

# Linguagem Procedural de Banco de Dados

## Aula 2: Conceitos Básicos

### Apresentação

---

Para desenvolver rotinas em LP, é necessário o entendimento das estruturas definidas na linguagem procedural (área de declaração, blocos de comandos), bem como suas normatizações e sintaxe, além de conceitos que já são amplamente utilizados nas linguagens de programação (declaração, atribuição e utilização de expressões). Veremos nesta aula como aplicá-los e praticaremos exemplos.

# Objetivo

---

- Definir estruturas básicas em LP, tais como: blocos, tipos de dados, declarações e expressões;
- Identificar os principais tipos de dados utilizados em LP;
- Aplicar comandos básicos de LP.

## Estruturas básicas em Linguagem Procedural (LP)

---

A programação utilizada em banco de dados, chamada LP, foi pensada para ser utilizada com blocos. Esses são distribuídos em áreas de declarações, de comandos e de exceções.

Os blocos são delimitados por palavras reservadas da LP como: DECLARE, BEGIN, EXCEPTION e END, dentre outras que estudaremos durante o andamento da disciplina.

**A seção DECLARE é opcional e é utilizada para a declaração de variáveis, cursores, tipos, subprogramas e outros elementos usados no programa. O bloco delimitado por BEGIN...END é obrigatório e deve conter pelo menos uma instrução NULL para que seja executado.**

Em caso de erros durante a execução no bloco de comandos, pode-se efetuar o tratamento do erro na seção EXCEPTION que, apesar de ser opcional, é importante para manter a qualidade do código.

Deve-se observar também a indentação (espacejamento de linhas da esquerda para a direita), para melhor organização da estrutura do bloco.

Dentro dos blocos, podemos ter outros blocos que, nesse caso, serão chamados de sub-blocos. E eles podem conter declarações e instruções finalizadas com ponto-e-vírgula.

Os sub-blocos podem ser usados, por exemplo, para indicar o agrupamento lógico ou para limitar o escopo de variáveis a um pequeno grupo de instruções.

Estrutura genérica (podendo variar de acordo com o SGDB específico)

```
-- Declarações
BEGIN
    -- Estruturas executáveis (comandos) e outros blocos LP
    BEGIN
        /* Início da rotina aonde iremos...
        ... */
        .
    EXCEPTION
        -- Tratamento de exceções (pode conter outros blocos)
    END;
END;
```

Outra boa prática em LP é o uso de comentários dentro do bloco de comandos; para isso podemos utilizar o hífen duplo (--) que começa um comentário e se estende até o final da linha. No caso de blocos inteiros de comentários, usa-se (/\*) no início do bloco de comentário, que se estende até a próxima ocorrência de (\*), indicando o fim do comentário.

## Declarações e Tipos de Dados

Para efetuar armazenamento de dados, são criadas áreas na memória que podem ser declaradas como variáveis, constantes, cursores, estruturas e tabelas. Na LP, as declarações devem ser “tipadas”, isto é, devem possuir um tipo de dado.

Para declaração de constantes, devemos usar a palavra reservada CONSTANT que deve aparecer antes do tipo de dado e possuir um valor atribuído. Para atribuir um conteúdo, utilizamos o símbolo (: =) ou a palavra reservada DEFAULT, seguida de um valor, que será o valor padrão adotado em caso de omissão de valor inicial.

No caso das variáveis que serão inicializadas, receberão um valor associado ao seu tipo de dado; caso contrário, receberão o valor NULL.

Além disso, podem ser aplicadas restrições, como NOT NULL, indicando que determinada variável não pode receber valor nulo. Nesse caso, obrigatoriamente, ela deve ser inicializada.

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

Atenção

A restrição NOT NULL deverá ser colocada entre a definição de tipo, tamanho e a inicialização.

Os principais tipos de dados são apresentados na tabela a seguir (tabela genérica, devendo ser consultado o manual de referência do SGDB).

Tipo	Descrição
Tipos de Data e Hora	
Date	Representa datas sem a hora. A representação padrão é a ISO 8601, no formato AAAA-MM-DD. Intervalo de datas possível: de 4713 A.C até 5874897 AD.
Timestamp	Data e hora, independentemente de fuso horário (com ou sem). Intervalo de datas possível: de 4713 A.C até 294276 AD.
Time	Somente a hora, sem a data, no formato hh:mm:ss.
Interval	Unidade de tempo que é utilizada para adicionar ou subtrair tempo de um timestamp.
Tipos Numéricos	
Smallint	Tipo inteiro com 2 bytes, na faixa de -32768 até +32767.
Int/ integer	Tipo inteiro com 4 bytes, na faixa de -2147483648 até +2147483647.
bigint/ int8	Inteiro com 8 bytes com sinal, na faixa de -9223372036854775808 até 9223372036854775807.
smallserial/ serial2	Tipo inteiro sequencial, usado em campos de autoincremento. Intervalo de 1 até 32767.
serial/ serial4	Serial com 4 bytes. Intervalo de 1 até 2147483647.
bigserial/ serial8	Serial com 8 bytes. Intervalo de 1 até 9223372036854775807.
numeric(p,s)	Números decimais, com escala s e precisão p. Até 131072 dígitos antes do ponto decimal e 16383 dígitos após o ponto decimal.
double precision	Números de ponto flutuante, precisão de 15 dígitos decimais.
Tipos de String	
varchar(n) ou character varying(n)	String com no máximo n caracteres, sem espaços no final.
char(n)	String com exatamente n caracteres.
Text	Texto de tamanho ilimitado.
Tipo Monetário	
Money	Representação de moedas, no intervalo de -92233720368547758,08 até +92233720368547758,07.
Outros Tipos	
Array	Vetor de um tipo específico.
Enum	Enumeração (conjunto ordenado e estático de valores).

Boolean	Valores true ou false (verdadeiro ou falso).
Bytea	Array de bytes usado para armazenar objetos binários, como arquivos.

As variáveis e constantes criadas em blocos LP podem herdar o tipo de dado de colunas, de outras variáveis e até da linha inteira de uma tabela. Esse tipo de definição diminui as manutenções oriundas de alterações nas definições das colunas das tabelas.

Para utilizar comando em SQL em conjunto com LP é necessário o uso de alguns tipos com funcionalidades especiais, tais como:

- nome\_da\_tabela%ROWTYPE: Define um tipo especial que representa registros de uma tabela.
- nome\_da\_tabela.campos%TYPE: Define um tipo especial que representa um campo específico de uma tabela.

**Exemplos de Declaração:**

```
Nr CONSTANT integer:= 15;
Nrlidos2 CONSTANT Smallint;
CdCliente Number;
id_usuario integer;
quantidade numeric(10,2) := 30.5;
minha_url varchar(30);
teste boolean := FALSE;
```

As declarações de cursores, estruturas e tabelas serão analisadas em detalhes nas próximas aulas.

Observe que, quando uma variável é definida dentro de um bloco, será considerada uma variável local. Isso significa que não será reconhecida fora dele, ou seja, ela é local para este bloco e global para os sub-blocos.

É importante ressaltar que um único bloco pode incluir variáveis locais e globais, enquanto blocos isolados não poderão referenciar variáveis declaradas em outros blocos.

Se uma variável com o mesmo nome de uma variável global for declarada em um sub-bloco, a declaração local prevalecerá.

Dentro da LP você poderá utilizar caracteres e operadores para auxiliar na codificação.

São eles:

- Caracteres: A a Z (maiúsculos e minúsculos), números de 0 a 9 e os caracteres especiais: ( ) + - \* / < > = ! ; : . ' @ % , "\$ & \_ \{ } [ ] | #.

Em PostgreSQL, uma maneira simples de criar rotinas utilizando LP é criar funções e, dessa forma, testar os módulos desenvolvidos (Essa função ficará armazenada no banco de dados).

Portanto, para escrever uma função, utiliza-se o comando CREATE OR REPLACE FUNCTION.


Sintaxe:

```
CREATE OR REPLACE FUNCTION funcao_teste(integer) RETURNS integer AS $$  
    ....  
$$ LANGUAGE plpgsql;
```

Após ser carregada na memória, a função pode ser executada com o comando SELECT, por exemplo:

```
Select funcao_teste(1);
```

## Utilizando operadores

 Clique no botão acima.

A LP do PostgreSQL, denominada PL/pgSQL, fornece um grande número de funções e operadores para os tipos de dados nativos.

Prevendo a portabilidade, deve-se ter em mente que a maioria das funções e operadores descritos nesta aula, com exceção dos operadores mais triviais de aritmética e de comparação, além de algumas funções indicadas explicitamente, não são especificadas pelo padrão SQL.

Algumas das funcionalidades estendidas estão presentes em outros sistemas gerenciadores de banco de dados SQL e, em muitos casos, essas funcionalidades são compatíveis e consistentes entre as várias implementações.

## Operadores lógicos

Estão disponíveis os operadores lógicos habituais:

- AND
- OR
- NOT

A SQL utiliza a lógica booleana de três valores, onde o valor nulo representa o “desconhecido”. Devem ser observadas as seguintes tabelas:

<i>a</i>	<i>b</i>	<i>a AND b</i>	<i>a OR b</i>
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

<i>a</i>	NOT <i>a</i>
TRUE	FALSE
FALSE	TRUE
NULL	NULL

 Fonte: O autor.

## Operadores de comparação

Estão disponíveis os operadores de comparação habituais, conforme mostrado a seguir:

Operador	Descrição
<	menor
>	maior
<=	menor ou igual
>=	maior ou igual
=	igual
<> ou !=	diferente

 Fonte: O autor.

Atenção:

O operador (!=) é convertido em <> no estágio de análise. Não é possível implementar os operadores (!=) e <> realizando operações diferentes.

Os operadores de comparação estão disponíveis para todos os tipos de dados onde façam sentido. Todos os operadores de comparação são operadores binários, que retornam valores do tipo boolean.

Além dos operadores de comparação, está disponível a construção especial BETWEEN.

a BETWEEN x AND y

equivale a  
a >= x AND a <= y

Analogamente,  
a NOT BETWEEN x AND y

## Operadores matemáticos

São fornecidos operadores matemáticos para muitos tipos de dados em LP. No caso dos tipos sem as convenções matemáticas habituais, podemos usar as permutações possíveis (por exemplo, os tipos de data e hora).

Operador	Descrição	Exemplo	Resultado
+	adição	2 + 3	5
-	subtração	2 - 3	-1
*	multiplicação	2 * 3	6
/	divisão (divisão inteira trunca o resultado)	4 / 2	2
%	módulo (resto)	5 % 4	1
^	exponenciação	2.0 ^ 3.0	8
/	raiz quadrada	/ 25.0	5
/	raiz cúbica	/ 27.0	3
!	fatorial	5 !	120
!!	fatorial (operador de prefixo)	!! 5	120
@	valor absoluto	@ -5.0	5
&	AND bit a bit	91 & 15	11
	OR bit a bit	32   3	35
#	XOR bit a bit	17 # 5	20
~	NOT bit a bit	~1	-2
<<	deslocamento à esquerda bit a bit	1 << 4	16
>>	deslocamento à direita bit a bit	8 >> 2	2

 Fonte: O autor.

Podemos também utilizar funções nativas para obtenção de resultados, como veremos e utilizaremos durante as próximas aulas.

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

# Aplicando exemplos de linguagens procedurais de sistemas gerenciadores de bancos de dados

Exemplificando declaração com blocos PL/pgSQL através de função:



```
CREATE FUNCTION hello_world() RETURNS text AS $$
BEGIN
RETURN 'Hello World';
END
$$ LANGUAGE plpgsql;

SELECT hello_world(); --Executando
```

Exemplificando o uso de variáveis em função no Oracle:

```
01: create or replace FUNCTION mensagem
02: RETURN VARCHAR2 IS
03: txt VARCHAR2(50);
04: BEGIN
05: txt := 'Ola Mundo';
06: RETURN txt;
07: END;

08: Select message from dual; --Executando
```

A tabela [Dual] é uma tabela vazia, utilizada no comando SELECT do Oracle onde não for necessário fazer extração de dados de tabelas, pois a cláusula FROM é obrigatória na SQL do Oracle.

Explicando o código acima:

Linha 01: Início da área de declaração da função mensagem.  
Linha 02: Por se tratar de função, temos que retornar um conteúdo (varchar2).  
Linha 03: Início do bloco de comandos.  
Linha 04: Declarando a variável txt com o tipo Varchar2 com o tamanho de 50 caracteres alfanuméricos.  
Linha 05: Atribuindo o conteúdo 'Olá, Mundo!' à variável txt  
Linha 06: Retornando o conteúdo da função.  
Linha 07: Fim do bloco de comandos.  
Linha 08: Executando a função.

No exemplo a seguir, usamos expressões matemáticas (Ex.: soma):

```
01: CREATE OR REPLACE FUNCTION SOMA(INTEGER, INTEGER)
02: RETURNS INTEGER
03: AS $$
04: DECLARE
05: resultado INTEGER;
06: BEGIN
07: resultado := $1 + $2;
08: RETURN resultado;
09: END;
10: $$ LANGUAGE plpgsql;

11: select soma(10,20);
```

Explicando o código acima:

Linha 01: Declaração da função SOMA(INTEGER, INTEGER).

Linha 02: Declaração do tipo de dados do retorno da função.

Linha 03: Delimitador \$\$ de início do código da função.

Linha 04: Início da seção de declarações locais da função.

Linha 05: Declaração da variável usada no retorno.

Linha 06: Delimitador BEGIN de início de bloco de comandos.

Linha 07: Comando de cálculo aritmético que executa a soma de dois inteiros passados como parâmetros.

Linha 08: Comando RETURN (obrigatório em funções que retornam valores).

Linha 09: Delimitador END de fim de bloco de comandos.

Linha 10: Delimitador \$\$ de fim do código da função seguido do nome da linguagem utilizada.

Linha 11: Chamada da função através do comando SELECT (no caso do PostgreSQL, dispensada a cláusula FROM).

Criando uma função PostgreSQL, combinando comandos SQL (exemplo meramente ilustrativo, não executável):

```
01: CREATE FUNCTION MESCLAR_CAMPOS(t_linha nome_da_tabela)
02: RETURNS text AS $$
03: DECLARE
04:     t2_linha nome_tabela2%ROWTYPE;
05: BEGIN
06:     SELECT * INTO t2_linha FROM nome_tabela2 WHERE ... ;
07:     RETURN t_linha.f1 || t2_linha.f3 || t_linha.f5 || t2_linha.f7;
08: END;
09: $$ LANGUAGE plpgsql;

10: SELECT mesclar_campos(t.*) FROM nome_da_tabela t WHERE ... ;
```

Explicando o código acima:

Linha 01: Declaração de uma função MESCLAR\_CAMPOS que recebe um conteúdo externo (parâmetros) que, neste caso, poderá receber uma linha da tabela (retorna uma linha de resultado de um comando SELECT, por exemplo).

Linha 02: Informamos aqui que o conteúdo retornado será um texto.

Linha 03: Início da declaração de variáveis.

Linha 04: Utilizando a notação nome\_da\_tabela%ROWTYPE como tipo para a variável t2\_linha.

Linha 05: Início do bloco de comandos.

Linha 06: Comando SELEC do conteúdo recebido como parâmetros para a tabela nome\_tabela2 que poderá ser condicionada.

Linha 07: Retornando os possíveis campos associados ao select.

Linha 08: Fim do bloco de comandos.

Linha 09: Representação do padrão PostgreSQL.

Linha 10: Teste da função.

Outro exemplo, combinando agora com campos SQL (exemplo meramente ilustrativo, não executável):

```
01: Declare -- Usando tipo a partir de uma tabela
02: Reg_Item    Item_Nota_Fiscal%Rowtype;
03: Aux CdProduto Constant Produto.Cd_Produto%Type := 1 ;
04: Vl_Total Produto.Vl_Custo_Medio%Type;

05: Begin
06: Reg_ltem.cd_produto := 2;
07: SELECT Sum (vl_unitario * qt_produto)
08: INTO Vl_Total
09: FROM ItemNotaFiscal
10: WHERE Cd_Produto = Reg_Item.cd_produto
11: GROUP BY Cd_Produto;

12: Dbms_Output.Put_Line (To_Char(Reg_Item.Cd_produto)
13: ||' - '||
14:    To_Char(Vl_Total, '$9,999 ,990.99' )) ;
15: End;
```

Explicando o código acima:

Linha 01; Início da declaração de variáveis.

Linha 02: Utilizando a notação nome\_da\_tabela%ROWTYPE como tipo para o objeto Reg\_Item.

Linha 03: Declarando uma constante, Aux\_cd para o campo da tabela Produto, utilizando a notação %type Produto.Cd\_Produto%Type.

Linha 04: Declarando uma variável V1\_Total para o campo da tabela Produto.

Linha 05: Início do bloco de comandos.

Linhas 06 a 11: Atribuindo o valor 2 ao campo CD\_Produto do objeto Reg\_Item. Comando SELECT a soma do total dos itens da nota e atribuindo a variável vl\_total, onde o cd\_produto seja = 2.

Linhas 12 a 14: Exibindo uma mensagem, convertendo seu conteúdo para o tipo Char (Item - Valor), estando o valor formatado para moeda.

Linha 15: Fim do bloco de comandos.

Ao longo da disciplina, veremos mais tipos com funcionalidades para manipulação de rotinas LP com SQL.

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

## Atividade

1. Sendo a LP pensada para ser utilizada com o bloco de comando, podemos afirmar que:

- a) Existe a área de declaração e a área de exceções.
- b) A área de declaração é obrigatória e as demais opcionais.
- c) Não se utiliza a área de comandos.
- d) A área de exceções é obrigatória.
- e) O bloco não precisa possuir nenhuma área em sua estrutura.

2. Selecione qual opção apresenta símbolos usados em comentários no bloco de comandos em LP:

- a) \$\$ , \_!;
- b) --, /\*, \*/ ;
- c) -,\*\*\*
- d) ?>, <?
- e) %%, //

3. Quanto à utilização de tipos na declaração em LP, podemos afirmar que:

- a) Apenas CONSTANTES e VARIÁVEIS devem ser associadas a tipos.
- b) Tipos numéricos se limitam a valores inteiros e a declaração de varáveis.
- c) Não se aplica a funções.
- d) Usamos tipos em variáveis, constantes, cursores, estruturas e tabelas.
- e) Em estruturas, tais como: procedimentos armazenados, funções e gatilhos não precisam atribuir tipos às variáveis.

## 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>

PostgreSQL: 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

---

- Estruturas condicionais simples e compostas;
- Estruturas de Repetição com teste no início;
- Estruturas de Repetição com teste no fim.

## Explore mais

---

- Guia do usuário do Oracle Solaris 11.1 Desktop:  
[https://docs.oracle.com/cd/E37936\\_01/html/E36782/glcrb.html](https://docs.oracle.com/cd/E37936_01/html/E36782/glcrb.html)
- Modelagem de Dados com o Oracle SQL Developer  
<https://www.oracle.com/br/database/technologies/appdev/datamodeler.html>
- Comparações entre o PostgreSQL, o Oracle, o SQL Server e o DB2 do SQL Server  
([http://msdn.microsoft.com/library/en-us/tsqlref/ts\\_fa-fz\\_7oqb.asp](http://msdn.microsoft.com/library/en-us/tsqlref/ts_fa-fz_7oqb.asp)), do DB2 e do PostgreSQL  
([ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en\\_US/db2s1e81.pdf](ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/db2s1e81.pdf))