

FIAP GRADUAÇÃO

Mastering Relational and Non-Relational Database

PROF. MILTON

Criando Instruções Executáveis

Objetivos

Ao concluir esta lição, você será capaz de:

- Identificar as unidades lexicais de um bloco PL/SQL
- Usar funções SQL predefinidas no código PL/SQL
- Descrever quando ocorrem conversões implícitas e quando conversões explícitas devem ser usadas
- Criar blocos aninhados e qualificar variáveis com labels
- Criar código legível com recuos adequados
- Usar sequências em expressões PL/SQL

Objetivos

Você aprendeu a declarar variáveis e criar instruções executáveis em um bloco PL/SQL. Nesta lição, você aprenderá como as unidades lexicais formam um bloco PL/SQL. Você aprenderá a criar blocos aninhados. Você também aprenderá sobre o escopo e a visibilidade das variáveis nos blocos aninhados e a qualificar variáveis com labels.

Agenda

- Criando instruções executáveis em um bloco PL/SQL
- Criando blocos aninhados
- Usando operadores e desenvolvendo código legível

Unidades Lexicais de um Bloco PL/SQL FIAP

Unidades lexicais:

- São blocos de construção para qualquer bloco PL/SQL
- São sequências de caracteres, incluindo letras, dígitos, tabulações, espaços, quebras de linha e símbolos
- Podem ser classificadas como:
 - Identificadores: `v_fname`, `c_percent`
 - Delimitadores: `;`, `+`, `-`
 - Literais: `John`, `428`, `True`
 - Comentários: `--`, `/* */`

6

Unidades Lexicais de um Bloco PL/SQL

As unidades lexicais incluem letras, números, caracteres especiais, tabulações, espaços, quebras de linha e símbolos.

- **Identificadores:** Os identificadores são os nomes dados aos objetos PL/SQL. Você aprendeu a reconhecer identificadores válidos e inválidos. Lembre-se de que as palavras-chave não podem ser usadas como identificadores.

Identificadores entre aspas:

- Fazem os identificadores distinguirem entre maiúsculas e minúsculas.
- Incluem caracteres, como espaços.
- Usam palavras reservadas.

Exemplos:

```
"begin date" DATE;  
"end date" DATE;  
"exception thrown" BOOLEAN DEFAULT TRUE;
```

Todos os usos subsequentes dessas variáveis deverão ter aspas duplas. No entanto, o uso de identificadores entre aspas não é recomendado.

- **Delimitadores:** Os delimitadores são símbolos que têm significado especial. Você já aprendeu que o ponto e vírgula (`;`) é usado para encerrar uma instrução SQL ou PL/SQL. Portanto, `;` é um exemplo de delimitador.

Para obter mais informações, consulte *PL/SQL User's Guide and Reference*.

Unidades Lexicais de um Bloco PL/SQL (continuação)

- **Delimitadores (continuação)**

Os delimitadores são símbolos simples ou compostos que têm significado especial em blocos PL/SQL.

Símbolos simples

Símbolo	Significado
+	Operador de adição
-	Operador de subtração/de negativo
*	Operador de multiplicação
/	Operador de divisão
=	Operador de igualdade
@	Indicador de acesso remoto
;	Finalizador de instrução

Símbolos compostos

Símbolo	Significado
<>	Operador de desigualdade
!=	Operador de desigualdade
	Operador de concatenação
--	Indicador de comentário de uma única linha
/*	Delimitador de início de comentário
*/	Delimitador de fim de comentário
:=	Operador de designação

Observação: Esse é apenas um subconjunto, e não uma lista completa dos delimitadores.

- **Literais:** Qualquer valor designado a uma variável é um literal. Qualquer valor de caractere, numérico, booleano ou de data que não seja um identificador é um literal. Os literais são classificados como:
 - **Literais de caractere:** Todos os literais de string têm o tipo de dados CHAR ou VARCHAR2 e são, portanto, chamados de literais de caractere (por exemplo, John e 12C).
 - **Literais numéricos:** Um literal numérico representa um valor inteiro ou real (por exemplo, 428 e 1.276).
 - **Literais booleanos:** Os valores designados a variáveis booleanas são literais booleanos. TRUE, FALSE e NULL são literais booleanos ou palavras-chave.
- **Comentários:** Em programação, convém explicar o que um trecho de código está tentando obter. No entanto, quando você inclui a explicação em um bloco PL/SQL, o compilador não pode interpretar essas instruções. Portanto, deve haver um modo de indicar que essas instruções não precisam ser compiladas. Os comentários são usados principalmente com essa finalidade. Qualquer instrução comentada não será interpretada pelo compilador.
 - Dois hífenes (--) são usados para uma única linha de comentário.
 - Os delimitadores inicial e final do comentário (/* e */) são usados para comentar

várias linhas.

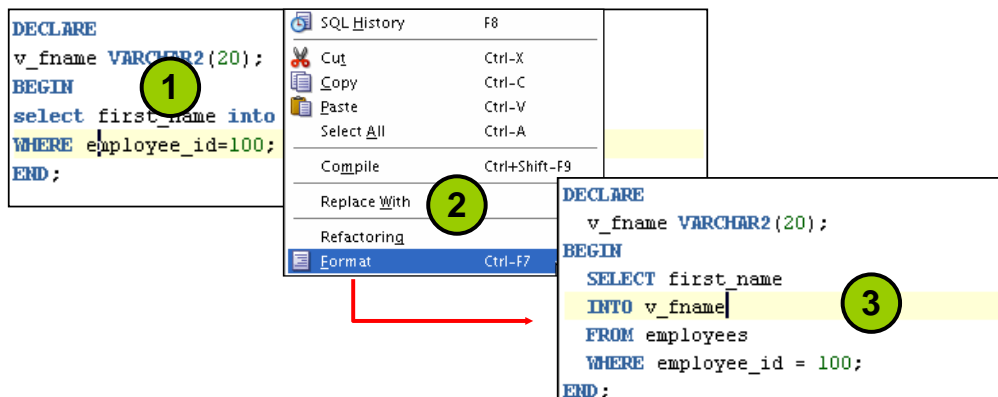
Diretrizes e Sintaxe de Blocos PL/SQL

- Usando Literais

- Os literais de caractere e de data devem ser delimitados por aspas simples.
- Os números podem estar em valores simples ou em notação científica.

```
v_name := 'Henderson';
```

- Formatando código: as instruções podem abranger várias linhas.



8

Diretrizes e Sintaxe de Blocos PL/SQL

Usando Literais

Um literal é um valor explícito numérico, de string de caracteres, de data ou booleano que não é representado por um identificador.

- Os literais de caracteres incluem todos os caracteres imprimíveis no conjunto de caracteres PL/SQL: letras, números, espaços e símbolos especiais.
- Os literais numéricos podem ser representados por um simples valor (por exemplo, -32.5) ou em notação científica (por exemplo, $2E5$ significa $2 * 10^5 = 200.000$).

Formatando Código

Em um bloco PL/SQL, uma instrução SQL pode abranger várias linhas (como mostrado no exemplo 3 do slide).

Você pode formatar uma instrução SQL não formatada (como mostrado no exemplo 1 do slide) usando o menu de atalho SQL Worksheet. Clique com o botão direito do mouse na SQL Worksheet ativa e, no menu de atalho que aparecer, selecione a opção Format (como mostrado no exemplo 2).

Observação: Também é possível usar a combinação de teclas de atalho Ctrl + F7 para formatar o código.

Comentando o Código

- Inicie os comentários de uma única linha com dois hífenos (--).
- Coloque um comentário em bloco entre os símbolos /* e */.

```
DECLARE
...
v_annual_sal NUMBER (9,2);
BEGIN
/* Compute the annual salary based on the
   monthly salary input from the user */
v_annual_sal := monthly_sal * 12;
--The following line displays the annual salary
DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

9

Comentando o Código

Comente o código para documentar cada fase e para ajudar a depuração. No código PL/SQL:

- Normalmente, o comentário de uma única linha é iniciado por dois hífenos (--)
- Você também pode colocar um comentário entre os símbolos /* e */

Observação: No caso de comentários de várias linhas, você pode iniciar cada linha do comentário com dois hífenos ou usar o formato de comentário em bloco.

Os comentários são estritamente informativos e não impõem condições ou comportamento na lógica ou nos dados. Comentários bem colocados são extremamente importantes para a legibilidade do código e a sua manutenção futura.

Funções SQL no Código PL/SQL

- Disponíveis em instruções procedurais:
 - Funções de uma única linha
- Não disponíveis em instruções procedurais:
 - DECODE
 - Funções de grupo

Funções SQL no Código PL/SQL

O código SQL oferece várias funções predefinidas que podem ser usadas em instruções SQL. A maioria delas (como funções de uma única linha de número e caractere, de conversão de tipo de dados e de data e timestamp) é válida em expressões PL/SQL.

Estas funções não estão disponíveis em instruções procedurais:

- DECODE
- Funções de grupo: AVG, MIN, MAX, COUNT, SUM, STDDEV e VARIANCE

As funções de grupo aplicam-se a grupos de linhas de uma tabela e, por isso, estão disponíveis apenas em instruções SQL de um bloco PL/SQL. As funções aqui mencionadas são apenas um subconjunto da lista completa.

Funções SQL no Código PL/SQL: Exemplos

- Obter o tamanho de uma string:

```
v_desc_size INTEGER(5);  
v_prod_description VARCHAR2(70):='You can use this  
product with your radios for higher frequency';  
  
-- get the length of the string in prod_description  
v_desc_size:= LENGTH(v_prod_description);
```

- Obtenha o número de meses que um funcionário trabalhou:

```
v_tenure:= MONTHS_BETWEEN (CURRENT_DATE, v_hiredate);
```

11

Funções SQL no Código PL/SQL: Exemplos

É possível usar funções SQL para manipular dados. Essas funções estão agrupadas nas seguintes categorias:

- Número
- Caractere
- Conversão
- Data
- Diversos

Usando Sequências em Expressões PL/SQL

Iniciando no 11g:

```
DECLARE
    v_new_id NUMBER;
BEGIN
    v_new_id := my_seq.NEXTVAL;
END;
/
```

Antes do 11g:

```
DECLARE
    v_new_id NUMBER;
BEGIN
    SELECT my_seq.NEXTVAL INTO v_new_id FROM Dual;
END;
/
```

12

Acessando Valores de Sequência

No Oracle Database 11g, você pode usar as pseudocolunas `NEXTVAL` e `CURRVAL` em qualquer contexto PL/SQL, no qual uma expressão de tipo de dados `NUMBER` possa ser exibida legalmente. Embora o estilo antigo de utilização de uma instrução `SELECT` para consultar uma sequência ainda seja válido, é recomendável não utilizá-lo.

Antes do lançamento do Oracle Database 11g, era obrigatório que você criasse uma instrução SQL para usar um valor de objeto de sequência em uma subrotina PL/SQL. Normalmente, você cria uma instrução `SELECT` para fazer referência às pseudocolunas `NEXTVAL` e `CURRVAL` a fim de obter um número de sequência. Esse método criava um problema de utilização.

No Oracle Database 11g, a limitação da obrigatoriedade da criação de uma instrução SQL para recuperar um valor de sequência é eliminada. Com o recurso de aprimoramento da sequência:

- A utilização da sequência é aprimorada
- O desenvolvedor digita menos
- O código resultante é mais claro

Conversão de Tipo de Dados

- Converte dados em tipos de dados comparáveis
- É de dois tipos:
 - Conversão implícita
 - Conversão explícita
- Funções:
 - TO_CHAR
 - TO_DATE
 - TO_NUMBER
 - TO_TIMESTAMP

13

Conversão de Tipo de Dados

Em qualquer linguagem de programação, converter um tipo de dados em outro é um requisito importante. O código PL/SQL pode tratar dessas conversões com tipos de dados escalares. As conversões de tipo de dados podem ser de dois tipos:

Conversões implícitas: O código PL/SQL tentará converter os tipos de dados dinamicamente se eles estiverem misturados em uma instrução. Considere o seguinte exemplo:

```
DECLARE
  v_salary NUMBER(6) := 6000;
  v_sal_hike VARCHAR2(5) := '1000';
  v_total_salary v_salary%TYPE;
BEGIN
  v_total_salary := v_salary + v_sal_hike;
END;
/
```

Nesse exemplo, a variável `sal_hike` é do tipo `VARCHAR2`. Ao calcular o salário total, o código PL/SQL primeiro converterá `sal_hike` em `NUMBER` e, em seguida, realizará a operação. O resultado é do tipo `NUMBER`.

As conversões implícitas podem acontecer entre:

- Caracteres e números
- Caracteres e datas

Conversão de Tipo de Dados (continuação)

Conversões explícitas Para converter valores de um tipo de dados para outro, use funções predefinidas. Por exemplo, para converter um valor CHAR em um valor DATE ou NUMBER, use TO_DATE ou TO_NUMBER, respectivamente.

Conversão de Tipo de Dados

1

```
-- implicit data type conversion  
v_date_of_joining DATE:= '02-Feb-2000';
```

2

```
-- error in data type conversion  
v_date_of_joining DATE:= 'February 02,2000';
```

3

```
-- explicit data type conversion  
v_date_of_joining DATE:= TO_DATE('February  
02,2000', 'Month DD, YYYY');
```

15

Conversão de Tipo de Dados (continuação)

Observe os três exemplos de conversões implícitas e explícitas do tipo de dados DATE do slide:

1. Como a literal de string designada à variável `date_of_joining` está no formato default, este exemplo executa a conversão implícita e designa a data especificada à `date_of_joining`.
2. O código PL/SQL retorna um erro porque a data que está sendo designada não está no formato default.
3. A função `TO_DATE` é usada para converter explicitamente a data determinada em um formato específico e designá-la ao tipo de dados DATE com a variável `date_of_joining`.

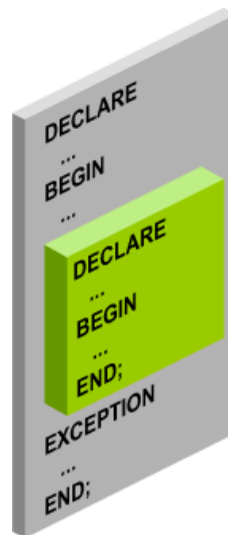
Agenda

- Criando instruções executáveis em um bloco PL/SQL
- Criando blocos aninhados
- Usando operadores e desenvolvendo código legível

Blocos Aninhados

Os blocos PL/SQL podem ser aninhados.

- Uma seção executável (`BEGIN ... END`) pode conter blocos aninhados.
- Uma seção de exceções pode conter blocos aninhados.



17

Blocos Aninhados

Por ser procedural, a linguagem PL/SQL permite aninhar instruções. É possível aninhar blocos onde quer que uma instrução executável seja permitida, transformando, então, o bloco aninhado em uma instrução. Se a seção executável tiver código para muitas funcionalidades logicamente relacionadas para suportar vários requisitos de negócios, você poderá dividir a seção executável em blocos menores. A seção de exceções também pode conter blocos aninhados.

Blocos Aninhados: Exemplo

```
DECLARE
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

```
anonymous block completed
LOCAL VARIABLE
GLOBAL VARIABLE
GLOBAL VARIABLE
```

18

Blocos Aninhados (continuação)

O exemplo mostrado no slide tem um bloco externo (pai) e um bloco aninhado (filho). A variável `v_outer_variable` é declarada no bloco externo e a variável `v_inner_variable` é declarada no bloco interno.

A variável `v_outer_variable` é local no bloco externo, mas é global no bloco interno. Se você acessar essa variável no bloco interno, o código PL/SQL primeiro procurará uma variável local no bloco interno com esse nome. Não há variável com o mesmo nome no bloco interno, por isso o PL/SQL procura a variável no bloco externo. Portanto, `v_outer_variable` é considerada a variável global para todos os blocos que contêm outros. Essa variável pode ser acessada nos blocos internos como é mostrado no slide. As variáveis declaradas em um bloco PL/SQL são consideradas locais para aquele bloco e globais para todos os seus sub-blocos.

A variável `v_inner_variable` é local para o bloco interno e não é global porque o bloco interno não possui blocos aninhados. Essa variável só pode ser acessada dentro do bloco interno. Se o bloco PL/SQL não encontrar a variável declarada localmente, ele procurará acima na seção declarativa dos blocos pais. O bloco PL/SQL não procura de cima para baixo, na direção dos blocos filhos.

Escopo e Visibilidade da Variável

```
DECLARE
  v_father_name VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father's Name: '||v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child's Name: '||v_child_name);
  END;
  DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
/
```

19

Escopo e Visibilidade da Variável

A saída do bloco mostrado no slide é a seguinte:

```
anonymous block completed
Father's Name: Patrick
Date of Birth: 12.12.02
Child's Name: Mike
Date of Birth: 20.04.72
```

Examine a data de nascimento que é impressa para pai e filho. A saída não oferece as informações corretas porque o escopo e a visibilidade das variáveis não estão aplicadas corretamente.

- O *escopo* de uma variável é a parte do programa em que a variável é declarada e está acessível.
- A *visibilidade* de uma variável é a parte do programa em que a variável pode ser acessada sem usar um qualificador.

Escopo

- A variável `v_father_name` e a primeira ocorrência da variável `v_date_of_birth` são declaradas no bloco externo. Essas variáveis têm o escopo do bloco em que foram declaradas. Portanto, o escopo dessas variáveis está limitado ao bloco externo.

Escopo e Visibilidade da Variável (continuação)

Escopo (continuação)

- As variáveis `v_child_name` e `v_date_of_birth` são declaradas no bloco interno ou no bloco aninhado. Essas variáveis podem ser acessadas apenas dentro do bloco aninhado, e não no bloco externo. Quando uma variável estiver fora do escopo, o código PL/SQL liberará a memória usada para armazenar a variável; desse modo, não será possível fazer referência a essas variáveis.

Visibilidade

- A variável `v_date_of_birth` declarada no bloco externo tem escopo mesmo no bloco interno. No entanto, essa variável não é visível no bloco interno porque ele tem uma variável local com o mesmo nome.
 1. Examine o código na seção executável do bloco PL/SQL. Você pode imprimir o nome do pai, o nome do filho e a data de nascimento. Apenas a data de nascimento do filho pode ser exibida aqui porque a data de nascimento do pai não é visível.
 2. A data de nascimento do pai é visível no bloco externo e, portanto, pode ser impressa.

Observação: Não é possível ter variáveis com o mesmo nome em um bloco. No entanto, como mostrado no exemplo, é possível declarar variáveis com nome igual em dois blocos diferentes (blocos aninhados). Os dois itens representados pelos identificadores são diferentes, e alterações em um não afetarão o outro.

Usando um Qualificador com Blocos Aninhados

```
BEGIN <<outer>>
DECLARE
  v_father_name VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father's Name: '||v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '
                        ||outer.v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child's Name: '||v_child_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
  END;
END;
END outer;
```

21

Usando um Qualificador com Blocos Aninhados

Um *qualificador* é um label dado a um bloco. Você pode usar um qualificador para acessar as variáveis que têm escopo, mas não são visíveis.

Exemplo

No código de exemplo:

- O bloco externo recebeu o label `outer`
- Dentro do bloco interno, o qualificador `outer` é usado para acessar a variável `v_date_of_birth` que está declarada no bloco externo. Portanto, a data de nascimento do pai e a do filho podem ser impressas de dentro do bloco interno.
- A saída do código do slide mostra as informações corretas:

```
anonymous block completed
Father's Name: Patrick
Date of Birth: 20-APR-72
Child's Name: Mike
Date of Birth: 12-DEC-02
```

Observação: O uso de labels não se limita ao bloco externo. Você pode pôr label em qualquer bloco.

Desafio: Determinação do Escopo de Variáveis

```
BEGIN <<outer>>
DECLARE
  v_sal      NUMBER(7,2) := 60000;
  v_comm     NUMBER(7,2) := v_sal * 0.20;
  v_message  VARCHAR2(255) := ' eligible for commission';
BEGIN
  DECLARE
    v_sal      NUMBER(7,2) := 50000;
    v_comm     NUMBER(7,2) := 0;
    v_total_comp NUMBER(7,2) := v_sal + v_comm;
  BEGIN
    1 → v_message := 'CLERK not' || v_message;
        outer.v_comm := v_sal * 0.30;
    END;
    2 → v_message := 'SALESMAN' || v_message;
  END;
END outer;
/
```

22

Desafio: Determinação do Escopo de Variáveis

Avalie o bloco PL/SQL no slide. Determine cada um dos seguintes valores, de acordo com as regras de escopo:

1. O valor de `v_message` na posição 1
2. O valor de `v_total_comp` na posição 2
3. O valor de `v_comm` na posição 1
4. O valor de `outer.v_comm` na posição 1
5. O valor de `v_comm` na posição 2
6. O valor de `v_message` na posição 2

Respostas: Determinação do Escopo de Variáveis

Estas são as respostas para as perguntas sobre escopo:

1. O valor de `v_message` na posição 1: **CLERK not eligible for commission**
2. O valor de `v_total_comp` na posição 2: **Error. v_total_comp is not visible here because it is defined within the inner block.**
3. O valor de `v_comm` na posição 1: **0**
4. O valor de `outer.v_comm` na posição 1: **12000**
5. O valor de `v_comm` na posição 2: **15000**
6. O valor de `v_message` na posição 2: **SALESMANCLERK not eligible for commission**

Agenda

- Criando instruções executáveis em um bloco PL/SQL
- Criando blocos aninhados
- Usando operadores e desenvolvendo código legível

Operadores no Código PL/SQL

- Lógicos
- Aritméticos
- Concatenação
- Parênteses para controlar a ordem das operações

Iguais aos do SQL

- Operador exponencial (**)

Operadores no Código PL/SQL

As operações dentro de uma expressão são executadas em uma ordem específica, dependendo da precedência (prioridade) delas. A tabela a seguir mostra a ordem default das operações, da prioridade mais alta à prioridade mais baixa.

Operator	Operation
**	Exponenciação
+, -	Identidade, negação
*, /	Multiplicação, divisão
+, -,	Adição, subtração, concatenação
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Comparação
NOT	Negação lógica
AND	Conjunção
OR	Inclusão

Operadores no Código PL/SQL: Exemplos

- Incrementar o contador para um loop.

```
loop_count := loop_count + 1;
```

- Definir o valor de um flag booleano.

```
good_sal := sal BETWEEN 50000 AND 150000;
```

- Validar se o número de um funcionário contém um valor.

```
valid := (empno IS NOT NULL);
```

Operadores no Código PL/SQL (continuação)

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

- Comparações envolvendo nulos sempre retornam NULL.
- A aplicação do operador lógico NOT a um nulo retorna NULL.
- Em instruções de controle condicional, se a condição retornar NULL, a sequência de instruções associadas não será executada.

Diretrizes de Programação

Facilite a manutenção do código:

- Documentando o código com comentários
- Desenvolvendo uma convenção de letras maiúsculas e minúsculas para o código
- Desenvolvendo convenções de nomes para identificadores e outros objetos
- Melhorando a legibilidade com o uso de recuos

27

Diretrizes de Programação

Siga as diretrizes de programação mostradas no slide para produzir um código claro e reduzir a manutenção ao desenvolver um bloco PL/SQL.

Convenções de Código

A tabela a seguir fornece diretrizes para criar código em caracteres maiúsculos ou minúsculos para ajudar a distinguir palavras-chave de objetos nomeados.

Categoria	Convenção para Maiúsculas/Minúsculas	Exemplos
Instruções SQL	Letras maiúsculas	SELECT, INSERT
Palavras-chave PL/SQL	Letras maiúsculas	DECLARE, BEGIN, IF
Tipos de dados	Letras maiúsculas	VARCHAR2, BOOLEAN
Identificadores e parâmetros	Letras minúsculas	v_sal, emp_cursor, g_sal, p_empno
Tabelas de bancos de dados	Letras minúsculas, plural	employees, departments
Colunas de bancos de dados	Letras minúsculas, singular	employee_id, department_id

Usando Recuos no Código

Para obter clareza, use recuos para cada nível de código.

```
BEGIN
  IF x=0 THEN
    y:=1;
  END IF;
END;
/
```

```
DECLARE
  deptno          NUMBER(4);
  location_id     NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    deptno,
          location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';

  ...
END;
/
```

28

Usando Recuos no Código

Para obter clareza e aumentar a legibilidade, use um recuo para cada nível de código. Para mostrar a estrutura, você pode dividir as linhas usando retorno de carro e criar recuos nas linhas usando espaços e tabulações. Compare a legibilidade das seguintes instruções IF:

```
IF x>y THEN max:=x;ELSE max:=y;END IF;
```

```
IF x > y THEN
  max := x;
ELSE
  max := y;
END IF;
```

Questionário

Você pode usar a maioria das funções SQL de uma única linha, como as de número, caractere, conversão e data em expressões PL/SQL.

- a. Verdadeiro
- b. Falso

29

Resposta: a

Funções SQL no Código PL/SQL

O código SQL oferece várias funções predefinidas que podem ser usadas em instruções SQL. A maioria delas (como funções de uma única linha de número e caractere, de conversão de tipo de dados e de data e timestamp) é válida em expressões PL/SQL.

Estas funções não estão disponíveis em instruções procedurais:

- DECODE
- Funções de grupo: AVG, MIN, MAX, COUNT, SUM, STDDEV e VARIANCE

As funções de grupo aplicam-se a grupos de linhas de uma tabela e, por isso, estão disponíveis apenas em instruções SQL de um bloco PL/SQL. As funções aqui mencionadas são apenas um subconjunto da lista completa.

Nesta lição, você aprendeu a:

- Identificar as unidades lexicais de um bloco PL/SQL
- Usar funções SQL predefinidas no código PL/SQL
- Criar blocos aninhados para fragmentar funcionalidades relacionadas logicamente
- Decidir quando executar conversões explícitas
- Qualificar variáveis em blocos aninhados
- Usar sequências em expressões PL/SQL

Sumário

Como PL/SQL é uma extensão do código SQL, as regras gerais de sintaxe que se aplicam ao SQL também se aplicam à linguagem PL/SQL.

Um bloco pode ter inúmeros blocos aninhados definidos dentro de sua parte executável. Os blocos definidos dentro de um bloco são chamados de sub-blocos. Só é possível aninhar blocos na parte executável de um bloco. Como a seção de exceções também faz parte da seção executável, ela também pode conter blocos aninhados. Assegure escopo e visibilidade corretos para as variáveis quando usar blocos aninhados. Evite usar identificadores iguais nos blocos pais e filhos.

A maioria das funções disponíveis no código SQL também é válida em expressões PL/SQL. As funções de conversão convertem um valor de um tipo de dados para outro. Os operadores de comparação comparam uma expressão com outra. O resultado é sempre TRUE, FALSE ou NULL. Normalmente, os operadores de comparação são usados em instruções de controle condicional e na cláusula WHERE de instruções de manipulação de dados SQL. Os operadores relacionais permitem comparar arbitrariamente expressões complexas.

Exercício 3: Visão Geral

Este exercício aborda os seguintes tópicos:

- Verificando regras de escopo e aninhamento
- Criando e testando blocos PL/SQL

Exercício 3: Visão Geral

Os exercícios 1 e 2 são impressos.

