SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Curso

TÉCNICO EM DESENVOLVIMENTO

DE SISTEMAS

SQL Views

Thafany Santos Passos

Sorocaba
Nov – 2024

SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Thafany Santos Passos

SQL Views

Esse trabalho se refere ao
registro da pesquisa acerca do
SQL Views
Prof. – Emerson Magalhães

Sorocaba
Nov – 2024

SUMÁRIO

INTRODUÇÃO

As views, ou visões em português, são consultas armazenadas como tabelas virtuais. Elas permitem que os usuários acessem e interajam com os dados de maneira simplificada, fornecendo uma camada de abstração sobre as tabelas subjacentes. As views não armazenam dados por si mesmas; em vez disso, elas extraem informações de outras tabelas ou views.

As SQL Views desempenham um papel fundamental em sistemas de banco de dados, atuando como uma camada de abstração que simplifica o acesso e a manipulação dos dados. Uma view é uma "tabela virtual" que permite consultar dados de uma ou mais tabelas de forma organizada e estruturada, sem necessidade de duplicação ou alteração dos dados originais. Com isso, as views facilitam a criação de consultas complexas, promovem a segurança ao limitar o acesso a informações sensíveis e ajudam a padronizar consultas utilizadas por diferentes usuários ou aplicativos.

Em sistemas de banco de dados relacionais, as views são essenciais para otimizar o acesso a dados, melhorar o desempenho e simplificar o desenvolvimento de aplicações. Elas são especialmente úteis em cenários em que o acesso a dados precisa ser controlado, como quando se deseja fornecer uma visão específica de um conjunto de informações para diferentes departamentos ou níveis de permissão de usuários. Além disso, views ajudam a reduzir a repetição de código ao encapsular consultas frequentemente utilizadas, o que torna a manutenção do banco de dados mais eficiente.

O objetivo desta pesquisa é explorar os conceitos, tipos e aplicações das SQL Views em projetos de banco de dados, destacando suas vantagens e mostrando exemplos práticos em contextos comuns, como em um sistema de e-commerce e um sistema de RH. A pesquisa abrange as melhores práticas na criação e gerenciamento de views, bem como suas implicações em segurança, desempenho e organização dos dados. Ao final, busca-se demonstrar como as views, quando bem utilizadas, agregam valor ao projeto de banco de dados, tornando-o mais eficiente, seguro e sustentável a longo prazo.

1. FUNDAMENTOS TEÓRICOS DA SQL VIEWS

Uma view em SQL é uma maneira alternativa de observação de dados de uma ou mais entidades (tabelas), que compõem uma base de dados. Pode ser considerada como uma tabela virtual ou uma consulta armazenada.

View é um resultado originado de uma consulta pré-definida. Essencialmente é um metadado que mapeia uma *query* para outra, por isto pode ser considerado como uma tabela virtual. Como o próprio nome diz, ela representa uma visão de dados e não contém dados. Com ela você tem a ilusão que está vendo uma tabela que não existe.

É basicamente um texto de um SELECT. Pense que ela é uma query que você armazena em uma variável e ao invés de escrever toda a query de novo sempre que for necessária usa a variável.

Ou seja, A view pode ser definida como uma tabela virtual composta por linhas e colunas de dados vindos de tabelas relacionadas em uma query (um agrupamento de SELECT's, por exemplo). As linhas e colunas da view são geradas dinamicamente no momento em que é feita uma referência a ela.

Elas não são carregadas, elas são geradas sempre que forem necessárias.

A não ser que a view seja materializada, aí você terá tabelas reais representando essas views. Mas a carga delas ocorrerão conforme a necessidade e otimizações do banco de dados. Só neste tipo de view é que os dados são atualizados quando as tabelas reais sofrem atualizações

Como já dito, a query que determina uma view pode vir de uma ou mais tabelas, ou até mesmo de outras views.

Ao criarmos uma view, podemos filtrar o conteúdo de uma tabela a ser exibida, já que a função da view é exatamente essa: filtrar tabelas, servindo para agrupá-las, protegendo certas colunas e simplificando o código de programação.

É importante salientar que, mesmo após o servidor do SQL Server ser desligado, a view continua "viva" no sistema, assim como as tabelas que criamos normalmente. As views não ocupam espaço no banco de dados.

Uma view é muito usada para ajudar dar entendimento do projeto lógico do banco de dados.

1.1. Diferença entre views e tabels comuns:

Uma tabela é um objeto formado por linhas e colunas que armazenam dados que podem ser consultados sempre que quisermos,

então vai ter as informações dispostas em linhas e colunas para facilitar a visualização e criar sua análise de dados.

Já as Views no SQL são a possibilidade de armazenar o resultado de um query dentro de uma tabela virtual que pode facilitar a utilização delas posteriormente.

1.2. TIPOS DE VIEWS

Simple View (View Simples):

A Simple View é a forma mais básica de view. Ela é criada a partir de uma única tabela e contém apenas uma única consulta SELECT. Essas views são úteis para oferecer uma visão resumida ou segmentada dos dados originais.

```sql
CREATE VIEW exemplo_simple_view AS
SELECT coluna1, coluna2
FROM tabela_origem
WHERE coluna3 = 'valor';
```

Complex View (View Complexa):

A Complex View é uma view que pode ser criada a partir de várias tabelas, usando joins, funções agregadas ou subconsultas. Isso permite que os usuários obtenham resultados consolidados ou personalizados a partir de várias fontes de dados. Essas views são especialmente úteis para simplificar consultas

complexas e fornecer uma visão abrangente dos dados.

```
CREATE VIEW exemplo_complex_view AS
SELECT t1.coluna1, t2.coluna2
FROM tabela1 t1
JOIN tabela2 t2 ON t1.chave = t2.chave
WHERE t1.coluna3 = 'valor';
```

Materialized View (View Materializada):

As Materialized Views são views que armazenam fisicamente os dados em disco. Isso permite que os resultados da consulta sejam pré-calculados e atualizados periodicamente, reduzindo a carga do servidor e melhorando o desempenho em consultas repetitivas. Essas views são ideais para consultas com alto consumo de recursos ou que envolvam agregações complexas.

```
CREATE MATERIALIZED VIEW exemplo_materialized_view
REFRESH COMPLETE ON DEMAND
AS
SELECT coluna1, COUNT(coluna2) AS total
FROM tabela_origem
GROUP BY coluna1;
```

2. VANTAGENS DAS VIEWS

Temos muitos motivos e vantagens para usarmos views em nossos projetos. A seguir são citados três que podem fazer a diferença:

- Reuso: as views são objetos de caráter permanente. Pensando pelo lado produtivo isso é excelente, já que elas podem ser lidas por vários usuários simultaneamente.
- Segurança: as views permitem que ocultemos determinadas colunas de uma tabela. Para isso, basta criarmos uma view com as colunas que acharmos necessário que sejam exibidas e as disponibilizarmos para o usuário.
- Simplificação do código: as views nos permitem criar um código de programação muito mais limpo, na medida em que podem conter

um SELECT complexo. Assim, criar views para os programadores a fim de poupá-los do trabalho de criar SELECT's é uma forma de aumentar a produtividade da equipe de desenvolvimento.

- Você pode usar este resultado em outras consultas diminuindo a complexidade, afinal você fará referência a uma tabela virtual montada fora desta consulta. De uma certa forma podemos considerar como syntax sugar para uma query. Você tem uma visão mais limitada dos dados sem grandes preocupações.
- Estas consultas pré-definidas ficam armazenadas e você não precisa lembrar de como criá-las. Não confundir com resultados.
- Como o próprio nome ajuda identificar elas permitem criar uma visão mais lógica para um humano entender a modelagem.
- Facilita a troca do modelo físico sem o perigo de quebrar queries existentes.
- Você pode colocar permissões na view, ou seja, você pode proibir acesso à tabelas em seu estado bruto, mas em uma certa condição o usuário pode ter acesso à informação da forma como você definiu na view. Você controla melhor o que e como o usuário pode acessar a informação.
- Se a view for materializada pode ter ganho de performance para o acesso aos dados já "consolidados".

3. DESVANTAGENS DAS VIEWS

- Esconde uma complexidade da query podendo enganar o desenvolvedor quanto à performance necessária para acessar determinada informação. E pode ser pior quando views usam outras views. Em alguns casos você pode estar fazendo consultas desnecessárias sem saber de forma muito intensiva.
- Cria uma camada extra. Mais objetos para administrar. Algumas pessoas consideram isto um aumento de complexidade. Uma outra forma de ver isto é que uma view pode ser mal usada.
- Pode limitar exageradamente o que o usuário pode acessar impedindo certas tarefas.

- Se a view for materializada fará com que alterações nas tabelas reais envolvidas sejam mais lentas, afinal são mais tabelas para atualizar

4. CRIANDO UMA VIEW

Para criar uma view é muito simples: vamos levar em conta a tabela Produtos, conforme a **Figura** ilustra.

Id	Nome	Fabricante	Quantidade	VlUnitario	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
4	GT-I6220 Quad Band	Samsung	300.00	499.00	Celular
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
6	Playstation 2	Sony	100.00	399.00	Console
7	Sofá Estofado	Coréia	200.00	499.00	Sofá
8	Armário de Serviço	Aracaju	50.00	129.00	Armário
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

Tendo a estrutura da tabela acima em mente, vamos criar a view, como mostra o exemplo.

```
1  CREATE VIEW vwProdutos AS
2  SELECT IdProduto AS Código,
3         Nome AS Produto,
4         Fabricante,
5         Quantidade,
6         VlUnitario AS [ValorUnitario],
7         Tipo
8  FROM Produtos
```

Para **consultarmos os dados na view usamos** o comando **SELECT**, da mesma forma que se estivéssemos fazendo uma consulta em uma tabela comum.

A próxima imagem exibe a consulta na view:

```
1  SELECT * FROM vwProdutos
```

E esse é o resultado dessa consulta:

Código	Produto	Fabricante	Quantidade	Valor Unitário	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
4	GT-I6220 Quad Band	Samsung	300.00	499.00	Celular
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
6	Playstation 2	Sony	100.00	399.00	Console
7	Sofá Estofado	Coréia	200.00	499.00	Sofá
8	Armário de Serviço	Aracaju	50.00	129.00	Armário
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

Com a view podemos incluir dados em uma tabela. Para isso é necessário que haja uma das seguintes situações: as colunas da tabela de origem que não são exibidas na view devem aceitar valores nulos, ser auto incrementais ou ter um valor padrão definido para elas.

Quando realizamos um INSERT, UPDATE ou DELETE dos dados de uma tabela de origem da view, essa ação se reflete automaticamente na view.

4.1. View de Filtragem:

A filtragem é uma técnica para selecionar apenas as linhas de dados que atendem a certos critérios, ignorando as que não são relevantes. A cláusula WHERE no SQL é frequentemente usada para aplicar esse tipo de filtro.

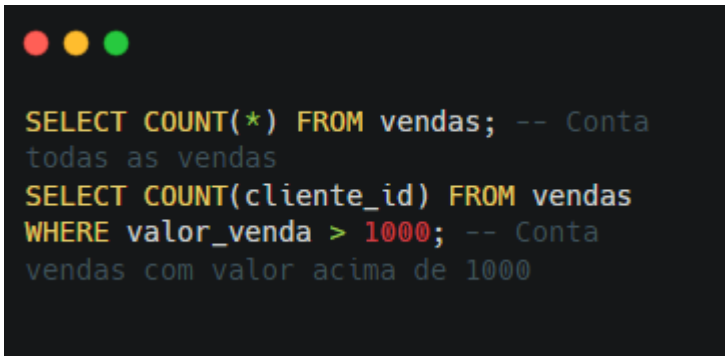
```sql
CREATE VIEW vendas_ult_mes AS
SELECT * FROM vendas
WHERE data_venda >= '2024-10-01';
```

Nesse exemplo temos um banco de dados de vendas, e foi criado uma view que mostre apenas as vendas realizadas no último mês.

4.2. View de Agregação:

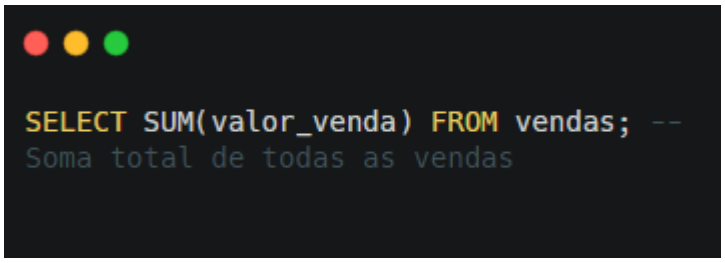
Permite agrupar dados e calcular estatísticas ou somatórios sobre esses grupos. Usada para resumir dados e gerar métricas. Com o SQL, usa-se a cláusula `GROUP BY` para criar agregações. Assim, há algumas funções que permitem a realização da agregação:

COUNT: A função `COUNT` conta o número de linhas em um conjunto de dados. Pode contar todas as linhas ou apenas as linhas que atendem a uma condição específica. É útil para saber quantos registros correspondem a certos critérios.



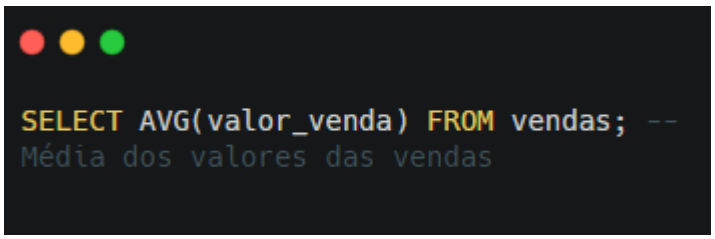
```sql
SELECT COUNT(*) FROM vendas; -- Conta todas as vendas
SELECT COUNT(cliente_id) FROM vendas WHERE valor_venda > 1000; -- Conta vendas com valor acima de 1000
```

SUM: calcula o total (soma) de valores em uma coluna numérica. É usado para somar todos os valores em um grupo ou conjunto de dados.



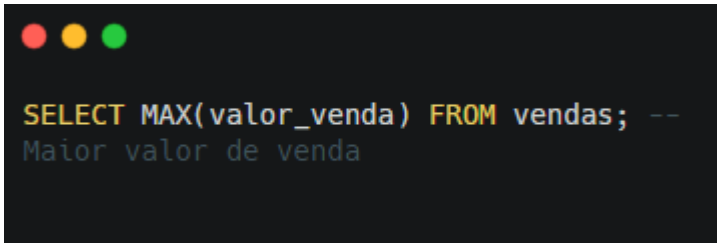
```sql
SELECT SUM(valor_venda) FROM vendas; -- Soma total de todas as vendas
```

AVG: A função `AVG` calcula a média dos valores de uma coluna numérica. É útil para obter o valor médio de um conjunto de dados.



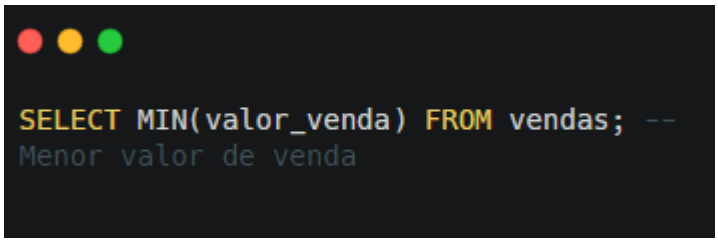
```sql
SELECT AVG(valor_venda) FROM vendas; -- Média dos valores das vendas
```

MAX: Essa função retorna o maior valor em uma coluna. Usada para encontrar o valor máximo dentro de um conjunto de dados, como a venda de maior valor.

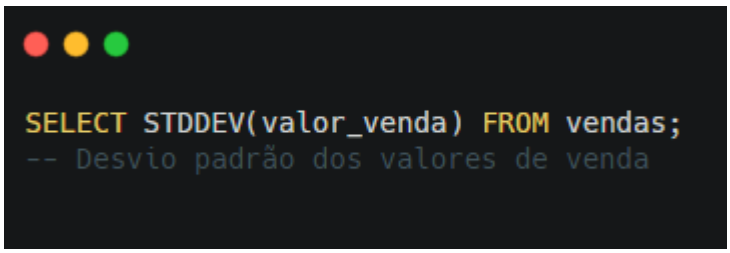


```sql
SELECT MAX(valor_venda) FROM vendas; -- Maior valor de venda
```

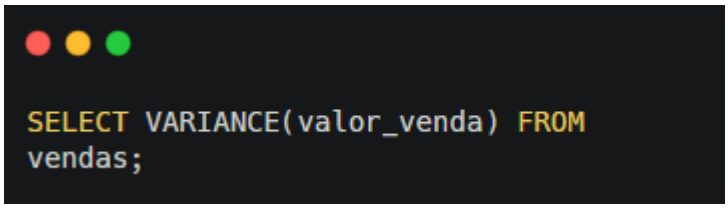
MIN: Retorna o menor valor em uma coluna, útil para encontrar o valor mínimo.



STDDEV: Essa função é responsável por calcular o desvio padrão dos valores de uma coluna numérica, que mede a dispersão dos dados em relação à média.



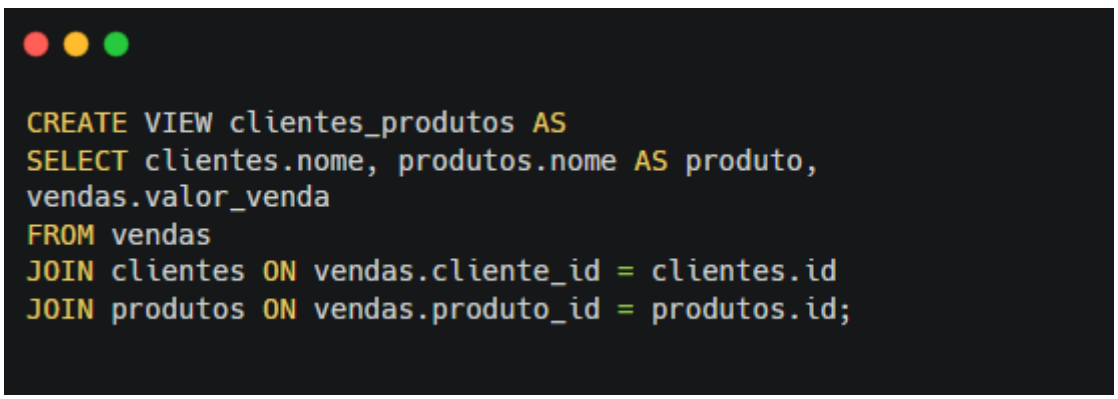
VARIANCE: Calcula a variância dos valores em uma coluna numérica, que representa o grau de dispersão dos dados. A variância é o quadrado do desvio padrão e oferece uma ideia sobre a distribuição dos dados.



4.3. View de Junção:

A junção (ou JOIN) permite combinar dados de duas ou mais tabelas com base em uma coluna comum.

Por exemplo, para criar uma view que mostra o nome dos clientes e os produtos que compraram, é possível juntar a tabela de clientes com a de vendas:



4.4. Alterando uma View

O comando ALTER VIEW é utilizado para atualizar uma view, após ela já ter sido criada e necessitar de alterações. . Seguindo o exemplo da view criada anteriormente, vamos alterá-la para que exiba apenas os produtos cujo valor unitário seja maior que 499,00. Para isso, devemos usar o seguinte código:

```
ALTER VIEW vwProdutos AS
SELECT IdProduto AS Código,
       Nome AS Produto,
       Fabricante,
       Quantidade,
       VlUnitario AS [ValorUnitario],
       Tipo
FROM Produtos
WHERE VlUnitario > 499.00
```

E o resultado será:

Código	Produto	Fabricante	Quantidade	Valor Unitário	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

4.5. Excluindo uma View

Para excluirmos uma view é bem simples: é só usar o comando DROP VIEW, como podemos ver a seguir:

```
DROP VIEW vwProdutos
```

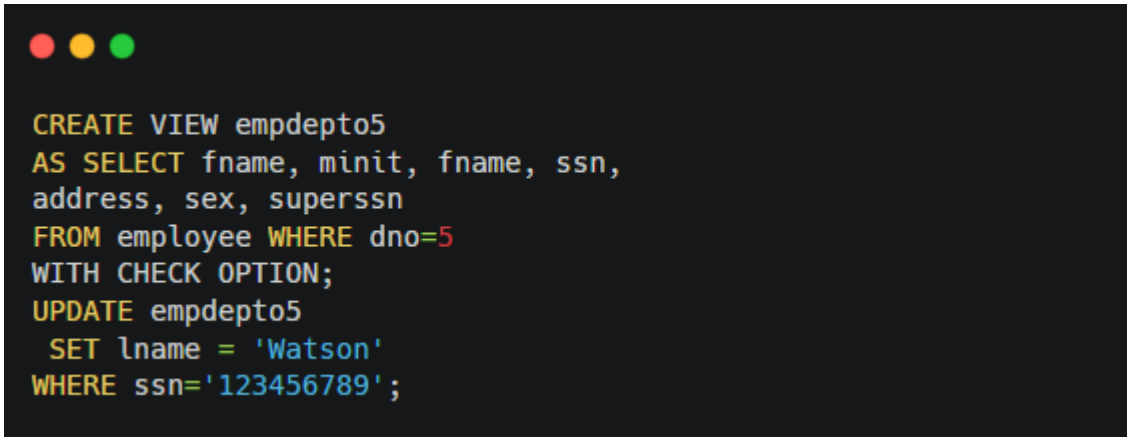
A exclusão de uma view implica na exclusão de todas as permissões que tenham sido dadas sobre ela. Dito isso, devemos usar o comando DROP VIEW apenas quando desejamos de fato retirar a view do sistema. Em caso contrário, podemos usar o comando ALTER VIEW.

5. VIEWS ATUALIZÁVEIS E NÃO ATUALIZÁVEIS

Views Atualizáveis:

As views são chamadas atualizáveis se permitem aos usuários realizarem alterações nos dados do banco de dados por meio dala e para isso a visão deve ser definida como atualizável.

Exemplo:



```
CREATE VIEW empdepto5
AS SELECT fname, minit, fname, ssn,
address, sex, superssn
FROM employee WHERE dno=5
WITH CHECK OPTION;
UPDATE empdepto5
 SET lname = 'Watson'
WHERE ssn='123456789';
```

OBS: observe que nesse exemplo a chave primária faz parte da visão, o que facilita a atualização do banco de dados.

Restrições para visões atualizáveis:

Em geral, para ser atualizável a visão não deve conter:

1. **Junção;**
2. **Função de agregação;**
3. **Subconsultas com tabela na cláusula FROM;**

Em geral, para ser atualizável, a visão deve ser derivada de apenas uma tabela base e deve conter a chave primária da tabela

Views Não Atualizáveis:

Uma view não atualizável é uma view em um banco de dados que não permite alterações diretas nos dados subjacentes, ou seja, não se pode realizar operações de INSERT, UPDATE ou DELETE diretamente nela. Esse tipo de view é apenas para consulta, sendo ideal para cenários em que se precisa apenas visualizar dados agregados, resultados de cálculos complexos ou combinações de múltiplas tabelas.

Características das Views Não Atualizáveis:

1. **Incluem Funções Agregada**

2. **Uso de Joins Complexos**: Quando uma view é criada a partir de junções complexas entre várias tabelas

3. **Subconsultas**: Se a view inclui subconsultas ou operadores como UNION, que combina resultados de múltiplas consultas, a view resultante também será não atualizável.

6. ESTUDO DE CASO

Estrutura do banco de dados fictício de uma loja ecommerce:

Nesse caso teríamos as seguintes tabelas:

1. Produtos:
    - produto_id (INT, PK)
    - nome (VARCHAR)
    - preco (DECIMAL)
    - estoque (INT)
    - categoria (VARCHAR)
2. Clientes
    - cliente_id (INT, PK)
    - nome (VARCHAR)
    - email (VARCHAR)
    - cidade (VARCHAR)
    - telefone (VARCHAR)
3. Pedidos
    - pedido_id (INT, PK)
    - cliente_id (INT)
    - data_pedido (DATE)
    - valor_total (DECIMAL)
4. Itens_Pedido
    - item_id (INT, PK)
    - pedido_id (INT)
    - produto_id (INT)

- quantidade (INT)
- preco_unitario (DECIMAL)

5. Funcionarios (Exemplo de sistema de RH)
   - funcionario_id (INT, PK)
   - nome (VARCHAR)
   - cargo (VARCHAR)
   - salario (DECIMAL)
   - data_admissao (DATE)

Exemplos de Views para o Banco de Dados:

```
mysql> CREATE DATABASE Loja_Ecommerce;
Query OK, 1 row affected (4.39 sec)

mysql> USE Loja_Ecommerce;
Database changed
mysql>
mysql> CREATE TABLE Produtos (
    ->     produto_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     nome VARCHAR(100) NOT NULL,
    ->     preco DECIMAL(10, 2) NOT NULL,
    ->     estoque INT NOT NULL,
    ->     categoria VARCHAR(50) NOT NULL
    -> );
Query OK, 0 rows affected (3.56 sec)

mysql>
mysql> CREATE TABLE Clientes (
    ->     cliente_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     nome VARCHAR(100) NOT NULL,
    ->     email VARCHAR(100) NOT NULL,
    ->     cidade VARCHAR(50) NOT NULL,
    ->     estado VARCHAR(2) NOT NULL
    -> );
Query OK, 0 rows affected (0.75 sec)
```

```
mysql>
mysql> CREATE TABLE Pedidos (
    ->     pedido_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     cliente_id INT NOT NULL,
    ->     data_pedido DATE NOT NULL,
    ->     valor_total DECIMAL(10, 2) NOT NULL,
    -> );
Query OK, 0 rows affected (0.63 sec)

mysql>
mysql> CREATE TABLE Itens_Pedido (
    ->     item_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     pedido_id INT NOT NULL,
    ->     produto_id INT NOT NULL,
    ->     quantidade INT NOT NULL,
    ->     preco_unitario DECIMAL(10, 2) NOT NULL,
Query OK, 0 rows affected (0.86 sec)

mysql>
mysql> CREATE TABLE Funcionarios (
    ->     funcionario_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     nome VARCHAR(100) NOT NULL,
    ->     cargo VARCHAR(50) NOT NULL,
    ->     salario DECIMAL(10, 2) NOT NULL,
    ->     data_admissao DATE NOT NULL
    -> );
Query OK, 0 rows affected (0.49 sec)
```

```
mysql> INSERT INTO Produtos (nome, preco, estoque, categoria)
VALUES
    -> ('Notebook', 2500.00, 10, 'Eletrônicos'),
    -> ('Celular', 1500.00, 15, 'Eletrônicos'),
    -> ('Camiseta', 30.00, 50, 'Roupas'),
    -> ('Calça', 80.00, 30, 'Roupas'),
    -> ('Tênis', 120.00, 20, 'Calçados');
Query OK, 5 rows affected (0.58 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql>
mysql> INSERT INTO Clientes (nome, email, cidade, estado)
VALUES
    -> ('Alice', 'alice@email.com', 'São Paulo', 'SP'),
    -> ('Bruno', 'bruno@email.com', 'Rio de Janeiro', 'RJ'),
    -> ('Carla', 'carla@email.com', 'Belo Horizonte', 'MG');
Query OK, 3 rows affected (0.36 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql> INSERT INTO Pedidos (cliente_id, data_pedido,
valor_total) VALUES

    -> (1, '2024-11-01', 2530.00),
    -> (2, '2024-11-02', 1530.00),
    -> (3, '2024-11-03', 30.00);
Query OK, 3 rows affected (0.20 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql>
mysql> INSERT INTO Itens_Pedido (pedido_id, produto_id,
quantidade, preco_unitario) VALUES
    -> (1, 1, 1, 2500.00),
    -> (1, 3, 1, 30.00),
    -> (2, 2, 1, 1500.00),
    -> (3, 3, 1, 30.00);
Query OK, 4 rows affected (0.16 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql>
mysql> INSERT INTO Funcionarios (nome, cargo, salario,
data_admissao) VALUES
    -> ('Daniel', 'Gerente', 5000.00, '2020-01-10'),
    -> ('Elisa', 'Vendedor', 2500.00, '2021-06-20'),
    -> ('Fernando', 'Vendedor', 2500.00, '2022-03-15');
Query OK, 3 rows affected (0.06 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CREATE VIEW relatorio_vendas_mensal AS
    -> SELECT
    ->     YEAR(data_pedido) AS ano,
    ->     MONTH(data_pedido) AS mes,
    ->     SUM(valor_total) AS total_vendas
    -> FROM
    ->     Pedidos
    -> GROUP BY
    ->     YEAR(data_pedido),
    ->     MONTH(data_pedido);
Query OK, 0 rows affected (0.24 sec)
```

```
mysql> CREATE VIEW consulta_estoque AS
    -> SELECT
    ->     produto_id,
    ->     nome AS produto,
    ->     estoque,
    ->     categoria
    -> FROM
    ->     Produtos
    -> WHERE
    ->     estoque > 0;
Query OK, 0 rows affected (0.20 sec)

mysql> CREATE VIEW folha_pagamento_por_cargo AS
    -> SELECT
    ->     cargo,
    ->     SUM(salario) AS total_salarios,
    ->     COUNT(funcionario_id) AS numero_funcionarios
    -> FROM
    ->     Funcionarios
    -> GROUP BY
    ->     cargo;
Query OK, 0 rows affected (0.18 sec)
```

```
mysql> -- Verificar a view de relatório de vendas mensal
mysql> SELECT * FROM relatorio_vendas_mensal;
+------+------+--------------+
| ano  | mes  | total_vendas |
+------+------+--------------+
| 2024 |   11 |      4090.00 |
+------+------+--------------+
1 row in set (0.21 sec)

mysql>
mysql> -- Verificar a view de consulta de estoque
mysql> SELECT * FROM consulta_estoque;
+------------+----------+---------+-------------+
| produto_id | produto  | estoque | categoria   |
+------------+----------+---------+-------------+
|          1 | Notebook |      10 | Eletrônicos |
|          2 | Celular  |      15 | Eletrônicos |
|          3 | Camiseta |      50 | Roupas      |
|          4 | Calça    |      30 | Roupas      |
|          5 | Tênis    |      20 | Calçados    |
+------------+----------+---------+-------------+
5 rows in set (0.04 sec)

mysql>
mysql> -- Verificar a view de folha de pagamento por cargo
mysql> SELECT * FROM folha_pagamento_por_cargo;
+----------+----------------+---------------------+
| cargo    | total_salarios | numero_funcionarios |
+----------+----------------+---------------------+
| Gerente  |        5000.00 |                   1 |
| Vendedor |        5000.00 |                   2 |
+----------+----------------+---------------------+
2 rows in set (0.03 sec)
```

CONCLUSÃO

Este trabalho abordou o conceito de SQL Views, suas funcionalidades, tipos e exemplos de uso em cenários práticos de banco de dados, como em uma aplicação de e-commerce e um sistema de RH. As views foram exploradas como estruturas que simplificam e organizam o acesso a dados, permitindo consultas otimizadas e de fácil manutenção. Foram apresentadas views para relatórios de vendas, consulta de estoque e folha de pagamento, demonstrando a aplicação dessas estruturas em diferentes contextos e suas vantagens em segurança, desempenho e padronização de consultas.

BIBLIOGRAFIA

SILVA, G. **Guia das Views no Banco de Dados Oracle: Tipos, Funcionalidades e Exemplos Práticos**. Disponível em: <https://www.profissionaloracle.com.br/2023/07/29/guia-das-views-no-banco-de-dados-oracle-tipos-funcionalidades-e-exemplos-praticos/>. Acesso em: 13 nov. 2024.

DE CAMARGO, W. B. **Conceitos e criação de views no SQL Server**. Disponível em: <https://www.devmedia.com.br/conceitos-e-criacao-de-views-no-sql-server/22390>. Acesso em: 13 nov. 2024.

Diferença de Tabelas e Views no SQL – Diferenças e Vantagens. Disponível em: <https://www.hashtagtreinamentos.com/diferenca-tabelas-e-views-no-sql>. 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.

DA SILVA ILMERIO ARROBBA UFU. BBR WWW. FACOM. UFU. BBR/~ILMERIO/GES, I. R. **GES013 - Sistema de Banco de Dados SQL/DDL - Visões**. Disponível em: <https://www.facom.ufu.br/~ilmerio/GES013/estSbd8sql_DDL_parte2_Visoes.pdf>. Acesso em: 13 nov. 2024.