

Gatilhos (Triggers)

Introdução

- ▶ Um TRIGGER ou gatilho é um objeto de banco de dados, associado a uma tabela, definido para ser disparado, respondendo a um evento em particular.
- ▶ Tais eventos são os comandos da DML (Data Manipulation Language): INSERT, REPLACE, DELETE ou UPDATE.
- ▶ Podemos definir inúmeros TRIGGERS em um banco de dados baseados diretamente em qual dos comandos acima irá dispará-lo, sendo que, para cada um, podemos definir apenas um TRIGGER.
- ▶ Os TRIGGERS poderão ser disparados para trabalharem antes ou depois do evento.

Exemplo

C:\WINDOWS\system32\cmd.exe - mysql -u root -p test

```
mysql> CREATE TABLE tbl_cliente (  
-> cliente_id int unsigned auto_increment primary key,  
-> cliente_nome char(80) not null,  
-> cliente_email char(80) not null,  
-> dt_cadastro timestamp default current_timestamp  
-> ) Engine =INNODB;
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> _
```

Exemplo

- ▶ Baseado na tabela **tbl_cliente**, podemos definir os TRIGGERS para serem disparados antes (BEFORE) ou depois (AFTER) de um INSERT.

Sintaxe

```
CREATE  
  [DEFINER = { user | CURRENT_USER }]  
  TRIGGER trigger_name trigger_time  
  trigger_event  
  ON tbl_name FOR EACH ROW trigger_stmt
```


Sintaxe

- ▶ **DEFINER:** Quando o TRIGGER for disparado, esta opção vai checar com quais privilégios será disparado. Utilizará os privilégios do usuário informado em user ('usuario'@'localhost') ou os privilégios do usuário atual (CURRENT_USER). Caso essa sentença seja omitida da criação do TRIGGER, o valor padrão desta opção é CURRENT_USER();
- ▶ **trigger_name:** define o nome do gatilho;
- ▶ **trigger_time:** define se o TRIGGER será ativado antes (BEFORE) ou depois (AFTER) do comando que o disparou

Sintaxe

- ▶ trigger_event: define qual será o evento, INSERT, REPLACE, DELETE ou UPDATE;
- ▶ tbl_name: nome da tabela onde o TRIGGER ficará “pendurado” aguardando o trigger_event;
- ▶ trigger_stmt: as definições do que o TRIGGER deverá fazer quando for disparado;

Definir dados de antes (OLD) e depois (NEW)

- ▶ Em meio aos TRIGGERS temos dois operadores que nos possibilitam acessar as colunas da tabela alvo do comando DML
- ▶ Podemos acessar os valores que serão enviados para a tabela `tbl_cliente` antes (BEFORE) ou depois (AFTER) de um UPDATE, por exemplo. Tais operadores nos permitirão então, ter dois momentos, o antes e o depois e também examinar os valores para que sejam ou não inseridos, atualizados ou excluídos da tabela.

Definir dados de antes (OLD) e depois (NEW)

- ▶ INSERT: o operador `NEW.nome_coluna`, nos permite verificar o valor enviado para ser inserido em uma coluna de uma tabela.
 - ▶ `OLD.nome_coluna` não está disponível.
- ▶ DELETE: o operador `OLD.nome_coluna` nos permite verificar o valor excluído ou a ser excluído.
 - ▶ `NEW.nome_coluna` não está disponível.
- ▶ UPDATE: tanto `OLD.nome_coluna` quanto `NEW.nome_coluna` estão disponíveis, antes (BEFORE) ou depois (AFTER) da atualização de uma linha.

Definir dados de antes (OLD) e depois (NEW)

- ▶ Percebemos então que, ao inserir uma nova linha em uma tabela, temos os valores das colunas disponível através do operador NEW.nome_coluna
- ▶ Quando excluimos uma linha, temos ainda os valores das colunas da linha excluída através do operador OLD.nome_coluna
- ▶ Temos os dois operadores disponíveis no UPDATE pois consiste em um DELETE seguido por um INSERT.

Criar um primeiro TRIGGER

- ▶ Objetiva validar se os dados foram passados em uma declaração INSERT antes (BEFORE) que sejam cadastrados na tabela de exemplo.
- ▶ Valida o nome com quantidade de caracteres maior ou igual a 4 (quatro).

Criar um primeiro TRIGGER

```
C:\WINDOWS\system32\cmd.exe - mysql -u root -p test

mysql> delimiter //
mysql> CREATE TRIGGER trg_1 BEFORE INSERT ON tbl_cliente
-> FOR EACH ROW
-> BEGIN
-> -- utilizando user variables
-> set @none = NEW.cliente_nome;
-> -- validando
-> IF ((CHAR_LENGTH(@none) <= 4) OR (@none = '')) THEN
-> SET NEW.cliente_nome =NULL;
-> END IF;
-> END//
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Criar um primeiro TRIGGER

- ▶ Ao tentarmos inserir um valor cujo número de caracteres é menor ou igual a 0 ou nada, o TRIGGER será disparado e setará o valor enviado para NULL através do operador NEW.nome_coluna.
- ▶ Como na tabela de exemplo a coluna cliente_nome foi configurada com a restrição NOT NULL, ou seja, não aceitará valores nulos, uma mensagem de erro será enviada e o INSERT falhará

Outro Exemplo

```
CREATE TABLE Produtos
(
  Referencia VARCHAR(3) PRIMARY KEY,
  Descricao VARCHAR(50) UNIQUE,
  Estoque INT NOT NULL DEFAULT 0
);
INSERT INTO Produtos VALUES ('001', 'Feijão', 10);
INSERT INTO Produtos VALUES ('002', 'Arroz', 5);
INSERT INTO Produtos VALUES ('003', 'Farinha', 15);
```


Outro Exemplo

```
CREATE TABLE ItensVenda  
(  
    Venda    INT,  
    Produto VARCHAR(3),  
    Quantidade INT  
);
```

Outro Exemplo

- ▶ Ao inserir e remover registro da tabela ItensVenda, o estoque do produto referenciado deve ser alterado na tabela Produtos.
- ▶ Para isso, serão criados dois triggers: um *AFTER INSERT* para dar baixa no estoque e um *AFTER DELETE* para fazer a devolução da quantidade do produto.

Outro Exemplo

```
DELIMITER $$
```

```
CREATE TRIGGER Tgr_ItensVenda_Insert AFTER INSERT
```

```
ON ItensVenda
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    UPDATE Produtos SET Estoque = Estoque - NEW.Quantidade
```

```
WHERE Referencia = NEW.Produto;
```

```
END$$
```

Outro Exemplo

```
CREATE TRIGGER Tgr_ItensVenda_Delete AFTER DELETE
ON ItensVenda
FOR EACH ROW
BEGIN
    UPDATE Produtos SET Estoque = Estoque + OLD.Quantidade
    WHERE Referencia = OLD.Produto;
END$$

DELIMITER ;
```


Outro Exemplo


- ▶ No primeiro gatilho, foi utilizado o registro NEW para obter as informações da linha que está sendo inserida na tabela.
- ▶ O mesmo é feito no segundo gatilho, onde se obtém os dados que estão sendo apagados da tabela através do registro OLD.
- ▶ Tendo criado os **triggers**, podemos testá-los inserindo dados na tabela ItensVenda. Nesse caso, vamos simular uma venda de número 1 que ontem três unidades do produto 001, uma unidade do produto 002 e cinco unidades do produto 003.

Outro Exemplo

```
INSERT INTO ItensVenda VALUES (1, '001',3);  
INSERT INTO ItensVenda VALUES (1, '002',1);  
INSERT INTO ItensVenda VALUES (1, '003',5);
```


Outro Exemplo

Filter:	<input type="text"/>		Edit:
	Referencia	Descricao	Estoque
▶	001	Feijão	10
	002	Arroz	5
	003	Farinha	15
*	NULL	NULL	NULL

Filter:	<input type="text"/>		Edit:
	Referencia	Descricao	Estoque
▶	001	Feijão	7
	002	Arroz	4
	003	Farinha	10
*	NULL	NULL	NULL

Outro Exemplo

- ▶ Nota-se que o estoque dos produtos foi corretamente reduzido, de acordo com as quantidades “vendidas”.
- ▶ Agora para testar o **trigger** da exclusão, removeremos o produto 001 dos itens vendidos. Com isso, o seu estoque deve ser alterado para o valor inicial, ou seja, 10.
- ▶ `DELETE FROM ItensVenda WHERE Venda = 1 AND Produto = '001';`
- ▶ Executando novamente um select na tabela Produtos, veremos que apenas o produto 001 teve o estoque atualizado, voltando a 10,

Outro Exemplo

Filter:			
	Referencia	Descricao	Estoque
▶	001	Feijão	7
	002	Arroz	4
	003	Farinha	10
*	NULL	NULL	NULL

Filter:			
	Referencia	Descricao	Estoque
▶	001	Feijão	10
	002	Arroz	4
	003	Farinha	10
*	NULL	NULL	NULL

Exemplo 3

```
CREATE TABLE produtos (  
  ID int NOT NULL primary key AUTO_INCREMENT,  
  NOME varchar(255) NOT NULL,  
  VALOR decimal(10,2) NOT NULL,  
);
```

- O preço desses produtos sofrerão reajustes ao longo do tempo e deverão ser mantidos históricos para a possibilidade da emissão de relatórios futuros.

Exemplo 3

```
CREATE TABLE produtos_historico (  
  ID int NOT NULL,  
  DATA_HORA datetime NOT NULL,  
  VALOR decimal(10,2) NOT NULL,  
  PRIMARY KEY (ID,DATA_HORA)  
)
```

- Mas e agora, como faço para que após toda atualização na tabela de produtos seja salvo o valor antigo daquele produto?

Exemplo 3

```
DELIMITER $$  
CREATE TRIGGER salvaValorProduto AFTER UPDATE ON produtos  
FOR EACH ROW BEGIN  
INSERT INTO produtos_historico SET ID = OLD.ID,  
DATA_HORA = NOW(), VALOR = OLD.VALOR;  
END;  
$$  
DELIMITER ;
```


Exemplo 3

- ▶ Quando o valor de qualquer produto for atualizado, automaticamente o banco de dados irá executar um INSERT na tabela de histórico armazenando o ID do produto a DATA e HORA em que o produto foi atualizado e o VALOR do produto antes do reajuste.
- ▶ Dessa forma poderemos consultar a variação daquele produto ao longo do tempo.
- ▶ Nesse trigger foi utilizado o AFTER ou seja, o gatilho será disparado DEPOIS que o comando UPDATE for executado na tabela 'produtos'
- ▶ Foi utilizado o operado OLD.VALOR para recuperarmos o valor do produto antes da alteração de reajuste.

O que faz este gatilho?

```
CREATE TABLE people (age INT, name varchar(150));
```

```
delimiter //
```

```
CREATE TRIGGER agecheck BEFORE INSERT ON people
```

```
FOR EACH ROW
```

```
IF NEW.age < 0 THEN
```

```
SET NEW.age = 0;
```

```
END IF; //
```

```
delimiter ;
```

```
INSERT INTO people VALUES (-20, 'Sid'), (30, 'Josh');
```

Exemplo 4

```
CREATE TABLE blog (  
    id mediumint(8) unsigned NOT NULL AUTO_INCREMENT,  
    title text,  
    content text,  
    deleted tinyint(1) unsigned NOT NULL DEFAULT '0',  
    PRIMARY KEY (`id`),  
    KEY ix_deleted (deleted)  
);
```

Exemplo 4

```
CREATE TABLE audit (  
    id mediumint(8) unsigned NOT NULL AUTO_INCREMENT,  
    blog_id mediumint(8) unsigned NOT NULL,  
    changetype enum('NEW','EDIT','DELETE') NOT NULL,  
    changetime timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
    PRIMARY KEY (id),  
    KEY ix_blog_id (blog_id),  
    KEY ix_changetype (changetype),  
    KEY ix_changetime (changetime),  
    CONSTRAINT FK_audit_blog_id FOREIGN KEY (blog_id) REFERENCES blog (id) ON DELETE CASCADE ON  
    UPDATE CASCADE  
);
```

Exemplo 4

```
DELIMITER $$
CREATE TRIGGER `blog_after_insert` AFTER INSERT ON `blog`
  FOR EACH ROW BEGIN
  IF NEW.deleted THEN
    SET @changetype = 'DELETE';
  ELSE
    SET @changetype = 'NEW';
  END IF;
  INSERT INTO audit (blog_id, changetype) VALUES (NEW.id, @changetype);
END$$
DELIMITER ;
```

Exemplo 4

```
DELIMITER $$
CREATE TRIGGER `blog_after_update` AFTER UPDATE ON `blog`
  FOR EACH ROW BEGIN
  IF NEW.deleted THEN
    SET @changetype = 'DELETE';
  ELSE
    SET @changetype = 'EDIT';
  END IF;
  INSERT INTO audit (blog_id, changetype) VALUES (NEW.id, @changetype);
END$$
DELIMITER ;
```


Exemplo 4

- ▶ INSERT INTO blog (title, content) VALUES ('Article One', 'Initial text.');

blog			
id	title	content	deleted
1	Article One	Initial text	0

audit			
id	blog_id	changetype	changetime
1	1	NEW	2011-05-20 09:00:00

Exemplo 4

- UPDATE blog SET content = 'Edited text' WHERE id = 1;

blog			
id	title	content	deleted
1	Article One	Edited text	0

audit			
id	blog_id	changetype	changetime
1	1	NEW	2011-05-20 09:00:00
2	1	EDIT	2011-05-20 09:01:00

Exemplo 4

- UPDATE blog SET deleted = 1 WHERE id = 1;

blog			
id	title	content	deleted
1	Article One	Edited text	1

audit			
id	blog_id	changetype	changetime
1	1	NEW	2011-05-20 09:00:00
2	1	EDIT	2011-05-20 09:01:00
3	1	DELETE	2011-05-20 09:03:00

Exercício 1

- ▶ Imagine uma tabela com os seguintes campos:
 - ▶ Nome (varchar(255))
 - ▶ Trim1 (decimal(3,1))
 - ▶ Trim2 (decimal(3,1))
 - ▶ Trim3 (decimal(3,1))
 - ▶ Status (aprovado ou reprovado)
- ▶ Ao cadastrar um aluno e as notas, calcule se o aluno está aprovado ou reprovado (<6.0 reprovado)
- ▶ Ao atualizar as notas, calcule novamente e atualize o status

Exercício 2

- ▶ Mesma situação do exercício 1, porém, com Rec1Trim1, Rec2Trim1,NF1, Rec1Trim2 , Rec2Trim2, NF2, Rec1Trim3, Rec2Trim3, NF3

Material copiado de

- ▶ <http://www.linhadecodigo.com.br/artigo/3567/mysql-basico-triggers.aspx>
- ▶ <http://www.devmedia.com.br/mysql-triggers/8088>
- ▶ <http://www.conexaototal.com.br/triggers-mysql/>
- ▶ http://kb.siteground.com/what_are_mysql_triggers_and_how_to_use_them
- ▶ <http://www.sitepoint.com/how-to-create-mysql-triggers/>