



**Instituto Federal**  
Campus Goiânia

**Bacharelado em Sistemas de Informação**

# **Banco de Dados II**



**Prof. Dory Gonzaga Rodrigues**





## Agenda

- STORED PROCEDURE





## Regra de Negócios

### Introdução

- Antigamente dados e programas compartilhavam o mesmo equipamento e, conseqüentemente, a mesma memória do computador;
- Com o surgimento dos SGBD's, ocorreu a evolução da gerência dos dados em sistemas de informação;
- Com os SGBD's, ocorre a independência dos dados em relação aos programas;
- Como o próprio nome já diz, SGBD's são aplicações especialistas em Gerência de DADOS.
- Os sistemas aplicativos devem focar no controle das funcionalidades do sistema e no processamento dos dados, denominado: Regras de Negócio;





## Regra de Negócios

### Conceito

- **Regras de Negócio** são expressões da linguagem natural utilizada para estabelecer políticas e práticas;

- Definem ou restringem algum aspecto do negócio e, conseqüentemente, devem ser representadas através de códigos/funcionalidades da aplicação desenvolvida;

- Exemplo de Regras de Negócio para um sistema acadêmico:

RN1. Quantidade máxima de matrículas por Semestre Letivo: em um semestre letivo, um aluno não pode se inscrever em uma quantidade de disciplinas cuja a soma de créditos ultrapasse 20.

RN2. Quantidade de alunos por disciplina: uma oferta de disciplina em uma turma não pode ter mais de 25 alunos matriculados.





## Regra de Negócios

### Classificando o uso dos SGBD's com foco nas regras de negócio

#### SGBD passivo (tradicional):

- Realizam as tarefas de Inclusão/Alteração/Exclusão/Seleção quando solicitadas ou pelo programa ou pelo usuário que acessa diretamente o banco de dados;
- No máximo executa ações de imposição de regras de integridade dos dados no banco;

#### SGDB ativo:

- SGBD passivo + Capacidade de detectar ocorrência de EVENTOS e executar funções, independente de uma solicitação específica, após a ocorrência de um EVENTO esperado (gatilho);





## Regra de Negócios

### Implementando regras de negócio em um SGBD

- Regras de Negócio atuam sobre:

Tabelas

Campos

Eventos

- Objetos utilizados para implementar regras de negócio:

Funções (Function)

Procedimentos (Store Procedures)

Gatilhos (triggers)





## Regra de Negócios

### Implementando regras de negócio em um SGBD

#### Vantagens:

- Simplificação da execução de instruções SQL pela aplicação.
- Transferência de parte da responsabilidade de processamento para o servidor.
- Facilidade na manutenção, reduzindo a quantidade de alterações na aplicação.

#### Desvantagens:

- Necessidade de maior conhecimento da sintaxe do banco de dados para escrita de rotinas em SQL.
- As rotinas ficam acessíveis. Alguém que tenha acesso ao banco poderá visualizar e alterar o código.





## Regra de Negócios

### Modelo de Implementando de uma regra de negócio em um SGBD

**RN1.** Quantidade de alunos por disciplina: uma oferta de disciplina em uma turma não pode ter mais de 25 alunos matriculados.

- R1 deve atuar sobre:

Tabelas Envolvidas: Alunos, Disciplinas, Turma\_Disciplina, Matricula

Campos: Turma\_Disciplina.vagas

Turma\_Disciplina.qtde\_matriculas

- **Objetos utilizados** para implementar a regra de negócio:

Objeto: Gatilhos (triggers)

Descrição: antes de inserir uma nova Matricula é feita a verificação da RN1 ( $qtde\_matriculas < vagas$ ).







## Funções

### Criando uma função no PostgreSQL

- Sintaxe:

```
CREATE FUNCTION nome (modo p1 tipo, modo p2 tipo, ..., modo pn tipo)
    RETURNS tipo AS
    $$ BEGIN
        lógica
    END; $$
LANGUAGE NomeLinguagem;
```

- Como toda função, após a cláusula CREATE FUNCTION vem o nome da função e a lista de parâmetros, bem como o tipo de dados de cada parâmetro;
- Após a palavra reservada RETURNS vem o tipo de dados que a função irá retornar;
- Entre BEGIN e END vem a lógica da função, onde cada instrução deve terminar com “;”





## Funções

### Criando uma função no PostgreSQL

- Sintaxe:

```
CREATE FUNCTION nome (modo p1 tipo, modo p2 tipo, ..., modo pn tipo)
    RETURNS tipo AS
    $$ BEGIN
        lógica
    END; $$
LANGUAGE NomeLinguagem;
```

- O sinal de dólar circunda a função, pois a definição da função deve ser fornecida como uma string;
- Após a palavra reservada LANGUAGE deve ser informada a linguagem utilizada para a declaração da função;
- O PostgreSQL aceita por padrão as linguagens: SQL, PL/PgSQL e C
- Outras Linguagens podem ser adicionadas como extensão;





## Funções

### PL/PgSQL: Criando uma função no PostgreSQL

- Exemplo:

```
CREATE FUNCTION incrementa ( x integer)
  RETURNS integer AS
  $$ BEGIN
      RETURN x + 1 ;
  END; $$
LANGUAGE PLPGSQL;
```

- Chamando um função:

```
SELECT incrementa ( 10 );
```

Data Output		Expla
		incrementa integer
1		11





## Funções

### PL/PgSQL: Os Parâmetros das funções

- A linguagem PL/PgSQL possui quatro tipos de parâmetros

IN, OUT, INOUT e VARIADIC.

- Por padrão, quando não especificado, os parâmetros são definidos como de entrada (IN), como mostrado no exemplo a seguir:

```
CREATE FUNCTION soma ( x integer, y integer )  
  RETURNS integer AS  
  $$ BEGIN  
    RETURN x + y ;  
  END; $$  
LANGUAGE PLPGSQL;
```

- A função soma recebe os parâmetros de entrada x e y retornando a soma dos valores fornecidos como parâmetro.





## Funções

### PL/PgSQL: Os Parâmetros das funções

- O parâmetro de saída **OUT** permite que o valor seja retornado como resultado após a execução da função, observe o exemplo a seguir:

```
CREATE FUNCTION MaxMin ( x int, y int, z int, OUT max int, OUT min int ) AS $$  
BEGIN  
    max :=GREATEST(x, y, z);  
    min := LEAST(x, y, z);  
END; $$  
LANGUAGE PLPGSQL;
```

- Os parâmetros **OUT** são retornados sem a necessidade de utilizar o **RETURN**
- A função recebe os parâmetros de entrada **x**, **y** e **z** retornando o conteúdo das variáveis de saída **max** e **min** declaradas como modo **OUT**.
- **GREATEST** e **LEAST** são funções internas do PostgreSQL, e retornam o maior e menor valores, respectivamente.



## Funções

## PL/PgSQL: Os Parâmetros das funções

- O parâmetro **INOUT** é uma combinação de **IN** e **OUT**, ou seja, permite que o parâmetro seja fornecido como entrada e retorne o resultado, observe o exemplo a seguir:

```
CREATE FUNCTION Quadrado ( IN OUT x int ) AS $$  
    BEGIN  
        x := x * x;  
    END; $$  
LANGUAGE PLPGSQL;
```

- Chamando um função:

```
SELECT Quadrado ( 3 );
```

Data Output		Explain	Messages
	quadrado integer		
1	9		
DOS	Ln 7, Col 1, Ch 105		





## Funções

### PL/PgSQL: Os Parâmetros das funções

- Atenção: o parâmetro tipo **VARIADIC** é um parâmetro de entrada seja de tamanho variável, observe o exemplo a seguir:

```
CREATE FUNCTION Soma ( VARIADIC x int[] , OUT total int ) AS $$  
BEGIN  
    SELECT INTO total sum( x[ i ] )  
    FROM generate_subscripts( x, 1 ) AS g( i );  
END; $$  
LANGUAGE PLPGSQL;
```

- **INTO** é responsável por atribuir o valor retornado pela função **sum( )** ao parâmetro **OUT total**;

- **generate\_subscripts()** pega os valores contidos no vetor e atribui seus dados a uma tabela apelidada de "g" com uma única coluna chamada "i"

- Chamando um função:

```
SELECT Quadrado ( 3, 7, 10 );
```

Data Output		Explain	Messages
	soma integer		
1	20		





## Funções

### PL/PgSQL: Exercícios

1) Faça uma função que retorne a idade de uma pessoa. Deve-se passar como parâmetro a data de nascimento.

Utilizando o Banco Loja CD's:

2) Faça uma função que retorne a média de preço de venda dos CDs de uma determinada gravadora. Deve-se passar como parâmetro o código da gravadora.

3) Faça uma função que retorne o nome do CD e o nome da Música. Deve-se passar como parâmetro o código do CD e o número da faixa do CD.

4) Faça uma função que retorne a quantidade de faixas e o tempo total de músicas de um CD. Deve-se passar como parâmetro o código da gravadora e o código do CD.

5) Faça uma função que retorne os nomes das músicas de um CD. Deve-se passar como parâmetro o código do CD.

