deci sunt departamente in care lucreaza angajati

Algebra relationala (expresia algebrica):

R1 = PROJECT(DEPARTMENTS, department_id)
R2 = PROJECT(EMPLOYEES, department_id)
Rezultat = DIFFERENCE(R1, R2)

Simularea diferentei cu ajutorul operatorului **NOT IN**:

Sa se obtina codurile departamentelor in care nu lucreaza nimeni

Cerere SQL:

```
SELECT department_id
FROM departments
WHERE department_id NOT IN (SELECT NVL(department_id, 0)
                            FROM employees);
```

Operatorul **INTERSECT** – intersectia a doua relatii R si S (elementele comune celor doua relatii afisate o singura data).

Notatie: ***INTERSECT(R, S)***

Exemplu: Sa se afiseze angajatii (codul lor) care sunt manageri in cadrul departamentelor si in acelasi timp sunt si managerii altor angajati.

Cerere SQL:

```
SELECT manager_id
FROM departments -- managerii de departament

INTERSECT

SELECT manager_id
FROM employees; -- managerii angajatilor
```

Algebra relationala (expresia algebrica):

R1 = PROJECT(DEPARTMENTS, manager_id)
R2 = PROJECT(EMPLOYEES, manager_id)
Rezultat = INTERSECT(R1, R2)

**OBS / ! **

Intersectia se poate simula utilizand operatorul **IN** sau operatorul **EXISTS**.

Cum procedam atunci cand utilizam operatorul IN?

Cerere SQL (simularea intersectiei folosind operatorul **IN**):

```
SELECT manager_id
FROM departments
WHERE manager_id IN (SELECT manager_id
                     FROM employees ) ;
```

Simularea intersectiei folosind operatorul **EXISTS**:

```
SELECT manager_id
FROM departments d
WHERE EXISTS (SELECT manager_id
              FROM employees e
              WHERE d.manager_id = e.manager_id
              );
```

In ceea ce priveste performanta, operatorul **EXISTS** este optim deoarece in momentul in care acesta regaseste o linie care satisface conditia, returneaza TRUE si nu mai continua cautarea in cererea interna.

Operatorul **PRODUCT** – produsul cartezian dintre relatia R si relatia S

Notatie: ***PRODUCT(R, S)***

Exemplu: Sa se afiseze toti angajatii (codurile si numele angajatilor) impreuna cu toate departamentele (numele si codurile departamentelor).

Cerere SQL:

```
SELECT employee_id, last_name, d.department_id, department_name
FROM employees e, departments d;
```

Algebra relationala (expresia algebrica):

R1 = PROJECT(EMPLOYEES, employee_id, last_name)
R2 = PROJECT(DEPARTMENTS, department_id, department_name)
Rezultat = PRODUCT(R1, R2)

Cerere SQL (utilizand CROSS JOIN):

```
SELECT employee_id, last_name, d.department_id, department_name
FROM employees e CROSS JOIN departments d;
```

Operatorul **JOIN** – permite regasirea informatiei din mai multe relatii corelate. Operatorul combina **produsul cartezian** cu **selectia** si **proiectia**.

Sunt 4 tipuri de **JOIN**:

- NATURAL JOIN
- θ-JOIN
- SEMI-JOIN
- OUTER JOIN

1. **NATURAL JOIN** – combina tupluri din doua relatii pe baza valorilor comune. Cu alte cuvinte, acest tip de join realizeaza join automat (implicit) bazat pe coloanele comune existe in cele doua tabele intre care se realizeaza operatia de join. Coloanele comune sunt cele care au acelasi nume in ambele tabele.

Notatie: ***JOIN(R, S)***

Exemplu: Sa se afiseze informatii despre angajatii care lucreaza in departamente si care sunt in acelasi timp atat manageri de departament, cat si managerii altor angajati.

Algoritmul compunerii naturale:

- Se calculeaza produsul cartezian dintre relatiile R si S (R reprezinta relatia employees, iar S relatia departments);

  SELECT employee_id, last_name, d.department_id, department_name, d.manager_id
  FROM ***employees e, departments d****;*

- Se selecteaza tuplurile pentru care exista valori comune atat in relatia R, cat si in relatia S (valorile comune sunt – department_id si manager_id – acestea facand parte din ambele relatii)

  WHERE e.department_id = d.department_id AND
  e.manager_id = d.manager_id;

- Pentru fiecare atribut existent in ambele relatii, in afisare se pastreaza doar o singura data coloana comuna (in cazul nostru o sa avem o singura data coloana department_id si o singura data coloana manager_id).

Algoritmul anterior se aplica automat atunci cand utilizam **NATURAL JOIN**, nefiind nevoie de aliasuri.

SELECT employee_id, last_name, department_id, department_name, manager_id
FROM employees ***NATURAL JOIN*** *departments;*

Algebra relationala (expresia algebrica):

Rezultat = JOIN(EMPLOYEES, DEPARTMENTS)

2. **θ-JOIN** – combina tupluri (coloane/atribute) din doua relatii R si S pe baza unei conditii specificata in cadrul operatiei de JOIN. Aceasta conditie poate fi bazata pe egalitatea unor coloane (de obicei cheia primara si cheia externa) – acesta fiind un caz particul de **θ-JOIN**, si anume **EQUIJOIN (sau inner join)**, dar sunt utilizate in egala masura si conditii care implica operatorii <, <=, >, >=, != (acest tip de join fiind un **NONEQUIJOIN**)

Notatie: ***JOIN(R, S, conditie)***

Exemplu: Sa se afiseze numele angajatilor impreuna cu denumirea jobului in cadrul caruia lucreaza.

Cerere SQL:

```
SELECT last_name, job_title
FROM employees e, jobs j
WHERE e.job_id = j.job_id;
```

Algebra relationala (expresia algebrica):

R1 = PROJECT(EMPLOYEES, last_name)
R2 = PROJECT(JOBS, job_title)
Rezultat = JOIN(R1, R2, EMPLOYEES.job_id = JOBS.job_id)

3. **SEMI-JOIN** – se utilizeaza atunci cand din cele doua relatii participante R si S se utilizeaza doar atributele relatiei R sau doar cele ale relatiei S.

Acest operator poate fi scris ca o combinatie intre proiectie si compunere naturala sau proiectie si θ-JOIN, dupa cum urmeaza:

Notatie: ***SEMIJOIN(R, S) = PROJECT(JOIN(R, S))***
SEMIJOIN(R, S, conditie) = PROJECT(JOIN(R, S, conditie))

Exemplu: Sa se afiseze departamentele (codul si denumirea) care au cel putin un angajat.

Daca se utilizeaza un JOIN conventional numele departamentului o sa fie afisat de un numar de ori egal cu numarul de angajati din departamentul respectiv. In aceasta situatie se poate utiliza **DISTINCT**.

```
SELECT d.department_id, department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

Algebra relationala (expresia algebrica):

R1 = JOIN(EMPLOYEES, DEPARTMENTS, EMPLOYEES.department_id = DEPARTMENTS.department_id)
Rezultat = PROJECT(R1, department_id, department_name)

De asemenea, fiind nevoie de coloane specifice doar unei relatii, si anume coloane doar din tabelul *DEPARTMENTS,* atunci in practica putem utiliza o varianta mai optima cu ajutorul operatorului **EXISTS**. Operatorul EXISTS nu mai continua cautarea atunci cand a intalnit o valoare care sa satisfaca conditia. In cazul nostru este suficient ca intr-un departament sa exista cel putin un angajat. In plus, utilizarea acestui operator elimina implicit si duplicatele.

```
SELECT department_id, department_name
FROM departments d
WHERE EXISTS (SELECT department_id
              FROM employees e
              WHERE d.department_id = e.department_id
             );
```

Operatorul EIXSTS se poate inlocui cu operatorul **IN**. Si acesta elimina duplicatele, dar nu se opreste din executie atunci cand intalneste o valoare care respecta conditia, verificand valoarea din cererea parinte cu toate valorile returnate de subcerere (cererea copil/ cererea interioara).

```
SELECT department_id, department_name
FROM departments
WHERE department_id IN (SELECT department_id
                        FROM employees
                       );
```

4. **OUTER JOIN** – pentru doua relatii R si S, acest operator returneaza valorile atributelor care indeplinesc conditia de corelare (adica intersectia celor doua relatii pe baza unei valori comune – join clasic), impreuna cu valori NULL in functie de tipul acestui operator:

   - R **LEFT OUTER JOIN** S – se vor afisa valorile din R si S care respecta conditia de JOIN (intersectia) impreuna cu valorile care sunt in R (LEFT) si nu sunt in S. In acest caz valorile care lipsesc vor fi automat completate cu NULL.

Exemplu:

Consideram tabelul **EMPLOYEES ca fiind relatia R** si tabelul **DEPARTMENTS ca fiind relatia S**

Sa se afiseze angajatii (cod si nume) impreuna cu departamentele in care lucreaza (cod si denumire). Rezultatul o sa includa toti angajatii chiar daca nu au departament.

SELECT employee_id, d.department_id, last_name, department_name
FROM employees e LEFT JOIN departments d ON (e.department_id = d.department_id);

In output se observa angajatii care lucreaza intr-un departament, adica valorile din *EMPLOYEES* si *DEPARTMENTS* (din R si S) care indeplinesc conditia de corelare (e.department_id = d.department_id), impreuna cu valorile care sunt in *EMPLOYEES* (R) si nu sunt in *DEPARTMENTS* (S), adica angajatii care nu au departament. Valorile care lipsesc vor fi completate automat cu NULL – in exemplul nostru valorea care lipseste este cea specifica departamentului deoarece sunt afisati si angajatii **care nu au departament,** ceea ce inseamna ca angajatul exista, dar departamentul nu. In acest caz o sa existe un rezultat ca acesta :

178 NULL Grant NULL – coloanele reprezentative departamentului (department_id si department_name sunt automat completate cu valoarea NULL)

In mod asemanator se vor comporta si variantele urmatoare de JOIN (RIGHT OUTER JOIN si FULL OUTER JOIN)

- R **RIGHT OUTER JOIN** S – se vor afisa valorile din R si S care respecta conditia de JOIN (intersectia) impreuna cu valorile care sunt in S (RIGHT) si nu sunt in R. In acest caz valorile care lipsesc vor fi automat completate cu NULL.

- R **FULL OUTER JOIN** S – se vor afisa valorile din R si S care respecta conditia de JOIN (intersectia) impreuna cu valorile care sunt in R si nu sunt in S si valorile care sunt in S si nu sunt in R. In acest caz valorile care lipsesc vor fi automat completate cu NULL.