

# Linguagem Procedural de Banco de Dados

## Aula 1: Introdução à linguagem procedural de banco de dados

### Apresentação

---

Um diferencial significativo de um SGBD (Sistema de Gerenciamento de Banco de Dados) é ter a capacidade de desenvolver rotinas e estruturas armazenadas que podem ser utilizadas pelo próprio banco de dados. A linguagem PL (Linguagem Procedural) é utilizada para tarefas complexas, fornecendo ao administrador do banco de dados o máximo de recursos. Os principais SGBDs relacionais do mercado possuem extensões procedurais à SQL, como PL-SQL no Oracle, PL-pgSQL no PostgreSQL, Transact-SQL no SQL Server, SQL PL no DB2, entre outros.

# Objetivo

---

- Definir Linguagem Procedural (PL), reconhecendo suas vantagens em bancos de dados;
- Aplicar exemplos de linguagens procedurais de sistemas gerenciadores de bancos de dados;
- Descrever formas de uso de uma linguagem de procedural: procedimentos armazenados, funções e gatilhos.

## Definição da Linguagem Procedural (PL)

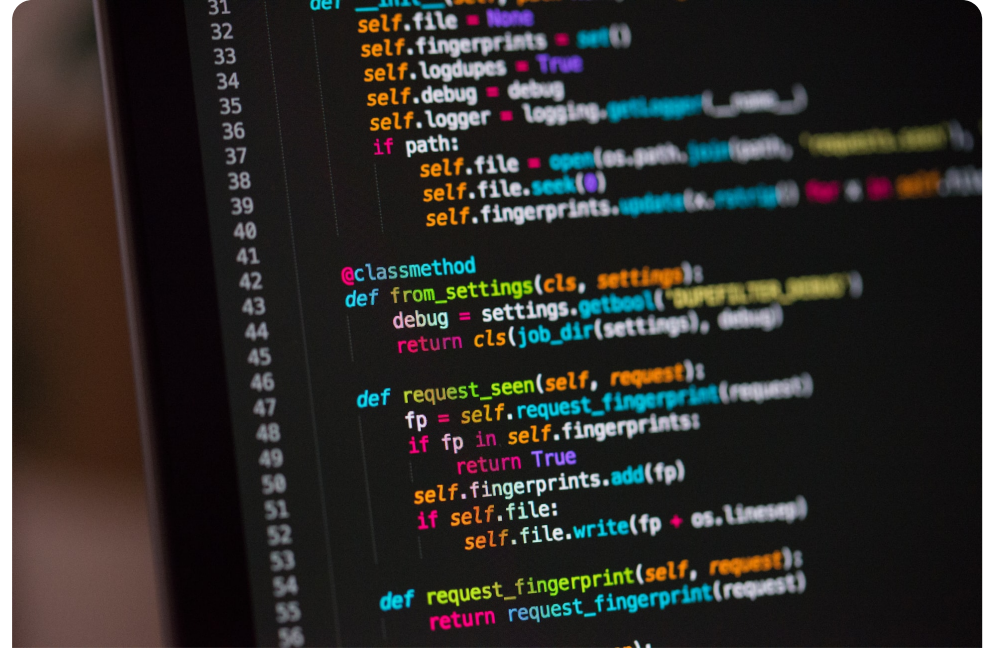
---

A programação aplicada a banco de dados chamada PL (Procedural Language) é uma linguagem procedural que tem como base a SQL (Structured Query Language). Ela é escrita em blocos de códigos e executada diretamente no banco de dados, permitindo assim o desenvolvimento de rotinas complexas e específicas para o SGDB.

Sendo uma linguagem de programação, a PL segue o padrão de linguagens procedurais e é utilizada para criação de procedimentos, funções, gatilhos (usando estruturas de controle e laços, variáveis, vetores, tratamento de erros etc.), podendo ser combinada em sua execução com os comandos SQL. Devido à sua fundamental aplicação, essa linguagem é mantida sempre atualizada desde as primeiras versões, com constante suporte e evolução em grande parte dos bancos de dados relacionais.



Com a PL, o desenvolvedor tem acesso a um conjunto de comandos procedurais (instruções, estruturas de decisão e repetição, atribuições etc.), estruturados em blocos que complementam e estendem os recursos do SQL.



Fonte: Chris Ried / Unsplash

Conforme citado anteriormente, em PL, podemos realizar operações, tais como:

- 1 Manipular variáveis e constantes, herdando o tipo de dados e o tamanho de outras variáveis e constantes ou de objetos do banco de dados (colunas, tabelas etc.).
- 2 Criar cursores para tratar o resultado de uma consulta (query) que retorna 0 ou mais linhas.
- 3 Criar registros para guardar o resultado de um cursor ou campo de tabelas.
- 4 Manipular tipos de dados e os tamanhos das colunas.
- 5 Gerenciar tratamento de erros.
- 6 Criar *labels* para controlar o fluxo de execução e utilizar comando de repetição e comparação.

Em sistemas de médio e grande porte, quanto menor a dependência da camada de aplicação, melhor a flexibilização e manutenção dos sistemas, independentemente da plataforma utilizada. Nesse sentido, deixar as funcionalidades do banco de dados no próprio servidor de banco de dados separa as responsabilidades entre a camada de aplicação e a camada de dados, além de permitir maior segurança.

Para melhor entendimento, é importante destacar a diferença entre SQL e PL, conforme quadro a seguir:

Sigla	Objetivo
SQL	A Structured Query Language é uma linguagem estruturada de acesso aos bancos de dados, declarativa, que usa comandos DDL (Data Definition Language), como CREATE, ALTER e DROP; e DML (Data Manipulation Language) como SELECT, INSERT, UPDATE e DELETE. É executada no servidor por meio de uma interface integrada, aplicada a todos os bancos de dados relacionais.
PL	Caracteriza-se por uma linguagem procedural, não declarativa, ou seja, não bastam apenas comandos SQL, é necessário também o uso de codificação em blocos, chamados blocos PL. Os códigos podem ser executados tanto no cliente quanto diretamente no servidor do banco de dados.

Para melhor entendimento, imagine a seguinte situação:

Você deverá efetuar uma carga de atualização de dados em uma tabela, considerando as seguintes restrições:

1. Os dados originários serão selecionados a partir de quatro tabelas, utilizando associação de chaves sendo que parte dos registros deve atender a certas restrições.  
  
Até aqui podemos fazer um *select*, implementado com *join*, por exemplo.
2. De acordo com o resultado dos dados, certas operações matemáticas deverão ser executadas.

Neste momento, a utilização de *select* se torna algo mais complexo de implementar.

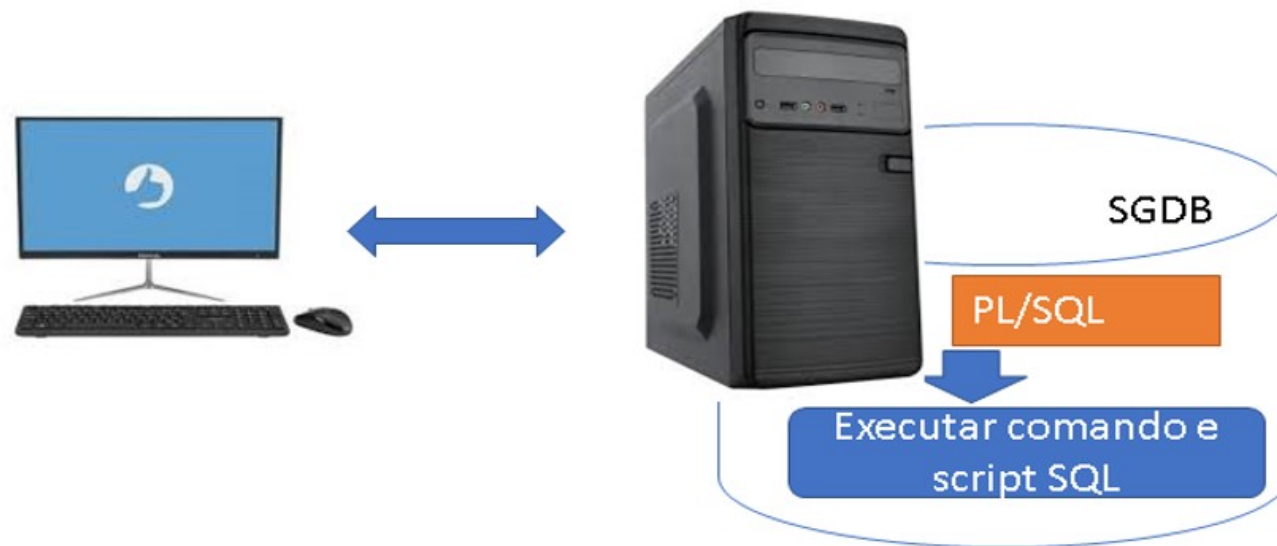
3. Por fim, os resultados calculados e exibidos deverão estar armazenados em uma nova tabela, e serão atualizados, automaticamente, após ações de INSERT e UPDATE.
- A partir deste momento, só poderemos utilizar *Trigger* que é uma estrutura de linguagem procedural.

Tal rotina poderá ser facilmente implementada, utilizando linguagem procedural.

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


## Vantagens no uso de uma linguagem procedural em banco de dados

Com o uso da PL, o administrador do banco de dados obtém independência da aplicação para rotinas, podendo ser executadas diretamente no SGBD, diminuindo assim a troca de dados e sobrecarga na rede, processos na aplicação e dependência da estrutura do cliente.



 Fonte: Autor.

Dentre as vantagens no uso de uma linguagem procedural em banco de dados, podemos destacar:

 Clique nos botões para ver as informações.

### Eficiência



**Eficiência:** por ser armazenada em um servidor de banco de dados, proporciona rapidez na execução de suas estruturas, deixando para o SGDB as funcionalidades específicas para banco de dados;

### Reutilização de rotinas



**Reutilização de rotinas:** sempre que necessário, poderá ser acionada em diversas situações, principalmente nas de manutenção e carga de dados;

### Segurança de dados



**Segurança de dados:** exclusivamente para usuários cadastrados no SGBD, observando seus privilégios e controle de acesso, ou por meio de chamadas executadas pelas aplicações do cliente;

### Portabilidade



**Portabilidade:** a maioria dos SGBDs faz uso de PL, sendo amplamente utilizados pelos principais bancos de dados relacionais do mercado, embora com sintaxes diferentes, por não existir um padrão;

### Integração com o gerenciador de banco de dados



**Integração com o gerenciador de banco de dados:** aplicada em objetos do banco de dados e integrada aos comandos SQL;

### Capacidade procedural



**Capacidade procedural:** estruturas de repetição, controle de fluxo e tratamento de erros, ampliando dessa forma as possibilidades de recursos implementados no banco de dados;



**Suporte a argumentos e tipos de dados de resultados:** via uso de funções e/ou rotinas, é possível utilizar parâmetros de entrada e retorno de dados;

Produtividade



**Produtividade:** Em formato de blocos de comandos podemos citar:

- stored procedures (procedimentos armazenados) –acionados via comando com a possibilidade de envio de parâmetros;
- functions (funções) - criadas pelo usuário com a possibilidade de recebimento de parâmetros e retorno de dados;
- triggers (Gatilhos) - acionados automaticamente pelo SGDBs quando houver manipulação de dados em uma ou mais tabelas.

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

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

Uma das características da PL é possuir estruturas em blocos, onde poderá ser dividida em área de declaração e área de comando e execução.

Dentre os blocos, podemos ter outros blocos que, neste caso, serão chamados de sub-blocos. Para melhor organização, devemos respeitar o padrão de indentação (espacejamento de linhas da esquerda para a direita); tal bloco é iniciado com BEGIN e finalizado com END.

Estrutura

```
DECLARE
    declarações
BEGIN
    estruturas executáveis (comandos) e outros blocos PL
    BEGIN
        .
        .
        .
    EXCEPTION
        tratamento de exceções (pode conter outros blocos)
    END;
END;
```

Ao longo desta disciplina, utilizaremos estruturas definidas em blocos de construção de programas.

Podemos exemplificar desde o simples “Olá, Mundo!” até rotinas mais complexas.

```
BEGIN
    DBMS_OUTPUT.put_line ('Olá Mundo!');
END; --Oracle
```

Utiliza-se (--) para indicar comentário de linha no banco de dados Oracle.

Exemplificando com o uso de variáveis Oracle:

```
DECLARE
    l_message
    VARCHAR2 (100): = 'Olá, Mundo!';
BEGIN
    DBMS_OUTPUT.put_line (l_message);
END; --Oracle
```

Exemplo de criação de função em PostgreSQL:

```
CREATE FUNCTION mov_taxi(subtotal real) RETURNS real AS $$
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

No exemplo a seguir combinamos comandos SQL (SELECT) em uma função PL em PostgreSQL:

```
CREATE FUNCTION locacao(p_itemno int)
RETURNS TABLE(quantidade int, total numeric) AS $$
BEGIN
    RETURN QUERY SELECT s. quantidade , s.quantidade * s.preco FROM venda AS s
        WHERE s.itemno = p_itemno;
END;
$$ LANGUAGE plpgsql;
```

Exemplo no Oracle utilizando comando SQL (UPDATE):



```

DECLARE
    l_dept_id
empregados.departamento_id%TYPE := 10;
BEGIN
    UPDATE empregados
        SET salario = salario * 1.2
        WHERE departamento_id = l_dept_id;
    DBMS_OUTPUT.put_line (SQL%ROWCOUNT);
END;

```

No exemplo anterior, foi criada uma rotina para efetuar atualização no campo salário de 20% da tabela empregada.

## Formas de uso de uma linguagem procedural: procedimentos armazenados, funções e gatilhos

Dentre os principais objetos do banco de dados que utiliza PL, podemos destacar: procedimentos armazenados.

Os procedimentos armazenados (stored procedures) são subprogramas destinados a executar uma tarefa específica, e se caracterizam por não retornarem valor, portanto, não são atribuídos a objetos com espera de dados. Eles podem receber argumentos (nomes de parâmetros que serão enviados como dados externos da rotina), ampliando assim seu poder de atuação. É apresentada a seguir a estrutura de uma procedure em Oracle:

```

CREATE OR REPLACE PROCEDURE Nome_da_Procedure
(Argumento! modo Tipo_de_Dados,
Argumento2 modo Tipo_de_Dados,
Argumenton modo Tipo_de_Dados)

IS ou AS
Variáveis locais, Constantes , ...

BEGIN

Bloco PL

END Nome_da_Procedure;

```

Implementado em PostgreSQL - Estrutura:

```

CREATE [ OR REPLACE ] PROCEDURE
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
{ LANGUAGE lang_name
  | TRANSFORM { FOR TYPE type_name } [, ... ]
  | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
  | SET configuration_parameter { TO value | = value | FROM CURRENT }
  | AS 'definition'

```



```
| AS 'obj_file', 'link_symbol'  
} ...
```

Exemplo:

```
CREATE PROCEDURE insert_data(a integer, b integer)  
LANGUAGE SQL  
AS $$  
INSERT INTO tbl VALUES (a);  
INSERT INTO tbl VALUES (b);  
$$;  
  
CALL insert_data(1, 2);
```

No exemplo anterior, estamos recebendo parâmetros e aplicando inserts na tabela tbl com os valores de entrada. Os comandos CREATE PROCEDURE, assim como o CALL, são novos no PostgreSQL. Foram implementados a partir da versão 11. Em versões anteriores, as *procedures* no PostgreSQL eram implementadas como *functions*.

Formato de função que não retorna valor combinado com comandos SQL:

```
CREATE OR REPLACE FUNCTION InsereFuncionario(codigo INTEGER,  
nome VARCHAR(100), email VARCHAR(150) , telefone VARCHAR(15) ,  
cidade VARCHAR(50) , estado VARCHAR(2))  
RETURNS void AS $  
BEGIN  
    INSERT INTO "Empresa".tb_funcionarios  
    VALUES (codigo, nome, email, telephone, cidade, estado);  
END;  
$ LANGUAGE 'plpgsql';
```

No exemplo anterior, estamos recebendo diversos parâmetros e aplicando um insert na tabela Empresa.

As funções (functions), assim como os procedimentos, permitem a criação de subprogramas. A principal diferença entre eles é que, neste caso, se espera da rotina um retorno de parâmetros.

A estrutura de uma função em Oracle é apresentada a seguir:

```
CREATE OR REPLACE FUNCTION Nome_Funcao  
(Argumento! IN Tipo de Dados,  
Argumento2 IN Tipo de Dados,  
Argumenton IN Tipo=de=Dados)
```

```

RETURN Tipo_de_Dado
IS ou AS
    declarações
BEGIN
    bloco PL
END nome_da_função;

```

Exemplo de uma função Oracle:

```

Create Or Replace Function prinome
    (ds_nome in cliente.nm_cliente%TYPE)
Return varchar2
Is
    Posicao number(2) := 0;
    Nome      varchar2(65);
Begin
    Posicao := Instr(ltrim(ds_nome), ' ');
    If Posicao = 0 then
        Nome := ds_nome;
    End If;
    Return Nome;
End prinome;

```

No exemplo anterior, estamos utilizando declaração de variáveis, atribuição de dados, comparação de dados e retornando o nome.

Exemplo de uma função PostgreSQL:

```

CREATE FUNCTION sales_tax(real) RETURNS real AS $$
DECLARE
    subtotal ALIAS FOR $1;
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;

```

No exemplo anterior, estamos utilizando declaração de variáveis, atribuição de dados e expressões matemáticas no retorno de um valor.

Já a funcionalidade dos Gatilhos (*Triggers*) é efetuar ações de forma automática, sempre que ocorrer um evento de manipulação de dados (INSERT, UPDATE ou DELETE) em uma determinada tabela.

As aplicabilidades de *triggers* são diversas, tais como:

- Manutenção de tabelas de auditorias (Ex.: Logs de Eventos);
- Implementação de níveis de segurança;
- Geração de valores de dados de forma automática. Por exemplo, atualização do saldo em estoque na tabela Produto a partir de uma venda registrada;
- Comparação de consistência de dados – posterior e anterior, após um UPDATE.

Estrutura de um *trigger* em Oracle:

```
CREATE OR REPLACE TRIGGER nome_da_trigger
{BEFORE / AFTER}
DELETE OR INSERT OR UPDATE OF (nome_coluna1. nome_coluna2 •... )
ON nome da tabela
FOR EACH ROW REFERENCING, OLD AS nome NEW AS nome
WHEN condição
DECLARE
    Variáveis. constantes. etc.
BEGIN
END {não colocar o nome_da_trigger};
```

Exemplo:

```
Create Or Replace Trigger Verifica_Produto Before Update
Of vl_custo_medio
On Produto
For Each Row
Begin
Insert into Tmp_Preco Prod
Values
(:old.cd_produto, :old.vl_custo_medio, :new.vl_custo_medio);
End;
```

No exemplo anterior, utilizamos o *trigger* Verifica\_Produto antes de aplicar uma atualização no campo VlcustoMedio da tabela Produto, efetuando uma inserção na tabela tem\_preco com o valor anterior e o valor posterior.

Estrutura em PostgreSQL:

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
ON table_name
[ FROM referenced_table_name ]
[ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
[ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
[ FOR [ EACH ] { ROW | STATEMENT } ]
```

```
[ WHEN ( condition ) ]  
EXECUTE { FUNCTION | PROCEDURE } function_name ( arguments )
```

Exemplo:

```
CREATE TRIGGER tbefore BEFORE INSERT OR UPDATE OR DELETE ON ttest  
FOR EACH ROW EXECUTE FUNCTION emp_stamp ();
```

Os temas aqui apresentados serão estudados detalhadamente durante as próximas aulas do curso.

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

## Atividade

---

1. São consideradas vantagens da PL:

- a) Capacidade de criar tabelas com grande volume de dados.
  - b) Apagar objetos sem permissão de usuários.
  - c) Eficiência, Segurança, Portabilidade, Reutilização de rotinas e Integração com SQL.
  - d) Criar visões exclusivas no banco de dados.
  - e) Criar relatórios estatísticos.
- 

2. Selecione qual opção representa uma estrutura de um bloco de comandos em PL:

- a) Begin, Declare, If, End If, End.
  - b) Alter Table, Drop View, Create User.
  - c) Select for Update of [colunas].
  - d) Create or replace procedure.
  - e) Create sequence seq\_cliente.
- 

3. São objetos do banco de dados que utiliza PL:

- a) Exclusivamente em criação de tabelas com chaves primárias e estrangeiras.
  - b) Apenas em rotinas para criação de usuários.
  - c) Não se aplica a funções.
  - d) Aplica-se a funções, mas não a procedimentos armazenados.
  - e) Em estruturas tais como: procedimentos armazenados, funções e gatilhos.
-

## 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

---

- Tipos de dados e Declarações;
- Expressões;
- Estruturas básicas.

## Explore mais

---

- Documentação interativa acerca de PL DEVELOPER:  
<https://refactoring.guru/pt-br/design-patterns>
- Tutorial PL Oracle:  
<https://www.oracletutorial.com/plsql-tutorial/>
- Tutorial PL PostgreSQL:  
<https://www.postgresqltutorial.com/>