FIAP ON

PROGRAMAÇÃO EM
BANCO DE DADOS

ESTRUTURAS DE CONTROLE

MILTON GOYA

2

LISTA DE QUADROS

Quadro 2.1 – Resultado do operador lógico AND 13
Quadro 2.2 – Resultado do operador lógico OR 13
Quadro 2.3 – Resultado do operador NOT 14
Quadro 2.4 – Saída da tabela populada pelo LOOP aninhado 20

LISTA DE CÓDIGOS-FONTE

Código-fonte 2.1 – Exemplo de bloco PL/SQL exibindo campo com 18 bytes.....5
Código-fonte 2.2 – Exemplo de bloco PL/SQL com erro devido a tamanho de variável errado.....6
Código-fonte 2.3 – Exemplo de bloco PL/SQL com uso do atributo %TYPE.....6
Código-fonte 2.4 – Sintaxe da declaração de variável.....7
Código-fonte 2.5 – Exemplos de declaração de variáveis usando %TYPE.....7
Código-fonte 2.6 – Sintaxe da instrução IF THEN.....8
Código-fonte 2.7 – Exemplo de bloco PL/SQL com IF THEN.....8
Código-fonte 2.8 – Sintaxe da instrução IF THEN ELSE.....9
Código-fonte 2.9 – Exemplo de bloco PL/SQL com IF THEN ELSE.....9
Código-fonte 2.10 – Sintaxe da instrução IF THEN ELSIF.....10
Código-fonte 2.11 – Exemplo de bloco PL/SQL com IF THEN ELSIF.....11
Código-fonte 2.12 – Exemplo de estrutura de decisão com o operador lógico AND.12
Código-fonte 2.13 – Exemplo de estrutura de decisão com o operador lógico OR...12
Código-fonte 2.14 – Sintaxe de LOOP Simples.....15
Código-fonte 2.15 – Exemplo de LOOP básico.....16
Código-fonte 2.16 – Sintaxe de LOOP FOR.....17
Código-fonte 2.17 – Exemplo de LOOP FOR.....17
Código-fonte 2.18 – Sintaxe de LOOP WHILE.....18
Código-fonte 2.19 – Exemplo de LOOP FOR.....18
Código-fonte 2.20 – Exemplo de LOOP FOR.....19

SUMÁRIO

2 ESTRUTURAS DE CONTROLE 5
2.1 Mais sobre tipos de dados.....5
2.2 Estruturas de seleção.....7
2.2.1 IF THEN 8
2.2.2 IF THEN ELSE 9
2.2.3 IF THEN ELSIF 10
2.2.4 AND OR 11
2.3 Estruturas de Repetição.....14
2.3.1 LOOP BÁSICO 15
2.3.2 LOOP FOR 16
2.3.3 LOOP WHILE 18
2.3.4 LOOP ANINHADO 19
CONCLUSÃO.....21
REFERÊNCIAS.....22

2 ESTRUTURAS DE CONTROLE

2.1 Mais sobre tipos de dados

Antes de começarmos a falar de estruturas de controle é importante que saibamos mais alguns detalhes sobre os tipos de dados.

Aprendemos que devemos declarar o nome da variável, seu tipo de dados e, eventualmente, precisamos iniciá-la com um valor. Em um exemplo simples tempos:

```
CREATE TABLE tabela1
(col1 VARCHAR2(18));

INSERT INTO tabela1
  VALUES ('Campo com 18 bytes');

SET SERVEROUTPUT ON

DECLARE
  v_col1 VARCHAR2(18);
BEGIN
  SELECT col1 INTO v_col1
    FROM tabela1;
  DBMS_OUTPUT.PUT_LINE ('Valor = ' || v_col1);
END;
/
```

Código-fonte 2.1 – Exemplo de bloco PL/SQL exibindo campo com 18 bytes
Fonte: Elaborado pelo autor (2020)

No entanto, o que acontece se alterarmos o tamanho de uma coluna da tabela? O bloco PL/SQL continuará funcionando normalmente? Vejamos em um teste simples:

```
TRUNCATE TABLE tabela1;

ALTER TABLE tabela1
MODIFY col1 VARCHAR2(30);

INSERT INTO tabela1
  VALUES ('Tamanho alterado para 30 bytes');

SET SERVEROUTPUT ON

DECLARE
  v_col1 VARCHAR2(18);
BEGIN
  SELECT col1 INTO v_col1
    FROM tabela1;
```

```
        DBMS_OUTPUT.PUT_LINE ('Valor = ' || v_col1);
      END;
      /

      ORA-06502: PL/SQL: numeric or value error: character
string buffer too small
      ORA-06512: at line 4
```

Código-fonte 2.2 – Exemplo de bloco PL/SQL com erro devido a tamanho de variável errado
Fonte: Elaborado pelo autor (2020)

Perceba que ocorreu um erro (ORA-06502) no bloco PL/SQL, porque o tamanho da coluna da tabela é maior do que o tamanho da variável definida na sessão de DECLARE.

Uma forma de evitarmos esse tipo de problema é declararmos a variável com o atributo %TYPE.

Segundo a Oracle (2016), o atributo %TYPE permite que você declare uma constante, variável, elemento de coleção, campo de registro ou subprograma para que ela seja do mesmo tipo de dados que uma variável ou coluna previamente declarada mesmo que não saiba qual é esse tipo. A vantagem de usarmos o atributo %TYPE é que se a declaração do item referenciado for alterada, a declaração do item de referência muda de acordo com ele.

No nosso caso, se tivéssemos usado o atributo %TYPE na declaração do tipo da variável, ela assumiria a nova definição automaticamente. Vamos ver o exemplo novamente, mas, desta vez, usaremos o %TYPE.

```
      DECLARE
        v_col1 tabela1.col1%TYPE;
      BEGIN
        SELECT col1 INTO v_col1
          FROM tabela1;
        DBMS_OUTPUT.PUT_LINE ('Valor = ' || v_col1);
      END;
      /
```

Código-fonte 2.3 – Exemplo de bloco PL/SQL com uso do atributo %TYPE
Fonte: Elaborado pelo autor (2020)

Perceba que desta vez não ocorreu o erro ORA-06512. Isso aconteceu porque a declaração da variável V_COL1 consultou o dicionário de dados do banco de dados e obteve o tipo de dados e tamanho da coluna COL1 da tabela TABELA1. Podemos ler a declaração da variável como “V_COL1 terá o mesmo tipo de dados e tamanho da coluna COL1 da tabela TABELA1”. É importante destacar que o uso de %TYPE

faz com que o campo declarado herde o tipo de dados, tamanho e restrições (ou CONSTRAINTS) do campo original. O campo não herda o valor inicial do item referenciado.

O atributo %TYPE pode ser utilizado para declarar variáveis com a mesma estrutura de uma tabela ou de uma variável já existente.

A sintaxe é exibida abaixo:

```
identificador [CONSTANT] {tabela.coluna%type | variavel%type}
[NOT NULL] [:= valor para inicialização | expr default]
```

Código-fonte 2.4 – Sintaxe da declaração de variável
Fonte: ORACLE (2016)

Alguns exemplos:

```
v_nome emp.ename%type; -- declaração da variável com a mesma
estrutura da coluna ename da tabela emp.

v_balance number(7,2);

v_min_balance v_balance%type; -- declaração da variável com a
mesma estrutura da variável declarada anteriormente.
```

Código-fonte 2.5 – Exemplos de declaração de variáveis usando %TYPE
Fonte: ORACLE (2016)

Relembrando que é uma boa prática de programação a adoção de uma convenção de nomeação para variáveis, por exemplo: o prefixo v_ representa uma variável; c_ representa uma constante.

2.2 Estruturas de seleção

Para Dillon et al. (2013), as estruturas de controle permitem que o desenvolvedor estabeleça o fluxo lógico de instruções que serão executadas. Para isso, podem ser utilizadas as estruturas de controle para repetição de blocos do programa (LOOP) e as estruturas de controle para avaliação de condições e seleção (IF).

As estruturas de seleção possibilitam que o fluxo de processamento das instruções PL/SQL seja direcionado de acordo com a condição especificada.

Existem três maneiras para se utilizar a instrução IF:

- IF THEN

- IF THEN ELSE
- IF THEN ELSIF

2.2.1 IF THEN

```
IF (condição) THEN
    conjunto de instruções;
END IF;
```

Código-fonte 2.6 – Sintaxe da instrução IF THEN
Fonte: ORACLE (2016)

onde,

condição é uma expressão ou variável Booleana (TRUE, FALSE ou NULL). Está associada a uma sequência de instruções, que será executada se e somente se a expressão for avaliada como TRUE.

THEN é uma cláusula que associa a expressão Booleana que a precede com a sequência de instruções posterior.

instruções podem ser uma ou mais instruções SQL ou PL/SQL. Podem incluir mais instruções IF contendo diversos IFs, ELSEs e ELSIFs aninhados.

Vejamos cada um dos casos

No caso do **IF THEN**, se o teste de avaliação da condição retornar verdadeiro o conjunto de instruções será realizado; caso contrário, o bloco de seleção é encerrado.

Veja o exemplo abaixo:

```
DECLARE
  v_col1    tabela1.col1%TYPE;
  v_tamanho NUMBER(3);
BEGIN
  SELECT LENGTH(col1), col1 INTO v_tamanho, v_col1
    FROM tabela1;
  IF v_tamanho > 25 THEN
     DBMS_OUTPUT.PUT_LINE ('Texto = ' || v_col1);
  END IF;
END;
/
```

Código-fonte 2.7 – Exemplo de bloco PL/SQL com IF THEN
Fonte: Elaborado pelo autor (2020)

Você pode observar que o bloco PL/SQL lê a coluna COL1 da tabela TABELA1. Usando a função LENGTH, ele determina o tamanho do texto armazenado dentro da coluna COL1 e armazena esse tamanho na variável V_TAMANHO, também armazena o texto existente na coluna COL1 na variável V_COL1. Logo após isso, usa a instrução IF para testar se o valor armazenado na variável V_TAMANHO é maior que 25, se for maior que 25 exibe uma mensagem e encerra a estrutura de decisão.

2.2.2 IF THEN ELSE

Vejamos o IF THEN ELSE

```
IF (condição) THEN
   conjunto de instruções 1;
ELSE
   conjunto de instruções 2;
END IF;
```

Código-fonte 2.8 – Sintaxe da instrução IF THEN ELSE
Fonte: ORACLE (2016)

Nesse caso, se o teste de avaliação da condição retornar verdadeiro o conjunto de instruções 1 será realizado; caso contrário, será realizado o conjunto de instruções 2.

Considere o código abaixo

```
DECLARE
  v_col1    tabela1.col1%TYPE;
  v_tamanho NUMBER(3);
BEGIN
  SELECT LENGTH(col1), col1 INTO v_tamanho, v_col1
    FROM tabela1;
  IF v_tamanho > 25 THEN
     DBMS_OUTPUT.PUT_LINE ('Texto = ' || v_col1);
  ELSE
     DBMS_OUTPUT.PUT_LINE ('Texto menor ou igual a 25');
  END IF;
END;
/
```

Código-fonte 2.9 – Exemplo de bloco PL/SQL com IF THEN ELSE
Fonte: Elaborado pelo autor (2020)

Como no programa anterior, o bloco PL/SQL lê a coluna COL1 da tabela TABELA1. Usando a função LENGTH, ele determina o tamanho do texto armazenado

dentro da coluna COL1 e armazena esse tamanho na variável V_TAMANHO, também armazena o texto existente na coluna COL1 na variável V_COL1. Logo após isso, usa a instrução IF para testar se o valor armazenado na variável V_TAMANHO é maior que 25, se for maior que 25 exibe uma mensagem, caso não seja maior que 25 então exibe a mensagem “Texto menor ou igual a 25” e encerra a estrutura de decisão.

2.2.3 IF THEN ELSIF

Examinemos o IF THEN ELSIF

```
IF (condição1 ) THEN
          conjunto de instruções 1;
ELSIF (condição 2)
          conjunto de instruções  2 ;
...
ELSE
           conjunto de instruções n;
END IF;
```

Código-fonte 2.10 – Sintaxe da instrução IF THEN ELSIF
Fonte: ORACLE (2016)

Nesse caso, se o teste de avaliação da condição retornar verdadeiro o conjunto de instruções 1 será realizado; caso contrário, será realizado o teste de avaliação da condição 2; se o resultado for verdadeiro, será realizado o conjunto de instruções 2; caso contrário, será realizado o teste de avaliação da condição 3, e assim por diante. Se nenhuma das condições testadas resultar verdadeiro será realizado o conjunto de instruções previsto após o ELSE.

Considere o código abaixo:

```
DECLARE
  v_col1    tabela1.col1%TYPE;
  v_tamanho NUMBER(3);
BEGIN
  SELECT LENGTH(col1), col1 INTO v_tamanho, v_col1
    FROM tabela1;
  IF v_tamanho > 25 THEN
     DBMS_OUTPUT.PUT_LINE ('Texto = ' || v_col1);
  ELSIF v_tamanho > 20 THEN
     DBMS_OUTPUT.PUT_LINE ('Texto maior que 20');
  ELSIF v_tamanho > 15 THEN
     DBMS_OUTPUT.PUT_LINE ('Texto maior que 15');
  ELSE
     DBMS_OUTPUT.PUT_LINE ('Texto menor ou igual a 15');
```

```
  END IF;
END;
/
```

Código-fonte 2.11 – Exemplo de bloco PL/SQL com IF THEN ELSIF
Fonte: Elaborado pelo autor (2020)

Como no programa anterior, o bloco PL/SQL lê a coluna COL1 da tabela TABELA1. Usando a função LENGTH, ele determina o tamanho do texto armazenado dentro da coluna COL1 e armazena esse tamanho na variável V_TAMANHO, também armazena o texto existente na coluna COL1 na variável V_COL1. Logo após isso, usa a instrução IF para testar se o valor armazenado na variável V_TAMANHO é maior que 25, se for maior que 25 exibe uma mensagem, caso não seja maior que 25, mas seja maior que 20 então exibe a mensagem “Texto maior que 20”, se não for maior que 20 mas for maior que 15, então exibe a mensagem “Texto maior que 15”, se nenhuma dessas condições forem avaliadas como verdadeiras, então exibe a mensagem “Texto menor ou igual a 15” e encerra a estrutura de decisão.

Importante: você pode usar qualquer quantidade de cláusulas ELSIF, mas só pode haver, no máximo, uma cláusula ELSE.

2.2.4 AND OR

Você pode testar mais de uma condição dentro de uma estrutura de decisão. Para isso, usamos os operadores lógicos AND (e) e OR (ou). Ao usarmos o operador lógico AND, a estrutura condicional só será avaliada como TRUE se todas as condições testadas forem verdadeiras. Ao usarmos o operador lógico OR, a estrutura condicional será avaliada como TRUE se, pelo menos, uma condição for verdadeira.

Veja no exemplo abaixo:

```
DECLARE
  v_tamanho NUMBER(3);
BEGIN
  SELECT LENGTH(col1) INTO v_tamanho
    FROM tabela1;
  IF v_tamanho > 25 AND
     TO_CHAR(SYSDATE, 'YYYY') = 2017 THEN
     DBMS_OUTPUT.PUT_LINE ('Maior que 25 bytes e ano 2017');
  END IF;
END;
/
```

Código-fonte 2.12 – Exemplo de estrutura de decisão com o operador lógico AND
Fonte: Elaborado pelo autor (2020)

Nesse exemplo, o bloco PL/SQL lê a coluna COL1 da tabela TABELA1. Usando a função LENGTH, ele determina o tamanho do texto armazenado dentro da coluna COL1 e armazena esse tamanho na variável V_TAMANHO. Logo após isso, usa a instrução IF para testar se o valor armazenado na variável V_TAMANHO é maior que 25 **E** se o ano corrente é 2017. Se ambas as condições forem atendidas (texto maior que vinte e cinco e ano 2017), então exibe a mensagem “Maior que 25 bytes e ano 2017” e encerra a estrutura de decisão. Reafirmando, a mensagem só será exibida se ambas as afirmações forem verdadeiras.

Vamos alterar o operador lógico AND para o operador lógico OR no exemplo anterior.

```
DECLARE
  v_tamanho NUMBER(3);
BEGIN
  SELECT LENGTH(col1) INTO v_tamanho
    FROM tabela1;
  IF v_tamanho > 25 OR
     TO_CHAR(SYSDATE, 'YYYY') = 2017 THEN
     DBMS_OUTPUT.PUT_LINE ('Maior que 25 bytes ou ano 2017');
  END IF;
END;
/
```

Código-fonte 2.13 – Exemplo de estrutura de decisão com o operador lógico OR
Fonte: Elaborado pelo autor (2020)

Quase nada mudou do exemplo anterior para este, o bloco PL/SQL lê a coluna COL1 da tabela TABELA1. Usando a função LENGTH ele determina o tamanho do texto armazenado dentro da coluna COL1 e armazena esse tamanho na variável V_TAMANHO. Logo após isso, usa a instrução IF para testar se o valor armazenado na variável V_TAMANHO é maior que 25 **OU** se o ano corrente é 2017. Se uma das condições forem atendidas (texto maior que vinte e cinco ou ano 2017) então exibe a mensagem “Maior que 25 bytes ou ano 2017” e encerra a estrutura de decisão. Reafirmando, a mensagem será exibida se qualquer uma das afirmações forem verdadeiras.

Podemos, então, criar condições compostas combinando as condições com os operadores lógicos AND, OR e NOT. Observe o quadro abaixo:

Operador AND		
Expressão 1	**Expressão 2**	**Resultado**
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
NULL	FALSE	FALSE
FALSE	NULL	FALSE

Quadro 2.1 – Resultado do operador lógico AND
Fonte: ORACLE (2016)

No quadro pode ser observado que, se a EXPRESSÃO 1 for avaliada como verdadeira (TRUE) **e** a EXPRESSÃO 2 for avaliada como verdadeira (TRUE) então a estrutura de decisão retornará o valor verdadeiro (TRUE) e executará a operação. Qualquer outra situação será avaliada como falsa (FALSE) e não executará a operação. É interessante notar que NULL AND TRUE sempre será avaliado como NULL, porque não se sabe se o segundo operando será avaliado como TRUE ou não.

Vejamos como se comporta o operador lógico OR:

Operador OR		
Expressão 1	**Expressão 2**	**Resultado**
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
NULL	FALSE	FALSE
FALSE	NULL	FALSE

Quadro 2.2 – Resultado do operador lógico OR
Fonte: ORACLE (2016)

No quadro pode ser observado que, se a EXPRESSÃO 1 for avaliada como verdadeira (TRUE) **ou** a EXPRESSÃO 2 for avaliada como verdadeira (TRUE), então a estrutura de decisão retornará o valor verdadeiro (TRUE) e executará a operação. A estrutura de decisão só retornará o valor falso (FALSE) se ambas as expressões

forem avaliadas como falso (FASE) ou se uma das expressões for avaliada como falso (FALSE) e a outra expressão for avaliada como nulo (NULL).

Em termos de precedência de operadores, FALSE tem precedência sobre uma condição AND e TRUE tem precedência sobre uma condição OR.

Vejamos como se comporta o operador NOT:

Operador NOT		
Expressão	**Resultado**	**Explicação**
TRUE	FALSE	O operador NOT faz com que o valor booleano TRUE seja avaliado como FALSE.
FALSE	TRUE	O operador NOT faz com que o valor booleano FALSE seja avaliado como TRUE
NULL	NULL	O operador NOT aplicado a um valor nulo (NULL) seja avaliado como nulo (NULL)

Quadro 2.3 – Resultado do operador NOT
Fonte: ORACLE (2016)

No quadro pode ser observado que, se o valor da expressão for verdadeiro (TRUE), o operador NOT produz o resultado falso (FALSE). Se o valor da expressão for falso (FALSE), o operador NOT produz o resultado verdadeiro (TRUE). Se o valor da expressão for nulo (NULL) então o operador NOT produz o resultado nulo (NULO).

2.3 Estruturas de Repetição

Estruturas de repetição também são conhecidas pelo nome de laço ou pelos termos em inglês ***LOOP*** ou ***LOOPING***.

Em determinadas situações temos a necessidade que um programa, ou parte dele, seja executado várias vezes. Reiniciar o programa para cada repetição não é uma solução muito prática, e algumas vezes é inviável. Uma solução comum é a utilização de estruturas de repetição.

Segundo Puga et al. (2015), o conceito de repetição (ou *LOOPING*) é utilizado quando se deseja repetir um certo trecho de instruções por um número de vezes. O

número de repetições pode ser conhecido anteriormente ou não, mas necessariamente precisa ser finito.

Nem todas as estruturas de repetição possuem recursos para fazer a contagem do número de vezes que o laço deverá ser repetido, nessas situações, deve-se utilizar uma variável de apoio sempre do tipo inteiro.

O PL/SQL oferece diversos recursos para estruturar laços de repetição:

- **LOOP básico** para fornecer ações repetitivas sem condições gerais.
- **LOOP FOR** para fornecer controle iterativo para ações com base em uma contagem.
- **LOOP WHILE** para fornecer controle iterativo para ações com base em uma condição.

A Instrução **EXIT** pode ser usada para terminar um laço de repetição.

2.3.1 LOOP BÁSICO

O ***LOOP básico*** permite a execução de sua instrução pelo menos uma vez, entretanto, se a condição **EXIT** for colocada no início do loop, antes de qualquer outra instrução executável, e ela for verdadeira, ocorrerá a saída do loop e as instruções jamais serão executadas. É importante notar que, sem a instrução **EXIT**, o **LOOP** nunca terminaria.

```
LOOP
 conjunto de instruções;
  EXIT [WHEN condição];
END LOOP;
```

Código-fonte 2.14 – Sintaxe de LOOP Simples
Fonte: ORACLE (2016)

onde,

LOOP é o delimitador de início do laço.

CONJUNTO DE INSTRUÇÔES são as instruções a serem executadas em cada iteração.

EXIT é o ponto de saída do laço.

[WHEN CONDIÇÂO] indica a condição de saída do laço

END LOOP; é o delimitador de fim do laço.

Veja o exemplo de LOOP básico abaixo:

```
DECLARE
  v_contador NUMBER(2) :=1;
BEGIN
  LOOP
    INSERT INTO tabela1
    VALUES ('Inserindo texto numero ' || v_contador);
    v_contador := v_contador + 1;
  EXIT WHEN v_contador > 10;
  END LOOP;
END;
/
```

Código-fonte 2.15 – Exemplo de LOOP básico
Fonte: Elaborado pelo autor (2020)

Nesse exemplo, o bloco PL/SQL define uma variável denominada V_CONTADOR e inicia a variável com o valor 1 (um). O programa, então, inicia um laço de inserção de dados na tabela TABELA1. O texto “Inserindo texto numero” concatenado com o número do contador será inserido na tabela. Em seguida o valor existente em V_CONTADOR é acrescido de 1 (um). O programa testa o valor atual do contador e sai do laço caso o valor seja superior a 10 (dez).

É interessante notar que podemos usar o comando EXIT como uma ação dentro de uma instrução IF ou como uma instrução independente dentro do laço. Quando a instrução EXIT é encontrada, a condição na cláusula WHEN é avaliada, se a condição produzir TRUE, o loop finalizará e o controle passará para a próxima instrução após o loop. Um loop básico pode conter várias instruções EXIT.

2.3.2 LOOP FOR

Já o **LOOP FOR** realiza as iterações de acordo com a instrução de controle que precede a palavra-chave **LOOP**.

```
FOR contador in [REVERSE] limite_inferior..limite_superior LOOP
  conjunto de instruções;
  . . .
END LOOP;
```

Código-fonte 2.16 – Sintaxe de LOOP FOR
Fonte: ORACLE (2016)

onde,

CONTADOR é um contador numérico inteiro declarado implicitamente. O valor do contador aumenta ou diminui automaticamente em 1 a cada iteração do loop até o limite superior ou inferior a ser alcançado. O valor do contador só diminuirá se a palavra-chave REVERSE for usada.

REVERSE faz o contador decrescer a cada iteração a partir do **limite superior** até o **limite inferior**. É importante notar que o limite inferior ainda é referenciado primeiro.

LIMITE_INFERIOR especifica o limite inferior da faixa de valores do contador.

LIMITE_SUPERIOR especifica o limite superior da faixa de valores do contador.

LOOP é o delimitador de início do laço.

CONJUNTO DE INSTRUÇÔES são as instruções a serem executadas em cada iteração.

END LOOP; é o delimitador de fim do laço.

Veja o exemplo de LOOP FOR abaixo:

```
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO tabela1
    VALUES ('Inserindo texto numero ' || i);
  END LOOP;
END;
/
```

Código-fonte 2.17 – Exemplo de LOOP FOR
Fonte: Elaborado pelo autor (2020)

No nosso exemplo, o bloco PL/SQL define implicitamente uma variável denominada **i**, a variável recebe o valor 1 (um) na primeira iteração do programa. Dentro do laço é efetuada uma operação de INSERT a cada iteração. O texto “Inserindo texto numero” concatenado com o número do contador será inserido na tabela. O valor do contador **i** será incrementado em 1 (um) e o processo é repetido até que o limite superior seja atingido.

É interessante notar que a sequência de instruções é executada sempre que o contador é incrementado, conforme determinado pelos dois limites. Os limites superior e inferior da faixa do LOOP podem ser literais, variáveis ou expressões, mas devem ser avaliados para inteiros. Se o limite inferior da faixa do loop for avaliado para um inteiro maior do que o limite superior, a sequência de instruções não será executada.

2.3.3 LOOP WHILE

O **LOOP WHILE** pode ser usado para repetir uma sequência de instruções até que a condição para controle não seja mais verdadeira. A condição é avaliada ao início de cada iteração, sendo assim, se a condição for falsa no início do LOOP, nenhuma iteração futura será executada.

```
WHILE condição LOOP
  conjunto de instruções;
    . . .
END LOOP;
```

Código-fonte 2.18 – Sintaxe de LOOP WHILE
Fonte: ORACLE (2016)

onde,

CONDIÇÃO é uma expressão ou variável booleana.

CONJUNTO DE INSTRUÇÔES são as instruções a serem executadas em cada iteração.

END LOOP; é o delimitador de fim do laço.

Veja o exemplo de LOOP WHILE abaixo:

```
DECLARE
  v_contador NUMBER(2) :=1;
BEGIN
  WHILE v_contador <= 10 LOOP
    INSERT INTO tabela1
    VALUES ('Inserindo texto numero ' || v_contador);
    v_contador := v_contador + 1;
  END LOOP;
END;
/
```

Código-fonte 2.19 – Exemplo de LOOP FOR
Fonte: Elaborado pelo autor (2020)

Nesse exemplo, o bloco PL/SQL define uma variável denominada V_CONTADOR e inicia a variável com o valor 1 (um). O programa, então, inicia um laço de inserção de dados na tabela TABELA1. O valor de V_CONTADOR é verificado e enquanto for menor ou igual a 10 (dez) o programa ficará executando o laço de inserção. O texto “Inserindo texto numero” concatenado com o número do contador será inserido na tabela. Em seguida o valor existente em V_CONTADOR é acrescido de 1 (um). O programa sai do laço, caso o valor seja superior a 10 (dez).

Se as variáveis envolvidas nas condições não se alterarem no curso do corpo do LOOP, a condição permanecerá TRUE e o LOOP não terminará. Se a condição produzir NULL, o LOOP será ignorado e o controle passará para a próxima instrução.

2.3.4 LOOP ANINHADO

Em algumas situações podemos ter a necessidade de aninhar LOOPs. Podemos aninhar loops para vários níveis. Você pode aninhar loops básicos, FOR e WHILE um dentro do outro. A terminação de um loop aninhado não terminará o loop delimitado a menos que seja criada uma exceção. Entretanto, pode-se colocar LABELS em laços e sair do laço externo com a instrução EXIT.

Os nomes de LABEL seguem as mesmas regras de outros identificadores. Um LABEL é colocado antes de uma instrução, seja na mesma linha ou em uma linha separada. Coloque o LABEL no LOOP colocando-o antes da palavra LOOP dentro dos delimitadores de LABEL (<<LABEL>>). Se for atribuído um LABEL ao LOOP, o nome do LABEL poderá ser opcionalmente incluído após a instrução END LOOP para clareza. Vejamos um exemplo de LOOP aninhado:

```
BEGIN
  FOR i IN 1..3 LOOP
    FOR j IN 1..5 LOOP
        INSERT INTO tabela1
        VALUES ('Inserindo texto numero ' || i || j);
    END LOOP;
  END LOOP;
END;
/
```

Código-fonte 2.20 – Exemplo de LOOP FOR
Fonte: Elaborado pelo autor (2020)

No nosso exemplo, o primeiro laço do bloco PL/SQL define uma variável denominada **i** e outra variável denominada **j**. Na primeira iteração, a variável **i** receberá o valor de 1 (um), o mesmo acontecendo com a variável **j**. O laço mais interno será executado até que o valor de **j** atinja o número 5 (cinco). Ao sair do laço mais interno, o programa volta para o laço mais externo que irá incrementar o contador **i** em 1 (um) e voltará a executar o laço mais externo. O conteúdo da tabela é listado abaixo para facilitar o entendimento. Note que o primeiro número inserido foi 11, porque os valores de **i** e **j** estavam com o valor 1. Na próxima iteração, o valor de **j** foi incrementado para 2 e o valor 12 foi incluído. Esse processo continuou até que o valor superior de **j** foi atingido. O valor de **i** foi incrementado em 1 (passou a ser 2) e o valor de **j** foi reiniciado em 1. Dessa forma o valor 21 foi inserido.

```
COL1
------------------------------
Inserindo texto numero 11
Inserindo texto numero 12
Inserindo texto numero 13
Inserindo texto numero 14
Inserindo texto numero 15
Inserindo texto numero 21
Inserindo texto numero 22
Inserindo texto numero 23
Inserindo texto numero 24
Inserindo texto numero 25
Inserindo texto numero 31
Inserindo texto numero 32
Inserindo texto numero 33
Inserindo texto numero 34
Inserindo texto numero 35
```

Quadro 2.4 – Saída da tabela populada pelo LOOP aninhado
Fonte: Elaborado pelo autor (2020)

CONCLUSÃO

As estruturas condicionais são a base da programação de sistemas computacionais e como vimos neste capítulo, aumentam as possibilidades de como tratar as informações antes mesmo que deixem o banco de dados.

REFERÊNCIAS

DILLON, Sean; BECK, Christopher; KYTE, Thomas; KALLMAN, Joel; ROGERS, Howard. **Beginning Oracle Programming**. Apress, 2013.

FEUERSTEIN, Steven; PRIBYL, Bill. **Oracle Pl/Sql Programming**. O'Reilly Media, 2014.

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

PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. **Banco de dados**. São Paulo, Pearson, 2015.