Level Up Your PL/SQL Skills: Mastering Functions

por Mayko Silva

Your PL/SQL Toolbox

Procedures

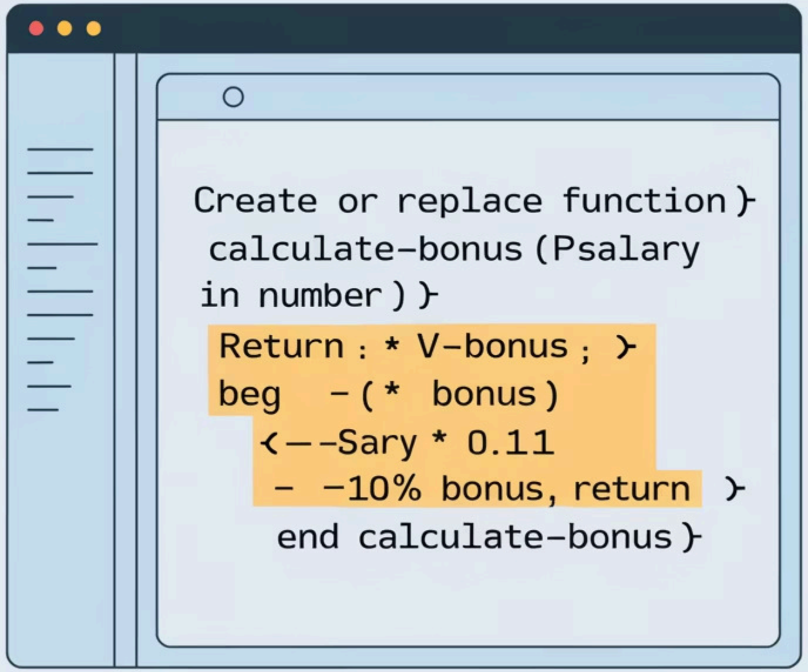
Think of procedures as the versatile screwdrivers in your toolbox. They're great for performing tasks but don't always return a specific value.

Functions

Functions are the precision instruments. They're designed to take input, perform calculations, and always return a result.

Functions: Like a Calculator

If procedures are like following a recipe, functions are like using a calculator. Input values, get a result. Just like asking 'What's 5 plus 3?' and receiving '8'.



```
Create or replace function)
 calculate-bonus (Psalary
in number ) )
  Return : * V-bonus ; )
  beg   - (*  bonus )
    <--Sary * 0.11
     -  -10% bonus, return  )
     end calculate-bonus )
```

Function Structure: A Closer Look

```
CREATE OR REPLACE FUNCTION calculate_bonus(p_salary IN NUMBER)
RETURN NUMBER IS v_bonus NUMBER; BEGIN v_bonus := p_salary *
0.1; -- 10% bonus RETURN v_bonus; END calculate_bonus;
```

Breaking Down the Function Code

1 Create or Replace

Tells Oracle to create a new function or update an existing one.

2 Function Name & Input

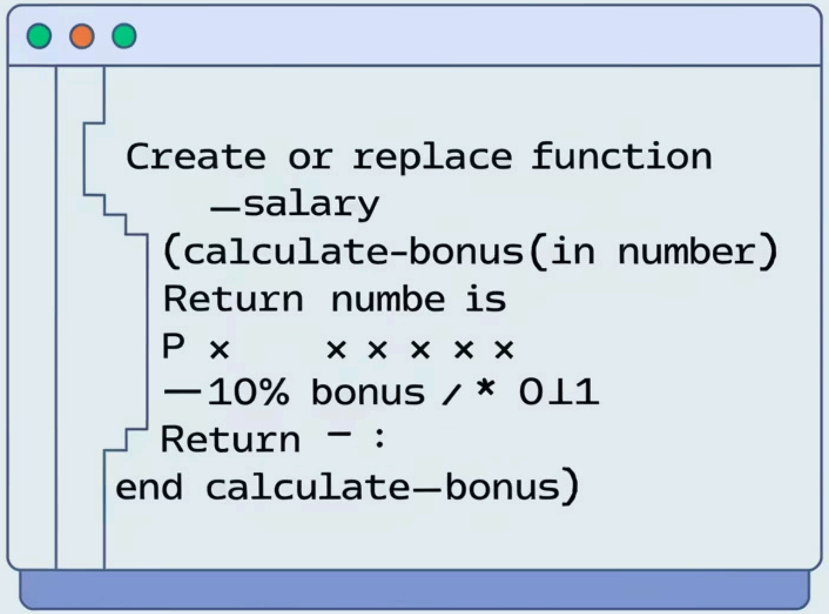
We name our function 'calculate_bonus' and specify the input ('p_salary').

3 Return Type

Promising that our function will always return a number.

4 Calculation & Return

We perform the calculation and use RETURN to send back the result.



```
 Create or replace function
    —salary
  (calculate-bonus(in number)
  Return  numbe is
  P ×     × × × × ×
  —10% bonus ⁄ * 0⊥1
  Return − :
end calculate—bonus)
```

Why Use Functions?

Procedures

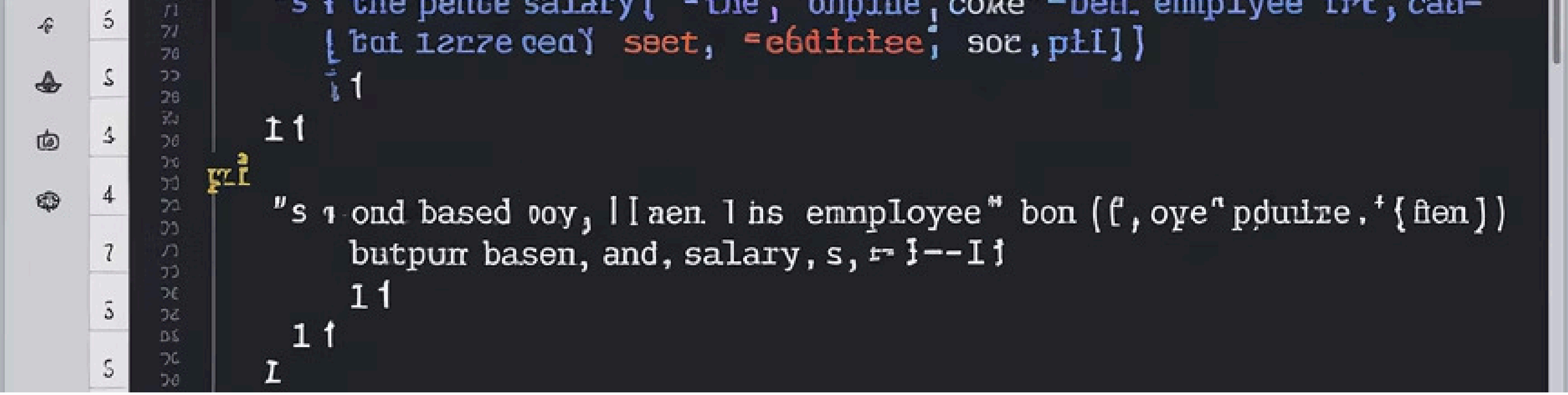
Like tasks, procedures can perform multiple things but don't always return a value. They are like instructions to organize files.

Functions

Functions are like calculations. They always return a value. It's like asking for the total of a set of numbers.

The Power of Reusability

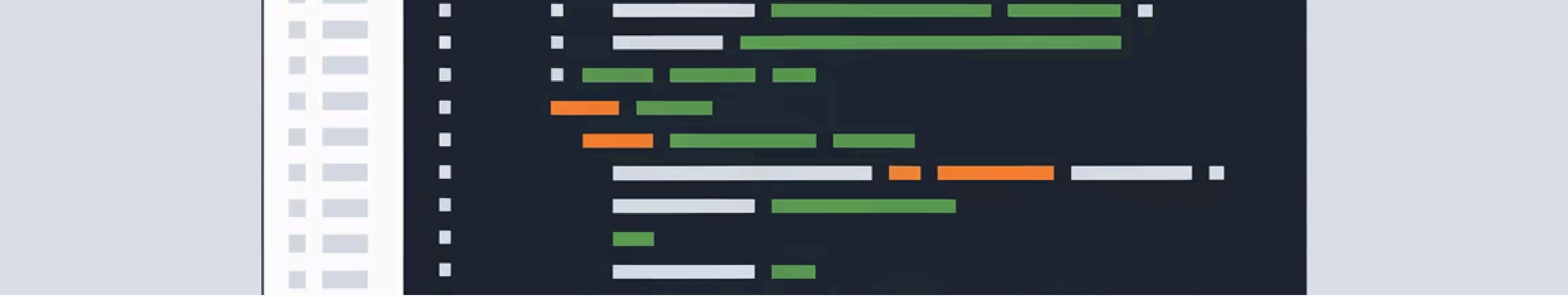
Once created, functions can be used over and over again in your code, just like you'd use a calculator for different problems. They make your code cleaner and more efficient.



```
s ı the pence salary( -the, onpine, coke -ben. emplyee ırt, can-
  [ but ıerze ceal) seet, =ebdıctee, soc, ptI]]
  ı1
 I1
rıl
 "s ı ond based ooy, ||aen. l ns emnployee" bon ((, oye"pduize,'{fıen])
   butpun basen, and, salary, s, r- }--I}
   I1
  1 1
 I
```

Using Our Bonus Function

```
DECLARE v_employee_salary NUMBER := 50000; v_bonus NUMBER; BEGIN v_bonus := calculate_bonus(v_employee_salary);
DBMS_OUTPUT.PUT_LINE('Employee bonus: $' || v_bonus); END;
```



The Results Are In!

By calling our 'calculate_bonus' function, we've calculated a bonus without re-writing the calculation code. That's the power of functions – write once, use many times!

What's Next: Nested Functions

Now that you've mastered the basics, get ready for nested functions – functions within other functions. It's like Russian nesting dolls, but way more powerful in programming! Stay tuned for our next segment.