SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Curso

TÉCNICO EM DESENVOLVIMENTO

DE SISTEMAS

SQL Views

Manuela Leme Morais Almeida

Sorocaba
Nov – 2024

SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Manuela Leme Morais Almeida

SQL Views

Pesquisa sobre o Conceito,
Benefícios e Aplicações Práticas
do SQL Views
Prof. – Emerson Magalhães

Sorocaba
Nov – 2024

SUMÁRIO

SQL Views

INTRODUÇÃO

SQL Views são tabelas virtuais baseadas em resultados de uma consulta SQL. Elas não armazenam dados fisicamente, mas exibem dados extraídos de uma ou mais tabelas do banco de dados. Views podem ser usadas como uma tabela comum em operações de consulta.

As views são essenciais para simplificar consultas complexas, melhorar a segurança restringindo o acesso a certos dados, e facilitar o controle de acessos, além de permitir a reutilização de consultas frequentes. Elas são amplamente usadas para criar uma camada de abstração em sistemas que necessitam de relatórios ou acesso restrito a informações.

A pesquisa visa explicar o conceito de views, sua importância, os tipos de views disponíveis, além de explorar o processo de criação e uso com exemplos práticos em um banco de dados fictício.

1. FUNDAMENTOS TEÓRICOS DAS SQL VIEWS

1.1. O QUE SÃO VIEWS E COMO FUNCIONAM NO SQL

Views são consultas pré-definidas armazenadas no banco de dados, funcionando como tabelas virtuais. Quando uma view é chamada, a consulta subjacente é executada para fornecer os resultados em tempo real.

1.2. DIFERENÇAS ENTRE VIEWS E TABELAS COMUNS

- **Tabelas** armazenam dados fisicamente, enquanto **views** apenas referenciam dados de tabelas existentes.
- Tabelas podem ser manipuladas diretamente (inserir, atualizar, deletar), mas views, dependendo de como são construídas, podem ter restrições quanto a essas operações.

1.3. TIPOS DE VIEWS

- **Views Simples**: Baseadas em uma única tabela sem funções agregadas.
- **Views Complexas**: Utilizam junções (JOINs) e funções agregadas (SUM, AVG, etc.) para combinar dados de várias tabelas.
- **Views Materializadas** (caso suportadas pelo banco): São views que armazenam fisicamente os dados da consulta, oferecendo melhor desempenho, mas requerem atualização periódica.

2. VANTAGENS E DESVANTAGENS DE USAR VIEWS

2.1. VANTAGENS

- **Simplificação de Consultas**: Reduz a complexidade de consultas frequentes e complexas.
- **Segurança**: Limita o acesso a dados sensíveis exibindo apenas colunas/linhas específicas.
- **Facilidade de Manutenção**: Permite alterações nas tabelas subjacentes sem impactar os sistemas que utilizam as views.

2.2. DESVANTAGENS

- **Impacto no Desempenho**: Views complexas podem ter um impacto negativo na performance, especialmente em grandes bases de dados.
- **Limitações nas Atualizações**: Nem todas as views são atualizáveis; em algumas, não é possível realizar operações de INSERT, UPDATE ou DELETE.
- **Manutenção em Views Materializadas**: Demandam recursos adicionais para atualização periódica dos dados armazenados.

FIGURA 1- NORMAS ABNT

3. PROCESSO DE CRIAÇÃO DE VIEWS NO SQL

3.1. INSTRUÇÃO CREATE VIEW: SINTAXE E PARÂMETROS

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

3.2. EXEMPLOS DE VIEWS SIMPLES

3.2.1. VIEW DE FILTRAGEM

```sql
CREATE VIEW Clientes_Ativos AS
SELECT nome, email
FROM clientes
WHERE status = 'ativo';
```

A view exibe apenas clientes ativos, selecionando colunas específicas e aplicando filtro em linhas.

3.2.2. VIEW DE AGREGAÇÃO

```sql
CREATE VIEW Total_Vendas AS
SELECT vendedor_id, SUM(valor) AS total_vendas
FROM vendas
GROUP BY vendedor_id;
```

View mostra o total de vendas por vendedor, usando funções agregadas (SUM, AVG, COUNT).

3.2.3. VIEW DE JUNÇÃO

```sql
CREATE VIEW Pedidos_Detalhados AS
SELECT p.id, c.nome, p.data_pedido, p.valor_total
FROM pedidos p
JOIN clientes c ON p.cliente_id = c.id;
```

View combina dados de clientes e pedidos, ou seja, usa a combinação de dados de várias tabelas para fornecer uma visão mais completa e detalhada.

3.3. EXEMPLO DE VIEW COMPLEXA

```sql
CREATE VIEW Relatorio_Vendas AS
SELECT c.nome AS Cliente, p.nome AS Produto, SUM(v.quantidade) AS Total
FROM vendas v
JOIN produtos p ON v.produto_id = p.id
JOIN clientes c ON v.cliente_id = c.id
GROUP BY c.nome, p.nome;
```

4. VIEWS ATUÁLIZAVEIS E NÃO ATUALIZÁVEIS

4.1. POSSIBILIDADE DE ATUALIZAR DADOS EM VIEWS

Uma view é considerada atualizável quando permite operações como INSERT, UPDATE e DELETE diretamente através dela. Contudo, views que

utilizam funções agregadas, junções complexas ou subconsultas geralmente não são atualizáveis.

4.2. CONDIÇÕES PARA QUE A VIEW SEJA ATUALIZÁVEL

- Deve ser baseada em uma única tabela.
- Não deve conter funções agregadas ou DISTINCT.
- Todas as colunas na view devem ser derivadas diretamente da tabela subjacente.

4.3. EXEMPLO PRÁTICO

4.3.1. VIEW ATUALIZÁVEL

```sql
CREATE VIEW Clientes_Simples AS
SELECT id, nome, email
FROM clientes
WHERE status = 'ativo';

UPDATE Clientes_Simples SET email = 'novo_email@exemplo.com' WHERE id = 1;
```

4.3.2. VIEW NÃO ATUALIZÁVEL

```sql
CREATE VIEW Vendas_Agregadas AS
SELECT vendedor_id, SUM(valor) AS total_vendas
FROM vendas
GROUP BY vendedor_id;
```

5. ESTUDO DE CASO

5.1. BANCO DE DADOS FICTÍCIO

Loja de E-commerce

```sql
-- ================================
-- 1. CRIAÇÃO DO BANCO DE DADOS
-- ================================

-- Criar um banco de dados chamado 'loja_ecommerce'
CREATE DATABASE loja_ecommerce;

-- Selecionar o banco de dados para uso
USE loja_ecommerce;

-- ====================================
-- 2. CRIAÇÃO DAS TABELAS PRINCIPAIS
-- ====================================

-- Criação da tabela 'clientes'
CREATE TABLE clientes (
    id INT AUTO_INCREMENT PRIMARY KEY,  -- ID único para cada cliente (chave primária)
    nome VARCHAR(100),                  -- Nome do cliente
    email VARCHAR(100),                 -- E-mail do cliente
    telefone VARCHAR(15),               -- Telefone para contato
    status VARCHAR(20) DEFAULT 'ativo', -- Status do cliente (ativo/inativo)
    data_cadastro DATE                  -- Data de cadastro do cliente
);

-- Criação da tabela 'produtos'
CREATE TABLE produtos (
    id INT AUTO_INCREMENT PRIMARY KEY,  -- ID único para cada produto (chave primária)
    nome VARCHAR(100),                  -- Nome do produto
    descricao TEXT,                     -- Descrição do produto
    preco DECIMAL(10, 2),               -- Preço do produto (até 10 dígitos com 2 casas decimais)
    quantidade INT,                     -- Quantidade em estoque
    data_adicionado DATE                -- Data em que o produto foi adicionado ao estoque
);

-- Criação da tabela 'vendas'
CREATE TABLE vendas (
    id INT AUTO_INCREMENT PRIMARY KEY,  -- ID único para cada venda (chave primária)
    cliente_id INT,                     -- ID do cliente que fez a compra (chave estrangeira)
    data_pedido DATE,                   -- Data em que o pedido foi realizado
    valor_total DECIMAL(10, 2),         -- Valor total da venda
    FOREIGN KEY (cliente_id) REFERENCES clientes(id) -- Definindo a chave estrangeira (cliente_id)
);
```

```sql
-- Criação da tabela 'itens_venda'
CREATE TABLE itens_venda (
    id INT AUTO_INCREMENT PRIMARY KEY,  -- ID único para cada item vendido (chave primária)
    venda_id INT,                       -- ID da venda (chave estrangeira)
    produto_id INT,                     -- ID do produto vendido (chave estrangeira)
    quantidade INT,                     -- Quantidade do produto vendido
    preco_unitario DECIMAL(10, 2),      -- Preço unitário do produto no momento da venda
    FOREIGN KEY (venda_id) REFERENCES vendas(id),     -- Definindo chave estrangeira (venda_id)
    FOREIGN KEY (produto_id) REFERENCES produtos(id)  -- Definindo chave estrangeira (produto_id)
);

-- ====================================
-- 3. INSERÇÃO DE DADOS NAS TABELAS
-- ====================================

-- Inserindo clientes
INSERT INTO clientes (nome, email, telefone, data_cadastro)
VALUES
('Alice Santos', 'alice@email.com', '11987654321', '2024-01-10'),
('Bruno Silva', 'bruno@email.com', '11912345678', '2024-02-15'),
('Carla Moreira', 'carla@email.com', '21998765432', '2024-03-05');
```


```sql
-- ====================================
-- 3. INSERÇÃO DE DADOS NAS TABELAS
-- ====================================

-- Inserindo clientes
INSERT INTO clientes (nome, email, telefone, data_cadastro)
VALUES
('Alice Santos', 'alice@email.com', '11987654321', '2024-01-10'),
('Bruno Silva', 'bruno@email.com', '11912345678', '2024-02-15'),
('Carla Moreira', 'carla@email.com', '21998765432', '2024-03-05');
```


```sql
-- Inserindo produtos
INSERT INTO produtos (nome, descricao, preco, quantidade, data_adicionado)
VALUES
('Notebook', 'Notebook Dell 15 polegadas', 3500.00, 15, '2024-01-20'),
('Smartphone', 'Smartphone Samsung Galaxy', 2500.00, 30, '2024-02-01'),
('Teclado Mecânico', 'Teclado RGB para gamers', 300.00, 50, '2024-03-10');

-- Inserindo vendas
INSERT INTO vendas (cliente_id, data_pedido, valor_total)
VALUES
(1, '2024-03-15', 3800.00),
(2, '2024-03-18', 2500.00),
(1, '2024-04-05', 600.00);

-- Inserindo itens vendidos em cada venda
INSERT INTO itens_venda (venda_id, produto_id, quantidade, preco_unitario)
VALUES
(1, 1, 1, 3500.00),  -- Cliente 1 comprou 1 Notebook
(1, 3, 1, 300.00),   -- Cliente 1 comprou 1 Teclado Mecânico
(2, 2, 1, 2500.00),  -- Cliente 2 comprou 1 Smartphone
(3, 3, 2, 300.00);   -- Cliente 1 comprou 2 Teclados Mecânicos
```

```sql
-- ====================================
-- 4. CRIAÇÃO DAS VIEWS
-- ====================================

-- View para Relatório de Vendas (detalhamento por cliente e produto)
CREATE VIEW Relatorio_Vendas AS
SELECT
    v.id AS venda_id,
    c.nome AS cliente,
    p.nome AS produto,
    iv.quantidade,
    iv.preco_unitario,
    (iv.quantidade * iv.preco_unitario) AS total_item,
    v.data_pedido,
    v.valor_total AS total_venda
FROM vendas v
JOIN clientes c ON v.cliente_id = c.id
JOIN itens_venda iv ON iv.venda_id = v.id
JOIN produtos p ON iv.produto_id = p.id;

-- View para Consulta de Estoque (produtos com quantidade abaixo de 20 unidades)
CREATE VIEW Estoque_Baixo AS
SELECT
    id AS produto_id,
    nome AS produto,
    quantidade,
    preco
FROM produtos
WHERE quantidade < 20;

-- ====================================
-- 5. TESTANDO AS VIEWS
-- ====================================

-- Consultando a view Relatorio_Vendas
SELECT * FROM Relatorio_Vendas;

-- Consultando a view Estoque_Baixo
SELECT * FROM Estoque_Baixo;
```

5.2. DISCUSSÃO SOBRE VANTAGENS DAS VIEWS ESTUDO DE CASO

No contexto da loja de e-commerce, as **views** desempenham um papel crucial na simplificação da análise de dados e na geração de relatórios gerenciais. Ao criar views, a equipe de TI pode fornecer aos gerentes de vendas e estoque acesso rápido a informações essenciais, sem a necessidade de escrever consultas SQL complexas sempre que precisarem acessar os dados.

CONCLUSÃO

As SQL Views são ferramentas poderosas que ajudam a simplificar consultas, melhorar a segurança de dados e facilitar a manutenção em sistemas de banco de dados relacionais.

As views devem ser usadas de forma estratégica, considerando as limitações de desempenho e atualização. Práticas recomendadas incluem a criação de views para consultas frequentes e a limitação do uso de views complexas em grandes bases de dados.

Em questão de sugestões de boas práticas há a indicação da utilização de views para consultas frequentes e relatórios, buscar a preferência por views simples para operações que exigem alta performance e evitar views complexas em cenários que demandam atualizações frequentes.

BIBLIOGRAFIA

SQL VIEWS. In: W3Schools. Disponível em:< https://www.w3schools.com/sql/sql_view.asp> Acesso em: 12 nov. 2024.

SQL VIEWS. In: GeeksforGeeks. Disponível em:< https://www.geeksforgeeks.org/sql-views/> Acesso em: 12 nov. 2024.

USING VIEWS. In: MySQL. Disponível em:< https://dev.mysql.com/doc/refman/8.4/en/views.html> Acesso em: 12 nov. 2024.

VIEW MATERIALIZADA. In: Rockcontent. Disponível em:<https://rockcontent.com/br/blog/view-materializada/> Acesso em: 12 nov. 2024.