

Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA Departamento de Informática Análise e Desenvolvimento de Sistemas / Licenciatura em Computação

Views SQL e Triggers

André L. R. Madureira <andre.madureira@ifba.edu.br>
Doutorando em Ciência da Computação (UFBA)
Mestre em Ciência da Computação (UFBA)
Engenheiro da Computação (UFBA)

Porque criar consultas em nível de view?

- Dificuldade em escrever, reescrever e entender consultas
 - Consultas SQL grandes, complexas, e derivadas de várias tabelas (joins e subconsultas)

Views SQL

Views não podem ter atributos com mesmo nome, logo devemos utilizar **nomes qualificados** sempre que acessamos algum atributo

- Consultas podem ser armazenadas como Views
 - View = consultas armazenadas, tabelas virtuais ou pseudotabelas

Objetivo:

- Facilitar o acesso a dados contidos em várias tabelas
- Facilitar o uso de consultas grandes ou complexas
- Implementar segurança nos dados de uma tabela
 - Restringir acesso de usuários do DB a apenas alguns dados (instâncias) ou atributos

Views SQL

- Criar view (sintaxe):
 - CREATE VIEW <nome_da_view>[(atributos_view)] AS <consulta_SQL>
- Verificar se a view foi criada:
 - SHOW TABLES
- Alterar View:
 - ALTER VIEW <nome_da_view> AS <nova_consulta_SQL>
- Excluir View:
 - DROP VIEW <nome_da_view>

Views podem ser usadas em qualquer lugar da consulta SQL que aceite o nome de uma relação (tabela)

Exemplo de View -Controle de Acesso a Dados

CREATE VIEW devedor_banco AS (
 SELECT nome, quantia, tx_juros
 FROM Devedor
);

Tabela base: tabela real (modelo lógico) usada para construir a view

Ex: Devedor é a tabela base da view devedor_banco

- SELECT nome, quantia FROM devedor_banco;
 - Mostre somente o nome e quantia do cliente que tomou um empréstimo no banco (devedor)
 - Perceba que a view serve para controlar quais atributos da tabela Devedor temos acesso
 - Maior segurança no uso do DB pelos usuários

Exemplo de View -Simplificação de Consultas SQL Complexas

```
    CREATE VIEW func_max_salario(depto, max_salario) AS (
        SELECT depto, MAX(salario)
        FROM Funcionario
        GROUP BY depto
);
```

- SELECT F.nome, F.depto
 FROM Funcionario AS F, func_max_salario AS M
 WHERE F.salario = M.max_salario AND F.depto = M.depto;
 - Encontre o funcionário com maior salário de cada departamento

Delete / Update em Views SQL

- É possível realizar operações de atualizações em views, que são realizadas na tabela base (modelo lógico), desde que:
 - A cláusula FROM possua apenas uma relação
 - SELECT possui apenas atributos
 (sem funções de agregação, expressões, ou DISTINCT)
 - Atributos não listados no SELECT podem ser definidos como NULL
 - SELECT não possui HAVING ou GROUP BY
- Além dessas restrições, precisamos também criar a view usando o comando WITH CHECK OPTION

Exemplo de Delete / Update em Views SQL

- CREATE VIEW vw_empregado1 AS SELECT * FROM Empregado
 WHERE nome_categoria = "Cat A" WITH CHECK OPTION;
- UPDATE vw_empregado1 SET nome = "João da Silva" WHERE nome = "João";

Apesar de ser possível a atualização de algumas views, esse procedimento é ALTAMENTE DESACONSELHÁVEL

Views devem ser utilizadas majoritariamente para consultas SELECT

Comando WITH CHECK OPTION

- Impor restrições na atualização de Views
- Atualizações que são emitidas sobre a view terão que se encaixar às condições definidas na cláusula WHERE do SELECT
 - Ex: CREATE VIEW vw_empregado2 AS SELECT * FROM
 Empregado WHERE nome_categoria = "Cat B" WITH CHECK
 OPTION;
 - UPDATE vw_empregado2 SET nome_categoria = "Cat A" WHERE nome = "Ricardo";

Implementação de Views no SQL

- Views podem ser implementadas nos SGBDs através de duas técnicas:
 - Modificação de consulta
 - SGBD substitui o nome das views pela consulta SQL inteira que compõem a view
 - Materialização de view
 - SGBD armazena as views no banco de dados (como se fosse uma tabela comum)
 - Views materializadas atuam como uma espécie de "cache" para consultas complexas em bancos de dados SQL

Modificação de Consulta com View

- A maioria dos DBMS implementa as views como uma substituição simples do nome da view pela consulta SQL inteira da view
 - Onde há o nome da tabela virtual (view), o DBMS substitui pela consulta SQL da view
 - Isso permite a expansão de view (uma view definida a partir de outra)
 - CREATE VIEW vw_empregado1 AS
 SELECT * FROM Empregado WHERE nome_categoria = 'Cat A';
 - CREATE VIEW vw_emp_nome ASSELECT nome FROM vw_empregado1;

Expansão de view

- O DBMS substitui as views sempre que eles forem utilizadas em consultas SQL até que nenhuma view exista mais na consulta
 - CREATE VIEW vw_empregado1 AS
 SELECT * FROM Empregado WHERE cat = 'A';
 - CREATE VIEW vw_emp_nome AS
 SELECT nome FROM vw_empregado1;

```
SELECT * FROM vw_emp_nome
```

SELECT * FROM (SELECT nome FROM vw_empregado1)

SELECT * FROM (SELECT nome FROM (SELECT * FROM Empregado **WHERE** cat = 'A'))

Views Materializadas

- Para obter maior desempenho, alguns DBMS implementam views materializadas
 - O DBMS n\u00e3o precisa executar a consulta SQL da view sempre que a view \u00e9 utilizada
 - O DBMS executa a consulta SQL uma vez e cria uma tabela dentro do DB, como uma espécie de "cache" da view
- Desvantagem: necessidade de atualizar a view materializada sempre que uma das tabelas base forem atualizadas

View por Modificação x Views Materializada

View por Modificação

- Consome pouco espaço do DB
- Apenas a consulta é armazenada
- Consulta SQL é realizada somente quando uma consulta é feita usando a view

View Materializada

- Consome mais espaço do DB
- O resultado da consulta é armazenado em uma nova tabela
- Consulta SQL realizada sempre que uma das tabelas usadas para construir a view é atualizada

Quando o SGBD utiliza cada tipo de view?

- Na prática, a maioria dos SGBDs usa ambas as views (por modificação e materializadas)
 - SGBD cria views materializadas e as mantem armazendas enquanto as views estão sendo consultadas
 - Se a view n\u00e3o for consultada por certo per\u00e1odo, o sistema remove automaticamente a view
 - É responsabilidade do SGBD manter as views atualizadas, independente da implementação

Comando WITH

- Permite criar uma VIEW temporária
 - Facilita o gerenciamento de consultas SQL complexas,
 semelhante a uma função de uma linguagem de programação
- O comando WITH só é valido para as cláusulas/comandos SQL que são executados junto com o WITH
- Sintaxe:
 - WITH <nome_da_view>(<atributos_view>) AS <consulta_SQL>;

O comando **WITH** é aceito apenas por alguns bancos de dados. Na dúvida, consulte a documentação do sistema.

Exemplo do Comando WITH

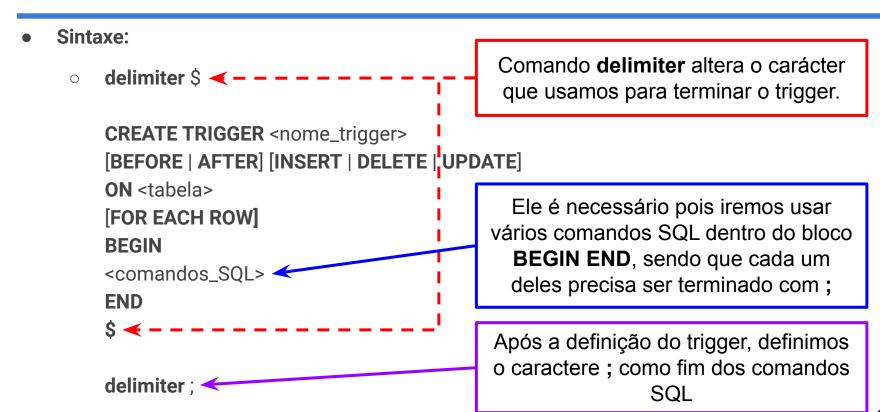
- Ex: WITH saldo_max(valor) AS SELECT MAX(saldo) FROM conta;
 SELECT num_conta FROM conta, saldo_max WHERE conta.saldo = saldo_max.valor;
 - Encontre o número da conta que contém o maior saldo dentre todas as contas
- Ex: WITH total_agencia(nome_agencia, soma_saldo) AS SELECT nome_agencia, SUM(saldo) FROM conta GROUP BY nome_agencia;

SELECT nome_agencia **FROM** total_agencia;

Gatilhos (*Triggers*) SQL

- Permitem que uma ação seja executada ANTES (BEFORE) ou DEPOIS (AFTER) que uma determinada condição (evento) ocorrer
- Triggers podem ser usados para monitorar o banco de dados
 - Por isso, bancos de dados que contém triggers são chamados de bancos de dados ativos ou orientados a eventos
 - São usados em conjunto com as restrições de integridade para impor regras sobre os dados

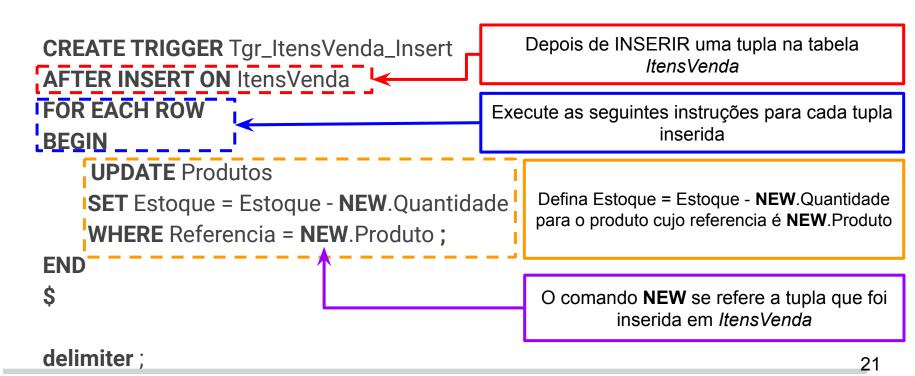
Gatilhos (*Triggers*) SQL



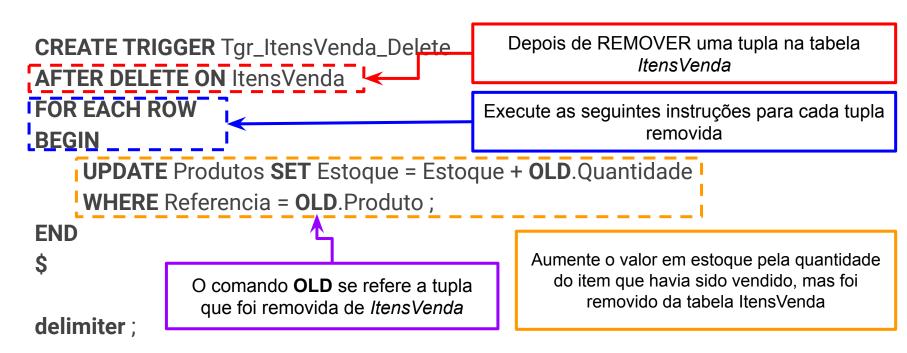
 Desejamos que ao inserir e remover registro da tabela *ItensVenda*, o estoque do produto referenciado seja alterado na tabela *Produtos* do seguinte DB:

```
CREATE TABLE Produtos (
     Referencia VARCHAR(3) PRIMARY KEY,
     Descrição VARCHAR(50) UNIQUE,
     Estoque
               INT NOT NULL DEFAULT 0
CREATE TABLE ItensVenda (
     Venda
                INT.
     Produto
                  VARCHAR(3),
     Ouantidade INT
                                INSERT INTO Produtos VALUES ("001", "Feijão", 10);
                                INSERT INTO Produtos VALUES ("002", "Arroz", 5);
                                INSERT INTO Produtos VALUES ("003", "Farinha", 15);
```

delimiter \$



delimiter \$



- Vamos testar os triggers inserindo alguns produtos vendidos no DB:
 - INSERT INTO ItensVenda VALUES (1, "001", 3);
 INSERT INTO ItensVenda VALUES (1, "002", 1);
 INSERT INTO ItensVenda VALUES (1, "003", 5);
- Como ficou o DB ? Faça alguns SELECTs nas tabelas para ver o estado do DB.
- Agora vamos fazer o estorno da venda (remoção do produto vendido):
 - DELETE FROM ItensVenda WHERE Venda = 1 AND Produto = "001";
- Como ficou o estado do DB?

Mostrando e Removendo Gatilhos (*Triggers*) SQL

- Mostrar triggers:
 - SHOW TRIGGERS;
- Remover trigger:
 - DROP TRIGGER nome_do_trigger;

Tutoriais sobre Triggers:

- https://www.devmedia.com.br/mysql-basico-triggers/37462
- https://www.dolthub.com/blog/2023-06-09-writing-mysql-triggers/
- https://dev.mysql.com/doc/refman/8.4/en/trigger-syntax.html

O que acontece se o gatilho (*Trigger*) SQL falhar?

- No MySQL, se qualquer comando SQL entre o BEGIN e o END do trigger falhar, a transação que causou o disparo da trigger também irá falhar
 - Isto pode ser algo interessante para garantir condições especiais como:
 - Estoque > 0 para poder vender um produto
 - Saldo > 0 para realizar uma transferência bancária
 - Dentre outros

Quando usar gatilhos (*Triggers*) SQL?

- Apesar de ser tentador usar triggers para facilitar o desenvolvimento do DB, é necessário tomar alguns cuidados:
 - Quanto mais triggers são usados, maior a sobrecarga no sistema
 - Várias transações sendo executadas uma após a outra, cada uma com suas condições próprias (CHECK, IF ELSE, etc)
 - Não há técnicas para verificar se os triggers estão corretos
 - Por exemplo, não temos garantia que um trigger não viola alguma condição de uma tabela do DB

Referencial Bibliográfico

 KORTH, H.; SILBERSCHATZ, A.; SUDARSHAN, S.
 Sistemas de bancos de dados. 5. ed. Rio de Janeiro: Ed. Campus, 2006.

 DATE, C. J. Introdução a sistemas de bancos de dados. Rio de Janeiro: Ed. Campus, 2004. Tradução da 8ª edição americana.