

FIAP GRADUAÇÃO

# **Tecnologia em Análise e Desenvolvimento de Sistemas**

Mastering Relational and Non-Relational Database

PROF. MILTON

## **Criando Estruturas de Controle**

# Objetivos

Ao concluir esta lição, você será capaz de:

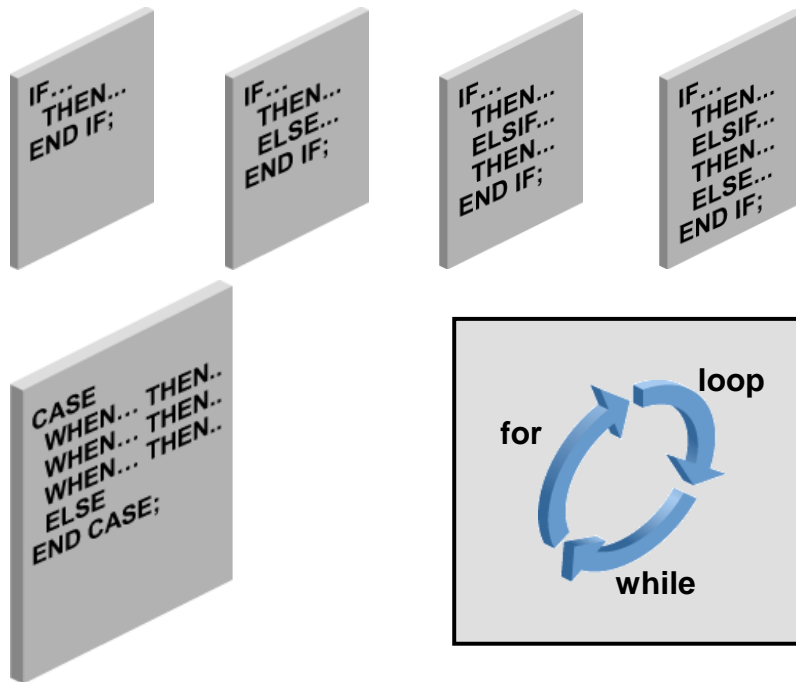
- Identificar os usos e os tipos de estruturas de controle
- Construir uma instrução `IF`
- Usar instruções `CASE` e expressões `CASE`
- Construir e identificar instruções de loop
- Usar diretrizes ao utilizar estruturas de controle condicional

## Objetivos

Você aprendeu a criar blocos PL/SQL contendo as seções declarativa e executável. Aprendeu também a incluir expressões e instruções SQL no bloco executável.

Nesta lição, você aprenderá a usar estruturas de controle, como instruções `IF`, expressões `CASE` e estruturas `LOOP` em um bloco PL/SQL.

# Controlando o Fluxo de Execução



## Controlando o Fluxo de Execução

Você pode alterar o fluxo lógico de instruções dentro do bloco PL/SQL com algumas estruturas de controle. Esta lição aborda quatro tipos de estruturas de controle PL/SQL: estruturas condicionais com a instrução `IF`, expressões `CASE`, estruturas de controle `LOOP` e a instrução `CONTINUE`.

# Agenda

FIAP

- Usando instruções `IF`
- Usando instruções `CASE` e expressões `CASE`
- Construindo e identificando instruções de loop

# Instrução IF

Sintaxe:

```
IF condition THEN
    statements;
[ELSIF condition THEN
    statements;]
[ELSE
    statements;]
END IF;
```

7

## Instrução IF

A estrutura da instrução IF do código PL/SQL é semelhante à das instruções IF de outras linguagens procedurais. Ela permite que o código PL/SQL realize ações seletivamente com base em condições.

Na sintaxe:

<i>condition</i>	É uma variável booleana ou expressão que retorna TRUE, FALSE ou NULL
THEN	Apresenta uma cláusula que associa a expressão booleana à sequência de instruções que vem depois dela
<i>statements</i>	Pode ser uma ou mais instruções PL/SQL ou SQL. (Podem incluir outras instruções IF que contêm várias instruções IF, ELSE e ELSIF aninhadas.) As instruções da cláusula THEN serão executadas somente se a condição da cláusula IF associada for avaliada como TRUE.

## Instrução IF (continuação)

Na sintaxe:

- ELSIF** É uma palavra-chave que apresenta uma expressão booleana (se a primeira condição resultar em FALSE ou NULL, a palavra-chave ELSIF apresentará condições adicionais.)
- ELSE** Apresenta a cláusula default que será executada se e somente se nenhum dos predicados anteriores (apresentados por IF e ELSIF) for igual a TRUE. Os testes são executados em sequência, de modo que um predicado posterior, que poderia ser verdadeiro, seja previamente esvaziado por um predicado anterior que seja verdadeiro.
- END IF** Marca o fim de uma instrução IF

**Observação:** ELSIF e ELSE são opcionais em uma instrução IF. Pode haver qualquer número de palavras-chave ELSIF, mas apenas uma palavra-chave ELSE na instrução IF. END IF marca o fim de uma instrução IF e deve ser encerrada por um ponto-e-vírgula.



# Instrução IF Simples

```
DECLARE
  v_myage number:=31;
BEGIN
  IF v_myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  END IF;
END;
/
```

anonymous block completed

9

## Instrução IF Simples

### Exemplo de Instrução IF Simples

O slide mostra um exemplo de uma instrução IF simples com a cláusula THEN.

- A variável v\_myage é inicializada com 31.
- A condição para a instrução IF retorna FALSE porque v\_myage não é menor que 11.
- Com isso, o controle nunca atinge a cláusula THEN.

### Adicionando Expressões Condicionais

Uma instrução IF pode ter várias expressões condicionais relacionadas com operadores lógicos, como AND, OR e NOT.

Por exemplo:

```
IF (myfirstname='Christopher' AND v_myage <11)
...
```

A condição usa o operador AND e, portanto, será avaliada como TRUE somente se ambas as condições forem avaliadas como TRUE. Não há limitação explícita em relação ao número de expressões condicionais. Entretanto, essas instruções devem estar relacionadas a operadores lógicos adequados.

## Instrução IF THEN ELSE

```
DECLARE
  v_myage number:=31;
BEGIN
  IF v_myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
/
```

```
anonymous block completed
I am not a child
```

10

### Instrução IF THEN ELSE

A cláusula ELSE foi adicionada ao código no slide anterior. A condição não foi alterada e, portanto, ela ainda é avaliada como FALSE. Lembre-se de que as instruções da cláusula THEN serão executadas somente se a condição retornar TRUE. Nesse caso, a condição retorna FALSE e o controle é movido para a instrução ELSE.

A saída do bloco é mostrada abaixo do código.

## Cláusula IF ELSIF ELSE

```
DECLARE
  v_myage number:=31;
BEGIN
  IF v_myage < 11 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSIF v_myage < 20 THEN
    DBMS_OUTPUT.PUT_LINE(' I am young ');
  ELSIF v_myage < 30 THEN
    DBMS_OUTPUT.PUT_LINE(' I am in my twenties');
  ELSIF v_myage < 40 THEN
    DBMS_OUTPUT.PUT_LINE(' I am in my thirties');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am always young ');
  END IF;
END;
/
```

```
anonymous block completed
I am in my thirties
```

11

### Cláusula IF ELSIF ELSE

A cláusula IF pode conter várias cláusulas ELSIF e uma ELSE. O exemplo ilustra as seguintes características dessas cláusulas:

- As cláusulas ELSIF podem ter condições, ao contrário da cláusula ELSE.
- A condição para ELSIF deve ser seguida pela cláusula THEN, que é executada se a condição para ELSIF retornar TRUE.
- Quando houver várias cláusulas ELSIF, se a primeira condição for FALSE ou NULL, o controle passará para a próxima cláusula ELSIF.
- As condições são avaliadas, uma a uma, a partir de cima.
- Se todas as condições forem FALSE ou NULL, as instruções da cláusula ELSE serão executadas.
- A cláusula ELSE final é opcional.

No exemplo, a saída do bloco é mostrada abaixo do código.

## Valor NULL na Instrução IF

```
DECLARE
  v_myage  number;
BEGIN
  IF v_myage < 11 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
/
```

```
anonymous block completed
 I am not a child
```

12

### Valor NULL na Instrução IF

No exemplo mostrado no slide, a variável `v_myage` é declarada, mas não é inicializada. A condição na instrução `IF` retorna `NULL` e não `TRUE` ou `FALSE`. Em casos como esse, o controle passa para a instrução `ELSE`.

#### Diretrizes

- Você pode realizar ações seletivamente com base nas condições que vão sendo atendidas.
- Quando estiver criando o código, lembre-se da ortografia das palavras-chave:
  - `ELSIF` é uma palavra.
  - `END IF` são duas palavras.
- Se a condição booleana de controle for `TRUE`, a sequência associada de instruções será executada; se a condição booleana de controle for `FALSE` ou `NULL`, a sequência associada de instruções será ultrapassada. São permitidas inúmeras cláusulas `ELSIF`.
- Utilize recuos nas instruções executadas condicionalmente para garantir clareza.

# Agenda

FIAP

- Usando instruções `IF`
- Usando instruções `CASE` e expressões `CASE`
- Construindo e identificando instruções de loop

## Expressões CASE

- A expressão CASE seleciona e retorna um resultado.
- Para selecionar o resultado, a expressão CASE usa expressões. O valor retornado por essas expressões é usado para selecionar uma das várias alternativas.

```
CASE selector
  WHEN expression1 THEN result1
  WHEN expression2 THEN result2
  ...
  WHEN expressionN THEN resultN
  [ELSE resultN+1]
END;
```

14

### Expressões CASE

A expressão CASE retorna um resultado baseado em uma ou mais alternativas. Para retornar o resultado, a expressão CASE usa um *seletor*, que é uma expressão cujo valor é usado para retornar uma entre várias alternativas. O seletor é seguido de uma ou mais cláusulas WHEN, que são verificadas sequencialmente. O valor do seletor determina qual resultado será retornado. Caso o valor do seletor seja igual ao valor de uma expressão da cláusula WHEN, essa cláusula WHEN será executada e esse resultado será retornado.

O código PL/SQL também fornece uma expressão CASE pesquisada, que tem esta forma:

```
CASE
  WHEN search_condition1 THEN result1
  WHEN search_condition2 THEN result2
  ...
  WHEN search_conditionN THEN resultN
  [ELSE resultN+1]
END;
```

Uma expressão CASE pesquisada não possui seletor. Além disso, as cláusulas WHEN em expressões CASE contêm condições de pesquisa que retornam um valor booleano em vez de expressões que possam retornar um valor de qualquer tipo.

## Expressões CASE: Exemplo

```
SET VERIFY OFF
DECLARE
    v_grade CHAR(1) := UPPER('&grade');
    v_appraisal VARCHAR2(20);
BEGIN
    v_appraisal := CASE v_grade
        WHEN 'A' THEN 'Excellent'
        WHEN 'B' THEN 'Very Good'
        WHEN 'C' THEN 'Good'
        ELSE 'No such grade'
    END;
    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade || '
                          Appraisal ' || v_appraisal);
END;
/
```

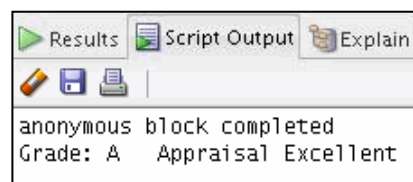
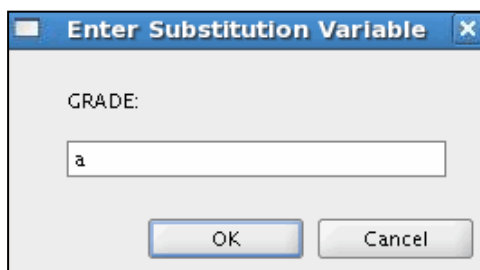
15

### Expressões CASE: Exemplo

No exemplo do slide, a expressão CASE usa o valor da variável `v_grade` como expressão. O valor é fornecido pelo usuário por meio de uma variável de substituição. Baseado no valor informado pelo usuário, a expressão CASE retorna o valor da variável `v_appraisal` com base no valor `v_grade`.

#### Resultado

Quando você informar a ou A para `v_grade`, como mostrado na janela Substitution Variable, a saída do exemplo será a seguinte:



# Expressões CASE Pesquisadas

```
DECLARE
    v_grade CHAR(1) := UPPER('&grade');
    v_appraisal VARCHAR2(20);
BEGIN
    v_appraisal := CASE
        WHEN v_grade = 'A' THEN 'Excellent'
        WHEN v_grade IN ('B','C') THEN 'Good'
        ELSE 'No such grade'
    END;
    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade || '
                          Appraisal ' || v_appraisal);
END;
/
```

16

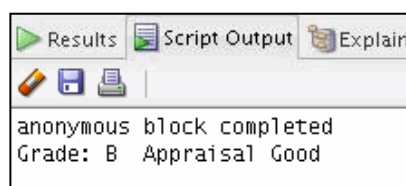
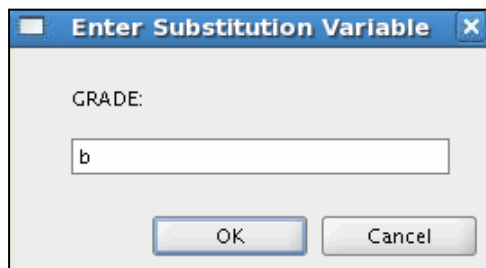
## Expressões CASE Pesquisadas

No exemplo anterior, você viu uma única expressão de teste, a variável `v_grade`. A cláusula `WHEN` compara um valor com essa expressão de teste.

Nas instruções `CASE` pesquisadas, não há uma expressão de teste. Em vez disso, a cláusula `WHEN` contém uma expressão que resulta em um valor booleano. O mesmo exemplo foi reescrito no slide para mostrar instruções `CASE` pesquisadas.

### Resultado

Quando você informar `b` ou `B` para `v_grade`, a saída do exemplo será a seguinte:





# Instrução CASE

```
DECLARE
  v_deptid NUMBER;
  v_deptname VARCHAR2(20);
  v_emps NUMBER;
  v_mngid NUMBER:= 108;
BEGIN
  CASE v_mngid
    WHEN 108 THEN
      SELECT department_id, department_name
        INTO v_deptid, v_deptname FROM departments
        WHERE manager_id=108;
      SELECT count(*) INTO v_emps FROM employees
        WHERE department_id=v_deptid;
    WHEN 200 THEN
      ...
  END CASE;
  DBMS_OUTPUT.PUT_LINE ('You are working in the '|| v_deptname||
    ' department. There are '||v_emps ||' employees in this
    department');
END;
/
```

17

## Instrução CASE

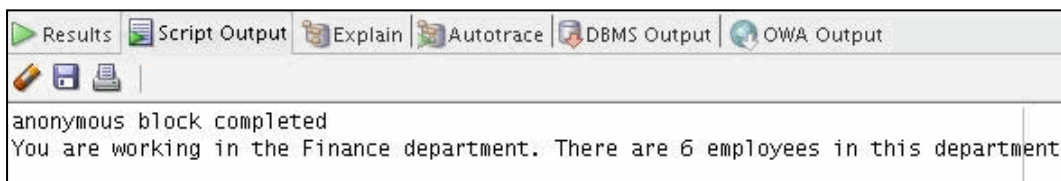
Lembre-se do uso da instrução IF. É possível incluir  $n$  instruções PL/SQL na cláusula THEN e também na cláusula ELSE. Da mesma forma, é possível incluir instruções na instrução CASE, que é mais legível do que várias instruções IF e ELSIF.

### Como uma Expressão CASE Difere de uma Instrução CASE

Uma expressão CASE avalia a condição e retorna um valor, enquanto uma instrução CASE avalia a condição e executa uma ação. A instrução CASE pode ser um bloco PL/SQL inteiro.

- Instruções CASE são finalizadas com END CASE;
- Expressões CASE são finalizadas com END;

A saída do código de exemplo do slide é a seguinte:



**Observação:** Enquanto uma instrução IF não é capaz de executar algo (todas as condições devem ser falsas e a cláusula ELSE não é obrigatória), uma instrução CASE deve executar uma instrução PL/SQL.

## Tratando Valores Nulos

Ao trabalhar com nulos, é possível evitar alguns erros comuns seguindo estas regras:

- Comparações simples envolvendo nulos sempre retornam `NULL`.
- A aplicação do operador lógico `NOT` a um nulo retorna `NULL`.
- Se a condição retornar `NULL` em instruções de controle condicional, a sequência associada de instruções não será executada.

18

### Tratando Valores Nulos

Considere o seguinte exemplo:

```
x := 5;
y := NULL;
...
IF x != y THEN -- yields NULL, not TRUE
  -- sequence_of_statements that are not executed
END IF;
```

É possível esperar que a sequência de instruções seja executada porque `x` e `y` parecem diferentes. Contudo, os valores nulos são indeterminados. Não se sabe se `x` é igual ou não a `y`. Portanto, a condição `IF` retorna `NULL` e a sequência de instruções é ignorada.

```
a := NULL;
b := NULL;
...
IF a = b THEN -- yields NULL, not TRUE
  -- sequence_of_statements that are not executed
END IF;
```

No segundo exemplo, você pode esperar que a sequência de instruções seja executada porque `a` e `b` parecem iguais. No entanto, novamente, a igualdade é desconhecida, portanto a condição `IF` gera `NULL` e a sequência de instruções é ignorada.

## Tabelas Lógicas

Construa uma condição booleana simples com um operador de comparação.

AND	TRUE	FALSE	NULL	OR	TRUE	FALSE	NULL	NOT	
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL

19

### Tabelas Lógicas

É possível construir uma condição booleana simples combinando expressões de número, caractere e data com operadores de comparação.

Você pode construir uma condição booleana complexa combinando condições booleanas simples com os operadores lógicos AND, OR e NOT. Os operadores lógicos são usados para verificar os valores da variável booleana e retornar TRUE, FALSE ou NULL. Nas tabelas lógicas mostradas no slide:

- FALSE tem precedência em uma condição AND e TRUE tem precedência em uma condição OR
- AND retornará TRUE somente se os dois operandos forem TRUE
- OR retornará FALSE somente se os dois operandos forem FALSE
- NULL AND TRUE sempre é avaliado como NULL, pois não se sabe se o segundo operador é avaliado como TRUE

**Observação:** A negação de NULL (NOT NULL) resulta em um valor nulo porque os valores nulos são indeterminados.

## Expressões Booleanas ou Expressão Lógica?

Qual é o valor do `flag` em cada caso?

```
flag := reorder_flag AND available_flag;
```

REORDER_FLAG	AVAILABLE_FLAG	FLAG
TRUE	TRUE	? (1)
TRUE	FALSE	? (2)
NULL	TRUE	? (3)
NULL	FALSE	? (4)

20

### Expressões Booleanas ou Expressão Lógica?

A tabela lógica AND pode ajudá-lo a avaliar as possibilidades para a condição booleana do slide.

#### Respostas

1. TRUE
2. FALSE
3. NULL
4. FALSE

## Agenda

- Usando instruções `IF`
- Usando instruções `CASE` e expressões `CASE`
- Construindo e identificando instruções de loop

## Controle Iterativo: Instruções LOOP

- Os loops repetem uma instrução (ou uma sequência de instruções) várias vezes.
- Há três tipos de loop:
  - Loop básico
  - Loop `FOR`
  - Loop `WHILE`



### Controle Iterativo: Instruções LOOP

O código PL/SQL oferece vários recursos para estruturar loops a fim de repetir uma instrução ou uma sequência de instruções várias vezes. Os loops são usados principalmente para executar instruções repetidamente até que uma condição de saída seja alcançada. É obrigatório que haja uma condição de saída em um loop; caso contrário, o loop será infinito.

Estruturas de loop são o terceiro tipo de estruturas de controle. O código PL/SQL contém os seguintes tipos de loops:

- Loop básico que executa ações repetitivas sem condições gerais
- Loops `FOR` que executam ações iterativas com base em uma contagem
- Loops `WHILE` que executam ações iterativas com base em uma condição

**Observação:** A instrução `EXIT` pode ser usada para encerrar loops. Um loop básico deve ter uma instrução `EXIT`. O loop de cursor `FOR` (que é um outro tipo de loop `FOR`) é abordado na lição “Usando Cursores Explícitos”.

# Loops Básicos

Sintaxe:

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;
```

23

## Loops Básicos

A forma mais simples de uma instrução de LOOP é o loop básico, que contém uma sequência de instruções entre as palavras-chave LOOP e END LOOP. Cada vez que o fluxo de execução atinge a instrução END LOOP, o controle é retornado à instrução LOOP correspondente acima dela. Um loop básico permite a execução de suas instruções pelo menos uma vez, mesmo que a condição EXIT seja atendida na entrada do loop. Sem a instrução EXIT, o loop seria infinito.

### Instrução **EXIT**

Você pode usar a instrução EXIT para encerrar um loop. O controle passará para a próxima instrução depois da instrução END LOOP. É possível executar EXIT como uma ação dentro de uma instrução IF ou como uma instrução stand-alone dentro do loop. A instrução EXIT precisa estar dentro de um loop. No último caso, você pode anexar uma cláusula WHEN para permitir o término condicional do loop. Quando a instrução EXIT for encontrada, a condição da cláusula WHEN será avaliada. Se a condição retornar TRUE, o loop será finalizado e o controle passará para a instrução seguinte ao loop.

Um loop básico pode conter várias instruções EXIT, mas recomenda-se que haja apenas um ponto EXIT.

## Loop Básico: Exemplo

```
DECLARE
  v_countryid  locations.country_id%TYPE := 'CA';
  v_loc_id     locations.location_id%TYPE;
  v_counter    NUMBER(2) := 1;
  v_new_city   locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 3;
  END LOOP;
END;
/
```

24

### Loop Básico: Exemplo

O exemplo de loop básico mostrado no slide é definido da seguinte maneira: “Inserir três novos IDs de locais para o código do país CA e para a cidade Montreal.”

#### Observação

- Um loop básico permite a execução de suas instruções até que a condição `EXIT WHEN` seja atendida.
- Se a condição estiver no loop de uma forma em que ela só será verificada após as instruções de loop serem executadas, o loop será executado pelo menos uma vez.
- No entanto, se a condição de saída for colocada no início do loop (antes de quaisquer outras instruções executáveis) e essa condição for verdadeira, o loop será encerrado e as instruções nunca serão executadas.

#### Resultados

Para exibir a saída, execute o código de exemplo: `code_05_22_s.sql`.



# Loops WHILE

Sintaxe:

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Use o loop `WHILE` para repetir instruções enquanto uma condição for `TRUE`.

25

## Loops WHILE

Você pode usar o loop `WHILE` para repetir uma sequência de instruções até que a condição de controle não seja mais `TRUE`. A condição é avaliada no início de cada iteração. O loop será encerrado quando a condição for `FALSE` ou `NULL`. Se a condição for `FALSE` ou `NULL` no início do loop, nenhuma outra iteração será executada. Portanto, é possível que nenhuma das instruções contidas no loop sejam executadas.

Na sintaxe:

<i>condition</i>	É uma variável booleana ou expressão ( <code>TRUE</code> , <code>FALSE</code> ou <code>NULL</code> )
<i>statement</i>	Pode ser uma ou mais instruções PL/SQL ou SQL

Caso as variáveis envolvidas nas condições não se alterem durante o corpo do loop, a condição permanecerá `TRUE` e o loop não será encerrado.

**Observação:** Se a condição retornar `NULL`, o loop será ignorado e o controle passará para a próxima instrução.

## Loops WHILE: Exemplo

```
DECLARE
  v_countryid  locations.country_id%TYPE := 'CA';
  v_loc_id     locations.location_id%TYPE;
  v_new_city   locations.city%TYPE := 'Montreal';
  v_counter    NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  WHILE v_counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
  END LOOP;
END;
/
```

26

### Loops WHILE: Exemplo

No exemplo do slide, três novos IDs de local para o código de país CA e a cidade de Montreal foram adicionados.

- Em cada iteração pelo loop WHILE, um contador (v\_counter) é incrementado.
- Se o número de iterações for menor que ou igual a 3, o código dentro do loop será executado e uma linha será inserida na tabela locations.
- Depois que v\_counter exceder o número de novos locais para esta cidade e este país, a condição que controla o loop será avaliada como FALSE e o loop será encerrado.

### Resultados

Para exibir a saída, execute o código de exemplo: code\_05\_24\_s.sql.

## Loops FOR

- Use um loop `FOR` para abreviar o teste diminuindo o número de iterações.
- Não declare o contador; ele é declarado implicitamente.

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

27

### Loops FOR

Os loops `FOR` têm a mesma estrutura geral de um loop básico. Além disso, eles têm uma instrução de controle antes da palavra-chave `LOOP` para definir o número de iterações que o código PL/SQL executará.

Na sintaxe:

<i>counter</i>	É um inteiro implicitamente declarado, cujo valor aumenta ou diminui de forma automática (diminui se for usada a palavra-chave <code>REVERSE</code> ) uma unidade a cada iteração do loop até que seja alcançado o limite superior ou inferior
<code>REVERSE</code>	Faz com que o contador seja diminuído a cada iteração, do limite superior para o limite inferior
	<b>Observação:</b> O limite inferior ainda é referenciado primeiro.
<i>lower_bound</i>	Especifica o limite inferior da faixa de valores do contador
<i>upper_bound</i>	Especifica o limite superior da faixa de valores do contador

Não declare o contador. Ele é declarado implicitamente como um inteiro.

## Loops FOR (continuação)

**Observação:** A sequência de instruções será executada cada vez que o contador for incrementado, conforme foi determinado pelos dois limites. O limite inferior e o superior da faixa do loop podem ser literais, variáveis ou expressões, mas devem ser avaliados como inteiros. Os limites são arredondados como inteiros, isto é,  $11/3$  e  $8/5$  são limites superior ou inferior válidos. O limite inferior e o superior estão incluídos na faixa do loop. Se o limite inferior da faixa do loop for avaliado como um inteiro maior do que o limite superior, a sequência de instruções não será executada.

Por exemplo, a seguinte instrução é executada apenas uma vez:

```
FOR i IN 3..3
LOOP
  statement1;
END LOOP;
```

## Loops FOR: Exemplo

```
DECLARE
  v_countryid  locations.country_id%TYPE := 'CA';
  v_loc_id     locations.location_id%TYPE;
  v_new_city   locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id
    FROM   locations
   WHERE  country_id = v_countryid;
  FOR i IN 1..3 LOOP
    INSERT INTO locations(location_id, city, country_id)
      VALUES((v_loc_id + i), v_new_city, v_countryid );
  END LOOP;
END;
/
```

29

### Loops FOR: Exemplo

Você já aprendeu como inserir três novos locais para o código de país CA e a cidade Montreal usando o loop básico e o loop WHILE. O exemplo do slide mostra como alcançar o mesmo usando o loop FOR.

#### Resultados

Para exibir a saída, execute o código de exemplo `code_05_27_s.sql`.

## Regras do Loop FOR

- Faça referência ao contador apenas dentro do loop; fora do loop ele é indefinido.
- Não faça referência ao contador como o destino de uma designação.
- Nenhum limite de loop pode ser `NULL`.

### Regras de Loop FOR

O slide lista as diretrizes a serem seguidas na criação de um loop FOR.

**Observação:** Os limites inferior e superior de uma instrução LOOP não precisam ser literais numéricos. Eles podem ser expressões convertidas em valores numéricos.

#### Exemplo:

```
DECLARE
  v_lower  NUMBER := 1;
  v_upper  NUMBER := 100;
BEGIN
  FOR i IN v_lower..v_upper LOOP
    ...
  END LOOP;
END;
/
```

## Sugestão de Utilização de Loops

- Use o loop básico quando as instruções dentro do loop precisarem ser executadas pelo menos uma vez.
- Use o loop `WHILE` se a condição precisar ser avaliada no início de cada iteração.
- Use um loop `FOR` se o número de iterações for conhecido.

### Sugestão de Utilização de Loops

Um loop básico permite a execução de sua instrução pelo menos uma vez, mesmo que a condição já seja atendida na entrada do loop. Sem a instrução `EXIT`, o loop seria infinito.

Você pode usar o loop `WHILE` para repetir uma sequência de instruções até que a condição de controle não seja mais `TRUE`. A condição é avaliada no início de cada iteração. O loop encerrará quando a condição for `FALSE`. Se a condição for `FALSE` no início do loop, nenhuma outra iteração será executada.

Loops `FOR` possuem uma instrução de controle antes da palavra-chave `LOOP` para determinar o número de iterações que o código `PL/SQL` executará. Use um loop `FOR` se o número de iterações for predeterminado.

## Loops Aninhados e Labels

- É possível aninhar loops em vários níveis.
- Use labels para diferenciar blocos e loops.
- Saia do loop externo com a instrução `EXIT` que faz referência ao label.

32

### Loops Aninhados e Labels

Você pode aninhar loops `FOR`, `WHILE` e básicos um dentro do outro. O encerramento de um loop aninhado não encerra o loop que o contém, a menos que uma exceção seja gerada. Entretanto, você pode utilizar labels nos loops e sair do loop externo com uma 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. Os espaços em branco são insignificantes em todos os parses do código PL/SQL, menos dentro de literais. Use label em loops básicos colocando-o antes da palavra `LOOP` e dentro de delimitadores de label (`<<label>>`). Nos loops `FOR` e `WHILE`, coloque o label antes de `FOR` ou `WHILE`.

Se o loop tiver um label, o nome do label poderá ser incluído (opcionalmente) após a instrução `END LOOP` para garantir clareza.



## Loops Aninhados e Labels: Exemplo

```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
    EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
/
```

33

### Loops Aninhados e Labels: Exemplo

No exemplo do slide, existem dois loops. O loop externo é identificado pelo label <<Outer\_Loop>> e o loop interno é identificado pelo label <<Inner\_Loop>>. Os identificadores foram colocados antes da palavra LOOP dentro dos delimitadores de label (<<label>>). O loop interno está aninhado no loop externo. Os nomes dos labels foram incluídos após as instruções END LOOP para garantir clareza.

.

# Instrução CONTINUE do Código PL/SQL FIAP

- **Definição**
  - Adiciona a funcionalidade para iniciar a próxima iteração de loop
  - Permite que programadores possam transferir o controle para a próxima iteração de um loop
  - Usa estrutura e semântica paralela na instrução `EXIT`
- **Vantagens**
  - Facilita o processo de programação
  - Pode oferecer um pequeno aperfeiçoamento do desempenho em relação às soluções de programação anteriores para simular a instrução `CONTINUE`



34

## Instrução CONTINUE do Código PL/SQL

A instrução `CONTINUE` permite que você transfira o controle de um loop de volta para uma nova iteração ou que você saia do loop. Muitas outras linguagens de programação possuem essa funcionalidade. Com a release do Oracle Database 11g, a linguagem PL/SQL também oferece essa funcionalidade. Antes do Oracle Database 11g, era possível codificar variáveis booleanas e instruções condicionais como soluções alternativas para simular a funcionalidade programática `CONTINUE`. Em alguns casos, as soluções alternativas são menos eficientes.

A instrução `CONTINUE` oferece um meio simplificado de controlar iterações de loop. Essa funcionalidade pode ser mais eficiente do que a codificação de soluções alternativas.

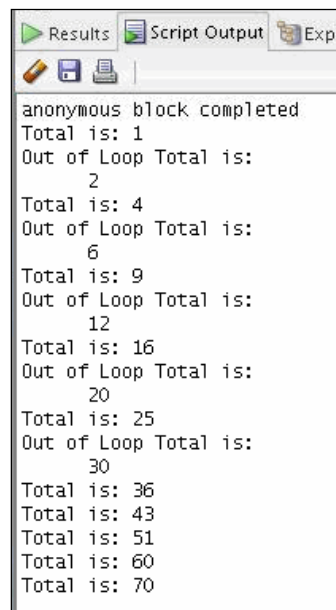
A instrução `CONTINUE` é comumente usada para filtrar os dados dentro do corpo de um loop antes do processamento principal ser iniciado.

## Instrução CONTINUE do Código PL/SQL: Exemplo 1

```

DECLARE
  v_total SIMPLE_INTEGER := 0;
BEGIN
  FOR i IN 1..10 LOOP
    1 v_total := v_total + i;
      dbms_output.put_line
        ('Total is: ' || v_total);
      CONTINUE WHEN i > 5;
    2 v_total := v_total + i;
      dbms_output.put_line
        ('Out of Loop Total is:
          ' || v_total);
      END LOOP;
END;
/

```



```

anonymous block completed
Total is: 1
Out of Loop Total is:
2
Total is: 4
Out of Loop Total is:
6
Total is: 9
Out of Loop Total is:
12
Total is: 16
Out of Loop Total is:
20
Total is: 25
Out of Loop Total is:
30
Total is: 36
Total is: 43
Total is: 51
Total is: 60
Total is: 70

```

35

## Instrução CONTINUE do Código PL/SQL: Exemplo 1

Este é um código exclusivamente 11g. A instrução continue foi introduzida somente no 11g.

No exemplo, há duas designações que usam a variável `v_total`:

1. A primeira designação é executada para cada uma das 10 iterações do loop.
2. A segunda designação é executada para as cinco primeiras iterações do loop. A instrução `CONTINUE` transfere o controle de um loop de volta para uma nova iteração, portanto, para as cinco últimas iterações do loop, a segunda designação `TOTAL` não é executada.


O resultado final da variável `TOTAL` é 70.

## Instrução CONTINUE do Código PL/SQL: Exemplo 2

```

DECLARE
  v_total NUMBER := 0;
BEGIN
  <<BeforeTopLoop>>
  FOR i IN 1..10 LOOP
    v_total := v_total + 1;
    dbms_output.put_line
      ('Total is: ' || v_total);
    FOR j IN 1..10 LOOP
      CONTINUE BeforeTopLoop WHEN i + j > 5;
      v_total := v_total + 1;
    END LOOP;
  END LOOP;
END two_loop;

```



Results Script Output Exp

anonymous block completed  
 Total is: 1  
 Total is: 6  
 Total is: 10  
 Total is: 13  
 Total is: 15  
 Total is: 16  
 Total is: 17  
 Total is: 18  
 Total is: 19  
 Total is: 20

36

### Instrução CONTINUE do Código PL/SQL: Exemplo 2

Você pode usar a instrução `CONTINUE` para ir para a próxima iteração de um loop externo. Este é um código exclusivamente 11g.

Para fazer isso, forneça um label ao loop externo para identificar o local para o qual a instrução `CONTINUE` deve ir.

A instrução `CONTINUE` do loop mais interno encerrará esse loop sempre que a condição `WHEN` for verdadeira (da mesma forma que a palavra-chave `EXIT`). Depois que o loop mais interno for encerrado pela instrução `CONTINUE`, o controle será transferido para a próxima iteração do loop mais externo com o label `BeforeTopLoop` neste exemplo.

Quando esse par de loops for concluído, o valor da variável `TOTAL` será 20.

Você também pode usar a instrução `CONTINUE` de um bloco de código interno, que não contém um loop, desde que o bloco esteja aninhado dentro do loop externo apropriado.

#### Restrições

- A instrução `CONTINUE` não pode, de forma alguma, ser exibida fora de um loop — isso gera um erro do compilador.
- Você não pode usar a instrução `CONTINUE` para ultrapassar o limite de um método, função ou procedure — isso gera um erro do compilador.

## Questionário

Existem três tipos de loops: básicos, `FOR` e `WHILE`.

- a. Verdadeiro
- b. Falso

37

**Resposta: a**

### Tipos de Loop

O código PL/SQL oferece os seguintes tipos de loops:

- Loops básicos que executam ações repetitivas sem condições gerais
- Loops `FOR` que executam ações iterativas com base em uma contagem
- Loops `WHILE` que executam ações iterativas com base em uma condição

Nesta lição, você aprendeu a alterar o fluxo lógico das instruções usando as seguintes estruturas de controle:

- Condicional (instrução `IF`)
- Expressões `CASE` e instruções `CASE`
- Loops:
  - Loop básico
  - Loop `FOR`
  - Loop `WHILE`
- Instrução `EXIT`
- Instrução `CONTINUE`

## Sumário

Uma linguagem só pode ser chamada de linguagem de programação se ela fornecer estruturas de controle para a implementação da lógica de negócios. Essas estruturas de controle também são usadas para controlar o fluxo do programa. O código PL/SQL é uma linguagem de programação que integra estruturas de programação ao código SQL.

Um bloco de controle condicional verifica a validade de uma condição e executa uma ação de acordo com ela. Use o bloco `IF` para realizar uma execução condicional de instruções.

Um bloco de controle iterativo executa uma sequência de instruções repetidamente, enquanto a condição especificada se mostrar `TRUE`. Use os vários blocos de loop para executar as operações iterativas.

## Exercício 5: Visão Geral

Este exercício aborda os seguintes tópicos:

- Executando ações condicionais usando instruções `IF`
- Executando etapas iterativas usando estruturas `LOOP`

### Exercício 5: Visão Geral

Neste exercício, você criará blocos PL/SQL que incorporam loops e estruturas de controle condicionais. Os exercícios testam sua compreensão de criação das várias instruções `IF` e blocos `LOOP`.

