Universidade do Minho

Ano Letivo: 2023/24

Turnos: PL3/PL7

# Bases de Dados

PL10 – SQL Avançada

**Docente**: Diana Ferreira

Email: diana.ferreira@algoritmi.uminho.pt

Horário de Atendimento:

5° feira 16h-17h



#### Sumário

- 1 Vistas Procedimentos
  - 2 Funções 4 Triggers

#### Bibliografia:

- Connolly, T., Begg, C., Database Systems, A Practical Approach to Design, Implementation, and Management, Addison-Wesley, 4a Edição, 2004. **(Chapter 6 e 7)**
- Belo, O., "Bases de Dados Relacionais: Implementação com MySQL", FCA Editora de Informática, 376p, Set 2021. ISBN: 978-972-722-921-5. (Capítulo 4 e 5)



Uma vista é uma tabela virtual derivada de uma ou mais tabelas ou vistas existentes. É definida por uma query SQL e tem a aparência de uma tabela, mas não armazena nenhum dado. Em vez disso, recupera os dados dinamicamente das tabelas/vistas subjacentes sempre que é consultada.

1) Apresentar valores estatísticos sobre os medicamentos prescritos, como o número de vezes prescrito, a quantidade total, média, minima e máxima

**CREATE VIEW** vw\_medication\_stats **AS** 

SELECT m.nome as 'Medicamento',

COUNT(\*) as 'Nr de vezes prescrito',

SUM(quantidade) as 'Quantidade Total',

ROUND(AVG(quantidade),0) as 'Quantidade Média',

MAX(quantidade) as 'Quantidade Máxima',

MIN(quantidade) as 'Quantidade Mínima'

FROM medicamentos m

INNER JOIN prescricoes p USING (id\_med)

GROUP BY m.nome;



#### PREPARED STATEMENTS



Uma query pré-compilada permite a criação de declarações SQL para posterior execução, otimizando a execução de consultas repetitivas sem variação sintática, com dinamismo apenas nos parâmetros.

-- Preparar a query pré-compilada

PREPARE ps\_medicos\_por\_especialidade FROM 'SELECT f.nome FROM funcionarios f INNER JOIN medicos m USING(nr\_mec) INNER JOIN especialidades e USING(cod\_especialidade) WHERE e.des\_especialidade = ?';

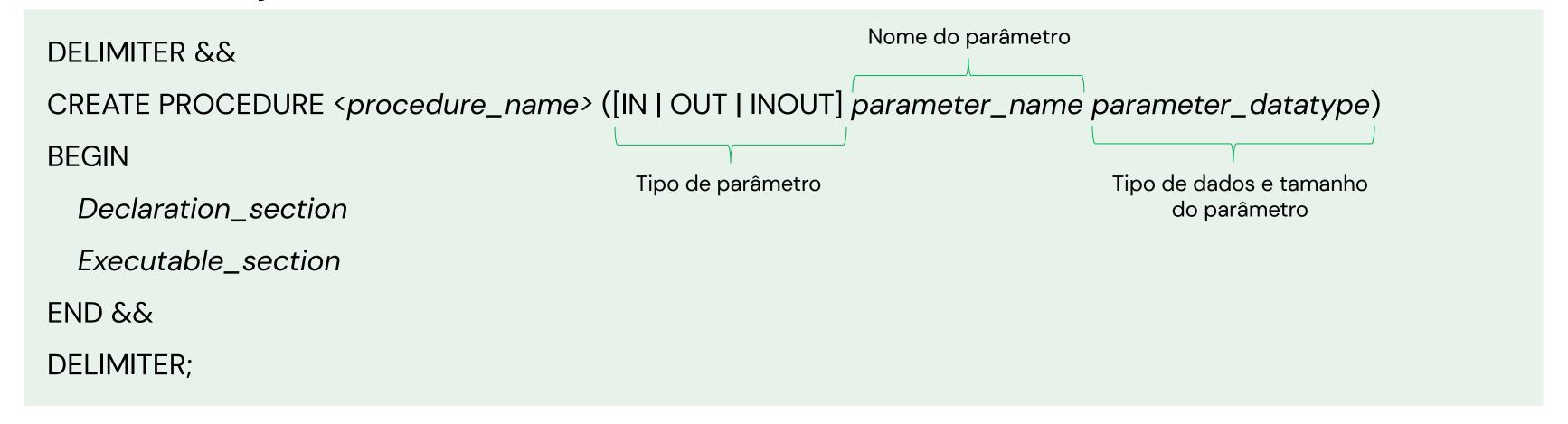
- -- Atribuir os valores aos parâmetros (caso existam) SET @especialidade = 'Neurologia';
- -- Executar a query pré-compilada EXECUTE ps\_medicos\_por\_especialidade USING @especialidade;
- -- Executar a query pré-compilada usando outros valores SET @especialidade = 'Cardiologia'; EXECUTE ps\_medicos\_por\_especialidade USING @especialidade;



#### **PROCEDIMENTOS**

Um procedimento é uma coleção de instruções SQL pré-compiladas armazenadas na BD que mais tarde podem ser invocadas.

Síntaxe para criação de um procedimento:



<u>IN</u> - É o modo padrão, permite passer parâmetros de entrada.

Tipo de Parâmetro

<u>OUT</u> - É usado para passar um parâmetro como saída. O seu valor pode ser alterado dentro do procedimento armazenado e o valor alterado (novo) é passado de volta para o programa que invoca o procedimento.

<u>INOUT</u> - É uma combinação dos modos IN e OUT.



Podemos criar um procedimento sem parâmetros. A rotina a baixo descrita é um procedimento que retorna todos os médicos do hospital.

DELIMITER &&

CREATE PROCEDURE GetMedicos()

BEGIN

SELECT \* FROM medicos;

END &&

CALL GetMedicos();

DELIMITER;



Sabemos que a tabela de preços está constantemente sujeita a alterações. De seguida, encontra-se uma rotina para atualizar o preço de um determinado procedimento clínico.

**DELIMITER &&** 

CREATE PROCEDURE UpdateProcedimento (IN proc INT, IN new\_preco DECIMAL(5,2))

**BEGIN** 

UPDATE procedimentos SET preco = new\_preco where cod\_proc=proc;

END &&

DELIMITER;





Sabemos que um hospital está constantemente sujeito a auditorias e a processos de avaliação dos serviços de saúde prestados. De seguida, encontra-se uma rotina para consultar o número total de consultas num determinado ano.

**DELIMITER &&** 

CREATE PROCEDURE GetConsultasYear (IN ano INT, OUT total\_consultas INT)

**BEGIN** 

SELECT count(\*) INTO total\_consultas FROM consultas WHERE YEAR(dta\_ini)=ano;

END &&

DELIMITER;

CALL GetConsultasYear(2020, @total\_consultas);
SELECT @total\_consultas AS total\_consultas\_2020;

#### **→** PROCEDIMENTOS

No exemplo a seguir, encontra-se uma rotina que aceita um parâmetro IN e outro INOUT. Este procedimento, incrementa o contador de acordo com o valor específico no parâmetro inc.

```
DELIMITER &&

CREATE PROCEDURE Contador (IN inc INT, INOUT contador INT)

BEGIN

SET contador = contador + inc;

END &&

DELIMITER;
```

```
    SET @contador = 1;
    CALL Contador(1, @contador); -- 2
    SELECT @contador;
```



Uma função é um programa armazenado que devolve um único valor. Tipicamente, usam-se funções para encapsular fórmulas ou regras de negócio comuns que são reutilizáveis entre instruções SQL ou programas armazenados.

#### Síntaxe para criar uma função:

DELIMITER &&

Nome do parâmetro

CREATE FUNCTION < function\_name > (parameter\_name parameter\_datatype) Tipo de dados e tamanho do parâmetro

RETURNS datatype [NOT] DETERMINISTIC

BEGIN Tipo de dados a retornar Uma função determinística retorna sempre o mesmo resultado para os mesmos parâmetros de entrada.

Body\_section — é preciso especificar pelo menos uma instrução RETURN.

END &&

DELIMITER;



EXEMPLO → Função Idade

DELIMITER &&

CREATE FUNCTION idade (dta DATE)

RETURNS INT

NOT DETERMINISTIC

BEGIN

RETURN TIMESTAMPDIFF(YEAR, dta, CURDATE());

END &&

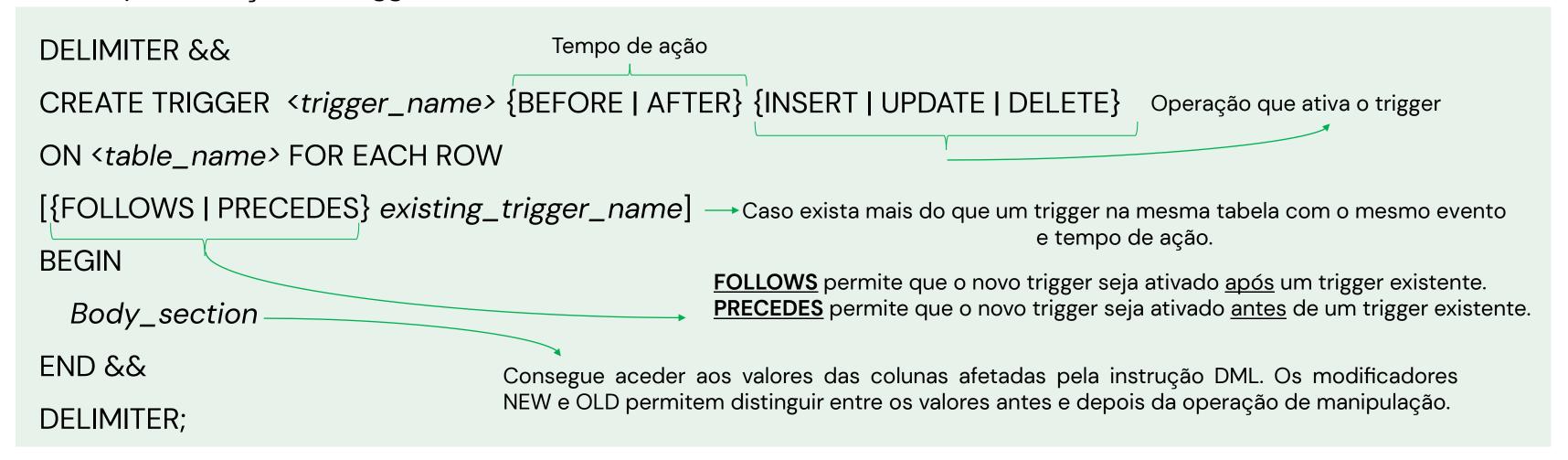
DELIMITER;

NOTA: pode precisar de alterar esta variável: SET GLOBAL log\_bin\_trust\_function\_creators = 1;



Um trigger é invocado automaticamente quando uma operação de alteração específica (instrução INSERT, UPDATE, ou DELETE) é executada sobre uma determinada tabela. Os triggers são úteis para tarefas como a aplicação de regras comerciais ou até para validação de dados quando inseridos na base de dados.

#### Síntaxe para criação de trigger:





EXEMPLO → Suponto que existe o atributo nr\_consultas na tabela especialidades, crie um trigger para atualizar o nr\_consultas, sempre que uma nova consulta é inserida

```
DELIMITER //
CREATE TRIGGER update_nrconsultas
AFTER INSERT ON consultas FOR EACH ROW
BEGIN
DECLARE id_esp INT;
SELECT e.cod_especialidade INTO id_esp FROM consultas c
INNER JOIN medicos m ON m.nr_mec=c.id_medico
INNER JOIN especialidades e USING(cod_especialidade)
WHERE c.nr_episodio=NEW.nr_episodio;
UPDATE especialidades SET nr_consultas=nr_consultas+1 WHERE
cod_especialidade=id_esp;
END //
DELIMITER;
```

Resolução de Exercícios

<u>Ficha de Excercícios PL10</u>