15 – Introdução

- O recurso denominado gatilho ou disparador é normalmente referenciado pelo seu termo em inglês: **Trigger**.

- Uma trigger é uma rotina semelhante a uma stored procedure, tendo como diferencial o fato de ser executada automaticamente quando um determinado evento ocorre. Uma trigger não pode ser chamada pelo comando **CALL**.

- Uma triggers é um poderoso mecanismo para assegurar a integridade dos dados, bem como um meio útil de automatizar algumas operações no banco de dados, tais como registros de auditoria, validações, atualizações em tabelas, etc.

15.1 – Sintaxe de uma Trigger

- O quadro abaixo mostra a estrutura de uma trigger:

```
DELIMITER $
CREATE TRIGGER <nome da trigger> <momento> <evento> ON <tabela>
FOR EACH ROW
BEGIN
  <corpo da trigger>
END$
DELIMITER ;
```

15.1 – Sintaxe de uma Trigger

- No esquema anterior, o <nome da trigger> é o identificador usado para referenciá-la caso seja necessário.

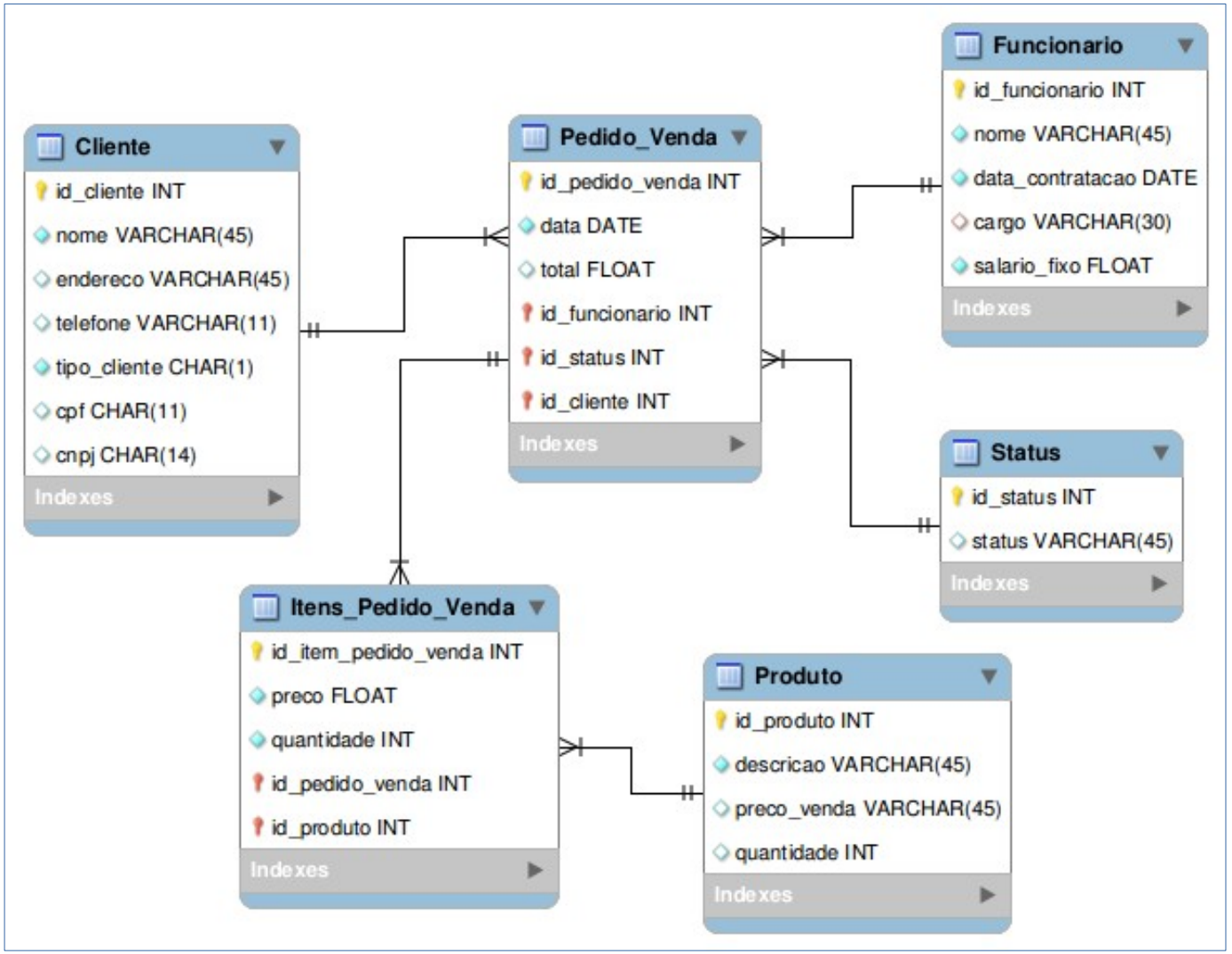
- O <momento> é o instante em que a trigger será executada, existindo duas possibilidades: **AFTER** (depois de ocorrer o evento) e **BEFORE** (antes de ocorrer o assunto).

- O <evento> é a situação que deve ocorrer para que a trigger seja executada. Existem três possibilidades possíveis, como ilustra a tabela abaixo:

Evento	Descrição
INSERT	Quando ocorrer uma inserção de registro
DELETE	Quando ocorrer uma deleção de registro
UPDATE	Quando ocorrer uma alteração de registro

15.1 – Sintaxe de uma Trigger

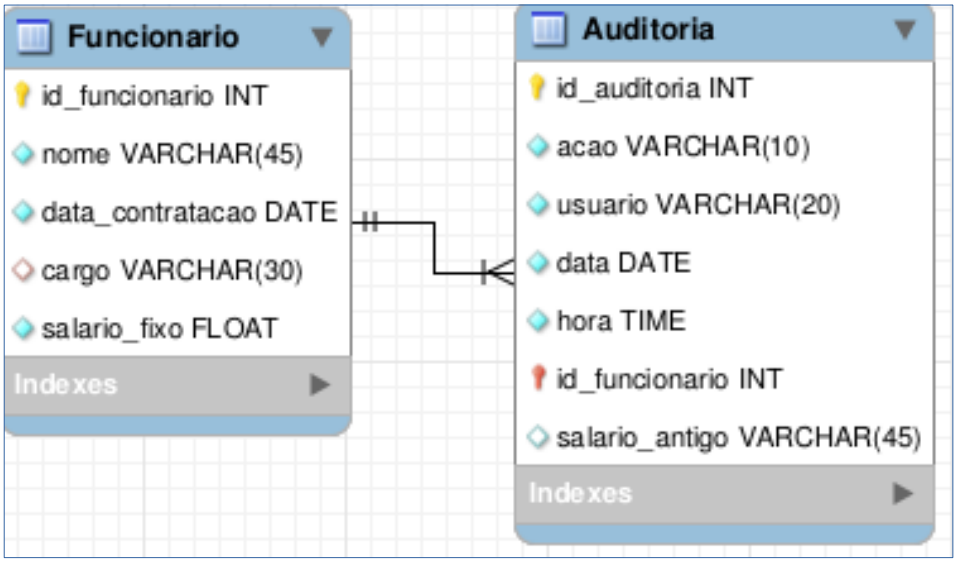
- O modelo utilizado nos exemplos é o banco de dados de vendas mostrado na figura abaixo:



15.2 – Trigger de Auditoria

- Para exemplificar o uso de triggers em situações de auditoria, é possível utilizar as duas tabelas mostradas na figura abaixo.

- A ideia consiste em monitorar e registrar em uma tabela de auditoria todas as operações de escrita realizadas na tabela de funcionários.



15.2 – Trigger de Auditoria

- A trigger abaixo será executada automaticamente logo após (AFTER) um novo registro na tabela de funcionário ser inserido por meio do comando INSERT. Esta trigger irá inserir na tabela auditoria os dados de controle sobre o novo registro inserido na tabela de funcionario.

```
DELIMITER $
CREATE TRIGGER auditoria_funcionario_insert AFTER INSERT ON funcionario
FOR EACH ROW
 BEGIN
  INSERT INTO auditoria SET
    id_funcionario = NEW.id_funcionario,
    usuario = USER(),
    acao = 'Cadastrou',
    data = CURDATE(),
    hora = CURTIME(),
    salario_antigo = OLD.salario_fixo;
 END$
```

15.2 – Trigger de Auditoria

- O comando **NEW** se refere ao novo id_funcionario gerado na momento da inserção do funcionário. A figura abaixo mostra o que aconteceu na tabela de auditoria ao se inserir um novo funcionário:

id_auditoria	acao	usuario	data	hora	id_funcionario	salario_antigo
1	Cadastrou	root@localhost	2017-03-20	14:43:05	3	800

15.2 – Trigger de Auditoria

- Para ter um melhor controle das mudanças na tabela funcionario, triggers também precisam ser disparadas quando ocorrem operações de UPDATE ou DELETE, como mostra o código abaixo:

```
DELIMITER $
CREATE TRIGGER auditoria_funcionario_update AFTER UPDATE ON funcionario
FOR EACH ROW
  BEGIN
    INSERT INTO auditoria SET
      id_funcionario = NEW.id_funcionario,
      usuario = USER(),
      acao = 'Alterou',
      data = CURDATE(),
      hora = CURTIME();
      salario_antigo = OLD.salario_fixo;
  END$
```

15.2 – Trigger de Auditoria

- Para ter um melhor controle das mudanças na tabela funcionario, triggers também precisam ser disparadas quando ocorrem operações de UPDATE ou DELETE, como mostra o código abaixo:

```
DELIMITER $
CREATE TRIGGER auditoria_funcionario_delete AFTER DELETE ON funcionario
FOR EACH ROW
  BEGIN
    INSERT INTO auditoria SET
      id_funcionario = NEW.id_funcionario,
      usuario = USER(),
      acao = 'Deletou',
      data = CURDATE(),
      hora = CURTIME(),
      salario_antigo = OLD.salario_fixo;
  END$
```

15.2 – Trigger de Auditoria

- O campo salário é importante ser monitorado, pois ele pode ser utilizado para fraldes no momento de gerar a folha de pagamento. Desta forma, sempre que houver um update ou delete na tabela funcionario, o valor do salário alterado será gravado.

id_auditoria	acao	usuario	data	hora	id_funcionario	salario_antigo
1	Cadastrou	root@localhost	2017-03-20	14:43:05	3	800
2	Alterou	root@localhost	2017-03-21	14:43:40	3	1000
3	Deletou	root@localhost	2017-03-21	14:43:55	3	1000

15.3 – Diferenças entre AFTER e BEFORE

- As cláusulas **BEFORE** e **AFTER** vão determinar quando a trigger será executada, ou seja, se ela será executada antes ou depois do comando DML que faz a trigger ser disparada.

- A diferença mais significativa entre o BEFORE e o AFTER é que em uma trigger com AFTER não é possível modificar os valores que já foram modificados ou inseridos na tabela do banco de dados. O exemplo abaixo ilustra a situação onde se deseja alterar um valor que já foi inserido, gerando um erro no momento de criação da trigger:

```
DELIMITER $
CREATE TRIGGER valida_salario AFTER UPDATE ON funcionario
FOR EACH ROW
  BEGIN
  IF (NEW.salario_fixo <= OLD.salario_fixo) THEN
     SET NEW.SALARIO = OLD.SALARIO;
  END IF;
END$
```

15.3 – Diferenças entre AFTER e BEFORE

- O banco de dados não aceitará a trigger anterior, pois ela tenta alterar o campo ‘salario_fixo’ depois que ele já foi modificado, pois a trigger foi disparada após o comando UPDATE.

- Já a trigger abaixo utiliza o **BEFORE**, que irá alterar o valor que será inserido no campo ‘salario_fixo’ antes do UPDATE ser executado. Desta forma, caso um salário seja alterado para um valor inferior ao salário anterior (redução), a trigger manterá o salário atual:

```
DELIMITER $
CREATE TRIGGER valida_salario BEFORE UPDATE ON funcionario
FOR EACH ROW
  BEGIN
  IF (NEW.salario_fixo <= OLD.salario_fixo) THEN
     SET NEW.salario_fixo = OLD.salario_fixo;
  END IF;
END$
```

15.4 – Atualização de Campos com Trigger

- Com o uso de triggers é possível garantir a integridade do sistema. No caso dos exemplos anteriores, existia uma trigger garantindo que um salário nunca fosse reduzido, pois tal operação seria ilegal.

- Outra possível funcionalidade de uma triggers é atualizar campos que se modificam conforme registros de outras tabelas sejam inseridos ou modificados.

- Como exemplo, podemos citar as tabelas envolvidas em uma operação de venda: Cliente, Produto, Pedido_Venda, Itens_Pedido_Venda.

- Sempre que um novo registro da tabela de Item_Pedido é feito, é preciso atualiza o campo ‘total’ da tabela de Pedido, bem como o campo ‘quantidade’ da tabela de produto (controle de estoque). Esta atualização pode ser feita com um ótimo desempenho por meio de triggers.

15.4 – Atualização de Campos com Trigger

- A trigger mostrada no quadro abaixo irá atualizar o campo ‘total’ da tabela de Pedido_Venda sempre que um novo registro for inserido na tabela de Item_pedido_Venda:

```
DELIMITER $
CREATE TRIGGER atualiza_insert_total_venda AFTER INSERT ON itens_pedido_venda
FOR EACH ROW
BEGIN
  UPDATE pedido_venda SET total=total+(NEW.quantidade*NEW.preco) WHERE
id_pedido_venda=NEW.id_pedido_venda;
END$
```

15.4 – Atualização de Campos com Trigger

- Para atualização do campo ‘total’ da tabela Pedido_Venda, também é necessário a criação de um trigger quando houver alguma operação de UPDATE na tabela de Item_Pedido_Venda, alterando os campos ‘quanitdade’ e ‘preco’:

```
DELIMITER $
CREATE TRIGGER atualiza_update_total_venda AFTER UPDATE ON itens_pedido_venda
FOR EACH ROW
BEGIN
   UPDATE pedido_venda SET total=total +(NEW.quantidade*NEW.preco) -
(OLD.quantidade*OLD.preco) WHERE id_pedido_venda=NEW.id_pedido_venda;
END$
```

15.4 – Atualização de Campos com Trigger

- Para atualização do campo ‘total’ da tabela Pedido_Venda, também é necessário a criação de um trigger quando houver alguma operação de DELETE na tabela de Item_Pedido_Venda, sendo necessário debitar o valor correspondente ao registro excluído do campo ‘total’:

```
DELIMITER $
CREATE TRIGGER atualiza_delete_total_venda AFTER DELETE ON itens_pedido_venda
FOR EACH ROW
BEGIN
  UPDATE pedido_venda SET total=total - (OLD.quantidade*OLD.preco) WHERE
id_pedido_venda=OLD.id_pedido_venda;
END$
```