Banco de dados

Triggers

Prof. Eldane Vieira

Introdução

- A maioria dos SGBDs atuais oferece o recurso das regras ativas, também chamadas de *triggers* e popularmente denominados gatilhos.
- Assim, como as *views*, as regras ativas devem ser criadas antes de as utilizarmos.
- Cada regra é associada a uma só tabela, mas uma tabela, dependendo do SGBD, pode suportar uma ou mais regras.

Trigger

- As regras ativas ou *triggers* são disparadas em resposta à ocorrência de eventos.
- Um *trigger* é composto por três componentes:
  - O evento;
  - A condição;
  - A ação.
- Quando um evento ocorre, a condição da regra é avaliada, se está é verdadeira, a ação é executada.

Trigger

- Um ponto importante é que as regras ativas definem ações sobre os dados, e são acionadas sem a intervenção do usuário, ou seja, o usuário não vê a regra sendo disparada.

Sintaxe de um *trigger*

- CREATE TRIGGER <nome da regra>

  AFTER | BEFORE

  DELETE | OR INSERT | OR UPDATE [OF<nome dos atributos>]

  ON <nome da tabela>

  [FOR EACH ROW | STATEMENT]

  [WHEN <CONDIÇÃO>]

  BEGIN

  …

  END

ROW ou STATEMENT

- Alguns comandos SQL podem afetar várias linhas de dados. Nesses casos, o *trigger* pode ser chamado de duas formas:
  - Opção ROW: Neste caso, o *trigger* é executado múltiplas vezes, uma para cada linha afetada pelo comando.
  - Opção STATEMENT: O *trigger* é chamado somente uma vez para a linha afetada ou um conjunto de linhas afetadas.
    - Esta opção não está disponível em todos os SGBDs, incluindo o *MySQL workbench community.*

Os registros NEW e OLD

- Como os triggers, são executados em conjunto com operações de inclusão e exclusão, é necessário referenciar os registros que estão sendo incluídos, removidos ou atualizados.
  - 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.
  - Na inserção a palavra reservada OLD não é usada.
- 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.
- A operação UPDATE pode utilizar tanto o NEW, referente a nova informação, e o OLD, referente aquilo que será atualizado.

Deletar um Trigger

- Para deletar um trigger basta executar o seguinte comando:
  - DROP TRIGGER <nome do trigger>;

Esquema de tabelas

- Os exemplos nesta aula serão feitos sobre os seguinte esquema de tabelas, disponibilizados no disco virtual:
  - Cliente (ID_Cliente, nome, endereco, cidade, telefone, tipo)

    Empresa (ID_Empresa, CNPJ, nome,endereco, cidade, telefone)

    Cliente_Empresa (ID_Empresa, ID_Cliente)

    Produto (ID_Produto, nome, qnt_vendida, valor, total_produzido)

    Produto_Empresa(ID_Produto,ID_Empresa)

    Produto_Comprado_Cliente(ID_Produto,ID_Cliente, quantidade)

Exemplos Trigger

- Crie a tabela Log no banco “aulabd”:
  - create table Log(

```
id integer auto_increment,
dataLog datetime,
obs varchar(50),
tabela varchar(20),
atributo varchar(20),
constraint primary key(id)
);
```

Exemplo 1

- Criação de um trigger que insere um registro na tabela Log caso o valor do produto inserido na tabela Produto seja negativo.
  - delimiter $$

    CREATE TRIGGER VerificaValor

    AFTER INSERT

    ON Produto

    FOR EACH ROW

    BEGIN

    if (NEW.valor < 0) then

    INSERT INTO Log

    SET

    datalog = now(),

    obs = ‘Valor do produto inválido’,

    tabela=’Produto’,

    atributo=’valor’;

    END IF;

    END;$$
  - *Teste: insert into produto (nome, qnt_vendida,valor,total_produzido) values (’novo produto’,0,-10,100);*

Exemplo 2

- Criação de um trigger que não permite a atualização do total produzido para um valor menor que o atual. Caso o valor seja menor, então o valor original será preservado.
  - delimiter $$

    CREATE TRIGGER VerificaProducao

    BEFORE UPDATE

    ON Produto

    FOR EACH ROW

    BEGIN

        if (NEW.total_produzido < OLD.total_produzido) then

            SET NEW.total_produzido = OLD.total_produzido;

        END IF;

    END;$$
  - *Teste: update produto set total_produzido = -10 where nome = 'p3';*
  - Talvez seja necessário eliminar as proteções de atualização executando o comando SET SQL_SAFE_UPDATES = 0;

Exemplo 3

- Criação de um *trigger* que impede qualquer deleção na tabela Produto.
  - delimiter $$

    CREATE TRIGGER ControleDelete

    BEFORE DELETE

    ON Produto

    FOR EACH ROW

    BEGIN

        DELETE FROM produto

        WHERE ID_Produto = -1; /*Altera o comando de deleção atribuindo o ID a um ID que não existe, logo a deleção não ocorrerá*/

    END;$$
  - *Teste: DELETE FROM produto WHERE ID_Produto = 1;*

Exemplo 4

- Criação de um *trigger* que não permite inserir um registro na tabela produto onde a quantidade vendida é maior que a produzida. Caso isso ocorra, os valores serão igualados.
  - delimiter $$

    CREATE TRIGGER ControleQnt

    BEFORE INSERT

    ON Produto

    FOR EACH ROW

    BEGIN

        if (new.qnt_vendida > new.total_produzido) then

            set new.qnt_vendida=new.total_produzido;

        end if;

    end;$$
- *Teste: insert into produto (nome, qnt_vendida,valor,total_produzido) values ('produto105',100,20,90);*

Exercícios

Exercício 1

- Crie um *trigger* onde toda inserção de registro na tabela Cliente, no qual a cidade é Curitiba, o tipo do cliente deverá, obrigatoriamente, ser Atacadista.

Cliente (ID_Cliente, nome, endereco, cidade, telefone, tipo)
Empresa (ID_Empresa, CNPJ, nome, endereco, cidade, telefone)
Cliente_Empresa (ID_Empresa, ID_Cliente)
Produto (ID_Produto, nome, qnt_vendida, valor, total_produzido)
Produto_Empresa(ID_Produto,ID_Empresa)
Produto_Comprado_Cliente(ID_Produto,ID_Cliente, quantidade)

Exercício 2

- Crie um *trigger* para garantir que caso ocorra uma atualização na tabela Empresa, o CNPJ continuará o mesmo.

Cliente (ID_Cliente, nome, endereco, cidade, telefone, tipo)
Empresa (ID_Empresa, CNPJ, nome, endereco, cidade, telefone)
Cliente_Empresa (ID_Empresa, ID_Cliente)
Produto (ID_Produto, nome, qnt_vendida, valor, total_produzido)
Produto_Empresa(ID_Produto,ID_Empresa)
Produto_Comprado_Cliente(ID_Produto,ID_Cliente, quantidade)

Exercício 3

- Crie um *trigger* ativado em uma atualização na tabela Produto_Comprado_Cliente, que impeça a atualização da quantidade de produtos comprados para um valor inferior que o valor anterior. Se isto ocorrer, o valor da atualização deve ser igual ao valor antigo.

Cliente (ID_Cliente, nome, endereco, cidade, telefone, tipo)
Empresa (ID_Empresa, CNPJ, nome, endereco, cidade, telefone)
Cliente_Empresa (ID_Empresa, ID_Cliente)
Produto (ID_Produto, nome, qnt_vendida, valor, total_produzido)
Produto_Empresa(ID_Produto,ID_Empresa)
Produto_Comprado_Cliente(ID_Produto,ID_Cliente, quantidade)

Exercício 4

- Crie um *trigger* que impeça a inserção de um registro na tabela Produto onde o total_produzido seja negativo. Se isso ocorrer o valor do total_produzido deve ser zero.

Cliente (ID_Cliente, nome, endereco, cidade, telefone, tipo)
Empresa (ID_Empresa, CNPJ, nome, endereco, cidade, telefone)
Cliente_Empresa (ID_Empresa, ID_Cliente)
Produto (ID_Produto, nome, qnt_vendida, valor, total_produzido)
Produto_Empresa(ID_Produto,ID_Empresa)
Produto_Comprado_Cliente(ID_Produto,ID_Cliente, quantidade)

Exercício 5

- Crie um *trigger* que impeça a inserção de um cliente cujo tipo seja diferente de atacadista ou varejista. Se isso ocorrer o campo tipo deve ficar em branco.

Cliente (ID_Cliente, nome, endereco, cidade, telefone, tipo)
Empresa (ID_Empresa, CNPJ, nome, endereco, cidade, telefone)
Cliente_Empresa (ID_Empresa, ID_Cliente)
Produto (ID_Produto, nome, qnt_vendida, valor, total_produzido)
Produto_Empresa(ID_Produto,ID_Empresa)
Produto_Comprado_Cliente(ID_Produto,ID_Cliente, quantidade)