

Clauza Connect By, Ierarhii, Cicluri, Recursivitate:

Referat Sisteme de gestiune a bazelor de date

Tender Laura - Maria

Grupa 234

December 1, 2020

Introducere

În acest referat vom descrie o metodă de abordare a cererilor ierarhice cu ajutorul clauzei *CONNECT BY*. Vom analiza parametrii care pot fi folosiți împreună cu această clauză pentru a afla mai multe informații despre ierarhie. Vom discuta modul în care se comportă ciclurile și funcțiile recursive infinite.

Descrierea datelor și a problemei abordate

Exemplele prezentate în acest referat folosesc tabelul *Persoana* care a fost creat folosind următoarele comenzi:

```
CREATE TABLE Persoana (
    id_persoana NUMBER,
    id_parinte NUMBER,
    CNP VARCHAR2(15),
    nume varchar2(25) NOT NULL,
    prenume varchar2(25) NOT NULL,
    telefon varchar2(15),
    judet varchar2(15),
    CONSTRAINT persoana_pk PRIMARY KEY (id_persoana),
    CONSTRAINT persoana_persoana_fk FOREIGN KEY (id_parinte)
        REFERENCES Persoana(id_persoana),
    CHECK (length(CNP)= 13));

ALTER TABLE Persoana
ADD CONSTRAINT UQ_CNP UNIQUE (CNP);
```

În tabel au fost inserate date astfel:

```
INSERT INTO Persoana VALUES (1, NULL, '1410721106449', 'Popa', 'Ioan', '0740000000', 'Buzau');

INSERT INTO Persoana VALUES (2, 1, '2651210226968', 'Popa', 'Maria', '0741111111', 'Iasi');

INSERT INTO Persoana VALUES (3, 1, '1680217228786', 'Ionescu', 'Marcel', '0742222222', 'Iasi');

INSERT INTO Persoana VALUES (4, 1, '1710428229854', 'Popa', 'George', '0743333333', 'Iasi');

INSERT INTO Persoana VALUES (5, 2, '1910815445766', 'Ionescu', 'Andrei', '0750000000', 'Bucuresti');

INSERT INTO Persoana VALUES (6, 4, '5011003127121', 'Popa', 'Alexandru', '0760000000', 'Cluj');

INSERT INTO Persoana VALUES (7, 2, '6011110417243', 'Ionescu', 'Ioana', '0751111111', 'Bucuresti');

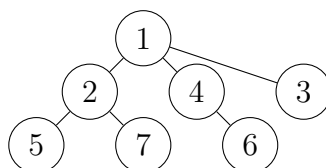
COMMIT;
```

Datele din tabel sunt următoarele:

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET
1	1	(null)	1410721106449	Popa	Ioan	0740000000	Buzau
2	2	1	2651210226968	Popa	Maria	0741111111	Iasi
3	3	1	1680217228786	Ionescu	Marcel	0742222222	Iasi
4	4	1	1710428229854	Popa	George	0743333333	Iasi
5	5	2	1910815445766	Ionescu	Andrei	0750000000	Bucuresti
6	6	4	5011003127121	Popa	Alexandru	0760000000	Cluj
7	7	2	6011110417243	Ionescu	Ioana	0751111111	Bucuresti

Între persoanele din tabel există o relație părinte - copil. Astfel persoanele din tabel pot fi reprezentate printr-un arbore. Acest arbore ne va ajuta să observăm mai bine modul

în care funcționează cererile din cadrul acestui referat.



Problema generală pe care o vom aborda în exemplele următoare este de a selecta persoanele din baza de date pe baza ierarhiei părinte - copil.

Introducere în CONNECT BY

Fie următorul cod care obține următorul rezultat:

```
SELECT *
FROM persoana
CONNECT BY PRIOR id_persoana = id_parinte;
```

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET
1	2		1 2651210226968	Popa	Maria	0741111111	Iasi
2	5		2 1910815445766	Ionescu	Andrei	0750000000	Bucuresti
3	7		2 6011110417243	Ionescu	Ioana	0751111111	Bucuresti
4	3		1 1680217228786	Ionescu	Marcel	0742222222	Iasi
5	4		1 1710428229854	Popa	George	0743333333	Iasi
6	6		4 5011003127121	Popa	Alexandru	0760000000	Cluj
7	5		2 1910815445766	Ionescu	Andrei	0750000000	Bucuresti
8	7		2 6011110417243	Ionescu	Ioana	0751111111	Bucuresti
9	6		4 5011003127121	Popa	Alexandru	0760000000	Cluj
10	1	(null)	1410721106449	Popa	Ioan	0740000000	Buzau
11	2		1 2651210226968	Popa	Maria	0741111111	Iasi
12	5		2 1910815445766	Ionescu	Andrei	0750000000	Bucuresti
13	7		2 6011110417243	Ionescu	Ioana	0751111111	Bucuresti
14	3		1 1680217228786	Ionescu	Marcel	0742222222	Iasi
15	4		1 1710428229854	Popa	George	0743333333	Iasi
16	6		4 5011003127121	Popa	Alexandru	0760000000	Cluj

O primă observație este aceea că apar duplicate, obținem mai multe rezultate decât numărul de rânduri din tabelul *Persoana*. Vom înțelege mai bine de ce obținem acest rezultat în următoarele exemple.

Ne dorim să nu obținem duplicate și să obținem rezultatele pornind din rădăcină. Astfel vom folosi *START WITH*.

```
SELECT *
FROM persoana
START WITH id_persoana = 1
CONNECT BY PRIOR id_persoana = id_parinte;
```

Script Output x Query Result x

SQL | All Rows Fetched: 7 in 0.016 seconds

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET
1	1	(null)	1410721106449	Popa	Ioan	0740000000	Buzau
2	2	1	2651210226968	Popa	Maria	0741111111	Iasi
3	5	2	1910815445766	Ionescu	Andrei	0750000000	Bucuresti
4	7	2	6011110417243	Ionescu	Ioana	0751111111	Bucuresti
5	3	1	1680217228786	Ionescu	Marcel	0742222222	Iasi
6	4	1	1710428229854	Popa	George	0743333333	Iasi
7	6	4	5011003127121	Popa	Alexandru	0760000000	Cluj

Folosind *START WITH* obținem toate nodurile subarborelui o singură dată, în ordinea dată de parcurgere.

LEVEL

Folosind *LEVEL* putem observa mai bine cum functioneaza *CONNECT BY*, *PRIOR* și ce se intampla atunci cand nu folosim *START WITH*. Vom adauga *LEVEL* la comanda anterioară pentru a observa întâi ce face acesta.

```
SELECT p.*, LEVEL
FROM persoana p
START WITH id_persoana = 2
CONNECT BY PRIOR id_persoana = id_parinte;
```

Script Output x Query Result x

SQL | All Rows Fetched: 3 in 0.018 seconds

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	LEVEL
1	2	1	2651210226968	Popa	Maria	0741111111	Iasi	1
2	5	2	1910815445766	Ionescu	Andrei	0750000000	Bucuresti	2
3	7	2	6011110417243	Ionescu	Ioana	0751111111	Bucuresti	2

Obținem subarborele rădăcinii precizate cu ajutorul *START WITH*, iar *LEVEL* indică nivelul nodului în subgraf, rădăcina primește nivelul 1, fiii acesteia nivelul 2 și tot așa.

Vom folosi *LEVEL* pentru cererea fără *START WITH* și vom obține următorul rezultat.

```
SELECT p.*, LEVEL
FROM persoana p
CONNECT BY PRIOR id_persoana = id_parinte;
```

ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	LEVEL
1	2	1 2651210226968	Popa	Maria	0741111111	Iasi	1
2	5	2 1910815445766	Ionescu	Andrei	0750000000	Bucuresti	2
3	7	2 6011110417243	Ionescu	Ioana	0751111111	Bucuresti	2
4	3	1 1680217228786	Ionescu	Marcel	0742222222	Iasi	1
5	4	1 1710428229854	Popa	George	0743333333	Iasi	1
6	6	4 5011003127121	Popa	Alexandru	0760000000	Cluj	2
7	5	2 1910815445766	Ionescu	Andrei	0750000000	Bucuresti	1
8	7	2 6011110417243	Ionescu	Ioana	0751111111	Bucuresti	1
9	6	4 5011003127121	Popa	Alexandru	0760000000	Cluj	1
10	1 (null)	1410721106449	Popa	Ioan	0740000000	Buzau	1
11	2	1 2651210226968	Popa	Maria	0741111111	Iasi	2
12	5	2 1910815445766	Ionescu	Andrei	0750000000	Bucuresti	3
13	7	2 6011110417243	Ionescu	Ioana	0751111111	Bucuresti	3
14	3	1 1680217228786	Ionescu	Marcel	0742222222	Iasi	2
15	4	1 1710428229854	Popa	George	0743333333	Iasi	2
16	6	4 5011003127121	Popa	Alexandru	0760000000	Cluj	3

Parcurea porneste din primul rând din tabel care îndeplinește condiția (1 nu îndeplinește condiția întrucât are id parinte NULL). În parcurea, nivelul celorlalte noduri se raporteaza la cele vizitate deja, fiii primind nivelul parintelui + 1. După ce parcurea din nodul curent a luat sfârșit, se pornește o nouă parcurea de la următorul nod din tabel. Din acest motiv 1, 2, 3 apar cu nivelul 1 în rezultat și obținem duplicate. Observăm că parcurea pornită din 1 nu obține rezultate contradictorii.

Vom face ca prima parcurea să pornească cu primul rând din tabel cu ajutorul *NVL*.

```
SELECT p.*, LEVEL
FROM persoana p
CONNECT BY PRIOR id_persoana = NVL(id_parinte, 0);
```

Script Output x Query Result x

SQL | All Rows Fetched: 16 in 0.017 seconds

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	LEVEL
1	1	(null)	1410721106449	Popa	Ioan	0740000000	Buzau	1
2	2	1	12651210226968	Popa	Maria	0741111111	Iasi	2
3	5	2	1910815445766	Ionescu	Andrei	0750000000	Bucuresti	3
4	7	2	6011110417243	Ionescu	Ioana	0751111111	Bucuresti	3
5	3	1	1680217228786	Ionescu	Marcel	0742222222	Iasi	2
6	4	1	1710428229854	Popa	George	0743333333	Iasi	2
7	6	4	5011003127121	Popa	Alexandru	0760000000	Cluj	3
8	2	1	12651210226968	Popa	Maria	0741111111	Iasi	1
9	5	2	1910815445766	Ionescu	Andrei	0750000000	Bucuresti	2
10	7	2	6011110417243	Ionescu	Ioana	0751111111	Bucuresti	2
11	3	1	1680217228786	Ionescu	Marcel	0742222222	Iasi	1
12	4	1	1710428229854	Popa	George	0743333333	Iasi	1
13	6	4	5011003127121	Popa	Alexandru	0760000000	Cluj	2
14	5	2	1910815445766	Ionescu	Andrei	0750000000	Bucuresti	1
15	7	2	6011110417243	Ionescu	Ioana	0751111111	Bucuresti	1
16	6	4	5011003127121	Popa	Alexandru	0760000000	Cluj	1

Următoarea problemă - Obțineți strămoșii persoanei cu id-ul 1 poate fi rezolvată ușor folosind LEVEL în clauza WHERE.

```
SELECT p.*, LEVEL
FROM persoana p
WHERE LEVEL > 1
START WITH id_persoana = 1
CONNECT BY PRIOR id_persoana = id_parinte;
```

Script Output x Query Result x

SQL | All Rows Fetched: 6 in 0.015 seconds

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	LEVEL
1	2	1	12651210226968	Popa	Maria	0741111111	Iasi	2
2	5	2	1910815445766	Ionescu	Andrei	0750000000	Bucuresti	3
3	7	2	6011110417243	Ionescu	Ioana	0751111111	Bucuresti	3
4	3	1	1680217228786	Ionescu	Marcel	0742222222	Iasi	2
5	4	1	1710428229854	Popa	George	0743333333	Iasi	2
6	6	4	5011003127121	Popa	Alexandru	0760000000	Cluj	3

PRIOR

În cererile anterioare am folosit *PRIOR* însă nu discutat ce face acesta și cum îl putem folosi. Pentru început, vom observa ce se întâmplă în cereri ce nu conțin *PRIOR*.

```
-- absenta prior
SELECT p.*, LEVEL
FROM persoana p
CONNECT BY id_persoana = id_parinte;
```

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	LEVEL
1	1	(null)	1410721106449	Popa	Ioan	0740000000	Buzau	1
2	2		12651210226968	Popa	Maria	0741111111	Iasi	1
3	3		11680217228786	Ionescu	Marcel	0742222222	Iasi	1
4	4		11710428229854	Popa	George	0743333333	Iasi	1
5	5		21910815445766	Ionescu	Andrei	0750000000	Bucuresti	1
6	6		45011003127121	Popa	Alexandru	0760000000	Cluj	1
7	7		26011110417243	Ionescu	Ioana	0751111111	Bucuresti	1

Observăm că obținem fiecare nod cu nivelul 1, absența *PRIOR* face ca cererea să nu se mai raporteze la nodul anterior, deci nu se mai întâmplă o parcurgere. Dacă indicăm din ce nod dorim să începă parcurgerea obținem doar acel nod.

```
SELECT p.*, LEVEL
FROM persoana p
START WITH id_persoana = 1
CONNECT BY id_persoana = id_parinte;
```

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	LEVEL
1	1	(null)	1410721106449	Popa	Ioan	0740000000	Buzau	1

Putem așeza *PRIOR* și în partea dreaptă a clauzei *CONNECT BY*.


```
SELECT p.*, LEVEL
FROM persoana p
START WITH id_persoana = 1
CONNECT BY id_parinte = PRIOR id_persoana;
```

Script Output x Query Result x

SQL | All Rows Fetched: 7 in 0.017 seconds

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	LEVEL
1	1	(null)	1410721106449	Popa	Ioan	0740000000	Buzau	1
2	2		12651210226968	Popa	Maria	0741111111	Iasi	2
3	5		21910815445766	Ionescu	Andrei	0750000000	Bucuresti	3
4	7		26011110417243	Ionescu	Ioana	0751111111	Bucuresti	3
5	3		11680217228786	Ionescu	Marcel	0742222222	Iasi	2
6	4		11710428229854	Popa	George	0743333333	Iasi	2
7	6		45011003127121	Popa	Alexandru	0760000000	Cluj	3

Penru a obține o parcurgere ascendentă, de la un copil către părinte putem scrie o astfel de cerere:

```
SELECT p.*, LEVEL
FROM persoana p
START WITH id_persoana = 7
CONNECT BY PRIOR id_parinte = id_persoana;
```

Script Output x Query Result x

SQL | All Rows Fetched: 3 in 0.02 seconds

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	LEVEL
1	7		26011110417243	Ionescu	Ioana	0751111111	Bucuresti	1
2	2		12651210226968	Popa	Maria	0741111111	Iasi	2
3	1	(null)	1410721106449	Popa	Ioan	0740000000	Buzau	3

Cicluri

Dacă graful nostru ar conține un ciclu, parcurgerea ar putea merge la infinit sau să își piardă însemnătatea. Am adăugat în mod eronat un ciclu în tabel pentru a vedea ce rezultate obținem.

UPDATE Persoana

```
SET id_parinte = 5

WHERE id_persoana = 1;

COMMIT;
```

Rulând aceeași cerere de până acum în acest caz obținem eroarea *CONNECT BY loop in user data*. Deci algoritmul detectează existența unui ciclu și aruncă o eroare în acest caz.

```
SELECT p.*, LEVEL
FROM persoana p
START WITH id_persoana = 1
CONNECT BY PRIOR id_persoana = id_parinte;
```

Script Output x Query Result x

SQL | Executing: SELECT p.*, LEVEL FROM persoana p START WITH id_persoana = 1

ORA-01436: CONNECT BY loop in user data
 01436. 00000 - "CONNECT BY loop in user data"
 *Cause:
 *Action:

Dacă dorim să ignorăm ciclul și să obținem rezultatele putem folosi atributul *NOCYCLE*.

```
SELECT p.*, LEVEL
FROM persoana p
START WITH id_persoana = 1
CONNECT BY NOCYCLE PRIOR id_persoana = id_parinte;
```

Script Output x Query Result x

SQL | All Rows Fetched: 7 in 0.014 seconds

	ID_PERSOANA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	LEVEL
1	1	5	1410721106449	Popa	Ioan	0740000000	Buzau	1
2	2	1	2651210226968	Popa	Maria	0741111111	Iasi	2
3	5	2	1910815445766	Ionescu	Andrei	0750000000	Bucuresti	3
4	7	2	6011110417243	Ionescu	Ioana	0751111111	Bucuresti	3
5	3	1	1680217228786	Ionescu	Marcel	0742222222	Iasi	2
6	4	1	1710428229854	Popa	George	0743333333	Iasi	2
7	6	4	5011003127121	Popa	Alexandru	0760000000	Cluj	3

Pentru a determina nodurile care închid un ciclu putem folosi atributul *CONNECT BY IS-CYCLE*. În plus, pentru a afișa informații despre parcurgere putem folosi *SYS CONNECT*

BY PATH.

```

131 SELECT p.*, CONNECT_BY_ISCYCLE as Ciclu, LEVEL, SYS_CONNECT_BY_PATH(ume || ' ' || preume, '/') as Ierarhie
132 FROM persoana p
133 START WITH id_persoana = 1
134 CONNECT BY NOCYCLE PRIOR id_persoana = id_parinte;
135

```

ID_PERSONA	ID_PARINTE	CNP	NUME	PRENUME	TELEFON	JUDET	CICLU	LEVEL	IERARHIE
1	1	51410721106449	Popa	Ioan	0740000000	Buzau	0	1	/Popa Ioan
2	2	12651210226968	Popa	Maria	0741111111	Iasi	0	2	/Popa Ioan/Popa Maria
3	5	21910815445766	Ionescu	Andrei	0750000000	Bucuresti	1	3	/Popa Ioan/Popa Maria/Ionescu Andrei
4	7	26011110417243	Ionescu	Ioana	0751111111	Bucuresti	0	3	/Popa Ioan/Popa Maria/Ionescu Ioana
5	3	11680217228786	Ionescu	Marcel	0742222222	Iasi	0	2	/Popa Ioan/Ionescu Marcel
6	4	11710428229854	Popa	George	0743333333	Iasi	0	2	/Popa Ioan/Popa George
7	6	45011003127121	Popa	Alexandru	0760000000	Cluj	0	3	/Popa Ioan/Popa George/Popa Alexandru

CONNECT BY ROOT, CONNECT BY ISLEAF

CONNECT BY ROOT ajută la obținerea de informație despre rădăcină, nodul din care a pornit parcurgerea.

```

=SELECT id_persoana, nume, prenume, id_parinte, CONNECT_BY_ROOT nume "Nume parinte", CONNECT_BY_ROOT prenume "Prenume parinte", LEVEL
FROM persoana
START WITH id_persoana = 1
CONNECT BY PRIOR id_persoana = id_parinte;

```

ID_PERSONA	NUME	PRENUME	ID_PARINTE	Nume parinte	Prenume parinte	LEVEL
1	1 Popa	Ioan	(null)	Popa	Ioan	1
2	2 Popa	Maria	1 Popa	Popa	Ioan	2
3	5 Ionescu	Andrei	2 Popa	Popa	Ioan	3
4	7 Ionescu	Ioana	2 Popa	Popa	Ioan	3
5	3 Ionescu	Marcel	1 Popa	Popa	Ioan	2
6	4 Popa	George	1 Popa	Popa	Ioan	2
7	6 Popa	Alexandru	4 Popa	Popa	Ioan	3

Observăm că pentru fiecare nod obținem informația despre nodul din *START WITH*.

```
SELECT id_persoana, nume, prenume, id_parinte, CONNECT_BY_ROOT nume "Nume parinte", CONNECT_BY_ROOT prenume "Prenume parinte", LEVEL
FROM persoana
CONNECT BY PRIOR id_persoana = id_parinte;
```

Query Result x

All Rows Fetched: 16 in 0.019 seconds

	ID_PERSOANA	NUME	PRENUME	ID_PARINTE	Nume parinte	Prenume parinte	LEVEL
1	2	Popa	Maria	1	Popa	Maria	1
2	5	Ionescu	Andrei	2	Popa	Maria	2
3	7	Ionescu	Ioana	2	Popa	Maria	2
4	3	Ionescu	Marcel	1	Ionescu	Marcel	1
5	4	Popa	George	1	Popa	George	1
6	6	Popa	Alexandru	4	Popa	George	2
7	5	Ionescu	Andrei	2	Ionescu	Andrei	1
8	7	Ionescu	Ioana	2	Ionescu	Ioana	1
9	6	Popa	Alexandru	4	Popa	Alexandru	1
10	1	Popa	Ioan	(null)	Popa	Ioan	1
11	2	Popa	Maria	1	Popa	Ioan	2
12	5	Ionescu	Andrei	2	Popa	Ioan	3
13	7	Ionescu	Ioana	2	Popa	Ioan	3
14	3	Ionescu	Marcel	1	Popa	Ioan	2
15	4	Popa	George	1	Popa	Ioan	2
16	6	Popa	Alexandru	4	Popa	Ioan	3

Pentru a afla dacă un nod este frunză putem folosi `CONNECT BY ISLEAF`. Astfel aflăm persoanele fără copii.

```
SELECT id_persoana, nume, prenume, id_parinte, connect_by_isleaf as Frunza
FROM persoana
START WITH id_persoana = 1
CONNECT BY PRIOR id_persoana = id_parinte;
```

Query Result x

All Rows Fetched: 7 in 0.022 seconds

	ID_PERSOANA	NUME	PRENUME	ID_PARINTE	FRUNZA
1	1	Popa	Ioan	(null)	0
2	2	Popa	Maria	1	0
3	5	Ionescu	Andrei	2	1
4	7	Ionescu	Ioana	2	1
5	3	Ionescu	Marcel	1	1
6	4	Popa	George	1	0
7	6	Popa	Alexandru	4	1

Parcurgerea

Vom descrie pașii pe care Oracle îi urmează pentru a crea ierarhia folosind `CONNECT BY`.

Sunt selectate rândurile care îndeplinescu condiția din *START WITH*. Apoi Oracle selectează rândurile copil pentru fiecare rădăcină. Copiii trebuie să îndeplinească condiția din *CONNECT BY* față de o rădăcină. Dacă cererea conține o clauza *WHERE* atunci Oracle evaluează fiecare rând și le păstrează pe cele care îndeplinesc condiția.¹

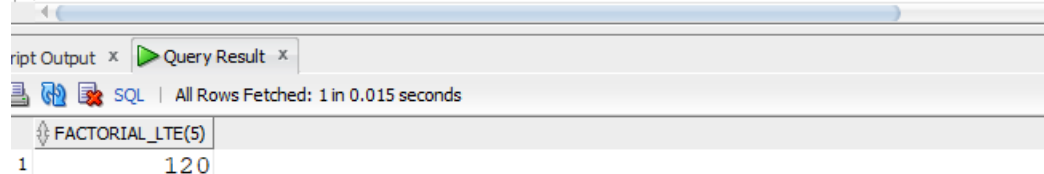
Observăm că rezultatul este o parcurgere Depth First Search (DFS).

Recursivitate

Fie funcția recursivă factorial din laboratorul 4 PL/SQL (8).

```
CREATE OR REPLACE FUNCTION factorial_lte(n NUMBER)
RETURN INTEGER IS
BEGIN
    IF (n=0) THEN RETURN 1;
    ELSE RETURN n*factorial_lte(n-1);
    END IF;
END factorial_lte;
/

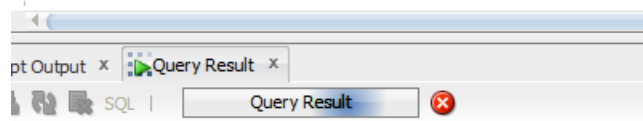
SELECT factorial_lte(5) FROM DUAL;
```



The screenshot shows the SQL Developer interface. The top pane contains the PL/SQL code for the `factorial_lte` function and the query `SELECT factorial_lte(5) FROM DUAL;`. The bottom pane shows the 'Query Result' tab with the result '120' for the query `FACTORIAL_LTE(5)`. The status bar indicates 'All Rows Fetched: 1 in 0.015 seconds'.

Am apelat funcția factorial pentru o valoare negativă.

```
SELECT factorial_lte(-1) FROM DUAL;
```



The screenshot shows the SQL Developer interface. The top pane contains the query `SELECT factorial_lte(-1) FROM DUAL;`. The bottom pane shows the 'Query Result' tab with an error message (indicated by a red 'X' icon) for the query `Query Result`. The status bar indicates 'All Rows Fetched: 1 in 0.015 seconds'.

1. Hierarchical Queries, Database SQL Reference, Oracle Documentation

Dacă subprogramul nu ajunge la o condiție terminală, recursia continuă până PL/SQL rămâne fără memorie și aruncă excepția predefinită *STORAGE ERROR*.²

Putem rescrie funcția factorial adaugând o excepție pentru numere negative.

```

173 CREATE OR REPLACE FUNCTION factorial_lte(n NUMBER)
174     RETURN INTEGER IS
175     negative_number EXCEPTION;
176     PRAGMA EXCEPTION_INIT (negative_number, -20001);
177     BEGIN
178     IF (n < 0) THEN
179         raise_application_error(-20001,'Nu poate fi calculat factorialul unui numar negativ');
180     ELSE
181         IF (n=0) THEN RETURN 1;
182         ELSE RETURN n*factorial_lte(n-1);
183         END IF;
184     END IF;
185 END factorial_lte;
186 /
187
188 SELECT factorial_lte(-1) FROM DUAL;

```

Script Output x Query Result x

SQL | Executing: SELECT factorial_lte(-1) FROM DUAL in 0 seconds

ORA-20001: Nu poate fi calculat factorialul unui numar negativ
ORA-06512: at "GRUPA234.FACTORIAL_LTE", line 7

Bibliografie

Hierarchical Queries in Oracle, Oracle-Base

Connect By, Snowflake Documentation

CONNECT BY clause in Oracle, Oradev

PL/SQL Raise Exceptions, Oracle Tutorial

2. Using Recursive PL/SQL Subprograms, Database PL/SQL Language Reference, Oracle Documentation