

Oracle Database 11g
PL SQL – Part III

Controlling PL/SQL Flow of Execution

Change the logical flow of statements by using control structures:

- **Conditional control structures (IF statement)**
- **Loop control structures**
 - **Basic loop**
 - **FOR loop**
 - **WHILE loop**

The IF Statement: Syntax

```
IF condition THEN
    statements;
[ELSIF condition THEN
    statements;]
[ELSE
    statements;]
END IF;
```

- **END IF** is two words.
- At most, one **ELSE** clause is permitted.
- **ELSIF** is one word.

IF-THEN-ELSIF Statements: Example

- Categorize the employees according to their salary into three groups, one for employees who earn more than 5000, one for employees who earn more than 3000, and one for the rest.

```
...  
IF sal > 5000 THEN  
    dbms_output.put_line ('Category one');  
ELSIF sal > 3000 THEN  
    dbms_output.put_line ('Category two');  
ELSE  
    dbms_output.put_line ('Category three');  
END IF;
```

Basic Loop: Syntax

- Iterate through your statements with a basic loop.

```
LOOP                               -- delimiter
  statement1;                     -- statements
  . . .
  EXIT [WHEN condition];         -- EXIT statement
END LOOP;                           -- delimiter
```

- Without the EXIT statement, the loop would be infinite.

Basic Loop: Example

Insert the first ten new items for order number 101.

```
declaer
  v_ord_id    s_item.ord_id%TYPE := 101;
  v_counter  NUMBER(2) := 1;
BEGIN
...
  LOOP
    INSERT INTO s_item (ord_id, item_id)
      VALUES (v_ord_id, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
...

```

FOR Loop: Syntax

- Use a FOR loop to shortcut the test for the number of iterations.

```
FOR index in [REVERSE]
    lower_bound..upper_bound LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

- Do not declare the index; it is declared implicitly.

FOR Loop Example

```
BEGIN
FOR i IN 1 .. 10 LOOP
  DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
END;
```

```
BEGIN
FOR i IN REVERSE 1 .. 10 LOOP
  DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
END;
```


FOR Loop: Example

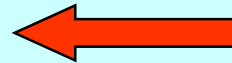
Guidelines

- **Reference the index within the loop only; it is undefined outside the loop.**
- **Use an expression to reference the existing value of an index.**
- **Index is read only, do not reference the index as the target of an assignment.**

WHILE Loop: Syntax

Use the WHILE loop to repeat statements while a condition is TRUE.

```
WHILE condition LOOP  
    statement1;  
    statement2;  
    . . .  
END LOOP;
```



Condition is evaluated at the beginning of each iteration.

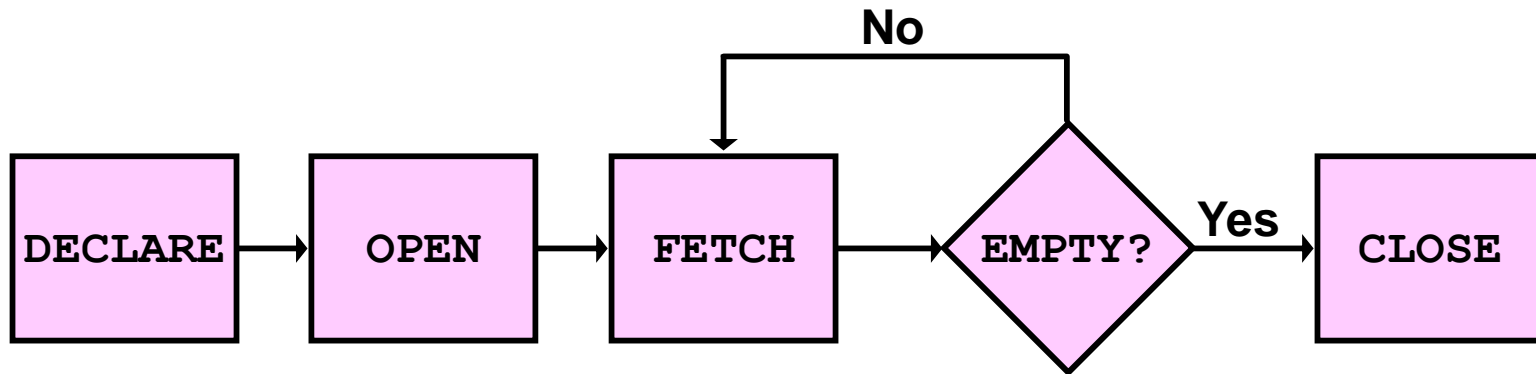
WHILE Loop: Example

Insert the first ten new items for order number 101.

```
. . .  
  v_ord_id    s_item.ord_id%TYPE := 101;  
  v_counter  NUMBER(2) := 1;  
BEGIN  
  
. . .  
  WHILE v_counter <= 10 LOOP  
    INSERT INTO s_item (ord_id, item_id)  
      VALUES (v_ord_id, v_counter);  
    v_counter := v_counter + 1;  
  END LOOP;  
  
. . .
```

Cursors

cursors are used to **select Multiple rows** inside PL/SQL block



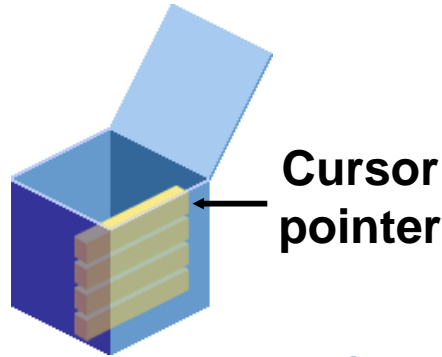
- Create a named SQL area
- Define the query
- Execute the query
- Identify the active set
- Load the current row into variables
- Test for existing rows
- Return to FETCH if rows are found
- Release the active set

Controlling Cursors

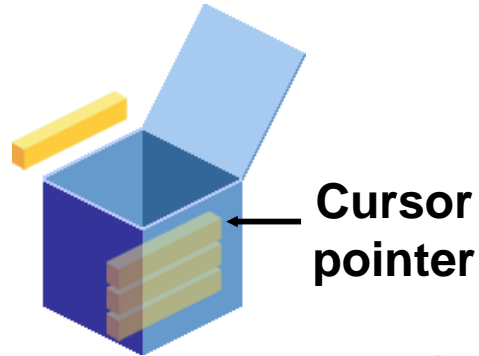
1. **Declare the cursor.** In the declarative section of a PL/SQL block, by naming it and defining the structure of the query to be associated with it.
2. **Open the cursor.** In the executable section of block. The OPEN statement executes the query. Rows identified by the query are called the active set and are now available for fetching.
3. **Fetch data from the cursor.** After each fetch, you test the cursor for any existing row. If there are no more rows to process, then you must close the cursor.
4. **Close the cursor.** The CLOSE statement releases the active set of rows. It is now possible to reopen the cursor to establish a fresh active set.

Controlling Cursors

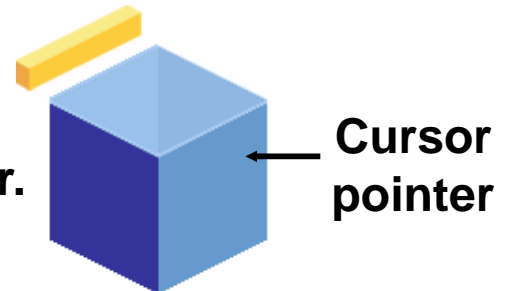
1 Open the cursor.



2 Fetch a row.



3 Close the cursor.



Declaring the Cursor

- **Syntax:**

```
CURSOR cursor_name IS  
    select_statement;
```

- **Example:**

```
DECLARE  
    CURSOR emp_cursor IS  
        SELECT employee_id, last_name FROM employees  
        WHERE department_id =30;
```

- The active set of a cursor is determined by the **SELECT** statement in the cursor declaration.
- It is mandatory to have an **INTO** clause for a **SELECT** statement in PL/SQL. However, note that the **SELECT** statement in **the cursor declaration cannot have an INTO clause.**
- If processing rows in a specific sequence is required, use the **ORDER BY** clause in the query.

Opening the Cursor

```
DECLARE
  CURSOR emp_cursor IS
  SELECT employee_id, last_name
  FROM   employees
  WHERE  department_id =30;

...
BEGIN
  OPEN emp_cursor;
```

- The OPEN statement :
 1. executes the query associated with the cursor
 2. identifies the active set
 3. positions the cursor pointer to the first row
- The OPEN statement is included in the executable section of the PL/SQL block.

Fetching One Record from the Cursor

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name
    FROM employees
    WHERE department_id =30;
  empno employees.employee_id%TYPE;
  lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO empno, lname;
  DBMS_OUTPUT.PUT_LINE( empno || ' ' ||lname);
END;
```

How many records are fetched here? Only 1

Fetching Data from the Cursor

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id =30;
  empno employees.employee_id%TYPE;
  lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO empno, lname;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( empno || ' ' ||lname);
  END LOOP;
  ...
END; /
```

- cursor attribute **%NOTFOUND** is used to test for the exit condition.

Closing the Cursor

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id =30;
    empno employees.employee_id%TYPE;
    lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO empno, lname;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( empno || ' ' ||lname);
  END LOOP;
  CLOSE emp_cursor;
  ...
END; /
```

Close Cursor

- The **CLOSE** statement disables the cursor
- You can reopen the cursor if required
- A cursor can be **reopened only if it is closed**
- If you attempt to fetch data from a cursor after it has been closed, then an **INVALID_CURSOR exception** will be raised.

Cursor: Complete Example

```
SET SERVEROUTPUT ON
DECLARE
    emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id =30;
    empno employees.employee_id%TYPE;
    lname employees.last_name%TYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO empno, lname;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( empno || ' ' ||lname);
    END LOOP;
    CLOSE emp_cursor;
END;
```

Cursors and Records

- Process the rows of the active set by fetching values into a PL/SQL RECORD.

```
DECLARE
    CURSOR emp_cursor IS
    SELECT employee_id, last_name
    FROM employees
    WHERE department_id =30;
    emp_record emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO emp_record;
        ...
    
```

- You can define a record based on the selected list of columns in an explicit cursor.

Cursor FOR Loops

- **Syntax:**

```
FOR record_name IN cursor_name LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

- The cursor **FOR loop** is a shortcut to process cursors.
- Implicit open, fetch, exit, and close occur.
- The record is implicitly declared.

Cursor FOR Loops

- It is a shortcut because: the cursor is opened, a row is fetched once for each iteration in the loop, the loop exits when the last row is processed, and the cursor is closed automatically.
- The loop itself is terminated automatically at the end of the iteration where the last row is fetched.

Cursor FOR Loops

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR emp_cursor IS
    SELECT employee_id, last_name
    FROM employees
    WHERE department_id =30;
BEGIN
    FOR emp_record IN emp_cursor LOOP
        DBMS_OUTPUT.PUT_LINE(
            emp_record.employee_id
            || ' ' || emp_record.last_name);
    END LOOP;
END;
```

Cursor Attributes

- Obtain status information about cursor.

Attribute	Type	Description
<code>%ISOPEN</code>	Boolean	Evaluates to <code>TRUE</code> if the cursor is open
<code>%NOTFOUND</code>	Boolean	Evaluates to <code>TRUE</code> if the most recent fetch does not return a row
<code>%FOUND</code>	Boolean	Evaluates to <code>TRUE</code> if the most recent fetch returns a row; complement of <code>%NOTFOUND</code>
<code>%ROWCOUNT</code>	Number	Evaluates to the number of rows returned so far

The %ISOPEN Attribute

- Fetch rows only when the cursor is open.
- Use the %ISOPEN cursor attribute before performing a fetch to test whether the cursor is open.
- Example:

```
IF NOT emp_cursor%ISOPEN THEN
    OPEN emp_cursor;
END IF;
LOOP
    FETCH emp_cursor...
```

Example of %ROWCOUNT and %NOTFOUND

```
SET SERVEROUTPUT ON
DECLARE
  empno  employees.employee_id%TYPE;
  ename  employees.last_name%TYPE;
  CURSOR emp_cursor IS
  SELECT employee_id,
         last_name FROM employees;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO empno, ename;
    EXIT WHEN emp_cursor%ROWCOUNT > 10 OR
             emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE (TO_CHAR (empno)
                          || ' ' || ename);
  END LOOP;
  CLOSE emp_cursor;
END ;
```

Practice1

Use cursor to display the employee number and name for all employees in department 50

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id = 50;
empnum employees.employee_id%TYPE;
empname employees.last_name%TYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
    FETCH emp_cursor INTO empnum, empname;
    EXIT WHEN emp_cursor%notfound;
    DBMS_OUTPUT.PUT_LINE( empnum || ' : ' || empname);
    END LOOP;
    CLOSE emp_cursor;
END;
```

Practice 2

- **Create a function that calculates the number of employees working in a specific department that will be provided by the user.**
- **Then execute this function for department 10**

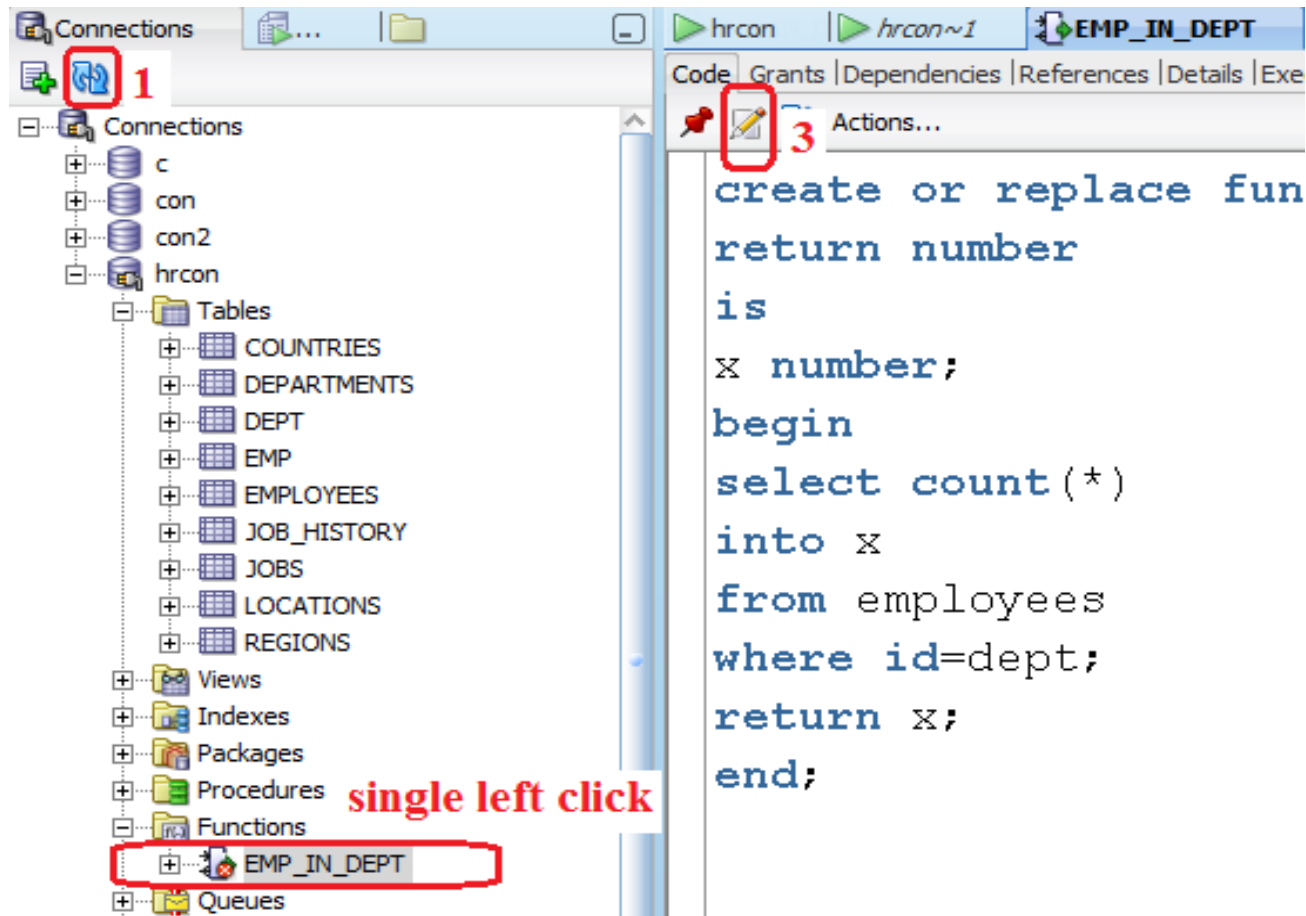
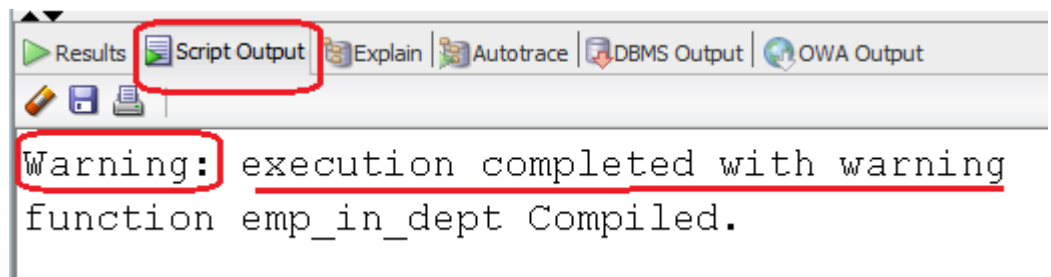
Creating Function

The screenshot displays the Oracle SQL Developer interface. On the left, the 'Connections' tree is expanded to show the 'hrcon' connection, with the 'Functions' folder highlighted in red and labeled '3'. The 'EMP_IN_DEPT' function is visible under 'Functions'. In the main editor window, the SQL code for creating the function is shown, with the 'CREATE OR REPLACE' statement highlighted in yellow and labeled '1'. The code is as follows:

```
create or replace function emp_in_dept (dept number)
return number
is
x number;
begin
select count (*)
into x
from employees
where department_id=dept;
return x;
end;
```

At the bottom of the editor, the 'Script Output' tab shows the message: function emp_in_dept Compiled.

Correcting errors in functions

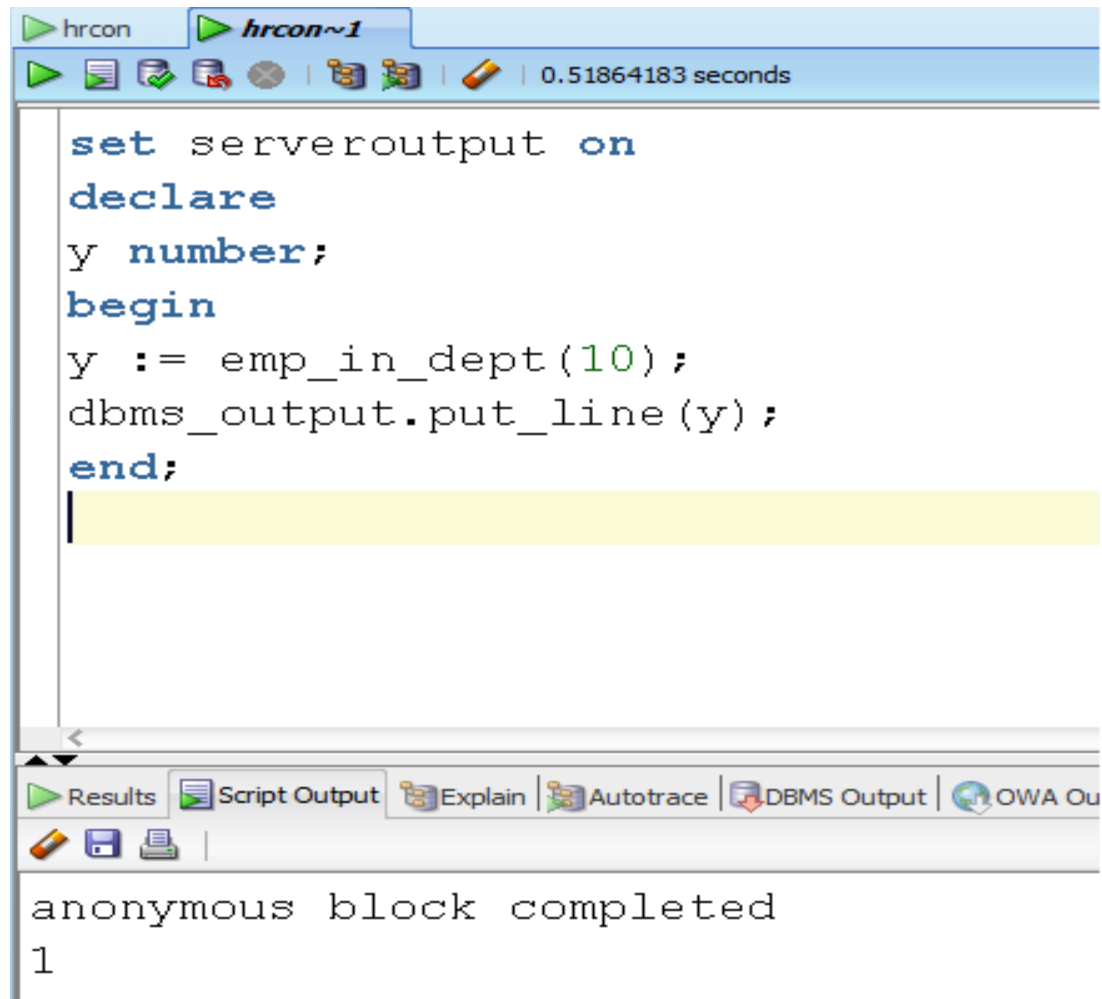


```
hrcon | hrcon~1 | EMP_IN_DEPT | EMP_IN_DEPT
create or replace
function emp_in_dept (dept number)
return number
is
x number;
begin
select count(*)
into x
from employees
where id=dept;
return x;
end;
```

2 Click to compile

1 correct the error

Executing function



```
hrcon hrcon~1
0.51864183 seconds

set serveroutput on
declare
y number;
begin
y := emp_in_dept(10);
dbms_output.put_line(y);
end;

anonymous block completed
1
```

Practice 3

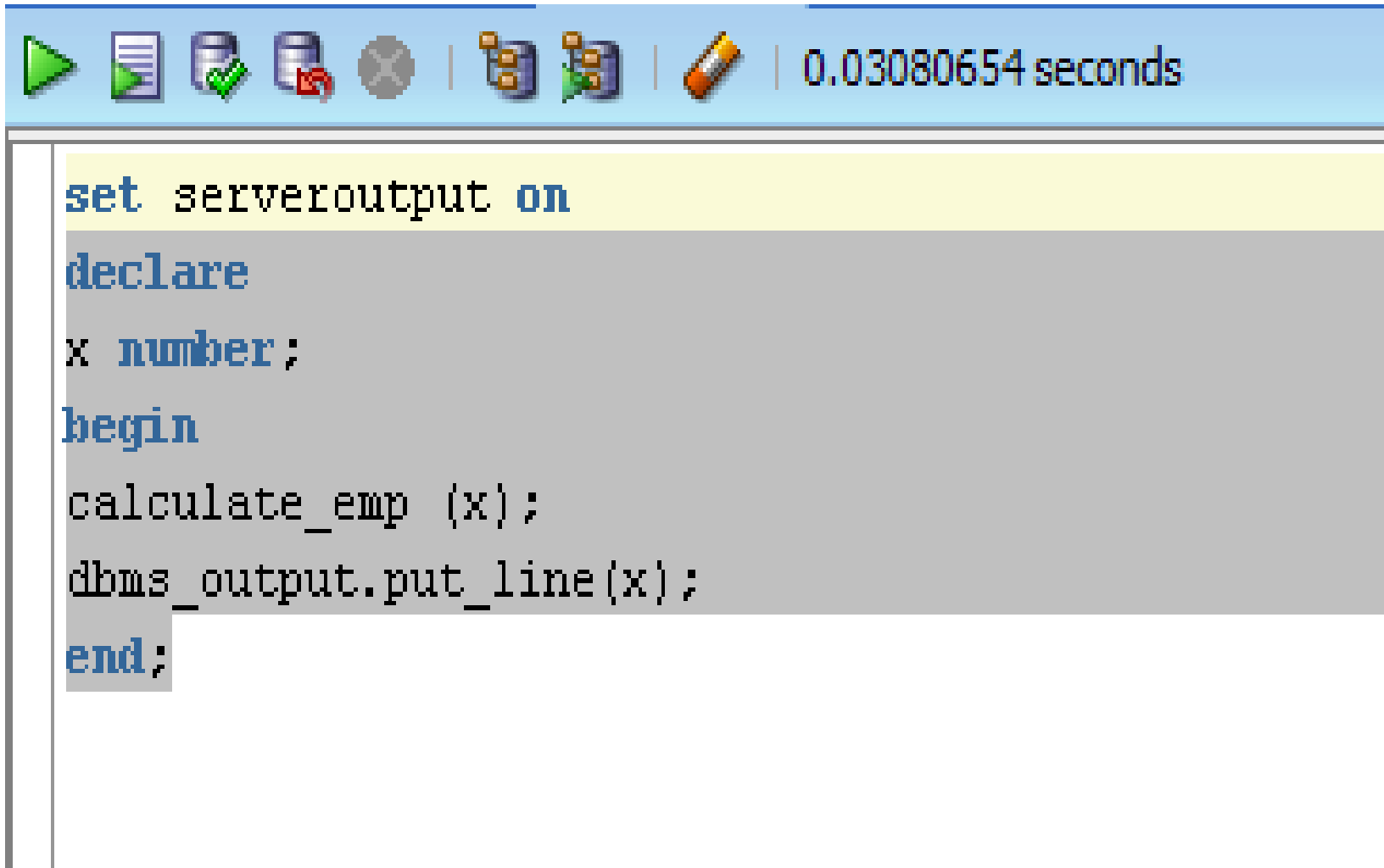
- **Create a procedure that displays the number of employees who have a manager.**

Creating Procedure with OUT parameter

```
create or REPLACE procedure calculate-emp( num_emp out number)
is
BEGIN
select count(*)
into num_emp
FROM emp
where mgr is not null;

end;
```

Executing Procedure with OUT parameter



The screenshot shows a database execution interface. At the top, a blue toolbar contains several icons: a green play button, a document with a green arrow, a database cylinder with a green checkmark, a database cylinder with a red 'X', a grey 'X' icon, a database cylinder with a green arrow, a database cylinder with a green arrow, and a battery icon. To the right of these icons, the execution time is displayed as '0.03080654 seconds'. Below the toolbar, a text area contains the following SQL code:

```
set serveroutput on
declare
x number;
begin
calculate_emp (x);
dbms_output.put_line(x);
end;
```