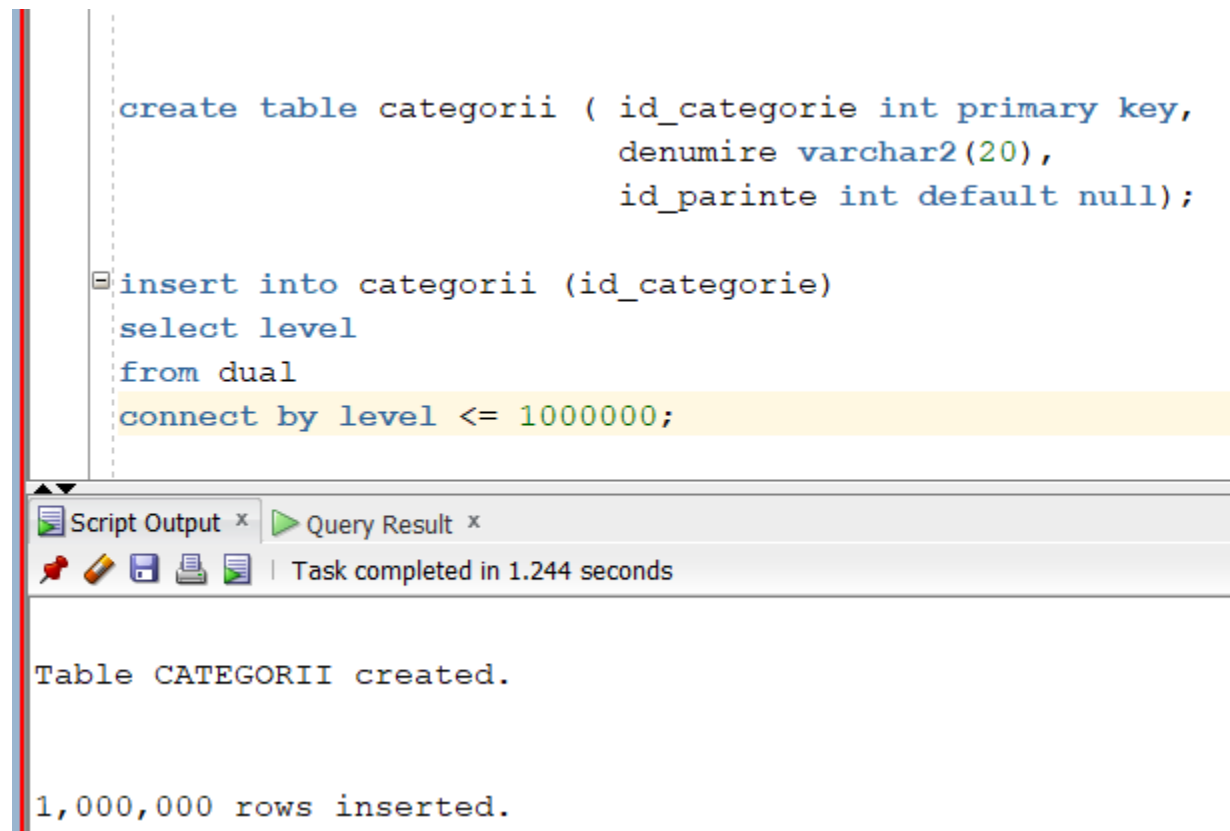


## ***SELECT BULK COLLECT INTO vs FETCH BULK COLLECT INTO***

*-Cursor implicit vs Cursor explicit-*

Creăm tabelul CATEGORII, asemănător exemplului 5.6, și inserăm 1.000.000 înregistrări în el, pentru a putea vizualiza diferența de timp dintre cele două (doar cu un număr foarte mare de înregistrări putem face acest lucru).



```
create table categorii ( id_categorie int primary key,  
                        denumire varchar2(20),  
                        id_parinte int default null);  
  
insert into categorii (id_categorie)  
select level  
from dual  
connect by level <= 1000000;
```

Script Output x Query Result x

Task completed in 1.244 seconds

Table CATEGORII created.

1,000,000 rows inserted.

*SELECT BULK COLLECT INTO (cursor implicit)*

```
DECLARE
    TYPE tab_imb IS TABLE OF categorii%ROWTYPE;
    v_categorii tab_imb;

    time1 NUMBER := dbms_utility.get_time;
BEGIN
    SELECT *
    BULK COLLECT INTO v_categorii
    FROM categorii
    WHERE id_parinte IS NULL;

    DBMS_OUTPUT.PUT_LINE('Elapsed time SELECT BULK COLLECT INTO = ' ||
                          (dbms_utility.get_time - time1)/100);
END;
/
```

*FETCH BULK COLLECT INTO (cursor explicit)*

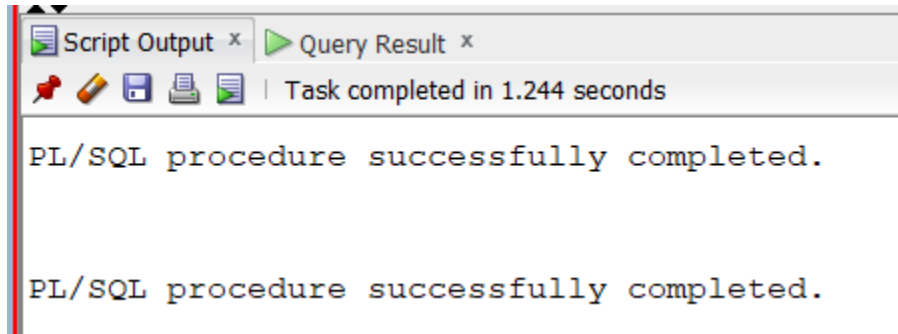
```
--5.6
DECLARE
    TYPE tab_imb IS TABLE OF categorii%ROWTYPE;
    v_categorii tab_imb;

    CURSOR c IS
        SELECT * FROM categorii
        WHERE id_parinte IS NULL;

    time1 NUMBER := dbms_utility.get_time;
BEGIN
    OPEN c;
    FETCH c BULK COLLECT INTO v_categorii;
    CLOSE c;

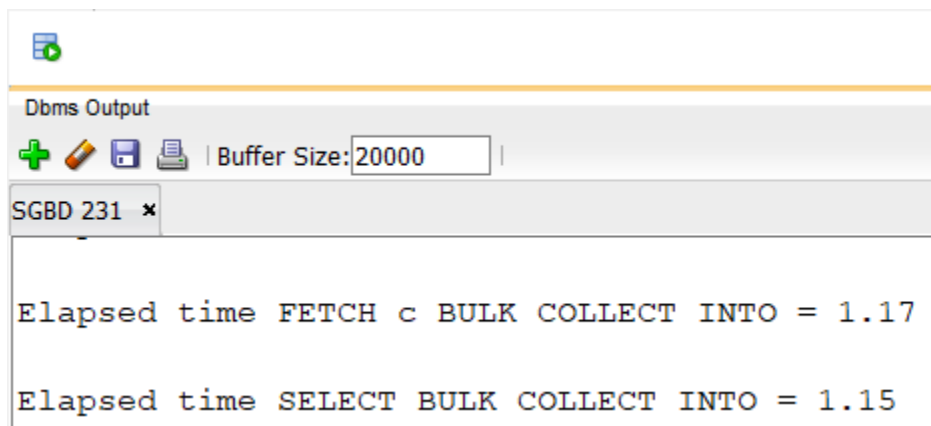
    DBMS_OUTPUT.PUT_LINE('Elapsed time FETCH c BULK COLLECT INTO = ' ||
                          (dbms_utility.get_time - time1)/100);
END;
/
```

Rulăm cele două blocuri de mai multe ori, pentru a putea avea o idee mai bună în legătură cu diferența de timp dintre cele două:



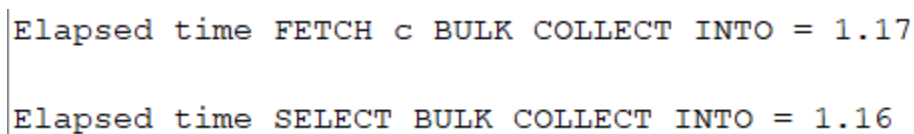
Script Output x Query Result x  
Task completed in 1.244 seconds  
PL/SQL procedure successfully completed.  
PL/SQL procedure successfully completed.

*Prima încercare:*



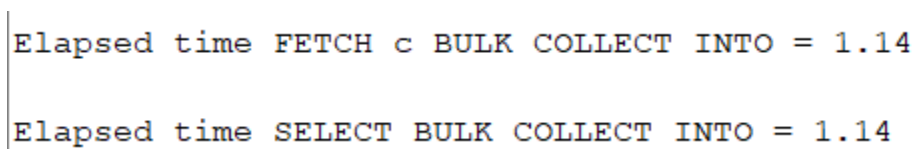
Dbms Output  
Buffer Size: 20000  
SGBD 231 x  
Elapsed time FETCH c BULK COLLECT INTO = 1.17  
Elapsed time SELECT BULK COLLECT INTO = 1.15

*A doua încercare:*



Elapsed time FETCH c BULK COLLECT INTO = 1.17  
Elapsed time SELECT BULK COLLECT INTO = 1.16

*A treia încercare:*



Elapsed time FETCH c BULK COLLECT INTO = 1.14  
Elapsed time SELECT BULK COLLECT INTO = 1.14

*A patra încercare:*

```
Elapsed time FETCH c BULK COLLECT INTO = 1.14
```

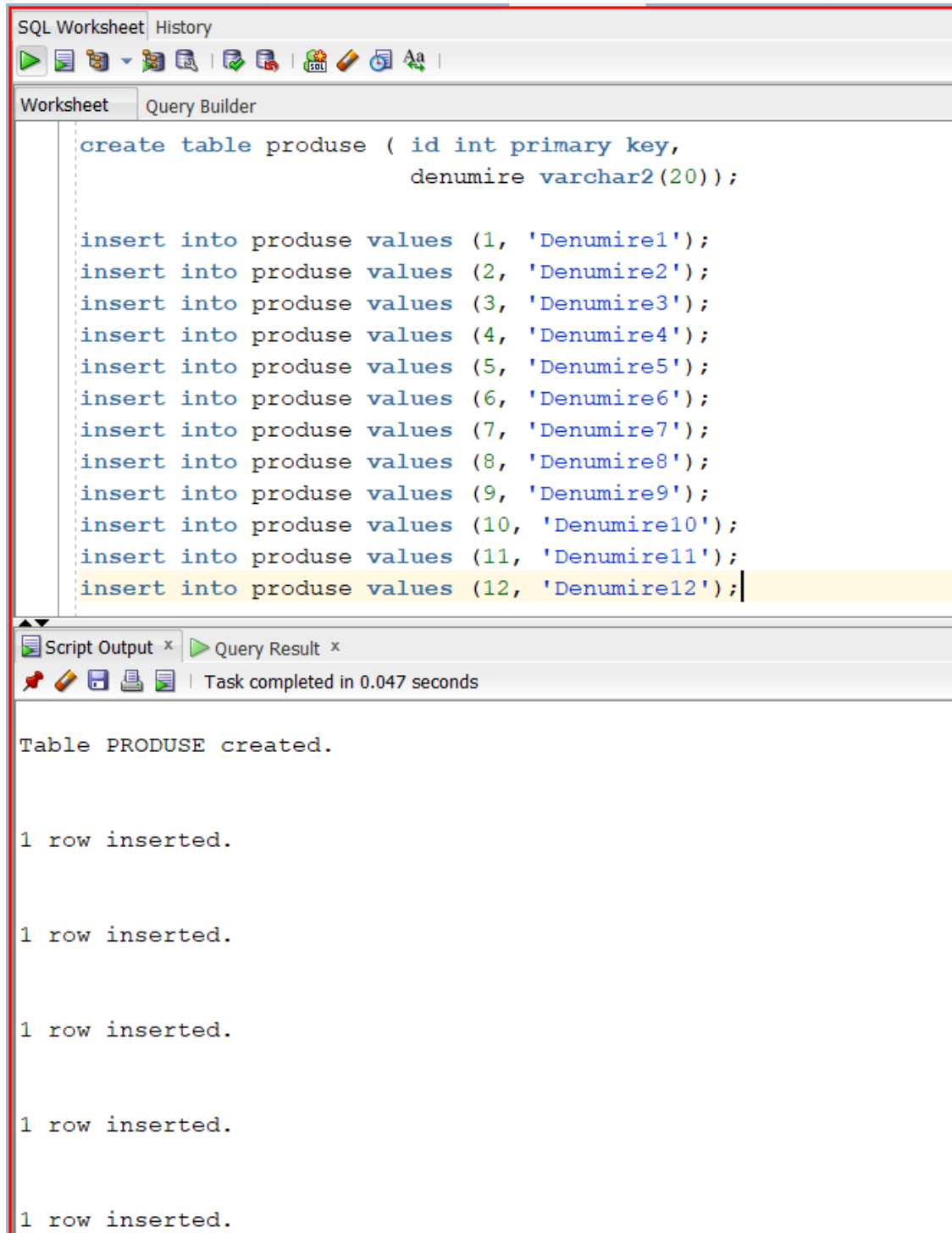
```
Elapsed time SELECT BULK COLLECT INTO = 1.14
```

Ne oprim. Observăm faptul că, după mai multe rulări, cele două devin egale ca timp de execuție.

Așadar, conform celor evidențiate mai sus, SELECT BULK COLLECT INTO (cursor implicit) are timpul de execuție mai mic decât FETCH BULK COLLECT INTO (cursor explicit).

## *Efectul utilizării mai multor FETCH-uri cu LIMIT*

Creăm tabelul PRODUSE din exercițiul 5.7 și adăugăm 12 înregistrări în acesta.



The screenshot shows an SQL Worksheet interface with a toolbar at the top containing icons for running queries, saving, and other database operations. The main area is divided into two tabs: 'Worksheet' and 'Query Builder'. The 'Worksheet' tab is active, displaying a SQL script. The script starts with a 'create table' statement for 'produse' with columns 'id' (primary key, integer) and 'denumire' (varchar2(20)). This is followed by 12 'insert into' statements, each adding a row with an 'id' from 1 to 12 and a 'denumire' from 'Denumire1' to 'Denumire12'. The last line of the script is highlighted in yellow. Below the script, there is a 'Script Output' tab and a 'Query Result' tab. The 'Script Output' tab is active, showing the execution results: 'Table PRODUSE created.', followed by six lines of '1 row inserted.', indicating that the script was executed in six batches of two rows each. A status bar at the bottom indicates 'Task completed in 0.047 seconds'.

```
create table produse ( id int primary key,  
                      denumire varchar2(20));  
  
insert into produse values (1, 'Denumire1');  
insert into produse values (2, 'Denumire2');  
insert into produse values (3, 'Denumire3');  
insert into produse values (4, 'Denumire4');  
insert into produse values (5, 'Denumire5');  
insert into produse values (6, 'Denumire6');  
insert into produse values (7, 'Denumire7');  
insert into produse values (8, 'Denumire8');  
insert into produse values (9, 'Denumire9');  
insert into produse values (10, 'Denumire10');  
insert into produse values (11, 'Denumire11');  
insert into produse values (12, 'Denumire12');
```

Table PRODUSE created.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

Rulăm exercițiul 5.7, ușor modificat (pentru problema noastră):

The screenshot displays an SQL Worksheet interface with a 'Query Builder' tab. The main area contains a PL/SQL procedure script. To the right, a 'Dbms Output' window shows the execution results. At the bottom, a status bar indicates the task is completed.

```
SQL Worksheet | History
0.058 seconds

Worksheet | Query Builder

DECLARE
    TYPE tab_imb IS TABLE OF produse.denumire%TYPE;
    v_produce tab_imb;
    v_denumire produse.denumire%TYPE;

    CURSOR c IS
        SELECT denumire
        FROM produse;
BEGIN
    OPEN c;
    DBMS_OUTPUT.PUT_LINE('First FETCH LIMIT 5:');
    FETCH c BULK COLLECT INTO v_produce LIMIT 5;
    FOR i IN 1..v_produce.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(v_produce(i));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('-----');

    DBMS_OUTPUT.PUT_LINE('Second FETCH LIMIT 5:');
    FETCH c BULK COLLECT INTO v_produce LIMIT 5;
    FOR i IN 1..v_produce.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(v_produce(i));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('-----');

    DBMS_OUTPUT.PUT_LINE('Third FETCH LIMIT 5:');
    FETCH c BULK COLLECT INTO v_produce LIMIT 5;
    CLOSE c;
    FOR i IN 1..v_produce.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(v_produce(i));
    END LOOP;
END;
/
```

Dbms Output  
Buffer Size: 20000  
SGBD 231 x

```
First FETCH LIMIT 5:
Denumire1
Denumire2
Denumire3
Denumire4
Denumire5
-----
Second FETCH LIMIT 5:
Denumire6
Denumire7
Denumire8
Denumire9
Denumire10
-----
Third FETCH LIMIT 5:
Denumire11
Denumire12
```

Script Output x | Query Result x  
Task completed in 0.058 seconds  
PL/SQL procedure successfully completed.

Remarcăm faptul că, de fiecare dată, cursorul explicit știe unde a rămas și returnează liniile tabelului fără a repeta nicio înregistrare.

De asemenea, observăm și că ultimul FETCH LIMIT 5 ne-a returnat doar 2 înregistrări și nici măcar o eroare. Acest lucru s-a întâmplat deoarece, înainte de al treilea FETCH LIMIT 5, mai existau doar 2 înregistrări în tabel care nu au fost afișate. Cursorul a știut unde a rămas la ultimul FETCH și și-a continuat treaba, fără a conta faptul că noi aveam doar 2 linii și limita era de 5.

Așadar, cursorul explicit are o multitudine de avantaje, printre care se numără și cel de mai sus: puterea de a reține, ca un pointer, unde a rămas în tabel.

## ***CURRENT OF vs ROWID***

### **Exemplul 5.12**

```
DECLARE
  CURSOR c IS
    SELECT id_produc
    FROM   produse
    WHERE  id_categorie IN
          (SELECT id_categorie
           FROM   categorii
           WHERE  denumire = 'Placi de retea Wireless')
    FOR UPDATE OF pret_unitar NOWAIT;

BEGIN
  FOR i IN c LOOP
    UPDATE produse
    SET    pret_unitar = pret_unitar*0.95
    WHERE CURRENT OF c;
  END LOOP;
  -- permanentizare si eliberare blocari
  COMMIT;
END;
/
```

### **Exemplul 5.13**

```
-- utilizare ROWID in loc de CURRENT OF

DECLARE
  CURSOR c IS
    SELECT id_produc, rowid
    FROM   produse
    WHERE  id_categorie IN
          (SELECT id_categorie
           FROM   categorii
           WHERE  denumire = 'Placi de retea Wireless')
    FOR UPDATE OF pret_unitar NOWAIT;

BEGIN
  FOR i IN c LOOP
    UPDATE produse
    SET    pret_unitar = pret_unitar*0.95
    WHERE ROWID = i.ROWID;
  END LOOP;
  COMMIT;
END;
/
```

Anterior, am observat că un cursor reține, ca un pointer, care a fost ultima linie returnată. Așadar, conform acestui lucru, consider că CURRENT OF (exemplul 5.12) este mai safe decât ROWID (exemplul 5.13), deoarece cursorul mereu va reține ultima linie returnată. În schimb, ROWID poate da greș uneori deoarece nu este în conexiune totală cu cursorul nostru.



## *EXISTS vs IN vs JOIN*

### *EXISTS*

The screenshot shows a SQL Worksheet interface with a query and its execution plan.

**SQL Query:**

```
--EXISTS
SELECT *
FROM employees e
WHERE
    EXISTS (SELECT 1
            FROM departments
            WHERE department_id = e.department_id);
```

**Execution Plan:**

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				106 3
TABLE ACCESS	EMPLOYEES	FULL		106 3

The execution plan also includes a filter predicate: **E.DEPARTMENT\_ID IS NOT NULL**.

**Other XML:**

```
{info}
  info type="db_version"
    11.1.0.6
  info type="parse_schema"
    "GRUPA231"
  info type="dynamic_sampling"
    yes
  info type="plan_hash"
    1445457117
  info type="plan_hash_2"
    3476115102
  {hint}
    FULL(@"SEL$0404685D" "E"@"SEL$1")
    OUTLINE(@"SEL$2")
    OUTLINE(@"SEL$1")
    UNNEST(@"SEL$2")
    OUTLINE(@"SEL$5DA710D3")
    ELIMINATE_JOIN(@"SEL$5DA710D3" "DEPARTMENTS"@"SEL$2")
    OUTLINE_LEAF(@"SEL$0404685D")
    ALL_ROWS
    DB_VERSION('11.1.0.6')
```

Cost EXISTS = 6

IN

SQL Worksheet | History

0.308 seconds

Worksheet | Query Builder

```
SELECT *
FROM employees e
WHERE department_id IN (SELECT department_id
                        FROM departments
                        WHERE department_id = e.department_id);
```

Script Output x | Query Result x | Explain Plan x

SQL | 0.308 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				106
NESTED LOOPS				106
TABLE ACCESS	EMPLOYEES	FULL		107
INDEX	DEPT_ID_PK	UNIQUE SCAN		1

Access Predicates  
DEPARTMENT\_ID=DEPARTMENT\_ID

Other XML

```
{info}
  info type="db_version"
  11.1.0.6
  info type="parse_schema"
  "GRUPA231"
  info type="dynamic_sampling"
  yes
  info type="plan_hash"
  169719308
  info type="plan_hash_2"
  3299912071
  {hint}
    USE_NL(@"SEL$5DA710D3" "DEPARTMENTS"@"SEL$2")
    LEADING(@"SEL$5DA710D3" "E"@"SEL$1" "DEPARTMENTS"@"SEL$2")
    INDEX(@"SEL$5DA710D3" "DEPARTMENTS"@"SEL$2" ("DEPARTMENTS"."DEPARTMENT_ID"))
    FULL(@"SEL$5DA710D3" "E"@"SEL$1")
    OUTLINE(@"SEL$2")
    OUTLINE(@"SEL$1")
    UNNEST(@"SEL$2")
    OUTLINE_LEAF(@"SEL$5DA710D3")
    ALL_ROWS
    DB_VERSION('11.1.0.6')
    OPTIMIZER_FEATURES_ENABLE('11.1.0.6')
    IGNORE_OPTIM_EMBEDDED_HINTS
```

Cost IN = 9

## JOIN

```
--JOIN
SELECT DISTINCT e.*
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

Script Output x Query Result x Explain Plan x

SQL 0.258 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				106
TABLE ACCESS	EMPLOYEES	FULL		106
Filter Predicates				3
E.DEPARTMENT_ID IS NOT NULL				3

Other XML

```
{info}
  info type="db_version"
  11.1.0.6
  info type="parse_schema"
  "GRUPA231"
  info type="dynamic_sampling"
  yes
  info type="plan_hash"
  1445457117
  info type="plan_hash_2"
  3476115102
  {hint}
    FULL(@"SEL$F7859CDE" "E"@"SEL$1")
    OUTLINE(@"SEL$1")
    ELIMINATE_JOIN(@"SEL$1" "D"@"SEL$1")
    OUTLINE_LEAF(@"SEL$F7859CDE")
    ALL_ROWS
    DB_VERSION("11.1.0.6")
    OPTIMIZER_FEATURES_ENABLE("11.1.0.6")
    IGNORE_OPTIM_EMBEDDED_HINTS
```

Cost JOIN = 6

Așadar, IN-ul este cel mai costisitor, iar EXISTS-ul și JOIN-ul sunt optime.