ENCONTRO 3 – Procedures e Functions (4h)

Objetivos

- Compreender a estrutura e sintaxe de **procedures** e **functions**.
- Criar e executar **procedimentos armazenados** e **funções** no Oracle.
- Utilizar **parâmetros IN, OUT e IN OUT**.
- Reutilizar lógica de negócio no banco de dados.

1.**Conceito e Estrutura de uma Procedure**

Uma **PROCEDURE** é um bloco PL/SQL armazenado no banco, que executa uma ação (por exemplo, inserir, atualizar ou excluir registros).
Diferente das **functions**, **não retorna valor diretamente**, mas pode devolver dados por **parâmetros de saída**.

Estrutura básica

```
CREATE OR REPLACE PROCEDURE nome_procedure (
    parametro1 IN tipo,
    parametro2 OUT tipo
) IS
    -- Declarações de variáveis locais
BEGIN
    -- Corpo da procedure
    -- Lógica do programa
EXCEPTION
    -- Tratamento de erros
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erro: ' || SQLERRM);
END nome_procedure;
/
```

Exemplo 1: Inserir cliente

```
CREATE OR REPLACE PROCEDURE inserir_cliente (
    p_nome IN VARCHAR2,
    p_cidade IN VARCHAR2
) IS
```

```
BEGIN
    INSERT INTO clientes (id_cliente, nome, cidade)
    VALUES (seq_cliente.NEXTVAL, p_nome, p_cidade);

    DBMS_OUTPUT.PUT_LINE('Cliente inserido com sucesso!');
END inserir_cliente;
/
```

Execução:

```
BEGIN
    inserir_cliente('Maria', 'Curitiba');
END;
/
```

Exemplo 2: Procedure com parâmetro OUT

```
CREATE OR REPLACE PROCEDURE contar_clientes (
    p_total OUT NUMBER
) IS
BEGIN
    SELECT COUNT(*) INTO p_total FROM clientes;
END contar_clientes;
/
```

Execução:

```
DECLARE
    v_total NUMBER;
BEGIN
    contar_clientes(v_total);
    DBMS_OUTPUT.PUT_LINE('Total de clientes: ' || v_total);
END;
/
```

2. Conceito e Estrutura de uma Function

Uma **FUNCTION** retorna **sempre um valor** e pode ser chamada em consultas SQL, expressões e outros blocos PL/SQL.

Estrutura básica

```
CREATE OR REPLACE FUNCTION nome_function (
    parametro1 IN tipo
) RETURN tipo_retorno IS
    variavel_local tipo_retorno;
BEGIN
    -- Corpo da função
    RETURN variavel_local;
END nome_function;
/
```

Exemplo 3: Calcular idade

```
CREATE OR REPLACE FUNCTION calcular_idade (
    p_data_nascimento IN DATE
) RETURN NUMBER IS
    v_idade NUMBER;
BEGIN
    v_idade := TRUNC(MONTHS_BETWEEN(SYSDATE, p_data_nascimento) / 12);
    RETURN v_idade;
END calcular_idade;
/
```

Uso:

```
SELECT nome, calcular_idade(data_nasc) AS idade
FROM clientes;
```

Exemplo 4: Retornar bônus de um funcionário

```
CREATE OR REPLACE FUNCTION calcular_bonus (
    p_salario IN NUMBER
) RETURN NUMBER IS
BEGIN
    RETURN p_salario * 0.10;
END calcular_bonus;
/
```

Uso em bloco:

```
DECLARE
    v_bonus NUMBER;
```

```
BEGIN
    v_bonus := calcular_bonus(3000);
    DBMS_OUTPUT.PUT_LINE('Bônus: ' || v_bonus);
END;
/
```

3. Tipos de Parâmetros

Tipo	Descrição	Direção
IN	Valor de entrada (padrão).	Entrada
OUT	Retorna valor de saída.	Saída
IN OUT	Entrada e saída (modifica o valor recebido).	Ambos

Exemplo 5: Parâmetro IN OUT

```
CREATE OR REPLACE PROCEDURE dobrar_valor (
    p_num IN OUT NUMBER
) IS
BEGIN
    p_num := p_num * 2;
END dobrar_valor;
/
```

Execução:

```
DECLARE
    v_num NUMBER := 5;
BEGIN
    dobrar_valor(v_num);
    DBMS_OUTPUT.PUT_LINE('Valor dobrado: ' || v_num);
END;
/
```

4. Escopo de Variáveis

- Variáveis **locais**: declaradas dentro do bloco da procedure/function, só existem ali.
- Variáveis **globais**: podem ser declaradas em pacotes (veremos no módulo seguinte).

- **Parâmetros** são visíveis apenas dentro do bloco.

Exercícios Práticos

Exercício 1

Crie uma procedure chamada `aumentar_salario` que receba o **id do funcionário** e um **percentual de aumento**, e atualize o salário.

Dica:
Use `UPDATE funcionarios SET salario = salario * (1 + percentual/100)`.

Exercício 2
Crie uma function chamada `situacao_cliente` que receba o **valor total de compras** e retorne:

- `'VIP'` se o valor > 5000
- `'REGULAR'` caso contrário

Exercício 3
Crie uma procedure `contar_pedidos_cliente` que receba o **id do cliente** e retorne (via parâmetro OUT) o número de pedidos desse cliente.

Exercício 4
Crie uma function `valor_total_pedido` que receba o **id do pedido** e retorne o **valor total** (quantidade * preço_unitário) dos itens do pedido.

Exercício 5 (Desafio)
Crie uma procedure `atualizar_estoque` com um parâmetro **IN OUT** que receba o valor atual do estoque e subtraia a quantidade vendida, retornando o novo valor.