Aula 12 – CASE - Controle de Fluxo, View - Exibição

Prof. Sérgio Luiz Rodrigues

CASE

- O **CASE** é uma estrutura condicional versátil que permite executar lógica "if-then-else" em SQL.
- Existem duas formas de CASE: simple CASE e searched CASE.
- É útil para categorização, transformação de dados e aplicação de lógica de negócios complexa em consultas.

CASE WHEN

- O **CASE** é uma outra forma de tratar condições dentro do MySQL. Abaixo, temos a sua sintaxe:
- **CASE**
  - **WHEN** condicao1 THEN resultado1
  - **WHEN** condicao2 THEN resultado2
  - **WHEN** condicao3 THEN resultado3
  - **ELSE** resultado4
- **END**;
- Podemos fazer uma "tradução" da estrutura acima da seguinte forma:
- CASO a condição 1 seja verdadeira, ENTÃO retorna o resultado1. CASO a condição 2 seja verdadeira, ENTÃO retorna o resultado2. CASO a condição 3 seja verdadeira, ENTÃO retorna o resultado3. CASO CONTRÁRIO, retorna o resultado4

3

Exemplo com CASE

```
use emporio_turquinho;

SELECT id_pedido, quantidade,
CASE
    WHEN quantidade > 30 THEN 'a quantidade é maior que 30'
    WHEN quantidade = 30 THEN 'a quantidade é 30'
    ELSE 'a quantidade é menor que 30'
END AS verificando_quantidade
FROM detalhes_pedido;
```

4

VIEWS: Introdução

- Até aqui vimos como criar diferentes consultas aos bancos de dados. Utilizamos comandos como o SELECT, GROUP BY, JOINs, etc para criar tabelas como a mostrada ao lado.
- Mas pra onde foram todas essas "tabelas" que a gente criou? Elas estão em algum lugar?
- **A resposta é: elas não estão em nenhum lugar**
- **Nenhuma das consultas que fizemos ficou salvo em algum lugar.**
- Inclusive, diversas vezes precisamos criar as mesmas consultas, pois elas se perdem a cada novo SELECT, ou quando fechamos uma consulta e abrimos uma nova.
- Existe uma solução pra gente conseguir salvar essas tabelas em algum lugar, e essa solução é a **View**.

5

O que é uma View?

- Uma **View** (ou traduzindo, uma **exibição**), é uma tabela virtual criada a partir de uma consulta a uma ou mais tabelas (ou até mesmo de outras views) no banco de dados.
- Ela contém linhas e colunas assim como uma tabela real. Nela podemos utilizar comandos como o JOIN, WHERE e outras funções.
- Sempre mostra resultados atualizados dos dados, ou seja, uma vez criada uma View, caso haja alterações no Banco de Dados, as Views são atualizadas automaticamente.
- Caso o servidor seja desligado (ou o BD fechado), a **view continua armazenada no sistema.**
- Através de uma View, conseguimos **armazenar o resultado de um SELECT e acessar sempre que precisar, como se fosse uma tabela.**

6

Por que criar uma View?

- São muitas as vantagens de uma View.
- Abaixo temos algumas das principais:

Reutilização

Sempre que necessário, podemos consultar aquela View, pois ela fica armazenada no sistema.

Segurança

Ao criar uma View, estamos ocultando linhas ou colunas da tabela original do banco de dados. Desta forma, apenas algumas informações relevantes serão visualizadas na View.

Ganho de tempo

Quando criamos Views, estamos poupando o tempo de recriar vários SELECTs, o que aumenta a produtividade.

7

CREATE VIEW: Criando a primeira view

- Para criar uma View, utilizamos o comando **CREATE VIEW**. Na imagem abaixo temos a estrutura padrão:
- **CREATE VIEW** nome_da_view **AS SELECT**
- Coluna1,
- Coluna2,
- Coluna3
- **FROM**
- Tabela

- Uma vez criada, a View ficará armazenada no banco de dados, na pasta de Views.
- E para selecionar essa View, utilizamos o comando SELECT

```
1  CREATE VIEW vwClientes AS
2  SELECT
3      Nome AS 'Nome do Cliente',
4      Data_Nascimento AS 'Data de Nascimento',
5      Email AS 'E-mail'
6  FROM clientes;
```

8

ALTER VIEW: Alterando a view criada

- Para alterar uma View criada, usamos o comando **ALTER VIEW**.
- Na imagem ao lado, temos um exemplo. A vwClientes, criada anteriormente, foi alterada para considerar apenas os clientes do sexo feminino.

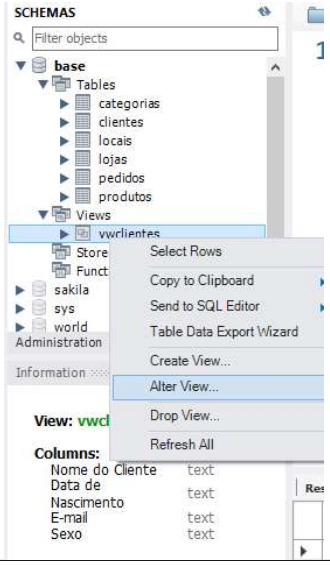
```
1  ALTER VIEW vwClientes AS
2  SELECT
3      Nome AS 'Nome do Cliente',
4      Data_Nascimento AS 'Data de Nascimento',
5      Email AS 'E-mail',
6      Sexo AS 'Sexo'
7  FROM clientes
8  WHERE Sexo = 'F';
9
10 SELECT * FROM vwclientes;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Nome do Cliente	Data de Nascimento	E-mail	Sexo
Christy	1988-02-10	christy12@hotmail.com	F
Elizabeth	1988-08-03	elizabeth5@gmail.com	F
Janet	1985-12-01	janet9@gmail.com	F
Rob	1984-07-02	rob4@hotmail.com	F
Jacquelyn	1984-02-01	jacquelyn20@hotmail.com	F
Lauren	1988-01-13	lauren41@hotmail.com	F
Sydney	1988-05-04	sydney23@gmail.com	F
Chloe	1999-02-22	chloe23@hotmail.com	F
Shannon	1964-06-21	shannon1@hotmail.com	F
Destiny	1998-08-29	destiny7@hotmail.com	F
Jill	1966-04-06	jill13@gmail.com	F
Bethany	1967-02-17	bethany10@hotmail.com	F

ALTER VIEW: Alterando a view criada

- Para alterar a view também podemos clicar nela com o botão direito e ir na opção Alter View.
- A janela que se abre mostra inclusive o código que deu origem à view criada.



Criando uma View com o comando WHERE (Exemplo 1)

- No exemplo a seguir vamos criar uma View que combina o comando SELECT com o WHERE

```
1  # Exemplos Views
2
3  # 1. VIEWS + WHERE: Criando Views com consultas filtradas.
4
5  -- Exemplo 1. Crie uma View chamada vwReceitaAcima4000 que armazene todas as colunas da tabela
   Pedidos. A sua View deverá conter apenas as vendas com receita acima de R$4.000.
```


Criando uma View com o comando WHERE (Exemplo 1)

```
5  -- Exemplo 1. Crie uma View chamada vwReceitaAcima4000 que armazene todas as colunas da tabela
   Pedidos. A sua View deverá conter apenas as vendas com receita acima de R$4.000.
6
7  SELECT * FROM pedidos;
8
9  CREATE VIEW vwReceitaAcima4000 AS
10 SELECT
11     *
12 FROM pedidos
13 WHERE Receita_Venda >= 4000;
```

Criando uma View com o comando GROUP BY

- Aplicando o que já sabemos, o gabarito é mostrado abaixo. Observe que não há nenhuma grande diferença na criação das views com comandos como o WHERE, GROUP BY, etc. Sempre começaremos com o CREATE VIEW, e em seguida o código da consulta que queremos realizar

```
CREATE VIEW vwReceitaECustoTotal AS
SELECT
    ID_Produto,
    SUM(Receita_Venda) AS 'Total Receita',
    SUM(Custo_Venda) AS 'Total Custo'
FROM pedidos
GROUP BY ID_Produto;
```

13

Alterando a View criada com o WHERE

- Criada uma View, podemos alterá-la usando o comando ALTER VIEW.
- Abaixo, temos um exemplo de como incluir o comando WHERE para trabalharmos em conjunto com o GROUP BY.

```
-- Exemplo 2. Altere a view anterior para mostrar o agrupamento apenas para os produtos da loja 2.

ALTER VIEW vwReceitaECustoTotal AS
SELECT
    ID_Produto,
    SUM(Receita_Venda) AS 'Total Receita',
    SUM(Custo_Venda) AS 'Total Custo'
FROM pedidos
WHERE ID_Loja = 2
GROUP BY ID_Produto;
```

14

Alterando a View criada com o HAVING

- Agora, podemos alterar nossa view usando o comando ALTER VIEW para realizar um filtro com o HAVING, conforme mostrado no exemplo abaixo.

```
-- Exemplo 3. Altere a view anterior para mostrar o agrupamento apenas para os produtos que tiveram uma
receita total maior que 1 milhão, na loja 2.

ALTER VIEW vwReceitaECustoTotal AS
SELECT
    ID_Produto,
    SUM(Receita_Venda) AS 'Total Receita',
    SUM(Custo_Venda) AS 'Total Custo'
FROM pedidos
WHERE ID_Loja = 2
GROUP BY ID_Produto
HAVING SUM(Receita_Venda) >= 1000000;
```


Criando uma View com o comando JOIN

```
# 1. VIEWS + JOIN e GROUP BY: Aplicando Join nas views criadas.

-- Exemplo 1. Crie uma view que seja a junção entre as tabelas de pedidos e de produtos. Ou seja, essa view
deve conter todas as colunas da tabela pedidos e as colunas Nome_Produto, Marca_Produto e Num_Serie da tabela
de produtos.

CREATE VIEW vwPedidosCompleta AS
SELECT
    pe.*,
    pr.Nome_Produto,
    pr.Marca_Produto,
    pr.Num_Serie
FROM pedidos AS pe
INNER JOIN produtos AS pr
    ON pe.ID_Produto = pr.ID_Produto;
```