Universidade do Minho

Ano Letivo 2021/22

Bases de Dados

PL11 – Procedures, Triggers, Functions and Handlers

Docente: Diana Ferreira
Email: diana.ferreira@algoritmi.uminho.pt

Horário de Atendimento:
4ª feira 10h-11h | DI 1.15

Sumário

Bibliografia:

- Connolly, T., Begg, C., Database Systems, A Practical Approach to Design, Implementation, and Management , Addison-Wesley, 4a Edição, 2004. **(Chapter 8)**
- Belo, O., "Bases de Dados Relacionais: Implementação com MySQL", FCA – Editora de Informática, 376p, Set 2021. ISBN: 978-972-722-921-5. **(Chapter 6 and 7)**

Variáveis, Constantes e Atribuições

Variáveis e **constantes** têm que ser declaradas, antes de serem referenciadas noutras instruções, através da instrução DECLARE. Se uma variável for declarada sem especificar um valor padrão, o seu valor será NULL.

```
DECLARE nome tipo_de_dados(tamanho) [NOT NULL] [DEFAULT default_value];
```

```
DECLARE nome CONSTANT tipo_de_dados(tamanho);
```

As variáveis podem ser **atribuídas** de duas maneiras:

- usando a instrução de atribuição normal através da instrução SET

```
SET nome = valor/expressão;
```

- como resultado de uma instrução SQL SELECT ou FETCH

```
SELECT valor/expressão INTO nome FROM table_name WHERE condition;
```

Estruturas de controlo de fluxo

CONDITIONAL STATEMENTS

- Conditional IF statement
- Conditional CASE statement

ITERATION STATEMENTS

- LOOP statement
- WHILE statement
- REPEAT statement
- FOR statement

Estruturas de controlo de fluxo

Conditional IF

A instrução IF tem três formas:

- instrução IF-THEN simples;
- instrução IF-THEN-ELSE;
- instrução IF-THEN-ELSEIF-ELSE.

Síntaxe:

```
IF (condition) THEN <SQL statement list>
[ELSEIF (condition) THEN <SQL statement list>]
[ELSE <SQL statement list>]
END IF;
```

Estruturas de controlo de fluxo

Conditional CASE

A instrução CASE tem duas formas:

- instrução CASE simples;
- instrução CASE de procura/pesquisa;

Síntaxe:

```
CASE variable
        WHEN value1 THEN statements
        WHEN value2 THEN statements
        ...
        [ELSE else-statements]
END CASE;
```

```
CASE
        WHEN condition1 THEN statements
        WHEN condition2 THEN statements
        ...
        [ELSE else-statements]
END CASE;
```

Estruturas de controlo de fluxo

Conditional IF vs. Conditional CASE

De seguida, encontram-se algumas orientações para escolher entre IF ou CASE:

- Uma instrução CASE simples é mais legível e eficiente do que uma instrução IF quando se compara uma única expressão com um intervalo de valores exclusivos.

- Quando se verifica expressões complexas com base em vários valores, a instrução IF é mais fácil de entender.

- Na instrução CASE é necessário garantir que pelo menos uma das condições CASE é correspondida. Caso contrário, é preciso definir um manipulador de erros (*handler*) para capturar o erro. Não é necessário ter este cuidado com a instrução IF.

Estruturas de controlo de fluxo

LOOP

A instrução LOOP permite executar uma ou mais instruções repetidamente. O LOOP termina quando uma condição é satisfeita usando a instrução LEAVE.

Síntaxe:

```
[label_name:]
LOOP
    statement_list
    [IF condition THEN
        LEAVE [label_name];
    END IF;]
END LOOP [label_name];
```

Estruturas de controlo de fluxo

WHILE

O ciclo WHILE é uma instrução que executa um bloco de código repetidamente enquanto uma condição for verdadeira. O WHILE verifica a condição no início de cada iteração.

Síntaxe:

```
[label_name:]
WHILE condition DO
    statement_list
    [IF condition THEN
        LEAVE [label_name];
    END IF;]
END WHILE [label_name];
```

Estruturas de controlo de fluxo

REPEAT

O ciclo REPEAT é uma instrução que executa um bloco de código repetidamente até que a condição seja verdadeira.

Síntaxe:

```
[label_name:]
REPEAT
    statement_list
    [IF condition THEN
        LEAVE [label_name];
    END IF;]
UNTIL condition
END REPEAT [label_name];
```

Estruturas de controlo de fluxo

FOR

O ciclo FOR é uma instrução que executa um bloco de código repetidamente até que um determinado índice seja alcançado. O WHILE verifica a condição no início de cada iteração.

Síntaxe:

```
[label_name:]
FOR index_variable
    AS query_specification DO
    statement_list
    [IF condition THEN
        LEAVE [label_name];
    END IF;]
END FOR [label_name];
```

Cursores

O que é um cursor e para que serve?

Uma instrução SELECT pode ser usada se a consulta retornar uma e apenas uma linha. Para manipular uma consulta que pode retornar um número arbitrário de linhas (ou seja, zero, uma ou mais linhas), usa-se um **cursor**. Um **cursor** permite iterar um conjunto de linhas retornadas por uma consulta e processar cada linha individualmente. Com efeito, o cursor atua como um ponteiro para uma determinada linha do resultado da consulta. Os cursores podem ser usados em procedimentos, *triggers* e funções.

O cursor do MySQL é:

- ***read-only*:** não é possível atualizar dados na tabela subjacente através do cursor;
- ***non-scrollable*:** as linhas apenas podem ser acedidas pela ordem determinada pela instrução SELECT. Não é possível saltar linhas ou pular para uma linha específica no conjunto de resultados.
- **assensitive**: aponta para os dados reais em vez de usar uma cópia temporária dos dados.

Cursores

Como se usa um cursor?

Um cursor deve ser declarado e aberto antes de ser usado e deve ser fechado quando já não for necessário.

1º Declarar o cursor – A declaração do cursor deve ser após qualquer declaração de variável.

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

2º Abrir o cursor – A instrução OPEN inicializa o *result set* para o cursor.

```
OPEN cursor_name;
```

3º Ler os resultados – A instrução FETCH é usada para recuperar a próxima linha apontada pelo.

```
FETCH cursor_name INTO variables list;
```

4º Fechar o cursor – Desative o cursor e liberte a memória associada a ele usando a instrução CLOSE.

```
CLOSE cursor_name;
```

Estruturas básicas de programas

As rotinas pertencem à BD e são armazenadas no servidor. Os três componentes principais são:

Procedimentos (*Stored Procedures*)

- Blocos de código armazenados na BD que são pré-compilados.
- Podem operar nas tabelas da BD e retornar escalares ou conjuntos de resultados.

Funções (*Functions*)

- Pode ser usado como uma função interna para fornecer capacidade expandida às instruções SQL.
- Pode receber qualquer número de argumentos e retornar um único valor ou conjuntos de resultados.

Gatilhos (*Triggers*)

- É despoletado em resposta a operações de BDs padrão numa tabela específica.
- Pode ser usado para executar automaticamente operações de BDs adicionais quando ocorre o evento de acionamento.

Procedimentos (*Procedure*)

O que é um procedimento?

Um **procedimento** (geralmente chamado de **procedimento armazenado**) é uma coleção de instruções SQL pré-compiladas armazenadas na base de dados que mais tarde podem ser invocadas.

Do ponto de vista empresarial, é geralmente necessário executar tarefas específicas como limpeza da base de dados, processamento de folhas de pagamento, entre outras. Essas tarefas envolvem várias instruções SQL que podem ser agrupadas numa única tarefa, criando um procedimento armazenado na base de dados.

Os procedimentos podem ser invocados usando gatilhos (*triggers*), outros procedimentos e aplicações em Java, Python, PHP, etc. Um procedimento que se invoca a ele mesmo é chamado de procedimento armazenado recursivo.

Procedimentos (*Procedure*)

Vantagens

– **Centraliza a lógica de negócios na BD:** os procedimentos armazenados são reutilizáveis e transparentes para qualquer aplicativo.

– **Aumenta a segurança da BD:** O administrador da BD pode conceder permissões de acesso apenas a procedimentos armazenados na BD sem conceder permissões nas tabelas subjacentes.

– **Reduz o tráfego de rede entre os aplicativos e o MySQL Server**: o aplicativo precisa apenas de enviar o nome e os parâmetros do procedimento armazenado em vez de enviar várias instruções SQL.

Procedimentos (*Procedure*)

Desvantagens

– **Consumo de recursos:** caso sejam usados muitos procedimentos armazenados, o uso de memória de cada conexão aumentará substancialmente. Além disso, o uso excessivo de um grande número de operações lógicas nos procedimentos armazenados aumentará o consumo de CPU;

– **Manutenção e Solução de Problemas:** É difícil depurar procedimentos armazenados no MySQL, porque, ao contrário de outros SGBDs, como Oracle e SQL Server, o MySQL não fornece nenhum recurso para depurar procedimentos.

Procedimentos (*Procedure*)

➡ Síntaxe para criação de um procedure

```
DELIMITER &&
CREATE PROCEDURE procedure_name ([IN | OUT | INOUT] parameter_name parameter_datatype)
BEGIN
    Declaration_section
    Executable_section
END &&
DELIMITER;
```

Nome do parâmetro

Tipo de parâmetro

Tipo de dados e tamanho do parâmetro

Tipo de Parâmetro

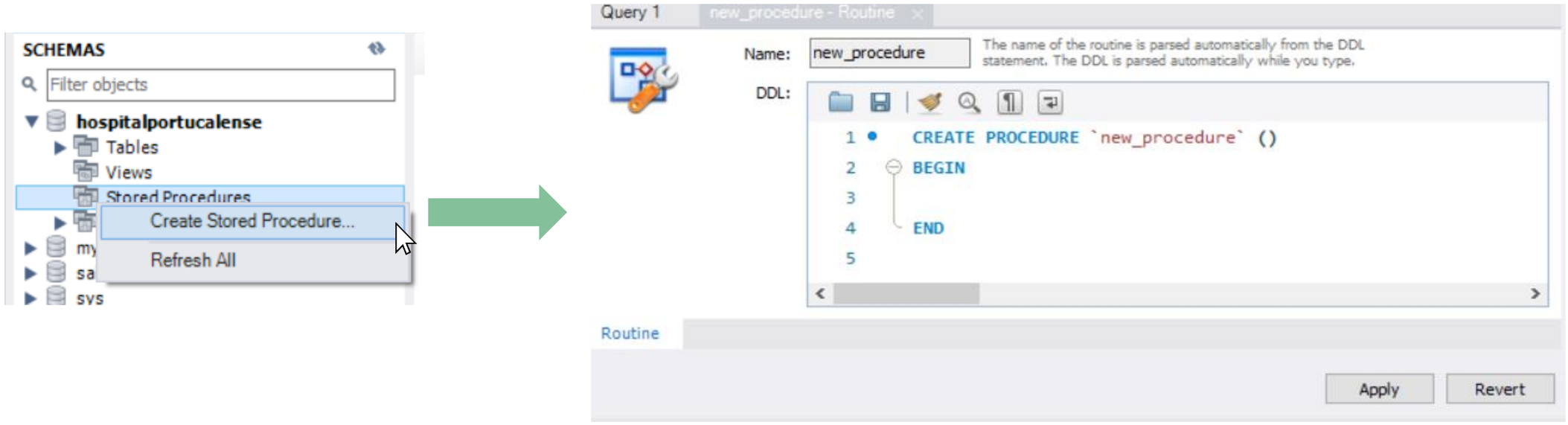
IN – É o modo padrão, permite passer parâmetros de entrada.

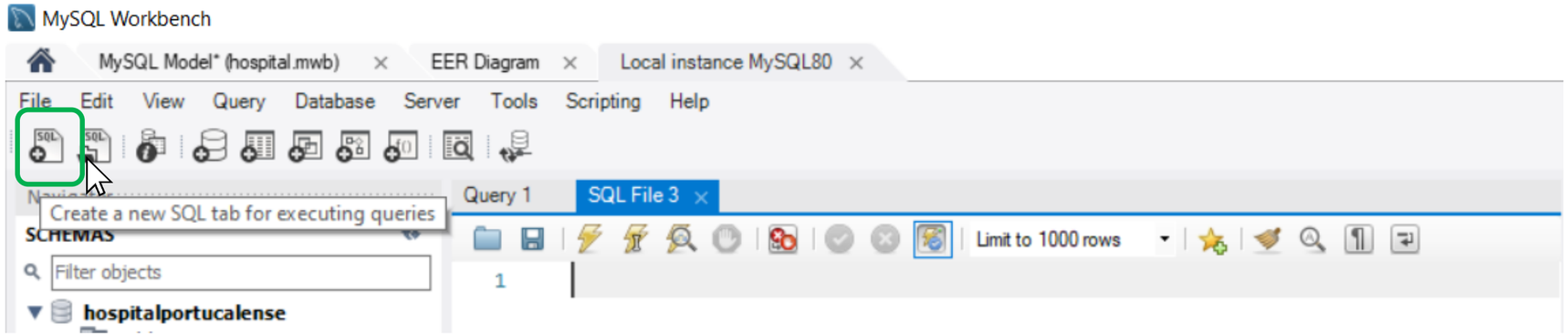
OUT – É usado para passar um parâmetro como saída. O seu valor pode ser alterado dentro do procedimento armazenado e o valor alterado (novo) é passado de volta para o programa que invoca o procedimento.

INOUT – É uma combinação dos modos IN e OUT.

Procedimentos (*Procedure*)

Criação de um procedure





Procedimentos (*Procedure*)

Procedimento sem parâmetros

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 &&
DELIMITER;
```

- CALL GetMedicos();

Procedimentos (*Procedure*)

Procedimento com parâmetros IN

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_procedimento=proc;
END &&
DELIMITER;
```

- CALL UpdateProcedimento(3000, 10.15);

Procedimentos (*Procedure*)

Procedimento com parâmetros OUT

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(hora_ini)=ano;
END &&
DELIMITER;
```

```
CALL GetConsultasYear(2020, @total_consultas);
SELECT @total_consultas AS total_consultas_2020;
```

Procedimentos (*Procedure*)

Procedimento com parâmetros INOUT

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;
```

Procedimentos (*Procedure*)

Exemplos de Procedimentos

```
DELIMITER $$
CREATE PROCEDURE GetCategoriaCliente( IN nr_cliente INT, OUT categoria VARCHAR(20))
BEGIN
DECLARE credito DECIMAL DEFAULT 0;
SELECT limite_credito INTO credito FROM clientes WHERE id_cliente = nr_cliente;
IF credito > 50000 THEN SET categoria = 'PLATINUM';
ELSEIF credito <= 50000 AND credito > 10000 THEN SET categoria = 'GOLD';
ELSE SET categoria = 'SILVER';
END IF;
END $$
DELIMITER ;
```

```
CALL GetCategoriaCliente(447, @level);
SELECT @level;
```

Procedimentos (*Procedure*)

Listar e apagar procedimentos no MySQL

Podemos listar todos os procedimentos armazenados no servidor MySQL da seguinte forma:

```
SHOW PROCEDURE STATUS [LIKE 'padrão' | WHERE condição_de_procura]
```

Se quisermos exibir procedimentos numa BDs específica, usamos a cláusula WHERE. Caso queiramos listar procedimentos com uma palavra específica, usamos a cláusula LIKE.

A instrução a seguir é usada para remover um procedimento armazenado no MySQL:

```
DROP PROCEDURE [ IF EXISTS ] nome_procedimento;
```

Handlers

Quando ocorre um erro dentro de um procedimento armazenado, é importante tratá-lo adequadamente, como continuar (CONTINUE) ou sair (EXIT) da execução do bloco de código atual e emitir uma mensagem de erro significativa. Para declarar um *handler*, usa-se a instrução DECLARE HANDLER.

Síntaxe:

Condição que ativa o handler

```
DECLARE action HANDLER FOR condition_value statement;
```

Tipo de acção **Instrução a executar**

A instrução DECLARE CONDITION permite declarar uma condição de erro. Após a sua declaração, a declaração do *handler* pode se referir à *condition_name* em vez de se referir à *condition_value*.

```
DECLARE condition_name CONDITION FOR condition_value;
```

Handlers

Condição que ativa o handler

```
DECLARE action HANDLER FOR condition_value statement;
```

Tipo de acção — **Instrução a executar**

A **acção** pode ser de um dos seguintes tipos:

- **CONTINUE:** a execução do bloco de código envolvente continua.
- **EXIT:** a execução do bloco de código envolvente termina.

A **condição de ativação** pode ser:

- Um código de erro do MySQL;
- Um valor SQLSTATE padrão: SQLWARNING , NOTFOUND ou SQLEXCEPTION.
- Uma condição nomeada associada a um código de erro MySQL ou a um SQLSTATE através da instrução instrução DECLARE CONDITION.

A **instrução a executar** pode ser uma instrução simples ou uma instrução composta delimitada pelas palavras-chave BEGIN e END.

Handlers

Regras de precedência

Caso existam vários *handlers* para o mesmo erro, o MySQL chamará o *handler* mais específico para tratar o erro com base nas seguintes regras:

- Um erro é sempre mapeado para um código de erro do MySQL porque no MySQL é o mais específico.
- Um SQLSTATE pode ser mapeado para muitos códigos de erro do MySQL, portanto, é menos específico.
- Um SQLEXCEPTION ou um SQLWARNING é a abreviação de uma classe de valores SQLSTATES, portanto, é o mais genérico.

Com base nas regras de precedência, o código de erro do MySQL, o SQLSTATE e o SQLEXCEPTION têm a primeira, a segunda e a terceira precedência, respectivamente,

Handlers

Exemplo

```
DELIMITER $$
CREATE PROCEDURE InserirFarmaco ( IN codigo INT, IN farmaco VARCHAR(45), IN desc VARCHAR(150))
BEGIN
DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error encountered' Message;
DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered' Message;
DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000' ErrorCode;
INSERT INTO Farmacos(id_farmaco,nome,descricao) VALUES(codigo,farmaco,desc);
SELECT COUNT(*) FROM Farmacos WHERE id_farmaco = codigo;
END$$
DELIMITER ;
```

Sair/terminar em caso de duplicação de chaves

Handlers

SIGNAL/RESIGNAL

Existe também a capacidade de sinalizar explicitamente as condições de exceção e conclusão através das instruções SIGNAL e RESIGNAL.

A instrução **SIGNAL** é usada para retornar um erro ou condição de aviso quando um programa armazenado é chamado, por exemplo, procedimento, função ou *trigger*. A instrução SIGNAL fornece controlo sobre quais informações que devem ser retornadas, como valor e mensagem.

```
SIGNAL SQLSTATE condition_value | condition_name
SET condition_information_1 = value_1, condition_information_2 = value_2, etc;
```

O *condition_information* pode ser MESSAGE_TEXT, MYSQL_ERRORNO, CURSOR_NAME, etc.

Handlers

SIGNAL/RESIGNAL

A instrução **RESIGNAL** também permite gerar um erro ou condição de aviso quando um programa armazenado é chamado, por exemplo, procedimento, função ou *trigger*. A instrução RESIGNAL é semelhante à SIGNAL em termos de funcionalidade e sintaxe, exceto:

- A instrução RESIGNAL deve ser usada num *handler* de erro ou aviso, caso contrário, a mensagem de erro "RESIGNAL quando o manipulador não está ativo" irá aparecer.

- É possível omitir todos os atributos da instrução RESIGNAL, até mesmo o valor SQLSTATE.

- Se usar a instrução RESIGNAL sozinha, todos os atributos serão os mesmos que foram passados para o *handler*.

Funções

O que é uma Função?

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

Ao contrário de um procedimento armazenado, é possível utilizar uma função armazenada em instruções SQL.

Funções

Síntaxe para criação de uma função

```
DELIMITER &&
CREATE FUNCTION function_name (parameter_name parameter_datatype)
RETURNS datatype [NOT] DETERMINISTIC
BEGIN
    Body_section
END &&
DELIMITER;
```

Nome do parâmetro

Tipo de dados e tamanho do parâmetro

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.

Síntaxe para remover uma função no MySQL:

```
DROP FUNCTION [ IF EXISTS ] nome_function;
```

Síntaxe para listar as funções no MySQL:

```
SHOW FUNCTION STATUS;
```

Funções

Exemplo de uma função

```
DELIMITER &&
CREATE FUNCTION idade (dta DATE)
RETURNS INT DETERMINISTIC
BEGIN
    RETURN TIMESTAMPPDIFF(YEAR, dta, CURDATE());
END &&
DELIMITER;
```

Gatilhos (*Triggers*)

➡ O que é um Trigger?

Um ***trigger*** é um programa armazenado que é **invocado automaticamente** quando uma operação de alteração específica (instrução SQL 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.

Gatilhos (*Triggers*)

Síntaxe para criação de um trigger

```
DELIMITER &&
CREATE TRIGGER  trigger_name {BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON table_name FOR EACH ROW
[{FOLLOWS | PRECEDES} existing_trigger_name]
BEGIN
    Body_section
END &&
DELIMITER;
```

Tempo de ação

Operação que ativa o trigger

Caso exista mais do que um trigger na mesma tabela com o mesmo evento e tempo de ação.

FOLLOWS permite que o novo trigger seja ativado após um trigger existente.
PRECEDES permite que o novo trigger seja ativado antes de um trigger existente.

Consegue aceder aos valores das colunas afetadas pela instrução DML. Os modificadores NEW e OLD permitem distinguir entre os valores antes e depois da operação de manipulação.

Síntaxe para remover um trigger no MySQL:

```
DROP TRIGGER [ IF EXISTS ] trigger_name;
```

Síntaxe para listar triggers no MySQL:

```
SHOW TRIGGERS [{FROM | IN} database_name]
[LIKE 'pattern' | WHERE search_condition];
```

Gatilhos (*Triggers*)

Exemplo de um trigger

```
DELIMITER $$
CREATE TRIGGER after_especialidades_update
AFTER UPDATE
ON especialidades FOR EACH ROW
BEGIN
    IF OLD.preco <> NEW.preco THEN
        UPDATE especialidades e, medicos m
        SET e.total_faturado = e. total_faturado + NEW.preco
        WHERE m.cod_especialidade=e.cod_especialidade
        AND m.num_mec=new.id_medico;
        END IF;
END$$
DELIMITER ;
```

Transações (*Transactions*)

➡ O que é uma Transação?

A transação MySQL permite executar um conjunto de operações para garantir que a base de dados nunca contém o resultado de operações parciais. Num conjunto de operações, se uma delas falhar, ocorre a reversão para restaurar a base de dados ao seu estado original. Se nenhum erro ocorrer, todo o conjunto de instruções será confirmado na base de dados.

- Para iniciar uma transação, usa-se a instrução START TRANSACTION. O BEGIN ou BEGIN WORK são os *aliases* da instrução START TRANSACTION.
- Para confirmar a transação atual e tornar as suas alterações permanentes, usa-se a instrução COMMIT.
- Para reverter a transação atual e cancelar as suas alterações, usa-se a instrução ROLLBACK.
- Para desabilitar ou habilitar o modo de auto_commit usa-se a instrução SET auto_commit.

```
SET autocommit = 0; ou SET autocommit = OFF;
```

Transações (*Transactions*)

➡ Exemplo de uma transação num procedimento

```
DELIMITER $$
CREATE PROCEDURE InserirFarmaco ( IN codigo INT, IN farmaco VARCHAR(45), IN desc VARCHAR(150), OUT res VARCHAR(100)
BEGIN
DECLARE erro INT DEFAULT 0;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET erro=1;
START TRANSACTION;
INSERT INTO Farmacos(id_farmaco,nome,descricao) VALUES(codigo,farmaco,desc);
SELECT COUNT(*) FROM Farmacos WHERE id_farmaco = codigo;
IF erro = 1 THEN
    ROLLBACK;
    SET res = 'Transação abortada - Cabeçalho de Venda.';
    LEAVE InserirFarmaco;
    END IF;
END$$
DELIMITER ;
```