Linguagem Procedural de Banco de Dados

Aula 3: Estruturas de controle

Apresentação

Um dos maiores recursos da programação computacional é a possibilidade de programar estruturas condicionais, o que viabiliza a verificação de resultados e alternativas para o sistema. Nesta aula, falaremos também sobre o processo de repetição programada de passos, permitindo assim, maior eficiência na programação associada ao banco de dados. Esses recursos de programação também estão presentes nas linguagens procedurais (LP) de banco de dados.

Objetivo

- Definir estruturas condicionais simples e compostas em LP;
- Definir estruturas de repetição em LP;
- Aplicar exemplos de comandos básicos em LP.

Estruturas condicionais

As estruturas condicionais são responsáveis pelo controle de tomada de decisão por parte do programador, sempre considerando uma condição que será analisada pelo processador, podendo assumir valores verdadeiro ou falso.

Todas as linguagens de programação utilizam estruturas condicionais, e no caso da LP não seria diferente: o comando IF (Se) executa um determinado conjunto de ações de acordo com uma ou mais condições (comparação de valores por meio dos operadores relacionais), que sempre terá um resultante lógico.

Comando IF

O comando simples de condição tem a sintaxe a seguir:

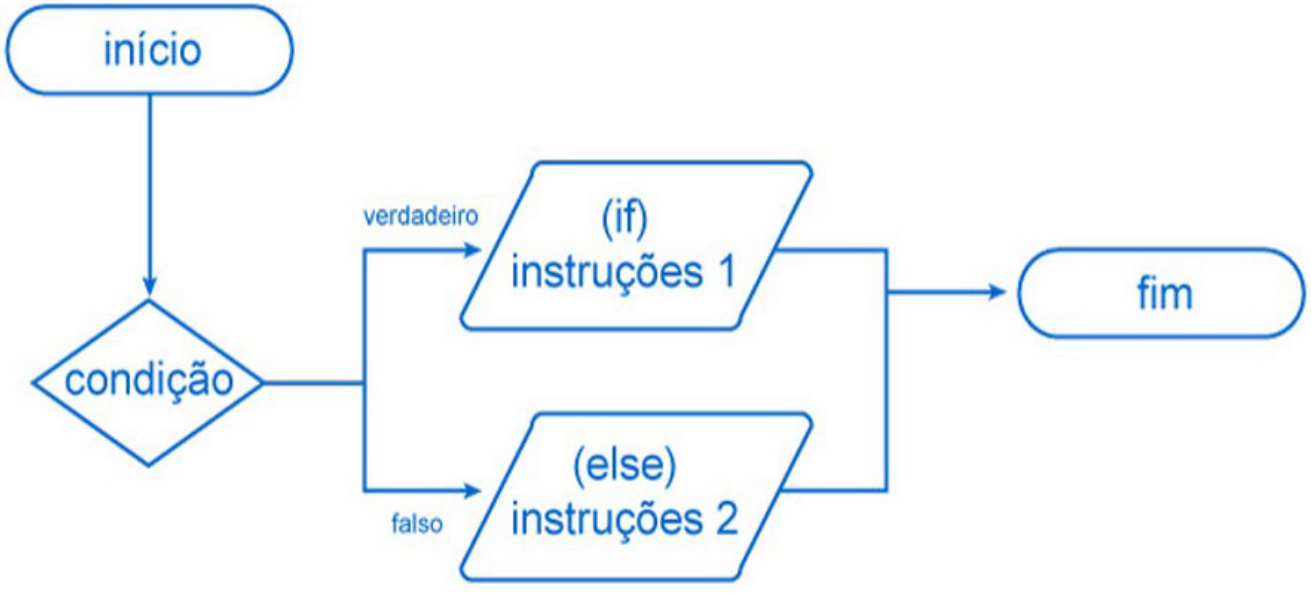
```
IF condição THEN
      {...As instruções que serão executadas vão aqui quando a condição é TRUE...}
END IF;
```

Veja o exemplo a seguir, combinando com comandos SQL:

```
IF userID <> 0 THEN
      UPDATE Usuario SET email = v_email WHERE user_id = userID;
END IF;
```

No caso de múltiplas condições, podemos obter um resultante lógico a partir do uso combinado de operadores lógicos.

Para tratarmos o resultante lógico falso podemos utilizar o comando ELSE que, conforme a figura a seguir demonstra, permitirá a execução de outro bloco de comando (Instruções 2).



Estrutura condicional simples com ELSE. Fonte: Novatech.

Sintaxe:

```
IF condição THEN
      {...As instruções que serão executadas vão aqui quando a condição é TRUE...}
ELSE
      {...As instruções que serão executadas vão aqui quando a condição é FALSE...}
END IF;
```

Exemplo combinado com SQL:

```
IF valor > 0 THEN
      INSERT INTO Usuario (Numero) VALUES (valor);
      RETURN 't';
ELSE
      RETURN 'f';
END IF;
```

Em caso de termos a necessidade de escolha entre várias alternativas, podemos utilizar a palavra-chave ELSIF (ou ELSE IF) que é responsável por introduzir condições adicionais à instrução básica, conforme sintaxe a seguir:

```
IF condição1 THEN
      {...As instruções que serão executadas vão aqui quando a condição é TRUE...}
ELSIF condição2 THEN
      {...As instruções que serão executadas vão aqui quando a condição é TRUE...}
END IF;
```

Analisando a sintaxe anterior, observamos que, caso a primeira condição resulte em FALSE ou NULL, a cláusula ELSIF realizará

o teste da outra condição especificada. Uma instrução IF pode ter mais de uma cláusula ELSIF. Se alguma condição for verdadeira, a sua sequência de declarações associadas e executadas passará para a próxima instrução.

Caso todas as instruções resultarem em FALSE, a cláusula ELSE será executada. Com base no que foi dito até o momento, vejamos um exemplo de cada uma das instruções para que o conceito seja melhor aproveitado. Veja o exemplo.

```
IF numero = 0 THEN
      resultado := 'zero';
ELSIF numero > 0 THEN
      resultado := 'positivo';
ELSIF numero < 0 THEN
      resultado := 'negativo';
ELSE
      -- a única outra possibilidade é que o número seja nulo
      resultado := 'NULL';
END IF;
```

A palavra-chave ELSIF também pode ser escrita ELSEIF ou ELSE IF.

Uma maneira alternativa de realizar a mesma tarefa é aninhar instruções IF-THEN-ELSE, como no exemplo a seguir.

```
IF sexo = 'm' THEN
      tipo := 'homem';
ELSE
      IF sexo = 'f' THEN
            tipo := 'mulher';
      END IF;
END IF;
```

Comando CASE (Caso)

Outra estrutura condicional que podemos utilizar é o comando CASE, que neste caso considera vários valores para uma determinada condição inicial.

Para o caso de termos mais de dois IF's, de certa forma, passa a ser mais aplicável a utilização de CASES.

A forma simples de CASE (ou SWITCH) fornece execução condicional com base na igualdade de operandos. A expressão de pesquisa é avaliada (uma vez) e comparada sucessivamente com cada expressão nas cláusulas WHEN. Se uma correspondência for encontrada, as instruções correspondentes serão executadas e o controle passará para a próxima instrução, após END CASE. Sendo assim, as expressões WHEN subsequentes não são avaliadas.

Se nenhuma correspondência for encontrada, as instruções ELSE serão executadas, mas, se ELSE não estiver presente, uma exceção CASE_NOT_FOUND será gerada.

```
      IF sexo = 'm' THEN
            tipo := 'homem';
      ELSE
IF sexo = 'f' THEN

                  tipo := 'mulher';
            END IF;
      END IF;
```

Sintaxe:

```
      CASE expressão_verificada
            WHEN expressao [, expressao [ ... ]] THEN
                  statements
            [ WHEN expressao [, expressao [ ... ]] THEN
                  argumento
            ... ]
            [ ELSE
                  argumento ]
      END CASE;
```

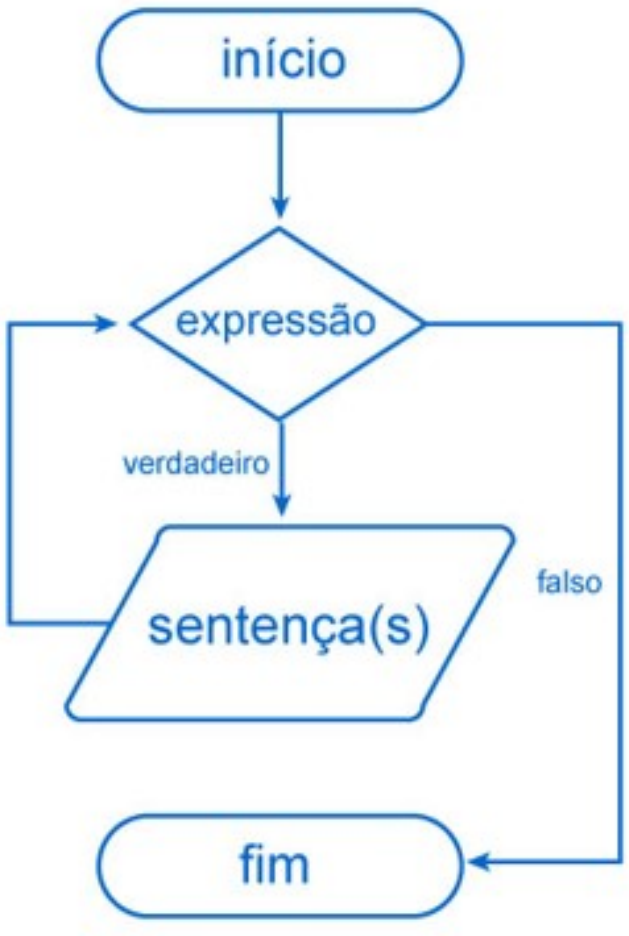
Veja o exemplo:

```
      CASE x
            WHEN 1, 2 THEN
                  msg := 'Um ou dois';
            ELSE
msg := 'Outro valor diferente de um ou dois';
      END CASE
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Estruturas de repetição

Em LP, podemos utilizar os Loops (Laços) para executar as mesmas estruturas de formas diferentes, com diferentes valores. Os loops realizam a execução de sequências de instruções até alcançar o objetivo. Observe na figura abaixo a representação geral dos laços.



Estrutura genérica de repetição. Fonte: Novatech.

Com as instruções LOOP, EXIT, CONTINUE, WHILE, FOR e FOREACH, você pode fazer com que sua função em LP repita uma série de comandos para loops simples.

Em Oracle, por exemplo, as estruturas de Loop são: Loop Básico, For Loop e While Loop.

O comando Loop...End Loop

Para cada execução do loop, todos os comandos internos serão executados e o controle retorna ao começo da instrução do loop.

Para que não tenhamos loops infinitos, precisamos utilizar declarações que finalizem o Loop ao encontrar uma resposta ou não, como é o caso do EXIT (Sair) e do CONTINUE (continuar).

Se nenhum rótulo for fornecido, o loop mais interno será encerrado e a instrução, após END LOOP, será executada em seguida. Se o rótulo for fornecido, este será o atual ou algum nível externo do loop ou bloco aninhado. Em seguida, o loop ou bloco nomeado é encerrado e o controle continua com a instrução, após o END correspondente do loop/ bloco.

```
[ rótulo ] LOOP
      argumentos
END LOOP [ rótulo ];
```

Atenção

Se WHEN for especificado, a saída do loop ocorrerá apenas se a expressão booleana for verdadeira. Caso contrário, o controle passa para a instrução após EXIT.

O comando EXIT pode ser usado com todos os tipos de loops, não se limitando ao uso com loops incondicionais. Quando usado com um bloco BEGIN, o comando EXIT passa o controle para a próxima instrução, após o final do bloco. Observe que um rótulo deve ser usado para esse propósito. Um comando EXIT sem rótulo nunca estará associado a um bloco BEGIN.

Veja os exemplos de estruturas de repetição em PostgreSQL:

```
LOOP
      -- Verificando
      IF count > 0 THEN
            EXIT; -- Sair do loop
      END IF;
END LOOP;
LOOP
      -- Calculando
      EXIT WHEN count > 100;
      CONTINUE WHEN count < 50;
      -- alguns cálculos para contagem IN [50 .. 100]
END LOOP;
```

Veja outro exemplo agora em Oracle:

```
01: DECLARE
02:     valor NUMBER := 0;
03: BEGIN
04;     LOOP
05:       DBMS_OUTPUT.PUT_LINE (' No interior do loop, temos o valor = ' || TO_CHAR(valor));
07:       valor := valor + 1;
08:       IF valor > 4 THEN
09:         EXIT;
10:       END IF;
11:     END LOOP;
12:     -- após a entrada na instrução EXIT, o controle é encerrado.
13:     DBMS_OUTPUT.PUT_LINE(' Após o Loop com valor = ' || TO_CHAR(valor));
14: END;

--Resultado exibido
No interior do loop, temos o valor = 0
No interior do loop, temos o valor = 1
No interior do loop, temos o valor = 2
No interior do loop, temos o valor = 3
Após o Loop com valor = 4
```

Saiba mais

Explicando o código acima:

Linha 01: Início da área de declaração.
Linha 02: Declarando a variável (valor) com o tipo NUMBER e atribuindo o valor 0.
Linha 03: Início do bloco de comandos.
Linha 04: Iniciando o laço de repetição.
Linha 05 e 06: Exibindo mensagem com o valor da variável.
Linha 07: Efetuando a atribuição de 1 para a variável valor.
Linha 08: Se a variável valor for maior que 40 então.
Linha 09: Saindo do Laço.
Linha 10: Finalizando a condição.
Linha 11: Finalizando o laço de repetição.
Linha 12: Comentários.
Linha 13: Exibindo uma mensagem na tela.
Linha 14: Finalizando o bloco BEGIN...END.

Outro exemplo em PostgreSql:

```
CREATE OR REPLACE FUNCTION LISTA01(INTEGER) RETURNS VARCHAR AS
$$
DECLARE
      valor integer := 0;
      mensagem varchar;
BEGIN
      LOOP
            valor := valor + 1;
      IF valor > $1 THEN
            EXIT;
      END IF;
      END LOOP;
      -- após a entrada na instrução EXIT, o controle é encerrado.
      mensagem := 'Após o Loop com valor = ' || ' ' || cast(valor AS varchar);
      return (mensagem);
END;
$$ LANGUAGE plpgsql;
select LISTA01(5);
-- Resultado exibido
Após o Loop com valor = 6
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

O comando While...End Loop

Também podemos utilizar laços com a instrução WHILE que repete uma sequência de instruções, desde que a expressão booleana seja avaliada como verdadeira. A expressão é verificada antes de cada entrada no corpo do loop.

Sintaxe:

```
[ <<label>> ]
WHILE boolean-expression LOOP
      statements
END LOOP [ label ];
```

Veja o exemplo em PostgreSQL:

```
WHILE valor > 0 AND certificado > 0 LOOP
      -- processando algo
END LOOP;
```

Agora outro exemplo completo em Oracle:

```
01: DECLARE
02:     v_mensagem VARCHAR(40):='Olá, Mundo';
03:     v_contador NUMBER;
04: BEGIN
05:     v_contador := 1;
06:     WHILE v_contador <=3 LOOP
07:         DBMS_OUTPUT.PUT_LINE (v_mensagem);
08:         v_contador := v_contador + 1;
09:     END LOOP;
10: END;

-- Resultado exibido
Olá, Mundo
Olá, Mundo
Olá, Mundo
```

Saiba mais

Explicando o código acima:

Linha 01: Início da área de declaração.
Linha 02: Declarando a variável v_mensagem com o tipo Varchar com o tamanho de 40 caracteres alfanuméricos e atribuindo a mensagem 'Olá, Mundo'.
Linha 03: Declarando a variável v_contador com o tipo Number.
Linha 04: Início do bloco BEGIN...END.
Linha 05: Atribuindo o valor inicial 1 à variável v_contador.
Linha 06: Iniciando o laço a partir da condição verdadeira (v_valor <= 3).
Linha 07: Exibe a mensagem da variável v_mensagem.
Linha 08: Acumula a variável v_contador em uma unidade.
Linha 09: Efetua a repetição do laço de acordo com avaliação da condição (verdadeiro repete a sequência de comandos, falso finaliza o laço).
Linha 10: Finalizando o bloco BEGIN...END.
O comando FOR LOOP...END LOOP

O laço, utilizando FOR, permite que determinada relação de comandos seja executada n vezes. Ou seja, os comandos existentes serão executados até que a variável contadora atinja seu valor final.

Sintaxe:

```
[ <<label>> ]
FOR name IN [ REVERSE ] expression .. expression [ BY expression ] LOOP
      statements
END LOOP [ label ];
```

A variável contadora não deve ser declarada previamente no espaço de declarações. No entanto, essa variável deixa de existir após a execução do comando END...LOOP.

Se a cláusula REVERSE for especificada, o valor da etapa será subtraído, em vez de adicionado, após cada iteração.

Alguns exemplos em PostgreSQL:

```
FOR i IN 1..10 LOOP
-- Será considerado os valores 1,2,3,4,5,6,7,8,9,10 dentro do loop
END LOOP;

FOR i IN REVERSE 10..1 LOOP
-- Será considerado os valores 10,9,8,7,6,5,4,3,2,1 dentro do loop
END LOOP;

FOR i IN REVERSE 10..1 BY 2 LOOP
-- Será considerado os valores 10,8,6,4,2 dentro do loop

END LOOP;
```

Outro exemplo com mais detalhes em Oracle:

```
01:SET SERVEROUTPUT ON;
02:DECLARE
03:     v_contador NUMBER(38);
04:     v_contadorInicial NUMBER(38);
05:     v_contadorFinal NUMBER(38);
06:BEGIN
07:     v_contadorInicial :=1;
08:     v_contadorfinal := 5;
09:     FOR v_contador in v_contadorInicial .. v_contadorfinal
10:     DBMS_OUTPUT.PUT_LINE (v_contador);
11:     LOOP
12:     END LOOP;
13:END;
-- Resultado exibido
1
2
3
4
5
];
```

Saiba mais

Explicando o código acima:

Linha 01: Definindo o padrão de saída.
Linha 02: Início da área de declaração.
Linha 03 a 05: Declarando as variáveis v_contador, v_contadorInicial,v_contadorFinal com o tipo NUMBER(38) (limite máximo da variável).
Linha 06: Início do bloco BEGIN...END.
Linha 07: Atribuindo o valor 1 para a variável v_contadorInicial.
Linha 08: Atribuindo o valor 10 para a variável v_contadorFinal.
Linha 09: Iniciando o laço de repetição definido (FOR).
Linha 10: Exibe uma mensagem na tela durante a execução do laço.
Linha 11: Comando para Loop.
Linha 12: Fim do Laço.
Linha 13: Finalizando o bloco BEGIN...END.

Exemplo anterior apenas com o resultado final em PostgreSql:

```
CREATE OR REPLACE FUNCTION LISTA02(INTEGER) RETURNS VARCHAR AS
$$
DECLARE
      contador integer;
      contadorincial integer := 1;
      contadorfinal integer := $1;
      mensagem varchar;

      BEGIN
         FOR contador IN contadorincial .. contadorfinal LOOP
            mensagem := 'Valor final: ' || cast(contador AS varchar);
         END LOOP;
         -- após a entrada na instrução EXIT, o controle é encerrado.
         return (mensagem);
      END;
$$ LANGUAGE plpgsql;

select LISTA02(5);

-- Resultado exibido
Valor final 5
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Atividade

1. Para a estrutura condicional CASE, se uma correspondência for encontrada, as instruções correspondentes serão executadas e o controle passará para a próxima instrução, após END CASE.

Sobre a cláusula WHEN é correto afirmar que:

a) A expressões WHEN subsequentes não serão avaliadas.

b) Devemos usar WHEN em estrutura de repetição apenas.

c) Devemos utilizar ELSE para o caso.

d) A exceção CASE_NOT_FOUND será gerada.

e) Não será aplicada para CASEs.

---

2. Sobre a variável contadora no laço FOR...LOOP...END LOOP é correto afirmar que:

a) Será uma variável global.

b) Deverá ser uma constante.

c) Apenas será reconhecida no bloco de repetição.

d) Sua finalidade atribui valores a outras variáveis.

e) Podemos utilizá-la fora do laço de repetição.

---

3. Sobre o comando EXIT utilizado em laços de repetição, é correto afirmar:

a) Apenas utilizado no laço LOOP...END LOOP.

b) Quando executado é acionada a área EXCEPTION do bloco BEGIN...END.

c) EXIT pode ser usado com todos os tipos de loops; não se limita ao uso com loops incondicionais.

d) Utilizado também nas estruturas condicionais.

e) Assim como CONTINUE, não deve ser utilizado em laços FOR...LOOP...END LOOP.

Referências

Notas

Fanderuff, Damaris. **Dominando o Oracle 9i: Modelagem e Desenvolvimento** [BV:PE]. 1ª Ed. São Paulo: Editora Pearson, 2003. 1.
Disponível em: https://plataforma.bvirtual.com.br/Leitor/Loader/283/pdf

PostgresSQl: Documentation 12 ChapTer 45 PL – SQL Procedural Language
Disponível em: https://www.postgresql.org/docs/12/plpgsql.html
Acesso em: 08.08.2020

LP para Desenvolvedores | Oracle Brasil
Disponível em: https://www.oracle.com/br/database/technologies/appdev/plsql.html
Acesso em: 10.08.2020

Próxima aula

- Cursores (definição, tipos, estruturas);
- Uso de cursores em operações de repetição;
- Aplicação com SQL.

Explore mais

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.