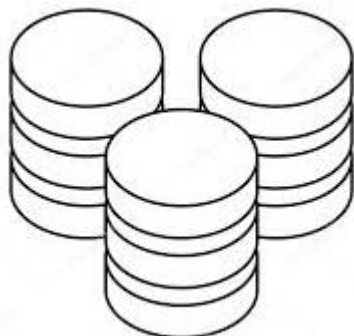


# Banco de Dados II

A Linguagem PL/pgSQL

Introdução

Blocos Anônimos




Profa. Damires Souza

[damires@ifpb.edu.br](mailto:damires@ifpb.edu.br)

# Linguagem SQL

---

- Data Definition Language (DDL)
  - CREATE, ALTER, DROP, RENAME, TRUNCATE.
- Data Manipulation Language (DML)
  - SELECT, INSERT, UPDATE, DELETE
- Transaction Control Language (TCL)
  - COMMIT, ROLLBACK, SAVEPOINT
- Data Control Language (DCL)
  - GRANT, REVOKE



**SQL é uma linguagem  
declarativa**

# E se precisar rodar e armazenar uma função no SGBD???

---

```
CREATE FUNCTION soma_dois_numeros(INTEGER, INTEGER) RETURNS INTEGER AS '  
    SELECT $1 + $2;  
' LANGUAGE SQL;  
  
SELECT soma_dois_numeros(3, 17);
```

- Função/procedure armazenada (*stored procedure*)
  - Subprograma que pode ser criado para efetuar tarefas específicas em tabelas do banco de dados, usando **comandos da linguagem SQL e lógica de programação.**

# Exemplo

---

```
CREATE OR REPLACE FUNCTION incrementa(i integer)
RETURNS integer AS $$
    BEGIN
        RETURN i + 1;
    END;
$$ LANGUAGE plpgsql;

select incrementa(2);
```

**\$\$: usado para delimitar uma string ao invés de utilizar aspas**

**\$\$ => delimitador de uma função ou bloco**

# Programação no Servidor: para quê??

---

- Implementação de **objetos auxiliares**: procedures, functions, triggers
  - Podem ser usados também para **testes**, independentemente da aplicação
- Segurança
  - As **stored procedures/functions e triggers** podem fazer **críticas sofisticadas**
- Evitar redundância no código
  - Se muitos programas compartilham um BD e houver **funcionalidade comum** a eles → **reuso**

# Programação no Servidor

---

- **Objetos auxiliares**

- Procedures, functions, triggers

## Bloco Anônimo:

=> código **sem nome**, executado de imediato

```
DO $$DECLARE i int:= 0;
BEGIN
    WHILE I <= 1000000 LOOP
        INSERT INTO testaEMP select
* from empregado;
        I := I + 1;
    END LOOP;
END$$;
```

# Outros Objetos

---

- **Procedimento:** bloco nomeado, guardado no BD, pode ser chamado por qualquer outro procedimento/função; pode receber argumentos na chamada. **Não retorna valor.**

e.g., `processarProdução(mês);`

- **Função:** bloco nomeado, guardado no BD, podendo ser chamado por qualquer outro procedimento/função; pode receber argumentos na chamada. **Retorna um valor e pode ser atribuído a uma variável.**

e.g., `Var_mensal:= calcularMensalidade(matriculaAluno);`

- **Trigger:** Rotina **disparada automaticamente** normalmente antes ou depois de um evento como comandos UPDATE, INSERT ou DELETE.

# Bloco PL/pgSQL

```
[ <<label>> ]  
[ DECLARE  
  declarations ]  
BEGIN  
  statements  
END [ label ];
```

- Palavras-chaves são **case-insensitive**
- Cada declaração deve vir seguida de ;
- Os blocos internos devem ter um ; depois do END
- O último END não necessita de ;
- Comentários: -- (linha) e /\* \*/ (bloco)

- Modularidade: códigos devem residir em **blocos pequenos**
- Processamento: decisão, repetição, sequência
- Variáveis: tipos primitivos (char, integer) e compostos (registros, vetores)



# Atribuição

---

```
variable :=/= expression;
```

```
BEGIN
```

```
    tax = subtotal * 0.06;
```

```
    my_record.user_id := 20;
```

```
    valor := (horas_trabalhadas * salario_hora) + bonus;
```

```
    país = 'France';
```

```
    país := UPPER('Canada');
```

```
    valid_id := TRUE;
```

```
    empRec.firstName = 'Antonio';
```

```
    empRec.lastName := 'Ortiz';
```

```
END;
```

# Declarações

## ■ Variáveis

**name [ CONSTANT ] type [ NOT NULL ] [ { DEFAULT | := | = } expression ];**

- qualquer datatype válido para SQL
- **DEFAULT** especifica o valor inicial atribuído à variável
  - Se não for fornecida, a variável será inicializada com valor nulo
- **CONSTANT** impede que a variável seja atribuída após a inicialização

```
user_id integer;
quantity numeric;
url varchar;
arow RECORD;
quantity integer DEFAULT 32;
url          varchar          :=
'http://mysite.com';
user_id CONSTANT integer = 10;
```

# Declarações de tipos

```
myfield tablename.columnname%TYPE;
```

- **%type**: sempre terá o **mesmo tipo** de uma **coluna da tabela**

```
idcli cliente.codcli%type;
```

```
nomecli cliente.nome%type;
```

cliente









General Columns Advanced Constraints Parameters Security SQL

Inherited from  
table(s)

Select to inherit from...

Columns

+

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
 	codcli	integer			<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes
 	nome	character varying	30		<input type="checkbox"/> No	<input type="checkbox"/> No
 	endereco	character varying	30		<input type="checkbox"/> No	<input type="checkbox"/> No
 	telefone	character varying	12		<input type="checkbox"/> No	<input type="checkbox"/> No

# Select ... Into...

---

**SELECT expressions INTO [STRICT] target FROM ...;**

Onde target pode ser uma **variável de registro**, uma **variável de linha** ou uma **lista de variáveis** (separadas por vírgula)

**Select x into strict y from tabela where id = 1;**

DECLARE

bonus numeric(8,2);

emp\_id integer := 100;

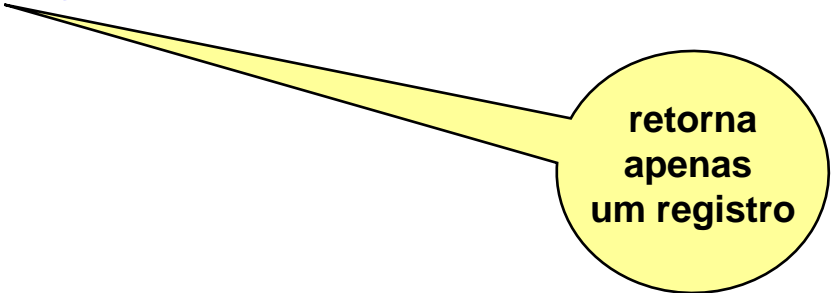
BEGIN

SELECT salary \* 0.10 **INTO** bonus

FROM employees

WHERE **employee\_id = emp\_id;**

END;



retorna  
apenas  
um registro

# Exemplo 1 - VerificaCliente

**DO \$\$**

**BD: PEDIDOS**

Declare nomeVar varchar(40);

**Begin**

select nome **into strict** nomeVar  
from cliente

where **codcli = 2;**

raise notice 'Nome = %', nomeVar;

**Exception**

When **no\_data\_found** then

raise notice 'Nenhum cliente com essa matrícula  
foi encontrado';

**End\$\$;**

# Visualizar dados e mensagens

---

**raise notice** 'Mensagem a ser mostrada';

Ou

**RAISE NOTICE** "O valor da variável é %",  
variavel;

**\*\*** No exemplo, o símbolo % será substituído pelo valor presente em variavel.

E.g., **raise notice** 'Nome = %', nomeVar;

# Comando IF ... THEN

---

IF boolean-expression THEN statements END IF;

```
IF v_user_id <> 0 THEN
    UPDATE users SET email = v_email
    WHERE user_id = v_user_id;
END IF;
```

IF boolean-expression THEN statements  
ELSE statements  
END IF;

```
IF v_count > 0 THEN
    INSERT INTO users_count (count) VALUES (v_count);
    RETURN 't';
ELSE RETURN 'f';
END IF;
```

# Comando IF ... THEN .. ELSIF

---

IF boolean-expression THEN statements [ ELSIF boolean-expression THEN statements [ ELSIF boolean-expression THEN statements ...]] [ ELSE statements ]  
END IF;

```
IF number = 0
  THEN result := 'zero';
  ELSIF number > 0
    THEN result := 'positive';
    ELSIF number < 0
      THEN result := 'negative';
      ELSE
        -- hmm, that number is null
        result := 'NULL';
      END IF;
    END IF;
```



# CASE...

---

CASE search-expression

```
    WHEN expression [, expression [ ... ]] THEN statements  
    [ WHEN expression [, expression [ ... ]] THEN statements  
... ]  
    [ ELSE statements ]  
END CASE;
```

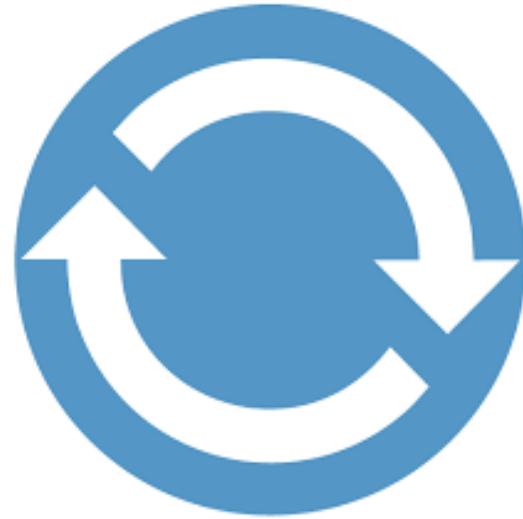
CASE

```
    WHEN x BETWEEN 0 AND 10  
        THEN msg := 'value is between zero and ten';  
    WHEN x BETWEEN 11 AND 20  
        THEN msg := 'value is between eleven and twenty';  
END CASE;
```

# Loops

---

- LOOP
- WHILE
- FOR



# Loop

---

[<<label>> ]

LOOP

statements

EXIT [ label ] [ WHEN boolean-expression ];

END LOOP [ label ];

<<ablock>>

LOOP

-- some computations

IF stocks > 100000

THEN EXIT ablock;

-- computations here will be skipped when stocks > 100000

END Loop;

# While

---

```
[ <<label>> ]  
WHILE boolean-expression LOOP  
    statements  
END LOOP [ label ];
```

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0  
LOOP  
    -- some computations here  
END LOOP;
```

# For (variável integer)

---

```
[ <label> ]  
FOR name IN [ REVERSE ] expression .. expression [ BY  
expression ] LOOP  
    statements  
END LOOP [ label ];
```

```
FOR i IN 1..10 LOOP  
    <comandos>  
END LOOP;  
FOR i IN REVERSE 10..1 LOOP  
    <comandos>  
END LOOP;  
FOR i IN REVERSE 10..1 BY 2 LOOP  
    <comandos>  
END LOOP;
```

## Exemplo 2- Instestabloco

Obs: criar a tabela **TestaBloco**

**create table testa\_bloco (coluna1 integer Primary Key, coluna2 date);**

Do \$\$

DECLARE

I INT := 0;

BEGIN

WHILE I <= 10 LOOP

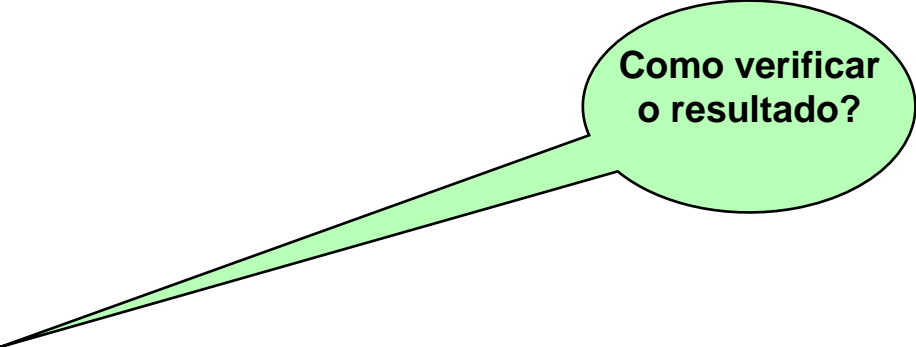
INSERT INTO TESTA\_BLOCO(coluna1,coluna2)

VALUES (I,current\_date);

I := I + 1;

END LOOP;

END\$\$;



Como verificar  
o resultado?

**Pode usar o BD PEDIDOS**

# Exemplo 3: atualiza\_status\_estoque.sql

**\*\* Antes:**

Alter table produto add  
status varchar(40);

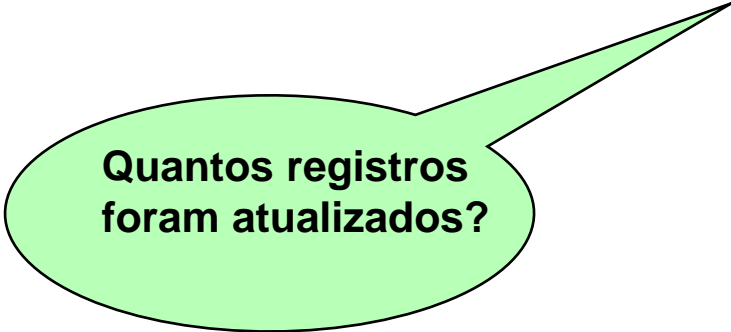
Alter table produto add  
quantest integer;

Update produto

Set quantest = 45

Where codprod = 1;

Select \* from produto;



Quantos registros  
foram atualizados?

**Do \$\$**

Declare **qtd\_atual** produto.quantest%type;

Begin

select **quantest** into **qtd\_atual** from produto  
where **codprod = 1**;

if **qtd\_atual > 30** then

update produto

set status = 'Estoque dentro do esperado'

where **codprod = 1**;

else

update produto

set status = 'Estoque fora do limite  
minimo'

where **codprod = 1**;

end if;

**End\$\$;**











# %type

**vendedor**

General **Columns** Advanced Constraints Parameters Security SQL

Inherited from table(s)

**Columns** +

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
 	codvend	integer			<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes
 	nome	character varying	30		<input type="checkbox"/> No	<input type="checkbox"/> No
 	datanasc	date			<input type="checkbox"/> No	<input type="checkbox"/> No
 	salariofixo	numeric	12	2	<input type="checkbox"/> No	<input type="checkbox"/> No
 	faixacomissao	character	1		<input type="checkbox"/> No	<input type="checkbox"/> No

Coluna ou campo

**cod** vendedor.codvend%type;

**nomeVar** vendedor.nome%type;



# Variável Registro

Nome\_variável **tabela%ROWTYPE;**

- Variável com exatamente a mesma **estrutura/tipo** de uma **tabela** = **REGISTRO**

Declare

**V\_vendedor\_rec** **vendedor%ROWTYPE;**



**vendedor**

General Columns Advanced Constraints Parameters Security SQL

Inherited from table(s)

Columns						
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
	codvend	integer			<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes
	nome	character varying	30		<input type="checkbox"/> No	<input type="checkbox"/> No
	datanasc	date			<input type="checkbox"/> No	<input type="checkbox"/> No
	salariofixo	numeric	12	2	<input type="checkbox"/> No	<input type="checkbox"/> No
	faixacomissao	character	1		<input type="checkbox"/> No	<input type="checkbox"/> No

## Exemplo 4

---

```
DO $$  
  DECLARE  
    v_vendedor vendedor%ROWTYPE;  
Begin  
  SELECT codvend, nome INTO v_vendedor  
  FROM vendedor  
  WHERE codvend = 1;  
  raise notice 'Vendedor selecionado = %',v_vendedor.nome;  
End$$;
```