FUNCTIONS - TRIGGERS

ANTONIO DAVID VINISKI
antonio.david@pucpr.br
PUCPR

FUNCTIONS

INTRODUÇÃO

FUNCTIONS

- Assim como as ***procedures***, uma função é um tipo de rotina armazenada no banco de dados. Uma função é utilizada para gerar um valor que pode ser usado em outras instruções.
    - Esse valor é geralmente baseado em um ou mais parâmetros fornecidos à função.
- As funções são executadas geralmente como parte de uma expressão (chamadas em uma instrução **SELECT** por exemplo).
- O MySQL possui diversas funções internas que o desenvolvedor pode utilizar (como mostradas anteriormente: operações com *String* e Data), e também permite que criemos nossas próprias funções.
- Uma função é considerada "determinística" se ela sempre retorna o mesmo resultado para os mesmos parâmetros de entrada, e "não determinística" caso contrário.

CRIANDO UMA FUNÇÃO

FUNCTIONS

```
DELIMITER $$
CREATE FUNCTION nome_funcao (parâmetros)
RETURNS retorna_valor [NOT] DETERMINISTIC
BEGIN
    /* CORPO DA FUNÇÃO */
END $$
DELIMITER ;
```

Especifica os parâmetros que a função recebe. Diferente da ***Stored Procedure***, a função não especifica o **MODO** do parâmetro, somente o "nome" e o "**TIPO**"

Especifica o "**TIPO**" do retorno da função definida

UTILIZANDO UMA FUNÇÃO

FUNCTIONS

- As funções são utilizadas em conjunto com outras instruções.
  - **SELECT** nome_funcao(parâmetros);
  - **INSERT INTO** tabela **VALUES**(nome_funcao(parâmetros));
  - **UPDATE** tabela **SET** campo = (nome_funcao(parâmetros));

EXEMPLO FUNÇÃO

FUNCTIONS

- Criar um função para retornar o dia da semana (em português) dado um número inteiro passado como parâmetro.

PROCEDURE - EXEMPLO

```
DELIMITER $$
CREATE FUNCTION weektext(day_of_week INT)
RETURNS VARCHAR(30) DETERMINISTIC
BEGIN
        DECLARE tex_week_day VARCHAR(20);
        SET tex_week_day := (SELECT  CASE WHEN day_of_week = 1 THEN "Domingo"
                                          WHEN day_of_week = 2 THEN "Segunda"
                                          WHEN day_of_week = 3 THEN "Terça"
                                          WHEN day_of_week = 4 THEN "Quarta"
                                          WHEN day_of_week = 5 THEN "Quinta"
                                          WHEN day_of_week = 6 THEN "Sexta"
                                          ELSE "Sábado"
                                    END);
        RETURN  tex_week_day;
END$$
```

EXERCÍCIOS IV

PROCEDURES

Crie funções para:

1. Contar o número de comandas de um cliente.
2. Retornar o valor a receber de comissão de um garçom.
3. Retornar a quantidade vendida de um produto.
4. Retornar o faturamento total de um produto.
5. Retornar o número de registros efetuados por um garçom.

TRIGGERS

INTRODUÇÃO

- É muito comum, em aplicações que utilizam bancos de dados, que ações sejam disparadas em resposta ou como consequência de outras, realizando diversas operações
    - Cálculos, validações e alterações na base de dados.
- Muitas vezes os programadores de banco de dados optam por executarem tais ações na própria aplicação
    - Várias instruções SQL em sequência são executadas para obter o resultado esperado.
    - É uma solução que pode até ser tida como mais segura.
    - Tende a tornar ainda mais "pesada" a execução de certas tarefas, requisitando mais recursos da máquina cliente.
- Uma alternativa a execução sequencial na máquina cliente é a utilização de **TRIGGERS**.

O QUE SÃO TRIGGERS?

- **Triggers** (“gatilhos” em português) são objetos do banco de dados que, relacionados a certa tabela, permitem a realização de processamentos em consequência de uma determinada ação como, por exemplo, a inserção de um registro.

VANTAGENS E DESVANTAGENS

- Vantagens:
  - Fornecem outra maneira de verificar a integridade dos dados e tratar possíveis erros.
  - Parte do processamento que seria executado na aplicação passa para o banco, poupando recursos da máquina cliente.
  - Facilita a manutenção, sem que seja necessário alterar o código fonte da aplicação.
  - Fornecem uma maneira alternativa de executar tarefas agendadas e auditar dados.
- Desvantagens:
  - Alguém que tenha acesso não autorizado ao banco de dados poderá visualizar e alterar o processamento realizado pelos gatilhos.
  - Requer maior conhecimento de manipulação do banco de dados (SQL) para realizar as operações internamente.

CRIAR TRIGGER

- CREATE [DEFINER = *user*] TRIGGER *trigger_name*

  trigger_time trigger_event ON *tbl_name*

  FOR EACH ROW

  [*trigger_order*]

  trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } *other_trigger_name*

CRIAR TRIGGER I

- **trigger_time** indica quando a acão será executada. Os valores possíveis são **BEFORE** e **AFTER** e indicam que a trigger será executada ANTES ou DEPOIS do evento disparado em cada registro sendo modificado.
- **trigger_event** indica o tipo de evento que ativa a **TRIGGER**. Os valores permitidos são:
  - **INSERT**: A trigger é ativada toda vez que um novo registro for inserido no banco de dados.
  - **UPDATE**: A trigger é ativada toda vez que um registro é modificado.
  - **DELETE**: A trigger é ativada toda vez que um registro é excluído da tabela.

CRIAR TRIGGER II

- É possível definir várias **TRIGGERS** para uma determinada tabela que tenham o mesmo evento de gatilho e tempo de ação.
- Por exemplo
  - Você pode ter dois gatilhos **BEFORE UPDATE** para uma tabela.
  - Por padrão, as TRIGGERS que têm o mesmo evento de gatilho e tempo de ação são ativados na ordem em que foram criados.
  - Para afetar a ordem do acionador, especifique uma cláusula **trigger_order** que indique **FOLLOWS** ou **PRECEDES** e o nome de um acionador existente que também tenha o mesmo evento de acionamento e tempo de ação.
    - Com **FOLLOWS**, o novo trigger é ativado após o trigger existente.
    - Com **PRECEDES**, o novo gatilho é ativado antes do gatilho existente.

CRIAR TRIGGER III

- **trigger_body** é a instrução a ser executada quando o gatilho é ativado.
  - Para executar várias instruções, use a construção de instrução composta **BEGIN** ... **END**.
  - Isso também permite que você use as mesmas instruções que são permitidas nas rotinas armazenadas (**PROCEDURES** e **FUNCTIONS**).
- Para realizar as respectivas operações, você pode fazer referência a colunas na tabela associada ao gatilho usando os *aliases* **OLD** e **NEW**.
  - **OLD**.col_name refere-se a uma coluna de uma linha existente antes de ser atualizada ou excluída.
  - **NEW**.col_name refere-se à coluna de uma nova linha a ser inserida ou a uma linha existente após ser atualizada.

EXEMPLO RESTAURANTE

- Criar **triggers** para atualizar as tabelas **comanda** e **produto** toda vez que alguma modificação (inserção, exclusão ou atualização) for realizada na tabela registro:
  - Caso uma instância na tabela registro seja inserida, removida ou alterada, precisamos atualizar o estoque disponível na tabela **produto** também é preciso atualizar o valor total e número total de itens da tabela **comanda**.