

DATABASE PROGRAMMING

CRIE SUAS PRÓPRIAS
**FUNÇÕES NO
ORACLE**

MILTON GOYA



08

LISTA DE QUADROS

Quadro 1 – Diferenças entre procedimentos e funções	6
---	---

EXEMPLO

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Sintaxe da criação de uma FUNCTION	6
Código-fonte 2 – Exemplo de criação de uma função.....	7
Código-fonte 3 – Teste da função DESCOBRIR_SALARIO.....	7
Código-fonte 4 – Teste da função DESCOBRIR_SALARIO usando bloco anônimo.	8
Código-fonte 5 – Exemplo de criação da função CONTADEPT	8
Código-fonte 6 – Teste da função CONTADEPT usando bloco anônimo	8
Código-fonte 7 – Exemplo de criação da função SAL_ANUAL	9
Código-fonte 8 – Teste da função SAL_ANUAL.....	9
Código-fonte 9 – Teste da função SAL_ANUAL usando bloco anônimo.....	10
Código-fonte 10 – Exemplo de criação da função SAL_ANUAL	10
Código-fonte 11 – Teste da função ORDINAL	10
Código-fonte 12 – Teste da função ORDINAL usando bloco anônimo	11
Código-fonte 13 – Descrição da <i>view</i> USER_ERRORS.....	12
Código-fonte 14 – Exibindo os erros da função ORDINAL.....	12
Código-fonte 15 – Exibindo código-fonte da função ORDINAL.....	13

SUMÁRIO

1 CRIE SUAS PRÓPRIAS FUNÇÕES NO ORACLE	5
1.1 Bloco PL/SQL nomeado	5
1.2 A função	5
1.3 Visualização de erros de compilação	11
CONCLUSÃO.....	14
REFERÊNCIAS.....	15

EXEMPLO

1 CRIE SUAS PRÓPRIAS FUNÇÕES NO ORACLE

Sim! A possibilidade de criar funções com o objetivo de evitar o *copy & paste* do nosso código, automatizando as tarefas, é possível em PL/SQL. Muito parecidas com as *procedures*, as funções têm como características o retorno de valores e a utilização como parte de uma expressão. Para criar as suas próprias funções dentro do Oracle, aproveite o conteúdo deste capítulo.

1.1 Bloco PL/SQL nomeado

Assim como o procedimento ou PROCEDURE, a função ou FUNCTION é um bloco de código PL/SQL que recebe um nome e pode ser executada por outras aplicações e/ou usuários. Outras aplicações/usuários poderão executar as funções e procedimentos, caso recebam permissão para isso.

Para a Oracle (2016), o bloco nomeado de uma função é conhecido por vários nomes. Pode ser chamado de função armazenada ou STORED FUNCTION, função do usuário, ou USER FUNCTION, função definida pelo usuário, ou USER-DEFINED FUNCTION, subprograma ou sub-rotina.

1.2 A função

Para Puga, França e Goya (2015), uma função ou FUNCTION é muito semelhante a um *procedimento* ou PROCEDURE. O que a difere, do ponto de vista estrutural, é a inclusão da cláusula RETURN. A cláusula RETURN também implementa a diferença conceitual entre os procedimentos e as funções, isto é, nas funções existe a obrigatoriedade de um retorno à *rotina chamadora*, enquanto no procedimento não existe essa obrigatoriedade. Pode-se dizer que uma função é chamada como parte de uma expressão.

Veja o quadro abaixo para visualizar as diferenças entre procedimentos e funções:

Procedimento	Função
É chamado em uma declaração SQL, blocos PL/SQL ou por uma aplicação.	É chamada como parte de uma expressão.
Não contém a cláusula RETURN no cabeçalho.	Contém a cláusula RETURN no cabeçalho.
Pode retornar nenhum, um ou vários valores.	Retorna somente um valor.
Pode devolver um retorno à rotina chamadora.	Retorna obrigatoriamente um valor à rotina chamadora.

Quadro 1 – Diferenças entre procedimentos e funções
Fonte: ORACLE (2016)

A sintaxe para a criação de uma função é similar à sintaxe para a criação de um procedimento, mas possui algumas diferenças marcantes:

```
CREATE [ OR REPLACE] FUNCTION nome_função
[(parâmetro [in] tipo_parâmetro,
...
return tipo_do_retorno
{IS ou AS}

BEGIN
    corpo_da_função
END nome_função;
```

Código-fonte 1 – Sintaxe da criação de uma FUNCTION
Fonte: Oracle (2016)

Em que:

CREATE OR REPLACE é a instrução para a criação ou a substituição da função.

nome_função é o nome que será dado à função.

parâmetro [parâmetro [in]] é nome do parâmetro que poderá ser somente de entrada.

tipo_parâmetro é o tipo de dado que o parâmetro poderá aceitar, o tipo de parâmetro somente pode ser IN.

return tipo_do_retorno indica o retorno do controle à rotina chamadora com o valor que será devolvido.

IS ou AS têm a mesma função e indicam o bloco que estará associado à função, substitui a palavra reservada DECLARE.

BEGIN corpo_do_procedimento END são, respectivamente, o início do bloco, o conjunto de instruções da função e o final do bloco.

Vejamos um exemplo simples de criação de função ou FUNCTION:

```
CREATE OR REPLACE FUNCTION descobrir_salario
  (p_id IN emp.empno%TYPE)
RETURN NUMBER
IS
  v_salario emp.sal%TYPE := 0;
BEGIN
  SELECT sal INTO v_salario
    FROM emp
   WHERE empno = p_id;
  RETURN v_salario;
END descobrir_salario;
/
```

Código-fonte 2 – Exemplo de criação de uma função
Fonte: Oracle (2016), adaptado pelo autor (2017)

O exemplo acima cria uma função denominada DESCOBRIIR_SALARIO, que recebe código de um funcionário e devolve para a *rotina chamadora* o salário do empregado pesquisado.

Vamos testar nossa função:

```
SELECT empno, DESCOBRIIR_SALARIO (empno)
FROM emp;
```

Código-fonte 3 – Teste da função DESCOBRIIR_SALARIO
Fonte: Elaborado pelo autor (2017)

O nosso teste exibe o código de cada funcionário e executa a função DESCOBRIIR_SALARIO com o código de cada funcionário como entrada e retorna o salário de cada um deles.

Perceba que a função DESCOBRIIR_SALARIO foi executada uma vez para cada linha retornada pela consulta que foi executada com o valor do código do empregado obtido pelo cursor implícito.

Também podemos executar a função em outra função, procedimento ou bloco anônimo. Vejamos um exemplo usando um bloco PL/SQL anônimo.

```
SET SERVEROUTPUT ON

BEGIN
    DBMS_OUTPUT.PUT_LINE (DESCOBRIR_SALARIO (7900) );
END;
/
```

Código-fonte 4 – Teste da função DESCOBRIR_SALARIO usando bloco anônimo
Fonte: Elaborado pelo autor (2017)

No exemplo, executamos a função DESCOBRIR_SALARIO em bloco PL/SQL anônimo. A função recebe como entrada o código do funcionário 7900 e retorna o seu salário para o programa chamador, que o exibe.

Vamos criar mais uma função, desta vez, não terá valores de entrada, apenas o valor de retorno:

```
CREATE OR REPLACE FUNCTION contadept
RETURN number IS
    total NUMBER(7) := 0;
BEGIN
    SELECT count(*) INTO total
    FROM dept;
    RETURN total;
END;
/
```

Código-fonte 5 – Exemplo de criação da função CONTADEPT
Fonte: Elaborado pelo autor (2017)

Perceba que a função não tem parâmetros de entrada. Ao ser executada, conta a quantidade de tuplas da tabela DEPT e retorna ao programa chamador.

Vamos testar nossa nova função, usando um bloco anônimo:

```
SET SERVEROUTPUT ON

DECLARE
    conta NUMBER(7);
BEGIN
    conta := CONTADEPT();
    DBMS_OUTPUT.PUT_LINE('Quantidade de Departamentos: ' ||
    conta);
END;
/
```

Código-fonte 6 – Teste da função CONTADEPT usando bloco anônimo
Fonte: Elaborado pelo autor (2017)

Note que a função foi executada sem a passagem de valores de entrada, mas retornou a contagem da quantidade de registros da tabela DEPT.

Vejamos outro exemplo de função, desta vez, receberá dois parâmetros e não acessará dados de uma tabela.

```
CREATE OR REPLACE FUNCTION sal_anual (  
    p_sal          NUMBER,  
    p_comm         NUMBER )  
RETURN NUMBER  
IS  
BEGIN  
    RETURN (p_sal + NVL(p_comm, 0)) * 12;  
END sal_anual;  
/
```

Código-fonte 7 – Exemplo de criação da função SAL_ANUAL
Fonte: Elaborado pelo autor (2017)

Na função SAL_ANUAL temos dois parâmetros de entrada. O primeiro receberá o valor do salário e o segundo receberá a comissão do funcionário. A função soma os dois valores e os multiplica por 12. O valor calculado será devolvido para o programa chamador. Vamos testá-la:

```
SELECT sal, comm, SAL_ANUAL(sal, comm)  
FROM emp;
```

Código-fonte 8 – Teste da função SAL_ANUAL
Fonte: Elaborado pelo autor (2017)

O nosso teste exibe o salário de cada funcionário e a comissão de cada um e executa a função SAL_ANUAL com o valor do salário e da comissão como parâmetro. A função SAL_ANUAL calcula e devolve o salário anual de cada um dos funcionários. Alguns funcionários não têm comissão e, por esse motivo, o campo de comissão está com valores nulos.

A função SAL_ANUAL trata os valores nulos usando a função NVL, que trata os valores nulos. Perceba que a nossa função está usando outra função em seu código. A função SAL_ANUAL é executada uma vez para cada linha retornada pela consulta. Vamos testar a nossa função em um bloco anônimo.

```
SET SERVEROUTPUT ON  
  
DECLARE  
    total NUMBER(7);
```

```
BEGIN
    total := SAL_ANUAL(900, 100);
    DBMS_OUTPUT.PUT_LINE('Salário anual: ' || total);
END;
/
```

Código-fonte 9 – Teste da função SAL_ANUAL usando bloco anônimo
Fonte: Elaborado pelo autor (2017)

Você percebeu que tanto as funções quanto os procedimentos podem usar as funções nativas do banco de dados, também as estruturas condicionais do PL/SQL para criar novas funcionalidades para seu SGBDR. Vejamos mais um exemplo:

```
CREATE OR REPLACE FUNCTION ordinal (
    p_numero          NUMBER)
RETURN VARCHAR2
IS
BEGIN
    CASE p_numero
        WHEN 1 THEN RETURN 'primeiro';
        WHEN 2 THEN RETURN 'segundo';
        WHEN 3 THEN RETURN 'terceiro';
        WHEN 4 THEN RETURN 'quarto';
        WHEN 5 THEN RETURN 'quinto';
        WHEN 6 THEN RETURN 'sexto';
        WHEN 7 THEN RETURN 'sétimo';
        WHEN 8 THEN RETURN 'oitavo';
        WHEN 9 THEN RETURN 'nono';
        ELSE RETURN 'não previsto';
    END CASE;
END ordinal;
/
```

Código-fonte 10 – Exemplo de criação da função SAL_ANUAL
Fonte: Elaborado pelo autor (2017)

Na função ORDINAL, temos um parâmetro numérico de entrada e o retorno de um texto. O programa testa valores entre um e nove e exibe o ordinal do número, ou seja, o número 1 será exibido como “primeiro”, o número 2 será mostrado como “segundo” e assim por diante até o número 9, que será exibido como “nono”. Qualquer número diferente desses nove números será mostrado como “não previsto”. Vamos testar nossa nova função:

```
SELECT ORDINAL(9)
FROM dual;
```

Código-fonte 11 – Teste da função ORDINAL
Fonte: Elaborado pelo autor (2017)

Em nosso teste, a consulta executa a função ORDINAL com o valor 9 como parâmetro. A função retorna o texto “nono”. A tabela DUAL é usada para testes e contém apenas uma tupla.

Vamos testar nossa função com um bloco PL/SQL anônimo:

```
SET SERVEROUTPUT ON

BEGIN
  FOR i IN 1..9 LOOP
    DBMS_OUTPUT.PUT_LINE (ORDINAL (i) );
  END LOOP;
END;
/
```

Código-fonte 12 – Teste da função ORDINAL usando bloco anônimo
Fonte: Elaborado pelo autor (2017)

Nesse novo teste, o bloco PL/SQL anônimo está executando um laço FOR, iterando a variável “i” do número um até o número nove. Dentro do laço, estamos chamando a função ORDINAL nove vezes. Na primeira vez, o valor de “i” será um, na segunda passagem, o valor de “i” será dois e assim por diante até chegar ao valor nove. Dessa forma, serão exibidos os valores ordinais para todos os números previstos em nosso código.

1.3 Visualização de erros de compilação

Para a Oracle (2016), durante a criação ou a alteração de uma função, o desenvolvedor pode receber a mensagem “**Function created with compilation errors**”. Isso indica que a função foi criada com erros de compilação.

Além do comando SHOW ERROS, visto no capítulo anterior, podemos obter mais informações sobre os erros por meio das **views** USER_ERRORS, ALL_ERRORS e DBA_ERRORS. A estrutura básica dessas **views** é vista abaixo:

Name	Null?	Type
NAME	NOT NULL	VARCHAR2 (30)
TYPE		VARCHAR2 (12)
SEQUENCE	NOT NULL	NUMBER
LINE	NOT NULL	NUMBER
POSITION	NOT NULL	NUMBER
TEXT	NOT NULL	VARCHAR2 (4000)
ATTRIBUTE		VARCHAR2 (9)
MESSAGE_NUMBER		NUMBER

Código-fonte 13 – Descrição da *view* USER_ERRORS
Fonte: Oracle (2016)

Em que,

NAME contém o nome do objeto compilado.

TYPE pode assumir os valores PROCEDURE, FUNCTION, TRIGGER, PACKAGE e PACKAGE_BODY.

SEQUENCE corresponde ao número de erros relativo ao número de erros na compilação.

LINE indica o número da linha que contém o erro.

POSITION indica a coluna que contém o erro.

TEXT contém o texto do erro, que pode ser, por exemplo: "PL/SQL: Statement ignored PL/SQL-00201 identifier 'DBMS_OUTPUT.PUT_LINE' must be declared".

ATTRIBUTE nem todo registro da *view* é um erro, alguns dos registros podem conter apenas uma advertência. Caso seja uma advertência, essa coluna conterá a palavra "**WARNING**"; se for um erro, conterá a palavra "**ERROR**".

MESSAGE_NUMBER contém o código do erro, sem o prefixo. Por exemplo, se o erro for PLS-00103, essa coluna conterá o código 103.

Usando essas *views*, é possível listar os erros das funções criadas. O código abaixo é um exemplo de consulta:

```
SELECT line, position, text
FROM user_errors
WHERE name = 'ORDINAL'
ORDER BY sequence;
```

Código-fonte 14 – Exibindo os erros da função ORDINAL
Fonte: Elaborado pelo autor (2017)

No exemplo, estamos exibindo linha, coluna e mensagem de erro da função ORDINAL. Caso não existam erros de compilação, nenhuma linha será exibida.

Se houver necessidade, é possível consultar a *view* USER_SOURCE para obter o código-fonte da função. Veja no exemplo:

```
SELECT text
  FROM user_source
 WHERE name = 'ORDINAL'
 ORDER BY line;
```

Código-fonte 15 – Exibindo código-fonte da função ORDINAL
Fonte: Elaborado pelo autor (2017)

No exemplo, estamos exigindo o código-fonte da função ORDINAL. Note que, via de regra, o nome dos objetos é armazenado em letras maiúsculas no dicionário de dados.

CONCLUSÃO

Procedimentos e funções são elementos úteis para armazenar processos e tratamentos de dados em um banco e aceitam todas as estruturas condicionais, repetições, tipos de variáveis e tratamentos de exceção descritos aqui. Utilize funções sempre que precisar transformar algum dado que queira exibir em uma consulta.

EXEMPLO

REFERÊNCIAS

DILLON, S.; BECK, C.; KYTE, T.; KALLMAN, J.; ROGERS, H. **Beginning Oracle Programming**. USA: Apress, 2013.

FEUERSTEIN, S.; PRIBYL, B. **Oracle PL/SQL Programming**. 6. ed. California USA: O'Reilly Media, 2014.

ORACLE. **Oracle Database: PL/SQL Language Reference 12c Release 2 (12.2)** B28370-05. USA: Oracle Press, 2016.

PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de dados**. São Paulo: Pearson, 2015.