

Optimizare SQL

Cerință: Tabelul CLASIFIC_CLIENTI(*id_client*, *id_categorie*, *nr_produce*, *clasificare*) conține 7 linii, existând deja o clasificare a clienților. Avem tabelul CLASIFIC_CLIENTI la care trebuie să ajungem și mai multe tipuri de rezolvări ale exercițiului în imaginile de mai jos. Care este diferența de optimizare dintre variantele 1, 2 și 3?

| id_categorie | clasificare | număr produse cumpărate |
|--------------|-------------|-------------------------|
| 1 | A | > 1000 |
| | B | ≥ 500 |
| | C | ≥ 0 |
| 2 | A | > 2000 |
| | B | ≥ 1000 |
| | C | ≥ 200 |
| | D | ≥ 0 |

Varianta1

```
UPDATE clasific_clienti
SET    clasificare = 'A'
WHERE  nr_produce > 1000
AND    id_categorie = 1;
```

```
UPDATE clasific_clienti
SET    clasificare = 'B'
WHERE  nr_produce BETWEEN 500 AND 1000
AND    id_categorie = 1;
...
```

Varianta2

```
UPDATE clasific_clienti
SET    clasificare = CASE WHEN nr_produce>1000
                          THEN 'A'
                          WHEN nr_produce BETWEEN 500
                          AND 1000 THEN 'B'
                          ELSE 'C' END
WHERE  id_categorie = 1;
```

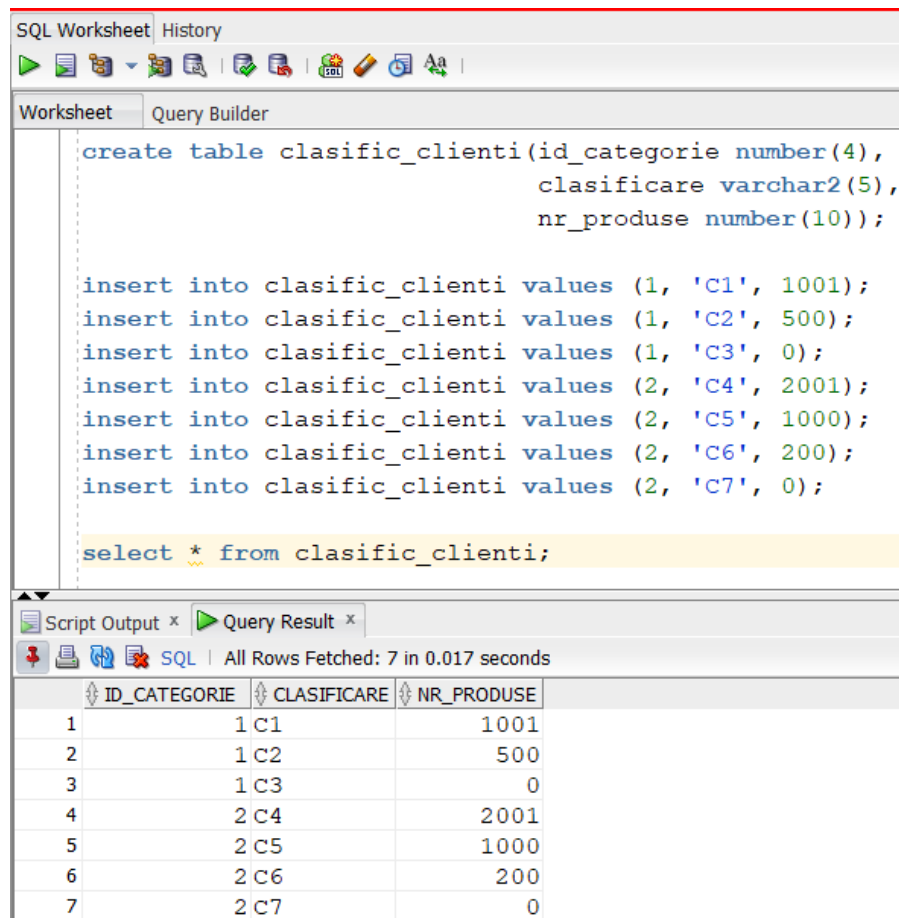
...

Varianta3

```
UPDATE clasific_clienti
SET    clasificare =
      CASE WHEN nr_produce>1000
            AND id_categorie=1 THEN 'A'
      WHEN nr_produce BETWEEN 500 AND 1000
            AND id_categorie = 1 THEN 'B'
      WHEN nr_produce BETWEEN 0 AND 499
            AND id_categorie=1 THEN 'C'
      WHEN nr_produce>2000
            AND id_categorie=2 THEN 'A'
      WHEN nr_produce BETWEEN 1000 AND 2000
            AND id_categorie = 2 THEN 'B'
      WHEN nr_produce BETWEEN 200 AND 999
            AND id_categorie=2 THEN 'C'
      ELSE 'D' END;
```

Rezolvare:

Creăm tabelul CLASIFIC_CLIENTI în SGBD-ul nostru:



The screenshot displays an SQL Worksheet interface. The main area contains SQL code for creating a table named 'clasific_clienti' with columns 'id_categorie' (number(4)), 'clasificare' (varchar2(5)), and 'nr_produce' (number(10)). It also includes seven INSERT statements for different categories and produce counts, and a SELECT statement to retrieve all data.

Below the code editor, the 'Query Result' tab is active, showing the results of the SELECT query. The results are presented in a table with three columns: ID_CATEGORIE, CLASIFICARE, and NR_PRODUSE. The data consists of seven rows, each representing a unique combination of category and produce count.

| ID_CATEGORIE | CLASIFICARE | NR_PRODUSE |
|--------------|-------------|------------|
| 1 | 1 C1 | 1001 |
| 2 | 1 C2 | 500 |
| 3 | 1 C3 | 0 |
| 4 | 2 C4 | 2001 |
| 5 | 2 C5 | 1000 |
| 6 | 2 C6 | 200 |
| 7 | 2 C7 | 0 |

Explain Plan ⇨ Varianta 1

SQL Worksheet History

0.15899999 seconds

Worksheet Query Builder

```
--Varianta 1
UPDATE clasific_clienti
SET clasificare = 'A'
WHERE nr_produce > 1000
AND id_categorie = 1;

UPDATE clasific_clienti
SET clasificare = 'B'
```

Script Output x Query Result x Explain Plan x

SQL 0.159 seconds

| OPERATION | OBJECT_NAME | OPTIONS | CARDINALITY | COST |
|---|------------------|---------|-------------|------|
| UPDATE STATEMENT | | | | 1 3 |
| UPDATE | CLASIFIC_CLIENTI | | | |
| TABLE ACCESS | CLASIFIC_CLIENTI | FULL | | 1 3 |
| Filter Predicates | | | | |
| AND | | | | |
| ID_CATEGORIE=1 | | | | |
| NR_PRODUCE>1000 | | | | |
| 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" | | | | |
| 133082358 | | | | |
| info type="plan_hash_2" | | | | |
| 1343190042 | | | | |
| {hint} | | | | |
| FULL(@"UPD\$1" "CLASIFIC_CLIENTI"@"UPD\$1") | | | | |
| OUTLINE_LEAF(@"UPD\$1") | | | | |
| ALL_ROWS | | | | |
| DB_VERSION('11.1.0.6') | | | | |
| OPTIMIZER_FEATURES_ENABLE('11.1.0.6') | | | | |
| IGNORE_OPTIM_EMBEDDED_HINTS | | | | |

Observăm costul unui singur update: 3. Având în vedere faptul că această variantă are nevoie de 7 update-uri, costul rezultat va fi 21.

Explain Plan ⇨ Varianta 2

SQL Worksheet | History

0.029 seconds

Worksheet | Query Builder

```
--Varianta 2
UPDATE clasific_clienti
SET clasificare = CASE WHEN nr_produce > 1000
                        THEN 'A'
                        WHEN nr_produce BETWEEN 500 AND 1000
                        THEN 'B'
                        ELSE 'C'
                        END
WHERE id_categorie = 1;
```

Script Output x | Query Result x | Query Result 1 x | Explain Plan x

SQL | 0.029 seconds

| OPERATION | OBJECT_NAME | OPTIONS | CARDINALITY | COST |
|------------------|------------------|---------|-------------|------|
| UPDATE STATEMENT | | | | 3 |
| UPDATE | CLASIFIC_CLIENTI | | | 3 |
| TABLE ACCESS | CLASIFIC_CLIENTI | FULL | | 3 |

Filter Predicates
ID_CATEGORIE=1

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"
    133082358
  info type="plan_hash_2"
    1343190042
  {hint}
    FULL(@"UPD$1" "CLASIFIC_CLIENTI"@"UPD$1")
    OUTLINE_LEAF(@"UPD$1")
    ALL_ROWS
    DB_VERSION('11.1.0.6')
    OPTIMIZER_FEATURES_ENABLE('11.1.0.6')
    IGNORE_OPTIM_EMBEDDED_HINTS
```

Observăm costul unui singur update: 3. Având în vedere faptul că această variantă are nevoie de 2 update-uri, costul rezultat va fi 6.

Explain Plan ⇨ Varianta 3

SQL Worksheet | History

0.026 seconds

Worksheet | Query Builder

```
--Varianta 3
UPDATE clasific_clienti
SET clasificare = CASE WHEN nr_produce>1000 AND id_categorie=1
                        THEN 'A'
                        WHEN nr_produce BETWEEN 500 AND 1000 AND id_categorie = 1
                        THEN 'B'
                        WHEN nr_produce BETWEEN 0 AND 499 AND id_categorie=1
                        THEN 'C'
                        WHEN nr_produce>2000 AND id_categorie=2
                        THEN 'A'
                        WHEN nr_produce BETWEEN 1000 AND 2000 AND id_categorie = 2
                        THEN 'B'
                        WHEN nr_produce BETWEEN 200 AND 999 AND id_categorie=2
```

Script Output x | Query Result x | Query Result 1 x | Explain Plan x

SQL | 0.026 seconds

| OPERATION | OBJECT_NAME | OPTIONS | CARDINALITY | COST |
|------------------|----------------------------------|---------|-------------|------|
| UPDATE STATEMENT | | | | 3 |
| UPDATE | CLASIFIC_CLIENTI | | 7 | 3 |
| TABLE ACCESS | CLASIFIC_CLIENTI | FULL | 7 | 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"
    133082358
  info type="plan_hash_2"
    1343190042
  {hint}
    FULL(@"UPD$1" "CLASIFIC_CLIENTI"@"UPD$1")
    OUTLINE_LEAF(@"UPD$1")
    ALL_ROWS
    DB_VERSION("11.1.0.6")
    OPTIMIZER_FEATURES_ENABLE("11.1.0.6")
    IGNORE_OPTIM_EMBEDDED_HINTS
```

Observăm costul unui singur update: 3. Având în vedere faptul că această variantă are nevoie doar de acest update, costul rezultat va fi 3.

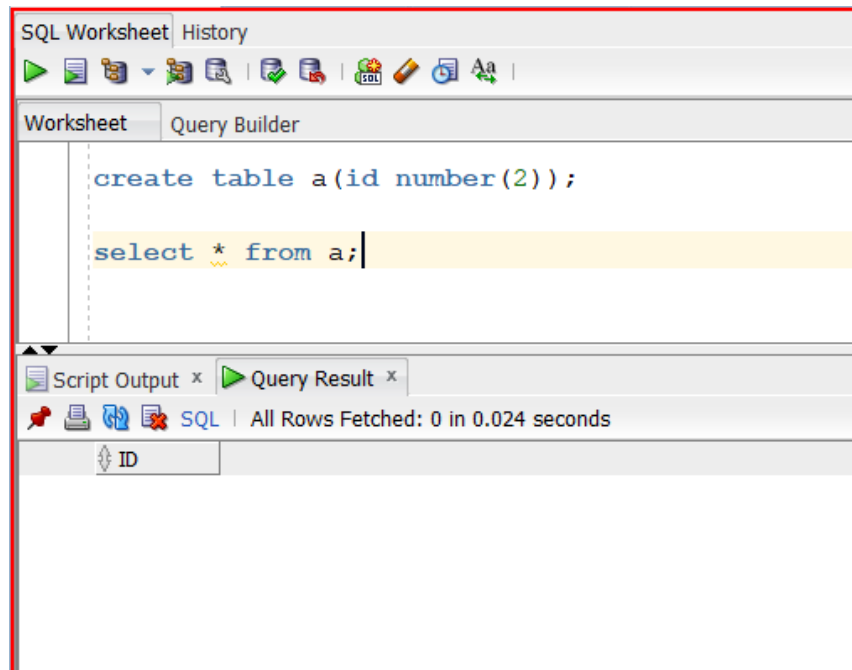
Așadar, conform costurilor calculate anterior, varianta optima este a treia.

Tablouri indexate (associative arrays/index-by tables)

Cerință: Generați excepția NO_DATA_FOUND pentru un tablou indexat, folosind și instrucțiunea SELECT ... INTO. Evidențiați locul unde apare excepția.

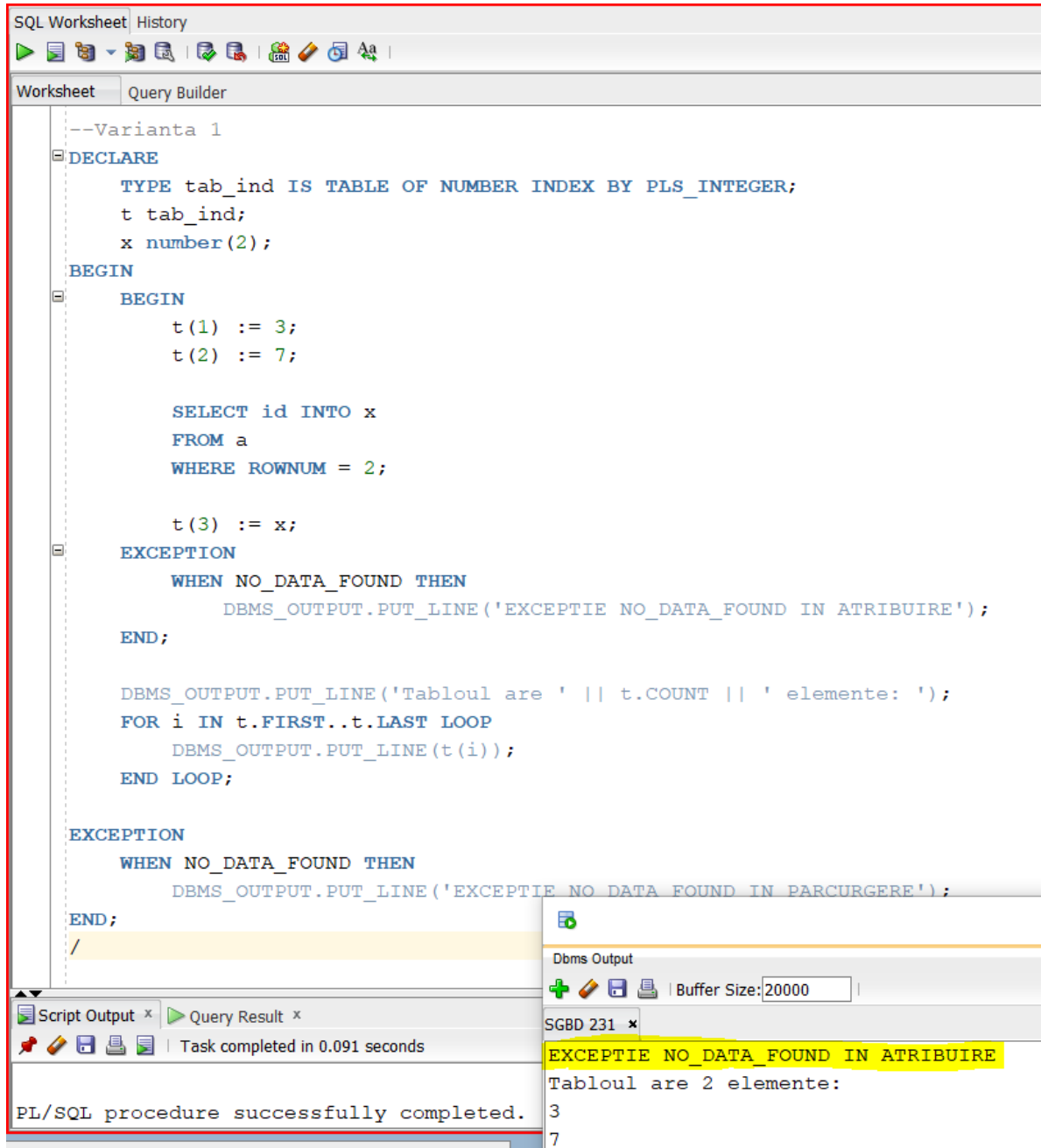
Varianta 1 (excepția apare în atribuire)

Creăm tabelul A cu un singur atribut: id. Nu inserăm nicio linie în tabel.



Ne creăm un tablou indexat care conține numere. Atribuim pentru pozițiile 1 și 2 ale tabloului valorile 3 și 7. Pentru poziția 3, încercăm să preluăm id-ul de la linia 2 din tabelul A care, desigur, nu există. Acest lucru ne conduce la excepția NO_DATA_FOUND în atribuirea elementelor.

Am folosit un bloc și un subbloc, deoarece astfel putem separa atribuirea elementelor de parcurgere, vizualizând eficient de unde apare excepția.



The screenshot displays an SQL Worksheet interface with a PL/SQL script and its execution results. The script, titled "--Varianta 1", declares an indexed table type and a variable, then attempts to populate it. It includes a subblock for attribute assignment and a main block for iteration. The execution output shows a "NO_DATA_FOUND" exception during the iteration phase.

```
--Varianta 1
DECLARE
  TYPE tab_ind IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  t tab_ind;
  x number(2);
BEGIN
  BEGIN
    t(1) := 3;
    t(2) := 7;

    SELECT id INTO x
    FROM a
    WHERE ROWNUM = 2;

    t(3) := x;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('EXCEPTIE NO_DATA_FOUND IN ATRIBUIRE');
  END;

  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT || ' elemente: ');
  FOR i IN t.FIRST..t.LAST LOOP
    DBMS_OUTPUT.PUT_LINE(t(i));
  END LOOP;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('EXCEPTIE NO DATA FOUND IN PARCURGERE');
END;
```

PL/SQL procedure successfully completed.

Dbms Output

SGBD 231

EXCEPTIE NO_DATA_FOUND IN ATRIBUIRE

Tabloul are 2 elemente:

3

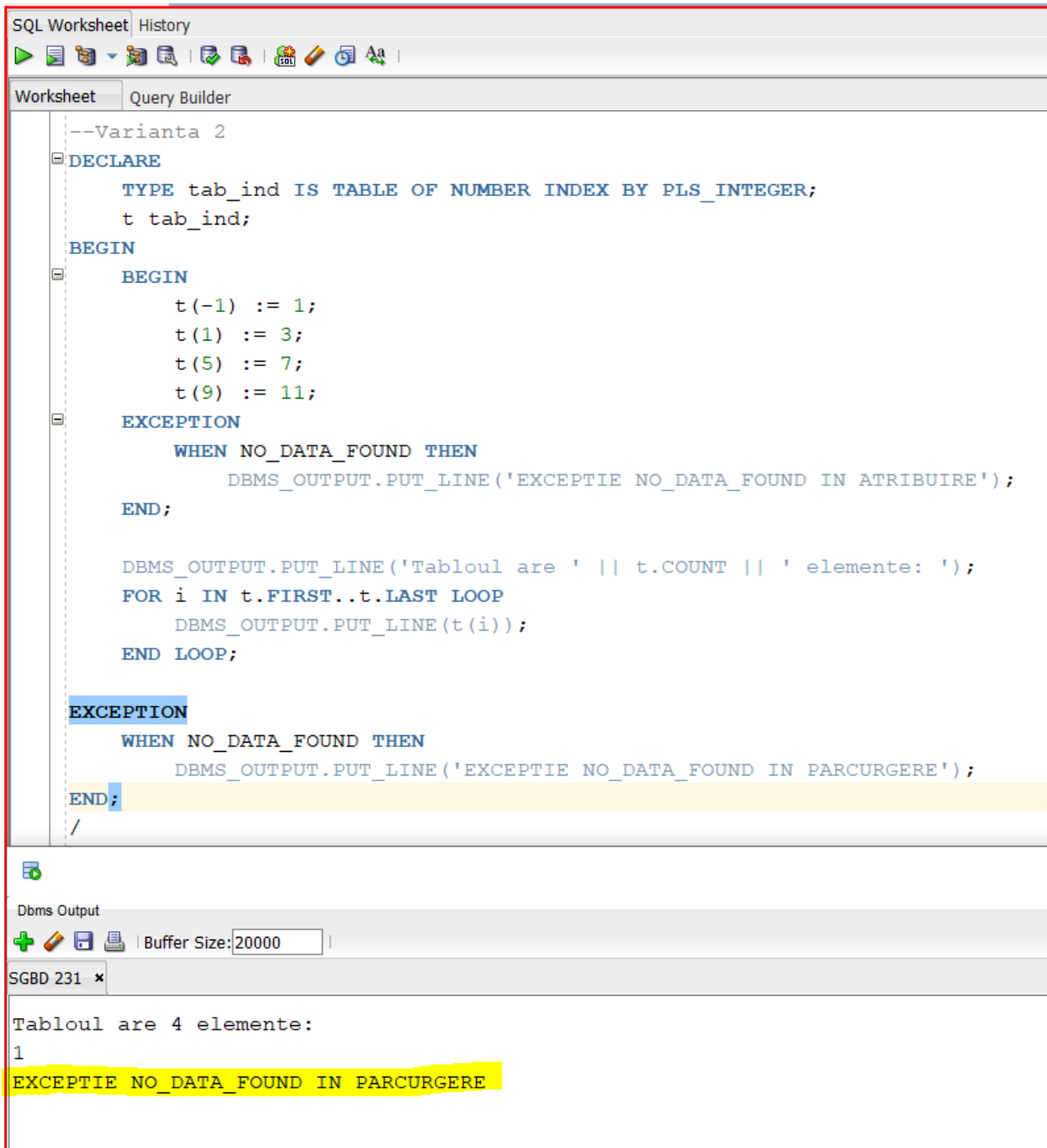
7

Rezolvarea acestei excepții se poate face în felul următor: adăugăm două înregistrări în tabelul A (pentru a putea exista linia 2).

Varianta 2 (exceptia apare în parcurgere)

Ne creăm un tablou indexat care conține numere. Atribuim valori pentru poziții neconsecutive: -1, 1, 5, 9. Atunci când dorim să afișăm elementele, utilizăm un FOR LOOP cu i-ul între prima poziție găsită în tablou și ultima poziție găsită în tablou. Acest lucru ne conduce la excepția NO_DATA_FOUND în parcurgerea elementelor.

Am folosit un bloc și un subbloc, deoarece astfel putem separa atribuirea elementelor de parcurgere, vizualizând eficient de unde apare excepția.



The screenshot displays an SQL Worksheet interface with a 'Worksheet' tab. The script is as follows:

```
--Varianta 2
DECLARE
    TYPE tab_ind IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
    t tab_ind;
BEGIN
    BEGIN
        t(-1) := 1;
        t(1) := 3;
        t(5) := 7;
        t(9) := 11;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('EXCEPTIE NO_DATA_FOUND IN ATRIBUIRE');
    END;

    DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT || ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(t(i));
    END LOOP;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('EXCEPTIE NO_DATA_FOUND IN PARCURGERE');
END;
```

The 'Dbms Output' window at the bottom shows the execution results:

```
Tabloul are 4 elemente:
1
EXCEPTIE NO_DATA_FOUND IN PARCURGERE
```


Rezolvarea acestei excepții se poate face în felul următor: în locul lui FOR LOOP, utilizăm WHILE LOOP alături de funcția NEXT().

The screenshot displays an SQL Worksheet interface with a PL/SQL procedure named 'Rezolvare VARIANTA 2'. The procedure uses a WHILE loop to traverse a table-like structure 'tab_ind' and output its elements. The output window shows the results of the procedure execution, listing the elements 1, 3, 7, and 11. A red checkmark is drawn next to the output.

```
--Rezolvare VARIANTA 2
DECLARE
    TYPE tab_ind IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
    t tab_ind;
    i PLS_INTEGER;
BEGIN
    BEGIN
        t(-1) := 1;
        t(1) := 3;
        t(5) := 7;
        t(9) := 11;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('EXCEPTIE NO_DATA_FOUND IN ATRIBUIRE');
    END;

    DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT || ' elemente: ');
    i := t.FIRST;
    WHILE i IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE(t(i));
        i := t.NEXT(i);
    END LOOP;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('EXCEPTIE NO_DATA_FOUND IN PARCURGERE');
END;
/
```

Dbms Output

SGBD 231 x

Tabloul are 4 elemente:

1
3
7
11

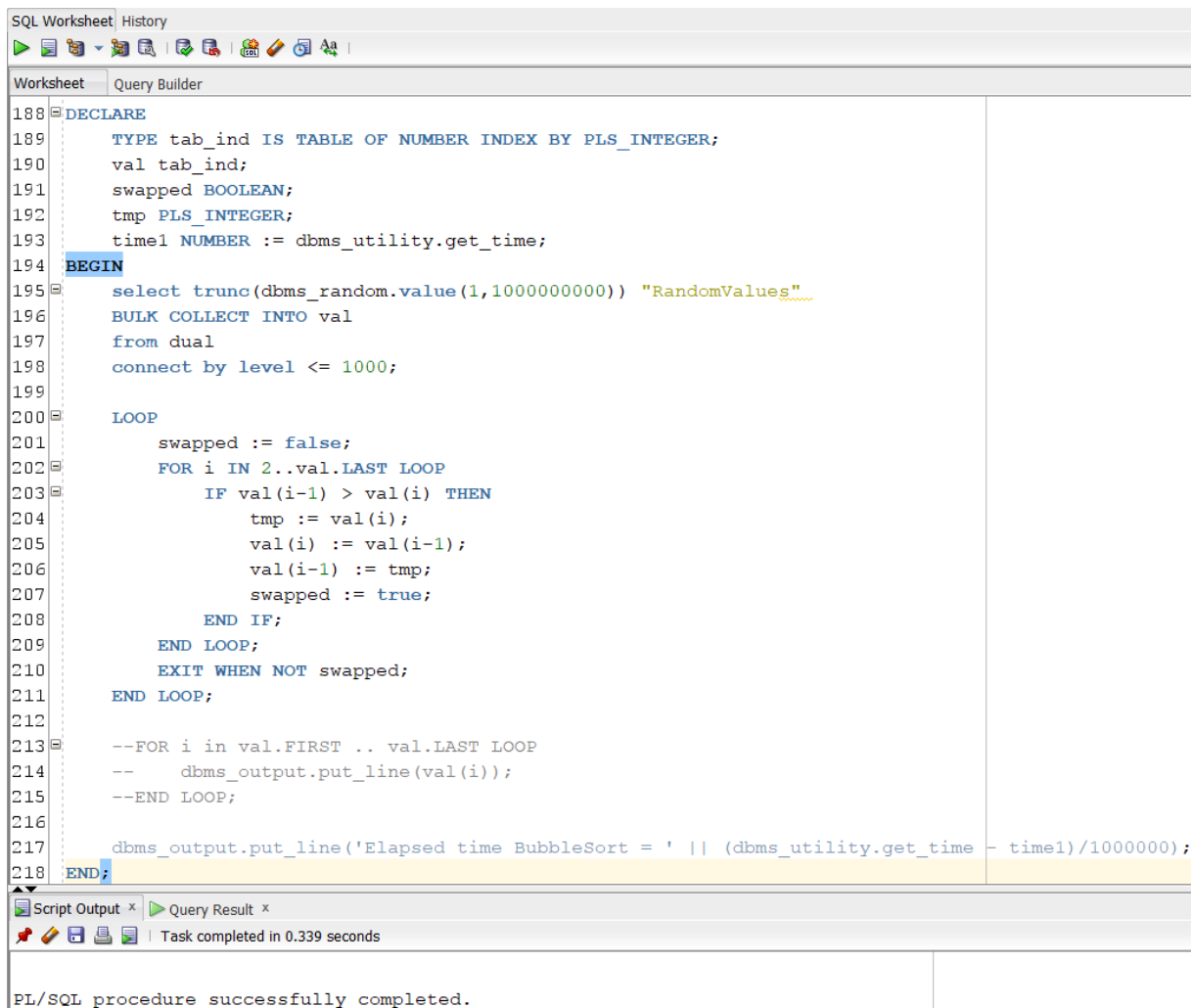
PL/SQL procedure successfully completed

Sortarea vectorilor

--Algorithm vs Order By--

Algorithm \Rightarrow Bubble Sort

Utilizăm PL/SQL pentru a implementa algoritmul de sortare Bubble Sort. Folosim pachetul DBMS_RANDOM pentru a genera 1 000 de numere random în intervalul 1-1 000 000 000 și pachetul DBMS_UTILITY pentru a genera timpul cu ajutorul funcției GET_TIME.



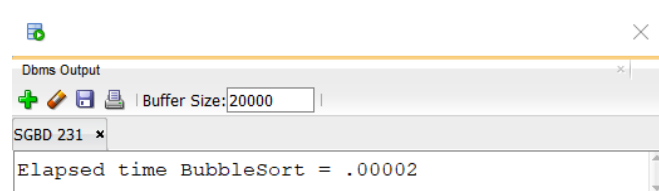
```
188 DECLARE
189     TYPE tab_ind IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
190     val tab_ind;
191     swapped BOOLEAN;
192     tmp PLS_INTEGER;
193     time1 NUMBER := dbms_utility.get_time;
194 BEGIN
195     select trunc(dbms_random.value(1,1000000000)) "RandomValues"
196     BULK COLLECT INTO val
197     from dual
198     connect by level <= 1000;
199
200     LOOP
201         swapped := false;
202         FOR i IN 2..val.LAST LOOP
203             IF val(i-1) > val(i) THEN
204                 tmp := val(i);
205                 val(i) := val(i-1);
206                 val(i-1) := tmp;
207                 swapped := true;
208             END IF;
209         END LOOP;
210         EXIT WHEN NOT swapped;
211     END LOOP;
212
213     --FOR i in val.FIRST .. val.LAST LOOP
214     --    dbms_output.put_line(val(i));
215     --END LOOP;
216
217     dbms_output.put_line('Elapsed time BubbleSort = ' || (dbms_utility.get_time - time1)/1000000);
218 END;
```

Script Output x Query Result x

Task completed in 0.339 seconds

PL/SQL procedure successfully completed.

Rezultat DBMS OUTPUT:



```
Dbms Output
+ + + Buffer Size: 20000
SGBD 231 x
Elapsed time BubbleSort = .00002
```

Order By

Utilizăm PL/SQL pentru a implementa putea evidenția diferența dintre cele două sortări. Folosim pachetul DBMS_RANDOM pentru a genera 1 000 de numere random în intervalul 1-1 000 000 000 și pachetul DBMS_UTILITY pentru a genera timpul cu ajutorul funcției GET_TIME.

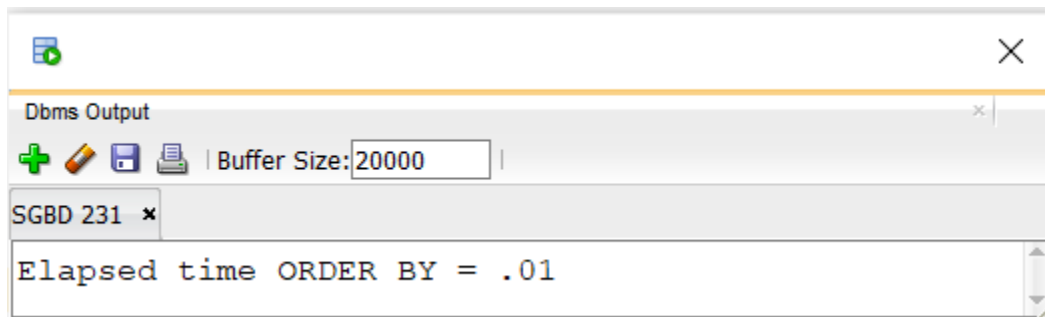
```
221 --Order By
222 DECLARE
223     TYPE tab_ind IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
224     val tab_ind;
225     swapped BOOLEAN;
226     tmp NUMBER(3);
227     time1 NUMBER := dbms_utility.get_time;
228 BEGIN
229     select trunc(dbms_random.value(1,1000000000)) "RandomValues"
230     BULK COLLECT INTO val
231     from dual
232     connect by level <= 1000
233     order by 1;
234
235     --FOR i in val.FIRST .. val.LAST LOOP
236     --     dbms_output.put_line(val(i));
237     --END LOOP;
238
239     dbms_output.put_line('Elapsed time ORDER BY = ' || (dbms_utility.get_time - time1)/100);
240 END;
241 /
```

Script Output x Query Result x

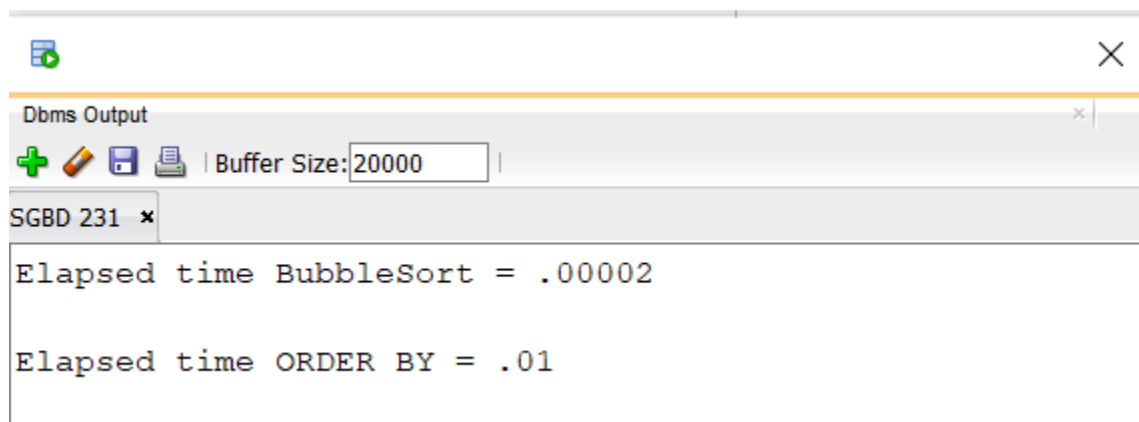
Task completed in 0.339 seconds

PL/SQL procedure successfully completed.

Rezultat DBMS OUTPUT:



Astfel, putem vedea *rezultatele ambelor sortări*:



The screenshot shows a 'Dbms Output' window with a 'Buffer Size' of 20000. It displays two lines of output: 'Elapsed time BubbleSort = .00002' and 'Elapsed time ORDER BY = .01'. The window title is 'SGBD 231'.

```
Elapsed time BubbleSort = .00002

Elapsed time ORDER BY = .01
```

Observăm că timpul de execuție al Bubble Sort-ului este mai mic decât cel al Order By-ului ($0.00002 < 0.01$).

În concluzie, Bubble Sort-ul este mai rapid decât Order By, ceea ce înseamnă că Order By-ul este foarte lent, având în vedere faptul că Bubble Sort este un algoritm de sortare destul de inefficient în comparație cu alții precum QuickSort, MergeSort, ș.a.m.d.