awari

SQL Avançado

Data Analytics | Aula 8

Views

Definição de View no SQL

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.

A query que **determina uma view** pode vir de uma ou mais tabelas, ou até mesmo de outras views.

Podemos **realizar qualquer query por meio de views**, assim como alterar dados por meio delas, o que é feito com algumas restrições.

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.

Vantagens das Views

Existem vários 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.

Criando uma View - Parte 1

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

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

Criando uma View - Parte 2

Tendo a estrutura da tabela anterior em mente, vamos criar a view, como mostra o **código abaixo**

```
CREATE VIEW vwProdutos AS
SELECT
     IdProduto AS Código,
     Nome AS Produto,
     Fabricante,
     Quantidade,
     VlUnitario AS [ValorUnitario],
     Tipo
FROM Produtos
```

Criando uma View - Parte 3

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

O **código abaixo** exibe a consulta na view, e a **figura** nos exibe o resultado desta consulta.

```
SELECT * FROM vwProdutos
```

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

Alterando uma View - Parte 1

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, exibido na **Listagem 3**.

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

Alterando uma View - Parte 2

O resultado da alteração pode ser visto na **figura abaixo**:

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

Excluindo uma View

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

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

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 para alterarmos o código da view da forma que acharmos mais conveniente.

HAVING

Cláusula HAVING

A cláusula **HAVING** tem funcionamento muito parecido ao funcionamento das cláusulas de restrição **WHERE**, porém, com uma grande diferença. Enquanto a cláusula **WHERE** é utilizada para filtrar informações baseadas em campos das tabelas, a cláusula HAVING é utilizada para filtrar informações baseadas em resultados das funções de grupo (SUM, AVG, MAX, MIN e COUNT).

Exemplo - HAVING

O código abaixo seleciona dos registros da tabela **pcnfsaid** os campos **codcli** e a soma do campo **vltotal**, agrupando as informações pelo campo **codcli**.

As informações só serão exibidas se o valor da soma do campo **vltotal** for **maior que 10000**.

```
select
    codcli,
    sum(vltotal)
from pcnfsaid
group by codcli
having sum(vltotal)  >
10000;
```

COALESCE

Definição e Exemplo

A função **COALESCE** na linguagem SQL apresenta a primeira expressão não-NULL entre os seus argumentos.

EXEMPLO: Se temos a tabela CONTACT_INFO e quisermos descobrir a melhor forma de contactar cada pessoa de acordo com as seguintes regras:

1. Se uma pessoa possuir um telefone da empresa, utilizar o número de telefone da empresa.
2. Se uma pessoa não possuir um telefone da empresa e possuir um telemóvel, utilizar o número do telemóvel.
3. Se uma pessoa não possuir um telefone da empresa, nem telemóvel, mas possuir telefone em casa, utilizar o número de telefone de casa.

Name	Business_Phone	Cell_Phone	Home_Phone
Jeff	531-2531	622-7813	565-9901
Laura	NULL	772-5588	312-4088
Peter	NULL	NULL	594-7477

```
SELECT
     Name,
     COALESCE (Business_Phone, Cell_Phone, Home_Phone) as Contact_Phone
FROM Contact_Info;
```

RESULTADO:

Name	Contact_Phone
Jeff	531-2531
Laura	772-5588
Peter	594-7477

NULLIF

Definição

A função **NULLIF** requer dois argumentos. Se os dois argumentos forem iguais, será obtido NULL. Caso contrário, é apresentado o primeiro argumento.

EXEMPLO: suponhamos que temos uma tabela que registra as vendas atuais e o objetivo de vendas.

Store_Name	Actual	Goal
Store A	50	50
Store B	40	50
Store C	25	30

Pretendemos mostrar NULL se as vendas atuais forem iguais ao objetivo de vendas e mostrar as vendas atuais se os dois valores forem diferentes. Para tal, utilizamos a seguinte instrução SQL:

```
SELECT
     Store_Name,
     NULLIF (Actual, Goal)
FROM Sales_Data;
```

RESULTADO:

Store_Name	NULLIF (Actual, Goal)
Store A	NULL
Store B	40
Store C	25

WITH

Definição

Use a cláusula **WITH** para melhorar a velocidade de consulta para subconsultas complexas, sem a necessidade de conversão. Ela permite que você dê um nome a um bloco de subconsulta (um processo também chamado de refatoração de subconsulta), que pode ser referenciado em vários lugares dentro da consulta SQL principal.

- A cláusula é usada para definir uma relação temporária de modo que a saída dessa relação temporária esteja disponível e seja usada pela consulta associada à cláusula WITH.
- As consultas que possuem uma cláusula WITH associada também podem ser escritas usando subconsultas aninhadas, mas isso adiciona mais complexidade para ler/depurar a consulta SQL.
- A cláusula WITH não é suportada por todos os sistemas de banco de dados.
- O nome atribuído à subconsulta é tratado como se fosse uma visualização ou tabela embutida.

Sintaxe

```
WITH
     temporaryTable (averageValue) as
     (
          SELECT
               avg(Attr1) as Attr1
          FROM Table
     )
    SELECT
          Attr1
    FROM Table, temporaryTable
    WHERE Table.Attr1 > temporaryTable.averageValue;
```

Importante

Quando uma consulta com uma cláusula WITH é executada, primeiro a consulta mencionada na cláusula é avaliada e a saída dessa avaliação é armazenada em uma relação temporária. Em seguida, finalmente é executada a consulta principal associada à cláusula WITH que utilizaria a relação temporária produzida.

Exemplo 1

Encontre todos os funcionários cujo salário é maior que o salário médio de todos os funcionários.
Nome da tabela: **Employee**

EmployeeID	Name	Salary
100011	Smith	50000
100022	Bill	94000
100027	Sam	70550
100845	Walden	80000
115585	Erik	60000
1100070	Kate	69000

QUERY:

```
WITH
temporaryTable(averageValue) as
(
      SELECT
            avg(Salary)
    from Employee
);

SELECT
      EmployeeID,
      Name,
      Salary
FROM Employee, temporaryTable
WHERE Employee.Salary > temporaryTable.averageValue;
```

RESULTADO

EmployeeID	Name	Salary
100022	Bill	94000
100845	Walden	80000

awari