SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Curso

TÉCNICO EM DESENVOLVIMENTO

DE SISTEMAS

SQL Views: Conceitos, Benefícios e Aplicações Práticas

Isabely Rodrigues Kusmitsch Marcelino

Sorocaba
Novembro 2024

SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Isabely Rodrigues Kusmitsch Marcelino

SQL Views: Conceitos, Benefícios e Aplicações Práticas

Pesquisa para compreender o que são SQL Views e sua importância no contexto de Bancos de Dados Relacionais

Prof. Emerson Magalhães

Sorocaba
Novembro 2024

Sumário

INTRODUÇÃO

As Views em SQL são consultas SQL que foram salvas em um sistema de banco de dados relacional. Essas views operam como uma tabela virtual, permitindo uma representação simples de certos dados e a modificação deles a partir de uma ou mais tabelas, sem a necessidade de repeticoes.

O propósito das views é armazenar e apresentar os dados sem que eles precisem ser guardados em uma tabela física, o que facilita o manejo dos dados de forma mais prática.

A utilizacao das views em sistemas de banco de dados relacionais traz diversos benefícios como a economia de tempo, uma vez que não é necessário escrever várias instruções, o acesso mais ágil aos dados e uma maior organização para o usuário.

Em suma, essa pesquisa procura trazer o que são as SQL Views, sua importância e como podem ser utilizadas para proporcionar facilidades como uma manipulação mais eficiente dos dados. Ademais, serão abordadas maneiras para a criação de views e exemplos práticos que demonstram suas aplicações no cotidiano.

1. FUNDAMENTOS TEÓRICOS DAS SQL VIEWS

1.1. O Que São Views e Como Elas Funcionam no SQL

As views, ou visões, são consultas que ficam registradas no banco de dados e são chamadas de tabelas virtuais. Isso ocorre porque essas consultas não são armazenadas fisicamente, seus dados são acessados diretamente de suas tabelas de origem. Essa abordagem permite que os dados sejam consultados de maneira simples e eficaz, oferecendo uma apresentação personalizada das informações e possibilitando a restrição do acesso aos dados nas tabelas reais.

Uma view é gerada a partir das instruções “SELECT” do usuário, resultando em uma "visão" específica dos dados que provem de uma ou mais tabelas no banco de dados.

1.2. Diferenças entre Views e Tabelas Comuns

Uma tabela comum é composta por linhas e colunas que armazenam dados que podem ser acessados a qualquer momento. Quando usamos o comando "SELECT * FROM", estamos visualizando a tabela, entretando ao realizar outras operações no SQL, essa visualização pode ser perdida.

Assim, enquanto uma tabela serve como um objeto que armazena dados dentro de um banco de dados, a view atua como um objeto que permite criar e recuperar um conjunto de dados a partir de consultas a outras tabelas.

Entre as diferenças principais, estao que a tabela é um objeto independente, enquanto a view depende de uma ou mais tabelas. Na tabela, é possível adicionar, atualizar e excluir dados, enquanto na view essas operações não são possíveis. Em resumo, a tabela armazena dados, enquanto a view armazena uma "query" (consulta).

1.3. Tipos de Views

1.3.1. Views simples

Uma view simples é aquela que se fundamenta em uma única tabela e não inclui junções ou cálculos elaborados, executando apenas as operações básicas. É importante destacar que ela também não armazena dados de forma física.

```
CREATE VIEW V_Produtos_Disponiveis AS
SELECT ProdutoID, Nome, Preço
FROM Produtos
WHERE Disponível = 1;
```

FIGURA 1- EXEMPLO VIEWS SIMPLES

No exemplo acima a view “V_Produtos_Disponiveis” exibe apenas os produtos disponíveis na tabela produtos.

1.3.2. Views complexas

As views complexas abrangem múltiplas tabelas e podem incluir junções, agregações e subconsultas, proporcionando uma visualização mais detalhada. Esse tipo de view é ideal para a elaboração de relatórios e análises mais sofisticadas.

```
CREATE VIEW V_Vendas_Cliente AS
SELECT C.Nome, SUM(V.Valor) AS TotalVendas
FROM Clientes C
JOIN Vendas V ON C.ClienteID = V.ClienteID
GROUP BY C.Nome;
```

FIGURA 2- EXEMPLO VIEWS COMPLEXAS

Nesse exemplo a view “V_Vendas_Cliente” exibe a combinação dos dados da das vendas e dos clientes para permitir a visualização do total de vendas realizadas pelo cliente.

1.3.3. Views materializada

Views materializadas (caso o banco de dados suporte): Quando a view está materializada, significa que houve a cópia dos dados da consulta de forma física.

```
CREATE MATERIALIZED VIEW MV_Vendas_Mensais AS
SELECT MONTH(DataVenda) AS Mês, SUM(Valor) AS Total
FROM Vendas
GROUP BY MONTH(DataVenda);
```

FIGURA 3- EXEMPLO VIEWS MATERIALIZADA

Nesse exemplo a view “M_Vendas_Mensais” armazena o total de vendas realizadas por mês para consultas mais eficientes.

2. VANTAGENS E DESVANTAGENS

2.1. Vantagens

Simplificação de consultas complexas.

Quando você possui consultas complexas (como comparar várias tabelas), pode ser difícil escrever essa consulta sempre que necessário. A view permite reduzir a repetição do código definindo uma consulta complexa uma vez e podendo reutilizá-la diversas vezes apenas colocando o nome da "View". Desse modo, a view permite salvar a consulta e utilizá-la mais facilmente, eliminando a repetição das consultas complexas e tornando o uso delas mais simples e eficaz.

Aumento da segurança ao restringir acesso a certas colunas e linhas.

Ao criar uma view, é permitido definir permissões específicas para essa view. Isto é, você pode restringir o acesso direto às tabelas, permitindo o acesso dos usuários aos dados através da view. Assim, pode-se configurar a view para mostrar apenas algumas informações que são relevantes ao usuário, sem a necessidade de dar acesso a toda a tabela. Dessa forma, o controle sobre os dados reduz o risco de exposição de informações sensíveis, ajudando a proteger a privacidade dos dados e adicionando uma camada extra na segurança dos mesmos.

Facilita o controle e a manutenção de consultas frequentemente usadas.

A utilização da view permite fazer manutenções de forma simplificada, uma vez que, quando criada, você pode usá-la em qualquer lugar do seu banco de dados. Assim, caso a lógica da consulta precise ser alterada, é só atualizar a definição da view, em vez de modificar todas as consultas onde ela é utilizada.

2.2. Desvantagens

Possíveis impactos de desempenho

Dependendo do nível de complexidade da view, o banco de dados precisa executar todas as operações cada vez que ela é consultada, o que pode consumir tempo e recursos do sistema, diminuindo a eficiência da execução.

Limitações em views complexas para operações de atualização.

Existem views complexas que envolvem diversas tabelas. Fazer atualizações nessas views faz com que o processo ocorra de forma mais lenta, uma vez que o banco de dados precisa processar as operações de atualizações em todas as tabelas, o que causa a diminuição do desempenho.

Manutenção de views materializadas (atualização periódica dos dados).

A atualização periódica dos dados pode ser uma desvantagem, uma vez que necessita de planejamento e gestão cuidadosa para que os dados sejam atualizados da maneira correta e de forma eficiente. Além disso, a utilização da atualização regular tende a utilizar diversos recursos, afetando o desempenho geral do sistema e tornando as operações de leitura e escrita mais lentas.

3. PROCESSO DE CRIAÇÃO DE VIEWS NO SQL

3.1. CREATE VIEW: Sintaxe e Parâmetros

```sql
CREATE VIEW nome_da_view AS
SELECT coluna1, coluna2, ...
FROM tabela
WHERE condição;
```

FIGURA 4- EXEMPLO CREATE VIEW

O comando CREATE VIEW é utilizado para criar uma view no banco de dados.

O SELECT é utilizado para selecionar as colunas desejadas

O FROM direciona a tabela onde os dados serão selecionados

O WHERE é uma condição para filtrar os dados

3.2. Exemplos de views simples

3.2.1. View de filtragem:

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

FIGURA 5- EXEMPLO VIEW DE FILTRAGEM

-Possui a view com o nome ClientesAtivos

- Selecionou as colunas nome, email e telefone.
- Demonstra que os dados fazem parte da tabela Clientes
- Filtra os clientes e seleciona apenas os com status “ativo”;

3.3. View de agregação: uso de funções

3.3.1. Agregação com a função SUM

```sql
CREATE VIEW VendasPorProduto AS
SELECT produto_id, SUM(valor_venda) AS total_vendas
FROM Vendas
GROUP BY produto_id;
```

FIGURA 6- EXEMPLO VIEW COM SUM

-Possui a view com o nome VendasPorProdutos

- Selecionou o ID do produto e a soma das vendas.
- Utiliza o comando SUM que é utilizado para calcular a soma total dos valores da coluna
- Demonstra que os dados fazem parte da tabela Vendas
- Agrupa os resultados por ID do produto.

3.4. Agregação com a função AVG

```sql
CREATE VIEW MediaVendasPorCliente AS
SELECT cliente_id, AVG(valor_venda) AS media_vendas
FROM Vendas
GROUP BY cliente_id;
```

FIGURA 7- EXEMPLO VIEW COM FUNCAO AVG

-Possui a view com o nome MediaVendasPorCliente

- Selecionou o as colunas cliente_id e media_vendas
- Utiliza o comando AVG que é utilizado para calcular a média dos valores da coluna
- Demonstra que os dados fazem parte da tabela Vendas
- Agrupa os resultados por ID do cliente

3.5. Agregação com a função COUNT

```sql
CREATE VIEW ContagemPedidosPorCliente AS
SELECT cliente_id, COUNT(pedido_id) AS total_pedidos
FROM Pedidos
GROUP BY cliente_id;
```

FIGURA 8- EXEMPLO VIEWS COM FUNÇÃO COUNT

-Possui a view com o nome ContagemPedidosPorCliente

- Selecionou o as colunas cliente_id e total_pedido
- Utiliza o comando COUNT que é utilizado para contar o id dos pedidos
- Demonstra que os dados fazem parte da tabela Pedidos
- Agrupa os resultados por ID do cliente.

3.6. View de junção

Tabela Clientes

cliente_id	nome
1	João Silva
2	Maria Santos
3	Carlos Pereira

FIGURA 9- TABELA CLIENTE

Tabela Pedidos

pedido_id	cliente_id	data_pedido	valor_pedido
101	1	2024-01-10	500
102	2	2024-01-15	300
103	1	2024-02-10	400

FIGURA 10- TABELA PEDIDOS

Junção das tabelas

```
CREATE VIEW PedidosPorCliente AS
SELECT c.nome, p.pedido_id, p.data_pedido, p.valor_pedido
FROM Clientes c
JOIN Pedidos p ON c.cliente_id = p.cliente_id;
```

FIGURA 11- JUNÇÃO DAS TABELAS

-Possui a view com o nome PedidosPorCliente;

- Selecionou as colunas c.nome, p.pedido_id, p.data_pedido e p.valor_pedido;
- A tabela Clientes é demostrada como c;
- Demonstra que os dados fazem parte da tabela Pedidos;
- Realiza a junção das tabelas Clientes e Pedidos com base campo cliente_id;
- Utiliza o JOIN para combinar dados das tabelas;
- O ON especifica a condição que relaciona as tabelas.

4. VIEWS ATUALIZÁVEIS E NÃO ATUALIZÁVEIS

Views atualizáveis podem receber declarações de atualização, como UPDATE e DELETE, que irão alterar as tabelas base, entretanto essas não podem ser views de agregação, por exemplo. Isto porque views atualizáveis são obrigatoriamente baseadas em uma única tabela, para que sua relação entre as colunas da view e das colunas da tabela base seja uma relação direta.

Exemplo:

```sql
CREATE VIEW V_Funcionarios AS
SELECT id, nome, salario
FROM Funcionarios;
```

FIGURA 12- EXEMPLOS ATUALIZACAO DE DADOS

- View simples sem agregações.

```sql
UPDATE Funcionarios
SET salario = 5000
WHERE id = 1;
```

FIGURA 13- EXEMPLO UPDATE

- UPDATE que pode ser feito nela.

```sql
CREATE VIEW V_Agregada AS
SELECT departamento, AVG(salario) AS salario_medio
FROM Funcionarios
GROUP BY departamento;
```

FIGURA 14- EXEMPLO AGREGAÇÃO

- View com agregações, ou seja, não atualizável.

5. Estudo de caso

5.1. Banco de Dados do Sistema de RH.

```
CREATE DATABASE SistemaRH;
USE SistemaRH;

CREATE TABLE Cargos (
    cargo_id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL
);
CREATE TABLE Funcionarios (
    funcionario_id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    sobrenome VARCHAR(100) NOT NULL,
    data_nascimento DATE,
    email VARCHAR(100) UNIQUE,
    telefone VARCHAR(20),
    cargo_id INT,
    FOREIGN KEY (cargo_id) REFERENCES Cargos(cargo_id)
);
CREATE TABLE Salarios (
    salario_id INT PRIMARY KEY AUTO_INCREMENT,
    funcionario_id INT,
    salario DECIMAL(10, 2) NOT NULL,
    data_inicio DATE NOT NULL,
    data_fim DATE,
    FOREIGN KEY (funcionario_id) REFERENCES Funcionarios(funcionario_id)
);
```

```
-- Inserir cargos
INSERT INTO Cargos (nome)
 VALUES
('Secretário'),
('Analista de RH'),
('Contador');

-- Inserir funcionários
INSERT INTO Funcionarios (nome, sobrenome, data_nascimento, telefone,
cargo_id)
VALUES
('João', 'Silva', '1985-01-15', '1111-1111', 1),
('Maria', 'Oliveira', '1990-05-25', '2222-2222', 2),
('Ana', 'Souza', '1982-03-30', '3333-3333', 3);

-- Inserir salários
INSERT INTO Salarios (funcionario_id, salario, data_inicio)
VALUES
(1, 5000.00, '2023-01-01'),
(2, 4500.00, '2023-01-01'),
(3, 5500.00, '2023-01-01');

-- View da folha de pagamento
CREATE VIEW Detalhes_Funcionarios_Simples AS
SELECT
    f.nome,
    f.sobrenome,
    c.nome AS cargo,
    s.salario
FROM
    Funcionarios f
JOIN
    Cargos c ON f.cargo_id = c.cargo_id
JOIN
```

```
    Salarios s ON f.funcionario_id = s.funcionario_id
WHERE
    s.data_fim IS NULL;
```

Esta view não é atualizável, pois é uma view que agrega as tabelas Funcionários, Cargos e Salários e não há como estabelecer uma relação direta entre a colunas da view e as colunas de uma tabela base.

Como folha de pagamento, não é necessário que a view seja atualizável, pois consta mais como um relatório do que algum tipo de documento que precise ser atualizado. Portanto, uma folha de pagamento só tem necessidade de mudanças fundamentais, que podem ser realizadas diretamente nas tabelas.

CONCLUSÃO

Em conclusão, as SQL Views são um componente importantes na arquitetura dos sistemas de banco de dados relacionais, oferecendo uma maneira mais eficinete de manusear apresentar dados. Ao funcionarem como tabelas virtuais, as views não apenas simplificam a consulta a dados complexos, mas também eliminam a necessidade de duplicação, promovendo uma gestão mais organizada e ágil das informações.

Ademais, a pesquisa traz a importância das views na redução do tempo e na melhoria da segurança dos dados, permitindo que os usuários acessem apenas as informações relevantes sem comprometer o acesso a dados sensíveis. Além disso, a flexibilidade proporcionada pelas views simples, complexas e materializadas facilita a criação de relatórios e a execução de análises detalhadas.

Dessa maneira, fica evidente que as SQL Views não são apenas ferramentas fundamentais para a construção de um ambiente de dados seguro, e organizado mantendo assim a eficiência e facilidade na manipulação dos dados.

BIBLIOGRAFIA

- YouTube. Disponível em: <https://youtu.be/MOXHM8tdiYU?si=-L9MLd-wTXB5prWN>. Acesso em: 13 nov. 2024.

ANDRE, T. **Views no MySQL: Criação, update e exclusão - Central de Ajuda KingHost**. Disponível em: <https://king.host/wiki/artigo/views-mysql/>. Acesso em: 13 nov. 2024.

Atualização de Views e Teoria Relacional. Disponível em: <https://issuu.com/novateceditora/docs/capitulo9788575224885>. Acesso em: 13 nov. 2024.

BIANCHI, W. **Trabalhando com Views**. Disponível em: <https://www.devmedia.com.br/mysql-trabalhando-com-views/8724>. Acesso em: 13 nov. 2024.

DA INFORMÁTICA, R. **VIEWS em SQL: Vantagens e desvantagens**. Disponível em: <https://ramosdainformatica.com.br/views-em-sql-vantagens-e-desvantagens/>. Acesso em: 13 nov. 2024.

DAMIN, D. **Linguagem SQL: Capítulo 7 - Views**. Disponível em: <https://kb.elipse.com.br/linguagem-sql-capitulo-7-views/>. Acesso em: 13 nov. 2024.

MIZEVSKI, H. **Views, Sequences e Synonyms: Um tutorial prático**. Disponível em: <https://dev.to/mizevski/views-sequences-e-synonyms-um-tutorial-pratico-379d>. Acesso em: 13 nov. 2024.

O que são views em SQL? Quais vantagens e desvantagens em utilizar? Disponível em: <https://pt.stackoverflow.com/questions/35413/o-que-s%C3%A3o-views-em-sql-quais-vantagens-e-desvantagens-em-utilizar>. Acesso em: 13 nov. 2024.

Tivoli Netcool/OMNIbus 8.1.0. Disponível em: <https://www.ibm.com/docs/pt-br/netcoolomnibus/8.1?topic=reference-create-view-command>. Acesso em: 12 nov. 2024.

Visão em SQL. Disponível em: <https://sae.unb.br/cae/conteudo/unbfga/lbd/banco2_visoes.html>. Acesso em: 13 nov. 2024.

BIBLIOTECA DE IMAGENS