Triggers e Procedures

Objetivos:

- *Criar e utilizar triggers e procedures utilizando linguagem SQL;*

Ferramentas necessárias:

Browser, Xampp, WampServer, EasyPHP, Notepad++, Sublime Text, MySQLWorkBench, SGBD MySQL, Bibliografias de livros da faculdade.

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.*
*Os **triggers** podem ser executados ANTES ou DEPOIS das operações de **INSERT**, **UPDATE** e **DELETE** de registros.*
Observação: O suporte a triggers foi incluído na versão 5.0.2 do MySQL.

Prós e Contras

*Os principais pontos positivos sobre os **triggers** são:*

- *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.*

Já contra sua utilização existem as seguintes considerações:

- *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.*

*A seguir é explicado o processo de criação de **triggers**, a sintaxe utilizada e o significado de cada instrução.*

Sintaxe

A sintaxe dos comandos para criar um novo trigger no MySQL é a seguinte:
***Listagem 1:** Sintaxe para criação de trigger*

```
1. CREATE TRIGGER nome momento evento
2. ON tabela
3. FOR EACH ROW
4. BEGIN
5. /*corpo do código*/
6. END
```

Onde se tem os seguintes parâmetros:

- ***nome**: nome do gatilho, segue as mesmas regras de nomeação dos demais objetos do banco.*
- ***momento**: quando o gatilho será executado. Os valores válidos são BEFORE (antes) e AFTER (depois).*
- ***evento**: evento que vai disparar o gatilho. Os valores possíveis são **INSERT**, **UPDATE** e **DELETE**. Vale salientar que os comandos **LOAD DATA** e **REPLACE** também disparam os eventos de inserção e exclusão de registros, com isso, os gatilhos também são executados.*
- ***tabela**: nome da tabela a qual o gatilho está associado.*

*Não é possível criar mais de um trigger para o mesmo evento e momento de execução na mesma tabela. Por exemplo, não se pode criar dois gatilhos **AFTER INSERT** na mesma tabela.*

Os registros NEW e OLD

Como os triggers, são executados em conjunto com operações de inclusão e exclusão, é necessário poder acessar os registros que estão sendo incluídos ou removidos. Isso pode ser feito através das palavras NEW e OLD.
Em gatilhos executados após a inserção de registros, a palavra reservada NEW dá acesso ao novo registro. Pode-se acessar as colunas da tabela como atributo do registro NEW, como veremos nos exemplos.
O operador OLD funciona de forma semelhante, porém em gatilhos que são executados com a exclusão de dados, o OLD dá acesso ao registro que está sendo removido.

Eliminando Triggers

A sintaxe dos comandos para excluir um trigger no MySQL é a seguinte:

```
DROP TRIGGER nome;
```

Utilização de trigger

Para exemplificar e tornar mais clara a utilização de gatilhos, simularemos a seguinte situação: um mercado que, ao realizar vendas, precisa que o estoque dos produtos seja automaticamente reduzido. A devolução do estoque deve também ser automática no caso de remoção de produtos da venda.
Como se trata de um ambiente hipotético, teremos apenas quatro tabelas de estrutura simples, cujo script de criação será mostrado neste roteiro.

*Criar tabelas, **triggers** e **procedures** para controlar o estoque de uma pequena loja:*
***produto** (id, status, descrição, estoqueminimo, estoquemaximo)*
***produtoEntrada** (id, idProduto, qtd, vlrUnitario, entradaData)*
***estoque** (id, idProduto, qtd, vlrUnitario)*
***produtoSaida** (id, idProduto, qtd, saidaData, vlrUnitario)*

*Pode-se criar um trigger para a operação de INSERT nas tabelas **produtoEntrada** e **produtoSaida**, que atualiza a quantidade do produto na tabela **estoque**.*

*Tabela: **produto***

- *Armazena informações básicas sobre os produtos que a loja vende.*

```
CREATE TABLE produto (
  id INT(11) NOT NULL AUTO_INCREMENT,
  status CHAR(1) NOT NULL DEFAULT 'A', -- Indica se o cadastro está ativo "A" ou inativo "I"
  descricao VARCHAR(50),
  estoqueMinimo INT(11),
  estoqueMaximo INT(11),
  PRIMARY KEY (id)
);
```

- *Cadastrando/Inserindo produtos na tabela.*

```
INSERT INTO produto (descricao, estoqueMinimo, estoqueMaximo) VALUES
('PENDRIVE', 10, 100),
('MOUSE', 10, 100),
('IOGURTE', 5, 50),
('TEQUILA', 5, 40),
('PRESUNTO', 5, 20);
```

*Tabela: **produtoEntrada** (compra)*

- *Armazena as compras de produtos efetuadas para loja.*
- *Obs.: através de **triggers** serão controladas inserções na tabela "**estoque**"*

```
CREATE TABLE produtoEntrada (
  id INT(11) NOT NULL AUTO_INCREMENT,
  idProduto INT(11),
  qtd INT(11),
  vlrUnitario DECIMAL(9,2) NULL DEFAULT '0.00',
  entradaData DATE,
  PRIMARY KEY (id)
);
```

*Tabela: **estoque***

- *Recebe os dados conforme as ações executadas nas tabelas "**produtoEntrada**" e "**produtoSaida**".*
- *O usuário não tem interação direta como INSERÇÕES, UPDATES e EXCLUSÕES,*
- *A tabela "**estoque**" armazena somente o resultado das ações de compra e venda de produtos.*

```
CREATE TABLE estoque (
  id INT(11) NOT NULL AUTO_INCREMENT,
  idProduto INT(11),
  qtd INT(11),
  vlrUnitario DECIMAL(9,2) NULL DEFAULT '0.00',
  PRIMARY KEY (id)
);
```

*Tabela: **produtoSaida** (venda)*

- *Nessa tabela serão gravadas todas as saídas (Vendas) de produtos.*
- *Obs.: através de **triggers** essas ações serão refletidas na tabela de "**estoque**"*

```
CREATE TABLE produtoSaida (
  id INT(11) NOT NULL AUTO_INCREMENT,
  idProduto INT(11),
  qtd INT(11),
  saidaData DATE,
  vlrUnitario DECIMAL(9,2) NULL DEFAULT '0.00',
  PRIMARY KEY (id));
```

*Procedure: "**SP_AtualizaEstoque**"*

- *Procedure para atualizar os estoques na tabela de "**estoque**".*
- *Nas quatro tabelas criadas existem dois campos em comum "**idProduto**" e "**qtd**", são estes campos que servirão como parâmetros para inserção e baixa de estoque nas procedures.*
- *Recebe três parâmetros (**var_idProduto**, **var_qtdComprada**, **var_vlrUnitario**) e tem a finalidade de inserir ou debitar produtos na tabela de "**estoque**" de acordo com o os parâmetros que são passados.*

```
DELIMITER $$
CREATE PROCEDURE SP_AtualizaEstoque(var_idProduto INT, var_qtdComprada INT, var_vlrUnitario
DECIMAL(9,2))
BEGIN
    DECLARE var_contador INT(11);
    SELECT COUNT(*) INTO var_contador FROM estoque WHERE idProduto = var_idProduto;
    IF var_contador > 0 THEN
        UPDATE estoque SET qtd=qtd + var_qtdComprada, vlrUnitario= var_vlrUnitario
        WHERE idProduto = var_idProduto;
    ELSE
        INSERT INTO estoque (idProduto, qtd, vlrUnitario) VALUES (var_idProduto, var_qtdComprada,
var_vlrUnitario);
    END IF;
END $$
DELIMITER ;
```

***Obs**.:*

- *Foi declarada uma variável **var_contador** para receber o valor da instrução SELECT COUNT(*)*
- *Caso exista um **produto** cadastrado no **estoque** com o mesmo **var_idProduto** passado como parâmetro, então será inserido na variável **var_contador** o número de linhas que atendem a essa condição.*
- *Posteriormente verifica-se o valor de **var_contador**, se for maior que 0 então executa-se um UPDATE na tabela "**estoque**", senão é feito um "INSERT".*
- *Essa verificação pode ser feita de diversas maneiras, o programador pode implementar da melhor maneira possível.*

Triggers – no MySQL
*Será criado um trigger para cada evento das tabelas **produtoEntrada** e **produtoSaida***
***Nota**: o MySQL não suporta múltiplos eventos em uma mesma trigger*

Serão criadas as seguintes triggers:

- *trg_produtoEntrada_AI*
- *trg_produtoEntrada_AU*
- *trg_produtoEntrada_AD*
- *trg_produtoSaida_AI*
- *trg_produtoSaida_AU*
- *trg_produtoSaida_AD*

*TRIGGER **trg_entradaProduto_AI***

- *Essa trigger será disparada após a inserção de um registro na tabela de "**produtoEntrada**".*

```
DELIMITER $$
CREATE TRIGGER trg_produtoEntrada_AI AFTER INSERT
ON produtoEntrada
FOR EACH ROW
BEGIN
  CALL sp_atualizaEstoque (new.idProduto,
                           new.qtd,
                           new.vlrUnitario);
END $$
DELIMITER ;
```

*TRIGGER **trg_produtoEntrada_AU***

- *Essa trigger será disparada após a atualização de um registro na tabela de "**produtoEntrada**".*

```
TRIGGER trg_produtoEntrada_AU
DELIMITER $$
CREATE TRIGGER trg_produtoEntrada_AU AFTER UPDATE
ON produtoEntrada
FOR EACH ROW
BEGIN
  CALL sp_atualizaEstoque (new.idProduto,
                           new.qtd – old.qtd,
                           new.vlrUnitario);
END $$
DELIMITER ;
```

*TRIGGER **trg_produtoEntrada_AD***

- *Essa trigger será disparada após a exclusão de um registro na tabela de "**produtoEntrada**".*

```
TRIGGER trg_produtoEntrada_AD
DELIMITER $$
CREATE TRIGGER trg_produtoEntrada_AD AFTER DELETE
ON produtoEntrada
FOR EACH ROW
BEGIN
  CALL sp_atualizaEstoque (old.idProduto,
                           old.qtd – old.qtd,
                           old.vlrUnitario);
END $$
DELIMITER ;
```

*TRIGGER **trg_produtoSaida_AI***

- *Essa trigger será disparada após a inserção de um registro na tabela de "**produtoSaida**".*

```
TRIGGER trg_produtoSaida_AI
DELIMITER $$
CREATE TRIGGER trg_produtoSaida_AI AFTER INSERT
ON produtoSaida
FOR EACH ROW
BEGIN
  CALL sp_atualizaEstoque (new.idProduto,
                           new.qtd * –1,
                           new.vlrUnitario);
END $$
DELIMITER ;
```

*TRIGGER **trg_produtoSaida_AU***

- *Essa trigger será disparada após a atualização de um registro na tabela "**produtoSaida**".*

```
TRIGGER trg_produtoSaida_AU
DELIMITER $$
CREATE TRIGGER trg_produtoSaida_AU AFTER UPDATE
ON produtoSaida
FOR EACH ROW
BEGIN
  CALL sp_atualizaEstoque (new.idProduto,
                           old.qtd * –new.qtd,
                           new.vlrUnitario);
END $$
DELIMITER ;
```

*TRIGGER **trg_produtoSaida_AD***
*Essa trigger será disparada após a exclusão de um registro na tabela de "**produtoSaida**".*

*TRIGGER **trg_produtoSaida_AD***

```
DELIMITER $$
CREATE TRIGGER trg_produtoSaida_AD AFTER DELETE
ON produtoSaida
FOR EACH ROW
BEGIN
  CALL sp_atualizaEstoque (old.idProduto,
                           old.qtd,
                           old.vlrUnitario);
END $$
DELIMITER ;
```

***Observações**:*
*Algumas chamadas da procedure "**SP_AtualizaEstoque**", antes de passar o parâmetro "**qtd**" multiplicam esse valor por **-1**, essa operação muda o sinal matemático do valor para negativo.*
Dentro da procedure somamos as quantidades.
Quando passamos o sinal negativo ocorre uma subtração dos valores resultando em débito no estoque.

Fazendo as contas
Se somarmos a quantidade em estoque com a quantidade vendida de um determinado produto, vamos obter a quantidade comprada.
Exemplo:
Foram comprados 10 PENDRIVEs;
No estoque constam 5 PENDRIVEs;
Foram vendidos 5 PENDRIVEs;
5 no estoque + 5 vendidos = 10 comprados.

Conclusão

Em ambientes reais, triggers podem ser utilizados para operações mais complexas, por exemplo, antes de vender um item, verificar se há estoque disponível e só então permitir a saída do produto.

Fonte

***Site Linha de Código** http://www.linhadecodigo.com.br*

Tarefas

1) *Crie um banco de dados (sobre um tema à sua escolha) contendo pelo menos três tabelas relacionadas. O banco deve ser "populado" (as tabelas devem possuir registros). Para tanto, crie scripts de inserção para todas as tabelas.*
2) *Crie pelo menos dois gatilhos (TRIGGERS) em pelo menos uma das tabelas do banco.*
3) *Teste os gatilhos criados com INSERT, DELETE e/ou UPDATE (de acordo com o propósito dos gatilhos).*
4) *Faça consultas para se certificar que os gatilhos estão funcionando.*