Tehnici de programare cu baze de date

#4 PL/SQL
Instrucţiuni în PL/SQL

https://www.runceanu.ro/adrian/activitate-didactica/

Curs 4

Instrucţiuni în PL/SQL

Cuprins

Instrucțiuni în PL/SQL

1. Instrucţiunea de atribuire
2. Instrucţiuni alternative
3. Instrucţiuni repetitive

Instrucțiuni în PL/SQL

Orice program poate fi scris utilizând structuri de control de bază care sunt combinate în diferite moduri pentru rezolvarea problemei propuse.

PL/SQL dispune de comenzi ce permit controlul execuţiei unui bloc.

Instrucţiuni PL/SQL:

1. de atribuire (**:=**)
2. condiţionale (***IF***, ***CASE***)
3. repetitive (***LOOP***, ***WHILE***, ***FOR***)
4. de salt (***GOTO***, ***EXIT***)
5. instrucţiunea vidă (***NULL***)

1. Instructiunea de atribuire

- Instrucţiunea de atribuire se realizează cu ajutorul operatorului de asignare (**:=**) şi are forma generală

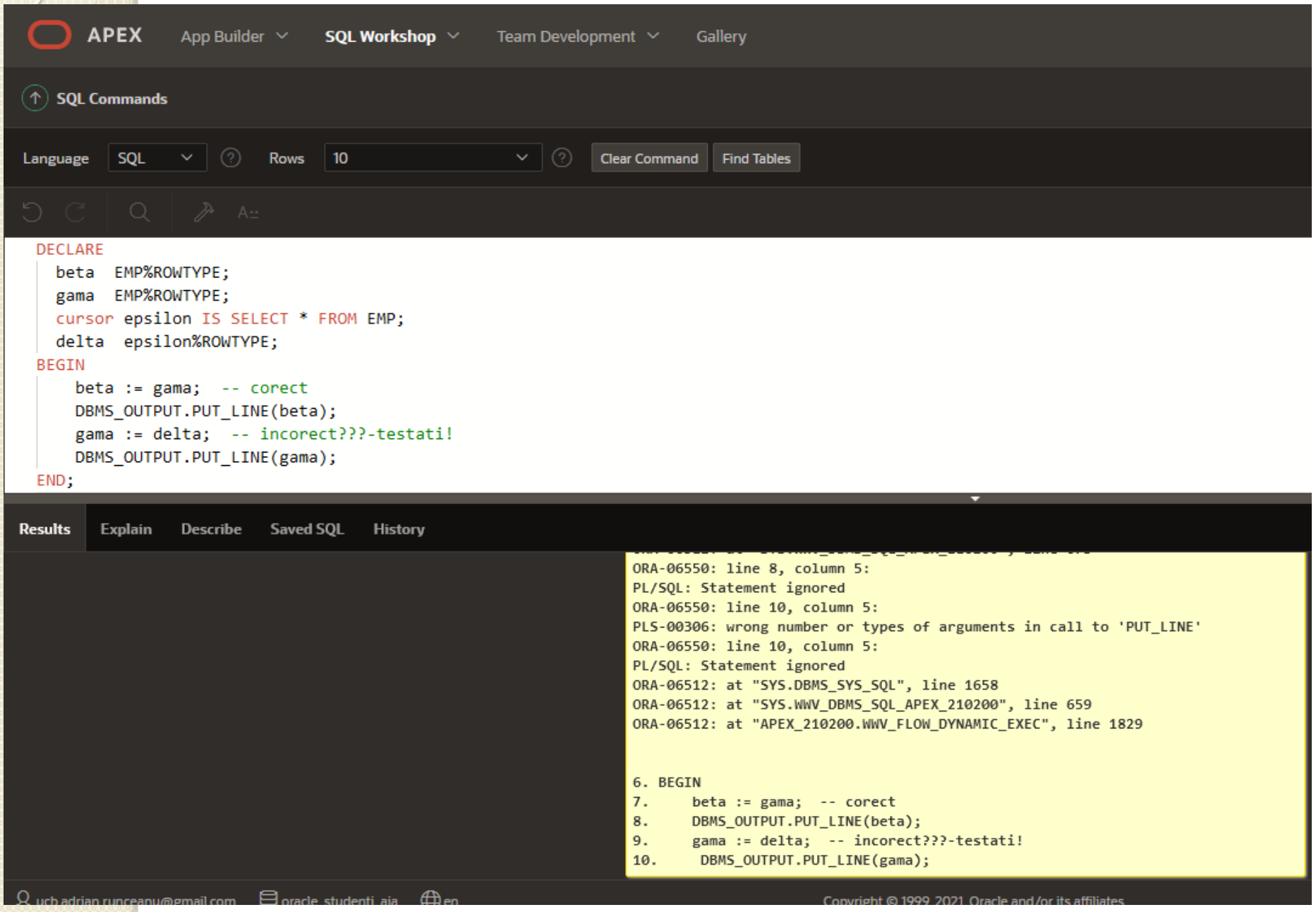
variabila := expresie

- Comanda respectă proprietăţile instrucţiunii de atribuire din clasa *LG3*.
- De remarcat că nu poate fi asignată valoarea *null* unei variabile care a fost declarată *NOT NULL*.

Exemplu:

Următorul exemplu prezintă modul în care acţionează instrucţiunea de atribuire în cazul unor tipuri de date particulare.

```
DECLARE
  beta  EMP%ROWTYPE;
  gama  EMP%ROWTYPE;
  cursor epsilon IS SELECT * FROM EMP;
  delta  epsilon%ROWTYPE;
BEGIN
    beta := gama;  -- corect
    DBMS_OUTPUT.PUT_LINE(beta);
    gama := delta;  -- incorect???-testati!
    DBMS_OUTPUT.PUT_LINE(gama);
END;
```



Cuprins

Instrucțiuni în PL/SQL

1. Instrucţiunea de atribuire
2. Instrucţiuni alternative
3. Instrucţiuni repetitive

2. Instructiuni alternative

- Un program *PL/SQL* poate executa diferite porţiuni de cod, în funcţie de rezultatul unui test.
- Instrucţiunile care realizează acest lucru sunt cele condiţionale (***IF***, ***CASE***).

- Structura instrucţiunii ***IF*** în *PL/SQL* este similară instrucţiunii ***IF*** din alte limbaje procedurale, permiţând efectuarea unor acţiuni în mod selectiv, în funcţie de anumite condiţii.

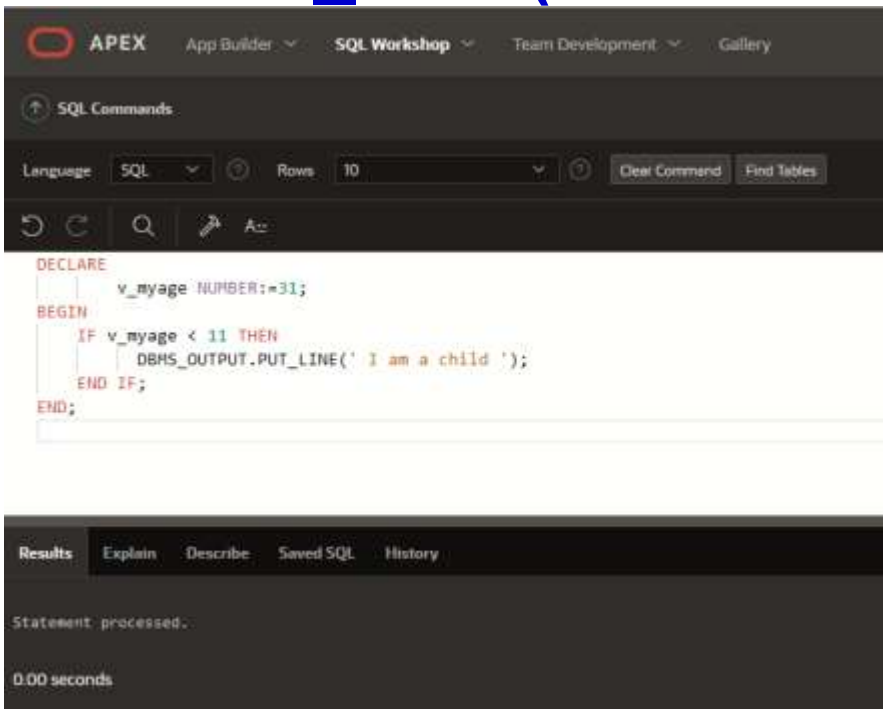
Instrucțiunea ***IF-THEN-ELSIF*** are următoarea formă sintactică:

```
IF condiție1 THEN
        secvența_de_comenzi_1
    [ELSIF condiție2 THEN
        secvența_de_comenzi_2]
    ...
    [ELSE
        secvența_de_comenzi_n]
    END IF;
```

- O secvenţă de comenzi din *IF* este executată numai în cazul în care condiţia asociată este *TRUE*.
- Atunci când condiţia este *FALSE* sau *NULL*, secvenţa nu este executată.
- Dacă pe ramura *THEN* se doreşte verificarea unei alternative, se foloseşte ramura *ELSIF* (atenţie, nu *ELSEIF*) cu o nouă condiţie.
- Este permis un număr arbitrar de opţiuni *ELSIF*, dar poate apărea cel mult o clauză *ELSE*. Aceasta se referă la ultimul *ELSIF*.

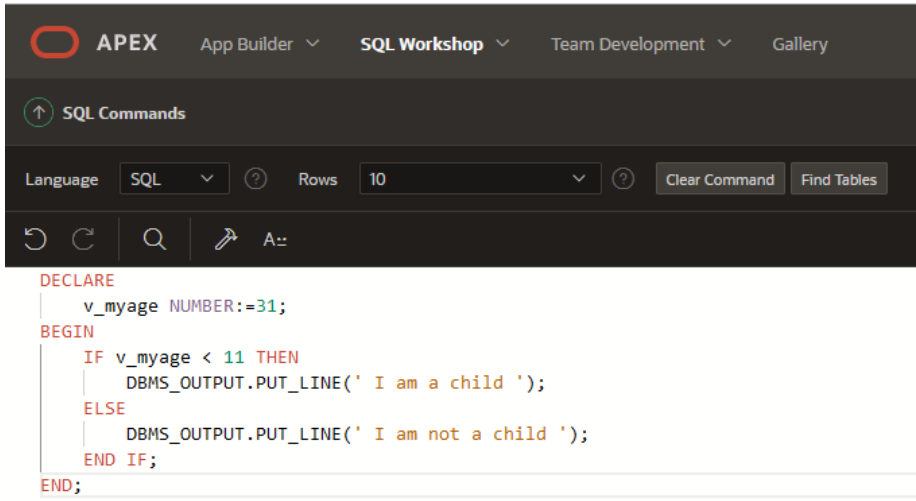
Instructiunea IF simpla

```
DECLARE
      v_myage NUMBER:=31;
BEGIN
  IF v_myage < 11 THEN
       DBMS_OUTPUT.PUT_LINE(' I am a
  child ');
  END IF;
END;
```



Instructiunea IF THEN ELSE


```
DECLARE
        v_myage NUMBER:=31;
BEGIN
   IF v_myage < 11 THEN
   DBMS_OUTPUT.PUT_LINE(' I am a child ');
   ELSE
   DBMS_OUTPUT.PUT_LINE(' I am not a child ');
   END IF;
END;
```



Clauza IF ELSIF ELSE

```
DECLARE
        v_myage NUMBER:=31;
BEGIN
   IF v_myage < 11 THEN
        DBMS_OUTPUT.PUT_LINE('I
   am a child');
   ELSIF v_myage < 20 THEN
        DBMS_OUTPUT.PUT_LINE('I
   am young');
        ELSIF v_myage < 30 THEN
        DBMS_OUTPUT.PUT_LINE('I
   am in my twenties');
          ELSIF v_myage < 40 THEN

        DBMS_OUTPUT.PUT_LINE('I
   am in my thirties');
                    ELSE
        DBMS_OUTPUT.PUT_LINE('I
   am always young ');
   END IF;
END;
```

APEX App Builder SQL Workshop Team Development Gallery

SQL Commands

Language SQL Rows 10 Clear Command Find Tables

```
DECLARE
        v_myage NUMBER:=31;
BEGIN
    IF v_myage < 11 THEN
        DBMS_OUTPUT.PUT_LINE('I am a child');
    ELSIF v_myage < 20 THEN
        DBMS_OUTPUT.PUT_LINE('I am young');
        ELSIF v_myage < 30 THEN
        DBMS_OUTPUT.PUT_LINE('I am in my twenties');
          ELSIF v_myage < 40 THEN          DBMS_OUTPUT.PUT_LINE('I am in my thirties');
            ELSE
        DBMS_OUTPUT.PUT_LINE('I am always young ');
    END IF;
END;
```

Results Explain Describe Saved SQL History

```
I am in my thirties

Statement processed.

0.01 seconds
```

- Daca instructiunea IF contine mai multe clauze si o conditie este evaluata ca FALSE sau NULL, atunci controlul se transfera urmatoarei clauze.

- Conditiile sunt evaluate una cate una incepand cu prima.

- Daca toate conditiile sunt FALSE sau NULL, atunci sunt executate instructiunile din clauza ELSE.

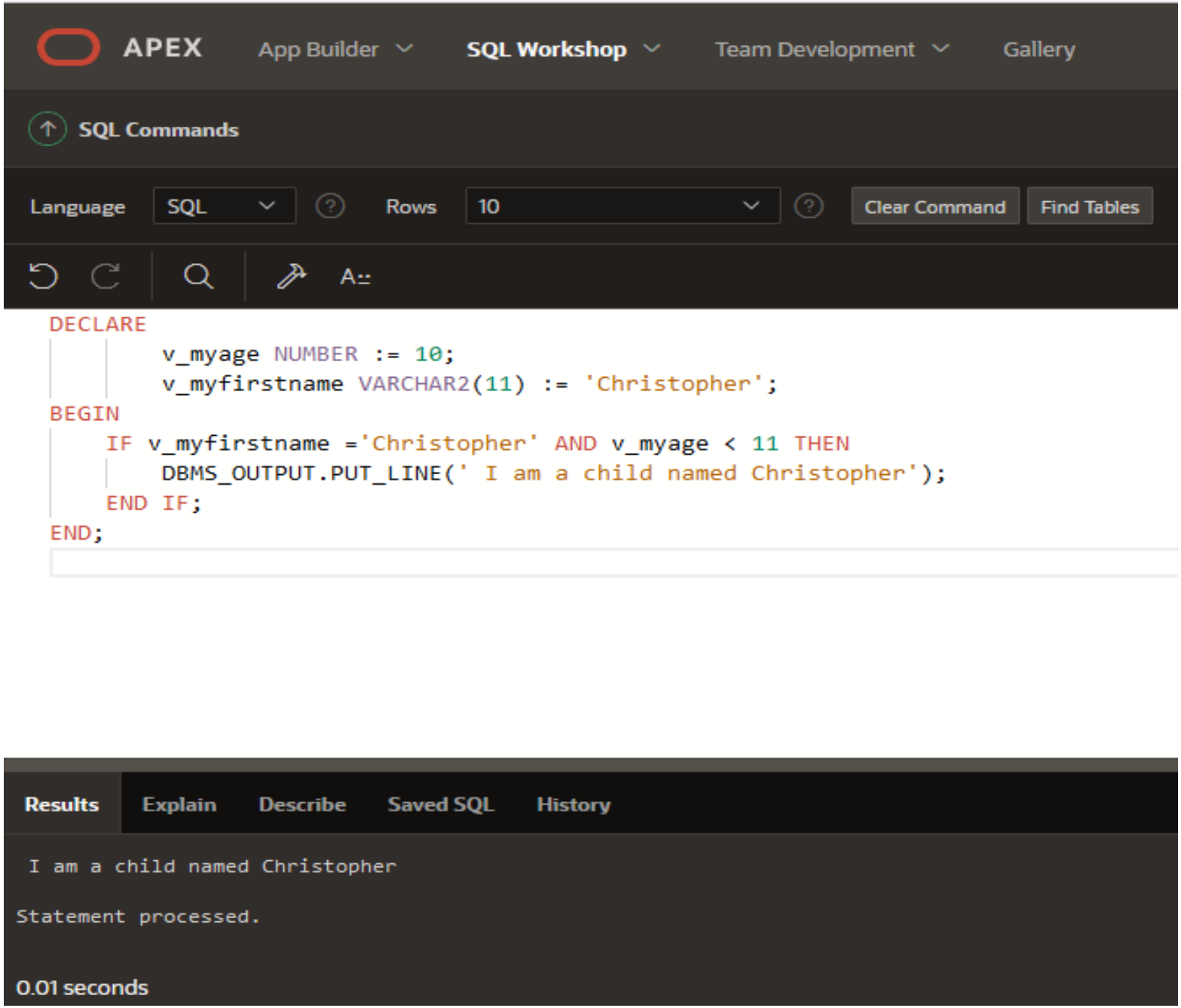
- Clauza ELSE este optionala.

Instructiunea IF cu expresii multiple

O instructiune IF poate avea multiple expresii conditionale legate prin operatori logici cum ar fi: **AND, OR, NOT**.

De exemplu:

```
DECLARE
        v_myage NUMBER := 10;
        v_myfirstname VARCHAR2(11) :=
   'Christopher';
BEGIN
   IF v_myfirstname ='Christopher' AND v_myage <
   11 THEN
        DBMS_OUTPUT.PUT_LINE(' I am a child
   named Christopher');
   END IF;
END;
```



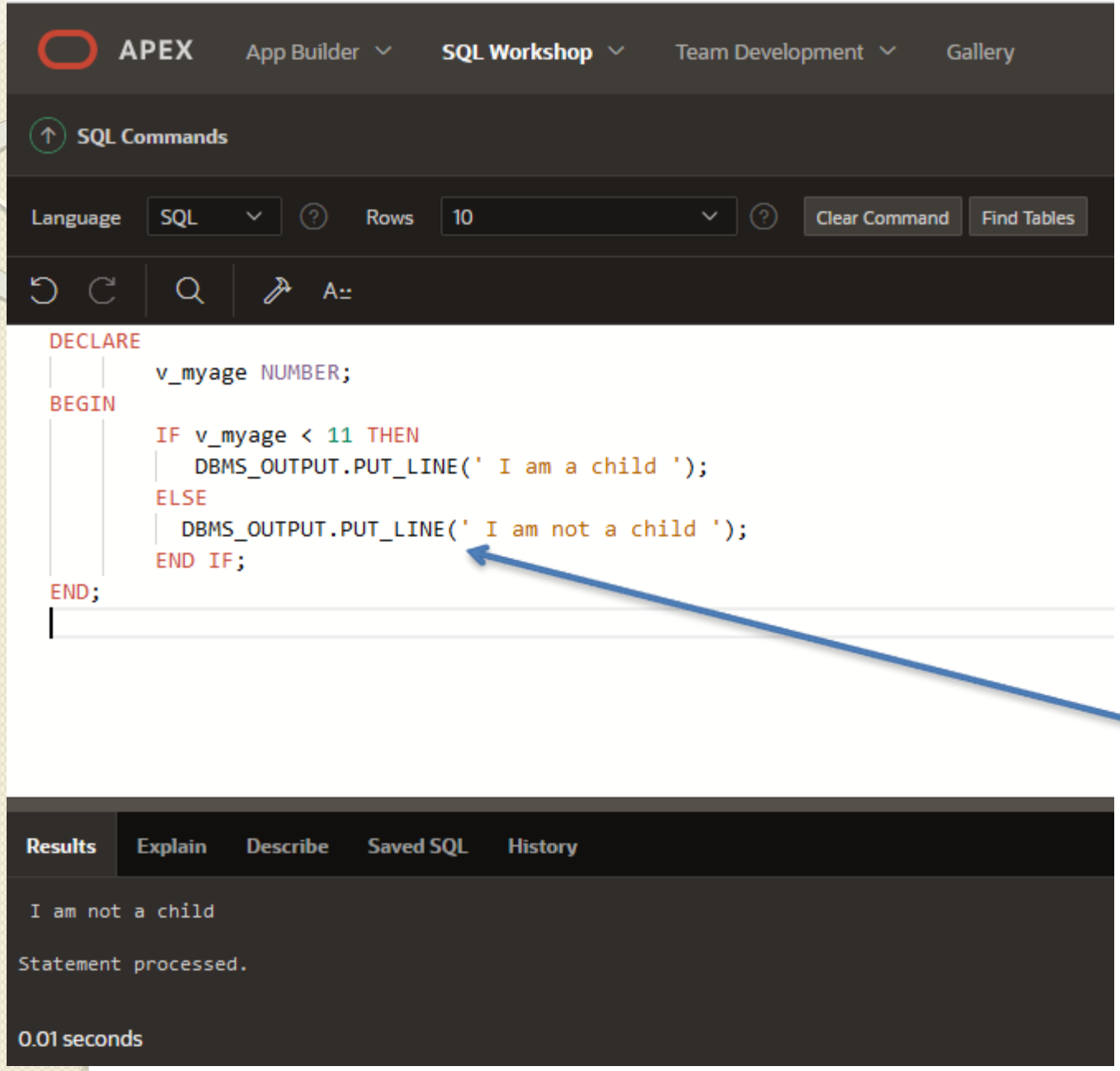
Valorile NULL in instructiunile IF

In urmatorul exemplu variabila ***v_myage*** este declarata dar nu este initializata.

Conditia din instructiunea IF intoarce NULL si nu TRUE sau FALSE.

In acest caz controlul este preluat de ELSE deoarece exact ca si FALSE, NULL nu este TRUE.

```
DECLARE
        v_myage NUMBER;
BEGIN
        IF v_myage < 11 THEN
          DBMS_OUTPUT.PUT_LINE(' I am a child ');
        ELSE
         DBMS_OUTPUT.PUT_LINE(' I am not a child ');
        END IF;
END;
```



Valoarea NULL preda controlul pe ramura ELSE

Folosirea valorilor NULL

- Atunci cand lucram cu valorile ***null*** putem evita cateva greseli uzuale retinand urmatoarele reguli:
  1. Comparatiile obisnuite care implica ***null*** intotdeauna produc ***null***
  2. Aplicand operatorul logic **NOT** unui ***null*** produce ***null***
  3. In instructiunile conditionale daca o conditie produce ***null*** acesta se comporta ca si cand ar fi **FALSE** si secventa de instructiuni asociata nu se executa.

Fie urmatorul exemplu:

```
x := 5;
y := NULL;
……
IF x != y THEN ……. ----- Rezultatul este
   NULL deci nu este TRUE si secventa de
   instructiuni nu se executa
END IF;
```

Conditia de la IF produce NULL si secventa de instructiuni nu se executa.

Fie urmatorul exemplu:

```
a := NULL;
b := NULL;
……
IF a = b THEN … --- returneaza NULL
END IF;
```

Rezultatul este NULL, deci nu se executa secventa de instructiuni.

Concluzii – instr. IF

- ***Reguli de utilizare a instructiunilor IF***

1. Se pot efectua operatii selectiv atunci cand este intalnita o conditie anume
2. Atunci cand scriem codul trebuie sa ne amintim cum se scriu cuvintele cheie
   - ***ELSIF*** – un singur cuvant
   - ***END IF*** – doua cuvinte
3. Daca conditia de control booleana este TRUE atunci se executa secventa de instructiuni asociata; daca conditia de control booleana este FALSE sau NULL atunci secventa de instructiuni nu se executa.
4. Sunt permise oricate clauze ELSIF.

Instructiunea CASE

INSTRUCTIUNEA CASE

- **Instructiunile CASE** sunt asemanatoare cu instructiunile IF, dar de obicei sunt mai usor de scris si de citit.
- **Expresiile CASE** sunt functii care intorc o valoare numerica intr-o variabila.

Daca avem urmatoarea instructiune IF, ce observam?

```
DECLARE
        v_numvar NUMBER;
BEGIN
……
   IF v_numvar = 5 THEN statement_1; statement_2;
   ELSIF v_numvar = 10 THEN statement_3;
   ELSIF v_numvar = 12 THEN statement_4; statement_5;
   ELSIF v_numvar = 27 THEN statement_6;
   ELSIF v_numvar ... ….
   ELSE statement_15;
   END IF;
……..
END;
```

Toate conditiile testeaza aceeasi variabila *v_numvar*.

Si codul este repetitiv – variabila *v_numvar* este folosita de multe ori.

Pentru a face acelasi lucru se poate folosi urmatoarea ***instructiune CASE***:

```
DECLARE
        v_numvar NUMBER;
BEGIN
……….
        CASE v_numvar
                WHEN 5 THEN statement_1; statement_2;
                WHEN 10 THEN statement_3;
                WHEN 12 THEN statement_4; statement_5;
                WHEN 27 THEN statement_6;
                WHEN ...
                ELSE statement_15;
        END CASE;
…….
END;
```

Evident este mai usor de scris, iar variabila ***v_numvar*** este referita o singura data.

Expresiile CASE

Uneori vrem sa atribuim o valoare unei variabile care depinde de valoarea altei variabile.

```
DECLARE
        v_out_var VARCHAR2(15);
        v_in_var NUMBER;
BEGIN
.....................
        IF v_in_var = 1 THEN v_out_var := 'Low value';
        ELSIF v_in_var = 50 THEN v_out_var :=
   'Middle value';
        ELSIF v_in_var = 99 THEN v_out_var := 'High
   value';
        ELSE v_out_var := 'Other value';
        END IF;
END;
```

Din nou codul este foarte repetitiv.

Acelasi lucru se poate realiza cu urmatoarea expresie CASE.

```
DECLARE
        v_out_var VARCHAR2(15);
        v_in_var NUMBER;
BEGIN
……..
        v_out_var :=
                CASE v_in_var
                        WHEN 1 THEN 'Low value'
                        WHEN 50 THEN 'Middle value'
                        WHEN 99 THEN 'High value'
                        ELSE 'Other value'
                END;
……..
END;
```

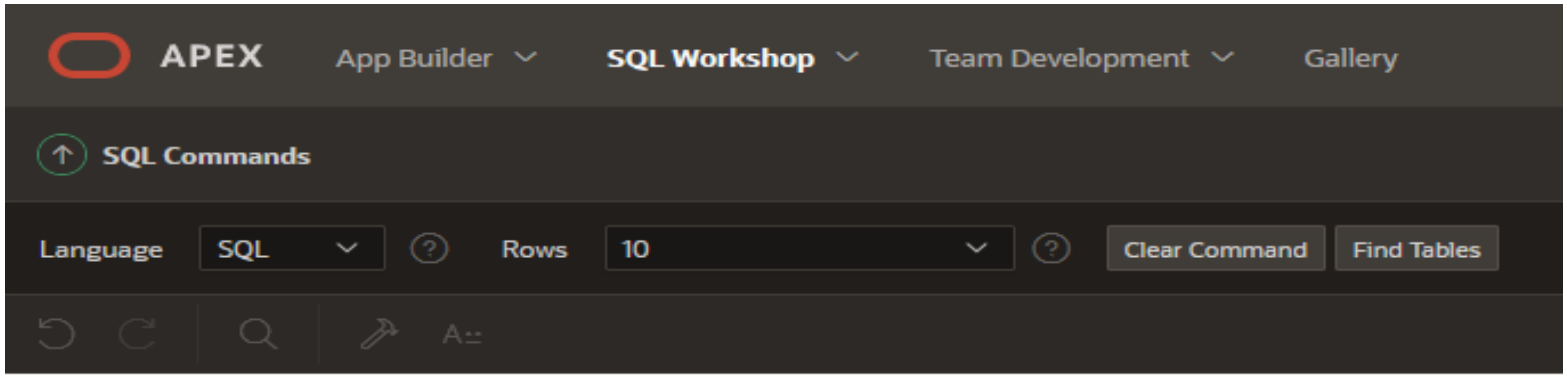
*O **expresie CASE** selecteaza un rezultat dintr-o serie de rezultate si-l returneaza intr-o variabila.*

```
variable_name :=
CASE selector
        WHEN expression1 THEN result1
        WHEN expression2 THEN result2
                ..............
        WHEN expressionN THEN resultN
        [ELSE resultN+1]
END;
```

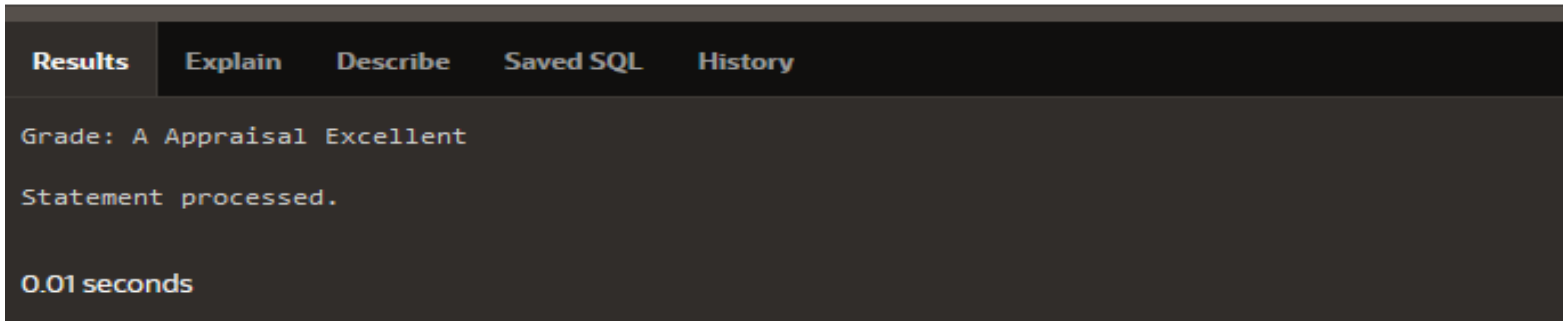
EXEMPLU 1

```
DECLARE
        v_grade CHAR(1) := 'A';
        v_appraisal VARCHAR2(20);
BEGIN
   v_appraisal :=
        CASE v_grade
                WHEN 'A' THEN 'Excellent'
                WHEN 'B' THEN 'Very Good'
                WHEN 'C' THEN 'Good'
                ELSE 'No such grade'
        END;
    DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade ||'
   Appraisal ' || v_appraisal);
END;
```

Ce se afiseaza?



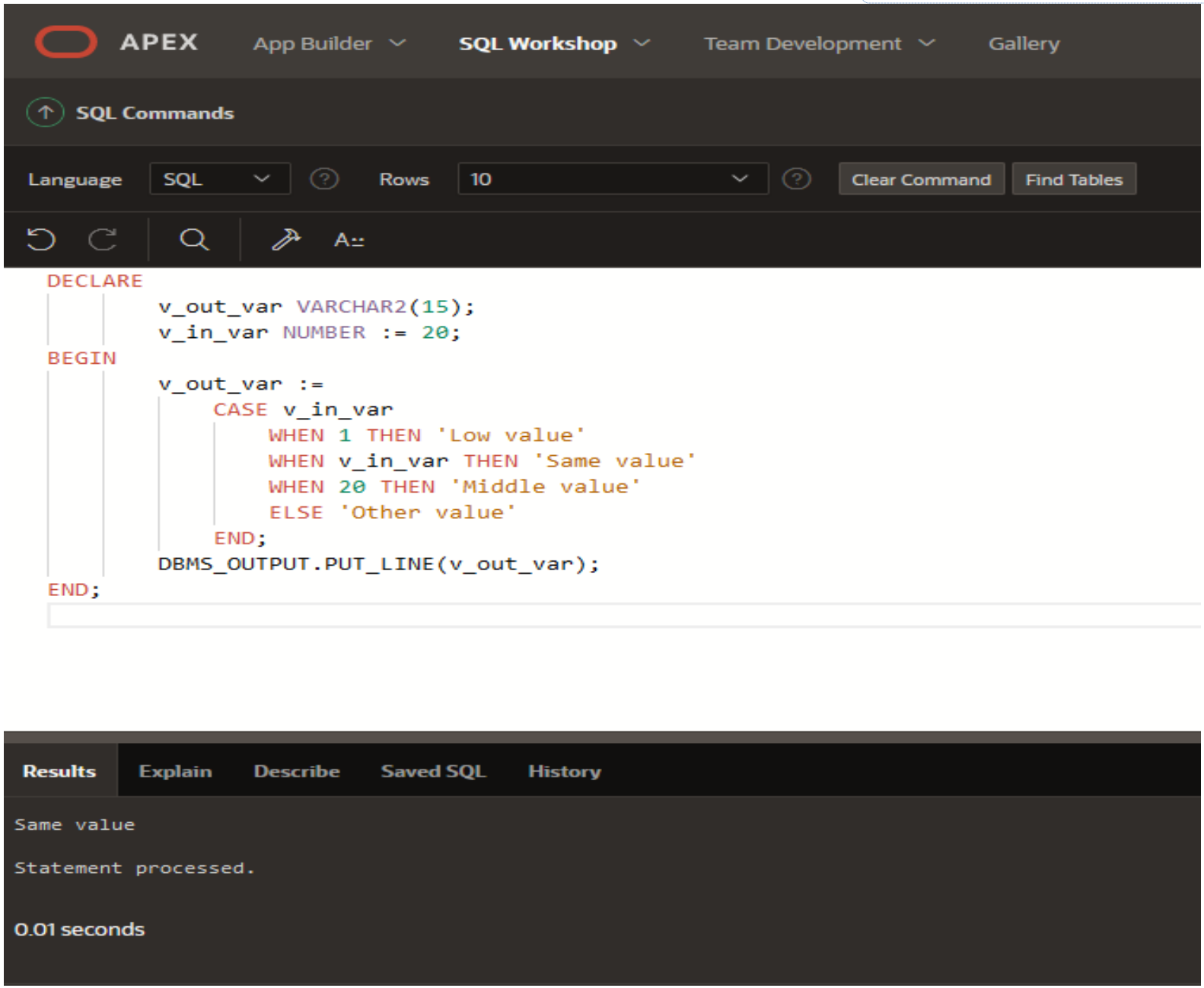

```
DECLARE
        v_grade CHAR(1) := 'A';
        v_appraisal VARCHAR2(20);
BEGIN
    v_appraisal :=
        CASE v_grade
            WHEN 'A' THEN 'Excellent'
            WHEN 'B' THEN 'Very Good'
            WHEN 'C' THEN 'Good'
            ELSE 'No such grade'
        END;
     DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade ||' Appraisal ' || v_appraisal);
END;
```



EXEMPLU 2

```
DECLARE
        v_out_var VARCHAR2(15);
        v_in_var NUMBER := 20;
BEGIN
        v_out_var :=
                CASE v_in_var
                        WHEN 1 THEN 'Low value'
                        WHEN v_in_var THEN 'Same value'
                        WHEN 20 THEN 'Middle value'
                        ELSE 'Other value'
                END;
        DBMS_OUTPUT.PUT_LINE(v_out_var);
END;
```

Ce se afiseaza?



Expresiile CASE de cautare

PL/SQL furnizeaza expresia CASE de cautare care are urmatoarea forma:

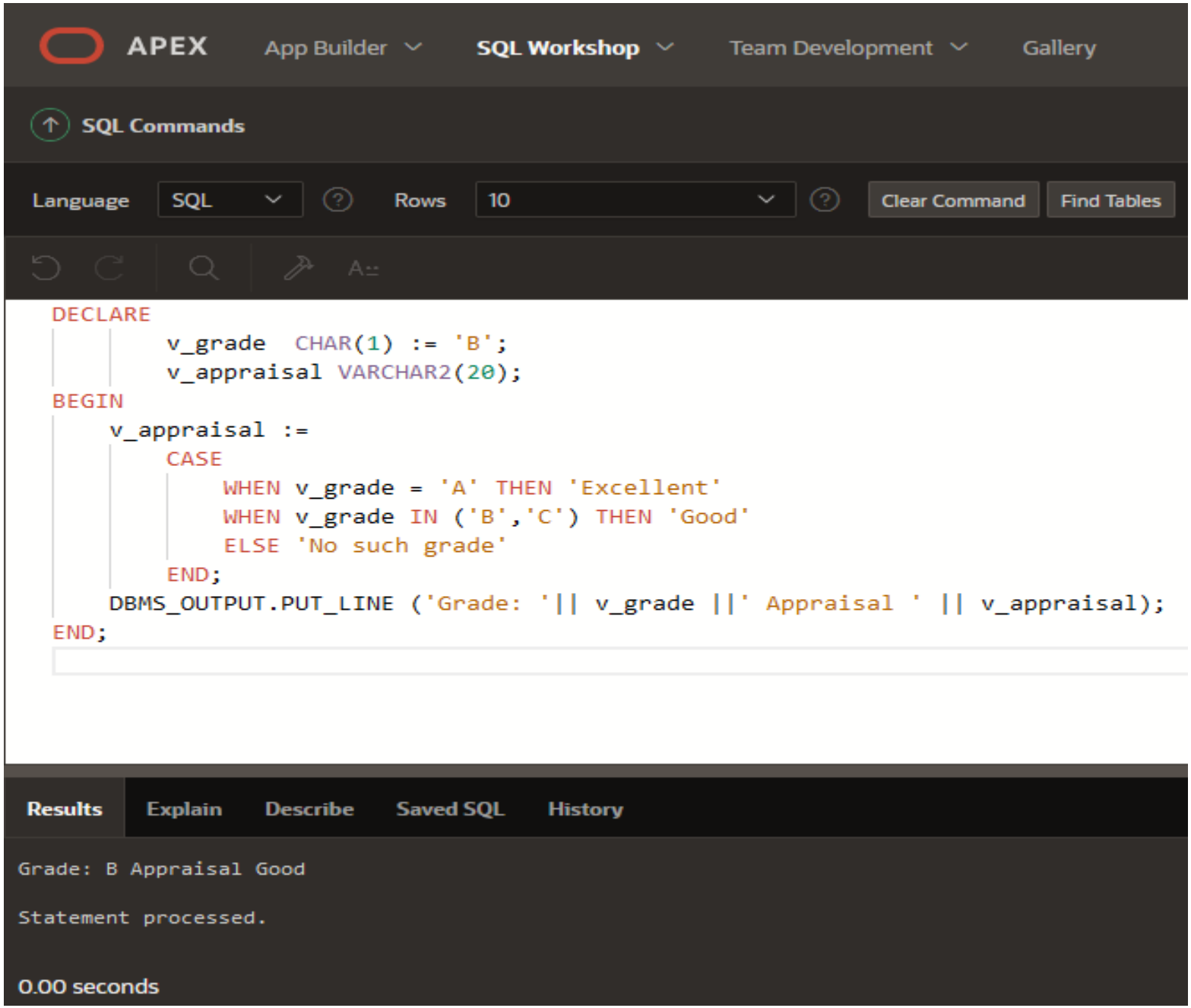
```
CASE
      WHEN search_condition1 THEN result1
      WHEN search_condition2 THEN result2
      ..........................
      WHEN search_conditionN THEN resultN
      [ELSE resultN+1]
END;
```

O expresie CASE de cautare nu are selector.

De asemenea, clauzele WHEN contin conditii de cautare care au valoare booleana, nu expresii care dau valori de orice tip.

EXEMPLU

```
DECLARE
        v_grade  CHAR(1) := 'B';
        v_appraisal VARCHAR2(20);
BEGIN
   v_appraisal :=
        CASE
                WHEN v_grade = 'A' THEN 'Excellent'
                WHEN v_grade IN ('B','C') THEN 'Good'
                ELSE 'No such grade'
        END;
   DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade ||' Appraisal '
   || v_appraisal);
END;
```



*Prin ce difera **expresiile CASE de instructiunile CASE**?*

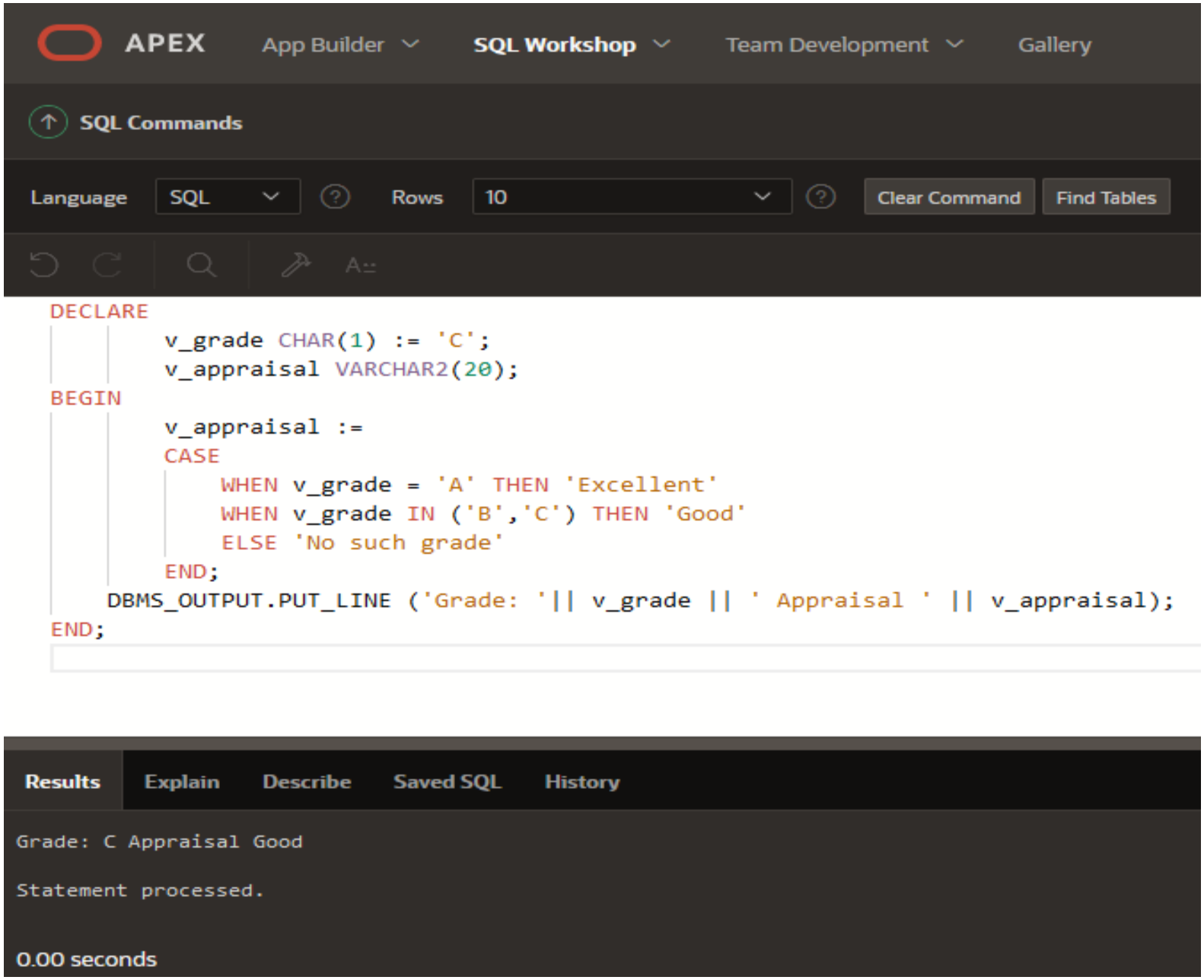
1) Expresiile CASE:

a) Expresiile CASE returneaza o valoare intr-o variabila

b) Expresiile CASE se incheie cu END;

c) O expresie CASE este o instructiune ***PL/SQL*** singulara

```
DECLARE
        v_grade CHAR(1) := 'C';
        v_appraisal VARCHAR2(20);
BEGIN
        v_appraisal :=
        CASE
                WHEN v_grade = 'A' THEN
  'Excellent'
                WHEN v_grade IN ('B','C') THEN
  'Good'
                ELSE 'No such grade'
        END;
  DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade
  || ' Appraisal ' || v_appraisal);
END;
```



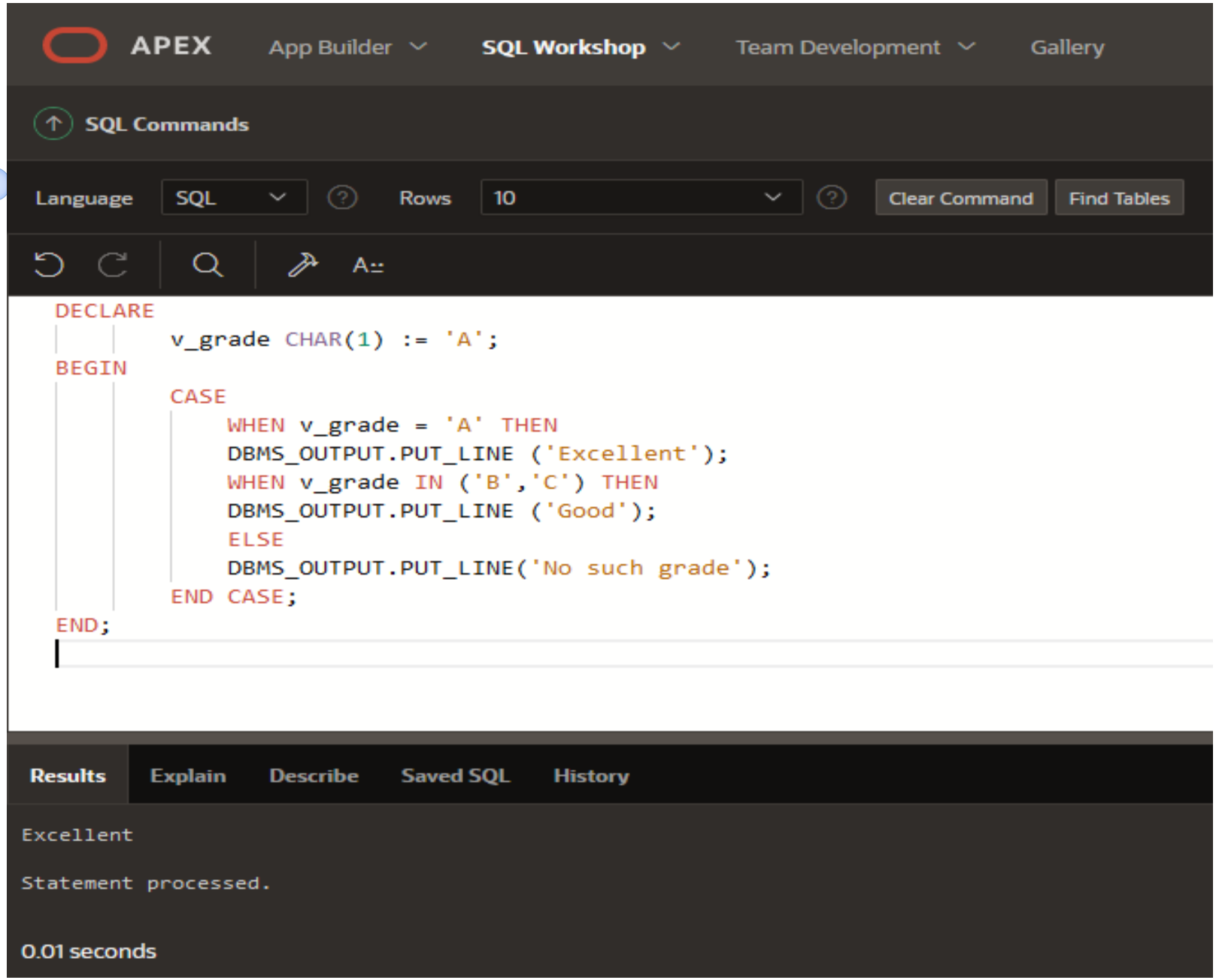
2) Instructiunile CASE:

a) Instructiunile CASE evalueaza conditii si executa operatii

b) O instructiune CASE poate contine mai multe instructiuni *PL/SQL*

c) Instructiunile CASE se incheie cu END CASE

```
DECLARE
        v_grade CHAR(1) := 'A';
BEGIN
        CASE
                WHEN v_grade = 'A' THEN
                DBMS_OUTPUT.PUT_LINE ('Excellent');
                WHEN v_grade IN ('B','C') THEN
                DBMS_OUTPUT.PUT_LINE ('Good');
                ELSE
                DBMS_OUTPUT.PUT_LINE('No such
  grade');
        END CASE;
END;
```



TABELE LOGICE

AND	TRUE	FALSE	NULL	**OR**	TRUE	FALSE	NULL	**NOT**	
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL

Cuprins

Instrucțiuni în PL/SQL

1. Instrucţiunea de atribuire
2. Instrucţiuni alternative
3. Instrucţiuni repetitive

3. Instructiuni repetitive

Structura repetitiva presupune repetarea unor operatii pana cand se ajunge la o conditie de oprire.

PL/SQL are trei tipuri de structuri repetitive:

1) **Instructiunea LOOP** de baza (basic LOOP) – *repeta operatii fara o conditie generala*
2) **Instructiunea FOR** – *repetarea unor operatii pe baza unui contor*
3) **Instructiunea WHILE** – *repetarea unor operatii pe baza unei conditii*

3.1.Instructiunea LOOP

Basic LOOP (structura repetitiva de baza)

- Cea mai simpla forma a unei instructiuni LOOP este **basic LOOP** care cuprinde o secventa de instructiuni intre cuvintele cheie ***LOOP*** si ***END LOOP***.
- In aceasta instructiune secventa de instructiuni se va executa cel putin o data.
- De fiecare data cand executia ajunge la ***END LOOP*** controlul este returnat instructiunii ***LOOP*** corespunzatoare de mai sus.

3.1.Instructiunea LOOP

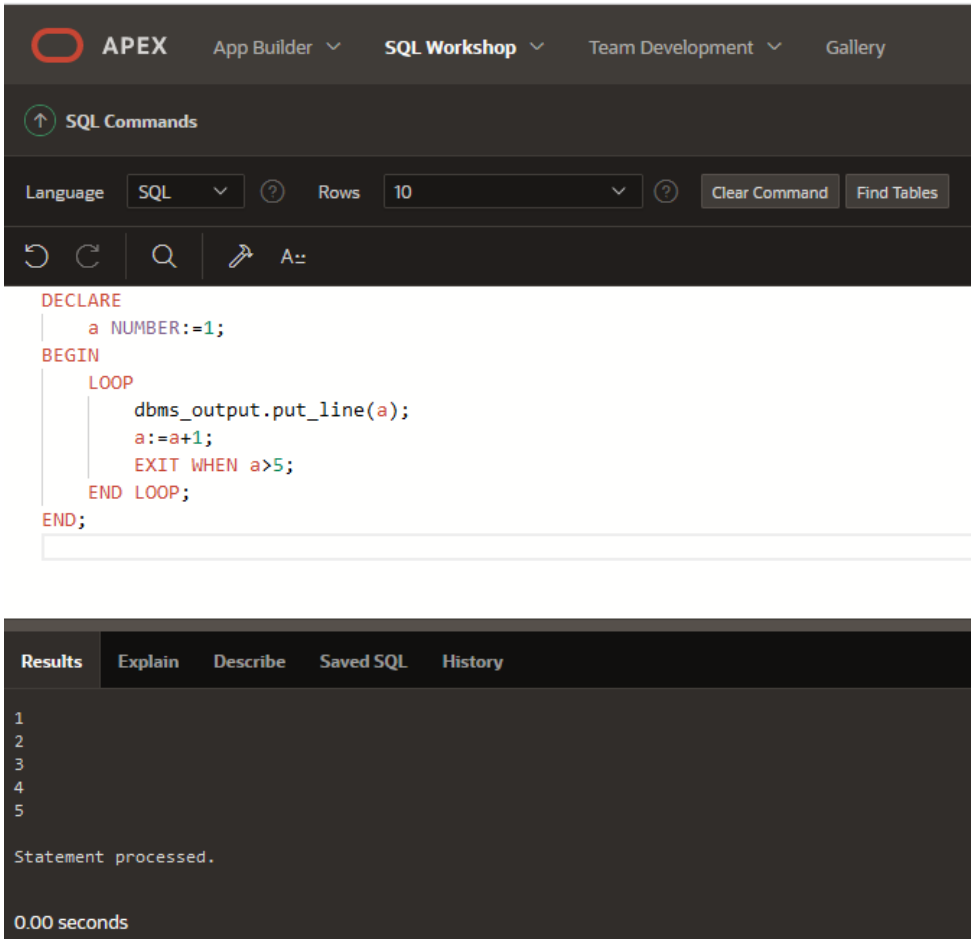
Sintaxa:

```
LOOP
        statement1;
        ..............
        EXIT [WHEN condition];
END LOOP;
```

- Un basic loop permite executarea instructiunilor sale cel putin o data chiar daca conditia ***EXIT*** este deja intalnita la intrarea in bucla.
- Fara instructiunea ***EXIT*** instructiunea ***LOOP*** ar fi infinita.

Exemplu:

```
DECLARE
   a NUMBER:=1;
BEGIN
   LOOP
      dbms_output.put_line(a);
      a:=a+1;
      EXIT WHEN a>5;
   END LOOP;
END;
```

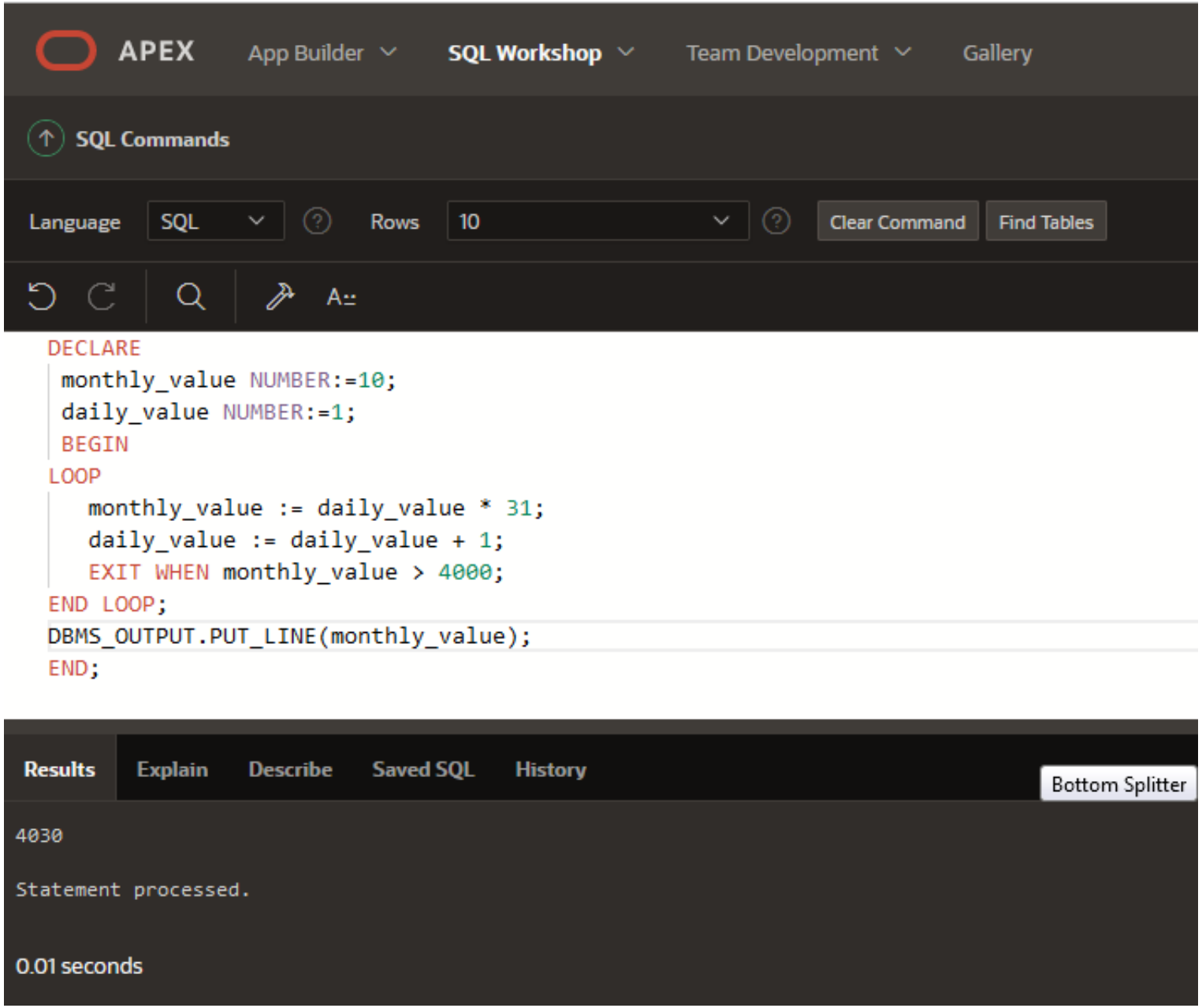


Instructiunea EXIT

- Instructiunea EXIT este folosita pentru a incheia un LOOP.
- Controlul este transmis instructiunii care urmeaza dupa END LOOP.
- Se poate folosi EXIT:
  - fie ca actiune intr-un **IF**
  - fie ca o instructiune intr-un LOOP

Exemplu:

```
DECLARE
   monthly_value NUMBER:=10;
   daily_value NUMBER:=1;
 BEGIN
        LOOP
                monthly_value := daily_value * 31;
                daily_value := daily_value + 1;
                EXIT WHEN monthly_value > 4000;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(monthly_value);
END;
```

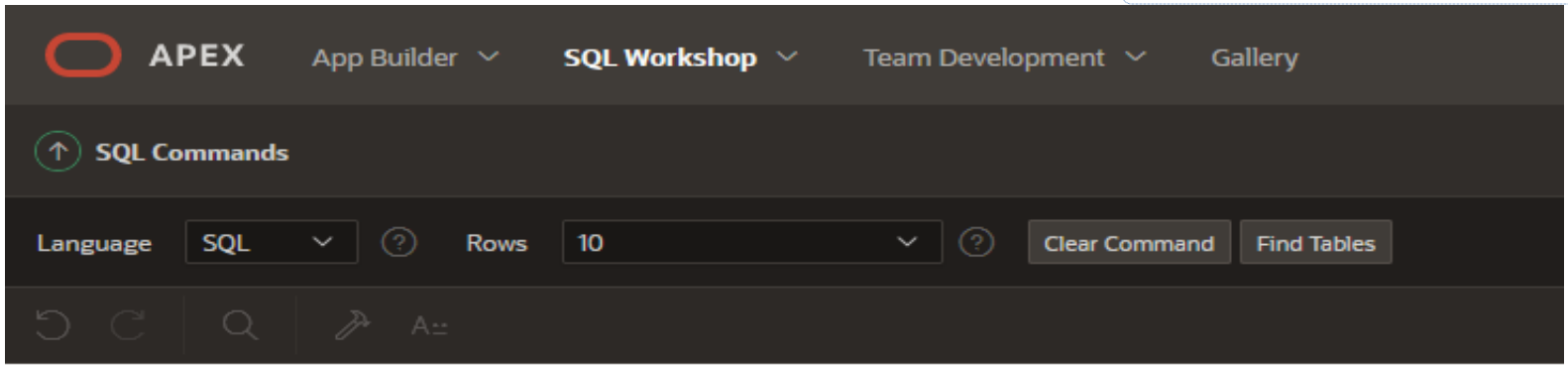


- Instructiunea EXIT trebuie plasata in interiorul unui LOOP.
- Daca o conditie EXIT este plasata la inceputul unui LOOP (inaintea oricarei instructiuni executabile) si daca conditia este initial TRUE atunci se iese din LOOP si celelalte instructiuni din LOOP nu se mai executa
- Un BASIC LOOP poate sa contina mai multe instructiuni EXIT, dar ar trebuie sa avem un singura iesire (EXIT).

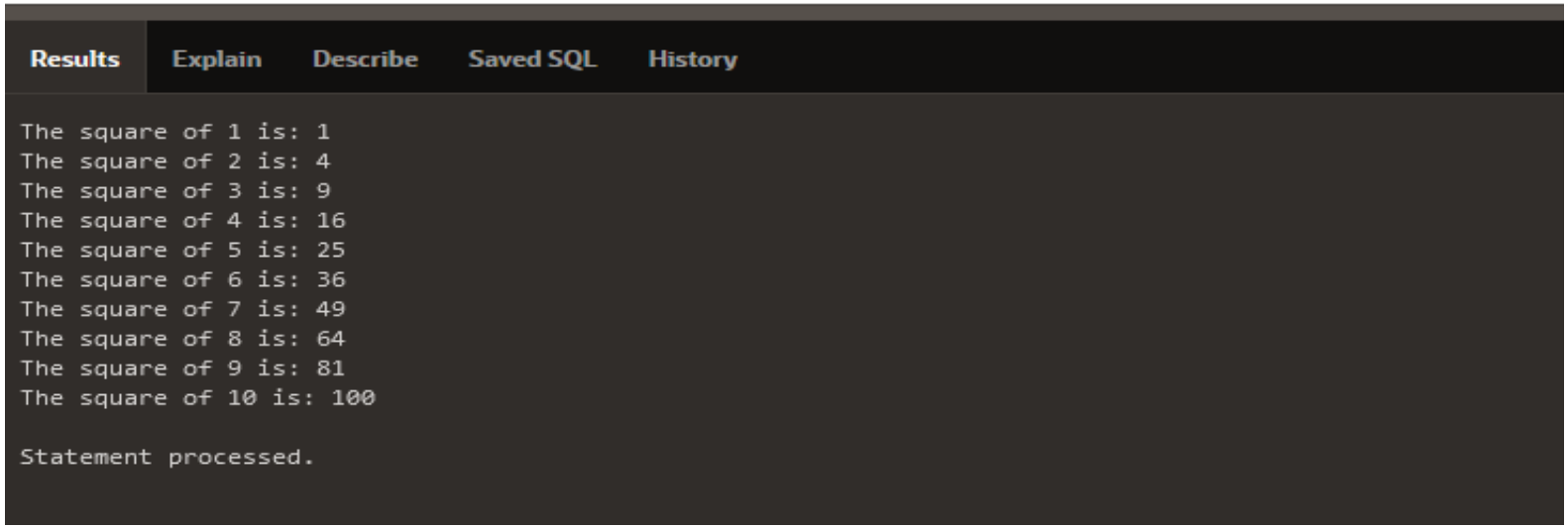
Instructiunea EXIT WHEN

- Clauza WHEN se foloseste pentru a permite o terminare conditionala a instructiunii LOOP.
- Atunci cand este intalnita instructiunea EXIT este evaluata conditia din clauza WHEN.
- Daca conditia ne da ca rezultat TRUE, atunci se incheie instructiunea LOOP si controlul este transmis instructiunii care urmeaza dupa LOOP.

```
DECLARE
   v_counter NUMBER := 1;
BEGIN
    LOOP
         DBMS_OUTPUT.PUT_LINE('The square of
   '||v_counter||' is: '|| POWER(v_counter,2));
         v_counter :=v_counter + 1;
         EXIT WHEN v_counter > 10;
    END LOOP;
END;
```




```
DECLARE
    v_counter NUMBER := 1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE('The square of '||v_counter||' is: '|| POWER(v_counter,2));
        v_counter :=v_counter + 1;
        EXIT WHEN v_counter > 10;
    END LOOP;
END;
```



3.2.Instructiunea WHILE

- *Instructiunea WHILE presupune repetarea unei secvente de instructiuni pana cand o conditie de control nu mai este adevarata.*
- Conditia este evaluata la inceputul fiecarei iteratii.
- Instructiunea se incheie atunci cand conditia este FALSE sau NULL.
- Daca conditia este FALSE sau NULL de la inceput atunci nu se executa nici o iteratie.

3.2.Instructiunea WHILE

Sintaxa

```
WHILE conditie LOOP
        instructiune1;
        instructiune 2;
        ……..
END LOOP;
```

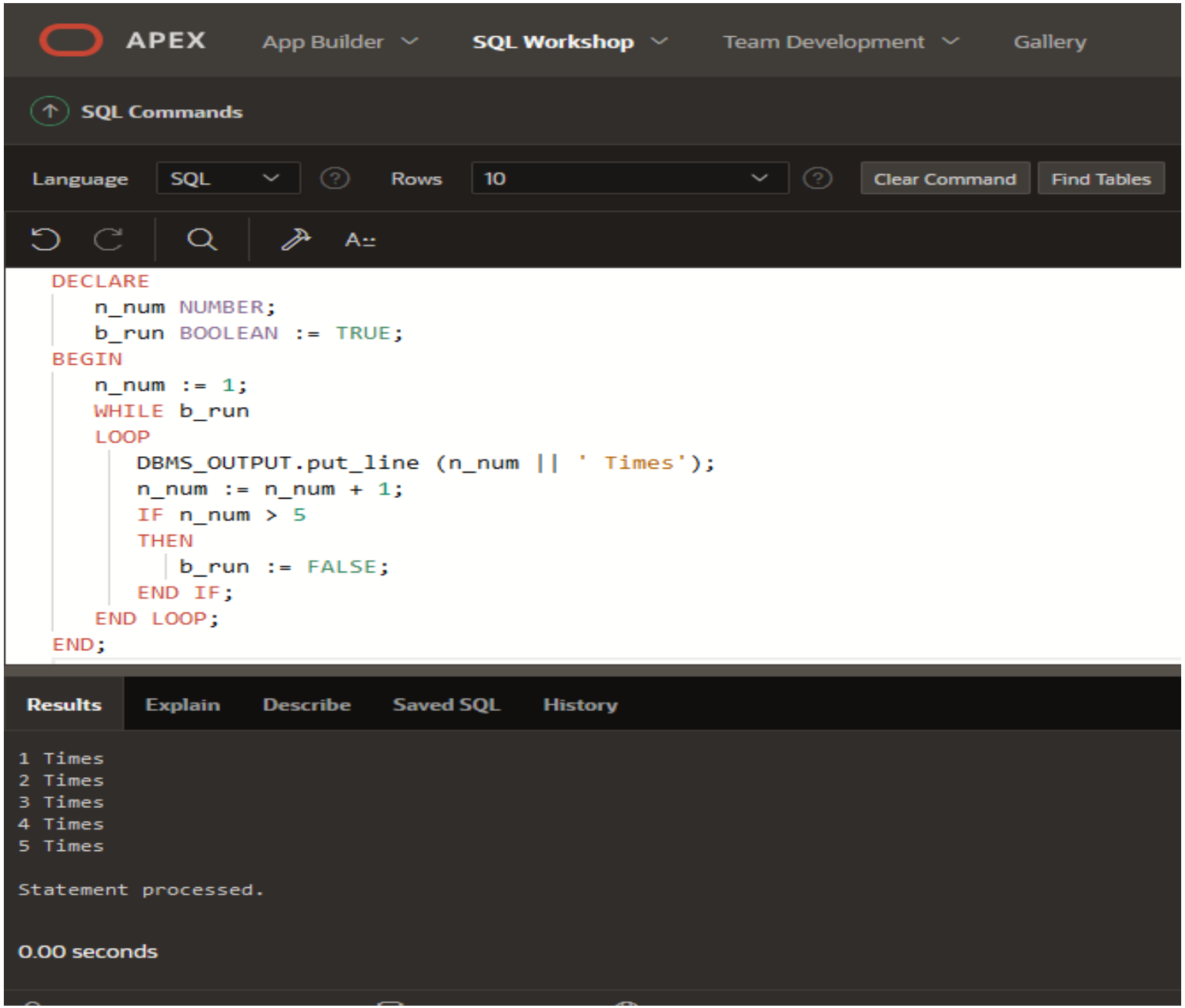
3.2.Instructiunea WHILE

- Conditia este o variabila sau o expresie booleana (TRUE, FALSE sau NULL).
- instructiune1, instructiune2,….. sunt grupari de una sau mai multe instructiuni
- Daca variabilele implicate in conditii nu se schimba pe parcursul instructiunii WHILE, atunci conditia ramane la valoarea TRUE si instructiunea se executa la infinit.

3.2.Instructiunea WHILE

Exemplu:

```
DECLARE
   n_num NUMBER;
   b_run BOOLEAN := TRUE;
BEGIN
   n_num := 1;
   WHILE b_run
   LOOP
      DBMS_OUTPUT.put_line (n_num || ' Times');
      n_num := n_num + 1;
      IF n_num > 5
      THEN
         b_run := FALSE;
      END IF;
   END LOOP;
END;
```



3.3.Instructiunea FOR

Instructiunea FOR are o structura asemanatoare unui basic loop, dar are in plus o instructiune de control inainte de cuvantul cheie LOOP pentru a seta numarul de iteratii pe care le executa PL/SQL.

3.3.Instructiunea FOR

Sintaxa

```
FOR counter IN [REVERSE]
lower_bound .. upper_bound LOOP
        instructiune 1;
        instructiune 2;
        ……..
END LOOP;
```

Se foloseste FOR pentru a da comenzi rapide de testare a numarului de iteratii

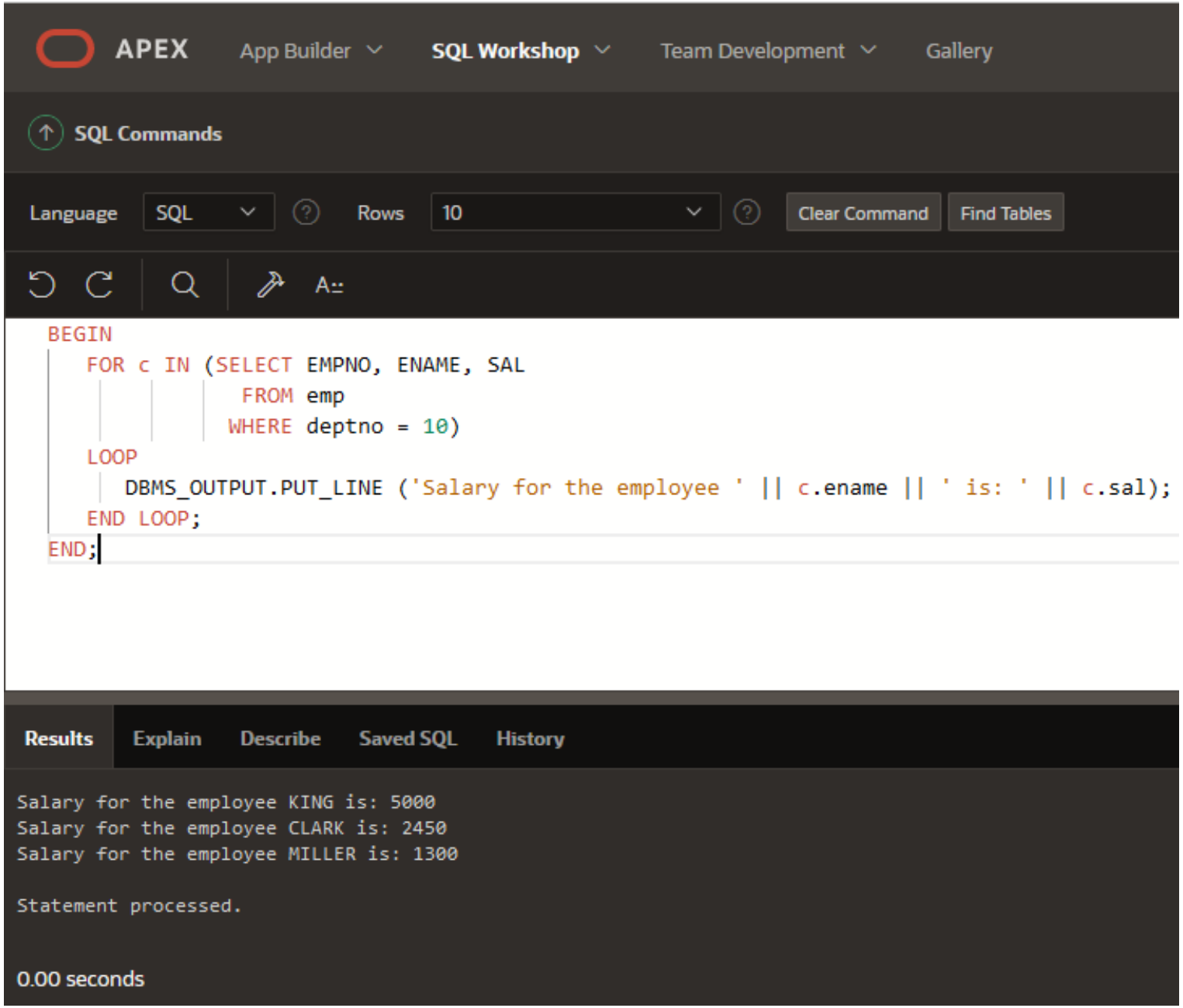
3.3.Instructiunea FOR

- Nu se declara contorul; el este declarat implicit
- Contorul este declarat implicit intreg si valoarea sa este marita sau micsorata (valoarea este micsorata daca se foloseste cuvantul cheie REVERSE) automat cu 1 la fiecare iteratie pana cand este atinsa valoarea finala.
- Intotdeauna valoarea mai mica **lower_bound** este prima specificata:
  - **lower_bound** specifica valoarea minima a domeniului de valori ale contorului
  - **upper_bound** specifica valoarea maxima a domeniului de valori ale contorului

3.3.Instructiunea FOR

Exemplu:

```
BEGIN
  FOR c IN (SELECT EMPNO, ENAME, SAL
            FROM emp
           WHERE deptno = 10)
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Salary for the
  employee ' || c.ename || ' is: ' || c.sal);
  END LOOP;
END;
```



3.3.Instructiunea FOR

Reguli de folosire a instructiunii FOR

- Contorul poate fi folosi numai in interiorul instructiunii; el este nedeclarat in afara ei
- Nu putem referi contorul pentru a-i atribui o valoare
- Valorile initiale si finale ale contorului nu pot fi NULL
- Cand scriem un FOR lower_bound si upper_bound nu trebuie sa fie neaparat literali numerici ci pot fi si expresii ce se convertesc la valori numerice.

3.3.Instructiunea FOR

Exemplu:

```
DECLARE
        v_lower NUMBER := 1;
        v_upper NUMBER := 100;

BEGIN
        FOR i IN v_lower..v_upper LOOP
                ……................
        END LOOP;
END;
```

Concluzii

- Se foloseste un **basic loop** atunci cand instructiunile trebuie sa se execute cel putin o data
- Se foloseste instructiunea **WHILE** atunci cand conditia trebuie sa fie evaluata la inceputul fiecarei iteratii
- Se foloseste instructiunea **FOR** atunci cand numarul de iteratii este cunoscut.

Întrebări?