Aula 16

MySQL – Triggers e Views

Trigger ou gatilho

- É associado a uma tabela;
- É disparado quando um determinado evento ocorre na tabela:
  - Eventos de INSERT, UPDATE, DELETE (comandos DML);
  - Para cada comando, pode haver apenas um TRIGGER;
- Podem existir vários TRIGGERS em um banco de dados.

Trigger ou gatilho

- Principais usos:
  - Executar verificações de valores; ou
  - Fazer cálculos sobre os valores informados em um evento.

Trigger ou gatilho

- O TRIGGER é ativado quando uma declaração de INSERT, UPDATE ou DELETE ocorre na tabela associada;
- O disparo do “gatilho” pode ser configurado para ocorrer ANTES ou DEPOIS do evento (comandos INSERT, UPDATE, DELETE).

Trigger ou gatilho

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

- DEFINER: Comando opcional (entre chaves [...]).
- <trigger_name>: define o nome para o processo TRIGGER;

Trigger ou gatilho

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

- <trigger_time>: define se o TRIGGER será ativado antes (BEFORE) ou depois (AFTER) do comando que o disparou:
  - BEFORE: dispara o TRIGGER antes dos dados chegarem à tabela;
  - AFTER: dispara o TRIGGER após a atualização dos dados em uma tabela;

Trigger ou gatilho

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

- <trigger_event>: define qual será o evento de disparo, INSERT, UPDATE, DELETE;
- <tbl_name>: nome da tabela cujo TRIGGER estará associado, aguardando o trigger_event;

Trigger ou gatilho

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

- <trigger_stmt>: procedimento que o TRIGGER realizará quando for disparado;
  - Inicia com o comando BEGIN e termina com comando END;
- Para apagar um TRIGGER, utilize o comando:
  - DROP TRIGGER <trigger_name>

Trigger ou gatilho

- No MySQL, diferentemente de outros SGBD's, não é permitido disparar mensagens, por exemplo, "registro atualizado/inserido/excluído com sucesso";
- Podem conter mais de um comando entre BEGIN-END;
- Podem trabalhar com o valor de "antes" e "depois" dos dados atualizados da tabela;
- Podem atuar em uma tabela com base em outra tabela de qualquer banco de dados dentro do MySQL;
- Os TRIGGERS ficam armazenados na tabela TRIGGERS, dentro do dicionário de dados;

Trigger ou gatilho

Exemplo 01

```
Create table conta (
        numero int,
        total float
);

CREATE TRIGGER ins_soma BEFORE INSERT
ON conta
FOR EACH ROW SET @soma = @soma + NEW.total;
```

Trigger ou gatilho

Exemplo 01

- Disparo de um TRIGGER:

```
SET @soma = 0;
INSERT INTO conta VALUES (137,14.98),(141,1937.50),(97,-100.00);
SELECT @soma;
```

- Repare que ao executar o comando

```
DELETE FROM conta WHERE numero >= 0;
```

- E novamente o comando

```
INSERT INTO conta VALUES (137,14.98),(141,1937.50),(97,-100.00);
```

- Temos que o valor de @soma é diferente, pois ele acumula o valor (não foi resetado).

Trigger ou gatilho

Exemplo 02

- TRIGGER de múltiplos SET (**delimiter** é necessário):

```
delimiter //
CREATE TRIGGER upd_check BEFORE UPDATE ON conta
FOR EACH ROW
BEGIN
        IF NEW.total < 0 THEN
                SET NEW.total = 0;
        ELSEIF NEW.total > 100 THEN
                SET NEW.total = 100;
        END IF;
END;//
delimiter ;
```

- O que ocorre ao executar o comando abaixo?

```
UPDATE conta SET total = 200 WHERE numero = 137;
```

Trigger ou gatilho

Operadores OLD e NEW

- OLD: operador para ser utilizado no interior da TRIGGER para recuperar o valor anterior à um UPDATE, DELETE;
- NEW: operador para ser utilizado no interior da TRIGGER para recuperar dados atuais a partir de comandos INSERT, UPDATE.

Trigger ou gatilho

Exemplo 03

```
delimiter//
CREATE TRIGGER trg_country_up AFTER UPDATE ON Country
FOR EACH ROW
BEGIN
        SET @var_old = OLD.Name;
        SET @var_new = NEW.Name;
END;//
delimiter ;
```

Trigger ou gatilho

Vantagens e Desvantagens

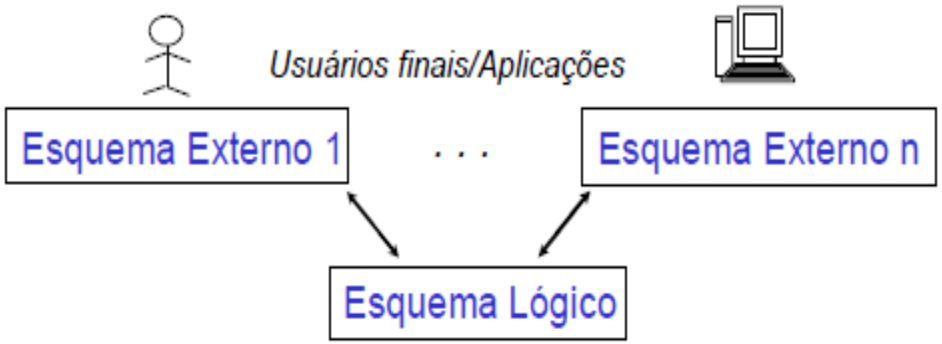
- Vantagens:
  - Parte do processamento que seria executado na aplicação passa para o banco, poupando recursos da máquina do cliente;
  - Facilita a manutenção, sem que seja necessário alterar o código fonte da aplicação.
- Desvantagens:
  - Alguém que tenha acesso não autorizado ao banco de dados poderá visualizar e alterar o processamento realizado pelos gatilhos;
  - Requer maior conhecimento de manipulação do banco de dados (SQL) para realizar as operações internamente.

View

Introdução

- É uma estrutura que não armazena dados;
  - Contém consultas pré-programadas ao banco de dados que permitem unir duas ou mais tabelas e retornar uma única tabela como resultado, quando invocadas.
  - Estas consultas são otimizadas e analisadas de forma automática a cada vez que é feita uma atualização em uma das tabelas.

- Uma VISÃO não existe de forma física. É considerada como uma TABELA VIRTUAL;
- Visões fazem parte dos esquemas externos da arquitetura de um SGBD.



Destaques das Views

- Simplificar a especificação de certas consultas, evitando erros;
- Elimina o uso de tabelas temporárias;
- Controle de acesso;
  - Podem ser configuradas para mostrar colunas diferentes para diferentes usuários do banco de dados.
  - Restringir informações que não podem ser disponibilizadas a todos.
- Integridade - Evitar alterações indevidas no BD (Tabelas);

Como criar, deletar e alterar?

- VISÕES
  - Estrutura básica.
    - **Criar:**

```
CREATE VIEW NomeView AS (SELECT * FROM NomeTabela);
```

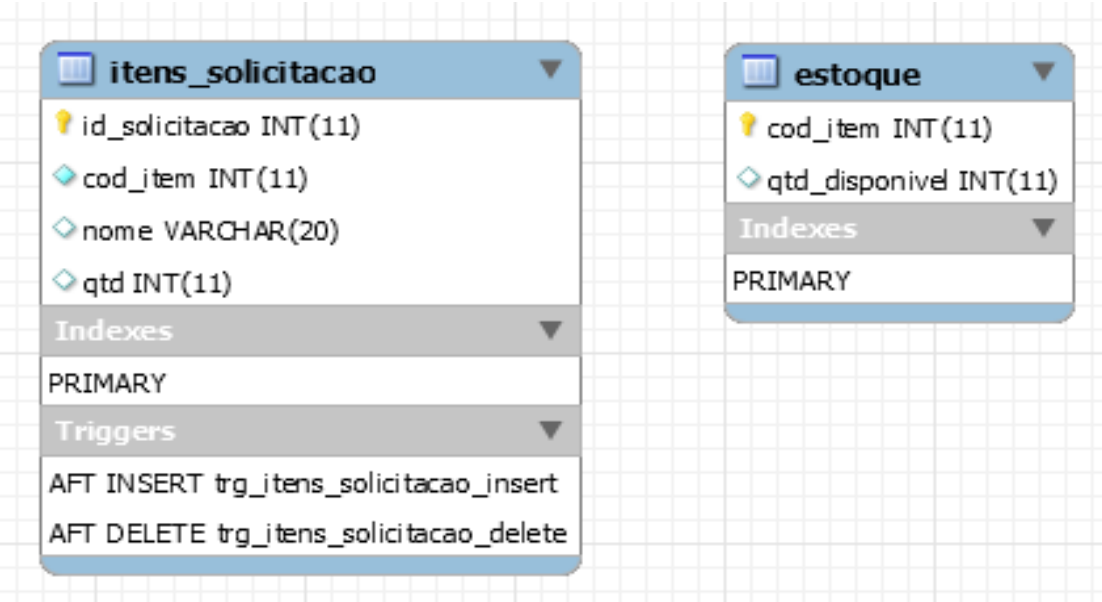
    - **Deletar:**

```
DROP VIEW NomeView;
```

    - **Alterar:**

```
ALTER VIEW NomeView AS (“Novo select”);
```

Exemplo




```
insert into itens_solicitacao (cod_item,nome,qtd) values (100,'Resistor 100k',5);
insert into itens_solicitacao (cod_item,nome,qtd) values (110,'Resistor 1k',2);
insert into itens_solicitacao (cod_item,nome,qtd) values (120,'Capacitor 100n ',2);
insert into itens_solicitacao (cod_item,nome,qtd) values (130,'Transistor TBJ',1);

insert into estoque (cod_item,qtd_disponivel) values (100,1500);
insert into estoque (cod_item,qtd_disponivel) values (110,500);
insert into estoque (cod_item,qtd_disponivel) values (120,300);
insert into estoque (cod_item,qtd_disponivel) values (130,600);
```

Exemplo

1) Crie uma view que retorne a seguinte resposta abaixo:

	Item	Quantidade no estoque
	Resistor 100k	1500
	Resistor 1k	500
	Capacitor 100n	300
	Transistor TBJ	600

Solução

```
# criando a view
CREATE VIEW qtd_estoque AS  (
    SELECT
        i.nome 'Item', e.qtd_disponivel 'Quantidade no estoque'
    FROM
        estoque e,
        itens_solicitacao i
    WHERE
        e.cod_item = i.cod_item

);

# testando a view
SELECT * FROM qtd_estoque;
```

Exercício p/ entregar

```
1     DROP DATABASE IF EXISTS loja;
2     CREATE DATABASE loja;
3     USE loja;
4     SET GLOBAL log_bin_trust_function_creators = 1;
5     CREATE TABLE compra(
6         id int not null auto_increment primary key,
7         preco float,
8         pagamento float
9     );
10
11    INSERT INTO compra VALUES (id, 9.5, 10.25);
12    INSERT INTO compra VALUES (id, 18.99, 25);
13    INSERT INTO compra VALUES (id, 3.99, 5);
14    INSERT INTO compra VALUES (id, 8.85, 8.89);
15    INSERT INTO compra VALUES (id, 19.49, 20);
```

Exercício p/ entregar

- Crie uma VIEW baseada no slide anterior(tabela compra) que busque a quantidade de compras cujo preço é maior ou igual a 10;

- Crie uma VIEW baseada nas tabelas do slide 20 (itens_solicitacao e estoque) que busque a quantidade solicitada(qtd) como 'Querem comprar' e a quantidade disponível(qtd_disponivel) como 'Podem Vender' apenas para os produtos que contenham Resistor no nome

Exercício p/ entregar

Imagine a seguinte situação: um mercado precisa que o estoque dos produtos seja automaticamente atualizado, tanto ao realizar vendas quanto na devolução do produto para o mercado.

```
CREATE TABLE produtos (
        id INT PRIMARY KEY,
        descricao VARCHAR(50) UNIQUE,
        estoque INT NOT NULL
);

INSERT INTO produtos VALUES ('1', 'Lasanha', ’10’);
INSERT INTO produtos VALUES ('2', 'Morango', ‘5’);
INSERT INTO produtos VALUES ('3', 'Farinha', ’15’);

CREATE TABLE itens_venda (
        id_venda INT PRIMARY KEY,
        id_produto INT,
        quantidade INT
);
```

Exercício p/ entregar

Ao inserir e remover registros da tabela itens_venda:

- O estoque do produto vendido deve ser alterado na tabela produtos. Para isso, serão criados dois TRIGGERs:
  - Um AFTER INSERT para dar baixa no estoque;
  - Um AFTER DELETE para fazer a devolução da quantidade vendida do produto.

OBS.: Digitar delimiter // antes do CREATE TRIGGER e digitar // delimiter ; após o comando END;